

STARPORTS

1.0

Generated by Doxygen 1.8.17

1 Starports APP	1
2 File Index	2
2.1 File List	2
3 File Documentation	4
3.1 ADXL355.c File Reference	4
3.1.1 Detailed Description	5
3.1.2 Function Documentation	5
3.2 ADXL355.c	17
3.3 ADXL355.h File Reference	20
3.3.1 Detailed Description	22
3.3.2 Data Structure Documentation	23
3.3.3 Macro Definition Documentation	23
3.3.4 Enumeration Type Documentation	23
3.3.5 Function Documentation	29
3.3.6 Variable Documentation	39
3.4 ADXL355.h	39
3.5 BME280.c File Reference	41
3.5.1 Detailed Description	42
3.5.2 Function Documentation	43
3.6 BME280.c	53
3.7 BME280.h File Reference	56
3.7.1 Detailed Description	58
3.7.2 Data Structure Documentation	58
3.7.3 Macro Definition Documentation	59
3.7.4 Enumeration Type Documentation	60
3.7.5 Function Documentation	64
3.8 BME280.h	74
3.9 Board.h File Reference	76
3.9.1 Macro Definition Documentation	77
3.10 Board.h	83
3.11 CC3220SF_LAUNCHXL.h File Reference	84
3.11.1 Macro Definition Documentation	85
3.11.2 Enumeration Type Documentation	85
3.11.3 Function Documentation	90
3.12 CC3220SF_LAUNCHXL.h	90
3.13 CC3220SF_STARPORTS.c File Reference	92
3.13.1 Macro Definition Documentation	93
3.13.2 Function Documentation	94
3.13.3 Variable Documentation	94
3.14 CC3220SF_STARPORTS.c	103
3.15 CC3220SF_STARPORTS.h File Reference	111

3.15.1 Macro Definition Documentation	112
3.15.2 Enumeration Type Documentation	113
3.15.3 Function Documentation	117
3.16 CC3220SF_STARPORTS.h	118
3.17 DS1374.c File Reference	119
3.17.1 Detailed Description	120
3.17.2 Function Documentation	120
3.18 DS1374.c	124
3.19 DS1374.h File Reference	126
3.19.1 Detailed Description	127
3.19.2 Enumeration Type Documentation	127
3.19.3 Function Documentation	128
3.20 DS1374.h	132
3.21 file_system.c File Reference	132
3.21.1 Detailed Description	134
3.21.2 Function Documentation	134
3.21.3 Variable Documentation	150
3.22 file_system.c	150
3.23 file_system.h File Reference	158
3.23.1 Detailed Description	160
3.23.2 Macro Definition Documentation	160
3.23.3 Function Documentation	162
3.24 file_system.h	177
3.25 hal_ADC.c File Reference	178
3.25.1 Detailed Description	178
3.25.2 Function Documentation	178
3.26 hal_ADC.c	179
3.27 hal_ADC.h File Reference	179
3.27.1 Detailed Description	180
3.27.2 Function Documentation	180
3.28 hal_ADC.h	181
3.29 hal_GPIO.c File Reference	181
3.29.1 Detailed Description	182
3.29.2 Function Documentation	182
3.30 hal_GPIO.c	191
3.31 hal_GPIO.h File Reference	193
3.31.1 Detailed Description	194
3.31.2 Function Documentation	194
3.32 hal_GPIO.h	203
3.33 hal_I2C.c File Reference	203
3.33.1 Detailed Description	203
3.33.2 Function Documentation	204

3.34 hal_I2C.c	206
3.35 hal_I2C.h File Reference	207
3.35.1 Detailed Description	207
3.35.2 Function Documentation	208
3.36 hal_I2C.h	210
3.37 hal_LORA.c File Reference	210
3.37.1 Detailed Description	212
3.37.2 Function Documentation	212
3.37.3 Variable Documentation	234
3.38 hal_LORA.c	235
3.39 hal_LORA.h File Reference	245
3.39.1 Detailed Description	247
3.39.2 Data Structure Documentation	247
3.39.3 Macro Definition Documentation	248
3.39.4 Function Documentation	254
3.40 hal_LORA.h	276
3.41 hal_PWM.c File Reference	278
3.41.1 Detailed Description	278
3.41.2 Function Documentation	278
3.42 hal_PWM.c	279
3.43 hal_PWM.h File Reference	279
3.43.1 Detailed Description	280
3.43.2 Function Documentation	280
3.44 hal_PWM.h	281
3.45 hal_SPI.c File Reference	281
3.45.1 Detailed Description	282
3.45.2 Function Documentation	282
3.46 hal_SPI.c	286
3.47 hal_SPI.h File Reference	287
3.47.1 Detailed Description	288
3.47.2 Function Documentation	288
3.48 hal_SPI.h	292
3.49 hal_Timer.c File Reference	292
3.49.1 Detailed Description	293
3.49.2 Function Documentation	293
3.49.3 Variable Documentation	296
3.50 hal_Timer.c	297
3.51 hal_Timer.h File Reference	297
3.51.1 Detailed Description	298
3.51.2 Function Documentation	298
3.52 hal_Timer.h	301
3.53 hal_TMP006.c File Reference	301

3.53.1 Detailed Description	301
3.53.2 Function Documentation	302
3.54 hal_TMP006.c	304
3.55 hal_TMP006.h File Reference	305
3.55.1 Detailed Description	306
3.55.2 Macro Definition Documentation	306
3.55.3 Function Documentation	306
3.56 hal_TMP006.h	309
3.57 hal_UART.c File Reference	309
3.57.1 Detailed Description	310
3.57.2 Function Documentation	310
3.57.3 Variable Documentation	314
3.58 hal_UART.c	314
3.59 hal_UART.h File Reference	316
3.59.1 Detailed Description	317
3.59.2 Macro Definition Documentation	317
3.59.3 Function Documentation	317
3.60 hal_UART.h	321
3.61 hal_WD.c File Reference	322
3.61.1 Detailed Description	322
3.61.2 Function Documentation	322
3.61.3 Variable Documentation	324
3.62 hal_WD.c	324
3.63 hal_WD.h File Reference	325
3.63.1 Detailed Description	325
3.63.2 Function Documentation	325
3.64 hal_WD.h	326
3.65 LDC1000.c File Reference	326
3.65.1 Detailed Description	327
3.65.2 Function Documentation	328
3.66 LDC1000.c	336
3.67 LDC1000.h File Reference	338
3.67.1 Detailed Description	340
3.67.2 Enumeration Type Documentation	340
3.67.3 Function Documentation	346
3.68 LDC1000.h	354
3.69 main_nortos.c File Reference	356
3.69.1 Detailed Description	357
3.69.2 Function Documentation	357
3.70 main_nortos.c	361
3.71 Sensors.c File Reference	361
3.71.1 Detailed Description	362

3.71.2 Function Documentation	362
3.71.3 Variable Documentation	367
3.72 Sensors.c	369
3.73 Sensors.h File Reference	372
3.73.1 Detailed Description	372
3.73.2 Function Documentation	372
3.74 Sensors.h	377
3.75 STARPORTS_App.c File Reference	377
3.75.1 Detailed Description	378
3.75.2 Macro Definition Documentation	378
3.75.3 Function Documentation	379
3.75.4 Variable Documentation	382
3.76 STARPORTS_App.c	384
3.77 STARPORTS_App.h File Reference	388
3.77.1 Detailed Description	389
3.77.2 Data Structure Documentation	389
3.77.3 Macro Definition Documentation	391
3.77.4 Enumeration Type Documentation	394
3.77.5 Variable Documentation	395
3.78 STARPORTS_App.h	395
3.79 wifi.c File Reference	396
3.79.1 Detailed Description	397
3.79.2 Function Documentation	398
3.80 wifi.c	405
3.81 wifi.h File Reference	408
3.81.1 Detailed Description	410
3.81.2 Data Structure Documentation	410
3.81.3 Macro Definition Documentation	411
3.81.4 Enumeration Type Documentation	413
3.81.5 Function Documentation	414
3.81.6 Variable Documentation	421
3.82 wifi.h	421
Index	423

1 Starports APP

1. Starports Sensor Node Application

This Software starts up the STARPORTS Sensor Node, reads Node configuration parameters from file system, collects data from several sensors and transmit it by wireless communications. Finally, powers off the node.

The node is provided with sensors:

1. ADC voltage sensor
2. ADXL355 accelerometer

3. BME280 pressure, temperature and humidity sensor
4. LDC1000 inductive sensor

And wireless communication:

1. LoRa (RN2483)
2. Wifi (CC3220SF)

2. Working flow

- 2.1 Configures GPIO's and initialization of serial ports communications
- 2.2 Reads Node parameters from file system files
- 2.3 Configures Peripherals (RN2483, i2c)
- 2.4 Sets Node WakeUp_Time in RTC
- 2.5 Configures wireless connection
 - 2.5.1 Lora Mode (MODE=0)
 - 2.5.1.1 Configures Lora connection parameters
 - 2.5.1.2 Joins Lora ABP
 - 2.5.1 Wifi Mode (MODE=2)
 - 2.5.1.1 Configures Wifi connection parameters
 - 2.5.1.2 Connects to wifi AP
- 2.6 Reads data from sensors
- 2.6 Packages data to send
 - 2.6.1 NBoot = 0 --> BME280
 - 2.6.2 NBoot = 1 --> ADXL255
- 2.7 Sends data through selected wireless communication
- 2.8 Powers off the Node during wake up interval (Configured in "wakeupinterval" file (in seconds)

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

ADXL355.c Functions to configure and read data from ADXL355 accelerometer	4
ADXL355.h Functions to configure and read data from ADXL355 accelerometer	20
BME280.c Functions to configure and read data from BME280 Pressure, Temperature and Humidity sensor	41
BME280.h Functions to configure and read data from BME280 Pressure, Temperature and Humidity sensor	56
Board.h	76
CC3220SF_LAUNCHXL.h	84
CC3220SF_STARPORTS.c	92
CC3220SF_STARPORTS.h	111
DS1374.c Functions to interact with Watchdog RTC	119
DS1374.h Functions to interact with Watchdog RTC	126
file_system.c Functions to read and write data from file system files	132

file_system.h	Functions to read and write data from file system files	158
hal_ADC.c	Functions for ADC	178
hal_ADC.h	Functions for ADC	179
hal_GPIO.c	Functions for GPIO utilities	181
hal_GPIO.h	Functions for GPIO utilities	193
hal_I2C.c	Functions for I2C protocol	203
hal_I2C.h	Functions for I2C protocol	207
hal_LORA.c	Functions for LORA utilities	210
hal_LORA.h	Functions for LORA utilities	245
hal_PWM.c	Functions to manage PWM (Pulse Width Modulated signals) on CC3220SF	278
hal_PWM.h	Functions to manage PWM (Pulse Width Modulated signals) on CC3220SF	279
hal_SPI.c	Functions to manage SPI on CC3220SF	281
hal_SPI.h	Functions to manage SPI on CC3220SF	287
hal_Timer.c	Functions for Timer	292
hal_Timer.h	Functions for Timer	297
hal_TMP006.c	Functions to read data and configure TMP006 Temperature sensor	301
hal_TMP006.h	Functions to read data and configure TMP006 Temperature sensor	305
hal_UART.c	Functions to manage UART on CC3220SF	309
hal_UART.h	Functions to manage UART on CC3220SF	316
hal_WD.c	Functions to manage Watchdog	322
hal_WD.h	Functions to manage Watchdog	325

LDC1000.c	Functions to read and write data to LDC1000 inductance sensor	326
LDC1000.h	Functions to read and write data to LDC1000 inductance sensor	338
main_nortos.c	Main no Rtos	356
Sensors.c	Functions to get data from sensors and packet it to send later	361
Sensors.h	Functions to get data from sensors and packet it to send later	372
STARPORTS_App.c	Starports Main program flow	377
STARPORTS_App.h	Starports Main program flow	388
wifi.c	Functions for CC3220SF wifi management	396
wifi.h	Functions for CC3220SF wifi management	408

3 File Documentation

3.1 ADXL355.c File Reference

Functions to configure and read data from ADXL355 accelerometer.

```
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
#include <ti/drivers/SPI.h>
#include "STARPORTS_App.h"
#include "ADXL355.h"
#include "hal_SPI.h"
```

Functions

- bool **ADXL355_Data_Rdy** (SPI_Handle spi)
ADXL355 Data ready Function.
- uint16_t **ADXL355_DevId** (SPI_Handle spi)
ADXL355 Get DevID Function.
- void **ADXL355_Fifo_Data** (SPI_Handle spi, int32_t *Ax, int32_t *Ay, int32_t *Az, uint8_t Samples)
ADXL355 FIFO data Function.
- uint8_t **ADXL355_Fifo_Entries** (SPI_Handle spi)
ADXL355 FIFO entries Function.
- bool **ADXL355_Fifo_Full** (SPI_Handle spi)
ADXL355 FIFO Full Function.

- void [ADXL355_Fifo_Samples](#) (SPI_Handle *spi*, uint8_t *Samples*)
ADXL355 FIFO samples Function.
- void [ADXL355_Filter_Conf](#) (SPI_Handle *spi*, uint8_t *Val*)
ADXL355 Filter configuration Function.
- void [ADXL355_Get_Accel_Frame](#) (SPI_Handle *spi*, uint16_t *Samples*, int32_t **DataSensor*)
ADXL355 Get acceleration data Function.
- uint16_t [ADXL355_PartId](#) (SPI_Handle *spi*)
ADXL355 Get PartID Function.
- void [ADXL355_Power_Conf](#) (SPI_Handle *spi*, uint8_t *Val*)
ADXL355 Power Configuration Function.
- void [ADXL355_Range_Conf](#) (SPI_Handle *spi*, uint8_t *Val*)
ADXL355 Range Configuration Function.
- void [ADXL355_Reset](#) (SPI_Handle *spi*)
ADXL355 Reset Function.
- float [ADXL355_Temp](#) (SPI_Handle *spi*)
ADXL355 Read Temperature register Function.
- int32_t [ADXL355_XData](#) (SPI_Handle *spi*)
ADXL355 Read data X Function.
- int32_t [ADXL355_YData](#) (SPI_Handle *spi*)
ADXL355 Read data Y Function.
- int32_t [ADXL355_ZData](#) (SPI_Handle *spi*)
ADXL355 Read data Z Function.
- int32_t [u20_to_s32](#) (uint32_t *uData*)
ADXL355 u20_to_s32 Function.

3.1.1 Detailed Description

Functions to configure and read data from ADXL355 accelerometer.

Version

1.0

Date

07/05/2019

Author

A.Irizar

Definition in file [ADXL355.c](#).

3.1.2 Function Documentation

3.1.2.1 [ADXL355_Data_Rdy\(\)](#) bool ADXL355_Data_Rdy (

SPI_Handle *spi*)

ADXL355 Data ready Function.

This function

1. Reads data ready register and returns a boolean

Parameters

in	SPI_Handle	spi
----	------------	-----

Returns

bool Data_Rdy

Definition at line 310 of file ADXL355.c.

```
00310                                     {  
00311  
00312     uint8_t RxData[2];  
00313     bool Data_Rdy;  
00314  
00315     SPI_read_8bits(spi, ADXL355_STATUS, RxData, 1, RNW_LSB);  
00316  
00317     Data_Rdy = (RxData[1] & DATA_RDY);  
00318  
00319     return Data_Rdy;  
00320 }
```

3.1.2.2 ADXL355_DevId() uint16_t ADXL355_DevId (
 SPI_Handle spi)

ADXL355 Get DevID Function.

This function

1. Gets DevID from ADXL355

Parameters

in	SPI_Handle	spi
----	------------	-----

Returns

uint16_t DevId

Definition at line 36 of file ADXL355.c.

```
00036                                     {  
00037  
00038     uint8_t RxBuffer[3];  
00039     uint16_t DevId;  
00040  
00041     SPI_read_8bits(spi, DEVID_AD, RxBuffer, 2, RNW_LSB);  
00042  
00043     DevId = (RxBuffer[1]<<8) + RxBuffer[2];  
00044  
00045     return DevId;  
00046 }
```

3.1.2.3 ADXL355_Fifo_Data() void ADXL355_Fifo_Data (
 SPI_Handle spi,

```

    int32_t * Ax,
    int32_t * Ay,
    int32_t * Az,
    uint8_t Samples )

```

ADXL355 FIFO data Function.

This function

1. Reads N samples of FIFO data and saves it on Ax,Ay,Az

Parameters

in	<i>SPI_Handle</i>	spi
in	<i>int32_t</i>	*Ax pointer to Ax
in	<i>int32_t</i>	*Ay pointer to Ay
in	<i>int32_t</i>	*Az pointer to Az
in	<i>uint8_t</i>	Samples N samples to read

Returns

None

Definition at line 394 of file [ADXL355.c](#).

```

00394
00395
00396     uint8_t RxData[10];
00397     int8_t i;
00398     uint32_t uAx, uAy, uAz;
00399
00400     while (!ADXL355_Fifo_Full(spi)) {}                                // Polling the FIFO_FULL
Register of STATUS reg
00401
00402     i=0;
00403     while (i<Samples) {
00404         SPI_read_8bits(spi, FIFO_DATA, RxData, 9, RNW_LSB);
00405         uAx = (RxData[1] « 12) + (RxData[2] « 4) + (RxData[3] » 4);
00406         uAy = (RxData[4] « 12) + (RxData[5] « 4) + (RxData[6] » 4);
00407         uAz = (RxData[7] « 12) + (RxData[8] « 4) + (RxData[9] » 4);
00408         Ax[i] = u20_to_s32(uAx);
00409         Ay[i] = u20_to_s32(uAy);
00410         Az[i] = u20_to_s32(uAz);
00411         i++;
00412     }
00413 }

```

3.1.2.4 ADXL355_Fifo_Entries()

```
uint8_t ADXL355_Fifo_Entries (
    SPI_Handle spi )
```

ADXL355 FIFO entries Function.

This function

1. Read FIFO entries register and returns a buffer

Parameters

in	SPI_Handle	spi
----	------------	-----

Returns

uint8_t RxBuffer

Definition at line 155 of file [ADXL355.c](#).

```
00155                                     {  
00156  
00157     uint8_t RxBuffer[2];  
00158  
00159     SPI_read_8bits(spi, FIFO_ENTRIES, RxBuffer, 1, RNW_LSB);  
00160  
00161     return (RxBuffer[1] & 0x7F);  
00162 }
```

3.1.2.5 ADXL355_Fifo_Full() `bool ADXL355_Fifo_Full (`
 `SPI_Handle spi)`

ADXL355 FIFO Full Function.

This function

1. Reads FIFO buffer register and returns a boolean

Parameters

in	SPI_Handle	spi
----	------------	-----

Returns

bool Fifo_Full

Definition at line 286 of file [ADXL355.c](#).

```
00286                                     {  
00287  
00288     uint8_t RxData[2];  
00289     bool Fifo_Full;  
00290  
00291     SPI_read_8bits(spi, ADXL355_STATUS, RxData, 1, RNW_LSB);  
00292  
00293     Fifo_Full = (RxData[1] & 0x02) » 1;  
00294  
00295     return Fifo_Full;  
00296 }
```

3.1.2.6 ADXL355_Fifo_Samples() `void ADXL355_Fifo_Samples (`
 `SPI_Handle spi,`
 `uint8_t Samples)`

ADXL355 FIFO samples Function.

This function

1. Writes N samples to FIFO register

Parameters

in	<i>SPI_Handle</i>	spi
in	<i>uint8_t</i>	Samples

Returns

None

Definition at line 138 of file [ADXL355.c](#).

```
00138
00139
00140     SPI_write_8bits(spi, FIFO_SAMPLES, Samples, RNW_LSB);
00141 }
```

3.1.2.7 ADXL355_Filter_Conf() void ADXL355_Filter_Conf (

```
    SPI_Handle spi,
    uint8_t Val )
```

ADXL355 Filter configuration Function.

This function

1. Sets value on Filter's configuration register

Parameters

in	<i>SPI_Handle</i>	spi
in	<i>uint8_t</i>	Val

Returns

None

Definition at line 177 of file [ADXL355.c](#).

```
00177
00178
00179     SPI_write_8bits(spi, FILTER, Val, RNW_LSB);
00180 }
```

3.1.2.8 ADXL355_Get_Accel_Frame() void ADXL355_Get_Accel_Frame (

```
    SPI_Handle spi,
    uint16_t Samples,
    int32_t * DataSensor )
```

ADXL355 Get acceleration data Function.

This function

1. Reads N samples of acceleration data and saves it on DataSensor

Parameters

in	<i>SPI_Handle</i>	spi
in	<i>uint16_t</i>	Samples N samples to read
in	<i>int32_t</i>	*DataSensor pointer to datasensor

Returns

None

Definition at line 336 of file [ADXL355.c](#).

```

00336
00337
00338     int i=0;
00339     int32_t Ax[512], Ay[512], Az[512];
00340     int32_t xMean, yMean, zMean;
00341     double xRms, yRms, zRms;
00342
00343     xMean=0;
00344     yMean=0;
00345     zMean=0;
00346     while (i<Samples) {
00347         while (!ADXL355_Data_Rdy(spi)) {}
00348         Ax[i] = ADXL355_XData(spi);
00349         Ay[i] = ADXL355_YData(spi);
00350         Az[i] = ADXL355_ZData(spi);
00351         xMean += Ax[i];
00352         yMean += Ay[i];
00353         zMean += Az[i];
00354         i++;
00355     }
00356
00357     xMean = xMean / Samples;
00358     yMean = yMean / Samples;
00359     zMean = zMean / Samples;
00360
00361     xRms = 0.0;
00362     yRms = 0.0;
00363     zRms = 0.0;
00364     for (i=0;i<Samples;i++) {
00365         xRms += ( (Ax[i]-xMean)*(Ax[i]-xMean) );
00366         yRms += ( (Ay[i]-yMean)*(Ay[i]-yMean) );
00367         zRms += ( (Az[i]-zMean)*(Az[i]-zMean) );
00368     }
00369
00370     DataSensor[0] = xMean;
00371     DataSensor[1] = yMean;
00372     DataSensor[2] = zMean;
00373     DataSensor[3] = (int32_t)sqrt(xRms/Samples);
00374     DataSensor[4] = (int32_t)sqrt(yRms/Samples);
00375     DataSensor[5] = (int32_t)sqrt(zRms/Samples);
00376 }
```

3.1.2.9 ADXL355_PartId() *uint16_t ADXL355_PartId (**SPI_Handle spi)*

ADXL355 Get PartID Function.

This function

1. Gets PartID from ADXL355

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

uint16_t PartID

Definition at line 60 of file [ADXL355.c](#).

```
00060                                     {  
00061  
00062     uint8_t RxBuffer[3];  
00063     uint16_t PartId;  
00064  
00065     SPI_read_8bits(spi, PARTID, RxBuffer, 2, RNW_LSB);  
00066  
00067     PartId = (RxBuffer[1]«8) + RxBuffer[2];  
00068  
00069     return PartId;  
00070 }
```

3.1.2.10 ADXL355_Power_Conf() void ADXL355_Power_Conf (

```
SPI_Handle spi,  
uint8_t Val )
```

ADXL355 Power Configuration Function.

This function

1. Sets Power configuration value on Power configuration register

Parameters

in	<i>SPI_Handle</i>	spi
in	<i>uint8_t</i>	Val

Returns

None

Definition at line 102 of file [ADXL355.c](#).

```
00102                                     {  
00103  
00104     SPI_write_8bits(spi, POWER_CTL, Val, RNW_LSB);  
00105 }
```

3.1.2.11 ADXL355_Range_Conf() void ADXL355_Range_Conf (

```
SPI_Handle spi,  
uint8_t Val )
```

ADXL355 Range Configuration Function.

This function

1. Sets Range configuration value on Range configuration register

Parameters

in	<i>SPI_Handle</i>	spi
in	<i>uint8_t</i>	Val

Returns

None

Definition at line 120 of file [ADXL355.c](#).

```
00120
00121
00122     SPI_write_8bits(spi, RANGE, Val, RNW_LSB);
00123 }
```

3.1.2.12 ADXL355_Reset() void ADXL355_Reset (
 SPI_Handle *spi*)

ADXL355 Reset Function.

This function

1. Resets ADXL355

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

None

Definition at line 84 of file [ADXL355.c](#).

```
00084
00085
00086     SPI_write_8bits(spi, ADXL355_RESET_REG, ADXL355_RESET, RNW_LSB);
00087 }
```

3.1.2.13 ADXL355_Temp() float ADXL355_Temp (
 SPI_Handle *spi*)

ADXL355 Read Temperature register Function.

This function

1. Reads Temperature from Temperature registers

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

$(uData - t_bias) / t_sens + 25.0$; Temperature data

Definition at line 263 of file [ADXL355.c](#).

```
00263                                     {  
00264  
00265     uint8_t RxData[3];  
00266     uint16_t uData;  
00267  
00268     SPI_read_8bits(spi, TEMP2, RxData, 2, RNW_LSB);  
00269     uData = ((RxData[1] & 0x0F) « 8) + RxData[2];  
00270  
00271     return (uData - t_bias) / t_sens + 25.0;  
00272 }
```

3.1.2.14 ADXL355_XData() int32_t ADXL355_XData (
 SPI_Handle spi)

ADXL355 Read data X Function.

This function

1. Reads data from X axis register

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

uData Axis X data

Definition at line 194 of file [ADXL355.c](#).

```
00194                                     {  
00195  
00196     uint8_t RxData[4];  
00197     uint32_t uData;  
00198  
00199     SPI_read_8bits(spi, XDATA3, RxData, 3, RNW_LSB);  
00200     uData = (int32_t)(RxData[1] « 12) + (int32_t)(RxData[2] « 4) + (int32_t)(RxData[3] « 4);  
00201  
00202     return u20_to_s32(uData);  
00203 }
```

3.1.2.15 ADXL355_YData() int32_t ADXL355_YData (
 SPI_Handle spi)

ADXL355 Read data Y Function.

This function

1. Reads data from Y axis register

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

uData Axis Y data

Definition at line 217 of file [ADXL355.c](#).

```
00217
00218
00219     uint8_t RxData[4];
00220     uint32_t uData;
00221
00222     SPI_read_8bits(spi, YDATA3, RxData, 3, RNW_LSB);
00223     uData = (int32_t)(RxData[1] << 12) + (int32_t)(RxData[2] << 4) + (int32_t)(RxData[3] >> 4);
00224
00225     return u20_to_s32(uData);
00226 }
```

3.1.2.16 ADXL355_ZData() int32_t ADXL355_ZData (
 SPI_Handle *spi*)

ADXL355 Read data Z Function.

This function

1. Reads data from Z axis register

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

uData Axis Z data

Definition at line 240 of file [ADXL355.c](#).

```
00240
00241
00242     uint8_t RxData[4];
00243     uint32_t uData;
00244
00245     SPI_read_8bits(spi, ZDATA3, RxData, 3, RNW_LSB);
00246     uData = (int32_t)(RxData[1] << 12) + (int32_t)(RxData[2] << 4) + (int32_t)(RxData[3] >> 4);
00247
00248     return u20_to_s32(uData);
00249 }
```

3.1.2.17 u20_to_s32() int32_t u20_to_s32 (
 uint32_t *uData*)

ADXL355 u20_to_s32 Function.

This function

1. Converts data to int

Parameters

in	<i>uint32_t</i> <i>_t</i>	uData Accelerometer data
----	------------------------------	--------------------------

Returns

int32_t iData Accelerometer data integer

Definition at line 427 of file [ADXL355.c](#).

```

00427
00428     int32_t iData;
00429
00430     if (uData & 0x80000) {
00431         iData = uData | 0xFFFF0000;
00432     } else {
00433         iData = (int32_t)uData;
00434     }
00435
00436     return iData;
00437 }
00438 }
```

3.2 ADXL355.c

```

00001
00009 //*****
00010 //      INCLUDES
00011 //*****
00012 #include <stdlib.h>
00013 #include <stdbool.h>
00014 #include <math.h>
00015 #include <ti/drivers/SPI.h>
00016 #include "STARPORTS_App.h"
00017 #include "ADXL355.h"
00018 #include "hal_SPI.h"
00019
00020 //*****
00021 //      FUNCTIONS
00022 //*****
00023
00024 //*****
00025 //
00034 //
00035 //*****
00036 uint16_t ADXL355_DevId(SPI_Handle spi) {
00037
00038     uint8_t RxBuffer[3];
00039     uint16_t DevId;
00040
00041     SPI_read_8bits(spi, DEVID_AD, RxBuffer, 2, RNW_LSB);
00042
00043     DevId = (RxBuffer[1] << 8) + RxBuffer[2];
00044
00045     return DevId;
00046 }
00047
00048 //*****
00049 //
00058 //
00059 //*****
00060 uint16_t ADXL355_PartId(SPI_Handle spi) {
00061
00062     uint8_t RxBuffer[3];
00063     uint16_t PartId;
00064
00065     SPI_read_8bits(spi, PARTID, RxBuffer, 2, RNW_LSB);
00066
00067     PartId = (RxBuffer[1] << 8) + RxBuffer[2];
00068
00069     return PartId;
00070 }
00071
00072 //*****
00073 //
00082 //
00083 //*****
```

```

00084 void ADXL355_Reset(SPI_HandleTypeDef spi) {
00085
00086     SPI_write_8bits(spi, ADXL355_RESET_REG, ADXL355_RESET, RNW_LSB);
00087 }
00088
00089 //*****
00090 //
00100 //
00101 //*****
00102 void ADXL355_Power_Conf(SPI_HandleTypeDef spi, uint8_t Val) {
00103
00104     SPI_write_8bits(spi, POWER_CTL, Val, RNW_LSB);
00105 }
00106
00107 //*****
00108 //
00118 //
00119 //*****
00120 void ADXL355_Range_Conf(SPI_HandleTypeDef spi, uint8_t Val) {
00121
00122     SPI_write_8bits(spi, RANGE, Val, RNW_LSB);
00123 }
00124
00125 //*****
00126 //
00136 //
00137 //*****
00138 void ADXL355_Fifo_Samples(SPI_HandleTypeDef spi, uint8_t Samples) {
00139
00140     SPI_write_8bits(spi, FIFO_SAMPLES, Samples, RNW_LSB);
00141 }
00142
00143 //*****
00144 //
00153 //
00154 //*****
00155 uint8_t ADXL355_Fifo_Entries(SPI_HandleTypeDef spi) {
00156
00157     uint8_t RxBuffer[2];
00158
00159     SPI_read_8bits(spi, FIFO_ENTRIES, RxBuffer, 1, RNW_LSB);
00160
00161     return (RxBuffer[1] & 0x7F);
00162 }
00163
00164 //*****
00165 //
00175 //
00176 //*****
00177 void ADXL355_Filter_Conf(SPI_HandleTypeDef spi, uint8_t Val) {
00178
00179     SPI_write_8bits(spi, FILTER, Val, RNW_LSB);
00180 }
00181
00182 //*****
00183 //
00192 //
00193 //*****
00194 int32_t ADXL355_XData(SPI_HandleTypeDef spi) {
00195
00196     uint8_t RxData[4];
00197     uint32_t uData;
00198
00199     SPI_read_8bits(spi, XDATA3, RxData, 3, RNW_LSB);
00200     uData = (int32_t)(RxData[1] << 12) + (int32_t)(RxData[2] << 4) + (int32_t)(RxData[3] >> 4);
00201
00202     return u20_to_s32(uData);
00203 }
00204
00205 //*****
00206 //
00215 //
00216 //*****
00217 int32_t ADXL355_YData(SPI_HandleTypeDef spi) {
00218
00219     uint8_t RxData[4];
00220     uint32_t uData;
00221
00222     SPI_read_8bits(spi, YDATA3, RxData, 3, RNW_LSB);
00223     uData = (int32_t)(RxData[1] << 12) + (int32_t)(RxData[2] << 4) + (int32_t)(RxData[3] >> 4);
00224
00225     return u20_to_s32(uData);
00226 }
00227
00228 //*****
00229 //
00238 //

```

```

00239 //*****
00240 int32_t ADXL355_ZData(SPI_Handle spi) {
00241
00242     uint8_t RxData[4];
00243     uint32_t uData;
00244
00245     SPI_read_8bits(spi, ZDATA3, RxData, 3, RNW_LSB);
00246     uData = (int32_t)(RxData[1] << 12) + (int32_t)(RxData[2] << 4) + (int32_t)(RxData[3] >> 4);
00247
00248     return u20_to_s32(uData);
00249 }
00250
00251 //*****
00252 //
00261 //
00262 //*****
00263 float ADXL355_Temp(SPI_Handle spi) {
00264
00265     uint8_t RxData[3];
00266     uint16_t uData;
00267
00268     SPI_read_8bits(spi, TEMP2, RxData, 2, RNW_LSB);
00269     uData = ((RxData[1] & 0x0F) << 8) + RxData[2];
00270
00271     return (uData-t_bias)/t_sens + 25.0;
00272 }
00273
00274 //*****
00275 //
00284 //
00285 //*****
00286 bool ADXL355_Fifo_Full(SPI_Handle spi) {
00287
00288     uint8_t RxData[2];
00289     bool Fifo_Full;
00290
00291     SPI_read_8bits(spi, ADXL355_STATUS, RxData, 1, RNW_LSB);
00292
00293     Fifo_Full = (RxData[1] & 0x02) >> 1;
00294
00295     return Fifo_Full;
00296 }
00297
00298 //*****
00299 //
00308 //
00309 //*****
00310 bool ADXL355_Data_Rdy(SPI_Handle spi) {
00311
00312     uint8_t RxData[2];
00313     bool Data_Rdy;
00314
00315     SPI_read_8bits(spi, ADXL355_STATUS, RxData, 1, RNW_LSB);
00316
00317     Data_Rdy = (RxData[1] & DATA_RDY);
00318
00319     return Data_Rdy;
00320 }
00321
00322 //*****
00323 //
00334 //
00335 //*****
00336 void ADXL355_Get_Accel_Frame(SPI_Handle spi, uint16_t Samples, int32_t *DataSensor) {
00337
00338     int i=0;
00339     int32_t Ax[512], Ay[512], Az[512];
00340     int32_t xMean, yMean, zMean;
00341     double xRms, yRms, zRms;
00342
00343     xMean=0;
00344     yMean=0;
00345     zMean=0;
00346     while (i<Samples) {
00347         while(!ADXL355_Data_Rdy(spi)) {}
00348         Ax[i] = ADXL355_XData(spi);
00349         Ay[i] = ADXL355_YData(spi);
00350         Az[i] = ADXL355_ZData(spi);
00351         xMean += Ax[i];
00352         yMean += Ay[i];
00353         zMean += Az[i];
00354         i++;
00355     }
00356
00357     xMean = xMean / Samples;
00358     yMean = yMean / Samples;
00359     zMean = zMean / Samples;

```

```

00360
00361     xRms = 0.0;
00362     yRms = 0.0;
00363     zRms = 0.0;
00364     for (i=0;i<Samples;i++) {
00365         xRms += ( (Ax[i]-xMean)*(Ax[i]-xMean) );
00366         yRms += ( (Ay[i]-yMean)*(Ay[i]-yMean) );
00367         zRms += ( (Az[i]-zMean)*(Az[i]-zMean) );
00368     }
00369
00370     DataSensor[0] = xMean;
00371     DataSensor[1] = yMean;
00372     DataSensor[2] = zMean;
00373     DataSensor[3] = (int32_t)sqrt(xRms/Samples);
00374     DataSensor[4] = (int32_t)sqrt(yRms/Samples);
00375     DataSensor[5] = (int32_t)sqrt(zRms/Samples);
00376 }
00377
00378 //*****
00379 //
00392 //
00393 //*****
00394 void ADXL355_Fifo_Data(SPI_Handle spi, int32_t *Ax, int32_t *Ay, int32_t *Az, uint8_t Samples) {
00395
00396     uint8_t RxData[10];
00397     int8_t i;
00398     uint32_t uAx, uAy, uAz;
00399
00400     while (!ADXL355_Fifo_Full(spi)) {}                                // Polling the FIFO_FULL
00401     Register of STATUS reg
00402     i=0;
00403     while (i<Samples) {
00404         SPI_read_8bits(spi, FIFO_DATA, RxData, 9, RNW_LSB);
00405         uAx = (RxData[1] « 12) + (RxData[2] « 4) + (RxData[3] » 4);
00406         uAy = (RxData[4] « 12) + (RxData[5] « 4) + (RxData[6] » 4);
00407         uAz = (RxData[7] « 12) + (RxData[8] « 4) + (RxData[9] » 4);
00408         Ax[i] = u20_to_s32(uAx);
00409         Ay[i] = u20_to_s32(uAy);
00410         Az[i] = u20_to_s32(uAz);
00411         i++;
00412     }
00413 }
00414
00415 //*****
00416 //
00425 //
00426 //*****
00427 int32_t u20_to_s32(uint32_t uData) {
00428
00429     int32_t iData;
00430
00431     if (uData & 0x80000) {
00432         iData = uData | 0xFFFF0000;
00433     } else {
00434         iData = (int32_t)uData;
00435     }
00436
00437     return iData;
00438 }

```

3.3 ADXL355.h File Reference

Functions to configure and read data from ADXL355 accelerometer.

```
#include <ti/drivers/SPI.h>
#include <stdbool.h>
```

Data Structures

- struct **ADXL355_calibdata_t**
struct to save ADXL355 calibration data [More...](#)

Macros

- #define ADXL355__DEVICE_AD 0xAD
- #define ADXL355_RESET 0x52

Enumerations

- enum `ADXL355_act_ctl_t` { `ACT_Z` = 0x04, `ACT_Y` = 0x02, `ACT_X` = 0x01 }
ADXL 355 Activity enable registers.
- enum `ADXL355_filter_ctl_t` {
`HPFOFF` = 0x00, `HPF247` = 0x10, `HPF62` = 0x20, `HPF15` = 0x30,
`HPF3` = 0x40, `HPF9` = 0x50, `HPF02` = 0x60, `ODR4000HZ` = 0x00,
`ODR2000HZ` = 0x01, `ODR1000HZ` = 0x02, `ODR500HZ` = 0x03, `ODR250HZ` = 0x04,
`ODR125Hz` = 0x05, `ODR62HZ` = 0x06, `ODR31Hz` = 0x07, `ODR15Hz` = 0x08,
`ODR7Hz` = 0x09, `ODR3HZ` = 0x0A }
High-Pass and Low-Pass Filter registers. Use this register to specify parameters for the internal high-pass and low-pass filters.
- enum `ADXL355_i2c_speed_t` { `I2C_HIGH_SPEED` = 0x80, `I2C_FAST_MODE` = 0x00 }
ADXL355 Speed registers.
- enum `ADXL355_int_ctl_t` { `ST2` = 0x02, `ST1` = 0x01 }
ADXL355 self test interrupt registers.
- enum `ADXL355_int_pol_t` { `INT_POL_LOW` = 0x00, `INT_POL_HIGH` = 0x40 }
ADXL355 polarity registers.
- enum `ADXL355_intmap_ctl_t` { `OVR_EN` = 0x04, `FULL_EN` = 0x02, `RDY_EN` = 0x01 }
ADXL355 Interrupt PIN function map register. The INT_MAP register configures the interrupt pins. Bits[7:0] select which function(s) generate an interrupt on the INT1 and INT2 pins. Multiple events can be configured. If the corresponding bit is set to 1, the function generates an interrupt on the interrupt pins.
- enum `ADXL355_modes_t` { `DRDY_OFF` = 0x04, `TEMP_OFF` = 0x02, `STANDBY` = 0x01, `MEASUREMENT` = 0x00 }
ADXL355 Power control modes registers.
- enum `ADXL355_range_ctl_t` { `RANGE2G` = 0x01, `RANGE4G` = 0x02, `RANGE8G` = 0x03 }
ADXL355 Range registers.
- enum `ADXL355_register_t` {
`DEVID_AD` = 0x00, `DEVID_MST` = 0x01, `PARTID` = 0x02, `REVID` = 0x03,
`ADXL355_STATUS` = 0x04, `FIFO_ENTRIES` = 0x05, `TEMP2` = 0x06, `TEMP1` = 0x07,
`XDATA3` = 0x08, `XDATA2` = 0x09, `XDATA1` = 0x0A, `YDATA3` = 0x0B,
`YDATA2` = 0x0C, `YDATA1` = 0x0D, `ZDATA3` = 0x0E, `ZDATA2` = 0x0F,
`ZDATA1` = 0x10, `FIFO_DATA` = 0x11, `OFFSET_X_H` = 0x1E, `OFFSET_X_L` = 0x1F,
`OFFSET_Y_H` = 0x20, `OFFSET_Y_L` = 0x21, `OFFSET_Z_H` = 0x22, `OFFSET_Z_L` = 0x23,
`ACT_EN` = 0x24, `ACT_THRESH_H` = 0x25, `ACT_THRESH_L` = 0x26, `ACT_COUNT` = 0x27,
`FILTER` = 0x28, `FIFO_SAMPLES` = 0x29, `INT_MAP` = 0x2A, `SYNC` = 0x2B,
`RANGE` = 0x2C, `POWER_CTL` = 0x2D, `SELF_TEST` = 0x2E, `ADXL355_RESET_REG` = 0x2F }
ADXL355 Registers.
- enum `ADXL355_status_t` {
`NVM_BUSY` = 0x10, `ACTIVITY` = 0x08, `FIFO_OVR` = 0x04, `FIFO_FULL` = 0x02,
`DATA_RDY` = 0x01 }
ADXL355 Status registers. This register includes bits that describe the various conditions of the ADXL355.
- enum `ADXL355_sync_ctl_t` { `EXT_CLK` = 0x04, `INT_SYNC` = 0x00, `EXT_SYNC_NO_INT` = 0x01,
`EXT_SYNC_INT` = 0x02 }
ADXL355 External timing register. Use this register to control the external timing triggers.

Functions

- bool `ADXL355_Data_Rdy` (SPI_Handle `spi`)
ADXL355 Data ready Function.
- uint16_t `ADXL355_DevId` (SPI_Handle `spi`)
ADXL355 Get DevID Function.
- void `ADXL355_Fifo_Data` (SPI_Handle `spi`, int32_t *Ax, int32_t *Ay, int32_t *Az, uint8_t Samples)
ADXL355 FIFO data Function.
- uint8_t `ADXL355_Fifo_Entries` (SPI_Handle `spi`)
ADXL355 FIFO entries Function.
- bool `ADXL355_Fifo_Full` (SPI_Handle `spi`)
ADXL355 FIFO Full Function.
- void `ADXL355_Fifo_Samples` (SPI_Handle `spi`, uint8_t Samples)
ADXL355 FIFO samples Function.
- void `ADXL355_Filter_Conf` (SPI_Handle `spi`, uint8_t Val)
ADXL355 Filter configuration Function.
- void `ADXL355_Get_Accel_Frame` (SPI_Handle `spi`, uint16_t Samples, int32_t *DataSensor)
ADXL355 Get acceleration data Function.
- uint16_t `ADXL355_PartId` (SPI_Handle `spi`)
ADXL355 Get PartID Function.
- void `ADXL355_Power_Conf` (SPI_Handle `spi`, uint8_t Val)
ADXL355 Power Configuration Function.
- void `ADXL355_Range_Conf` (SPI_Handle `spi`, uint8_t Val)
ADXL355 Range Configuration Function.
- void `ADXL355_Reset` (SPI_Handle `spi`)
ADXL355 Reset Function.
- float `ADXL355_Temp` (SPI_Handle `spi`)
ADXL355 Read Temperature register Function.
- int32_t `ADXL355_XData` (SPI_Handle `spi`)
ADXL355 Read data X Function.
- int32_t `ADXL355_YData` (SPI_Handle `spi`)
ADXL355 Read data Y Function.
- int32_t `ADXL355_ZData` (SPI_Handle `spi`)
ADXL355 Read data Z Function.
- int32_t `u20_to_s32` (uint32_t uData)
ADXL355 u20_to_s32 Function.

Variables

- float `axis355_sens`
- `ADXL355_calibdata_t calib_data`

3.3.1 Detailed Description

Functions to configure and read data from ADXL355 accelerometer.

Version

1.0

Date

07/05/2019

Author

A.Irizar

Definition in file [ADXL355.h](#).

3.3.2 Data Structure Documentation

3.3.2.1 struct ADXL355_calibdata_t struct to save ADXL355 calibration data

Definition at line [38](#) of file [ADXL355.h](#).

Data Fields

float	B[2][0]	Offset float var.
float	S[2][2]	Sensitivity float var.
float	St	Sensitivity float var.

3.3.3 Macro Definition Documentation

3.3.3.1 ADXL355__DEVICE_AD #define ADXL355__DEVICE_AD 0xAD

Analog Devices ID

Definition at line [21](#) of file [ADXL355.h](#).

3.3.3.2 ADXL355_RESET #define ADXL355_RESET 0x52

ADXL355 Reset Register

Definition at line [22](#) of file [ADXL355.h](#).

3.3.4 Enumeration Type Documentation

3.3.4.1 ADXL355_act_ctl_t enum ADXL355_act_ctl_t

ADXL 355 Activity enable registers.

Enumerator

<code>ACT_Z</code>	Z-axis data is a component of the activity detection algorithm
<code>ACT_Y</code>	Y-axis data is a component of the activity detection algorithm
<code>ACT_X</code>	X-axis data is a component of the activity detection algorithm

Definition at line 111 of file [ADXL355.h](#).

```
00111      {
00112      ACT_Z = 0x04,
00113      ACT_Y = 0x02,
00114      ACT_X = 0x01
00115 } ADXL355_act_ctl_t;
```

3.3.4.2 `ADXL355_filter_ctl_t` enum [ADXL355_filter_ctl_t](#)

High-Pass and Low-Pass Filter registers. Use this register to specify parameters for the internal high-pass and low-pass filters.

Enumerator

<code>HPFOFF</code>	High-Pass and Low-Pass Filter
<code>HPF247</code>	High-Pass and Low-Pass Filter
<code>HPF62</code>	High-Pass and Low-Pass Filter
<code>HPF15</code>	High-Pass and Low-Pass Filter
<code>HPF3</code>	High-Pass and Low-Pass Filter
<code>HPF09</code>	High-Pass and Low-Pass Filter
<code>HPF02</code>	High-Pass and Low-Pass Filter
<code>ODR4000HZ</code>	High-Pass and Low-Pass Filter
<code>ODR2000HZ</code>	High-Pass and Low-Pass Filter
<code>ODR1000HZ</code>	High-Pass and Low-Pass Filter
<code>ODR500HZ</code>	High-Pass and Low-Pass Filter
<code>ODR250HZ</code>	High-Pass and Low-Pass Filter
<code>ODR125Hz</code>	High-Pass and Low-Pass Filter
<code>ODR62Hz</code>	High-Pass and Low-Pass Filter
<code>ODR31Hz</code>	High-Pass and Low-Pass Filter
<code>ODR15Hz</code>	High-Pass and Low-Pass Filter
<code>ODR7Hz</code>	High-Pass and Low-Pass Filter
<code>ODR3HZ</code>	High-Pass and Low-Pass Filter

Definition at line 121 of file [ADXL355.h](#).

```
00121      {
00122      HPFOFF = 0x00,
00123      HPF247 = 0x10,
00124      HPF62 = 0x20,
00125      HPF15 = 0x30,
00126      HPF3 = 0x40,
00127      HPF09 = 0x50,
00128      HPF02 = 0x60,
00129      ODR4000HZ = 0x00,
00130      ODR2000HZ = 0x01,
00131      ODR1000HZ = 0x02,
```

```

00132     ODR500HZ = 0x03,
00133     ODR250HZ = 0x04,
00134     ODR125HZ = 0x05,
00135     ODR62HZ = 0x06,
00136     ODR31Hz = 0x07,
00137     ODR15Hz = 0x08,
00138     ODR7Hz = 0x09,
00139     ODR3Hz = 0x0A
00140 } ADXL355_filter_ctl_t;

```

3.3.4.3 ADXL355_i2c_speed_t enum ADXL355_i2c_speed_t

ADXL355 Speed registers.

Enumerator

I2C_HIGH_SPEED	I2C High speed
I2C_FAST_MODE	I2C Fast speed

Definition at line 167 of file [ADXL355.h](#).

```

00167     {
00168     I2C_HIGH_SPEED = 0x80,
00169     I2C_FAST_MODE = 0x00
00170 } ADXL355_i2c_speed_t;

```

3.3.4.4 ADXL355_int_ctl_t enum ADXL355_int_ctl_t

ADXL355 self test interrupt registers.

Enumerator

ST2	Set to 1 to enable self test force
ST1	Set to 1 to enable self test mode

Definition at line 192 of file [ADXL355.h](#).

```

00192     {
00193     ST2 = 0x02,
00194     ST1 = 0x01
00195 } ADXL355_int_ctl_t;

```

3.3.4.5 ADXL355_int_pol_t enum ADXL355_int_pol_t

ADXL355 polarity registers.

Enumerator

INT_POL_LOW	Polarity Low
INT_POL_HIGH	Polarity High

Definition at line 175 of file [ADXL355.h](#).

```
00175      {
00176      INT_POL_LOW = 0x00,
00177      INT_POL_HIGH = 0x40
00178 } ADXL355_int_pol_t;
```

3.3.4.6 ADXL355_intmap_ctl_t enum [ADXL355_intmap_ctl_t](#)

ADXL355 Interrupt PIN function map register. The INT_MAP register configures the interrupt pins. Bits[7:0] select which function(s) generate an interrupt on the INT1 and INT2 pins. Multiple events can be configured. If the corresponding bit is set to 1, the function generates an interrupt on the interrupt pins.

Enumerator

OVR_EN	FIFO_OVR interrupt enable
FULL_EN	FIFO_FULL interrupt enable
RDY_EN	DATA_RDY interrupt enable

Definition at line 147 of file [ADXL355.h](#).

```
00147      {
00148      OVR_EN = 0x04,
00149      FULL_EN = 0x02,
00150      RDY_EN = 0x01
00151 } ADXL355_intmap_ctl_t;
```

3.3.4.7 ADXL355_modes_t enum [ADXL355_modes_t](#)

ADXL355 Power control modes registers.

Enumerator

DRDY_OFF	Set to 1 to force the DRDY output to 0 in modes where it is normally signal data ready
TEMP_OFF	Set to 1 to disable temperature processing. Temperature processing is also disabled when STANDBY=1
STANDBY	Standby mode
MEASUREMENT	Measurement mode

Definition at line 101 of file [ADXL355.h](#).

```
00101      {
00102      DRDY_OFF = 0x04,
00103      TEMP_OFF = 0x02,
00104      STANDBY = 0x01,
00105      MEASUREMENT = 0x00
00106 } ADXL355_modes_t;
```

3.3.4.8 ADXL355_range_ctl_t enum [ADXL355_range_ctl_t](#)

ADXL355 Range registers.

Enumerator

RANGE2G	Rango = 2g
RANGE4G	Rango = 4g
RANGE8G	Rango = 8g

Definition at line 183 of file [ADXL355.h](#).

```
00183 {
00184     RANGE2G = 0x01,
00185     RANGE4G = 0x02,
00186     RANGE8G = 0x03
00187 } ADXL355_range_ctl_t;
```

3.3.4.9 ADXL355_register_t enum [ADXL355_register_t](#)

ADXL355 Registers.

Enumerator

DEVID_AD	Analog Devices ID
DEVID_MST	Analog Devices MEMS ID
PARTID	Device ID (355 octal)
REVID	Mask revision
ADXL355_STATUS	Status
FIFO_ENTRIES	FIFO Entries
TEMP2	Temperature 2
TEMP1	Temperature 1
XDATA3	x-axis acceleration data
XDATA2	x-axis acceleration data
XDATA1	x-axis acceleration data
YDATA3	y-axis acceleration data
YDATA2	y-axis acceleration data
YDATA1	y-axis acceleration data
ZDATA3	z-axis acceleration data
ZDATA2	z-axis acceleration data
ZDATA1	z-axis acceleration data
FIFO_DATA	FIFO Data
OFFSET_X_H	Offset added to x-axis data after all other signal processing
OFFSET_X_L	Offset added to x-axis data after all other signal processing
OFFSET_Y_H	Offset added to y-axis data after all other signal processing
OFFSET_Y_L	Offset added to y-axis data after all other signal processing
OFFSET_Z_H	Offset added to z-axis data after all other signal processing
OFFSET_Z_L	Offset added to z-axis data after all other signal processing
ACT_EN	Activity enable
ACT_THRESH_H	Threshold for activity detection.
ACT_THRESH_L	Threshold for activity detection.
ACT_COUNT	Number of consecutive events above threshold required to detect activity
FILTER	Parameters for the internal high-pass and low-pass filters.
FIFO_SAMPLES	To specify the number of samples to store in the FIFO

Enumerator

INT_MAP	Configures the interrupt pins
SYNC	Control the external timing triggers
RANGE	I2c speed, polarity and range
POWER_CTL	Power control
SELF_TEST	Self test
ADXL355_RESET_REG	Resets the device

Definition at line 47 of file [ADXL355.h](#).

```

00047      {
00048      DEVID_AD = 0x00,
00049      DEVID_MST = 0x01,
00050      PARTID = 0x02,
00051      REVID = 0x03,
00052      ADXL355_STATUS = 0x04,
00053      FIFO_ENTRIES = 0x05,
00054      TEMP2 = 0x06,
00055      TEMP1 = 0x07,
00056      XDATA3 = 0x08,
00057      XDATA2 = 0x09,
00058      XDATA1 = 0x0A,
00059      YDATA3 = 0x0B,
00060      YDATA2 = 0x0C,
00061      YDATA1 = 0x0D,
00062      ZDATA3 = 0x0E,
00063      ZDATA2 = 0x0F,
00064      ZDATA1 = 0x10,
00065      FIFO_DATA = 0x11,
00066      OFFSET_X_H = 0x1E,
00067      OFFSET_X_L = 0x1F,
00068      OFFSET_Y_H = 0x20,
00069      OFFSET_Y_L = 0x21,
00070      OFFSET_Z_H = 0x22,
00071      OFFSET_Z_L = 0x23,
00072      ACT_EN = 0x24,
00073      ACT_THRESH_H = 0x25,
00074      ACT_THRESH_L = 0x26,
00075      ACT_COUNT = 0x27,
00076      FILTER = 0x28,
00077      FIFO_SAMPLES = 0x29,
00078      INT_MAP = 0x2A,
00079      SYNC = 0x2B,
00080      RANGE = 0x2C,
00081      POWER_CTL = 0x2D,
00082      SELF_TEST = 0x2E,
00083      ADXL355_RESET_REG = 0x2F
00084 } ADXL355_register_t;

```

3.3.4.10 ADXL355_status_t enum [ADXL355_status_t](#)

ADXL355 Status registers. This register includes bits that describe the various conditions of the ADXL355.

Enumerator

NVM_BUSY	NVM controller is busy with either refresh, programming, or built-in, self test (BIST).
ACTIVITY	Activity, as defined in the THRESH_ACT and COUNT_ACT registers, is detected
FIFO_OVR	FIFO has overrun, and the oldest data is lost
FIFO_FULL	FIFO watermark is reached
DATA_RDY	A complete x-axis, y-axis, and z-axis measurement was made and results can be read

Definition at line 90 of file [ADXL355.h](#).

```

00090      {
00091      NVM_BUSY = 0x10,

```

```

00092     ACTIVITY = 0x08,
00093     FIFO_OVR = 0x04,
00094     FIFO_FULL = 0x02,
00095     DATA_RDY = 0x01
00096 } ADXL355_status_t;

```

3.3.4.11 ADXL355_sync_ctl_t enum ADXL355_sync_ctl_t

ADXL355 External timing register. Use this register to control the external timing triggers.

Enumerator

EXT_CLK	Enable external clock
INT_SYNC	Internal sync
EXT_SYNC_NO_INT	External sync, no interpolation filter
EXT_SYNC_INT	External sync, interpolation filter

Definition at line 157 of file [ADXL355.h](#).

```

00157     {
00158     EXT_CLK = 0x04,
00159     INT_SYNC = 0x00,
00160     EXT_SYNC_NO_INT = 0x01,
00161     EXT_SYNC_INT = 0x02
00162 } ADXL355_sync_ctl_t;

```

3.3.5 Function Documentation

3.3.5.1 ADXL355_Data_Rdy() bool ADXL355_Data_Rdy (

SPI_Handle *spi*)

ADXL355 Data ready Function.

This function

1. Reads data ready register and returns a boolean

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
----	-------------------	------------

Returns

bool Data_Rdy

Definition at line 310 of file [ADXL355.c](#).

```

00310
00311
00312     uint8_t RxData[2];
00313     bool Data_Rdy;

```

```
00314     SPI_read_8bits(spi, ADXL355_STATUS, RxData, 1, RNW_LSB);
00315     Data_Rdy = (RxData[1] & DATA_RDY);
00316     return Data_Rdy;
00317 }
00318
00319
00320 }
```

3.3.5.2 ADXL355_DevId() `uint16_t ADXL355_DevId (` `SPI_Handle spi)`

ADXL355 Get DevID Function.

This function

1. Gets DevID from ADXL355

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
----	-------------------	------------

Returns

`uint16_t DevId`

Definition at line 36 of file [ADXL355.c](#).

```
00036
00037
00038     uint8_t RxBuffer[3];
00039     uint16_t DevId;
00040
00041     SPI_read_8bits(spi, DEVID_AD, RxBuffer, 2, RNW_LSB);
00042
00043     DevId = (RxBuffer[1]<<8) + RxBuffer[2];
00044
00045     return DevId;
00046 }
```

3.3.5.3 ADXL355_Fifo_Data() `void ADXL355_Fifo_Data (` `SPI_Handle spi,` `int32_t * Ax,` `int32_t * Ay,` `int32_t * Az,` `uint8_t Samples)`

ADXL355 FIFO data Function.

This function

1. Reads N samples of FIFO data and saves it on Ax,Ay,Az

Parameters

in	<i>SPI_Handle</i>	spi
in	<i>int32_t</i>	*Ax pointer to Ax
in	<i>int32_t</i>	*Ay pointer to Ay
in	<i>int32_t</i>	*Az pointer to Az
in	<i>uint8_t</i>	Samples N samples to read

Returns

None

Definition at line 394 of file ADXL355.c.

```

00394
00395
00396     uint8_t RxData[10];
00397     int8_t i;
00398     uint32_t uAx, uAy, uAz;
00399
00400     while (!ADXL355_Fifo_Full(spi)) {}                                // Polling the FIFO_FULL
Register of STATUS reg
00401
00402     i=0;
00403     while (i<Samples) {
00404         SPI_read_8bits(spi, FIFO_DATA, RxData, 9, RNW_LSB);
00405         uAx = (RxData[1] « 12) + (RxData[2] « 4) + (RxData[3] » 4);
00406         uAy = (RxData[4] « 12) + (RxData[5] « 4) + (RxData[6] » 4);
00407         uAz = (RxData[7] « 12) + (RxData[8] « 4) + (RxData[9] » 4);
00408         Ax[i] = u20_to_s32(uAx);
00409         Ay[i] = u20_to_s32(uAy);
00410         Az[i] = u20_to_s32(uAz);
00411         i++;
00412     }
00413 }
```

3.3.5.4 ADXL355_Fifo_Entries() *uint8_t* ADXL355_Fifo_Entries (*SPI_Handle* *spi*)

ADXL355 FIFO entries Function.

This function

1. Read FIFO entries register and returns a buffer

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns*uint8_t* RxBuffer**Definition at line 155 of file ADXL355.c.**

```

00155
00156
00157     uint8_t RxBuffer[2];
00158 }
```

```
00159     SPI_read_8bits(spi, FIFO_ENTRIES, RxBuffer, 1, RNW_LSB);  
00160  
00161     return (RxBuffer[1] & 0x7F);  
00162 }
```

3.3.5.5 ADXL355_Fifo_Full()

```
bool ADXL355_Fifo_Full (
```

SPI_Handle *spi*)

ADXL355 FIFO Full Function.

This function

1. Reads FIFO buffer register and returns a boolean

Parameters

in	SPI_Handle	<i>spi</i>
----	------------	------------

Returns

bool Fifo_Full

Definition at line 286 of file **ADXL355.c**.

```
00286  
00287  
00288     uint8_t RxData[2];  
00289     bool Fifo_Full;  
00290  
00291     SPI_read_8bits(spi, ADXL355_STATUS, RxData, 1, RNW_LSB);  
00292  
00293     Fifo_Full = (RxData[1] & 0x02) » 1;  
00294  
00295     return Fifo_Full;  
00296 }
```

3.3.5.6 ADXL355_Fifo_Samples()

```
void ADXL355_Fifo_Samples (
```

SPI_Handle *spi*,
uint8_t *Samples*)

ADXL355 FIFO samples Function.

This function

1. Writes N samples to FIFO register

Parameters

in	SPI_Handle	<i>spi</i>
in	uint8_t	<i>Samples</i>

Returns

None

Definition at line 138 of file [ADXL355.c](#).

```
00138
00139
00140     SPI_write_8bits(spi, FIFO_SAMPLES, Samples, RNW_LSB);
00141 }
```

3.3.5.7 ADXL355_Filter_Conf() `void ADXL355_Filter_Conf (`

```
    SPI_Handle spi,
    uint8_t Val )
```

ADXL355 Filter configuration Function.

This function

1. Sets value on Filter's configuration register

Parameters

in	<i>SPI_Handle</i>	spi
in	<i>uint8_t</i>	Val

Returns

None

Definition at line 177 of file [ADXL355.c](#).

```
00177
00178
00179     SPI_write_8bits(spi, FILTER, Val, RNW_LSB);
00180 }
```

3.3.5.8 ADXL355_Get_Accel_Frame() `void ADXL355_Get_Accel_Frame (`

```
    SPI_Handle spi,
    uint16_t Samples,
    int32_t * DataSensor )
```

ADXL355 Get acceleration data Function.

This function

1. Reads N samples of acceleration data and saves it on DataSensor

Parameters

in	<i>SPI_Handle</i>	spi
in	<i>uint16_t</i>	Samples N samples to read
in	<i>int32_t</i>	*DataSensor pointer to datasensor

Generated by Doxygen

Returns

None

Definition at line 336 of file [ADXL355.c](#).

```

00336
00337
00338     int i=0;
00339     int32_t Ax[512], Ay[512], Az[512];
00340     int32_t xMean, yMean, zMean;
00341     double xRms, yRms, zRms;
00342
00343     xMean=0;
00344     yMean=0;
00345     zMean=0;
00346     while (i<Samples) {
00347         while(!ADXL355_Data_Rdy(spi)) {}
00348         Ax[i] = ADXL355_XData(spi);
00349         Ay[i] = ADXL355_YData(spi);
00350         Az[i] = ADXL355_ZData(spi);
00351         xMean += Ax[i];
00352         yMean += Ay[i];
00353         zMean += Az[i];
00354         i++;
00355     }
00356
00357     xMean = xMean / Samples;
00358     yMean = yMean / Samples;
00359     zMean = zMean / Samples;
00360
00361     xRms = 0.0;
00362     yRms = 0.0;
00363     zRms = 0.0;
00364     for (i=0;i<Samples;i++) {
00365         xRms += ( (Ax[i]-xMean)*(Ax[i]-xMean) );
00366         yRms += ( (Ay[i]-yMean)*(Ay[i]-yMean) );
00367         zRms += ( (Az[i]-zMean)*(Az[i]-zMean) );
00368     }
00369
00370     DataSensor[0] = xMean;
00371     DataSensor[1] = yMean;
00372     DataSensor[2] = zMean;
00373     DataSensor[3] = (int32_t)sqrt(xRms/Samples);
00374     DataSensor[4] = (int32_t)sqrt(yRms/Samples);
00375     DataSensor[5] = (int32_t)sqrt(zRms/Samples);
00376 }
```

3.3.5.9 ADXL355_PartId() `uint16_t ADXL355_PartId (`
`SPI_Handle Spi)`

ADXL355 Get PartID Function.

This function

1. Gets PartID from ADXL355

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns`uint16_t PartID`Definition at line 60 of file [ADXL355.c](#).

```

00060
00061
00062     uint8_t RxBuffer[3];
00063     uint16_t PartId;
00064
00065     SPI_read_8bits(spi, PARTID, RxBuffer, 2, RNW_LSB);
00066
00067     PartId = (RxBuffer[1]«8) + RxBuffer[2];
00068
00069     return PartId;
00070 }
```

3.3.5.10 ADXL355_Power_Conf() void ADXL355_Power_Conf (

```

    SPI_Handle spi,
    uint8_t Val )
```

ADXL355 Power Configuration Function.

This function

1. Sets Power configuration value on Power configuration register

Parameters

in	<i>SPI_Handle</i>	spi
in	<i>uint8_t</i>	Val

Returns

None

Definition at line 102 of file [ADXL355.c](#).

```

00102
00103
00104     SPI_write_8bits(spi, POWER_CTL, Val, RNW_LSB);
00105 }
```

3.3.5.11 ADXL355_Range_Conf() void ADXL355_Range_Conf (

```

    SPI_Handle spi,
    uint8_t Val )
```

ADXL355 Range Configuration Function.

This function

1. Sets Range configuration value on Range configuration register

Parameters

in	<i>SPI_Handle</i>	spi
in	<i>uint8_t</i>	Val

Returns

None

Definition at line 120 of file [ADXL355.c](#).

```
00120
00121
00122     SPI_write_8bits(spi, RANGE, Val, RNW_LSB);
00123 }
```

3.3.5.12 ADXL355_Reset() void ADXL355_Reset (

SPI_Handle *spi*)

ADXL355 Reset Function.

This function

1. Resets ADXL355

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
----	-------------------	------------

Returns

None

Definition at line 84 of file [ADXL355.c](#).

```
00084
00085
00086     SPI_write_8bits(spi, ADXL355_RESET_REG, ADXL355_RESET, RNW_LSB);
00087 }
```

3.3.5.13 ADXL355_Temp() float ADXL355_Temp (

SPI_Handle *spi*)

ADXL355 Read Temperature register Function.

This function

1. Reads Temperature from Temperature registers

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
----	-------------------	------------

Returns

$(\text{uData} - \text{t_bias}) / \text{t_sens} + 25.0$; Temperature data

Definition at line 263 of file [ADXL355.c](#).

```
00263                                     {
00264     uint8_t RxData[3];
00265     uint16_t uData;
00266
00267     SPI_read_8bits(spi, TEMP2, RxData, 2, RNW_LSB);
00268     uData = ((RxData[1] & 0x0F) << 8) + RxData[2];
00270
00271     return (uData - t_bias) / t_sens + 25.0;
00272 }
```

3.3.5.14 ADXL355_XData() int32_t ADXL355_XData (

SPI_Handle *spi*)

ADXL355 Read data X Function.

This function

1. Reads data from X axis register

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
----	-------------------	------------

Returns

uData Axis X data

Definition at line 194 of file [ADXL355.c](#).

```
00194                                     {
00195
00196     uint8_t RxData[4];
00197     uint32_t uData;
00198
00199     SPI_read_8bits(spi, XDATA3, RxData, 3, RNW_LSB);
00200     uData = (int32_t)(RxData[1] << 12) + (int32_t)(RxData[2] << 4) + (int32_t)(RxData[3] >> 4);
00201
00202     return u20_to_s32(uData);
00203 }
```

3.3.5.15 ADXL355_YData() int32_t ADXL355_YData (

SPI_Handle *spi*)

ADXL355 Read data Y Function.

This function

1. Reads data from Y axis register

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

uData Axis Y data

Definition at line 217 of file [ADXL355.c](#).

```
00217
00218
00219     uint8_t RxData[4];
00220     uint32_t uData;
00221
00222     SPI_read_8bits(spi, YDATA3, RxData, 3, RNW_LSB);
00223     uData = (int32_t)(RxData[1] << 12) + (int32_t)(RxData[2] << 4) + (int32_t)(RxData[3] >> 4);
00224
00225     return u20_to_s32(uData);
00226 }
```

3.3.5.16 ADXL355_ZData() int32_t ADXL355_ZData (

SPI_Handle *spi*)

ADXL355 Read data Z Function.

This function

1. Reads data from Z axis register

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

uData Axis Z data

Definition at line 240 of file [ADXL355.c](#).

```
00240
00241
00242     uint8_t RxData[4];
00243     uint32_t uData;
00244
00245     SPI_read_8bits(spi, ZDATA3, RxData, 3, RNW_LSB);
00246     uData = (int32_t)(RxData[1] << 12) + (int32_t)(RxData[2] << 4) + (int32_t)(RxData[3] >> 4);
00247
00248     return u20_to_s32(uData);
00249 }
```

3.3.5.17 u20_to_s32() int32_t u20_to_s32 (

uint32_t *uData*)

ADXL355 u20_to_s32 Function.

This function

1. Converts data to int

Parameters

in	<i>uint32_t</i>	uData Accelerometer data
	<i>_t</i>	

Returns

int32_t iData Accelerometer data integer

Definition at line 427 of file [ADXL355.c](#).

```
00427
00428
00429     int32_t iData;
00430
00431     if (uData & 0x80000) {
00432         iData = uData | 0xFFFF0000;
00433     } else {
00434         iData = (int32_t)uData;
00435     }
00436
00437     return iData;
00438 }
```

3.3.6 Variable Documentation**3.3.6.1 axis355_sens float axis355_sens**

Definition at line 29 of file [ADXL355.h](#).

3.3.6.2 calib_data [ADXL355_calibdata_t](#) calib_data

Definition at line 200 of file [ADXL355.h](#).

3.4 ADXL355.h

```
00001
00009 #ifndef ADXL355_H_
00010 #define ADXL355_H_
00011
00012 //*****
00013 //           INCLUDES
00014 //*****
00015 #include <ti/drivers/SPI.h>
00016 #include <stdbool.h>
00017
00018 //*****
00019 //           DEFINES
00020 //*****
00021 #define ADXL355_DEVICE_ID      0xAD
00022 #define ADXL355_RESET          0x52
00024 //*****
00025 //           CONSTANTS & VARIABLES
00026 //*****
00027 const static float t_sens = -9.05;
00028 const static float t_bias = 1852;
00029 float axis355_sens;
00030
00031 //*****
00032 //           TYPEDEFS
00033 //*****
```

```

00034
00038 typedef struct {
00039     float S[2][2];
00040     float St;
00041     float B[2][0];
00042 } ADXL355_calibdata_t;
00043
00047 typedef enum {
00048     DEVID_AD = 0x00,
00049     DEVID_MST = 0x01,
00050     PARTID = 0x02,
00051     REVID = 0x03,
00052     ADXL355_STATUS = 0x04,
00053     FIFO_ENTRIES = 0x05,
00054     TEMP2 = 0x06,
00055     TEMP1 = 0x07,
00056     XDATA3 = 0x08,
00057     XDATA2 = 0x09,
00058     XDATA1 = 0x0A,
00059     YDATA3 = 0x0B,
00060     YDATA2 = 0x0C,
00061     YDATA1 = 0x0D,
00062     ZDATA3 = 0x0E,
00063     ZDATA2 = 0x0F,
00064     ZDATA1 = 0x10,
00065     FIFO_DATA = 0x11,
00066     OFFSET_X_H = 0x1E,
00067     OFFSET_X_L = 0x1F,
00068     OFFSET_Y_H = 0x20,
00069     OFFSET_Y_L = 0x21,
00070     OFFSET_Z_H = 0x22,
00071     OFFSET_Z_L = 0x23,
00072     ACT_EN = 0x24,
00073     ACT_THRESH_H = 0x25,
00074     ACT_THRESH_L = 0x26,
00075     ACT_COUNT = 0x27,
00076     FILTER = 0x28,
00077     FIFO_SAMPLES = 0x29,
00078     INT_MAP = 0x2A,
00079     SYNC = 0x2B,
00080     RANGE = 0x2C,
00081     POWER_CTL = 0x2D,
00082     SELF_TEST = 0x2E,
00083     ADXL355_RESET_REG = 0x2F
00084 } ADXL355_register_t;
00085
00090 typedef enum {
00091     NVM_BUSY = 0x10,
00092     ACTIVITY = 0x08,
00093     FIFO_OVR = 0x04,
00094     FIFO_FULL = 0x02,
00095     DATA_RDY = 0x01
00096 } ADXL355_status_t;
00097
00101 typedef enum {
00102     DRDY_OFF = 0x04,
00103     TEMP_OFF = 0x02,
00104     STANDBY = 0x01,
00105     MEASUREMENT = 0x00
00106 } ADXL355_modes_t;
00107
00111 typedef enum {
00112     ACT_Z = 0x04,
00113     ACT_Y = 0x02,
00114     ACT_X = 0x01
00115 } ADXL355_act_ctl_t;
00116
00121 typedef enum {
00122     HPFOFF = 0x00,
00123     HPF247 = 0x10,
00124     HPF62 = 0x20,
00125     HPF15 = 0x30,
00126     HPF3 = 0x40,
00127     HPF09 = 0x50,
00128     HPF02 = 0x60,
00129     ODR4000HZ = 0x00,
00130     ODR2000HZ = 0x01,
00131     ODR1000HZ = 0x02,
00132     ODR500HZ = 0x03,
00133     ODR250HZ = 0x04,
00134     ODR125Hz = 0x05,
00135     ODR62HZ = 0x06,
00136     ODR31Hz = 0x07,
00137     ODR15Hz = 0x08,
00138     ODR7Hz = 0x09,
00139     ODR3HZ = 0x0A
00140 } ADXL355_filter_ctl_t;

```

```

00141
00147 typedef enum {
00148     OVR_EN = 0x04,
00149     FULL_EN = 0x02,
00150     RDY_EN = 0x01
00151 } ADXL355_intmap_ctl_t;
00152
00157 typedef enum {
00158     EXT_CLK = 0x04,
00159     INT_SYNC = 0x00,
00160     EXT_SYNC_NO_INT = 0x01,
00161     EXT_SYNC_INT = 0x02
00162 } ADXL355_sync_ctl_t;
00163
00167 typedef enum {
00168     I2C_HIGH_SPEED = 0x80,
00169     I2C_FAST_MODE = 0x00
00170 } ADXL355_i2c_speed_t;
00171
00175 typedef enum {
00176     INT_POL_LOW = 0x00,
00177     INT_POL_HIGH = 0x40
00178 } ADXL355_int_pol_t;
00179
00183 typedef enum {
00184     RANGE2G = 0x01,
00185     RANGE4G = 0x02,
00186     RANGE8G = 0x03
00187 } ADXL355_range_ctl_t;
00188
00192 typedef enum {
00193     ST2 = 0x02,
00194     ST1 = 0x01
00195 } ADXL355_int_ctl_t;
00196
00197 //*****
00198 //          GLOBALS
00199 //*****
00200 ADXL355_calibdata_t calib_data;
00201
00202 //*****
00203 //          FUNCTION PROTOTYPES
00204 //*****
00205 void ADXL355_Reset(SPI_Handle spi);
00206 uint16_t ADXL355_DevId(SPI_Handle spi);
00207 uint16_t ADXL355_PartId(SPI_Handle spi);
00208 void ADXL355_Power_Conf(SPI_Handle spi, uint8_t Val);
00209 void ADXL355_Range_Conf(SPI_Handle spi, uint8_t Val);
00210 void ADXL355_Fifo_Samples(SPI_Handle spi, uint8_t Samples);
00211 bool ADXL355_Fifo_Full(SPI_Handle spi);
00212 bool ADXL355_Data_Rdy(SPI_Handle spi);
00213 uint8_t ADXL355_Fifo_Entries(SPI_Handle spi);
00214 void ADXL355_Filter_Conf(SPI_Handle spi, uint8_t Val);
00215 int32_t ADXL355_XData(SPI_Handle spi);
00216 int32_t ADXL355_YData(SPI_Handle spi);
00217 int32_t ADXL355_ZData(SPI_Handle spi);
00218 float ADXL355_Temp(SPI_Handle spi);
00219 void ADXL355_Get_Accel_Frame(SPI_Handle spi, uint16_t Samples, int32_t *DataSensor);
00220 void ADXL355_Fifo_Data(SPI_Handle spi, int32_t *Ax, int32_t *Ay, int32_t *Az, uint8_t Samples);
00221 int32_t u20_to_s32(uint32_t uData);
00222
00223 #endif /* ADXL355_H_ */

```

3.5 BME280.c File Reference

Functions to configure and read data from BME280 Pressure, Temperature and Humidity sensor.

```

#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
#include <ti/drivers/SPI.h>
#include <ti/drivers/I2C.h>
#include "STARPORTS_App.h"
#include "BME280.h"
#include "hal_SPI.h"
#include "hal_I2C.h"

```

Functions

- `uint8_t BME280_DevId (SPI_HandleTypeDef spi)`
BME280 Get DevID Function.
- `void BME280_Read_Calib (SPI_HandleTypeDef spi, struct bme280_calib_data *MyCalib)`
BME280 Read Calibration Function.
- `uint8_t BME280_Read_Config (SPI_HandleTypeDef spi)`
BME280 Read Configuration Function.
- `uint8_t BME280_Read_Ctrl_Hum (SPI_HandleTypeDef spi)`
BME280 Read Control Humidity Function.
- `uint8_t BME280_Read_Ctrl_Meas (SPI_HandleTypeDef spi)`
BME280 Read Control Measurement Function.
- `uint16_t BME280_Read_Humidity (SPI_HandleTypeDef spi)`
BME280 Read Humidity Function.
- `uint32_t BME280_Read_Pressure (SPI_HandleTypeDef spi)`
BME280 Read Pressure Function.
- `uint8_t BME280_Read_Status (SPI_HandleTypeDef spi)`
BME280 Read Status Function.
- `uint32_t BME280_Read_Temperature (SPI_HandleTypeDef spi)`
BME280 Read Temperature Function.
- `void BME280_Reset (SPI_HandleTypeDef spi)`
BME280 Reset Function.
- `void BME280_Write_Config (SPI_HandleTypeDef spi, uint8_t Val)`
BME280 Write Configuration Function.
- `void BME280_Write_Ctrl_Hum (SPI_HandleTypeDef spi, uint8_t Val)`
BME280 Write Control Humidity Function.
- `void BME280_Write_Ctrl_Meas (SPI_HandleTypeDef spi, uint8_t Val)`
BME280 Write Control Measurement Function.
- `uint32_t compensate_humidity (uint16_t HumidityUncomp, struct bme280_calib_data *MyCalib)`
BME280 Compensate Humidity Function.
- `uint32_t compensate_pressure (uint32_t PressureUncomp, struct bme280_calib_data *MyCalib)`
BME280 Compensate pressure Function.
- `int32_t compensate_temperature (uint32_t TemperatureUncomp, struct bme280_calib_data *MyCalib)`
BME280 Compensate temperature Function.

3.5.1 Detailed Description

Functions to configure and read data from BME280 Pressure, Temperature and Humidity sensor.

Version

1.0

Date

07/05/2019

Author

A.Irizar @title BME280 Pressure, Temperature and Humidity sensor functions

Definition in file [BME280.c](#).

3.5.2 Function Documentation

3.5.2.1 BME280_DevId() `uint8_t BME280_DevId (`
`SPI_HandleTypeDef spi)`

BME280 Get DevID Function.

This function

1. Gets DevID from BME280 sensor

Parameters

in	<i>SPI_HandleTypeDef</i>	spi
----	--------------------------	-----

Returns

`uint8_t RxBuffer DEVID`

Definition at line 56 of file [BME280.c](#).

```
00056
00057
00058     uint8_t RxBuffer[2];
00059
00060     SPI_read_8bits(spi, DEVID, RxBuffer, 1, RNW_MSB);
00061     return RxBuffer[1];
00062 }
```

3.5.2.2 BME280_Read_Calib() `void BME280_Read_Calib (`
`SPI_HandleTypeDef spi,`
`struct bme280_calib_data * MyCalib)`

BME280 Read Calibration Function.

This function

1. Reads calibration register on BME280 sensor

Parameters

in	<i>SPI_HandleTypeDef</i>	spi
	<i>struct bme280_calib_data</i>	*MyCalib pointer to BME280 calibration data

Returns

None

Definition at line 214 of file [BME280.c](#).

```

00214
00215
00216     uint8_t RxBuffer00[CALIB00_NDATA+1];
00217     uint8_t RxBuffer26[CALIB26_NDATA+1];
00218
00219     SPI_read_8bits(spi, CALIB00, RxBuffer00, CALIB00_NDATA, RNW_MSB);
00220     SPI_read_8bits(spi, CALIB26, RxBuffer26, CALIB26_NDATA, RNW_MSB);
00221
00222     MyCalib->dig_T1 = (RxBuffer00[1] + (RxBuffer00[2]«8));
00223     MyCalib->dig_T2 = (RxBuffer00[3] + (RxBuffer00[4]«8));
00224     MyCalib->dig_T3 = (RxBuffer00[5] + (RxBuffer00[6]«8));
00225
00226
00227     MyCalib->dig_P1 = (RxBuffer00[7] + (RxBuffer00[8]«8));
00228     MyCalib->dig_P2 = (RxBuffer00[9] + (RxBuffer00[10]«8));
00229     MyCalib->dig_P3 = (RxBuffer00[11] + (RxBuffer00[12]«8));
00230     MyCalib->dig_P4 = (RxBuffer00[13] + (RxBuffer00[14]«8));
00231     MyCalib->dig_P5 = (RxBuffer00[15] + (RxBuffer00[16]«8));
00232     MyCalib->dig_P6 = (RxBuffer00[17] + (RxBuffer00[18]«8));
00233     MyCalib->dig_P7 = (RxBuffer00[19] + (RxBuffer00[20]«8));
00234     MyCalib->dig_P8 = (RxBuffer00[21] + (RxBuffer00[22]«8));
00235     MyCalib->dig_P9 = (RxBuffer00[23] + (RxBuffer00[24]«8));
00236
00237     MyCalib->dig_H1 = RxBuffer00[26];
00238     MyCalib->dig_H2 = RxBuffer26[1] + (RxBuffer26[2]«8);
00239     MyCalib->dig_H3 = RxBuffer26[3];
00240     MyCalib->dig_H4 = (RxBuffer26[4]«4) + (RxBuffer26[5] & 0x0F);
00241     MyCalib->dig_H5 = ((RxBuffer26[5]«4) & 0x0F) + (RxBuffer26[6]«4);
00242     MyCalib->dig_H6 = RxBuffer26[7];
00243 }
```

3.5.2.3 BME280_Read_Config() `uint8_t BME280_Read_Config (`

`SPI_Handle Spi)`

BME280 Read Configuration Function.

This function

1. Reads Configuration register on BME280 sensor

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
----	-------------------	------------

Returns

`uint8_t RxBuffer`

Definition at line 171 of file [BME280.c](#).

```

00171
00172
00173     uint8_t RxBuffer[2];
00174
00175     SPI_read_8bits(spi, CONFIG, RxBuffer, 1, RNW_MSB);
00176
00177     return RxBuffer[1];
00178 }
```

3.5.2.4 BME280_Read_Ctrl_Hum() `uint8_t BME280_Read_Ctrl_Hum (`
`SPI_Handle spi)`

BME280 Read Control Humidity Function.

This function

1. Reads control humidity register on BME280 sensor

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

`uint8_t RxBuffer` Humidity value

Definition at line 94 of file [BME280.c](#).

```
00094                                     {  
00095  
00096     uint8_t RxBuffer[2];  
00097  
00098     SPI_read_8bits(spi, CTRL_HUM, RxBuffer, 1, RNW_MSB);  
00099     return RxBuffer[1];  
00100 }
```

3.5.2.5 BME280_Read_Ctrl_Meas() `uint8_t BME280_Read_Ctrl_Meas (`
`SPI_Handle spi)`

BME280 Read Control Measurement Function.

This function

1. Reads control measurement register on BME280 sensor

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

`uint8_t RxBuffer`

Definition at line 132 of file [BME280.c](#).

```
00132                                     {  
00133  
00134     uint8_t RxBuffer[2];  
00135  
00136     SPI_read_8bits(spi, CTRL_MEAS, RxBuffer, 1, RNW_MSB);  
00137     return RxBuffer[1];  
00138 }  
00139 }
```

3.5.2.6 BME280_Read_Humidity() `uint16_t BME280_Read_Humidity (`
 `SPI_Handle spi)`

BME280 Read Humidity Function.

This function

1. Reads Humidity register on BME280 sensor

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

`uint16_t Humidity`

Definition at line 280 of file [BME280.c](#).

```
00280 {  
00281  
00282     uint8_t RxBuffer[3];  
00283     uint16_t Humidity;  
00284     SPI_read_8bits(spi, HUM_MSB, RxBuffer, 2, RNW_MSB);  
00285     Humidity = (RxBuffer[1]«8) + RxBuffer[2];  
00286     return Humidity;  
00287 }  
00288  
00289 }
```

3.5.2.7 BME280_Read_Pressure() `uint32_t BME280_Read_Pressure (`
 `SPI_Handle spi)`

BME280 Read Pressure Function.

This function

1. Reads pressure register on BME280 sensor

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

`uint32_t Pressure`

Definition at line 257 of file [BME280.c](#).

```
00257 {  
00258  
00259     uint8_t RxBuffer[4];  
00260     uint32_t Pressure;  
00261     SPI_read_8bits(spi, PRESS_MSB, RxBuffer, 3, RNW_MSB);  
00262 }
```

```

00263     Pressure = ((RxBuffer[1]«12) + (RxBuffer[2]«4) + (RxBuffer[3]«4));
00264
00265     return Pressure;
00266 }
```

3.5.2.8 BME280_Read_Status()

uint8_t BME280_Read_Status (

SPI_Handle *spi*)

BME280 Read Status Function.

This function

1. Reads Status register on BME280 sensor

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
----	-------------------	------------

Returns

uint8_t RxBuffer

Definition at line 192 of file [BME280.c](#).

```

00192
00193
00194     uint8_t RxBuffer[2];
00195
00196     SPI_read_8bits(spi, BME280_STATUS, RxBuffer, 1, RNW_MSB);
00197
00198     return RxBuffer[1];
00199 }
```

3.5.2.9 BME280_Read_Temperature()

uint32_t BME280_Read_Temperature (

SPI_Handle *spi*)

BME280 Read Temperature Function.

This function

1. Reads Temperature register on BME280 sensor

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
----	-------------------	------------

Returns

uint32_t Temperature

Definition at line 303 of file [BME280.c](#).

```
00303     uint8_t RxBuffer[4];  
00304     uint32_t Temperature;  
00305     SPI_read_8bits(spi, TEMP_MSB, RxBuffer, 3, RNW_MSB);  
00306     Temperature = ((RxBuffer[1]«12) + (RxBuffer[2]«4) + (RxBuffer[3]«4));  
00307     return Temperature;  
00312 }
```

3.5.2.10 BME280_Reset() void BME280_Reset (

SPI_Handle *spi*)

BME280 Reset Function.

This function

1. Resets BME280 sensor

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
----	-------------------	------------

Returns

none

Definition at line 39 of file [BME280.c](#).

```
00039 {  
00040  
00041     SPI_write_8bits(spi, BME280_RESET_REG, BME280_RESET, RNW_MSB);  
00042 }
```

3.5.2.11 BME280_Write_Config() void BME280_Write_Config (

SPI_Handle *spi*,
uint8_t *Val*)

BME280 Write Configuration Function.

This function

1. Writes Configuration register on BME280 sensor

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
in	<i>uint8_t</i>	<i>Val</i>

Returns

none

Definition at line 154 of file [BME280.c](#).

```
00154
00155
00156     SPI_write_8bits(spi, CONFIG, Val, RNW_MSB);
00157 }
```

3.5.2.12 BME280_Write_Ctrl_Hum() `void BME280_Write_Ctrl_Hum (`

```
    SPI_Handle spi,
    uint8_t Val )
```

BME280 Write Control Humidity Function.

This function

1. Writes control humidity register on BME280 sensor

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

none

Definition at line 76 of file [BME280.c](#).

```
00076
00077
00078
00079     SPI_write_8bits(spi, CTRL_HUM, Val, RNW_MSB);
00080 }
```

3.5.2.13 BME280_Write_Ctrl_Meas() `void BME280_Write_Ctrl_Meas (`

```
    SPI_Handle spi,
    uint8_t Val )
```

BME280 Write Control Measurement Function.

This function

1. Writes control measurement register on BME280 sensor

Parameters

in	<i>SPI_Handle</i>	spi
in	<i>uint8_t</i>	Val

Returns

none

Definition at line 115 of file BME280.c.

```
00115
00116
00117     SPI_write_8bits(spi, CTRL_MEAS, Val, RNW_MSB);
00118 }
```

3.5.2.14 compensate_humidity() `uint32_t compensate_humidity (`

```
    uint16_t Humi di tyUncomp,
    struct bme280_calib_data * MyCal i b )
```

BME280 Compensate Humidity Function.

This function

1. Compensates the value of the humidity read from the BME280 registers

Precondition`compensate_humidity()` has been called**Parameters**

in	<code>uint16_t</code>	HumidityUncomp Uncompensated value of Humidity as read from the BME280 registers
in	<code>struct</code>	<code>bme280_calib_data</code> Pointer to an struct that holds the calibration values of the sensor

Returns`uint32_t humidity` Returns the compensated humidity value in units 1024 * relative humidity**Definition at line 329 of file BME280.c.**

```
00329
00330     int32_t var1;
00331     int32_t var2;
00332     int32_t var3;
00333     int32_t var4;
00334     int32_t var5;
00335     uint32_t humidity;
00336     uint32_t humidity_max = 102400;
00337
00338     var1 = MyCalib->t_fine - ((int32_t)76800);
00339     var2 = (int32_t)(HumidityUncomp * 16384);
00340     var3 = (int32_t)((int32_t)MyCalib->dig_H4) * 1048576;
00341     var4 = ((int32_t)MyCalib->dig_H5) * var1;
00342     var5 = (((var2 - var3) - var4) + (int32_t)16384) / 32768;
00343     var2 = (var1 * ((int32_t)MyCalib->dig_H6)) / 1024;
00344     var3 = (var1 * ((int32_t)MyCalib->dig_H3)) / 2048;
00345     var4 = ((var2 * (var3 + (int32_t)32768)) / 1024) + (int32_t)2097152;
00346     var2 = ((var4 * ((int32_t)MyCalib->dig_H2)) + 8192) / 16384;
00347     var3 = var5 * var2;
00348     var4 = ((var3 / 32768) * (var3 / 32768)) / 128;
00349     var5 = var3 - ((var4 * ((int32_t)MyCalib->dig_H1)) / 16);
00350     var5 = (var5 < 0 ? 0 : var5);
00351     var5 = (var5 > 419430400 ? 419430400 : var5);
00352     humidity = (uint32_t)(var5 / 4096);
00353     if (humidity > humidity_max)
00354     {
00355         humidity = humidity_max;
00356     }
00357
00358     return humidity;
```

```
00359 }
```

3.5.2.15 `compensate_pressure()` `uint32_t compensate_pressure (`
 `uint32_t PressureUncomp,`
 `struct bme280_calib_data * MyCalib)`

BME280 Compensate pressure Function.

This function

1. Compensates the value of the pressure read from the BME280 registers

Precondition

`compensate_pressure()` has been called

Parameters

in	<code>uint32_t</code>	PressureUncomp Uncompensated value of Pressure as read from the BME280 registers
in	<code>struct</code>	<code>bme280_calib_data</code> Pointer to an struct that holds the calibration values of the sensor

Returns

`uint32_t pressure` Returns the compensated pressure value in units of Pascals

Definition at line 414 of file [BME280.c](#).

```
00415 {
00416     int32_t var1;
00417     int32_t var2;
00418     int32_t var3;
00419     int32_t var4;
00420     uint32_t var5;
00421     uint32_t pressure;
00422     uint32_t pressure_min = 30000;
00423     uint32_t pressure_max = 110000;
00424
00425     var1 = (((int32_t)MyCalib->t_fine) / 2) - (int32_t)64000;
00426     var2 = (((var1 / 4) * (var1 / 4)) / 2048) * ((int32_t)MyCalib->dig_P6);
00427     var2 = var2 + ((var1 * ((int32_t)MyCalib->dig_P5)) * 2);
00428     var2 = (var2 / 4) + (((int32_t)MyCalib->dig_P4) * 65536);
00429     var3 = ((MyCalib->dig_P3) * (((var1 / 4) * (var1 / 4)) / 8192)) / 8;
00430     var4 = (((int32_t)MyCalib->dig_P2) * var1) / 2;
00431     var1 = (var3 + var4) / 262144;
00432     var1 = (((32768 + var1)) * ((int32_t)MyCalib->dig_P1)) / 32768;
00433
00434     /* avoid exception caused by division by zero */
00435     if (var1)
00436     {
00437         var5 = (uint32_t)((uint32_t)1048576) - PressureUncomp;
00438         pressure = ((uint32_t)(var5 - (uint32_t)(var2 / 4096))) * 3125;
00439         if (pressure < 0x80000000)
00440         {
00441             pressure = (pressure << 1) / ((uint32_t)var1);
00442         }
00443         else
00444         {
00445             pressure = (pressure / (uint32_t)var1) * 2;
00446         }
00447         var1 = (((int32_t)MyCalib->dig_P9) * ((int32_t)((pressure / 8) * (pressure / 8)) / 8192)) /
00448             4096;
00449         var2 = (((int32_t)(pressure / 4)) * ((int32_t)MyCalib->dig_P8)) / 8192;
```

```

00449     pressure = (uint32_t)((int32_t)pressure + ((var1 + var2 + MyCalib->dig_P7) / 16));
00450     if (pressure < pressure_min)
00451     {
00452         pressure = pressure_min;
00453     }
00454     else if (pressure > pressure_max)
00455     {
00456         pressure = pressure_max;
00457     }
00458 }
00459 else
00460 {
00461     pressure = pressure_min;
00462 }
00463
00464 return pressure;
00465 }
```

3.5.2.16 compensate_temperature()

```
int32_t compensate_temperature (
    uint32_t TemperatureUncomp,
    struct bme280_calib_data * MyCalib )
```

BME280 Compensate temperature Function.

This function

1. Compensates the value of the temperature read from the BME280 registers

Precondition

`compensate_temperature()` has been called

Parameters

in	<code>uint32_t</code>	TemperatureUncomp Uncompensated value of Temperature as read from the BME280 registers
in	<code>struct</code>	<code>bme280_calib_data</code> Pointer to an struct that holds the calibration values of the sensor

Returns

`uint32_t` temperature Returns the compensated temperature value in units $100 \times C$

Definition at line 375 of file [BME280.c](#).

```

00375
00376     int32_t var1;
00377     int32_t var2;
00378     int32_t temperature;
00379     int32_t temperature_min = -4000;
00380     int32_t temperature_max = 8500;
00381
00382     var1 = (int32_t)((TemperatureUncomp / 8) - ((int32_t)MyCalib->dig_T1 * 2));
00383     var1 = (var1 * ((int32_t)MyCalib->dig_T2)) / 2048;
00384     var2 = (int32_t)((TemperatureUncomp / 16) - ((int32_t)MyCalib->dig_T1));
00385     var2 = (((var2 * var2) / 4096) * ((int32_t)MyCalib->dig_T3)) / 16384;
00386     MyCalib->t_fine = var1 + var2;
00387     temperature = (MyCalib->t_fine * 5 + 128) / 256;
00388     if (temperature < temperature_min)
00389     {
00390         temperature = temperature_min;
00391     }
00392     else if (temperature > temperature_max)
00393     {
```

```

00394     temperature = temperature_max;
00395 }
00396
00397     return temperature;
00398 }
```

3.6 BME280.c

```

00001
00010 //*****
00011 //           INCLUDES
00012 //*****
00013 #include <stdlib.h>
00014 #include <stdbool.h>
00015 #include <math.h>
00016 #include <ti/drivers/SPI.h>
00017 #include <ti/drivers/I2C.h>
00018 #include "STARPORTS_App.h"
00019 #include "BME280.h"
00020 #include "hal_SPI.h"
00021 #include "hal_I2C.h"
00022
00023 //*****
00024 //           FUNCTIONS
00025 //*****
00026
00027 //*****
00028 //
00029 //
00030 //
00031 //
00032 //
00033 //
00034 void BME280_Reset(SPI_Handle spi) {
00040
00041     SPI_write_8bits(spi, BME280_RESET_REG, BME280_RESET, RNW_MSB);
00042 }
00043
00044 //*****
00045 //
00046 //
00047 //
00048 uint8_t BME280_DevId(SPI_Handle spi) {
00055
00056     uint8_t RxBuffer[2];
00057
00058     SPI_read_8bits(spi, DEVID, RxBuffer, 1, RNW_MSB);
00059     return RxBuffer[1];
00060 }
00061
00062
00063
00064 //*****
00065 //
00066 //
00067 void BME280_Write_Ctrl_Hum(SPI_Handle spi, uint8_t Val) {
00077
00078
00079     SPI_write_8bits(spi, CTRL_HUM, Val, RNW_MSB);
00080 }
00081
00082 //*****
00083 //
00084 //
00085 uint8_t BME280_Read_Ctrl_Hum(SPI_Handle spi) {
00095
00096     uint8_t RxBuffer[2];
00097
00098     SPI_read_8bits(spi, CTRL_HUM, RxBuffer, 1, RNW_MSB);
00099     return RxBuffer[1];
00100 }
00101
00102 //*****
00103 //
00104 //
00105 void BME280_Write_Ctrl_Meas(SPI_Handle spi, uint8_t Val) {
00116
00117     SPI_write_8bits(spi, CTRL_MEAS, Val, RNW_MSB);
00118 }
00119
00120 //*****
00121 //
00130 //
00131 //
00132 uint8_t BME280_Read_Ctrl_Meas(SPI_Handle spi) {
00133 }
```

```

00134     uint8_t RxBuffer[2];
00135
00136     SPI_read_8bits(spi, CTRL_MEAS, RxBuffer, 1, RNW_MSB);
00137
00138     return RxBuffer[1];
00139 }
00140
00141 //*****
00142 //
00152 //
00153 //*****
00154 void BME280_Write_Config(SPI_HandleTypeDef spi, uint8_t Val) {
00155
00156     SPI_write_8bits(spi, CONFIG, Val, RNW_MSB);
00157 }
00158
00159 //*****
00160 //
00169 //
00170 //*****
00171 uint8_t BME280_Read_Config(SPI_HandleTypeDef spi) {
00172
00173     uint8_t RxBuffer[2];
00174
00175     SPI_read_8bits(spi, CONFIG, RxBuffer, 1, RNW_MSB);
00176
00177     return RxBuffer[1];
00178 }
00179
00180 //*****
00181 //
00190 //
00191 //*****
00192 uint8_t BME280_Read_Status(SPI_HandleTypeDef spi) {
00193
00194     uint8_t RxBuffer[2];
00195
00196     SPI_read_8bits(spi, BME280_STATUS, RxBuffer, 1, RNW_MSB);
00197
00198     return RxBuffer[1];
00199 }
00200
00201 //*****
00202 //
00212 //
00213 //*****
00214 void BME280_Read_Calib(SPI_HandleTypeDef spi, struct bme280_calib_data *MyCalib) {
00215
00216     uint8_t RxBuffer00[CALIB00_NDATA+1];
00217     uint8_t RxBuffer26[CALIB26_NDATA+1];
00218
00219     SPI_read_8bits(spi, CALIB00, RxBuffer00, CALIB00_NDATA, RNW_MSB);
00220     SPI_read_8bits(spi, CALIB26, RxBuffer26, CALIB26_NDATA, RNW_MSB);
00221
00222     MyCalib->dig_T1 = (RxBuffer00[1] + (RxBuffer00[2]<<8));
00223     MyCalib->dig_T2 = (RxBuffer00[3] + (RxBuffer00[4]<<8));
00224     MyCalib->dig_T3 = (RxBuffer00[5] + (RxBuffer00[6]<<8));
00225
00226
00227     MyCalib->dig_P1 = (RxBuffer00[7] + (RxBuffer00[8]<<8));
00228     MyCalib->dig_P2 = (RxBuffer00[9] + (RxBuffer00[10]<<8));
00229     MyCalib->dig_P3 = (RxBuffer00[11] + (RxBuffer00[12]<<8));
00230     MyCalib->dig_P4 = (RxBuffer00[13] + (RxBuffer00[14]<<8));
00231     MyCalib->dig_P5 = (RxBuffer00[15] + (RxBuffer00[16]<<8));
00232     MyCalib->dig_P6 = (RxBuffer00[17] + (RxBuffer00[18]<<8));
00233     MyCalib->dig_P7 = (RxBuffer00[19] + (RxBuffer00[20]<<8));
00234     MyCalib->dig_P8 = (RxBuffer00[21] + (RxBuffer00[22]<<8));
00235     MyCalib->dig_P9 = (RxBuffer00[23] + (RxBuffer00[24]<<8));
00236
00237     MyCalib->dig_H1 = RxBuffer00[26];
00238     MyCalib->dig_H2 = RxBuffer26[1] + (RxBuffer26[2]<<8);
00239     MyCalib->dig_H3 = RxBuffer26[3];
00240     MyCalib->dig_H4 = (RxBuffer26[4]<<4) + (RxBuffer26[5] & 0x0F);
00241     MyCalib->dig_H5 = ((RxBuffer26[5]<<4) & 0x0F) + (RxBuffer26[6]<<4);
00242     MyCalib->dig_H6 = RxBuffer26[7];
00243 }
00244
00245 //*****
00246 //
00255 //
00256 //*****
00257 uint32_t BME280_Read_Pressure(SPI_HandleTypeDef spi) {
00258
00259     uint8_t RxBuffer[4];
00260     uint32_t Pressure;
00261     SPI_read_8bits(spi, PRESS_MSB, RxBuffer, 3, RNW_MSB);
00262

```

```

00263     Pressure = ((RxBuffer[1]«12) + (RxBuffer[2]«4) + (RxBuffer[3]«4));
00264
00265     return Pressure;
00266 }
00267
00268 //*****
00269 //
00270 //
00271 //*****
00272 uint16_t BME280_Read_Humidity(SPI_HandleTypeDef spi) {
00273
00274     uint8_t RxBuffer[3];
00275     uint16_t Humidity;
00276     SPI_read_8bits(spi, HUM_MSB, RxBuffer, 2, RNW_MSB);
00277
00278     Humidity = (RxBuffer[1]«8) + RxBuffer[2];
00279
00280     return Humidity;
00281 }
00282
00283
00284
00285
00286
00287
00288
00289
00290
00291 //*****
00292 //
00293 //
00294 //*****
00295 uint32_t BME280_Read_Temperature(SPI_HandleTypeDef spi) {
00296
00297     uint8_t RxBuffer[4];
00298     uint32_t Temperature;
00299     SPI_read_8bits(spi, TEMP_MSB, RxBuffer, 3, RNW_MSB);
00300
00301     Temperature = ((RxBuffer[1]«12) + (RxBuffer[2]«4) + (RxBuffer[3]«4));
00302
00303     return Temperature;
00304 }
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314 //*****
00315 //
00316 //| 2. Function compensate_temperature() has to be called before this function
00317 //
00318 //*****
00319 uint32_t compensate_humidity(uint16_t HumidityUncomp, struct bme280_calib_data *MyCalib) {
00320
00321     int32_t var1;
00322     int32_t var2;
00323     int32_t var3;
00324     int32_t var4;
00325     int32_t var5;
00326     uint32_t humidity;
00327     uint32_t humidity_max = 102400;
00328
00329     var1 = MyCalib->t_fine - ((int32_t)76800);
00330     var2 = (int32_t)(HumidityUncomp * 16384);
00331     var3 = (int32_t)((int32_t)MyCalib->dig_H4) * 1048576;
00332     var4 = ((int32_t)MyCalib->dig_H5) * var1;
00333     var5 = (((var2 - var3) - var4) + (int32_t)16384) / 32768;
00334     var2 = (var1 * ((int32_t)MyCalib->dig_H6)) / 1024;
00335     var3 = (var1 * ((int32_t)MyCalib->dig_H3)) / 2048;
00336     var4 = ((var2 * (var3 + (int32_t)32768)) / 1024) + (int32_t)2097152;
00337     var2 = ((var4 * ((int32_t)MyCalib->dig_H2)) + 8192) / 16384;
00338     var3 = var5 * var2;
00339     var4 = ((var3 / 32768) * (var3 / 32768)) / 128;
00340     var5 = var3 - ((var4 * ((int32_t)MyCalib->dig_H1)) / 16);
00341     var5 = (var5 < 0 ? 0 : var5);
00342     var5 = (var5 > 419430400 ? 419430400 : var5);
00343     humidity = (uint32_t)(var5 / 4096);
00344     if (humidity > humidity_max)
00345     {
00346         humidity = humidity_max;
00347     }
00348
00349     return humidity;
00350 }
00351
00352
00353
00354
00355
00356
00357
00358
00359 }
00360
00361 //*****
00362 //
00363 //
00364 //*****
00365 int32_t compensate_temperature(uint32_t TemperatureUncomp, struct bme280_calib_data *MyCalib) {
00366
00367     int32_t var1;
00368     int32_t var2;
00369     int32_t temperature;
00370     int32_t temperature_min = -4000;
00371     int32_t temperature_max = 8500;
00372
00373     var1 = (int32_t)((TemperatureUncomp / 8) - ((int32_t)MyCalib->dig_T1 * 2));
00374     var1 = (var1 * ((int32_t)MyCalib->dig_T2)) / 2048;
00375     var2 = (int32_t)((TemperatureUncomp / 16) - ((int32_t)MyCalib->dig_T1));
00376     var2 = (((var2 * var1) / 4096) * ((int32_t)MyCalib->dig_T3)) / 16384;
00377
00378
00379
00380
00381
00382
00383
00384
00385

```

```

00386     MyCalib->t_fine = var1 + var2;
00387     temperature = (MyCalib->t_fine * 5 + 128) / 256;
00388     if (temperature < temperature_min)
00389     {
00390         temperature = temperature_min;
00391     }
00392     else if (temperature > temperature_max)
00393     {
00394         temperature = temperature_max;
00395     }
00396
00397     return temperature;
00398 }
00399
00400 //*****
00401 //
00412 //
00413 //*****
00414 uint32_t compensate_pressure(uint32_t PressureUncomp, struct bme280_calib_data *MyCalib)
00415 {
00416     int32_t var1;
00417     int32_t var2;
00418     int32_t var3;
00419     int32_t var4;
00420     uint32_t var5;
00421     uint32_t pressure;
00422     uint32_t pressure_min = 30000;
00423     uint32_t pressure_max = 110000;
00424
00425     var1 = (((int32_t)MyCalib->t_fine) / 2) - (int32_t)64000;
00426     var2 = (((var1 / 4) * (var1 / 4)) / 2048) * ((int32_t)MyCalib->dig_P6);
00427     var2 = var2 + ((var1 * ((int32_t)MyCalib->dig_P5)) * 2);
00428     var2 = (var2 / 4) + (((int32_t)MyCalib->dig_P4) * 65536);
00429     var3 = (MyCalib->dig_P3 * (((var1 / 4) * (var1 / 4)) / 8192)) / 8;
00430     var4 = (((int32_t)MyCalib->dig_P2) * var1) / 2;
00431     var1 = (var3 + var4) / 262144;
00432     var1 = (((32768 + var1)) * ((int32_t)MyCalib->dig_P1)) / 32768;
00433
00434     /* avoid exception caused by division by zero */
00435     if (var1)
00436     {
00437         var5 = (uint32_t)((uint32_t)1048576) - PressureUncomp;
00438         pressure = ((uint32_t)(var5 - (uint32_t)(var2 / 4096))) * 3125;
00439         if (pressure < 0x80000000)
00440         {
00441             pressure = (pressure << 1) / ((uint32_t)var1);
00442         }
00443         else
00444         {
00445             pressure = (pressure / (uint32_t)var1) * 2;
00446         }
00447         var1 = (((int32_t)MyCalib->dig_P9) * ((int32_t)((pressure / 8) * (pressure / 8)) / 8192)) /
4096;
00448         var2 = (((int32_t)(pressure / 4)) * ((int32_t)MyCalib->dig_P8)) / 8192;
00449         pressure = (uint32_t)((int32_t)pressure + ((var1 + var2 + MyCalib->dig_P7) / 16));
00450         if (pressure < pressure_min)
00451         {
00452             pressure = pressure_min;
00453         }
00454         else if (pressure > pressure_max)
00455         {
00456             pressure = pressure_max;
00457         }
00458     }
00459     else
00460     {
00461         pressure = pressure_min;
00462     }
00463
00464     return pressure;
00465 }

```

3.7 BME280.h File Reference

Functions to configure and read data from BME280 Pressure, Temperature and Humidity sensor.

```
#include <stdbool.h>
#include <ti/drivers/SPI.h>
#include <ti/drivers/I2C.h>
```

Data Structures

- struct `bme280_calib_data`
BME280 calibration data struct. [More...](#)

Macros

- `#define BME280_DEVICE_AD 0x60`
- `#define BME280_RESET 0xB6`
- `#define CALIB00_NDATA 26`
- `#define CALIB26_NDATA 16`

Enumerations

- enum `BME280_filter_t`
`FILTER_OFF` = $(0x00 << 2) \& 0x1C$, `FILTER_2` = $(0x01 << 2) \& 0x1C$, `FILTER_4` = $(0x02 << 2) \& 0x1C$,
`FILTER_8` = $(0x03 << 2) \& 0x1C$,
`FILTER_16` = $(0x04 << 2) \& 0x1C$ }
BME280 filter configuration enum.
- enum `BME280_modes_t` { `SLEEP` = `0x00`, `FORCED` = `0x01`, `NORMAL` = `0x03` }
BME280 Modes enum.
- enum `BME280_osrs_h_t`{
`OSRS_HX1` = `0x01`, `OSRS_HX2` = `0x02`, `OSRS_HX4` = `0x03`, `OSRS_HX8` = `0x04`,
`OSRS_HX16` = `0x05` }
BME280 osrs humidity registers enum.
- enum `BME280_osrs_p_t`{
`OSRS_PX1` = $(0x01 << 2) \& 0x1C$, `OSRS_PX2` = $(0x02 << 2) \& 0x1C$, `OSRS_PX4` = $(0x03 << 2) \& 0x1C$,
`OSRS_PX8` = $(0x04 << 2) \& 0x1C$,
`OSRS_PX16` = $(0x05 << 2) \& 0x1C$ }
BME280 osrs pressure registers enum.
- enum `BME280_osrs_t_t`{
`OSRS_TX1` = $(0x01 << 5) \& 0xE0$, `OSRS_TX2` = $(0x02 << 5) \& 0xE0$, `OSRS_TX4` = $(0x03 << 5) \& 0xE0$,
`OSRS_TX8` = $(0x04 << 5) \& 0xE0$,
`OSRS_TX16` = $(0x05 << 5) \& 0xE0$ }
BME280 osrs temperature registers enum.
- enum `BME280_register_t`{
`HUM_LSB` = `0xFE`, `HUM_MSB` = `0xFD`, `TEMP_XLSB` = `0xFC`, `TEMP_LSB` = `0xFB`,
`TEMP_MSB` = `0xFA`, `PRESS_XLSB` = `0xF9`, `PRESS_LSB` = `0xF8`, `PRESS_MSB` = `0xF7`,
`CONFIG` = `0xF5`, `CTRL_MEAS` = `0xF4`, `BME280_STATUS` = `0xF3`, `CTRL_HUM` = `0xF2`,
`CALIB26` = `0xE1`, `BME280_RESET_REG` = `0xE0`, `DEVID` = `0xD0`, `CALIB00` = `0x88` }
BME280 Registers enum.
- enum `BME280_spi3w_t` { `SPI3W_ON` = `0x01`, `SPI3W_OFF` = `0x00` }
BME280 SPI configuration enum.
- enum `BME280_status_t` { `MEASURING` = `0x08`, `UPDATE` = `0x01` }
BME280 status enum.
- enum `BME280_t_sb_t`{
`T_SB_0p5` = $(0x00 << 5) \& 0xE0$, `T_SB_62p5` = $(0x01 << 5) \& 0xE0$, `T_SB_125` = $(0x02 << 5) \& 0xE0$,
`T_SB_250` = $(0x03 << 5) \& 0xE0$,
`T_SB_500` = $(0x04 << 5) \& 0xE0$, `T_SB_1000` = $(0x05 << 5) \& 0xE0$, `T_SB_10` = $(0x06 << 5) \& 0xE0$,
`T_SB_20` = $(0x07 << 5) \& 0xE0$ }
BME280 t sb resgisters enum.

Functions

- `uint8_t BME280_DevId (SPI_Handle spi)`
BME280 Get DevID Function.
- `void BME280_Read_Calib (SPI_Handle spi, struct bme280_calib_data *MyCalib)`
BME280 Read Calibration Function.
- `uint8_t BME280_Read_Config (SPI_Handle spi)`
BME280 Read Configuration Function.
- `uint8_t BME280_Read_Ctrl_Hum (SPI_Handle spi)`
BME280 Read Control Humidity Function.
- `uint8_t BME280_Read_Ctrl_Meas (SPI_Handle spi)`
BME280 Read Control Measurement Function.
- `uint16_t BME280_Read_Humidity (SPI_Handle spi)`
BME280 Read Humidity Function.
- `uint32_t BME280_Read_Pressure (SPI_Handle spi)`
BME280 Read Pressure Function.
- `uint8_t BME280_Read_Status (SPI_Handle spi)`
BME280 Read Status Function.
- `uint32_t BME280_Read_Temperature (SPI_Handle spi)`
BME280 Read Temperature Function.
- `void BME280_Reset (SPI_Handle spi)`
BME280 Reset Function.
- `void BME280_Write_Config (SPI_Handle spi, uint8_t Val)`
BME280 Write Configuration Function.
- `void BME280_Write_Ctrl_Hum (SPI_Handle spi, uint8_t Val)`
BME280 Write Control Humidity Function.
- `void BME280_Write_Ctrl_Meas (SPI_Handle spi, uint8_t Val)`
BME280 Write Control Measurement Function.
- `uint32_t compensate_humidity (uint16_t HumidityUncomp, struct bme280_calib_data *MyCalib)`
BME280 Compensate Humidity Function.
- `uint32_t compensate_pressure (uint32_t PressureUncomp, struct bme280_calib_data *MyCalib)`
BME280 Compensate pressure Function.
- `int32_t compensate_temperature (uint32_t TemperatureUncomp, struct bme280_calib_data *MyCalib)`
BME280 Compensate temperature Function.

3.7.1 Detailed Description

Functions to configure and read data from BME280 Pressure, Temperature and Humidity sensor.

Version

1.0

Date

07/05/2019

Author

A.Irizar @title ADXL355 accelerometer functions

Definition in file [BME280.h](#).

3.7.2 Data Structure Documentation

3.7.2.1 `struct bme280_calib_data` BME280 calibration data struct.

Definition at line 141 of file [BME280.h](#).

Data Fields

uint8_t	dig_H1	calibration data
int16_t	dig_H2	calibration data
uint8_t	dig_H3	calibration data
int16_t	dig_H4	calibration data
int16_t	dig_H5	calibration data
int8_t	dig_H6	calibration data
uint16_t	dig_P1	calibration data
int16_t	dig_P2	calibration data
int16_t	dig_P3	calibration data
int16_t	dig_P4	calibration data
int16_t	dig_P5	calibration data
int16_t	dig_P6	calibration data
int16_t	dig_P7	calibration data
int16_t	dig_P8	calibration data
int16_t	dig_P9	calibration data
uint16_t	dig_T1	calibration data
int16_t	dig_T2	calibration data
int16_t	dig_T3	calibration data
int32_t	t_fine	calibration data

3.7.3 Macro Definition Documentation**3.7.3.1 BME280_DEVICE_AD** `#define BME280_DEVICE_AD 0x60`

Definition at line [23](#) of file [BME280.h](#).

3.7.3.2 BME280_RESET `#define BME280_RESET 0xB6`

Definition at line [24](#) of file [BME280.h](#).

3.7.3.3 CALIB00_NDATA `#define CALIB00_NDATA 26`

Definition at line [26](#) of file [BME280.h](#).

3.7.3.4 CALIB26_NDATA `#define CALIB26_NDATA 16`

Definition at line [27](#) of file [BME280.h](#).

3.7.4 Enumeration Type Documentation

3.7.4.1 BME280_filter_t enum BME280_filter_t

BME280 filter configuration enum.

Enumerator

FILTER_OFF	FILTER OFF
FILTER_2	FILTER 2
FILTER_4	FILTER 4
FILTER_8	FILTER 8
FILTER_16	FILTER 16

Definition at line 122 of file [BME280.h](#).

```
00122     {
00123     FILTER_OFF = (0x00 << 2) & 0x1C,
00124     FILTER_2 = (0x01 << 2) & 0x1C,
00125     FILTER_4 = (0x02 << 2) & 0x1C,
00126     FILTER_8 = (0x03 << 2) & 0x1C,
00127     FILTER_16 = (0x04 << 2) & 0x1C
00128 } BME280_filter_t;
```

3.7.4.2 BME280_modes_t enum [BME280_modes_t](#)

BME280 Modes enum.

Enumerator

SLEEP	SLEEP mode
FORCED	FORCED mode
NORMAL	NORMAL mode

Definition at line 80 of file [BME280.h](#).

```
00080     {
00081     SLEEP = 0x00,
00082     FORCED = 0x01,
00083     NORMAL = 0x03
00084 } BME280_modes_t;
```

3.7.4.3 BME280_osrs_h_t enum [BME280_osrs_h_t](#)

BME280 osrs humidity registers enum.

Enumerator

OSRS_HX1	OSRS HX1 reg
OSRS_HX2	OSRS HX2 reg
OSRS_HX4	OSRS HX4 reg
OSRS_HX8	OSRS HX8 reg
OSRS_HX16	OSRS HX16 reg

Definition at line 89 of file [BME280.h](#).

```
00089     {
00090     OSRS_HX1 = 0x01,
00091     OSRS_HX2 = 0x02,
00092     OSRS_HX4 = 0x03,
00093     OSRS_HX8 = 0x04,
```

```
00094     OSRS_HX16 = 0x05
00095 } BME280_osrs_h_t;
```

3.7.4.4 BME280_osrs_p_t enum BME280_osrs_p_t

BME280 osrs pressure registers enum.

Enumerator

OSRS_PX1	OSRS PX1 reg
OSRS_PX2	OSRS PX2 reg
OSRS_PX4	OSRS PX4 reg
OSRS_PX8	OSRS PX8 reg
OSRS_PX16	OSRS PX16 reg

Definition at line 69 of file [BME280.h](#).

```
00069     {
00070     OSRS_PX1 = (0x01 << 2) & 0x1C,
00071     OSRS_PX2 = (0x02 << 2) & 0x1C,
00072     OSRS_PX4 = (0x03 << 2) & 0x1C,
00073     OSRS_PX8 = (0x04 << 2) & 0x1C,
00074     OSRS_PX16 = (0x05 << 2) & 0x1C
00075 } BME280_osrs_p_t;
```

3.7.4.5 BME280_osrs_t_t enum BME280_osrs_t_t

BME280 osrs temperature registers enum.

Enumerator

OSRS_TX1	OSRS TX1 reg
OSRS_TX2	OSRS TX2 reg
OSRS_TX4	OSRS TX4 reg
OSRS_TX8	OSRS TX8 reg
OSRS_TX16	OSRS TX16 reg

Definition at line 58 of file [BME280.h](#).

```
00058     {
00059     OSRS_TX1 = (0x01 << 5) & 0xE0,
00060     OSRS_TX2 = (0x02 << 5) & 0xE0,
00061     OSRS_TX4 = (0x03 << 5) & 0xE0,
00062     OSRS_TX8 = (0x04 << 5) & 0xE0,
00063     OSRS_TX16 = (0x05 << 5) & 0xE0
00064 } BME280_osrs_t_t;
```

3.7.4.6 BME280_register_t enum BME280_register_t

BME280 Registers enum.

Enumerator

HUM_LSB	HUM LSB reg
HUM_MSB	HUM MSB reg
TEMP_XLSB	TEMP XLSB reg
TEMP_LSB	TEMP LSB reg
TEMP_MSB	TEMP MSB reg
PRESS_XLSB	PRESS XLSB reg
PRESS_LSB	PRESS LSB reg
PRESS_MSB	PRESS MSB reg
CONFIG	CONFIG reg
CTRL_MEAS	CTRL MEAS reg
BME280_STATUS	BME280 STATUS reg
CTRL_HUM	CTRL HUM reg
CALIB26	CALIB26 reg
BME280_RESET_REG	BME280 RESET REG reg
DEVID	DEVID reg
CALIB00	CALIB00 reg

Definition at line 36 of file [BME280.h](#).

```

00036     {
00037     HUM_LSB = 0xFE,
00038     HUM_MSB = 0xFD,
00039     TEMP_XLSB = 0xFC,
00040     TEMP_LSB = 0xFB,
00041     TEMP_MSB = 0xFA,
00042     PRESS_XLSB = 0xF9,
00043     PRESS_LSB = 0xF8,
00044     PRESS_MSB = 0xF7,
00045     CONFIG = 0xF5,
00046     CTRL_MEAS = 0xF4,
00047     BME280_STATUS = 0xF3,
00048     CTRL_HUM = 0xF2,
00049     CALIB26 = 0xE1,
00050     BME280_RESET_REG = 0xE0,
00051     DEVID = 0xD0,
00052     CALIB00 = 0x88
00053 } BME280_register_t;

```

3.7.4.7 BME280_spi3w_t enum BME280_spi3w_t

BME280 SPI configuration enum.

Enumerator

SPI3W_ON	SPI on
SPI3W_OFF	SPI off

Definition at line 133 of file [BME280.h](#).

```

00133     {
00134     SPI3W_ON = 0x01,
00135     SPI3W_OFF = 0x00
00136 } BME280_spi3w_t;

```

3.7.4.8 BME280_status_t enum BME280_status_t

BME280 status enum.

Enumerator

MEASURING	MEASURING
UPDATE	UPDATE

Definition at line 100 of file [BME280.h](#).

```
00100      {
00101      MEASURING = 0x08,
00102      UPDATE = 0x01
00103 } BME280_status_t;
```

3.7.4.9 BME280_t_sb_t enum BME280_t_sb_t

BME280 t sb registers enum.

Enumerator

T_SB_0p5	T SB 0p5
T_SB_62p5	T SB 62p5
T_SB_125	T SB 125
T_SB_250	T SB 250
T_SB_500	T SB 500
T_SB_1000	T SB 1000
T_SB_10	T SB 10
T_SB_20	T SB 20

Definition at line 108 of file [BME280.h](#).

```
00108      {
00109      T_SB_0p5 = (0x00 << 5) & 0xE0,
00110      T_SB_62p5 = (0x01 << 5) & 0xE0,
00111      T_SB_125 = (0x02 << 5) & 0xE0,
00112      T_SB_250 = (0x03 << 5) & 0xE0,
00113      T_SB_500 = (0x04 << 5) & 0xE0,
00114      T_SB_1000 = (0x05 << 5) & 0xE0,
00115      T_SB_10 = (0x06 << 5) & 0xE0,
00116      T_SB_20 = (0x07 << 5) & 0xE0
00117 } BME280_t_sb_t;
```

3.7.5 Function Documentation

3.7.5.1 BME280_DevId() uint8_t BME280_DevId (SPI_HandleTypeDef spi)

BME280 Get DevID Function.

This function

1. Gets DevID from BME280 sensor

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

uint8_t RxBuffer DEVID

Definition at line 56 of file [BME280.c](#).

```
00056                                     {
00057
00058     uint8_t RxBuffer[2];
00059
00060     SPI_read_8bits(spi, DEVID, RxBuffer, 1, RNW_MSB);
00061     return RxBuffer[1];
00062 }
```

3.7.5.2 BME280_Read_Calib() void BME280_Read_Calib (

```
    SPI_Handle spi,
    struct bme280_calib_data * MyCalib )
```

BME280 Read Calibration Function.

This function

1. Reads calibration register on BME280 sensor

Parameters

in	<i>SPI_Handle</i>	spi
	<i>struct bme280_calib_data</i> *	MyCalib pointer to BME280 calibration data

Returns

None

Definition at line 214 of file [BME280.c](#).

```
00214                                     {
00215
00216     uint8_t RxBuffer00[CALIB00_NDATA+1];
00217     uint8_t RxBuffer26[CALIB26_NDATA+1];
00218
00219     SPI_read_8bits(spi, CALIB00, RxBuffer00, CALIB00_NDATA, RNW_MSB);
00220     SPI_read_8bits(spi, CALIB26, RxBuffer26, CALIB26_NDATA, RNW_MSB);
00221
00222     MyCalib->dig_T1 = (RxBuffer00[1] + (RxBuffer00[2]<<8));
00223     MyCalib->dig_T2 = (RxBuffer00[3] + (RxBuffer00[4]<<8));
00224     MyCalib->dig_T3 = (RxBuffer00[5] + (RxBuffer00[6]<<8));
00225
00226     MyCalib->dig_P1 = (RxBuffer00[7] + (RxBuffer00[8]<<8));
00227     MyCalib->dig_P2 = (RxBuffer00[9] + (RxBuffer00[10]<<8));
00228     MyCalib->dig_P3 = (RxBuffer00[11] + (RxBuffer00[12]<<8));
00229     MyCalib->dig_P4 = (RxBuffer00[13] + (RxBuffer00[14]<<8));
00230     MyCalib->dig_P5 = (RxBuffer00[15] + (RxBuffer00[16]<<8));
00231     MyCalib->dig_P6 = (RxBuffer00[17] + (RxBuffer00[18]<<8));
00232     MyCalib->dig_P7 = (RxBuffer00[19] + (RxBuffer00[20]<<8));
00233     MyCalib->dig_P8 = (RxBuffer00[21] + (RxBuffer00[22]<<8));
```

```
00235     MyCalib->dig_P9 = (RxBuffer00[23] + (RxBuffer00[24]«8));
00236
00237     MyCalib->dig_H1 = RxBuffer00[26];
00238     MyCalib->dig_H2 = RxBuffer26[1] + (RxBuffer26[2]«8);
00239     MyCalib->dig_H3 = RxBuffer26[3];
00240     MyCalib->dig_H4 = (RxBuffer26[4]«4) + (RxBuffer26[5] & 0x0F);
00241     MyCalib->dig_H5 = ((RxBuffer26[5]»4) & 0x0F) + (RxBuffer26[6]«4);
00242     MyCalib->dig_H6 = RxBuffer26[7];
00243 }
```

3.7.5.3 BME280_Read_Config() `uint8_t BME280_Read_Config (SPI_Handle spi)`

BME280 Read Configuration Function.

This function

1. Reads Configuration register on BME280 sensor

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

`uint8_t RxBuffer`

Definition at line 171 of file [BME280.c](#).

```
00171
00172
00173     uint8_t RxBuffer[2];
00174
00175     SPI_read_8bits(spi, CONFIG, RxBuffer, 1, RNW_MSB);
00176
00177     return RxBuffer[1];
00178 }
```

3.7.5.4 BME280_Read_Ctrl_Hum() `uint8_t BME280_Read_Ctrl_Hum (SPI_Handle spi)`

BME280 Read Control Humidity Function.

This function

1. Reads control humidity register on BME280 sensor

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

```
uint8_t RxBuffer Humidity value
```

Definition at line 94 of file [BME280.c](#).

```
00094
00095
00096     uint8_t RxBuffer[2];
00097
00098     SPI_read_8bits(spi, CTRL_HUM, RxBuffer, 1, RNW_MSB);
00099     return RxBuffer[1];
00100 }
```

3.7.5.5 BME280_Read_Ctrl_Meas() `uint8_t BME280_Read_Ctrl_Meas (`
`SPI_HandleTypeDef spi)`

BME280 Read Control Measurement Function.

This function

1. Reads control measurement register on BME280 sensor

Parameters

in	<i>SPI_HandleTypeDef</i>	spi
----	--------------------------	-----

Returns

```
uint8_t RxBuffer
```

Definition at line 132 of file [BME280.c](#).

```
00132
00133
00134     uint8_t RxBuffer[2];
00135
00136     SPI_read_8bits(spi, CTRL_MEAS, RxBuffer, 1, RNW_MSB);
00137
00138     return RxBuffer[1];
00139 }
```

3.7.5.6 BME280_Read_Humidity() `uint16_t BME280_Read_Humidity (`
`SPI_HandleTypeDef spi)`

BME280 Read Humidity Function.

This function

1. Reads Humidity register on BME280 sensor

Parameters

in	<i>SPI_HandleTypeDef</i>	spi
----	--------------------------	-----

Returns

uint16_t Humidity

Definition at line 280 of file BME280.c.

```
00280          {  
00281  
00282     uint8_t RxBuffer[3];  
00283     uint16_t Humidity;  
00284     SPI_read_8bits(spi, HUM_MSB, RxBuffer, 2, RNW_MSB);  
00285     Humidity = (RxBuffer[1]«8) + RxBuffer[2];  
00286     return Humidity;  
00287 }  
00289 }
```

3.7.5.7 BME280_Read_Pressure() `uint32_t BME280_Read_Pressure (`
`SPI_Handle spi)`

BME280 Read Pressure Function.

This function

1. Reads pressure register on BME280 sensor

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

uint32_t Pressure

Definition at line 257 of file BME280.c.

```
00257          {  
00258  
00259     uint8_t RxBuffer[4];  
00260     uint32_t Pressure;  
00261     SPI_read_8bits(spi, PRESS_MSB, RxBuffer, 3, RNW_MSB);  
00262     Pressure = ((RxBuffer[1]«12) + (RxBuffer[2]«4) + (RxBuffer[3]«4));  
00263     return Pressure;  
00264 }  
00266 }
```

3.7.5.8 BME280_Read_Status() `uint8_t BME280_Read_Status (`
`SPI_Handle spi)`

BME280 Read Status Function.

This function

1. Reads Status register on BME280 sensor

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

uint8_t RxBuffer

Definition at line 192 of file [BME280.c](#).

```

00192
00193
00194     uint8_t RxBuffer[2];
00195
00196     SPI_read_8bits(spi, BME280_STATUS, RxBuffer, 1, RNW_MSB);
00197
00198     return RxBuffer[1];
00199 }
```

3.7.5.9 BME280_Read_Temperature()

uint32_t BME280_Read_Temperature (

SPI_Handle *spi*)

BME280 Read Temperature Function.

This function

1. Reads Temperature register on BME280 sensor

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

uint32_t Temperature

Definition at line 303 of file [BME280.c](#).

```

00303
00304
00305     uint8_t RxBuffer[4];
00306     uint32_t Temperature;
00307     SPI_read_8bits(spi, TEMP_MSB, RxBuffer, 3, RNW_MSB);
00308
00309     Temperature = ((RxBuffer[1]«12) + (RxBuffer[2]«4) + (RxBuffer[3]«4));
00310
00311     return Temperature;
00312 }
```

3.7.5.10 BME280_Reset()

void BME280_Reset (

SPI_Handle *spi*)

BME280 Reset Function.

This function

1. Resets BME280 sensor

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

none

Definition at line 39 of file [BME280.c](#).

```
00039                                     {  
00040  
00041     SPI_write_8bits(spi, BME280_RESET_REG, BME280_RESET, RNW_MSB);  
00042 }
```

3.7.5.11 BME280_Write_Config() void BME280_Write_Config (
 SPI_Handle *spi*,
 uint8_t *Val*)

BME280 Write Configuration Function.

This function

1. Writes Configuration register on BME280 sensor

Parameters

in	<i>SPI_Handle</i>	spi
in	<i>uint8_t</i>	Val

Returns

none

Definition at line 154 of file [BME280.c](#).

```
00154                                     {  
00155  
00156     SPI_write_8bits(spi, CONFIG, Val, RNW_MSB);  
00157 }
```

3.7.5.12 BME280_Write_Ctrl_Hum() void BME280_Write_Ctrl_Hum (
 SPI_Handle *spi*,
 uint8_t *Val*)

BME280 Write Control Humidity Function.

This function

1. Writes control humidity register on BME280 sensor

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

none

Definition at line 76 of file [BME280.c](#).

```
00076
00077
00078
00079     SPI_write_8bits(spi, CTRL_HUM, Val, RNW_MSB);
00080 }
```

3.7.5.13 BME280_Write_Ctrl_Meas() void BME280_Write_Ctrl_Meas (
 SPI_Handle *spi*,
 uint8_t *Val*)

BME280 Write Control Measurement Function.

This function

1. Writes control measurement register on BME280 sensor

Parameters

in	<i>SPI_Handle</i>	spi
in	<i>uint8_t</i>	Val

Returns

none

Definition at line 115 of file [BME280.c](#).

```
00115
00116
00117     SPI_write_8bits(spi, CTRL_MEAS, Val, RNW_MSB);
00118 }
```

3.7.5.14 compensate_humidity() *uint32_t* compensate_humidity (
 uint16_t *Humi di tyUncomp*,
 struct bme280_calib_data * *MyCalib*)

BME280 Compensate Humidity Function.

This function

1. Compensates the value of the humidity read from the BME280 registers

Precondition*compensate_humidity()* has been called

Parameters

in	<code>uint16_t</code>	HumidityUncomp Uncompensated value of Humidity as read from the BME280 registers
in	<code>struct</code>	<code>bme280_calib_data</code> Pointer to an struct that holds the calibration values of the sensor

Returns

`uint32_t humidity` Returns the compensated humidity value in units $1024 * \text{relative humidity}$

Definition at line 329 of file [BME280.c](#).

```

00329
00330     int32_t var1;
00331     int32_t var2;
00332     int32_t var3;
00333     int32_t var4;
00334     int32_t var5;
00335     uint32_t humidity;
00336     uint32_t humidity_max = 102400;
00337
00338     var1 = MyCalib->t_fine - ((int32_t)76800);
00339     var2 = (int32_t)(HumidityUncomp * 16384);
00340     var3 = (int32_t)((int32_t)MyCalib->dig_H4) * 1048576);
00341     var4 = ((int32_t)MyCalib->dig_H5) * var1;
00342     var5 = (((var2 - var3) - var4) + (int32_t)16384) / 32768;
00343     var2 = (var1 * ((int32_t)MyCalib->dig_H6)) / 1024;
00344     var3 = (var1 * ((int32_t)MyCalib->dig_H3)) / 2048;
00345     var4 = ((var2 * (var3 + (int32_t)32768)) / 1024) + (int32_t)2097152;
00346     var2 = ((var4 * ((int32_t)MyCalib->dig_H2)) + 8192) / 16384;
00347     var3 = var5 * var2;
00348     var4 = ((var3 / 32768) * (var3 / 32768)) / 128;
00349     var5 = var3 - ((var4 * ((int32_t)MyCalib->dig_H1)) / 16);
00350     var5 = (var5 < 0 ? 0 : var5);
00351     var5 = (var5 > 419430400 ? 419430400 : var5);
00352     humidity = (uint32_t)(var5 / 4096);
00353     if (humidity > humidity_max)
00354     {
00355         humidity = humidity_max;
00356     }
00357
00358     return humidity;
00359 }
```

3.7.5.15 `compensate_pressure()` `uint32_t compensate_pressure (`
 `uint32_t PressureUncomp,`
 `struct bme280_calib_data * MyCalib)`

BME280 Compensate pressure Function.

This function

1. Compensates the value of the pressure read from the BME280 registers

Precondition

`compensate_pressure()` has been called

Parameters

in	<code>uint32_t</code>	PressureUncomp Uncompensated value of Pressure as read from the BME280 registers
in	<code>struct</code>	<code>bme280_calib_data</code> Pointer to an struct that holds the calibration values of the sensor

Returns

`uint32_t pressure` Returns the compensated pressure value in units of Pascals

Definition at line 414 of file [BME280.c](#).

```

00415 {
00416     int32_t var1;
00417     int32_t var2;
00418     int32_t var3;
00419     int32_t var4;
00420     uint32_t var5;
00421     uint32_t pressure;
00422     uint32_t pressure_min = 30000;
00423     uint32_t pressure_max = 110000;
00424
00425     var1 = (((int32_t)MyCalib->t_fine) / 2) - (int32_t)64000;
00426     var2 = (((var1 / 4) * (var1 / 4)) / 2048) * ((int32_t)MyCalib->dig_P6);
00427     var2 = var2 + ((var1 * ((int32_t)MyCalib->dig_P5)) * 2);
00428     var2 = (var2 / 4) + (((int32_t)MyCalib->dig_P4) * 65536);
00429     var3 = (MyCalib->dig_P3 * (((var1 / 4) * (var1 / 4)) / 8192)) / 8;
00430     var4 = (((int32_t)MyCalib->dig_P2) * var1) / 2;
00431     var1 = (var3 + var4) / 262144;
00432     var1 = (((32768 + var1)) * ((int32_t)MyCalib->dig_P1)) / 32768;
00433
00434 /* avoid exception caused by division by zero */
00435 if (var1)
00436 {
00437     var5 = (uint32_t)((uint32_t)1048576) - PressureUncomp;
00438     pressure = ((uint32_t)(var5 - (uint32_t)(var2 / 4096))) * 3125;
00439     if (pressure < 0x80000000)
00440     {
00441         pressure = (pressure << 1) / ((uint32_t)var1);
00442     }
00443     else
00444     {
00445         pressure = (pressure / (uint32_t)var1) * 2;
00446     }
00447     var1 = (((int32_t)MyCalib->dig_P9) * ((int32_t)((pressure / 8) * (pressure / 8)) / 8192)) /
4096;
00448     var2 = (((int32_t)(pressure / 4)) * ((int32_t)MyCalib->dig_P8)) / 8192;
00449     pressure = (uint32_t)((int32_t)pressure + ((var1 + var2 + MyCalib->dig_P7) / 16));
00450     if (pressure < pressure_min)
00451     {
00452         pressure = pressure_min;
00453     }
00454     else if (pressure > pressure_max)
00455     {
00456         pressure = pressure_max;
00457     }
00458 }
00459 else
00460 {
00461     pressure = pressure_min;
00462 }
00463
00464 return pressure;
00465 }
```

3.7.5.16 `compensate_temperature()` `int32_t compensate_temperature (`
 `uint32_t TemperatureUncomp,`
 `struct bme280_calib_data * MyCalib)`

BME280 Compensate temperature Function.

This function

1. Compensates the value of the temperature read from the BME280 registers

Precondition

`compensate_temperature()` has been called

Parameters

in	<i>uint32_t</i>	Uncompensated value of Temperature as read from the BME280 registers
in	<i>struct</i>	bme280_calib_data Pointer to an struct that holds the calibration values of the sensor

Returns

uint32_t temperature Returns the compensated temperature value in units 100 * C

Definition at line 375 of file [BME280.c](#).

```

00375
00376     int32_t var1;
00377     int32_t var2;
00378     int32_t temperature;
00379     int32_t temperature_min = -4000;
00380     int32_t temperature_max = 8500;
00381
00382     var1 = (int32_t)((TemperatureUncomp / 8) - ((int32_t)MyCalib->dig_T1 * 2));
00383     var1 = (var1 * ((int32_t)MyCalib->dig_T2) / 2048;
00384     var2 = (int32_t)((TemperatureUncomp / 16) - ((int32_t)MyCalib->dig_T1));
00385     var2 = (((var2 * var2) / 4096) * ((int32_t)MyCalib->dig_T3) / 16384;
00386     MyCalib->t_fine = var1 + var2;
00387     temperature = (MyCalib->t_fine * 5 + 128) / 256;
00388     if (temperature < temperature_min)
00389     {
00390         temperature = temperature_min;
00391     }
00392     else if (temperature > temperature_max)
00393     {
00394         temperature = temperature_max;
00395     }
00396
00397     return temperature;
00398 }
```

3.8 BME280.h

```

00001
00010 #ifndef BME280_H_
00011 #define BME280_H_
00012
00013 //*****
00014 //           INCLUDES
00015 //*****
00016 #include <stdbool.h>
00017 #include <ti/drivers/SPI.h>
00018 #include <ti/drivers/I2C.h>
00019
00020 //*****
00021 //           DEFINES
00022 //*****
00023 #define BME280_DEVICE_AD      0x60
00024 #define BME280_RESET          0xB6
00025
00026 #define CALIB00_NDATA          26
00027 #define CALIB26_NDATA          16
00028
00029 //*****
00030 //           TYPEDEFS
00031 //*****
00032
00033 typedef enum {
00034     HUM_LSB = 0xFE,
00035     HUM_MSB = 0xFD,
00036     TEMP_XLSB = 0xFC,
00037     TEMP_LSB = 0xFB,
00038     TEMP_MSB = 0xFA,
00039     PRESS_XLSB = 0xF9,
00040     PRESS_LSB = 0xF8,
00041     PRESS_MSB = 0xF7,
00042     CONFIG = 0xF5,
00043     CTRL_MEAS = 0xF4,
00044     BME280_STATUS = 0xF3,
00045     CTRL_HUM = 0xF2,
00046     CALIB26 = 0xE1,
```

```
00050     BME280_RESET_REG = 0xE0,
00051     DEVID = 0xD0,
00052     CALIBOO = 0x88
00053 } BME280_register_t;
00054
00058 typedef enum {
00059     OSRS_TX1 = (0x01 « 5) & 0xE0,
00060     OSRS_TX2 = (0x02 « 5) & 0xE0,
00061     OSRS_TX4 = (0x03 « 5) & 0xE0,
00062     OSRS_TX8 = (0x04 « 5) & 0xE0,
00063     OSRS_TX16 = (0x05 « 5) & 0xE0
00064 } BME280_osrs_t_t;
00065
00069 typedef enum {
00070     OSRS_PX1 = (0x01 « 2) & 0x1C,
00071     OSRS_PX2 = (0x02 « 2) & 0x1C,
00072     OSRS_PX4 = (0x03 « 2) & 0x1C,
00073     OSRS_PX8 = (0x04 « 2) & 0x1C,
00074     OSRS_PX16 = (0x05 « 2) & 0x1C
00075 } BME280_osrs_p_t;
00076
00080 typedef enum {
00081     SLEEP = 0x00,
00082     FORCED = 0x01,
00083     NORMAL = 0x03
00084 } BME280_modes_t;
00085
00089 typedef enum {
00090     OSRS_HX1 = 0x01,
00091     OSRS_HX2 = 0x02,
00092     OSRS_HX4 = 0x03,
00093     OSRS_HX8 = 0x04,
00094     OSRS_HX16 = 0x05
00095 } BME280_osrs_h_t;
00096
00100 typedef enum {
00101     MEASURING = 0x08,
00102     UPDATE = 0x01
00103 } BME280_status_t;
00104
00108 typedef enum {
00109     T_SB_0p5 = (0x00 « 5) & 0xE0,
00110     T_SB_62p5 = (0x01 « 5) & 0xE0,
00111     T_SB_125 = (0x02 « 5) & 0xE0,
00112     T_SB_250 = (0x03 « 5) & 0xE0,
00113     T_SB_500 = (0x04 « 5) & 0xE0,
00114     T_SB_1000 = (0x05 « 5) & 0xE0,
00115     T_SB_10 = (0x06 « 5) & 0xE0,
00116     T_SB_20 = (0x07 « 5) & 0xE0
00117 } BME280_t_sb_t;
00118
00122 typedef enum {
00123     FILTER_OFF = (0x00 « 2) & 0x1C,
00124     FILTER_2 = (0x01 « 2) & 0x1C,
00125     FILTER_4 = (0x02 « 2) & 0x1C,
00126     FILTER_8 = (0x03 « 2) & 0x1C,
00127     FILTER_16 = (0x04 « 2) & 0x1C
00128 } BME280_filter_t;
00129
00133 typedef enum {
00134     SPI3W_ON = 0x01,
00135     SPI3W_OFF = 0x00
00136 } BME280_spi3w_t;
00137
00141 struct bme280_calib_data
00142 {
00143     uint16_t dig_T1;
00144     int16_t dig_T2;
00145     int16_t dig_T3;
00146     uint16_t dig_P1;
00147     int16_t dig_P2;
00148     int16_t dig_P3;
00149     int16_t dig_P4;
00150     int16_t dig_P5;
00151     int16_t dig_P6;
00152     int16_t dig_P7;
00153     int16_t dig_P8;
00154     int16_t dig_P9;
00155     uint8_t dig_H1;
00156     int16_t dig_H2;
00157     uint8_t dig_H3;
00158     int16_t dig_H4;
00159     int16_t dig_H5;
00160     int8_t dig_H6;
00161     int32_t t_fine;
00162 };
00163
```

```

00164 //*****
00165 //          FUNCTION PROTOTYPES
00166 //*****
00167 void BME280_Reset(SPI_HandleTypeDef spi);
00168 uint8_t BME280_DevId(SPI_HandleTypeDef spi);
00169 void BME280_Write_Ctrl_Hum(SPI_HandleTypeDef spi, uint8_t Val);
00170 uint8_t BME280_Read_Ctrl_Hum(SPI_HandleTypeDef spi);
00171 void BME280_Write_Ctrl_Meas(SPI_HandleTypeDef spi, uint8_t Val);
00172 uint8_t BME280_Read_Ctrl_Meas(SPI_HandleTypeDef spi);
00173 void BME280_Write_Config(SPI_HandleTypeDef spi, uint8_t Val);
00174 uint8_t BME280_Read_Config(SPI_HandleTypeDef spi);
00175 uint8_t BME280_Read_Status(SPI_HandleTypeDef spi);
00176 void BME280_Read_Calib(SPI_HandleTypeDef spi, struct bme280_calib_data *MyCalib);
00177 uint32_t BME280_Read_Pressure(SPI_HandleTypeDef spi);
00178 uint16_t BME280_Read_Humidity(SPI_HandleTypeDef spi);
00179 uint32_t BME280_Read_Temperature(SPI_HandleTypeDef spi);
00180 uint32_t compensate_humidity(uint16_t HumidityUncomp, struct bme280_calib_data *MyCalib);
00181 int32_t compensate_temperature(uint32_t TemperatureUncomp, struct bme280_calib_data *MyCalib);
00182 uint32_t compensate_pressure(uint32_t PressureUncomp, struct bme280_calib_data *MyCalib);
00183
00184 #endif /* BME280_H */

```

3.9 Board.h File Reference

```
#include "CC3220SF_STARPORTS.h"
```

Macros

- #define Board_ADC0 CC3220SF_STARPORTS_ADC0
- #define Board_AXDL355_CS CC3220SF_STARPORTS_AXDL355_CS
- #define Board_BME280_CS CC3220SF_STARPORTS_BME280_CS
- #define Board_CAPTURE0 CC3220SF_STARPORTS_CAPTURE0
- #define Board_CAPTURE1 CC3220SF_STARPORTS_CAPTURE1
- #define Board_CC3220SF_STARPORTS
- #define Board_CRYPTO0 CC3220SF_STARPORTS_CRYPTO0
- #define Board_CS_ASGPIO CC3220SF_STARPORTS_CS
- #define Board_DS1374_INTB CC3220SF_STARPORTS_DS1374_INTB
- #define Board_EN_AXDL355 CC3220SF_STARPORTS_EN_AXDL355
- #define Board_EN_BME280 CC3220SF_STARPORTS_EN_BME280
- #define Board_EN_LDC1000 CC3220SF_STARPORTS_EN_LDC1000
- #define Board_EN_NODE CC3220SF_STARPORTS_EN_NODE
- #define Board_GPIO_LED0 CC3220SF_STARPORTS_GPIO_LED_D10
- #define Board_GPIO_LED_OFF CC3220SF_STARPORTS_GPIO_LED_OFF
- #define Board_GPIO_LED_ON CC3220SF_STARPORTS_GPIO_LED_ON
- #define Board_I2C0 CC3220SF_STARPORTS_I2C0
- #define Board_I2C_TMP CC3220SF_STARPORTS_I2C0
- #define Board_I2C_TMP006_ADDR Board_TMP006_ADDR
- #define Board_I2S0 CC3220SF_STARPORTS_I2S0
- #define Board_init CC3220SF_STARPORTS_initGeneral
- #define Board_initGeneral CC3220SF_STARPORTS_initGeneral
- #define Board_LDC1000_CS CC3220SF_STARPORTS_LDC1000_CS
- #define Board_MOSI_ASGPIO CC3220SF_STARPORTS_MOSI
- #define Board_PWM0 CC3220SF_STARPORTS_PWM5
- #define Board_PWM1 CC3220SF_STARPORTS_PWM7
- #define Board_RN2483_MCLR CC3220SF_STARPORTS_RN2483_MCLR
- #define Board_SCL_ASGPIO CC3220SF_STARPORTS_SCL
- #define Board_SCLK_ASGPIO CC3220SF_STARPORTS_SCLK
- #define Board_SD0 CC3220SF_STARPORTS_SD0
- #define Board_SDA_ASGPIO CC3220SF_STARPORTS_SDA

- #define Board_SDFatFS0 CC3220SF_STARPORTS_SD0
- #define Board_SPI0 CC3220SF_STARPORTS_SPI1
- #define Board_SPI_MASTER CC3220SF_STARPORTS_SPI1
- #define Board_SPI_MASTER_READY CC3220SF_STARPORTS_SPI_MASTER_READY
- #define Board_SPI_SLAVE CC3220SF_STARPORTS_SPI1
- #define Board_SPI_SLAVE_READY CC3220SF_STARPORTS_SPI_SLAVE_READY
- #define Board_TIMER0 CC3220SF_STARPORTS_TIMER0
- #define Board_TIMER1 CC3220SF_STARPORTS_TIMER1
- #define Board_TIMER2 CC3220SF_STARPORTS_TIMER2
- #define Board_TMP006_ADDR (0x41)
- #define Board_UART0 CC3220SF_STARPORTS_UART0
- #define Board_UART1 CC3220SF_STARPORTS_UART1
- #define Board_WATCHDOG0 CC3220SF_STARPORTS_WATCHDOG0

3.9.1 Macro Definition Documentation

3.9.1.1 Board_ADC0 #define Board_ADC0 CC3220SF_STARPORTS_ADC0

Definition at line 47 of file [Board.h](#).

3.9.1.2 Board_ADXL355_CS #define Board_ADXL355_CS CC3220SF_STARPORTS_ADXL355_CS

Definition at line 63 of file [Board.h](#).

3.9.1.3 Board_BME280_CS #define Board_BME280_CS CC3220SF_STARPORTS_BME280_CS

Definition at line 64 of file [Board.h](#).

3.9.1.4 Board_CAPTURE0 #define Board_CAPTURE0 CC3220SF_STARPORTS_CAPTURE0

Definition at line 49 of file [Board.h](#).

3.9.1.5 Board_CAPTURE1 #define Board_CAPTURE1 CC3220SF_STARPORTS_CAPTURE1

Definition at line 50 of file [Board.h](#).

3.9.1.6 Board_CC3220SF_STARPORTS #define Board_CC3220SF_STARPORTS

Definition at line 36 of file [Board.h](#).

3.9.1.7 Board_CRYPTO0 #define Board_CRYPTO0 CC3220SF_STARPORTS_CRYPTO0

Definition at line 52 of file [Board.h](#).

3.9.1.8 Board_CS_ASGPIO #define Board_CS_ASGPIO CC3220SF_STARPORTS_CS

Definition at line 75 of file [Board.h](#).

3.9.1.9 Board_DS1374_INTB #define Board_DS1374_INTB CC3220SF_STARPORTS_DS1374_INTB

Definition at line 70 of file [Board.h](#).

3.9.1.10 Board_EN_ADXL355 #define Board_EN_ADXL355 CC3220SF_STARPORTS_EN_ADXL355

Definition at line 67 of file [Board.h](#).

3.9.1.11 Board_EN_BME280 #define Board_EN_BME280 CC3220SF_STARPORTS_EN_BME280

Definition at line 68 of file [Board.h](#).

3.9.1.12 Board_EN_LDC1000 #define Board_EN_LDC1000 CC3220SF_STARPORTS_EN_LDC1000

Definition at line 69 of file [Board.h](#).

3.9.1.13 Board_EN_NODE #define Board_EN_NODE CC3220SF_STARPORTS_EN_NODE

Definition at line 66 of file [Board.h](#).

3.9.1.14 Board_GPIO_LED0 #define Board_GPIO_LED0 CC3220SF_STARPORTS_GPIO_LED_D10

Definition at line 56 of file [Board.h](#).

3.9.1.15 Board_GPIO_LED_OFF #define Board_GPIO_LED_OFF CC3220SF_STARPORTS_GPIO_LED_OFF

Definition at line 55 of file [Board.h](#).

3.9.1.16 Board_GPIO_LED_ON #define Board_GPIO_LED_ON CC3220SF_STARPORTS_GPIO_LED_ON

Definition at line 54 of file [Board.h](#).

3.9.1.17 Board_I2C0 #define Board_I2C0 CC3220SF_STARPORTS_I2C0

Definition at line 81 of file [Board.h](#).

3.9.1.18 Board_I2C_TMP #define Board_I2C_TMP CC3220SF_STARPORTS_I2C0

Definition at line 82 of file [Board.h](#).

3.9.1.19 Board_I2C_TMP006_ADDR #define Board_I2C_TMP006_ADDR Board_TMP006_ADDR

Definition at line 111 of file [Board.h](#).

3.9.1.20 Board_I2S0 #define Board_I2S0 CC3220SF_STARPORTS_I2S0

Definition at line 84 of file [Board.h](#).

3.9.1.21 Board_init #define Board_init CC3220SF_STARPORTS_initGeneral

Definition at line 44 of file [Board.h](#).

3.9.1.22 Board_initGeneral #define Board_initGeneral CC3220SF_STARPORTS_initGeneral

Definition at line 45 of file [Board.h](#).

3.9.1.23 Board_LDC1000_CS #define Board_LDC1000_CS CC3220SF_STARPORTS_LDC1000_CS

Definition at line 65 of file [Board.h](#).

3.9.1.24 Board_MOSI_ASGPIO #define Board_MOSI_ASGPIO CC3220SF_STARPORTS_MOSI

Definition at line 74 of file [Board.h](#).

3.9.1.25 Board_PWM0 #define Board_PWM0 CC3220SF_STARPORTS_PWM5

Definition at line 86 of file [Board.h](#).

3.9.1.26 Board_PWM1 #define Board_PWM1 CC3220SF_STARPORTS_PWM7

Definition at line 87 of file [Board.h](#).

3.9.1.27 Board_RN2483_MCLR #define Board_RN2483_MCLR CC3220SF_STARPORTS_RN2483_MCLR

Definition at line 79 of file [Board.h](#).

3.9.1.28 Board_SCL_ASGPIO #define Board_SCL_ASGPIO CC3220SF_STARPORTS_SCL

Definition at line 71 of file [Board.h](#).

3.9.1.29 Board_SCLK_ASGPIO #define Board_SCLK_ASGPIO CC3220SF_STARPORTS_SCLK

Definition at line 73 of file [Board.h](#).

3.9.1.30 Board_SD0 #define Board_SD0 CC3220SF_STARPORTS_SD0

Definition at line 89 of file [Board.h](#).

3.9.1.31 Board_SDA_ASGPIO #define Board_SDA_ASGPIO CC3220SF_STARPORTS_SDA

Definition at line 72 of file [Board.h](#).

3.9.1.32 Board_SDFatFS0 #define Board_SDFatFS0 CC3220SF_STARPORTS_SD0

Definition at line 91 of file [Board.h](#).

3.9.1.33 Board_SPI0 #define Board_SPI0 CC3220SF_STARPORTS_SPI1

Definition at line 94 of file [Board.h](#).

3.9.1.34 Board_SPI_MASTER #define Board_SPI_MASTER CC3220SF_STARPORTS_SPI1

Definition at line 95 of file [Board.h](#).

3.9.1.35 Board_SPI_MASTER_READY #define Board_SPI_MASTER_READY CC3220SF_STARPORTS_SPI_MASTER_READY

Definition at line 97 of file [Board.h](#).

3.9.1.36 Board_SPI_SLAVE #define Board_SPI_SLAVE CC3220SF_STARPORTS_SPI1

Definition at line 96 of file [Board.h](#).

3.9.1.37 Board_SPI_SLAVE_READY #define Board_SPI_SLAVE_READY CC3220SF_STARPORTS_SPI_SLAVE_READY

Definition at line 98 of file [Board.h](#).

3.9.1.38 Board_TIMER0 #define Board_TIMER0 CC3220SF_STARPORTS_TIMER0

Definition at line 100 of file [Board.h](#).

3.9.1.39 Board_TIMER1 #define Board_TIMER1 CC3220SF_STARPORTS_TIMER1

Definition at line 101 of file [Board.h](#).

3.9.1.40 Board_TIMER2 #define Board_TIMER2 CC3220SF_STARPORTS_TIMER2

Definition at line 102 of file [Board.h](#).

3.9.1.41 Board_TMP006_ADDR #define Board_TMP006_ADDR (0x41)

Definition at line 110 of file [Board.h](#).

3.9.1.42 Board_UART0 #define Board_UART0 CC3220SF_STARPORTS_UART0

Definition at line 104 of file [Board.h](#).

3.9.1.43 Board_UART1 #define Board_UART1 CC3220SF_STARPORTS_UART1

Definition at line 105 of file [Board.h](#).

3.9.1.44 Board_WATCHDOG0 #define Board_WATCHDOG0 CC3220SF_STARPORTS_WATCHDOG0

Definition at line 107 of file [Board.h](#).

3.10 Board.h

```

00001 /*
00002 * Copyright (c) 2016-2018, Texas Instruments Incorporated
00003 * All rights reserved.
00004 *
00005 * Redistribution and use in source and binary forms, with or without
00006 * modification, are permitted provided that the following conditions
00007 * are met:
00008 *
00009 * * Redistributions of source code must retain the above copyright
00010 * notice, this list of conditions and the following disclaimer.
00011 *
00012 * * Redistributions in binary form must reproduce the above copyright
00013 * notice, this list of conditions and the following disclaimer in the
00014 * documentation and/or other materials provided with the distribution.
00015 *
00016 * * Neither the name of Texas Instruments Incorporated nor the names of
00017 * its contributors may be used to endorse or promote products derived
00018 * from this software without specific prior written permission.
00019 *
00020 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00021 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
00022 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
00023 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
00024 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
00025 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00026 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
00027 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
00028 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
00029 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
00030 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00031 */
00032
00033 #ifndef __BOARD_H
00034 #define __BOARD_H
00035
00036 #define Board_CC3220SF_STARPORTS
00037
00038 #ifdef __cplusplus
00039 extern "C" {
00040 #endif
00041
00042 #include "CC3220SF_STARPORTS.h"
00043
00044 #define Board_init CC3220SF_STARPORTS_initGeneral
00045 #define Board_initGeneral CC3220SF_STARPORTS_initGeneral
00046
00047 #define Board_ADC0 CC3220SF_STARPORTS_ADC0
00048
00049 #define Board_CAPTURE0 CC3220SF_STARPORTS_CAPTURE0
00050 #define Board_CAPTURE1 CC3220SF_STARPORTS_CAPTURE1
00051
00052 #define Board_CRYPTO0 CC3220SF_STARPORTS_CRYPTO0
00053
00054 #define Board_GPIO_LED_ON CC3220SF_STARPORTS_GPIO_LED_ON
00055 #define Board_GPIO_LED_OFF CC3220SF_STARPORTS_GPIO_LED_OFF
00056 #define Board_GPIO_LED0 CC3220SF_STARPORTS_GPIO_LED_D10
00057 /*
00058 * CC3220SF_STARPORTS_GPIO_LED_D8 and CC3220SF_STARPORTS_GPIO_LED_D9 are shared with the I2C
00059 * and PWM peripherals. In order for those examples to work, these LEDs are
00060 * taken out of gpioPinConfig[]
00061 */
00062
00063 #define Board_ADXL355_CS CC3220SF_STARPORTS_ADXL355_CS
00064 #define Board_BME280_CS CC3220SF_STARPORTS_BME280_CS
00065 #define Board_LDC1000_CS CC3220SF_STARPORTS_LDC1000_CS
00066 #define Board_EN_NODE CC3220SF_STARPORTS_EN_NODE
00067 #define Board_EN_ADXL355 CC3220SF_STARPORTS_EN_ADXL355
00068 #define Board_EN_BME280 CC3220SF_STARPORTS_EN_BME280
00069 #define Board_EN_LDC1000 CC3220SF_STARPORTS_EN_LDC1000
00070 #define Board_DS1374_INTB CC3220SF_STARPORTS_DS1374_INTB
00071 #define Board_SCL_ASGPIO CC3220SF_STARPORTS_SCL
00072 #define Board_SDA_ASGPIO CC3220SF_STARPORTS_SDA
00073 #define Board_SCLK_ASGPIO CC3220SF_STARPORTS_SCLK
00074 #define Board_MOSI_ASGPIO CC3220SF_STARPORTS_MOSI
00075 #define Board_CS_ASGPIO CC3220SF_STARPORTS_CS
00076
00077
00078
00079 #define Board_RN2483_MCLR CC3220SF_STARPORTS_RN2483_MCLR
00080
00081 #define Board_I2C0 CC3220SF_STARPORTS_I2C0
00082 #define Board_I2C_TMP CC3220SF_STARPORTS_I2C0
00083
00084 #define Board_I2S0 CC3220SF_STARPORTS_I2S0
00085

```

```

00086 #define Board_PWM0          CC3220SF_STARPORTS_PWM5
00087 #define Board_PWM1          CC3220SF_STARPORTS_PWM7
00088
00089 #define Board_SD0           CC3220SF_STARPORTS_SDO
00090
00091 #define Board_SDFatFS0      CC3220SF_STARPORTS_SDO
00092
00093 /* CC3220SF_STARPORTS_SPI0 is reserved for the NWP */
00094 #define Board_SPI0           CC3220SF_STARPORTS_SPI1
00095 #define Board_SPI_MASTER     CC3220SF_STARPORTS_SPI1
00096 #define Board_SPI_SLAVE       CC3220SF_STARPORTS_SPI1
00097 #define Board_SPI_MASTER_READY CC3220SF_STARPORTS_SPI_MASTER_READY
00098 #define Board_SPI_SLAVE_READY  CC3220SF_STARPORTS_SPI_SLAVE_READY
00099
00100 #define Board_TIMER0          CC3220SF_STARPORTS_TIMER0
00101 #define Board_TIMER1          CC3220SF_STARPORTS_TIMER1
00102 #define Board_TIMER2          CC3220SF_STARPORTS_TIMER2
00103
00104 #define Board_UART0          CC3220SF_STARPORTS_UART0
00105 #define Board_UART1          CC3220SF_STARPORTS_UART1
00106
00107 #define Board_WATCHDOG0      CC3220SF_STARPORTS_WATCHDOG0
00108
00109 /* Board specific I2C address */
00110 #define Board_TMP006_ADDR      (0x41)
00111 #define Board_I2C_TMP006_ADDR   Board_TMP006_ADDR
00112
00113 #ifdef __cplusplus
00114 }
00115 #endif
00116
00117 #endif /* __BOARD_H */

```

3.11 CC3220SF_LAUNCHXL.h File Reference

Macros

- `#define CC3220SF_LAUNCHXL_GPIO_LED_OFF (0)`
- `#define CC3220SF_LAUNCHXL_GPIO_LED_ON (1)`

Enumerations

- enum `CC3220SF_LAUNCHXL_ADCName` { `CC3220SF_LAUNCHXL_ADC0` = 0, `CC3220SF_LAUNCHXL_ADC1`, `CC3220SF_LAUNCHXL_ADCCOUNT` }
- enum `CC3220SF_LAUNCHXL_CaptureName` { `CC3220SF_LAUNCHXL_CAPTURE0` = 0, `CC3220SF_LAUNCHXL_CAPTURE1`, `CC3220SF_LAUNCHXL_CAPTURECOUNT` }
- enum `CC3220SF_LAUNCHXL_CryptoName` { `CC3220SF_LAUNCHXL_CRYPTO0` = 0, `CC3220SF_LAUNCHXL_CRYPTO1`, ... }
- enum `CC3220SF_LAUNCHXL_GPIOName` {
`CC3220SF_LAUNCHXL_GPIO_SW2` = 0, `CC3220SF_LAUNCHXL_GPIO_SW3`, `CC3220SF_LAUNCHXL_SPI_MASTER_READY`, `CC3220SF_LAUNCHXL_SPI_SLAVE_READY`, `CC3220SF_LAUNCHXL_GPIO_LED_D10`, `CC3220SF_LAUNCHXL_GPIO_TMP116_EN`, `CC3220SF_LAUNCHXL_LCD_CS`, `CC3220SF_LAUNCHXL_LCD_POWER`, `CC3220SF_LAUNCHXL_LCD_ENABLE`, `CC3220SF_LAUNCHXL_GPIOCOUNT` }
- enum `CC3220SF_LAUNCHXL_I2CName` { `CC3220SF_LAUNCHXL_I2C0` = 0, `CC3220SF_LAUNCHXL_I2CCOUNT` }
- enum `CC3220SF_LAUNCHXL_I2SName` { `CC3220SF_LAUNCHXL_I2S0` = 0, `CC3220SF_LAUNCHXL_I2SCOUNT` }
- enum `CC3220SF_LAUNCHXL_PWMName` { `CC3220SF_LAUNCHXL_PWM6` = 0, `CC3220SF_LAUNCHXL_PWM7`, `CC3220SF_LAUNCHXL_PWMCOUNT` }
- enum `CC3220SF_LAUNCHXL_SDName` { `CC3220SF_LAUNCHXL_SD0` = 0, `CC3220SF_LAUNCHXL_SDCount` }
- enum `CC3220SF_LAUNCHXL_SDName` { `CC3220SF_LAUNCHXL_SD0` = 0, `CC3220SF_LAUNCHXL_SDCount` }

- enum `CC3220SF_LAUNCHXL_SPIName` { `CC3220SF_LAUNCHXL_SPI0` = 0, `CC3220SF_LAUNCHXL_SPI1`, `CC3220SF_LAUNCHXL_SPICOUNT` }
- enum `CC3220SF_LAUNCHXL_TimerName` { `CC3220SF_LAUNCHXL_TIMER0` = 0, `CC3220SF_LAUNCHXL_TIMER1`, `CC3220SF_LAUNCHXL_TIMER2`, `CC3220SF_LAUNCHXL_TIMERCOUNT` }
- enum `CC3220SF_LAUNCHXL_UARTName` { `CC3220SF_LAUNCHXL_UART0` = 0, `CC3220SF_LAUNCHXL_UART1`, `CC3220SF_LAUNCHXL_UARTCOUNT` }
- enum `CC3220SF_LAUNCHXL_WatchdogName` { `CC3220SF_LAUNCHXL_WATCHDOG0` = 0, `CC3220SF_LAUNCHXL_WATCHDOG1` }

Functions

- void `CC3220SF_LAUNCHXL_initGeneral` (void)

Initialize the general board specific settings.

3.11.1 Macro Definition Documentation

3.11.1.1 CC3220SF_LAUNCHXL_GPIO_LED_OFF #define CC3220SF_LAUNCHXL_GPIO_LED_OFF (0)

Definition at line 52 of file `CC3220SF_LAUNCHXL.h`.

3.11.1.2 CC3220SF_LAUNCHXL_GPIO_LED_ON #define CC3220SF_LAUNCHXL_GPIO_LED_ON (1)

Definition at line 53 of file `CC3220SF_LAUNCHXL.h`.

3.11.2 Enumeration Type Documentation

3.11.2.1 CC3220SF_LAUNCHXL_ADCName enum CC3220SF_LAUNCHXL_ADCName

Enumerator

<code>CC3220SF_LAUNCHXL_ADC0</code>	
<code>CC3220SF_LAUNCHXL_ADC1</code>	
<code>CC3220SF_LAUNCHXL_ADCCOUNT</code>	

Definition at line 59 of file `CC3220SF_LAUNCHXL.h`.

```
00059
00060     CC3220SF_LAUNCHXL_ADC0 = 0,
00061     CC3220SF_LAUNCHXL_ADC1,
00062
00063     CC3220SF_LAUNCHXL_ADCCOUNT
00064 } CC3220SF_LAUNCHXL_ADCName;
```

3.11.2.2 CC3220SF_LAUNCHXL_CaptureName enum CC3220SF_LAUNCHXL_CaptureName

Enumerator

CC3220SF_LAUNCHXL_CAPTURE0	
CC3220SF_LAUNCHXL_CAPTURE1	
CC3220SF_LAUNCHXL_CAPTURECOUNT	

Definition at line 70 of file [CC3220SF_LAUNCHXL.h](#).

```
00070 {  
00071     CC3220SF_LAUNCHXL_CAPTURE0 = 0,  
00072     CC3220SF_LAUNCHXL_CAPTURE1,  
00073  
00074     CC3220SF_LAUNCHXL_CAPTURECOUNT  
00075 } CC3220SF_LAUNCHXL_CaptureName;
```

3.11.2.3 CC3220SF_LAUNCHXL_CryptoName enum CC3220SF_LAUNCHXL_CryptoName

Enumerator

CC3220SF_LAUNCHXL_CRYPTO0	
CC3220SF_LAUNCHXL_CRYPTOCOUNT	

Definition at line 81 of file [CC3220SF_LAUNCHXL.h](#).

```
00081 {  
00082     CC3220SF_LAUNCHXL_CRYPTO0 = 0,  
00083  
00084     CC3220SF_LAUNCHXL_CRYPTOCOUNT  
00085 } CC3220SF_LAUNCHXL_CryptoName;
```

3.11.2.4 CC3220SF_LAUNCHXL_GPIOName enum CC3220SF_LAUNCHXL_GPIOName

Enumerator

CC3220SF_LAUNCHXL_GPIO_SW2	
CC3220SF_LAUNCHXL_GPIO_SW3	
CC3220SF_LAUNCHXL_SPI_MASTER_READY	
CC3220SF_LAUNCHXL_SPI_SLAVE_READY	
CC3220SF_LAUNCHXL_GPIO_LED_D10	
CC3220SF_LAUNCHXL_GPIO_TMP116_EN	
CC3220SF_LAUNCHXL_LCD_CS	
CC3220SF_LAUNCHXL_LCD_POWER	
CC3220SF_LAUNCHXL_LCD_ENABLE	
CC3220SF_LAUNCHXL_GPIOCOUNT	

Definition at line 91 of file [CC3220SF_LAUNCHXL.h](#).

```
00091 {  
00092     CC3220SF_LAUNCHXL_GPIO_SW2 = 0,  
00093     CC3220SF_LAUNCHXL_GPIO_SW3,  
00094     CC3220SF_LAUNCHXL_SPI_MASTER_READY,  
00095     CC3220SF_LAUNCHXL_SPI_SLAVE_READY,  
00096     CC3220SF_LAUNCHXL_GPIO_LED_D10,
```

```

00097
00098 /* 
00099 * CC3220SF_LAUNCHXL_GPIO_LED_D8 and CC3220SF_LAUNCHXL_GPIO_LED_D9 are shared with the
00100 * I2C and PWM peripherals. In order for those examples to work, these
00101 * LEDs are taken out of gpioPinConfig[]
00102 */
00103 /* CC3220SF_LAUNCHXL_GPIO_LED_D9, */
00104 /* CC3220SF_LAUNCHXL_GPIO_LED_D8, */
00105
00106 CC3220SF_LAUNCHXL_GPIO_TMP116_EN,
00107
00108 /* Sharp LCD Pins */
00109 CC3220SF_LAUNCHXL_LCD_CS,
00110 CC3220SF_LAUNCHXL_LCD_POWER,
00111 CC3220SF_LAUNCHXL_LCD_ENABLE,
00112
00113 CC3220SF_LAUNCHXL_GPIOCOUNT
00114 } CC3220SF_LAUNCHXL_GPIOName;

```

3.11.2.5 CC3220SF_LAUNCHXL_I2CName enum CC3220SF_LAUNCHXL_I2CName

Enumerator

CC3220SF_LAUNCHXL_I2C0	
CC3220SF_LAUNCHXL_I2CCOUNT	

Definition at line 120 of file [CC3220SF_LAUNCHXL.h](#).

```

00120 {
00121     CC3220SF_LAUNCHXL_I2C0 = 0,
00122
00123     CC3220SF_LAUNCHXL_I2CCOUNT
00124 } CC3220SF_LAUNCHXL_I2CName;

```

3.11.2.6 CC3220SF_LAUNCHXL_I2SName enum CC3220SF_LAUNCHXL_I2SName

Enumerator

CC3220SF_LAUNCHXL_I2S0	
CC3220SF_LAUNCHXL_I2SCOUNT	

Definition at line 130 of file [CC3220SF_LAUNCHXL.h](#).

```

00130 {
00131     CC3220SF_LAUNCHXL_I2S0 = 0,
00132
00133     CC3220SF_LAUNCHXL_I2SCOUNT
00134 } CC3220SF_LAUNCHXL_I2SName;

```

3.11.2.7 CC3220SF_LAUNCHXL_PWMName enum CC3220SF_LAUNCHXL_PWMName

Enumerator

CC3220SF_LAUNCHXL_PWM6	
CC3220SF_LAUNCHXL_PWM7	
CC3220SF_LAUNCHXL_PWMCOUNT	

Definition at line 140 of file [CC3220SF_LAUNCHXL.h](#).

```
00140     {  
00141         CC3220SF_LAUNCHXL_PWM6 = 0,  
00142         CC3220SF_LAUNCHXL_PWM7,  
00143         CC3220SF_LAUNCHXL_PWMCOUNT  
00145 } CC3220SF_LAUNCHXL_PWMName;
```

3.11.2.8 CC3220SF_LAUNCHXL_SDFatFSName enum [CC3220SF_LAUNCHXL_SDFatFSName](#)

Enumerator

CC3220SF_LAUNCHXL_SDFatFS0	
CC3220SF_LAUNCHXL_SDFatFSCOUNT	

Definition at line 151 of file [CC3220SF_LAUNCHXL.h](#).

```
00151     {  
00152         CC3220SF_LAUNCHXL_SDFatFS0 = 0,  
00153         CC3220SF_LAUNCHXL_SDFatFSCOUNT  
00155 } CC3220SF_LAUNCHXL_SDFatFSName;
```

3.11.2.9 CC3220SF_LAUNCHXL_SDName enum [CC3220SF_LAUNCHXL_SDName](#)

Enumerator

CC3220SF_LAUNCHXL_SD0	
CC3220SF_LAUNCHXL_SDCOUNT	

Definition at line 161 of file [CC3220SF_LAUNCHXL.h](#).

```
00161     {  
00162         CC3220SF_LAUNCHXL_SD0 = 0,  
00163         CC3220SF_LAUNCHXL_SDCOUNT  
00165 } CC3220SF_LAUNCHXL_SDName;
```

3.11.2.10 CC3220SF_LAUNCHXL_SPIName enum [CC3220SF_LAUNCHXL_SPIName](#)

Enumerator

CC3220SF_LAUNCHXL_SPI0	
CC3220SF_LAUNCHXL_SPI1	
CC3220SF_LAUNCHXL_SPICOUNT	

Definition at line 171 of file [CC3220SF_LAUNCHXL.h](#).

```
00171     {  
00172         CC3220SF_LAUNCHXL_SPI0 = 0,  
00173         CC3220SF_LAUNCHXL_SPI1,  
00174         CC3220SF_LAUNCHXL_SPICOUNT  
00176 } CC3220SF_LAUNCHXL_SPIName;
```

3.11.2.11 CC3220SF_LAUNCHXL_TimerName enum [CC3220SF_LAUNCHXL_TimerName](#)

Enumerator

CC3220SF_LAUNCHXL_TIMER0
CC3220SF_LAUNCHXL_TIMER1
CC3220SF_LAUNCHXL_TIMER2
CC3220SF_LAUNCHXL_TIMERCOUNT

Definition at line 182 of file [CC3220SF_LAUNCHXL.h](#).

```
00182 {  
00183     CC3220SF_LAUNCHXL_TIMER0 = 0,  
00184     CC3220SF_LAUNCHXL_TIMER1,  
00185     CC3220SF_LAUNCHXL_TIMER2,  
00186     CC3220SF_LAUNCHXL_TIMERCOUNT  
00188 } CC3220SF_LAUNCHXL_TimerName;
```

3.11.2.12 CC3220SF_LAUNCHXL_UARTName enum [CC3220SF_LAUNCHXL_UARTName](#)

Enumerator

CC3220SF_LAUNCHXL_UART0
CC3220SF_LAUNCHXL_UART1
CC3220SF_LAUNCHXL_UARTCOUNT

Definition at line 194 of file [CC3220SF_LAUNCHXL.h](#).

```
00194 {  
00195     CC3220SF_LAUNCHXL_UART0 = 0,  
00196     CC3220SF_LAUNCHXL_UART1,  
00197     CC3220SF_LAUNCHXL_UARTCOUNT  
00199 } CC3220SF_LAUNCHXL_UARTName;
```

3.11.2.13 CC3220SF_LAUNCHXL_WatchdogName enum [CC3220SF_LAUNCHXL_WatchdogName](#)

Enumerator

CC3220SF_LAUNCHXL_WATCHDOG0
CC3220SF_LAUNCHXL_WATCHDOGCOUNT

Definition at line 205 of file [CC3220SF_LAUNCHXL.h](#).

```
00205 {  
00206     CC3220SF_LAUNCHXL_WATCHDOG0 = 0,  
00207     CC3220SF_LAUNCHXL_WATCHDOGCOUNT  
00209 } CC3220SF_LAUNCHXL_WatchdogName;
```

3.11.3 Function Documentation

3.11.3.1 CC3220SF_LAUNCHXL_initGeneral() void CC3220SF_LAUNCHXL_initGeneral (void)

Initialize the general board specific settings.

This function initializes the general board specific settings.

3.12 CC3220SF_LAUNCHXL.h

```
00001 /*
00002 * Copyright (c) 2016–2018, Texas Instruments Incorporated
00003 * All rights reserved.
00004 *
00005 * Redistribution and use in source and binary forms, with or without
00006 * modification, are permitted provided that the following conditions
00007 * are met:
00008 *
00009 * * Redistributions of source code must retain the above copyright
00010 * notice, this list of conditions and the following disclaimer.
00011 *
00012 * * Redistributions in binary form must reproduce the above copyright
00013 * notice, this list of conditions and the following disclaimer in the
00014 * documentation and/or other materials provided with the distribution.
00015 *
00016 * * Neither the name of Texas Instruments Incorporated nor the names of
00017 * its contributors may be used to endorse or promote products derived
00018 * from this software without specific prior written permission.
00019 *
00020 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00021 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
00022 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
00023 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
00024 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
00025 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00026 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
00027 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
00028 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
00029 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
00030 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00031 */
00045 #ifndef __CC3220SF_LAUNCHXL_H
00046 #define __CC3220SF_LAUNCHXL_H
00047
00048 #ifdef __cplusplus
00049 extern "C" {
00050 #endif
00051
00052 #define CC3220SF_LAUNCHXL_GPIO_LED_OFF (0)
00053 #define CC3220SF_LAUNCHXL_GPIO_LED_ON (1)
00054
00055 typedef enum CC3220SF_LAUNCHXL_ADCName {
00056     CC3220SF_LAUNCHXL_ADC0 = 0,
00057     CC3220SF_LAUNCHXL_ADC1,
00058
00059     CC3220SF_LAUNCHXL_ADCCOUNT
00060 } CC3220SF_LAUNCHXL_ADCName;
00061
00062
00063 typedef enum CC3220SF_LAUNCHXL_CaptureName {
00064     CC3220SF_LAUNCHXL_CAPTURE0 = 0,
00065     CC3220SF_LAUNCHXL_CAPTURE1,
00066
00067     CC3220SF_LAUNCHXL_CAPTURECOUNT
00068 } CC3220SF_LAUNCHXL_CaptureName;
00069
00070 typedef enum CC3220SF_LAUNCHXL_CryptoName {
00071     CC3220SF_LAUNCHXL_CRYPTO0 = 0,
00072     CC3220SF_LAUNCHXL_CRYPTO1,
00073
00074     CC3220SF_LAUNCHXL_CRYPTOCOUNT
00075 } CC3220SF_LAUNCHXL_CryptoName;
00076
00077
00078 typedef enum CC3220SF_LAUNCHXL_GPIOName {
00079     CC3220SF_LAUNCHXL_GPIO_SW2 = 0,
00080     CC3220SF_LAUNCHXL_GPIO_SW3,
```

```

00094     CC3220SF_LAUNCHXL_SPI_MASTER_READY,
00095     CC3220SF_LAUNCHXL_SPI_SLAVE_READY,
00096     CC3220SF_LAUNCHXL_GPIO_LED_D10,
00097
00098     /*
00099      * CC3220SF_LAUNCHXL_GPIO_LED_D8 and CC3220SF_LAUNCHXL_GPIO_LED_D9 are shared with the
00100      * I2C and PWM peripherals. In order for those examples to work, these
00101      * LEDs are taken out of gpioPinConfig[]
00102      */
00103     /* CC3220SF_LAUNCHXL_GPIO_LED_D9, */
00104     /* CC3220SF_LAUNCHXL_GPIO_LED_D8, */
00105
00106     CC3220SF_LAUNCHXL_GPIO_TMP116_EN,
00107
00108     /* Sharp LCD Pins */
00109     CC3220SF_LAUNCHXL_LCD_CS,
00110     CC3220SF_LAUNCHXL_LCD_POWER,
00111     CC3220SF_LAUNCHXL_LCD_ENABLE,
00112
00113     CC3220SF_LAUNCHXL_GPIOCOUNT
00114 } CC3220SF_LAUNCHXL_GPIOName;
00115
00120 typedef enum CC3220SF_LAUNCHXL_I2CName {
00121     CC3220SF_LAUNCHXL_I2C0 = 0,
00122
00123     CC3220SF_LAUNCHXL_I2CCOUNT
00124 } CC3220SF_LAUNCHXL_I2CName;
00125
00130 typedef enum CC3220SF_LAUNCHXL_I2SName {
00131     CC3220SF_LAUNCHXL_I2S0 = 0,
00132
00133     CC3220SF_LAUNCHXL_I2SCOUNT
00134 } CC3220SF_LAUNCHXL_I2SName;
00135
00140 typedef enum CC3220SF_LAUNCHXL_PWMName {
00141     CC3220SF_LAUNCHXL_PWM6 = 0,
00142     CC3220SF_LAUNCHXL_PWM7,
00143
00144     CC3220SF_LAUNCHXL_PWMCOUNT
00145 } CC3220SF_LAUNCHXL_PWMName;
00146
00151 typedef enum CC3220SF_LAUNCHXL_SDName {
00152     CC3220SF_LAUNCHXL_SD0 = 0,
00153
00154     CC3220SF_LAUNCHXL_SDCount
00155 } CC3220SF_LAUNCHXL_SDName;
00156
00161 typedef enum CC3220SF_LAUNCHXL_SDName {
00162     CC3220SF_LAUNCHXL_SD0 = 0,
00163
00164     CC3220SF_LAUNCHXL_SDCOUNT
00165 } CC3220SF_LAUNCHXL_SDName;
00166
00171 typedef enum CC3220SF_LAUNCHXL_SPIName {
00172     CC3220SF_LAUNCHXL_SPI0 = 0,
00173     CC3220SF_LAUNCHXL_SPI1,
00174
00175     CC3220SF_LAUNCHXL_SPICOUNT
00176 } CC3220SF_LAUNCHXL_SPIName;
00177
00182 typedef enum CC3220SF_LAUNCHXL_TimerName {
00183     CC3220SF_LAUNCHXL_TIMER0 = 0,
00184     CC3220SF_LAUNCHXL_TIMER1,
00185     CC3220SF_LAUNCHXL_TIMER2,
00186
00187     CC3220SF_LAUNCHXL_TIMERCOUNT
00188 } CC3220SF_LAUNCHXL_TimerName;
00189
00194 typedef enum CC3220SF_LAUNCHXL_UARTName {
00195     CC3220SF_LAUNCHXL_UART0 = 0,
00196     CC3220SF_LAUNCHXL_UART1,
00197
00198     CC3220SF_LAUNCHXL_UARTCOUNT
00199 } CC3220SF_LAUNCHXL_UARTName;
00200
00205 typedef enum CC3220SF_LAUNCHXL_WatchdogName {
00206     CC3220SF_LAUNCHXL_WATCHDOGO = 0,
00207
00208     CC3220SF_LAUNCHXL_WATCHDOGCOUNT
00209 } CC3220SF_LAUNCHXL_WatchdogName;
00210
00216 extern void CC3220SF_LAUNCHXL_initGeneral(void);
00217
00218 #ifdef __cplusplus
00219 }
00220 #endif
00221
00222

```

```
00222 #endif /* __CC3220SF_LAUNCHXL_H */
```

3.13 CC3220SF_STARPORTS.c File Reference

```
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <ti/devices/cc32xx/inc/hw_ints.h>
#include <ti/devices/cc32xx/inc/hw_memmap.h>
#include <ti/devices/cc32xx/inc/hw_types.h>
#include <ti/devices/cc32xx/driverlib/rom.h>
#include <ti/devices/cc32xx/driverlib/rom_map.h>
#include <ti/devices/cc32xx/driverlib/adc.h>
#include <ti/devices/cc32xx/driverlib/gpio.h>
#include <ti/devices/cc32xx/driverlib/pin.h>
#include <ti/devices/cc32xx/driverlib/prcm.h>
#include <ti/devices/cc32xx/driverlib/spi.h>
#include <ti/devices/cc32xx/driverlib/sdhost.h>
#include <ti/devices/cc32xx/driverlib/timer.h>
#include <ti/devices/cc32xx/driverlib/uart.h>
#include <ti/devices/cc32xx/driverlib/udma.h>
#include <ti/devices/cc32xx/driverlib/wdt.h>
#include <ti/drivers/Power.h>
#include <ti/drivers/power/PowerCC32XX.h>
#include "CC3220SF_STARPORTS.h"
#include <ti/drivers/ADC.h>
#include <ti/drivers/adc/ADCCC32XX.h>
#include <ti/drivers/Capture.h>
#include <ti/drivers/capture/CaptureCC32XX.h>
#include <ti/drivers/crypto/CryptoCC32XX.h>
#include <ti/drivers/dma/UDMACC32XX.h>
#include <ti/drivers/GPIO.h>
#include <ti/drivers/gpio/GPIOCC32XX.h>
#include <ti/drivers/I2C.h>
#include <ti/drivers/i2c/I2CCC32XX.h>
#include <ti/drivers/PWM.h>
#include <ti/drivers/pwm/PWMTimerCC32XX.h>
#include <ti/drivers/SPI.h>
#include <ti/drivers/spi/SPICC32XXDMA.h>
#include <ti/drivers/Timer.h>
#include <ti/drivers/timer/TimerCC32XX.h>
#include <ti/drivers/UART.h>
#include <ti/drivers/uart/UARTCC32XX.h>
#include <ti/drivers/Watchdog.h>
#include <ti/drivers/watchdog/WatchdogCC32XX.h>
```

Macros

- `#define TI_DRIVERS_UART_DMA 0`

Functions

- void [CC3220SF_STARPORTS_initGeneral](#)(void)
Initialize the general board specific settings.
- void [WakeupFromLPDS](#)(`uint_least8_t`)

Variables

- const ADC_Config ADC_config [CC3220SF_STARPORTS_ADCCOUNT]
- const uint_least8_t ADC_count = CC3220SF_STARPORTS_ADCCOUNT
- const ADCCC32XX_HWAttrsV1 adcCC3220SHWAttrs [CC3220SF_STARPORTS_ADCCOUNT]
- ADCCC32XX_Object adcCC3220SObjects [CC3220SF_STARPORTS_ADCCOUNT]
- const Capture_Config Capture_config [CC3220SF_STARPORTS_CAPTURECOUNT]
- const uint_least8_t Capture_count = CC3220SF_STARPORTS_CAPTURECOUNT
- const CaptureCC32XX_HWAttrs captureCC3220SFHWAttrs [CC3220SF_STARPORTS_CAPTURECOUNT]
- CaptureCC32XX_Object captureCC3220SFObjets [CC3220SF_STARPORTS_CAPTURECOUNT]
- CryptoCC32XX_Object cryptoCC3220SObjects [CC3220SF_STARPORTS_CRYPTOCOUNT]
- const CryptoCC32XX_Config CryptoCC32XX_config [CC3220SF_STARPORTS_CRYPTOCOUNT]
- const uint_least8_t CryptoCC32XX_count = CC3220SF_STARPORTS_CRYPTOCOUNT
- GPIO_CallbackFxn gpioCallbackFunctions []
- const GPIOCC32XX_Config GPIOCC32XX_config
- GPIO_PinConfig gpioPinConfigs []
- const I2C_Config I2C_config [CC3220SF_STARPORTS_I2CCOUNT]
- const uint_least8_t I2C_count = CC3220SF_STARPORTS_I2CCOUNT
- const I2CCC32XX_HWAttrsV1 i2cCC3220SHWAttrs [CC3220SF_STARPORTS_I2CCOUNT]
- I2CCC32XX_Object i2cCC3220SObjects [CC3220SF_STARPORTS_I2CCOUNT]
- PowerCC32XX_ParkInfo parkInfo []
- const PowerCC32XX_ConfigV1 PowerCC32XX_config
- const PWM_Config PWM_config [CC3220SF_STARPORTS_PWMCOUNT]
- const uint_least8_t PWM_count = CC3220SF_STARPORTS_PWMCOUNT
- const PWMTimerCC32XX_HWAttrsV2 pwmTimerCC3220SHWAttrs [CC3220SF_STARPORTS_PWMCOUNT]
- PWMTimerCC32XX_Object pwmTimerCC3220SObjects [CC3220SF_STARPORTS_PWMCOUNT]
- const SPI_Config SPI_config [CC3220SF_STARPORTS_SPICOUNT]
- const uint_least8_t SPI_count = CC3220SF_STARPORTS_SPICOUNT
- const SPICC32XXDMA_HWAttrsV1 spiCC3220SDMAHWAttrs [CC3220SF_STARPORTS_SPICOUNT]
- SPICC32XXDMA_Object spiCC3220SDMAObjects [CC3220SF_STARPORTS_SPICOUNT]
- uint32_t spiCC3220SDMAscratchBuf [CC3220SF_STARPORTS_SPICOUNT]
- const Timer_Config Timer_config [CC3220SF_STARPORTS_TIMERCOUNT]
- const uint_least8_t Timer_count = CC3220SF_STARPORTS_TIMERCOUNT
- const TimerCC32XX_HWAttrs timerCC3220SFHWAttrs [CC3220SF_STARPORTS_TIMERCOUNT]
- TimerCC32XX_Object timerCC3220SFObjets [CC3220SF_STARPORTS_TIMERCOUNT]
- const UART_Config UART_config [CC3220SF_STARPORTS_UARTCOUNT]
- const uint_least8_t UART_count = CC3220SF_STARPORTS_UARTCOUNT
- const UARTCC32XX_HWAttrsV1 uartCC3220SHWAttrs [CC3220SF_STARPORTS_UARTCOUNT]
- UARTCC32XX_Object uartCC3220SObjects [CC3220SF_STARPORTS_UARTCOUNT]
- unsigned char uartCC3220SRingBuffer [CC3220SF_STARPORTS_UARTCOUNT][32]
- const UDMACC32XX_HWAttrs udmaCC3220SHWAttrs
- UDMACC32XX_Object udmaCC3220SObject
- const UDMACC32XX_Config UDMACC32XX_config
- const Watchdog_Config Watchdog_config [CC3220SF_STARPORTS_WATCHDOGCOUNT]
- const uint_least8_t Watchdog_count = CC3220SF_STARPORTS_WATCHDOGCOUNT
- const WatchdogCC32XX_HWAttrs watchdogCC3220SHWAttrs [CC3220SF_STARPORTS_WATCHDOGCOUNT]
- WatchdogCC32XX_Object watchdogCC3220SObjects [CC3220SF_STARPORTS_WATCHDOGCOUNT]

3.13.1 Macro Definition Documentation

3.13.1.1 TI_DRIVERS_UART_DMA #define TI_DRIVERS_UART_DMA 0

Definition at line 71 of file [CC3220SF_STARPORTS.c](#).

3.13.2 Function Documentation**3.13.2.1 CC3220SF_STARPORTS_initGeneral()** void CC3220SF_STARPORTS_initGeneral (void)

Initialize the general board specific settings.

This function initializes the general board specific settings.

Definition at line 190 of file [CC3220SF_STARPORTS.c](#).

```
00191 {  
00192     PRCMCC3200MCUInit();  
00193     Power_init();  
00194 }
```

3.13.2.2 WakeupFromLPDS() void WakeupFromLPDS (uint_least8_t var)

Definition at line 332 of file [STARPORTS_App.c](#).

```
00332 {  
00333     LPDSOut = 1;  
00334 }
```

3.13.3 Variable Documentation**3.13.3.1 ADC_config** const ADC_Config ADC_config[[CC3220SF_STARPORTS_ADCCOUNT](#)]**Initial value:**

```
= {  
    .fxnTablePtr = &ADCCC32XX_fxnTable,  
    .object = &adcCC3220SObjects[CC3220SF\_STARPORTS\_ADC0],  
    .hwAttrs = &adccc3220SHWAttrs[CC3220SF\_STARPORTS\_ADC0]  
}
```

Definition at line 88 of file [CC3220SF_STARPORTS.c](#).

3.13.3.2 ADC_count const uint_least8_t ADC_count = [CC3220SF_STARPORTS_ADCCOUNT](#)

Definition at line 96 of file [CC3220SF_STARPORTS.c](#).

3.13.3.3 adcCC3220SHWAttrs const ADCCC32XX_HWAttrsV1 adcCC3220SHWAttrs[CC3220SF_STARPORTS_ADCCOUNT]

Initial value:

```
= {
    {
        .adcPin = ADCCC32XX_PIN_60_CH_3
    }
}
```

Definition at line 82 of file [CC3220SF_STARPORTS.c](#).

3.13.3.4 adcCC3220SObjects ADCCC32XX_Object adcCC3220SObjects[CC3220SF_STARPORTS_ADCCOUNT]

Definition at line 80 of file [CC3220SF_STARPORTS.c](#).

3.13.3.5 Capture_config const Capture_Config Capture_config[CC3220SF_STARPORTS_CAPTURECOUNT]

Initial value:

```
= {
    {
        .fxnTablePtr = &CaptureCC32XX_fxnTable,
        .object = &captureCC3220SObjects[CC3220SF_STARPORTS_CAPTURE0],
        .hwAttrs = &captureCC3220SFHWAttrs[CC3220SF_STARPORTS_CAPTURE0]
    },
    {
        .fxnTablePtr = &CaptureCC32XX_fxnTable,
        .object = &captureCC3220SObjects[CC3220SF_STARPORTS_CAPTURE1],
        .hwAttrs = &captureCC3220SFHWAttrs[CC3220SF_STARPORTS_CAPTURE1]
    }
}
```

Definition at line 118 of file [CC3220SF_STARPORTS.c](#).

3.13.3.6 Capture_count const uint_least8_t Capture_count = CC3220SF_STARPORTS_CAPTURECOUNT

Definition at line 131 of file [CC3220SF_STARPORTS.c](#).

3.13.3.7 captureCC3220SFHWAttrs const CaptureCC32XX_HWAttrs captureCC3220SFHWAttrs[CC3220SF_STARPORTS_CAPTURECOUNT]

Initial value:

```
=
{
    {
        .capturePin = CaptureCC32XX_PIN_04,
        .intPriority = ~0
    },
    {
        .capturePin = CaptureCC32XX_PIN_05,
        .intPriority = ~0
    },
}
```

Definition at line 106 of file [CC3220SF_STARPORTS.c](#).

3.13.3.8 captureCC3220SFObjects CaptureCC32XX_Object captureCC3220SFObjects[[CC3220SF_STARPORTS_CAPTURECOUNT](#)]

Definition at line 104 of file [CC3220SF_STARPORTS.c](#).

3.13.3.9 cryptoCC3220SObjects CryptoCC32XX_Object cryptoCC3220SObjects[[CC3220SF_STARPORTS_CRYPTOCOUNT](#)]

Definition at line 138 of file [CC3220SF_STARPORTS.c](#).

3.13.3.10 CryptoCC32XX_config const CryptoCC32XX_Config CryptoCC32XX_config[[CC3220SF_STARPORTS_CRYPTOCOUNT](#)]**Initial value:**

```
= {  
    .object = &cryptoCC3220SObjects[CC3220SF_STARPORTS_CRYPTO0]  
}
```

Definition at line 140 of file [CC3220SF_STARPORTS.c](#).

3.13.3.11 CryptoCC32XX_count const uint_least8_t CryptoCC32XX_count = [CC3220SF_STARPORTS_CRYPTOCOUNT](#)

Definition at line 146 of file [CC3220SF_STARPORTS.c](#).

3.13.3.12 gpioCallbackFunctions GPIO_CallbackFxn gpioCallbackFunctions[]**Initial value:**

```
= {  
    NULL,  
    NULL,  
    NULL,  
    NULL,  
    NULL  
}
```

Definition at line 238 of file [CC3220SF_STARPORTS.c](#).

3.13.3.13 GPIOCC32XX_config const GPIOCC32XX_Config GPIOCC32XX_config**Initial value:**

```
= {  
    .pinConfigs = (GPIO_PinConfig *)gpioPinConfigs,  
    .callbacks = (GPIO_CallbackFxn *)gpioCallbackFunctions,  
    .numberOfPinConfigs = sizeof(gpioPinConfigs)/sizeof(GPIO_PinConfig),  
    .numberOfCallbacks = sizeof(gpioCallbackFunctions)/sizeof(GPIO_CallbackFxn),  
    .intPriority = (-0)  
}
```

Definition at line 246 of file [CC3220SF_STARPORTS.c](#).

3.13.3.14 gpioPinConfigs GPIO_PinConfig gpioPinConfigs[]**Initial value:**

```
= {
    GPIOCC32XX_GPIO_30 | GPIO_DO_NOT_CONFIG,
    GPIOCC32XX_GPIO_12 | GPIO_DO_NOT_CONFIG,
    GPIOCC32XX_GPIO_13 | GPIO_DO_NOT_CONFIG,
    GPIOCC32XX_GPIO_22 | GPIO_DO_NOT_CONFIG,
    GPIOCC32XX_GPIO_08 | GPIO_DO_NOT_CONFIG,
    GPIOCC32XX_GPIO_06 | GPIO_DO_NOT_CONFIG,
    GPIOCC32XX_GPIO_00 | GPIO_DO_NOT_CONFIG,
    GPIOCC32XX_GPIO_28 | GPIO_DO_NOT_CONFIG,
    GPIOCC32XX_GPIO_07 | GPIO_DO_NOT_CONFIG,
    GPIOCC32XX_GPIO_10 | GPIO_DO_NOT_CONFIG,
    GPIOCC32XX_GPIO_11 | GPIO_DO_NOT_CONFIG,
    GPIOCC32XX_GPIO_14 | GPIO_DO_NOT_CONFIG,
    GPIOCC32XX_GPIO_16 | GPIO_DO_NOT_CONFIG,
    GPIOCC32XX_GPIO_17 | GPIO_DO_NOT_CONFIG,
}
```

Definition at line 210 of file [CC3220SF_STARPORTS.c](#).

3.13.3.15 I2C_config const I2C_Config I2C_config[[CC3220SF_STARPORTS_I2CCOUNT](#)]**Initial value:**

```
= {
    {
        .fxnTablePtr = &I2CCC32XX_fxnTable,
        .object = &I2CC3220SObjects\[CC3220SF\_STARPORTS\_I2C0\],
        .hwAttrs = &I2CCC3220SHWAttrs\[CC3220SF\_STARPORTS\_I2C0\]
    }
}
```

Definition at line 272 of file [CC3220SF_STARPORTS.c](#).

3.13.3.16 I2C_count const uint_least8_t I2C_count = [CC3220SF_STARPORTS_I2CCOUNT](#)

Definition at line 280 of file [CC3220SF_STARPORTS.c](#).

3.13.3.17 i2cCC3220SHWAttrs const I2CCC32XX_HWAttrsV1 i2cCC3220SHWAttrs[[CC3220SF_STARPORTS_I2CCOUNT](#)]**Initial value:**

```
= {
    {
        .baseAddr = I2CA0_BASE,
        .intNum = INT_I2CA0,
        .intPriority = (~0),
        .clkPin = I2CCC32XX_PIN_01_I2C_SCL,
        .dataPin = I2CCC32XX_PIN_02_I2C_SDA
    }
}
```

Definition at line 262 of file [CC3220SF_STARPORTS.c](#).

3.13.3.18 i2cCC3220SObjects `i2CCC32XX_Object i2cCC3220SObjects[CC3220SF_STARPORTS_I2CCOUNT]`

Definition at line 260 of file [CC3220SF_STARPORTS.c](#).

3.13.3.19 parkInfo `PowerCC32XX_ParkInfo parkInfo[]`

Definition at line 294 of file [CC3220SF_STARPORTS.c](#).

3.13.3.20 PowerCC32XX_config `const PowerCC32XX_ConfigV1 PowerCC32XX_config`**Initial value:**

```
= {  
    .policyInitFxn = &PowerCC32XX_initPolicy,  
    .policyFxn = &PowerCC32XX_sleepPolicy,  
    .enterLPDSHookFxn = NULL,  
    .resumeLPDSHookFxn = NULL,  
    .enablePolicy = true,  
    .enableGPIOWakeUpLPDS = true,  
    .enableGPIOWakeUpShutdown = true,  
    .enableNetworkWakeUpLPDS = true,  
    .wakeUpGPIOSourceLPDS = PRCM_LPDS_GPIO2,  
    .wakeUpGPIOTypeLPDS = PRCM_LPDS_FALL_EDGE,  
    .wakeUpGPIOFnLPDS = &WakeUpFromLPDS,  
    .wakeUpGPIOFnLPDSArg = 0,  
    .wakeUpGPIOSourceShutdown = PRCM_HIB_GPIO2,  
    .wakeUpGPIOTypeShutdown = PRCM_HIB_FALL_EDGE,  
    .ramRetentionMaskLPDS = PRCM_SRAM_COL_1 | PRCM_SRAM_COL_2 |  
        PRCM_SRAM_COL_3 | PRCM_SRAM_COL_4,  
    .keepDebugActiveDuringLPDS = true,  
    .ioRetentionShutdown = PRCM_IO_RET_GRP_1,  
    .pinParkDefs = parkInfo,  
    .numPins = sizeof(parkInfo) / sizeof(PowerCC32XX_ParkInfo)  
}
```

Definition at line 341 of file [CC3220SF_STARPORTS.c](#).

3.13.3.21 PWM_config `const PWM_Config PWM_config[CC3220SF_STARPORTS_PWMCOUNT]`**Initial value:**

```
= {  
    {  
        .fxnTablePtr = &PWMTimerCC32XX_fxnTable,  
        .object = &pwmTimerCC3220SObjects\[CC3220SF\_STARPORTS\_PWM5\],  
        .hwAttrs = &pwmTimerCC3220SHWAttrs\[CC3220SF\_STARPORTS\_PWM5\]  
    }  
}
```

Definition at line 378 of file [CC3220SF_STARPORTS.c](#).

3.13.3.22 PWM_count `const uint_least8_t PWM_count = CC3220SF_STARPORTS_PWMCOUNT`

Definition at line 386 of file [CC3220SF_STARPORTS.c](#).

3.13.3.23 pwmTimerCC3220SHWAttrs const PWMTimerCC32XX_HWAttrsV2 pwmTimerCC3220SHWAttrs[[CC3220SF_STARPORTS_PWMCOUNT](#)]

Initial value:

```
= {
    {
        .pwmPin = PWMTimerCC32XX_PIN_64
    }
}
```

Definition at line [372](#) of file [CC3220SF_STARPORTS.c](#).

3.13.3.24 pwmTimerCC3220SObjects PWMTimerCC32XX_Object pwmTimerCC3220SObjects[[CC3220SF_STARPORTS_PWMCOUNT](#)]

Definition at line [370](#) of file [CC3220SF_STARPORTS.c](#).

3.13.3.25 SPI_config const SPI_Config SPI_config[[CC3220SF_STARPORTS_SPICOUNT](#)]

Initial value:

```
= {
    {
        .fxnTablePtr = &SPIICC32XXDMA_fxnTable,
        .object = &spiICC3220SDMAObjects\[CC3220SF\_STARPORTS\_SPI0\],
        .hwAttrs = &spiICC3220SDMAHWAAttrs\[CC3220SF\_STARPORTS\_SPI0\]
    },
    {
        .fxnTablePtr = &SPIICC32XXDMA_fxnTable,
        .object = &spiICC3220SDMAObjects\[CC3220SF\_STARPORTS\_SPI1\],
        .hwAttrs = &spiICC3220SDMAHWAAttrs\[CC3220SF\_STARPORTS\_SPI1\]
    }
}
```

Definition at line [501](#) of file [CC3220SF_STARPORTS.c](#).

3.13.3.26 SPI_count const uint_least8_t SPI_count = [CC3220SF_STARPORTS_SPICOUNT](#)

Definition at line [514](#) of file [CC3220SF_STARPORTS.c](#).

3.13.3.27 spiICC3220SDMAHWAAttrs const SPIICC32XXDMA_HWAttrsV1 spiICC3220SDMAHWAAttrs[[CC3220SF_STARPORTS_SPICOUNT](#)]

Definition at line [459](#) of file [CC3220SF_STARPORTS.c](#).

3.13.3.28 spiICC3220SDMAObjects SPIICC32XXDMA_Object spiICC3220SDMAObjects[[CC3220SF_STARPORTS_SPICOUNT](#)]

Definition at line [455](#) of file [CC3220SF_STARPORTS.c](#).

3.13.3.29 spiCC3220SDMAscratchBuf `uint32_t spiCC3220SDMAscratchBuf[CC3220SF_STARPORTS_SPICOUNT]`

Definition at line 457 of file [CC3220SF_STARPORTS.c](#).

3.13.3.30 Timer_config `const Timer_Config Timer_config[CC3220SF_STARPORTS_TIMERCOUNT]`**Initial value:**

```
= {
    {
        .fxnTablePtr = &TimerCC32XX_fxnTable,
        .object = &timerCC3220SFObjets[CC3220SF_STARPORTS_TIMER0],
        .hwAttrs = &timerCC3220SFHWAtrs[CC3220SF_STARPORTS_TIMER0]
    },
    {
        .fxnTablePtr = &TimerCC32XX_fxnTable,
        .object = &timerCC3220SFObjets[CC3220SF_STARPORTS_TIMER1],
        .hwAttrs = &timerCC3220SFHWAtrs[CC3220SF_STARPORTS_TIMER1]
    },
    {
        .fxnTablePtr = &TimerCC32XX_fxnTable,
        .object = &timerCC3220SFObjets[CC3220SF_STARPORTS_TIMER2],
        .hwAttrs = &timerCC3220SFHWAtrs[CC3220SF_STARPORTS_TIMER2]
    },
}
```

Definition at line 545 of file [CC3220SF_STARPORTS.c](#).

3.13.3.31 Timer_count `const uint_least8_t Timer_count = CC3220SF_STARPORTS_TIMERCOUNT`

Definition at line 563 of file [CC3220SF_STARPORTS.c](#).

3.13.3.32 timerCC3220SFHWAtrs `const TimerCC32XX_HWAtrs timerCC3220SFHWAtrs[CC3220SF_STARPORTS_TIMERCOUNT]`**Initial value:**

```
= {
    {
        .baseAddress = TIMERA0_BASE,
        .subTimer = TimerCC32XX_timer32,
        .intNum = INT_TIMERA0A,
        .intPriority = ~0
    },
    {
        .baseAddress = TIMERA1_BASE,
        .subTimer = TimerCC32XX_timer16A,
        .intNum = INT_TIMERA1A,
        .intPriority = ~0
    },
    {
        .baseAddress = TIMERA1_BASE,
        .subTimer = TimerCC32XX_timer16B,
        .intNum = INT_TIMERA1B,
        .intPriority = ~0
    },
}
```

Definition at line 524 of file [CC3220SF_STARPORTS.c](#).

3.13.3.33 timerCC3220SFObjects TimerCC32XX_Object timerCC3220SFObjects[CC3220SF_STARPORTS_TIMERCOUNT]

Definition at line 522 of file [CC3220SF_STARPORTS.c](#).

3.13.3.34 UART_config const UART_Config UART_config[CC3220SF_STARPORTS_UARTCOUNT]**Initial value:**

```
= {
    {
        .fxnTablePtr = &UARTCC32XX_fxnTable,
        .object = &uartCC3220SObjects[CC3220SF_STARPORTS_UART0],
        .hwAttrs = &uartCC3220SHWAttrs[CC3220SF_STARPORTS_UART0]
    },
    {
        .fxnTablePtr = &UARTCC32XX_fxnTable,
        .object = &uartCC3220SObjects[CC3220SF_STARPORTS_UART1],
        .hwAttrs = &uartCC3220SHWAttrs[CC3220SF_STARPORTS_UART1]
    }
}
```

Definition at line 653 of file [CC3220SF_STARPORTS.c](#).

3.13.3.35 UART_count const uint_least8_t UART_count = CC3220SF_STARPORTS_UARTCOUNT

Definition at line 667 of file [CC3220SF_STARPORTS.c](#).

3.13.3.36 uartCC3220SHWAttrs const UARTCC32XX_HWAttrsV1 uartCC3220SHWAttrs[CC3220SF_STARPORTS_UARTCOUNT]**Initial value:**

```
= {
    {
        .baseAddr = UARTA0_BASE,
        .intNum = INT_UARTA0,
        .intPriority = (~0),
        .flowControl = UARTCC32XX_FLOWCTRL_NONE,
        .ringBufPtr = uartCC3220SRingBuffer[CC3220SF_STARPORTS_UART0],
        .ringBufSize = sizeof(uartCC3220SRingBuffer[CC3220SF_STARPORTS_UART0]),
        .rxPin = UARTCC32XX_PIN_57_UART0_RX,
        .txPin = UARTCC32XX_PIN_55_UART0_TX,
        .ctsPin = UARTCC32XX_PIN_UNASSIGNED,
        .rtsPin = UARTCC32XX_PIN_UNASSIGNED,
        .errorFxn = NULL
    },
    {
        .baseAddr = UARTA1_BASE,
        .intNum = INT_UARTA1,
        .intPriority = (~0),
        .flowControl = UARTCC32XX_FLOWCTRL_NONE,
        .ringBufPtr = uartCC3220SRingBuffer[CC3220SF_STARPORTS_UART1],
        .ringBufSize = sizeof(uartCC3220SRingBuffer[CC3220SF_STARPORTS_UART1]),
        .rxPin = UARTCC32XX_PIN_59_UART1_RX,
        .txPin = UARTCC32XX_PIN_58_UART1_TX,
        .ctsPin = UARTCC32XX_PIN_UNASSIGNED,
        .rtsPin = UARTCC32XX_PIN_UNASSIGNED,
        .errorFxn = NULL
    }
}
```

Definition at line 624 of file [CC3220SF_STARPORTS.c](#).

3.13.3.37 uartCC3220SObjects `UARTCC32XX_Object uartCC3220SObjects[CC3220SF_STARPORTS_UARTCOUNT]`

Definition at line 620 of file [CC3220SF_STARPORTS.c](#).

3.13.3.38 uartCC3220SRingBuffer `unsigned char uartCC3220SRingBuffer[CC3220SF_STARPORTS_UARTCOUNT] [32]`

Definition at line 621 of file [CC3220SF_STARPORTS.c](#).

3.13.3.39 udmaCC3220SHWAttrs `const UDMACC32XX_HWAttrs udmaCC3220SHWAttrs`**Initial value:**

```
= {  
    .controlBaseAddr = (void *)dmaControlTable,  
    .dmaErrorFxn = (UDMACC32XX_ErrorFxn)dmaErrorFxn,  
    .intNum = INT_UDMAERR,  
    .intPriority = (~0)  
}
```

Definition at line 172 of file [CC3220SF_STARPORTS.c](#).

3.13.3.40 udmaCC3220SObject `UDMACC32XX_Object udmaCC3220SObject`

Definition at line 170 of file [CC3220SF_STARPORTS.c](#).

3.13.3.41 UDMACC32XX_config `const UDMACC32XX_Config UDMACC32XX_config`**Initial value:**

```
= {  
    .object = &udmaCC3220SObject,  
    .hwAttrs = &udmaCC3220SHWAttrs  
}
```

Definition at line 179 of file [CC3220SF_STARPORTS.c](#).

3.13.3.42 Watchdog_config `const Watchdog_Config Watchdog_config[CC3220SF_STARPORTS_WATCHDOGCOUNT]`**Initial value:**

```
= {  
    {  
        .fxnTablePtr = &WatchdogCC32XX_fxnTable,  
        .object = &watchdogCC3220SObjects[CC3220SF_STARPORTS_WATCHDOG0],  
        .hwAttrs = &watchdogCC3220SHWAttrs[CC3220SF_STARPORTS_WATCHDOG0]  
    }  
}
```

Definition at line 686 of file [CC3220SF_STARPORTS.c](#).

3.13.3.43 Watchdog_count const uint_least8_t Watchdog_count = CC3220SF_STARPORTS_WATCHDOGCOUNT

Definition at line 694 of file [CC3220SF_STARPORTS.c](#).

3.13.3.44 watchdogCC3220SHWAttrs const WatchdogCC32XX_HWAttrs watchdogCC3220SHWAttrs[CC3220SF_STARPORTS_WATCHDOGCOUNT]**Initial value:**

```
= {
    {
        .baseAddr = WDT_BASE,
        .intNum = INT_WDT,
        .intPriority = (~0),
        .reloadValue = 80000000
    }
}
```

Definition at line 677 of file [CC3220SF_STARPORTS.c](#).

3.13.3.45 watchdogCC3220SObjects WatchdogCC32XX_Object watchdogCC3220SObjects[CC3220SF_STARPORTS_WATCHDOGCOUNT]

Definition at line 675 of file [CC3220SF_STARPORTS.c](#).

3.14 CC3220SF_STARPORTS.c

```
00001 /*
00002 * Copyright (c) 2016-2018, Texas Instruments Incorporated
00003 * All rights reserved.
00004 *
00005 * Redistribution and use in source and binary forms, with or without
00006 * modification, are permitted provided that the following conditions
00007 * are met:
00008 *
00009 * * Redistributions of source code must retain the above copyright
00010 * notice, this list of conditions and the following disclaimer.
00011 *
00012 * * Redistributions in binary form must reproduce the above copyright
00013 * notice, this list of conditions and the following disclaimer in the
00014 * documentation and/or other materials provided with the distribution.
00015 *
00016 * * Neither the name of Texas Instruments Incorporated nor the names of
00017 * its contributors may be used to endorse or promote products derived
00018 * from this software without specific prior written permission.
00019 *
00020 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
00021 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
00022 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
00023 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
00024 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
00025 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
00026 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
00027 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
00028 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
00029 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
00030 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00031 */
00032
00033 /*
00034 * ===== CC3220SF_STARPORTS.c =====
00035 * This file is responsible for setting up the board specific items for the
00036 * CC3220SF_STARPORTS board.
00037 */
00038
00039 #include <stdbool.h>
00040 #include <stddef.h>
00041 #include <stdint.h>
00042
00043 #include <ti/devices/cc32xx/inc/hw_ints.h>
00044 #include <ti/devices/cc32xx/inc/hw_memmap.h>
00045 #include <ti/devices/cc32xx/inc/hw_types.h>
```

```

00046 #include <ti/devices/cc32xx/driverlib/rom.h>
00048 #include <ti/devices/cc32xx/driverlib/rom_map.h>
00049 #include <ti/devices/cc32xx/driverlibadc.h>
00050 #include <ti/devices/cc32xx/driverlib/gpio.h>
00051 #include <ti/devices/cc32xx/driverlib/pin.h>
00052 #include <ti/devices/cc32xx/driverlib/prcm.h>
00053 #include <ti/devices/cc32xx/driverlib/spi.h>
00054 #include <ti/devices/cc32xx/driverlib/sdhost.h>
00055 #include <ti/devices/cc32xx/driverlib/timer.h>
00056 #include <ti/devices/cc32xx/driverlib/uart.h>
00057 #include <ti/devices/cc32xx/driverlib/udma.h>
00058 #include <ti/devices/cc32xx/driverlib/wdt.h>
00059
00060 #include <ti/drivers/Power.h>
00061 #include <ti/drivers/power/PowerCC32XX.h>
00062
00063 #include "CC3220SF_STARPORTS.h"
00064
00065 /*
00066 * This define determines whether to use the UARTCC32XXDMA driver
00067 * or the UARTCC32XX (no DMA) driver. Set to 1 to use the UARTCC32XXDMA
00068 * driver.
00069 */
00070 #ifndef TI_DRIVERS_UART_DMA
00071 #define TI_DRIVERS_UART_DMA 0
00072 #endif
00073
00074 /*
00075 * ===== ADC =====
00076 */
00077 #include <ti/drivers/ADC.h>
00078 #include <ti/drivers/adc/ADCCC32XX.h>
00079
00080 ADCCC32XX_Object adcCC3220SObjects[CC3220SF_STARPORTS_ADCCOUNT];
00081
00082 const ADCCC32XX_HWAttrsV1 adcCC3220SHWAttrs[CC3220SF_STARPORTS_ADCCOUNT] = {
00083 {
00084 .adcPin = ADCCC32XX_PIN_60_CH_3
00085 }
00086 };
00087
00088 const ADC_Config ADC_config[CC3220SF_STARPORTS_ADCCOUNT] = {
00089 {
00090 .fxnTablePtr = &ADCCC32XX_fxnTable,
00091 .object = &adcCC3220SObjects[CC3220SF_STARPORTS_ADC0],
00092 .hwAttrs = &adcCC3220SHWAttrs[CC3220SF_STARPORTS_ADC0]
00093 }
00094 };
00095
00096 const uint_least8_t ADC_count = CC3220SF_STARPORTS_ADCCOUNT;
00097
00098 /*
00099 * ===== Capture =====
00100 */
00101 #include <ti/drivers/Capture.h>
00102 #include <ti/drivers/capture/CaptureCC32XX.h>
00103
00104 CaptureCC32XX_Object captureCC3220SFObjets[CC3220SF_STARPORTS_CAPTURECOUNT];
00105
00106 const CaptureCC32XX_HWAttrs captureCC3220SFHWAAttrs[CC3220SF_STARPORTS_CAPTURECOUNT] = {
00107 {
00108 {
00109 .capturePin = CaptureCC32XX_PIN_04,
00110 .intPriority = ~0
00111 },
00112 {
00113 .capturePin = CaptureCC32XX_PIN_05,
00114 .intPriority = ~0
00115 },
00116 },
00117
00118 const Capture_Config Capture_config[CC3220SF_STARPORTS_CAPTURECOUNT] = {
00119 {
00120 .fxnTablePtr = &CaptureCC32XX_fxnTable,
00121 .object = &captureCC3220SFObjets[CC3220SF_STARPORTS_CAPTURE0],
00122 .hwAttrs = &captureCC3220SFHWAAttrs[CC3220SF_STARPORTS_CAPTURE0]
00123 },
00124 {
00125 .fxnTablePtr = &CaptureCC32XX_fxnTable,
00126 .object = &captureCC3220SFObjets[CC3220SF_STARPORTS_CAPTURE1],
00127 .hwAttrs = &captureCC3220SFHWAAttrs[CC3220SF_STARPORTS_CAPTURE1]
00128 }
00129 };
00130
00131 const uint_least8_t Capture_count = CC3220SF_STARPORTS_CAPTURECOUNT;
00132

```

```

00133 /*
00134 * ====== Crypto ======
00135 */
00136 #include <ti/drivers/crypto/CryptoCC32XX.h>
00137
00138 CryptoCC32XX_Object cryptoCC3220SObjects[CC3220SF_STARPORTS_CRYPTOCOUNT];
00139
00140 const CryptoCC32XX_Config CryptoCC32XX_config[CC3220SF_STARPORTS_CRYPTOCOUNT] = {
00141     {
00142         .object = &cryptoCC3220SObjects[CC3220SF_STARPORTS_CRYPTO0]
00143     }
00144 };
00145
00146 const uint_least8_t CryptoCC32XX_count = CC3220SF_STARPORTS_CRYPTOCOUNT;
00147
00148 /*
00149 * ====== DMA ======
00150 */
00151 #include <ti/drivers/dma/UDMACC32XX.h>
00152
00153 static tDMAControlTable dmaControlTable[64] __attribute__ ((aligned (1024)));
00154
00155 /*
00156 * ====== dmaErrorFxn ======
00157 * This is the handler for the uDMA error interrupt.
00158 */
00159 static void dmaErrorFxn(uintptr_t arg)
00160 {
00161     int status = MAP_uDMAErrorStatusGet ();
00162     MAP_uDMAErrorStatusClear ();
00163
00164     /* Suppress unused variable warning */
00165     (void)status;
00166
00167     while (1);
00168 }
00169
00170 UDMACC32XX_Object udmaCC3220SObject;
00171
00172 const UDMACC32XX_HWAttrs udmaCC3220SHWAttrs = {
00173     .controlBaseAddr = (void *)dmaControlTable,
00174     .dmaErrorFxn = (UDMACC32XX_ErrorFxn)dmaErrorFxn,
00175     .intNum = INT_UDMAERR,
00176     .intPriority = (~0)
00177 };
00178
00179 const UDMACC32XX_Config UDMACC32XX_config = {
00180     .object = &udmaCC3220SObject,
00181     .hwAttrs = &udmaCC3220SHWAttrs
00182 };
00183
00184 /*
00185 * ====== General ======
00186 */
00187 /*
00188 * ====== CC3220SF_STARPORTS_initGeneral ======
00189 */
00190 void CC3220SF_STARPORTS_initGeneral(void)
00191 {
00192     PRCMCC3200MCUInit();
00193     Power_init();
00194 }
00195
00196 /*
00197 * ====== GPIO ======
00198 */
00199 #include <ti/drivers/GPIO.h>
00200 #include <ti/drivers/gpio/GPIOPCC32XX.h>
00201
00202 /*
00203 * Array of Pin configurations
00204 * NOTE: The order of the pin configurations must coincide with what was
00205 * defined in CC3220SF_STARPORTS.h
00206 * NOTE: Pins not used for interrupts should be placed at the end of the
00207 * array. Callback entries can be omitted from callbacks array to
00208 * reduce memory usage.
00209 */
00210 GPIO_PinConfig gpioPinConfigs[] = {
00211
00212     GPIOCC32XX_GPIO_30 | GPIO_DO_NOT_CONFIG, /* RN2483 /MCLR */
00213
00214     /* STARPORTS GPIO Pins */
00215     GPIOCC32XX_GPIO_12 | GPIO_DO_NOT_CONFIG, /* ADXL355_CS */
00216     GPIOCC32XX_GPIO_13 | GPIO_DO_NOT_CONFIG, /* BME280_CS */
00217     GPIOCC32XX_GPIO_22 | GPIO_DO_NOT_CONFIG, /* LDC1000_CS */
00218     GPIOCC32XX_GPIO_08 | GPIO_DO_NOT_CONFIG, /* EN_NODE */
00219     GPIOCC32XX_GPIO_06 | GPIO_DO_NOT_CONFIG, /* EN_ADXL355 */

```

```

00220     GPIOCC32XX_GPIO_00 | GPIO_DO_NOT_CONFIG, /* EN_BME280 */
00221     GPIOCC32XX_GPIO_28 | GPIO_DO_NOT_CONFIG, /* EN_LDC1000 */
00222     GPIOCC32XX_GPIO_07 | GPIO_DO_NOT_CONFIG, /* DS1374_INTB */
00223     GPIOCC32XX_GPIO_10 | GPIO_DO_NOT_CONFIG, /* SCL */
00224     GPIOCC32XX_GPIO_11 | GPIO_DO_NOT_CONFIG, /* SDA */
00225     GPIOCC32XX_GPIO_14 | GPIO_DO_NOT_CONFIG, /* SCLK */
00226     GPIOCC32XX_GPIO_16 | GPIO_DO_NOT_CONFIG, /* MOSI */
00227     GPIOCC32XX_GPIO_17 | GPIO_DO_NOT_CONFIG, /* CS */
00228 }
00229 */
00230 */
00231 /*
00232 * Array of callback function pointers
00233 * NOTE: The order of the pin configurations must coincide with what was
00234 * defined in CC3220SF_STARPORTS.h
00235 * NOTE: Pins not used for interrupts can be omitted from callbacks array to
00236 * reduce memory usage (if placed at end of gpioPinConfigs array).
00237 */
00238 GPIO_CallbackFxn gpioCallbackFunctions[] = {
00239     NULL, /* CC3220SF_STARPORTS_GPIO_SW2 */
00240     NULL, /* CC3220SF_STARPORTS_GPIO_SW3 */
00241     NULL, /* CC3220SF_STARPORTS_SPI_MASTER_READY */
00242     NULL /* CC3220SF_STARPORTS_SPI_SLAVE_READY */
00243 };
00244
00245 /* The device-specific GPIO_config structure */
00246 const GPIOCC32XX_Config GPIOCC32XX_config = {
00247     .pinConfigs = (GPIO_PinConfig *)gpioPinConfigs,
00248     .callbacks = (GPIO_CallbackFxn *)gpioCallbackFunctions,
00249     .numberOfPinConfigs = sizeof(gpioPinConfigs)/sizeof(GPIO_PinConfig),
00250     .numberOfCallbacks = sizeof(gpioCallbackFunctions)/sizeof(GPIO_CallbackFxn),
00251     .intPriority = (~0)
00252 };
00253
00254
00255 /* ===== I2C =====
00256 */
00257 #include <ti/drivers/I2C.h>
00258 #include <ti/drivers/i2c/I2CCC32XX.h>
00259
00260 I2CCC32XX_Object i2cCC3220SObjects[CC3220SF_STARPORTS_I2CCOUNT];
00261
00262 const I2CCC32XX_HWAttrsV1 i2cCC3220SHWAttrs[CC3220SF_STARPORTS_I2CCOUNT] = {
00263     {
00264         .baseAddr = I2CA0_BASE,
00265         .intNum = INT_I2CA0,
00266         .intPriority = (~0),
00267         .clkPin = I2CCC32XX_PIN_01_I2C_SCL,
00268         .dataPin = I2CCC32XX_PIN_02_I2C_SDA
00269     }
00270 };
00271
00272 const I2C_Config I2C_config[CC3220SF_STARPORTS_I2CCOUNT] = {
00273     {
00274         .fxnTablePtr = &I2CCC32XX_fxnTable,
00275         .object = &i2cCC3220SObjects[CC3220SF_STARPORTS_I2C0],
00276         .hwAttrs = &i2cCC3220SHWAttrs[CC3220SF_STARPORTS_I2C0]
00277     }
00278 };
00279
00280 const uint_least8_t I2C_count = CC3220SF_STARPORTS_I2CCOUNT;
00281
00282 /*
00283 * ===== Power =====
00284 */
00285 /*
00286 * This table defines the parking state to be set for each parkable pin
00287 * during LPDS. (Device pins must be parked during LPDS to achieve maximum
00288 * power savings.) If the pin should be left unparked, specify the state
00289 * PowerCC32XX_DONT_PARK. For example, for a UART TX pin, the device
00290 * will automatically park the pin in a high state during transition to LPDS,
00291 * so the Power Manager does not need to explicitly park the pin. So the
00292 * corresponding entries in this table should indicate PowerCC32XX_DONT_PARK.
00293 */
00294 PowerCC32XX_ParkInfo parkInfo[] = {
00295     /* PIN                      PARK STATE          PIN ALIAS (FUNCTION) */
00296     -----
00297     {PowerCC32XX_PIN01, PowerCC32XX_DRIVE_LOW}, /* I2C_SCL           */
00298     {PowerCC32XX_PIN02, PowerCC32XX_DRIVE_LOW}, /* I2C_SDA           */
00299     {PowerCC32XX_PIN03, PowerCC32XX_DRIVE_LOW}, /* SPI_CS_ADXL355   */
00300     {PowerCC32XX_PIN04, PowerCC32XX_DRIVE_LOW}, /* SPI_CS_BME280    */
00301     {PowerCC32XX_PIN05, PowerCC32XX_DRIVE_LOW}, /* SPI_SCLK          */
00302     {PowerCC32XX_PIN06, PowerCC32XX_DRIVE_LOW}, /* SPI_MISO          */
00303     {PowerCC32XX_PIN07, PowerCC32XX_DRIVE_LOW}, /* SPI_MOSI          */
00304     {PowerCC32XX_PIN08, PowerCC32XX_DRIVE_LOW}, /* SPI_CS            */
00305     {PowerCC32XX_PIN13, PowerCC32XX_WEAK_PULL_DOWN_STD}, /* FLASH_SPI_DIN    */
00306     {PowerCC32XX_PIN15, PowerCC32XX_DRIVE_LOW}, /* SPI_CS_LDC1000   */

```

```

00307 {PowerCC32XX_PIN16, PowerCC32XX_WEAK_PULL_DOWN_STD}, /* TDI (JTAG DEBUG) */
00308 {PowerCC32XX_PIN17, PowerCC32XX_WEAK_PULL_DOWN_STD}, /* TDO (JTAG DEBUG) */
00309 {PowerCC32XX_PIN19, PowerCC32XX_WEAK_PULL_DOWN_STD}, /* TCK (JTAG DEBUG) */
00310 {PowerCC32XX_PIN20, PowerCC32XX_WEAK_PULL_DOWN_STD}, /* TMS (JTAG DEBUG) */
00311 {PowerCC32XX_PIN18, PowerCC32XX_DRIVE_LOW}, /* EN_LDC1000 */ */
00312 {PowerCC32XX_PIN21, PowerCC32XX_WEAK_PULL_DOWN_STD}, /* SOP2 */
00313 {PowerCC32XX_PIN29, PowerCC32XX_WEAK_PULL_DOWN_STD}, /* ANTSEL1 */
00314 {PowerCC32XX_PIN30, PowerCC32XX_WEAK_PULL_DOWN_STD}, /* ANTSEL2 */
00315 {PowerCC32XX_PIN45, PowerCC32XX_WEAK_PULL_DOWN_STD}, /* DCDC_ANA2_SW_P */
00316 {PowerCC32XX_PIN50, PowerCC32XX_DRIVE_LOW}, /* EN_BME280 */ */
00317 {PowerCC32XX_PIN52, PowerCC32XX_WEAK_PULL_DOWN_STD}, /* RTC_XTAL_N */
00318 {PowerCC32XX_PIN53, PowerCC32XX_DRIVE_LOW}, /* RN2483_MCLR */
00319 {PowerCC32XX_PIN55, PowerCC32XX_DRIVE_LOW}, /* XDS_UART_RX */
00320 {PowerCC32XX_PIN57, PowerCC32XX_DRIVE_LOW}, /* XDS_UART_TX */
00321 {PowerCC32XX_PIN58, PowerCC32XX_DRIVE_LOW}, /* UART_TX */
00322 {PowerCC32XX_PIN59, PowerCC32XX_DRIVE_LOW}, /* UART_RX */
00323 {PowerCC32XX_PIN60, PowerCC32XX_WEAK_PULL_DOWN_STD}, /* ADCIN */
00324 {PowerCC32XX_PIN61, PowerCC32XX_DRIVE_LOW}, /* EN_AXDL355 */
00325 {PowerCC32XX_PIN62, PowerCC32XX_WEAK_PULL_DOWN_STD}, /* INTB_RTC */
00326 {PowerCC32XX_PIN63, PowerCC32XX_DRIVE_LOW}, /* EN_NODE */
00327 {PowerCC32XX_PIN64, PowerCC32XX_DRIVE_LOW}, /* LDC1000_CLK */
00328 };
00329 */
00330 */
00331 * This structure defines the configuration for the Power Manager.
00332 */
00333 * In this configuration the Power policy is disabled by default (because
00334 * enablePolicy is set to false). The Power policy can be enabled dynamically
00335 * at runtime by calling Power_enablePolicy(), or at build time, by changing
00336 * enablePolicy to true in this structure.
00337 */
00338
00339 void WakeupFromLPDS(uint_least8_t);
00340
00341 const PowerCC32XX_ConfigV1 PowerCC32XX_config = {
00342 .policyInitFxn = &PowerCC32XX_initPolicy,
00343 .policyFxn = &PowerCC32XX_sleepPolicy,
00344 .enterLPDShookFxn = NULL,
00345 .resumeLPDShookFxn = NULL,
00346 .enablePolicy =
00339

00339
sizeofg0G[((uint_le01rg001RG[-600(PoFparkInfo9)]TJ0.380.25J83.)TJ29.29b02TJ0-7.99Eo0-7.971Td[(sizeofg0G[((uint_le32XX_PIN64,)
```

```

00394 */
00395 /*
00396 * Note: The SDFatFS driver provides interface functions to enable FatFs
00397 * but relies on the SD driver to communicate with SD cards. Opening a
00398 * SDFatFs driver instance will internally try to open a SD driver instance
00399 * reusing the same index number (opening SDFatFs driver at index 0 will try to
00400 * open SD driver at index 0). This requires that all SDFatFs driver instances
00401 * have an accompanying SD driver instance defined with the same index. It is
00402 * acceptable to have more SD driver instances than SDFatFs driver instances
00403 * but the opposite is not supported & the SDFatFs will fail to open.
00404 */
00405 /*SDFatFS_Object sdfatfsObjects[CC3220SF_STARPORTS_SDFatFSCOUNT];
00406
00407 const SDFatFS_Config SDFatFS_config[CC3220SF_STARPORTS_SDFatFSCOUNT] = {
00408     {
00409         .object = &sdfatfsObjects[CC3220SF_STARPORTS_SDFatFS0]
00410     }
00411 };
00412
00413 const uint_least8_t SDFatFS_count = CC3220SF_STARPORTS_SDFatFSCOUNT;*/
00414
00415 /*
00416 * ===== SD =====
00417 */
00418 /*
00419 #include <ti/drivers/SD.h>
00420 #include <ti/drivers/sd/SDHostCC32XX.h>
00421
00422 SDHostCC32XX_Object sdhostCC3220SObjects[CC3220SF_STARPORTS_SDCOUNT];
00423
00424 SDHost configuration structure, describing which pins are to be used
00425 const SDHostCC32XX_HWAttrsV1 sdhostCC3220SHWAttrs[CC3220SF_STARPORTS_SDCOUNT] = {
00426     {
00427         .clkRate = 8000000,
00428         .intPriority = ~0,
00429         .baseAddr = SDHOST_BASE,
00430         .rxChIdx = UDMA_CH23_SDHOST_RX,
00431         .txChIdx = UDMA_CH24_SDHOST_TX,
00432         .dataPin = SDHostCC32XX_PIN_06_SDCARD_DATA,
00433         .cmdPin = SDHostCC32XX_PIN_08_SDCARD_CMD,
00434         .clkPin = SDHostCC32XX_PIN_07_SDCARD_CLK
00435     }
00436 };
00437
00438 const SD_Config SD_config[CC3220SF_STARPORTS_SDCOUNT] = {
00439     {
00440         .fxnTablePtr = &sdHostCC32XX_fxnTable,
00441         .object = &sdhostCC3220SObjects[CC3220SF_STARPORTS_SD0],
00442         .hwAttrs = &sdhostCC3220SHWAttrs[CC3220SF_STARPORTS_SD0]
00443     },
00444 };
00445
00446 const uint_least8_t SD_count = CC3220SF_STARPORTS_SDCOUNT;
00447 */
00448
00449 /*
00450 * ===== SPI =====
00451 */
00452 #include <ti/drivers/SPI.h>
00453 #include <ti/drivers/spi/SPICC32XXDMA.h>
00454
00455 SPICC32XXDMA_Object spiCC3220SDMAObjects[CC3220SF_STARPORTS_SPICOUNT];
00456
00457 uint32_t spiCC3220SDMAscratchBuf[CC3220SF_STARPORTS_SPICOUNT];
00458
00459 const SPICC32XXDMA_HWAttrsV1 spiCC3220SDMAHWAttrs[CC3220SF_STARPORTS_SPICOUNT] = {
00460     /* index 0 is reserved for LSPI that links to the NWP */
00461     {
00462         .baseAddr = LSPI_BASE,
00463         .intNum = INT_LSPI,
00464         .intPriority = (~0),
00465         .spiPRCM = PRCM_LSPI,
00466         .csControl = SPI_SW_CTRL_CS,
00467         .csPolarity = SPI_CS_ACTIVEHIGH,
00468         .pinMode = SPI_4PIN_MODE,
00469         .turboMode = SPI_TURBO_OFF,
00470         .scratchBufPtr = &spiCC3220SDMAscratchBuf[CC3220SF_STARPORTS_SPI0],
00471         .defaultTxBufValue = 0,
00472         .rxChannelIndex = UDMA_CH12_LSPI_RX,
00473         .txChannelIndex = UDMA_CH13_LSPI_TX,
00474         .minDmaTransferSize = 100,
00475         .mosiPin = SPIICC32XXDMA_PIN_NO_CONFIG,
00476         .misoPin = SPIICC32XXDMA_PIN_NO_CONFIG,
00477         .clkPin = SPIICC32XXDMA_PIN_NO_CONFIG,
00478         .csPin = SPIICC32XXDMA_PIN_NO_CONFIG
00479     },
00480     {

```

```

00481     .baseAddr = GSPI_BASE,
00482     .intNum = INT_GSPI,
00483     .intPriority = (~0),
00484     .spiPRCM = PRCM_GSPI,
00485     .csControl = SPI_SW_CTRL_CS,
00486     .csPolarity = SPI_CS_ACTIVELOW,
00487     .pinMode = SPI_4PIN_MODE,
00488     .turboMode = SPI_TURBO_OFF,
00489     .scratchBufPtr = &spiCC3220SDMAscratchBuf[CC3220SF_STARPORTS_SPI1],
00490     .defaultTxBufValue = 0,
00491     .rxChannelIndex = UDMA_CH6_GSPI_RX,
00492     .txChannelIndex = UDMA_CH7_GSPI_TX,
00493     .minDmaTransferSize = 10,
00494     .mosiPin = SPICC32XXDMA_PIN_07_MOSI,
00495     .misoPin = SPICC32XXDMA_PIN_06_MISO,
00496     .clkPin = SPICC32XXDMA_PIN_05_CLK,
00497     .csPin = SPICC32XXDMA_PIN_08_CS
00498   },
00499 };
00500
00501 const SPI_Config SPI_config[CC3220SF_STARPORTS_SPICOUNT] = {
00502   {
00503     .fxnTablePtr = &SPICC32XXDMA_fxnTable,
00504     .object = &spiCC3220SDMAObjects[CC3220SF_STARPORTS_SPI0],
00505     .hwAttrs = &spiCC3220SDMAHWAAttrs[CC3220SF_STARPORTS_SPI0]
00506   },
00507   {
00508     .fxnTablePtr = &SPICC32XXDMA_fxnTable,
00509     .object = &spiCC3220SDMAObjects[CC3220SF_STARPORTS_SPI1],
00510     .hwAttrs = &spiCC3220SDMAHWAAttrs[CC3220SF_STARPORTS_SPI1]
00511   }
00512 };
00513
00514 const uint_least8_t SPI_count = CC3220SF_STARPORTS_SPICOUNT;
00515
00516 /*
00517 * ===== Timer =====
00518 */
00519 #include <ti/drivers/Timer.h>
00520 #include <ti/drivers/timer/TimerCC32XX.h>
00521
00522 TimerCC32XX_Object timerCC3220SFObjects[CC3220SF_STARPORTS_TIMERCOUNT];
00523
00524 const TimerCC32XX_HWAttrs timerCC3220SFHWAAttrs[CC3220SF_STARPORTS_TIMERCOUNT] = {
00525   {
00526     .baseAddress = TIMER0A0_BASE,
00527     .subTimer = TimerCC32XX_timer32,
00528     .intNum = INT_TIMER0A0A,
00529     .intPriority = ~0
00530   },
00531   {
00532     .baseAddress = TIMER0A1_BASE,
00533     .subTimer = TimerCC32XX_timer16A,
00534     .intNum = INT_TIMER0A1A,
00535     .intPriority = ~0
00536   },
00537   {
00538     .baseAddress = TIMER0A1_BASE,
00539     .subTimer = TimerCC32XX_timer16B,
00540     .intNum = INT_TIMER0A1B,
00541     .intPriority = ~0
00542   },
00543 };
00544
00545 const Timer_Config Timer_config[CC3220SF_STARPORTS_TIMERCOUNT] = {
00546   {
00547     .fxnTablePtr = &TimerCC32XX_fxnTable,
00548     .object = &timerCC3220SFObjects[CC3220SF_STARPORTS_TIMER0],
00549     .hwAttrs = &timerCC3220SFHWAAttrs[CC3220SF_STARPORTS_TIMER0]
00550   },
00551   {
00552     .fxnTablePtr = &TimerCC32XX_fxnTable,
00553     .object = &timerCC3220SFObjects[CC3220SF_STARPORTS_TIMER1],
00554     .hwAttrs = &timerCC3220SFHWAAttrs[CC3220SF_STARPORTS_TIMER1]
00555   },
00556   {
00557     .fxnTablePtr = &TimerCC32XX_fxnTable,
00558     .object = &timerCC3220SFObjects[CC3220SF_STARPORTS_TIMER2],
00559     .hwAttrs = &timerCC3220SFHWAAttrs[CC3220SF_STARPORTS_TIMER2]
00560   },
00561 };
00562
00563 const uint_least8_t Timer_count = CC3220SF_STARPORTS_TIMERCOUNT;
00564
00565 /*
00566 * ===== UART =====
00567 */

```

```

00568 #include <ti/drivers/UART.h>
00569 #if TI_DRIVERS_UART_DMA
00570 #include <ti/drivers/uart/UARTCC32XXDMA.h>
00571
00572 UARTCC32XXDMA_Object uartCC3220SDmaObjects[CC3220SF_STARPORTS_UARTCOUNT];
00573
00574 /* UART configuration structure */
00575 const UARTCC32XXDMA_HWAttrsV1 uartCC3220SDmaHwAttrs[CC3220SF_STARPORTS_UARTCOUNT] = {
00576 {
00577     .baseAddr = UARTA0_BASE,
00578     .intNum = INT_UARTA0,
00579     .intPriority = (~0),
00580     .flowControl = UARTCC32XXDMA_FLOWCTRL_NONE,
00581     .rxChannelIndex = UDMA_CH8_UARTA0_RX,
00582     .txChannelIndex = UDMA_CH9_UARTA0_TX,
00583     .rxPin = UARTCC32XXDMA_PIN_57_UART0_RX,
00584     .txPin = UARTCC32XXDMA_PIN_55_UART0_TX,
00585     .ctsPin = UARTCC32XXDMA_PIN_UNASSIGNED,
00586     .rtsPin = UARTCC32XXDMA_PIN_UNASSIGNED,
00587     .errorFxn = NULL
00588 },
00589 {
00590     .baseAddr = UARTA1_BASE,
00591     .intNum = INT_UARTA1,
00592     .intPriority = (~0),
00593     .flowControl = UARTCC32XXDMA_FLOWCTRL_NONE,
00594     .rxChannelIndex = UDMA_CH10_UARTA1_RX,
00595     .txChannelIndex = UDMA_CH11_UARTA1_TX,
00596     .rxPin = UARTCC32XXDMA_PIN_08_UART1_RX,
00597     .txPin = UARTCC32XXDMA_PIN_07_UART1_TX,
00598     .ctsPin = UARTCC32XXDMA_PIN_UNASSIGNED,
00599     .rtsPin = UARTCC32XXDMA_PIN_UNASSIGNED,
00600     .errorFxn = NULL
00601 },
00602 };
00603
00604 const UART_Config UART_config[CC3220SF_STARPORTS_UARTCOUNT] = {
00605 {
00606     .fxnTablePtr = &UARTCC32XXDMA_fxnTable,
00607     .object = &uartCC3220SDmaObjects[CC3220SF_STARPORTS_UART0],
00608     .hwAttrs = &uartCC3220SDmaHwAttrs[CC3220SF_STARPORTS_UART0]
00609 },
00610 {
00611     .fxnTablePtr = &UARTCC32XXDMA_fxnTable,
00612     .object = &uartCC3220SDmaObjects[CC3220SF_STARPORTS_UART1],
00613     .hwAttrs = &uartCC3220SDmaHwAttrs[CC3220SF_STARPORTS_UART1]
00614 },
00615 };
00616
00617 #else
00618 #include <ti/drivers/uart/UARTCC32XX.h>
00619
00620 UARTCC32XX_Object uartCC3220SObjects[CC3220SF_STARPORTS_UARTCOUNT];
00621 unsigned char uartCC3220SRingBuffer[CC3220SF_STARPORTS_UARTCOUNT][32];
00622
00623 /* UART configuration structure */
00624 const UARTCC32XX_HWAttrsV1 uartCC3220SHWAttrs[CC3220SF_STARPORTS_UARTCOUNT] = {
00625 {
00626     .baseAddr = UARTA0_BASE,
00627     .intNum = INT_UARTA0,
00628     .intPriority = (~0),
00629     .flowControl = UARTCC32XX_FLOWCTRL_NONE,
00630     .ringBufPtr = uartCC3220SRingBuffer[CC3220SF_STARPORTS_UART0],
00631     .ringBufSize = sizeof(uartCC3220SRingBuffer[CC3220SF_STARPORTS_UART0]),
00632     .rxPin = UARTCC32XX_PIN_57_UART0_RX,
00633     .txPin = UARTCC32XX_PIN_55_UART0_TX,
00634     .ctsPin = UARTCC32XX_PIN_UNASSIGNED,
00635     .rtsPin = UARTCC32XX_PIN_UNASSIGNED,
00636     .errorFxn = NULL
00637 },
00638 {
00639     .baseAddr = UARTA1_BASE,
00640     .intNum = INT_UARTA1,
00641     .intPriority = (~0),
00642     .flowControl = UARTCC32XX_FLOWCTRL_NONE,
00643     .ringBufPtr = uartCC3220SRingBuffer[CC3220SF_STARPORTS_UART1],
00644     .ringBufSize = sizeof(uartCC3220SRingBuffer[CC3220SF_STARPORTS_UART1]),
00645     .rxPin = UARTCC32XX_PIN_59_UART1_RX,
00646     .txPin = UARTCC32XX_PIN_58_UART1_TX,
00647     .ctsPin = UARTCC32XX_PIN_UNASSIGNED,
00648     .rtsPin = UARTCC32XX_PIN_UNASSIGNED,
00649     .errorFxn = NULL
00650 },
00651 };
00652
00653 const UART_Config UART_config[CC3220SF_STARPORTS_UARTCOUNT] = {
00654 {

```

```

00655     .fxnTablePtr = &UARTCC32XX_fxnTable,
00656     .object = &uartCC3220SObjects[CC3220SF_STARPORTS_UART0],
00657     .hwAttrs = &uartCC3220SHWAttrs[CC3220SF_STARPORTS_UART0]
00658 },
00659 {
00660     .fxnTablePtr = &UARTCC32XX_fxnTable,
00661     .object = &uartCC3220SObjects[CC3220SF_STARPORTS_UART1],
00662     .hwAttrs = &uartCC3220SHWAttrs[CC3220SF_STARPORTS_UART1]
00663 }
00664 };
00665 #endif /* TI_DRIVERS_UART_DMA */
00666
00667 const uint_least8_t UART_count = CC3220SF_STARPORTS_UARTCOUNT;
00668
00669 /*
0070 * ===== Watchdog =====
0071 */
0072 #include <ti/drivers/Watchdog.h>
0073 #include <ti/drivers/watchdog/WatchdogCC32XX.h>
0074
0075 WatchdogCC32XX_Object watchdogCC3220SObjects[CC3220SF_STARPORTS_WATCHDOGCOUNT];
0076
0077 const WatchdogCC32XX_HWAttrs watchdogCC3220SHWAttrs[CC3220SF_STARPORTS_WATCHDOGCOUNT] = {
0078 {
0079     .baseAddr = WDT_BASE,
0080     .intNum = INT_WDT,
0081     .intPriority = (~0),
0082     .reloadValue = 80000000 /* 1 second period at default CPU clock freq */
0083 }
0084 };
0085
0086 const Watchdog_Config Watchdog_config[CC3220SF_STARPORTS_WATCHDOGCOUNT] = {
0087 {
0088     .fxnTablePtr = &WatchdogCC32XX_fxnTable,
0089     .object = &watchdogCC3220SObjects[CC3220SF_STARPORTS_WATCHDOGO],
0090     .hwAttrs = &watchdogCC3220SHWAttrs[CC3220SF_STARPORTS_WATCHDOGO]
0091 }
0092 };
0093
0094 const uint_least8_t Watchdog_count = CC3220SF_STARPORTS_WATCHDOGCOUNT;
0095
0096 /*
0097 * ===== Board_debugHeader =====
0098 * This structure prevents the CC32XXSF bootloader from overwriting the
0099 * internal FLASH; this allows us to flash a program that will not be
0100 * overwritten by the bootloader with the encrypted program saved in
0101 * "secure/serial flash".
0102 *
0103 * This structure must be placed at the beginning of internal FLASH (so
0104 * the bootloader is able to recognize that it should not overwrite
0105 * internal FLASH).
0106 */
0107 #if defined(__SF_DEBUG__) || defined(__SF_NODEBUG__)
0108 #if defined(__TI_COMPILER_VERSION__)
0109 #pragma DATA_SECTION(Board_debugHeader, ".dbghdr")
0110 #pragma RETAIN(Board_debugHeader)
0111 #elif defined(__IAR_SYSTEMS_ICC__)
0112 #pragma location=".dbghdr"
0113 #elif defined(__GNUC__)
0114 __attribute__ ((section(".dbghdr")))
0115 #endif
0116 #if defined(__SF_DEBUG__)
0117 const uint32_t Board_debugHeader[] = {
0118     0x5A5A55A,
0119     0x000FF800,
0120     0xEFA3247D
0121 };
0122 #elif defined(__SF_NODEBUG__)
0123 const uint32_t Board_debugHeader[] = {
0124     0xFFFFFFFF,
0125     0xFFFFFFFF,
0126     0xFFFFFFFF
0127 };
0128#endif
0129#endif

```

3.15 CC3220SF_STARPORTS.h File Reference

Macros

- #define CC3220SF_STARPORTS_GPIO_LED_OFF (0)
- #define CC3220SF_STARPORTS_GPIO_LED_ON (1)

Enumerations

- enum `CC3220SF_STARPORTS_ADCName` { `CC3220SF_STARPORTS_ADC0` = 0, `CC3220SF_STARPORTS_ADCCOUNT` }
- enum `CC3220SF_STARPORTS_CaptureName` { `CC3220SF_STARPORTS_CAPTURE0` = 0, `CC3220SF_STARPORTS_CAPTURECOUNT` }
- enum `CC3220SF_STARPORTS_CryptoName` { `CC3220SF_STARPORTS_CRYPTO0` = 0, `CC3220SF_STARPORTS_CRYPTOCOUNT` }
- enum `CC3220SF_STARPORTS_GPIOName` {
 `CC3220SF_STARPORTS_RN2483_MCLR`, `CC3220SF_STARPORTS_AXDL355_CS`, `CC3220SF_STARPORTS_BME280_CS`,
 `CC3220SF_STARPORTS_LDC1000_CS`,
 `CC3220SF_STARPORTS_EN_NODE`, `CC3220SF_STARPORTS_EN_AXDL355`, `CC3220SF_STARPORTS_EN_BME280`,
 `CC3220SF_STARPORTS_EN_LDC1000`,
 `CC3220SF_STARPORTS_DS1374_INTB`, `CC3220SF_STARPORTS_SCL`, `CC3220SF_STARPORTS_SDA`,
 `CC3220SF_STARPORTS_SCLK`,
 `CC3220SF_STARPORTS_MOSI`, `CC3220SF_STARPORTS_CS`, `CC3220SF_STARPORTS_GPIOCOUNT`
}
- enum `CC3220SF_STARPORTS_I2CName` { `CC3220SF_STARPORTS_I2C0` = 0, `CC3220SF_STARPORTS_I2CCOUNT` }
- enum `CC3220SF_STARPORTS_PWMName` { `CC3220SF_STARPORTS_PWM5` = 0, `CC3220SF_STARPORTS_PWMCOUNT` }
- enum `CC3220SF_STARPORTS_SDName` { `CC3220SF_STARPORTS_SD0` = 0, `CC3220SF_STARPORTS_SDCOUNT` }
- enum `CC3220SF_STARPORTS_SPIName` { `CC3220SF_STARPORTS_SPI0` = 0, `CC3220SF_STARPORTS_SPI1`,
`CC3220SF_STARPORTS_SPICOUNT` }
- enum `CC3220SF_STARPORTS_TimerName` { `CC3220SF_STARPORTS_TIMER0` = 0, `CC3220SF_STARPORTS_TIMER1`,
`CC3220SF_STARPORTS_TIMER2`, `CC3220SF_STARPORTS_TIMERCOUNT` }
- enum `CC3220SF_STARPORTS_UARTName` { `CC3220SF_STARPORTS_UART0` = 0, `CC3220SF_STARPORTS_UART1`,
`CC3220SF_STARPORTS_UARTCOUNT` }
- enum `CC3220SF_STARPORTS_WatchdogName` { `CC3220SF_STARPORTS_WATCHDOG0` = 0,
`CC3220SF_STARPORTS_WATCHDOGCOUNT` }

Functions

- void `CC3220SF_STARPORTS_initGeneral` (void)
Initialize the general board specific settings.

3.15.1 Macro Definition Documentation

3.15.1.1 `CC3220SF_STARPORTS_GPIO_LED_OFF` #define `CC3220SF_STARPORTS_GPIO_LED_OFF` (0)

Definition at line 52 of file `CC3220SF_STARPORTS.h`.

3.15.1.2 `CC3220SF_STARPORTS_GPIO_LED_ON` #define `CC3220SF_STARPORTS_GPIO_LED_ON` (1)

Definition at line 53 of file `CC3220SF_STARPORTS.h`.

3.15.2 Enumeration Type Documentation

3.15.2.1 CC3220SF_STARPORTS_ADCName enum CC3220SF_STARPORTS_ADCName

Enumerator

CC3220SF_STARPORTS_ADC0	
CC3220SF_STARPORTS_ADCCOUNT	

Definition at line 59 of file [CC3220SF_STARPORTS.h](#).

```

00059
00060     CC3220SF_STARPORTS_ADC0 = 0,
00061
00062     CC3220SF_STARPORTS_ADCCOUNT
00063 } CC3220SF_STARPORTS_ADCName;

```

3.15.2.2 CC3220SF_STARPORTS_CaptureName enum [CC3220SF_STARPORTS_CaptureName](#)

Enumerator

CC3220SF_STARPORTS_CAPTURE0	
CC3220SF_STARPORTS_CAPTURE1	
CC3220SF_STARPORTS_CAPTURECOUNT	

Definition at line 69 of file [CC3220SF_STARPORTS.h](#).

```

00069
00070     CC3220SF_STARPORTS_CAPTURE0 = 0,
00071     CC3220SF_STARPORTS_CAPTURE1,
00072
00073     CC3220SF_STARPORTS_CAPTURECOUNT
00074 } CC3220SF_STARPORTS_CaptureName;

```

3.15.2.3 CC3220SF_STARPORTS_CryptoName enum [CC3220SF_STARPORTS_CryptoName](#)

Enumerator

CC3220SF_STARPORTS_CRYPTO0	
CC3220SF_STARPORTS_CRYPTOCOUNT	

Definition at line 80 of file [CC3220SF_STARPORTS.h](#).

```

00080
00081     CC3220SF_STARPORTS_CRYPTO0 = 0,
00082
00083     CC3220SF_STARPORTS_CRYPTOCOUNT
00084 } CC3220SF_STARPORTS_CryptoName;

```

3.15.2.4 CC3220SF_STARPORTS_GPIOName enum [CC3220SF_STARPORTS_GPIOName](#)

Enumerator

CC3220SF_STARPORTS_RN2483_MCLR	
CC3220SF_STARPORTS_AXDL355_CS	
CC3220SF_STARPORTS_BME280_CS	

Enumerator

CC3220SF_STARPORTS_LDC1000_CS	
CC3220SF_STARPORTS_EN_NODE	
CC3220SF_STARPORTS_EN_ADXL355	
CC3220SF_STARPORTS_EN_BME280	
CC3220SF_STARPORTS_EN_LDC1000	
CC3220SF_STARPORTS_DS1374_INTB	
CC3220SF_STARPORTS_SCL	
CC3220SF_STARPORTS_SDA	
CC3220SF_STARPORTS_SCLK	
CC3220SF_STARPORTS_MOSI	
CC3220SF_STARPORTS_CS	
CC3220SF_STARPORTS_GPIOCOUNT	

Definition at line 90 of file [CC3220SF_STARPORTS.h](#).

```

00090      /* /MCLR Signal for RN2483 */
00091      CC3220SF_STARPORTS_RN2483_MCLR,
00092
00093      /* STARPORTS GPIO Pins */
00094      CC3220SF_STARPORTS_ADXL355_CS,
00095      CC3220SF_STARPORTS_BME280_CS,
00096      CC3220SF_STARPORTS_LDC1000_CS,
00097      CC3220SF_STARPORTS_EN_NODE,
00098      CC3220SF_STARPORTS_EN_ADXL355,
00099      CC3220SF_STARPORTS_EN_BME280,
00100      CC3220SF_STARPORTS_EN_LDC1000,
00101      CC3220SF_STARPORTS_DS1374_INTB,
00102      CC3220SF_STARPORTS_SCL,
00103      CC3220SF_STARPORTS_SDA,
00104      CC3220SF_STARPORTS_SCLK,
00105      CC3220SF_STARPORTS_MOSI,
00106      CC3220SF_STARPORTS_CS,
00107
00108      CC3220SF_STARPORTS_GPIOCOUNT
00109 } CC3220SF_STARPORTS_GPIOName;
00110 }
```

3.15.2.5 CC3220SF_STARPORTS_I2CName enum [CC3220SF_STARPORTS_I2CName](#)

Enumerator

CC3220SF_STARPORTS_I2C0	
CC3220SF_STARPORTS_I2CCOUNT	

Definition at line 116 of file [CC3220SF_STARPORTS.h](#).

```

00116
00117     CC3220SF_STARPORTS_I2C0 = 0,
00118
00119     CC3220SF_STARPORTS_I2CCOUNT
00120 } CC3220SF_STARPORTS_I2CName;
```

3.15.2.6 CC3220SF_STARPORTS_PWMName enum [CC3220SF_STARPORTS_PWMName](#)

Enumerator

CC3220SF_STARPORTS_PWM5	
CC3220SF_STARPORTS_PWMCOUNT	

Definition at line 126 of file [CC3220SF_STARPORTS.h](#).

```
00126     {
00127     CC3220SF_STARPORTS_PWM5 = 0,
00128
00129     CC3220SF_STARPORTS_PWMCOUNT
00130 } CC3220SF_STARPORTS_PWMName;
```

3.15.2.7 CC3220SF_STARPORTS_SDFatFSName enum [CC3220SF_STARPORTS_SDFatFSName](#)

Enumerator

CC3220SF_STARPORTS_SDFatFS0	
CC3220SF_STARPORTS_SDFatFSCOUNT	

Definition at line 136 of file [CC3220SF_STARPORTS.h](#).

```
00136     {
00137     CC3220SF_STARPORTS_SDFatFS0 = 0,
00138
00139     CC3220SF_STARPORTS_SDFatFSCOUNT
00140 } CC3220SF_STARPORTS_SDFatFSName;
```

3.15.2.8 CC3220SF_STARPORTS_SDName enum [CC3220SF_STARPORTS_SDName](#)

Enumerator

CC3220SF_STARPORTS_SD0	
CC3220SF_STARPORTS_SDCOUNT	

Definition at line 146 of file [CC3220SF_STARPORTS.h](#).

```
00146     {
00147     CC3220SF_STARPORTS_SD0 = 0,
00148
00149     CC3220SF_STARPORTS_SDCOUNT
00150 } CC3220SF_STARPORTS_SDName;
```

3.15.2.9 CC3220SF_STARPORTS_SPIName enum [CC3220SF_STARPORTS_SPIName](#)

Enumerator

CC3220SF_STARPORTS_SPI0	
CC3220SF_STARPORTS_SPI1	
CC3220SF_STARPORTS_SPICOUNT	

Definition at line 156 of file [CC3220SF_STARPORTS.h](#).

```
00156     {
00157     CC3220SF_STARPORTS_SPI0 = 0,
00158     CC3220SF_STARPORTS_SPI1,
00159
00160     CC3220SF_STARPORTS_SPICOUNT
00161 } CC3220SF_STARPORTS_SPIName;
```

3.15.2.10 CC3220SF_STARPORTS_TimerName enum [CC3220SF_STARPORTS_TimerName](#)**Enumerator**

CC3220SF_STARPORTS_TIMER0	
CC3220SF_STARPORTS_TIMER1	
CC3220SF_STARPORTS_TIMER2	
CC3220SF_STARPORTS_TIMERCOUNT	

Definition at line 167 of file [CC3220SF_STARPORTS.h](#).

```
00167     {  
00168     CC3220SF_STARPORTS_TIMER0 = 0,  
00169     CC3220SF_STARPORTS_TIMER1,  
00170     CC3220SF_STARPORTS_TIMER2,  
00171     CC3220SF_STARPORTS_TIMERCOUNT  
00172 } CC3220SF_STARPORTS_TimerName;
```

3.15.2.11 CC3220SF_STARPORTS_UARTName enum [CC3220SF_STARPORTS_UARTName](#)**Enumerator**

CC3220SF_STARPORTS_UART0	
CC3220SF_STARPORTS_UART1	
CC3220SF_STARPORTS_UARTCOUNT	

Definition at line 179 of file [CC3220SF_STARPORTS.h](#).

```
00179     {  
00180     CC3220SF_STARPORTS_UART0 = 0,  
00181     CC3220SF_STARPORTS_UART1,  
00182     CC3220SF_STARPORTS_UARTCOUNT  
00183 } CC3220SF_STARPORTS_UARTName;
```

3.15.2.12 CC3220SF_STARPORTS_WatchdogName enum [CC3220SF_STARPORTS_WatchdogName](#)**Enumerator**

CC3220SF_STARPORTS_WATCHDOG0	
CC3220SF_STARPORTS_WATCHDOGCOUNT	

Definition at line 190 of file [CC3220SF_STARPORTS.h](#).

```
00190     {  
00191     CC3220SF_STARPORTS_WATCHDOG0 = 0,  
00192     CC3220SF_STARPORTS_WATCHDOGCOUNT  
00193 } CC3220SF_STARPORTS_WatchdogName;
```

3.15.3 Function Documentation

3.15.3.1 CC3220SF_STARPORTS_initGeneral() void CC3220SF_STARPORTS_initGeneral (void)

Initialize the general board specific settings.

This function initializes the general board specific settings.

Definition at line 190 of file [CC3220SF_STARPORTS.c](#).

```
00191 {  
00192     PRCMCC3200MCUInit();  
00193     Power_init();  
00194 }
```

3.16 CC3220SF_STARPORTS.h

```
00001 /*  
00002 * Copyright (c) 2016-2018, Texas Instruments Incorporated  
00003 * All rights reserved.  
00004 *  
00005 * Redistribution and use in source and binary forms, with or without  
00006 * modification, are permitted provided that the following conditions  
00007 * are met:  
00008 *  
00009 * * Redistributions of source code must retain the above copyright  
00010 * notice, this list of conditions and the following disclaimer.  
00011 *  
00012 * * Redistributions in binary form must reproduce the above copyright  
00013 * notice, this list of conditions and the following disclaimer in the  
00014 * documentation and/or other materials provided with the distribution.  
00015 *  
00016 * * Neither the name of Texas Instruments Incorporated nor the names of  
00017 * its contributors may be used to endorse or promote products derived  
00018 * from this software without specific prior written permission.  
00019 *  
00020 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
00021 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,  
00022 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR  
00023 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR  
00024 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,  
00025 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,  
00026 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;  
00027 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,  
00028 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR  
00029 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,  
00030 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  
00031 */  
00045 #ifndef __CC3220SF_STARPORTS_H  
00046 #define __CC3220SF_STARPORTS_H  
00047  
00048 #ifdef __cplusplus  
00049 extern "C" {  
00050 #endif  
00051  
00052 #define CC3220SF_STARPORTS_GPIO_LED_OFF (0)  
00053 #define CC3220SF_STARPORTS_GPIO_LED_ON (1)  
00054  
00059 typedef enum CC3220SF_STARPORTS_ADCName {  
00060     CC3220SF_STARPORTS_ADC0 = 0,  
00061  
00062     CC3220SF_STARPORTS_ADCCOUNT  
00063 } CC3220SF_STARPORTS_ADCName;  
00064  
00069 typedef enum CC3220SF_STARPORTS_CaptureName {  
00070     CC3220SF_STARPORTS_CAPTURE0 = 0,  
00071     CC3220SF_STARPORTS_CAPTURE1,  
00072  
00073     CC3220SF_STARPORTS_CAPTURECOUNT  
00074 } CC3220SF_STARPORTS_CaptureName;  
00075  
00080 typedef enum CC3220SF_STARPORTS_CryptoName {  
00081     CC3220SF_STARPORTS_CRYPTO00 = 0,  
00082  
00083     CC3220SF_STARPORTS_CRYPTOCOUNT  
00084 } CC3220SF_STARPORTS_CryptoName;  
00085  
00090 typedef enum CC3220SF_STARPORTS_GPIOName {  
00091     /* /MCLR Signal for RN2483 */  
00092     CC3220SF_STARPORTS_RN2483_MCLR,  
00093  
00094     /* STARPORTS GPIO Pins */  
00095     CC3220SF_STARPORTS_ADXL355_CS,
```

```

00096     CC3220SF_STARPORTS_BME280_CS,
00097     CC3220SF_STARPORTS_LDC1000_CS,
00098     CC3220SF_STARPORTS_EN_NODE,
00099     CC3220SF_STARPORTS_EN_AXDL355,
00100     CC3220SF_STARPORTS_EN_BME280,
00101     CC3220SF_STARPORTS_EN_LDC1000,
00102     CC3220SF_STARPORTS_DS1374_INTB,
00103     CC3220SF_STARPORTS_SCL,
00104     CC3220SF_STARPORTS_SDA,
00105     CC3220SF_STARPORTS_SCLK,
00106     CC3220SF_STARPORTS_MOSI,
00107     CC3220SF_STARPORTS_CS,
00108
00109     CC3220SF_STARPORTS_GPIOCOUNT
00110 } CC3220SF_STARPORTS_GPIOName;
00111
00116 typedef enum CC3220SF_STARPORTS_I2CName {
00117     CC3220SF_STARPORTS_I2C0 = 0,
00118
00119     CC3220SF_STARPORTS_I2CCOUNT
00120 } CC3220SF_STARPORTS_I2CName;
00121
00126 typedef enum CC3220SF_STARPORTS_PWMName {
00127     CC3220SF_STARPORTS_PWM5 = 0,
00128
00129     CC3220SF_STARPORTS_PWMCOUNT
00130 } CC3220SF_STARPORTS_PWMName;
00131
00136 typedef enum CC3220SF_STARPORTS_SDFatFSName {
00137     CC3220SF_STARPORTS_SDFatFS0 = 0,
00138
00139     CC3220SF_STARPORTS_SDFatFSCOUNT
00140 } CC3220SF_STARPORTS_SDFatFSName;
00141
00146 typedef enum CC3220SF_STARPORTS_SDName {
00147     CC3220SF_STARPORTS_SD0 = 0,
00148
00149     CC3220SF_STARPORTS_SDCOUNT
00150 } CC3220SF_STARPORTS_SDName;
00151
00156 typedef enum CC3220SF_STARPORTS_SPIName {
00157     CC3220SF_STARPORTS_SPI0 = 0,
00158     CC3220SF_STARPORTS_SPI1,
00159
00160     CC3220SF_STARPORTS_SPICOUNT
00161 } CC3220SF_STARPORTS_SPIName;
00162
00167 typedef enum CC3220SF_STARPORTS_TimerName {
00168     CC3220SF_STARPORTS_TIMER0 = 0,
00169     CC3220SF_STARPORTS_TIMER1,
00170     CC3220SF_STARPORTS_TIMER2,
00171
00172     CC3220SF_STARPORTS_TIMERCOUNT
00173 } CC3220SF_STARPORTS_TimerName;
00174
00179 typedef enum CC3220SF_STARPORTS_UARTName {
00180     CC3220SF_STARPORTS_UART0 = 0,
00181     CC3220SF_STARPORTS_UART1,
00182
00183     CC3220SF_STARPORTS_UARTCOUNT
00184 } CC3220SF_STARPORTS_UARTName;
00185
00190 typedef enum CC3220SF_STARPORTS_WatchdogName {
00191     CC3220SF_STARPORTS_WATCHDOG0 = 0,
00192
00193     CC3220SF_STARPORTS_WATCHDOGCOUNT
00194 } CC3220SF_STARPORTS_WatchdogName;
00195
00201 extern void CC3220SF_STARPORTS_initGeneral(void);
00202
00203 #ifdef __cplusplus
00204 }
00205 #endif
00206
00207 #endif /* __CC3220SF_STARPORTS_H */

```

3.17 DS1374.c File Reference

Functions to interact with Watchdog RTC.

```
#include <ti/drivers/I2C.h>
#include "hal_I2C.h"
```

```
#include "hal_UART.h"
#include "DS1374.h"
```

Functions

- void [DS1374_Clear_AF](#) (I2C_Handle *i2c*)
DS1374 Clear AF Function.
- uint8_t [DS1374_Read_Ctrl](#) (I2C_Handle *i2c*)
DS1374 Read Control Function.
- uint8_t [DS1374_Read_Status](#) (I2C_Handle *i2c*)
DS1374 Read Status Function.
- uint32_t [DS1374_Read_Tod](#) (I2C_Handle *i2c*)
DS1374 Read Tod Function.
- uint32_t [DS1374_Read_WdAlmb](#) (I2C_Handle *i2c*)
DS1374 Read WdAlmb Function.
- void [DS1374_Write_Ctrl](#) (I2C_Handle *i2c*)
DS1374 Write Control Function.
- void [DS1374_Write_Tod](#) (I2C_Handle *i2c*, uint32_t Val)
DS1374 Write Tod Function.
- void [DS1374_Write_WdAlmb](#) (I2C_Handle *i2c*, uint32_t Val)
DS1374 Write WdAlmb Function.

3.17.1 Detailed Description

Functions to interact with Watchdog RTC.

Version

1.0

Date

07/05/2019

Author

A.Irizar @tittle DS1374 Watchdog RTC functions

Definition in file [DS1374.c](#).

3.17.2 Function Documentation

3.17.2.1 [DS1374_Clear_AF\(\)](#) void DS1374_Clear_AF (

I2C_Handle *i2c*)

DS1374 Clear AF Function.

This function

1. Clears AF register of DS1374

Parameters

in	<i>I2C_Handle</i>	i2c
----	-------------------	-----

Returns

None

Definition at line 179 of file DS1374.c.

```

00179
00180
00181     uint8_t RxBuffer;
00182     uint8_t Val;
00183     I2C_read_8bits(i2c, 0x68, DS1374_STATUS, &RxBuffer, 1);
00184     Val = RxBuffer & 0xFE;
00185     I2C_write_8bits(i2c, 0x68, DS1374_STATUS, Val);
00186 }
```

3.17.2.2 DS1374_Read_Ctrl() `uint8_t DS1374_Read_Ctrl (`
 `I2C_Handle i2c)`

DS1374 Read Control Function.

This function

1. Reads Control register of DS1374 and returns its value

Parameters

in	<i>I2C_Handle</i>	i2c
----	-------------------	-----

Returns`uint8_t RxBuffer`**Definition at line 138 of file DS1374.c.**

```

00138
00139     uint8_t RxBuffer;
00140
00141     I2C_read_8bits(i2c, 0x68, DS1374_CTRL, &RxBuffer, 1);
00142
00143     return RxBuffer;
00144 }
```

3.17.2.3 DS1374_Read_Status() `uint8_t DS1374_Read_Status (`
 `I2C_Handle i2c)`

DS1374 Read Status Function.

This function

1. Reads Status register of DS1374 and returns its value

Parameters

in	<i>I2C_Handle</i>	i2c
----	-------------------	-----

Returns

uint8_t RxBuffer

Definition at line 158 of file [DS1374.c](#).

```
00158
00159
00160     uint8_t RxBuffer;
00161
00162     I2C_read_8bits(i2c, 0x68, DS1374_STATUS, &RxBuffer, 1);
00163
00164     return RxBuffer;
00165 }
```

3.17.2.4 DS1374_Read_Tod() uint32_t DS1374_Read_Tod (

 I2C_Handle i2c)

DS1374 Read Tod Function.

This function

1. Reads Tod register of DS1374 and returns its value

Parameters

in	<i>I2C_Handle</i>	i2c
----	-------------------	-----

Returns

uint8_t RxBuffer

Definition at line 34 of file [DS1374.c](#).

```
00034
00035
00036     uint8_t RxBuffer[4];
00037
00038     I2C_read_8bits(i2c, 0x68, TODO, RxBuffer, 4);
00039
00040     return RxBuffer[0] + (RxBuffer[1]<<8) + (RxBuffer[2]<<16) + (RxBuffer[3]<<24);
00041 }
```

3.17.2.5 DS1374_Read_WdAlmb() uint32_t DS1374_Read_WdAlmb (

 I2C_Handle i2c)

DS1374 Read WdAlmb Function.

This function

1. Reads WdAlmb register of DS1374 and returns its value

Parameters

in	<i>I2C_Handle</i>	i2c
----	-------------------	-----

Returns

uint8_t RxBuffer

Definition at line 76 of file DS1374.c.

```

00076
00077
00078     uint8_t RxBuffer[3];
00079
00080     I2C_read_8bits(i2c, 0x68, WDALMBO, RxBuffer, 3);
00081
00082     return RxBuffer[0] + (RxBuffer[1]«8) + (RxBuffer[2]«16);
00083 }
```

3.17.2.6 DS1374_Write_Ctrl() void DS1374_Write_Ctrl (
 I2C_Handle *i2c*)

DS1374 Write Control Function.

This function

1. Writes Control register of DS1374

Parameters

in	<i>I2C_Handle</i>	i2c
----	-------------------	-----

Returns

None

Definition at line 117 of file DS1374.c.

```

00117
00118
00119     uint8_t Val;
00120
00121     Val = (1«6) | (1«3) | 1;
00122
00123     I2C_write_8bits(i2c, 0x68, DS1374_CTRL, Val);
00124 }
```

3.17.2.7 DS1374_Write_Tod() void DS1374_Write_Tod (
 I2C_Handle *i2c*,
 uint32_t *Val*)

DS1374 Write Tod Function.

This function

1. Writes Tod register of DS1374

Parameters

in	<i>I2C_Handle</i>	i2c
in	<i>uint32_t</i>	Val

Returns

None

Definition at line 56 of file DS1374.c.

```

00056
00057
00058     I2C_write_8bits(i2c, 0x68, TOD3, (uint8_t)( (Val>>24) & 0xFF));
00059     I2C_write_8bits(i2c, 0x68, TOD2, (uint8_t)( (Val>>16) & 0xFF));
00060     I2C_write_8bits(i2c, 0x68, TOD1, (uint8_t)( (Val>>8) & 0xFF));
00061     I2C_write_8bits(i2c, 0x68, TODO, (uint8_t)( Val & 0xFF));
00062 }
```

3.17.2.8 DS1374_Write_WdAlmb() void DS1374_Write_WdAlmb (

```

    I2C_Handle i2c,
    uint32_t Val )
```

DS1374 Write WdAlmb Function.

This function

1. Writes WdAlmb register of DS1374

Parameters

in	<i>I2C_Handle</i>	i2c
in	<i>uint32_t</i>	Val

Returns

None

Definition at line 98 of file DS1374.c.

```

00098
00099
00100     I2C_write_8bits(i2c, 0x68, WDALMB2, (uint8_t)( (Val>>16) & 0xFF));
00101     I2C_write_8bits(i2c, 0x68, WDALMB1, (uint8_t)( (Val>>8) & 0xFF));
00102     I2C_write_8bits(i2c, 0x68, WDALMB0, (uint8_t)( Val & 0xFF));
00103 }
```

3.18 DS1374.c

```

00001
00010 //*****
00011 //           INCLUDES
00012 //*****
00013 #include <ti/drivers/I2C.h>
00014 #include "hal_I2C.h"
00015 #include "hal_UART.h"
00016 #include "DS1374.h"
```

```

00017
00018 //*****
00019 //      FUNCTIONS
00020 //*****
00021
00022 //*****
00023 //
00024 //
00025 //*****
00026 uint32_t DS1374_Read_Tod(I2C_Handle i2c) {
00027
00028     uint8_t RxBuffer[4];
00029
00030     I2C_read_8bits(i2c, 0x68, TODO, RxBuffer, 4);
00031
00032     return RxBuffer[0] + (RxBuffer[1]«8) + (RxBuffer[2]«16) + (RxBuffer[3]«24);
00033 }
00034
00035
00036
00037
00038
00039
00040
00041 }
00042
00043 //*****
00044 //
00045 //
00046 //*****
00047 void DS1374_Write_Tod(I2C_Handle i2c, uint32_t Val) {
00048
00049     I2C_write_8bits(i2c, 0x68, TOD3, (uint8_t)( (Val»24) & 0xFF));
00050     I2C_write_8bits(i2c, 0x68, TOD2, (uint8_t)( (Val»16) & 0xFF));
00051     I2C_write_8bits(i2c, 0x68, TOD1, (uint8_t)( (Val»8) & 0xFF));
00052     I2C_write_8bits(i2c, 0x68, TODO, (uint8_t)( Val & 0xFF));
00053 }
00054
00055
00056
00057
00058
00059
00060
00061
00062 }
00063
00064 //*****
00065 //
00066 //
00067 //*****
00068 uint32_t DS1374_Read_WdAlmb(I2C_Handle i2c) {
00069
00070     uint8_t RxBuffer[3];
00071
00072     I2C_read_8bits(i2c, 0x68, WDALMB0, RxBuffer, 3);
00073
00074     return RxBuffer[0] + (RxBuffer[1]«8) + (RxBuffer[2]«16);
00075 }
00076
00077
00078
00079
00080
00081
00082
00083 }
00084
00085 //*****
00086 //
00087 //
00088 //*****
00089 void DS1374_Write_WdAlmb(I2C_Handle i2c, uint32_t Val) {
00090
00091     I2C_write_8bits(i2c, 0x68, WDALMB2, (uint8_t)( (Val»16) & 0xFF));
00092     I2C_write_8bits(i2c, 0x68, WDALMB1, (uint8_t)( (Val»8) & 0xFF));
00093     I2C_write_8bits(i2c, 0x68, WDALMB0, (uint8_t)( Val & 0xFF));
00094 }
00095
00096
00097
00098
00099
00100
00101
00102
00103 }
00104
00105 //*****
00106 //
00107 //
00108 //*****
00109 void DS1374_Write_Ctrl(I2C_Handle i2c) {
00110
00111     uint8_t Val;
00112
00113     Val = (1«6) | (1«3) | 1;
00114
00115     I2C_write_8bits(i2c, 0x68, DS1374_CTRL, Val);
00116 }
00117
00118
00119
00120
00121
00122
00123
00124 }
00125
00126 //*****
00127 //
00128 //
00129 //*****
00130 uint8_t DS1374_Read_Ctrl(I2C_Handle i2c) {
00131
00132     uint8_t RxBuffer;
00133
00134     I2C_read_8bits(i2c, 0x68, DS1374_CTRL, &RxBuffer, 1);
00135
00136     return RxBuffer;
00137 }
00138
00139
00140
00141
00142
00143
00144 }
00145
00146 //*****
00147 //
00148 //
00149 //*****
00150 uint8_t DS1374_Read_Status(I2C_Handle i2c) {
00151
00152     uint8_t RxBuffer;
00153
00154
00155
00156
00157
00158
00159
00160
00161

```

```

00162     I2C_read_8bits(i2c, 0x68, DS1374_STATUS, &RxBuffer, 1);
00163     return RxBuffer;
00165 }
00166
00167 //*****
00168 //
00177 //*****
00178 //*****
00179 void DS1374_Clear_AF(I2C_Handle i2c) {
00180
00181     uint8_t RxBuffer;
00182     uint8_t Val;
00183     I2C_read_8bits(i2c, 0x68, DS1374_STATUS, &RxBuffer, 1);
00184     Val = RxBuffer & 0xFE;
00185     I2C_write_8bits(i2c, 0x68, DS1374_STATUS, Val);
00186 }
```

3.19 DS1374.h File Reference

Functions to interact with Watchdog RTC.

```
#include <ti/drivers/I2C.h>
```

Enumerations

- enum DS1374_register_t {
 TOD0 = 0x00, TOD1 = 0x01, TOD2 = 0x02, TOD3 = 0x03,
 WDALMB0 = 0x04, WDALMB1 = 0x05, WDALMB2 = 0x06, DS1374_CTRL = 0x07,
 DS1374_STATUS = 0x08, DS1374_TC = 0x09 }
- DS1374 Address map.*

Functions

- void DS1374_Clear_AF (I2C_Handle i2c)

DS1374 Clear AF Function.
- uint8_t DS1374_Read_Ctrl (I2C_Handle i2c)

DS1374 Read Control Function.
- uint8_t DS1374_Read_Status (I2C_Handle i2c)

DS1374 Read Status Function.
- uint32_t DS1374_Read_Tod (I2C_Handle i2c)

DS1374 Read Tod Function.
- uint32_t DS1374_Read_WdAlmb (I2C_Handle i2c)

DS1374 Read WdAlmb Function.
- void DS1374_Write_Ctrl (I2C_Handle i2c)

DS1374 Write Control Function.
- void DS1374_Write_Tod (I2C_Handle i2c, uint32_t Val)

DS1374 Write Tod Function.
- void DS1374_Write_WdAlmb (I2C_Handle i2c, uint32_t Val)

DS1374 Write WdAlmb Function.

3.19.1 Detailed Description

Functions to interact with Watchdog RTC.

Version

1.0

Date

07/05/2019

Author

A.Irizar @title ADXL355 accelerometer functions

Definition in file [DS1374.h](#).

3.19.2 Enumeration Type Documentation

3.19.2.1 DS1374_register_t enum DS1374_register_t

DS1374 Address map.

Enumerator

TOD0	Time-of-Day Counter byte 0 reg
TOD1	Time-of-Day Counter byte 1 reg
TOD2	Time-of-Day Counter byte 2 reg
TOD3	Time-of-Day Counter byte 3 reg
WDALMB0	Watchdog/Alarm Counter byte 0 reg
WDALMB1	Watchdog/Alarm Counter byte 1 reg
WDALMB2	Watchdog/Alarm Counter byte 2 reg
DS1374_CTRL	Control reg
DS1374_STATUS	Status reg
DS1374_TC	Trickle Charger reg

Definition at line 25 of file [DS1374.h](#).

```
00025 {
00026     TODO = 0x00,
00027     TOD1 = 0x01,
00028     TOD2 = 0x02,
00029     TOD3 = 0x03,
00030     WDALMB0 = 0x04,
00031     WDALMB1 = 0x05,
00032     WDALMB2 = 0x06,
00033     DS1374_CTRL = 0x07,
00034     DS1374_STATUS = 0x08,
00035     DS1374_TC = 0x09
00036 } DS1374_register_t;
```

3.19.3 Function Documentation

3.19.3.1 DS1374_Clear_AF() void DS1374_Clear_AF (I2C_Handle i2c)

DS1374 Clear AF Function.

This function

1. Clears AF register of DS1374

Parameters

in	I2C_Handle	i2c
----	------------	-----

Returns

None

Definition at line 179 of file [DS1374.c](#).

```
00179                                         {  
00180                                         uint8_t RxBuffer;  
00181                                         uint8_t Val;  
00182                                         I2C_read_8bits(i2c, 0x68, DS1374_STATUS, &RxBuffer, 1);  
00183                                         Val = RxBuffer & 0xFE;  
00184                                         I2C_write_8bits(i2c, 0x68, DS1374_STATUS, Val);  
00185                                         }  
00186 }
```

3.19.3.2 DS1374_Read_Ctrl() uint8_t DS1374_Read_Ctrl (I2C_Handle i2c)

DS1374 Read Control Function.

This function

1. Reads Control register of DS1374 and returns its value

Parameters

in	I2C_Handle	i2c
----	------------	-----

Returns

uint8_t RxBuffer

Definition at line 138 of file [DS1374.c](#).

```

00138             {
00139     uint8_t RxBuffer;
00140
00141     I2C_read_8bits(i2c, 0x68, DS1374_CTRL, &RxBuffer, 1);
00142
00143     return RxBuffer;
00144 }
```

3.19.3.3 DS1374_Read_Status() uint8_t DS1374_Read_Status (

I2C_Handle i2c)

DS1374 Read Status Function.

This function

1. Reads Status register of DS1374 and returns its value

Parameters

in	I2C_Handle	i2c
----	------------	-----

Returns

uint8_t RxBuffer

Definition at line 158 of file [DS1374.c](#).

```

00158                                         {
00159
00160     uint8_t RxBuffer;
00161
00162     I2C_read_8bits(i2c, 0x68, DS1374_STATUS, &RxBuffer, 1);
00163
00164     return RxBuffer;
00165 }
```

3.19.3.4 DS1374_Read_Tod() uint32_t DS1374_Read_Tod (

I2C_Handle i2c)

DS1374 Read Tod Function.

This function

1. Reads Tod register of DS1374 and returns its value

Parameters

in	I2C_Handle	i2c
----	------------	-----

Returns

uint8_t RxBuffer

Definition at line 34 of file [DS1374.c](#).

```
00034     {  
00035  
00036     uint8_t RxBuffer[4];  
00037  
00038     I2C_read_8bits(i2c, 0x68, TODO, RxBuffer, 4);  
00039  
00040     return RxBuffer[0] + (RxBuffer[1]«8) + (RxBuffer[2]«16) + (RxBuffer[3]«24);  
00041 }
```

3.19.3.5 DS1374_Read_WdAlmb() uint32_t DS1374_Read_WdAlmb (
I2C_Handle i2c)

DS1374 Read WdAlmb Function.

This function

1. Reads WdAlmb register of DS1374 and returns its value

Parameters

in	I2C_Handle	i2c
----	------------	-----

Returns

uint8_t RxBuffer

Definition at line 76 of file [DS1374.c](#).

```
00076     {  
00077  
00078     uint8_t RxBuffer[3];  
00079  
00080     I2C_read_8bits(i2c, 0x68, WDALMB0, RxBuffer, 3);  
00081  
00082     return RxBuffer[0] + (RxBuffer[1]«8) + (RxBuffer[2]«16);  
00083 }
```

3.19.3.6 DS1374_Write_Ctrl() void DS1374_Write_Ctrl (
I2C_Handle i2c)

DS1374 Write Control Function.

This function

1. Writes Control register of DS1374

Parameters

in	<i>I2C_Handle</i>	i2c
----	-------------------	-----

Returns

None

Definition at line 117 of file DS1374.c.

```

00117
00118
00119     uint8_t Val;
00120
00121     Val = (1<<6) | (1<<3) | 1;
00122
00123     I2C_write_8bits(i2c, 0x68, DS1374_CTRL, Val);
00124 }
```

3.19.3.7 DS1374_Write_Tod() void DS1374_Write_Tod (

```
    I2C_Handle i2c,
    uint32_t Val )
```

DS1374 Write Tod Function.

This function

1. Writes Tod register of DS1374

Parameters

in	<i>I2C_Handle</i>	i2c
in	<i>uint32_t</i>	Val

Returns

None

Definition at line 56 of file DS1374.c.

```

00056
00057
00058     I2C_write_8bits(i2c, 0x68, TOD3, (uint8_t)( (Val>>24) & 0xFF));
00059     I2C_write_8bits(i2c, 0x68, TOD2, (uint8_t)( (Val>>16) & 0xFF));
00060     I2C_write_8bits(i2c, 0x68, TOD1, (uint8_t)( (Val>>8) & 0xFF));
00061     I2C_write_8bits(i2c, 0x68, TODO, (uint8_t)( Val & 0xFF));
00062 }
```

3.19.3.8 DS1374_Write_WdAlmb() void DS1374_Write_WdAlmb (

```
    I2C_Handle i2c,
    uint32_t Val )
```

DS1374 Write WdAlmb Function.

This function

1. Writes WdAlmb register of DS1374

Parameters

in	<i>I2C_Handle</i>	i2c
in	<i>uint32_t</i>	Val

Returns

None

Definition at line 98 of file DS1374.c.

```
00098
00099
00100     I2C_write_8bits(i2c, 0x68, WDALMB2, (uint8_t)( (Val»16) & 0xFF));
00101     I2C_write_8bits(i2c, 0x68, WDALMB1, (uint8_t)( (Val»8) & 0xFF));
00102     I2C_write_8bits(i2c, 0x68, WDALMB0, (uint8_t)( Val & 0xFF));
00103 }
```

3.20 DS1374.h

```
00001
00010 #ifndef DS1374_H_
00011 #define DS1374_H_
00012
00013 //*****
00014 //      INCLUDES
00015 //*****
00016 #include <ti/drivers/I2C.h>
00017
00018 //*****
00019 //      TYPEDEFS
00020 //*****
00021
00025 typedef enum {
00026     TODO0 = 0x00,
00027     TODO1 = 0x01,
00028     TODO2 = 0x02,
00029     TODO3 = 0x03,
00030     WDALMB0 = 0x04,
00031     WDALMB1 = 0x05,
00032     WDALMB2 = 0x06,
00033     DS1374_CTRL = 0x07,
00034     DS1374_STATUS = 0x08,
00035     DS1374_TC = 0x09
00036 } DS1374_register_t;
00037
00038 //*****
00039 //      FUNCTION PROTOTYPES
00040 //*****
00041 uint32_t DS1374_Read_Tod(I2C_Handle i2c);
00042 void DS1374_Write_Tod(I2C_Handle i2c, uint32_t Val);
00043 uint32_t DS1374_Read_WdAlmb(I2C_Handle i2c);
00044 void DS1374_Write_WdAlmb(I2C_Handle i2c, uint32_t Val);
00045 void DS1374_Write_Ctrl(I2C_Handle i2c);
00046 uint8_t DS1374_Read_Ctrl(I2C_Handle i2c);
00047 uint8_t DS1374_Read_Status(I2C_Handle i2c);
00048 void DS1374_Clear_AF(I2C_Handle i2c);
00049
00050 #endif /* DS1374_H_ */
```

3.21 file_system.c File Reference

Functions to read and write data from file system files.

```
#include <ti/devices/cc32xx/inc/hw_types.h>
#include <ti/devices/cc32xx/driverlib/prcm.h>
#include "hal_UART.h"
#include "file_system.h"
```

Functions

- int `deleteFile ()`
File System write NFails Function.
- int `st_readFileChangeWakeUp ()`
File System read ChangeWakeUp Function.
- int `st_readFileDnCntr ()`
File System read DnCntr Function.
- int `st_read FileMode ()`
File System read Mode Function.
- int `st_readFileNBoot ()`
File System read NBoot Function.
- int `st_readFileNCycles ()`
File System read NCycles Function.
- int `st_readFileNFails ()`
File System read NFails Function.
- int `st_readFileNodeld ()`
File System read Nodeld Function.
- int `st_readFileSSID (unsigned char *ssid)`
File System read SSID Function.
- int `st_readFileUpCntr ()`
File System read UpCntr Function.
- int `st_readFileWakeUp ()`
File System read WakeUp Interval Function.
- int `writeChangeWakeUp (uint8_t ChangeWakeUp)`
File System write ChangeWakeUp Function.
- int `writeDnCntr (uint32_t Dnctr)`
File System write NFails Function.
- int `writeMode (uint8_t Mode)`
File System write Mode Function.
- int `writeNBoot (uint8_t FirstBoot)`
File System write NBoot Function.
- int `writeNCycles (uint8_t NCycles)`
File System write NCycles Function.
- int `writeNFails (uint16_t NFails)`
File System write NFails Function.
- int `writeNodeld (uint16_t Nodeld)`
File System write Nodeld Function.
- int `writeSSID (unsigned char *SSID)`
File System write SSID Function.
- int `writeUpCntr (uint32_t Upctr)`
File System write Up Counter Function.
- int `writeWakeUp (uint16_t WakeUpInterval)`
File System write WakeUp Function.

Variables

- UART_Handle `uart0`

3.21.1 Detailed Description

Functions to read and write data from file system files.

Version

1.0

Date

07/05/2019

Author

A.Pz @title File system functions

Definition in file [file_system.c](#).

3.21.2 Function Documentation

3.21.2.1 `deleteFile()` int deleteFile ()

File System write NFails Function.

This function

1. Writes the number of LoRa OTAA connection fails on NFails file

Parameters

in	NFails	Number of LoRa OTAA connection fails
----	--------	--------------------------------------

Returns

_i32 offset

Definition at line [875](#) of file [file_system.c](#).

```
00875          {  
00876  
00877     sl_FsDel(FS_UPCNTR, 0);  
00878     st_listFiles(0);  
00879     return 0;  
00880 }
```

3.21.2.2 st_readFileChangeWakeUp() int st_readFileChangeWakeUp ()

File System read ChangeWakeUp Function.

This function

1. Reads the ChangeWakeUp value of the node from FS_CHANGEWAKEUP file

Returns

0

Definition at line 742 of file [file_system.c](#).

```
00742
00743
00744     int offset = 0;
00745     int RetVal = 0;
00746     _i8 buffer[MAX_FILE_SIZE];
00747     int32_t fd;
00748
00749     fd = sl_FsOpen(FS_CHANGEWAKEUP, SL_FS_READ, 0);
00750     if (fd < 0){
00751         UART_PRINT("\r\nError opening the file : %s\n\r", FS_CHANGEWAKEUP);
00752     }
00753
00754     while(RetVal=sl_FsRead(fd, offset, &buffer[0], MAX_FILE_SIZE)>1){
00755         if(strlen(buffer)!=0){
00756             if(RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE){
00757                 break;
00758             }
00759             if(RetVal < 0){
00760                 UART_PRINT("sl_FsRead error: %d\n\r", RetVal);
00761                 return RetVal;
00762             }
00763             offset += strlen(buffer);
00764         }
00765     }
00766     UART_PRINT("READING CHANGE WAKE UP: %d\n\r", (atoi(&buffer)));
00767
00768     RetVal = sl_FsClose(fd, 0, 0, 0);
00769     if (RetVal < 0){
00770         UART_PRINT("Error closing the file : %s\n\r", FS_CHANGEWAKEUP);
00771     }
00772
00773     return (atoi(&buffer));
00774 }
```

3.21.2.3 st_readFileDnCntr() int st_readFileDnCntr ()

File System read DnCntr Function.

This function

1. Reads the Down Counter parameter of Lora GW from FS_DNCNTR file

Returns

0

Definition at line 653 of file file_system.c.

```

00653     {
00654
00655     int offset = 0;
00656     int RetVal = 0;
00657     _i8 buffer[MAX_FILE_SIZE];
00658     int32_t fd;
00659
00660     fd = sl_FsOpen(FS_DNCNTR, SL_FS_READ, 0);
00661     if (fd < 0) {
00662         UART_PRINT("\r\nError opening the file : %s\r", FS_DNCNTR);
00663     }
00664
00665     while (RetVal=sl_FsRead(fd, offset, &buffer[0], MAX_FILE_SIZE)>1) {
00666         if (strlen(buffer) != 0) {
00667             if (RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE) {
00668                 break;
00669             }
00670             if (RetVal < 0) {
00671                 UART_PRINT("sl_FsRead error: %d\r", RetVal);
00672                 return RetVal;
00673             }
00674             offset += strlen(buffer);
00675         }
00676
00677     }
00678
00679     RetVal = sl_FsClose(fd, 0, 0, 0);
00680     if (RetVal < 0) {
00681         UART_PRINT("Error closing the file : %s\r", FS_DNCNTR);
00682     }
00683
00684     return (atoi(&buffer));
00685 }
```

3.21.2.4 st_read FileMode() int st_read FileMode ()

File System read Mode Function.

This function

1. Reads the Mode of the node from FS_MODE file

Returns

0

Definition at line 697 of file file_system.c.

```

00697     {
00698
00699     int offset = 0;
00700     int RetVal = 0;
00701     _i8 buffer[MAX_FILE_SIZE];
00702     int32_t fd;
00703
00704     fd = sl_FsOpen(FS_MODE, SL_FS_READ, 0);
00705     if (fd < 0) {
00706         UART_PRINT("\r\nError opening the file : %s\r", FS_MODE);
00707     }
00708
00709     while (RetVal=sl_FsRead(fd, offset, &buffer[0], MAX_FILE_SIZE)>1) {
00710         if (strlen(buffer) != 0) {
00711             if (RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE) {
00712                 break;
00713             }
00714             if (RetVal < 0) {
00715                 UART_PRINT("sl_FsRead error: %d\r", RetVal);
00716             }
00717         }
00718     }
00719
00720     RetVal = sl_FsClose(fd, 0, 0, 0);
00721     if (RetVal < 0) {
00722         UART_PRINT("Error closing the file : %s\r", FS_MODE);
00723     }
00724
00725     return (atoi(&buffer));
00726 }
```

```

00716         return RetVal;
00717     }
00718     offset += strlen(buffer);
00719 }
00720 }
00721
00722 UART_PRINT("READING MODE: %d\n\r", (atoi(&buffer)));
00723
00724 RetVal = sl_FsClose(fd, 0, 0, 0);
00725 if (RetVal < 0){
00726     UART_PRINT("Error closing the file : %s\n\r", FS_NBOOT);
00727 }
00728
00729 return (atoi(&buffer));
00730 }
```

3.21.2.5 st_readFileNBoot() int st_readFileNBoot ()

File System read NBoot Function.

This function

1. Reads the NBoot value from FS_NBOOT file. Is used to change to alternate of sending the sensors data

Returns

0

Definition at line 522 of file [file_system.c](#).

```

00522 {
00523
00524     int offset = 0;
00525     int RetVal = 0;
00526     _i8 buffer[MAX_FILE_SIZE];
00527     int32_t fd;
00528
00529     fd = sl_FsOpen(FS_NBOOT, SL_FS_READ, 0);
00530     if (fd < 0){
00531         UART_PRINT("\r\nError opening the file : %s\n\r", FS_NBOOT);
00532     }
00533
00534     while(RetVal=sl_FsRead(fd, offset, &buffer[0], MAX_FILE_SIZE)>1){
00535         if(strlen(buffer)!=0){
00536             if(RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE){
00537                 break;
00538             }
00539             if(RetVal < 0){
00540                 UART_PRINT("sl_FsRead error: %d\n\r", RetVal);
00541                 return RetVal;
00542             }
00543             offset += strlen(buffer);
00544         }
00545     }
00546
00547     RetVal = sl_FsClose(fd, 0, 0, 0);
00548     if (RetVal < 0){
00549         UART_PRINT("Error closing the file : %s\n\r", FS_NBOOT);
00550     }
00551
00552     return (atoi(&buffer));
00553 }
```

3.21.2.6 st_readFileNCycles() int st_readFileNCycles ()

File System read NCycles Function.

This function

1. Reads the number of cycles of the node value from FS_NCYCLES file.

Returns

0

Definition at line 565 of file [file_system.c](#).

```
00565     {
00566     int offset = 0;
00567     int RetVal = 0;
00568     _i8 buffer[MAX_FILE_SIZE];
00569     int32_t fd;
00570
00571     fd = sl_FsOpen(FS_NCYCLES, SL_FS_READ, 0);
00572     if (fd < 0) {
00573         UART_PRINT("\r\nError opening the file : %s\r\n", FS_NCYCLES);
00574     }
00575
00576     while(RetVal=sl_FsRead(fd, offset, &buffer[0], MAX_FILE_SIZE)>1) {
00577         if(strlen(buffer)!=0) {
00578             if(RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE) {
00579                 break;
00580             }
00581             if(RetVal < 0) {
00582                 UART_PRINT("sl_FsRead error: %d\r\n", RetVal);
00583                 return RetVal;
00584             }
00585             offset += strlen(buffer);
00586         }
00587     }
00588     UART_PRINT("READING NCYCLES: %d \r\n", atoi(&buffer));
00589
00590     RetVal = sl_FsClose(fd, 0, 0, 0);
00591     if (RetVal < 0) {
00592         UART_PRINT("Error closing the file : %s\r\n", FS_NCYCLES);
00593     }
00594
00595     return (atoi(&buffer));
00596 }
00597 }
```

3.21.2.7 st_readFileNfails() int st_readFileNfails ()

File System read Nfails Function.

This function

1. Reads the number of Lora OTAA connection fails from FS_NFAILS file. Is used to change to wifi if Lora OTAA fails n times.

Returns

0

Definition at line 477 of file [file_system.c](#).

```
00477
00478
00479     int offset = 0;
00480     int RetVal = 0;
00481     _i8 buffer[MAX_FILE_SIZE];
00482     int32_t fd;
00483
00484     fd = sl_FsOpen(FS_NFAILS, SL_FS_READ, 0);
00485     if (fd < 0) {
00486         UART_PRINT("\r\nnError opening the file : %s\n\r", FS_NFAILS);
00487     }
00488
00489     while (RetVal=sl_FsRead(fd, offset, &buffer[0], MAX_FILE_SIZE)>1)
00490     {
00491         if(strlen(buffer)!=0) {
00492             if(RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE) {
00493                 break;
00494             }
00495             if(RetVal < 0) {
00496                 UART_PRINT("sl_FsRead error: %d\n\r", RetVal);
00497                 return RetVal;
00498             }
00499             offset += strlen(buffer);
00500         }
00501     }
00502     UART_PRINT("READING NFAILS: %d \n\r", atoi(&buffer));
00503
00504     RetVal = sl_FsClose(fd, 0, 0, 0);
00505     if (RetVal < 0) {
00506         UART_PRINT("Error closing the file : %s\n\r", FS_NFAILS);
00507     }
00508
00509     return (atoi(&buffer));
00510 }
```

3.21.2.8 st_readFileNodeId() int st_readFileNodeId ()

File System read NodId Function.

This function

1. Reads the NodId from FS_NODEID file

Returns

0

Definition at line 829 of file [file_system.c](#).

```
00829
00830
00831     int offset = 0;
00832     int RetVal = 0;
00833     _i8 buffer[MAX_FILE_SIZE];
00834     int32_t fd;
00835
00836     fd = sl_FsOpen(FS_NODEID, SL_FS_READ, 0);
00837     if (fd < 0) {
00838         UART_PRINT("\r\nError opening the file : %s\n\r", FS_NODEID);
00839     }
00840
00841     while (RetVal=sl_FsRead(fd, offset, &buffer[0], MAX_FILE_SIZE)>1) {
00842         if(strlen(buffer)!=0) {
00843             if(RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE) {
00844                 break;
00845             }
00846             if(RetVal < 0) {
```

```

00847     UART_PRINT("sl_FsRead error: %d\n\r", RetVal);
00848     return RetVal;
00849 }
00850 offset += strlen(buffer);
00851 }
00852 }
00853
00854 UART_PRINT("READING NODEID: %d \n\r", atoi(&buffer));
00855
00856 RetVal = sl_FsClose(fd, 0, 0, 0);
00857 if (RetVal < 0) {
00858     UART_PRINT("Error closing the file : %s\n\r", FS_NODEID);
00859 }
00860 return (atoi(&buffer));
00861 }

```

3.21.2.9 st_readFileSSID()

```
int st_readFileSSID (
    unsigned char * ssid)
```

File System read SSID Function.

This function

1. Reads the wifi SSID from FS_SSID file

Parameters

<i>unsigned</i>	char pointer to ssid
-----------------	----------------------

Returns

0

Definition at line 441 of file [file_system.c](#).

```

00441     int RetVal = 0;
00442     int32_t fd;
00443
00444     fd = sl_FsOpen(FS_SSID, SL_FS_READ, 0);
00445     if (fd < 0) {
00446         UART_PRINT("\r\nError opening the file : %s\n\r", FS_SSID);
00447     }
00448
00449     RetVal = sl_FsRead(fd, 0, ssid, strlen(ssid));
00450
00451     if(RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE) {
00452         return RetVal;
00453     }else if (RetVal < 0) {
00454         UART_PRINT("sl_FsRead error: %d\n\r", RetVal);
00455         return RetVal;
00456     }
00457
00458     RetVal = sl_FsClose(fd, 0, 0, 0);
00459     if (RetVal < 0) {
00460         UART_PRINT("Error closing the file : %s\n\r", FS_SSID);
00461     }
00462
00463     return 0;
00464 }
00465 }
```

3.21.2.10 st_readFileUpCntr() int st_readFileUpCntr ()

File System read UpCntr Function.

This function

1. Reads the Up Counter parameter of Lora GW from FS_UPCNTR file

Returns

0

Definition at line 609 of file [file_system.c](#).

```
00609     {
00610
00611     int offset = 0;
00612     int RetVal = 0;
00613     _i8 buffer[MAX_FILE_SIZE];
00614     int32_t fd;
00615
00616     fd = sl_FsOpen(FS_UPCNTR, SL_FS_READ, 0);
00617     if (fd < 0){
00618         UART_PRINT("\r\nError opening the file : %s\n\r", FS_UPCNTR);
00619     }
00620
00621     while(RetVal=sl_FsRead(fd, offset, &buffer[0], MAX_FILE_SIZE)>1){
00622         if(strlen(buffer)!=0){
00623             if(RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE){
00624                 break;
00625             }
00626             if(RetVal < 0){
00627                 UART_PRINT("sl_FsRead error: %d\n\r", RetVal);
00628                 return RetVal;
00629             }
00630             offset += strlen(buffer);
00631         }
00632     }
00633
00634     RetVal = sl_FsClose(fd, 0, 0, 0);
00635     if (RetVal < 0){
00636         UART_PRINT("Error closing the file : %s\n\r", FS_UPCNTR);
00637     }
00638
00639     return (atoi(&buffer));
00640 }
```

3.21.2.11 st_readFileWakeUp() int st_readFileWakeUp ()

File System read WakeUp Interval Function.

This function

1. Reads the wake up interval value of the node from FS_WAKEUP file

Returns

0

Definition at line 786 of file file_system.c.

```

00786     {
00787
00788     int offset = 0;
00789     int RetVal = 0;
00790     _i8 buffer[MAX_FILE_SIZE];
00791     int32_t fd;
00792
00793     fd = sl_FsOpen(FS_WAKEUP, SL_FS_READ, 0);
00794     if (fd < 0) {
00795         UART_PRINT("\r\nError opening the file : %s\r\n", FS_WAKEUP);
00796     }
00797
00798     while (RetVal=sl_FsRead(fd, offset, &buffer[0], MAX_FILE_SIZE)>1) {
00799         if(strlen(buffer)!=0) {
00800             if(RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE) {
00801                 break;
00802             }
00803             if(RetVal < 0) {
00804                 UART_PRINT("sl_FsRead error: %d\r\n", RetVal);
00805                 return RetVal;
00806             }
00807             offset += strlen(buffer);
00808         }
00809     }
00810     UART_PRINT("\r\nREADING WAKEUPINTERVAL: %d\r\n", atoi(&buffer));
00811
00812     RetVal = sl_FsClose(fd, 0, 0, 0);
00813     if (RetVal < 0) {
00814         UART_PRINT("Error closing the file : %s\r\n", FS_WAKEUP);
00815     }
00816     return (atoi(&buffer));
00817 }
```

3.21.2.12 writeChangeWakeUp() int writeChangeWakeUp (

uint8_t ChangeWakeUp)

File System write ChangeWakeUp Function.

This function

1. Writes the ChangeWakeUp value on FS_CHANGEWAKEUP file. Is used to determine if is necessary to change the node wake up time.

Parameters

in	<i>uint8_t</i>	ChangeWakeUp
	<i>t</i>	

Returns*_i32 offset***Definition at line 322 of file file_system.c.**

```

00322
00323
00324     int RetVal = 0;
00325     _i32 offset = 0;
00326     _i32 fd;
00327     unsigned char changewakeup[32];
00328
00329     fd = sl_FsOpen(FS_CHANGEWAKEUP, SL_FS_WRITE, 0);
00330     if (fd < 0) {
00331         UART_PRINT("Error opening the file : %s\r\n", FS_CHANGEWAKEUP);
00332     }
00333
00334     RetVal = sl_FsWrite(fd, offset, &changewakeup, 32);
00335     if (RetVal < 0) {
00336         UART_PRINT("Error writing the file : %s\r\n", FS_CHANGEWAKEUP);
00337     }
00338
00339     RetVal = sl_FsClose(fd, 0, 0, 0);
00340     if (RetVal < 0) {
00341         UART_PRINT("Error closing the file : %s\r\n", FS_CHANGEWAKEUP);
00342     }
00343
00344     return RetVal;
00345 }
```

```

00328     sprintf(&changewakeup, "%d", ChangeWakeUp );
00329
00330     fd = sl_FsOpen(FS_CHANGEWAKEUP, SL_FS_OVERWRITE, 0);
00331     if (fd < 0){
00332         UART_PRINT("Error opening the file : %s\n\r", FS_CHANGEWAKEUP);
00333     }
00334     else{
00335         RetVal = sl_FsWrite(fd, 0, changewakeup, strlen(changewakeup));
00336         if (RetVal <= 0){
00337             UART_PRINT("Writing error: %d\n\r", RetVal);
00338             return RetVal;
00339         }
00340     }
00341
00342     RetVal = sl_FsClose(fd, 0, 0, 0);
00343     if (RetVal < 0){
00344         UART_PRINT("Error closing the file : %s\n\r", FS_CHANGEWAKEUP);
00345     }
00346 }
00347 return offset;
00348 }
```

3.21.2.13 writeDnCntr() int writeDnCntr (

	uint32_t Dnctr)
--	------------------

File System write NFails Function.

This function

1. Writes the number of LoRa OTAA connection fails on NFails file

Parameters

in	<i>NFails</i>	Number of LoRa OTAA connection fails
----	---------------	--------------------------------------

Returns

_i32 offset

Definition at line 242 of file [file_system.c](#).

```

00242
00243
00244     int RetVal = 0;
00245     _i32 offset = 0;
00246     _i32 fd;
00247     unsigned char dnctr[32];
00248
00249     sprintf(&dnctr, "%d", Dnctr);
00250
00251     fd = sl_FsOpen(FS_DNCNTR, SL_FS_OVERWRITE, 0);
00252     if (fd < 0){
00253         UART_PRINT("Error opening the file : %s\n\r", FS_DNCNTR);
00254     }
00255     else{
00256         RetVal = sl_FsWrite(fd, 0, dnctr, strlen(dnctr));
00257         if (RetVal <= 0){
00258             UART_PRINT("Writing error: %d\n\r", RetVal);
00259             return RetVal;
00260         }
00261
00262         RetVal = sl_FsClose(fd, 0, 0, 0);
00263         if (RetVal < 0){
00264             UART_PRINT("Error closing the file : %s\n\r", FS_DNCNTR);
00265         }
00266     }
00267     return offset;
00268 }
```

3.21.2.14 writeMode() int writeMode (uint8_t Mode)

File System write Mode Function.

This function

1. Writes the working mode of the node (Lora or Wifi) on FS_MODE file

Parameters

in	Mode	working mode of the node (0 for LoRa, 2 for Wifi)
----	------	---

Returns

_i32 offset

Definition at line 282 of file [file_system.c](#).

```
00282     {
00283
00284     int RetVal = 0;
00285     _i32 offset = 0;
00286     _i32 fd;
00287     unsigned char mode[32];
00288
00289     sprintf(&mode, "%d", Mode );
00290
00291     fd = sl_FsOpen(FS_MODE, SL_FS_OVERWRITE, 0);
00292     if (fd < 0){
00293         UART_PRINT("Error opening the file : %s\n\r", FS_MODE);
00294     }
00295     else{
00296         RetVal = sl_FsWrite(fd, 0, mode, strlen(mode));
00297         if (RetVal <= 0){
00298             UART_PRINT("Writing error: %d\n\r", RetVal);
00299             return RetVal;
00300         }
00301
00302         RetVal = sl_FsClose(fd, 0, 0, 0);
00303         if (RetVal < 0){
00304             UART_PRINT("Error closing the file : %s\n\r", FS_MODE);
00305         }
00306     }
00307     return offset;
00308 }
```

3.21.2.15 writeNBoot() int writeNBoot (uint8_t FirstBoot)

File System write NBoot Function.

This function

1. Writes the number of NBoot on FS_NBOOT file. This value is used to alternate the data sent between sensors.

Parameters

in	<i>uint8_t</i>	FirstBoot yes or no
	<i>_t</i>	

Returns*_i32 offset***Definition at line 118 of file file_system.c.**

```

00118
00119
00120     int RetVal = 0;
00121     _i32 offset = 0;
00122     _i32 fd;
00123     unsigned char fboot[32];
00124
00125     sprintf(&fboot,"%d",FirstBoot );
00126
00127     fd = sl_FsOpen(FS_NBOOT, SL_FS_OVERWRITE, 0);
00128     if (fd < 0){
00129         UART_PRINT("Error opening the file : %s\n\r", FS_NBOOT);
00130     }
00131     else{
00132         RetVal = sl_FsWrite(fd, 0, fboot, strlen(fboot));
00133         if (RetVal <= 0){
00134             UART_PRINT("Writing error: %d\n\r", RetVal);
00135             return RetVal;
00136         }
00137
00138         RetVal = sl_FsClose(fd, 0, 0, 0);
00139         if (RetVal < 0){
00140             UART_PRINT("Error closing the file : %s\n\r", FS_NBOOT);
00141         }
00142     }
00143     return offset;
00144 }
```

3.21.2.16 writeNCycles() *int writeNCycles (**uint8_t NCycles)*

File System write NCycles Function.

This function

1. Writes the number of cycles of the node on FS_NCYCLES file

Parameters

in	<i>uint8_t</i>	NCycles Number of cycles
	<i>_t</i>	

Returns*_i32 offset***Definition at line 158 of file file_system.c.**

```

00158
00159 }
```

```

00160     int RetVal = 0;
00161     _i32 offset = 0;
00162     _i32 fd;
00163     unsigned char ncycles[32];
00164
00165     sprintf(&ncycles, "%d", NCycles );
00166
00167     fd = sl_FsOpen(FS_NCYCLES, SL_FS_OVERWRITE, 0);
00168     if (fd < 0){
00169         UART_PRINT("Error opening the file : %s\n\r", FS_NCYCLES);
00170     }
00171     else{
00172         RetVal = sl_FsWrite(fd, 0, ncycles, strlen(ncycles));
00173         if (RetVal <= 0){
00174             UART_PRINT("Writing error: %d\n\r", RetVal);
00175             return RetVal;
00176         }
00177
00178         RetVal = sl_FsClose(fd, 0, 0, 0);
00179         if (RetVal < 0){
00180             UART_PRINT("Error closing the file : %s\n\r", FS_NCYCLES);
00181         }
00182     }
00183     return offset;
00184 }
```

3.21.2.17 writeNFails() int writeNFails (uint16_t NFails)

File System write NFails Function.

This function

1. Writes the number of LoRa OTAA connection fails on NFails file

Parameters

in	<i>uint16_t</i>	FS_NFAILS Number of LoRa OTAA connection fails
----	-----------------	--

Returns

_i32 offset

Definition at line 79 of file [file_system.c](#).

```

00079
00080
00081     int RetVal = 0;
00082     _i32 offset = 0;
00083     _i32 fd;
00084     unsigned char fails[32];
00085
00086     sprintf(&fails, "%d", NFails );
00087
00088     fd = sl_FsOpen(FS_NFAILS, SL_FS_OVERWRITE, 0);
00089     if (fd < 0){
00090         UART_PRINT("Error opening the file : %s\n\r", FS_NFAILS);
00091     }
00092     else{
00093         RetVal = sl_FsWrite(fd, 0, fails, strlen(fails));
00094         if (RetVal <= 0){
00095             UART_PRINT("Writing error: %d\n\r", RetVal);
00096             return RetVal;
00097         }
00098         RetVal = sl_FsClose(fd, 0, 0, 0);
00099         if (RetVal < 0){
00100             UART_PRINT("Error closing the file : %s\n\r", FS_NFAILS);
```

```

00101         }
00102     }
00103     return offset;
00104 }
```

3.21.2.18 writeNodeid()

```
int writeNodeid (
    uint16_t NodeId )
```

File System write NodeId Function.

This function

1. Writes the NodeId on NodeId file

Parameters

in	<i>uint16_t</i>	NodeId
----	-----------------	--------

Returns

_i32 offset

Definition at line 402 of file [file_system.c](#).

```

00402
00403
00404     int RetVal = 0;
00405     _i32 offset = 0;
00406     _i32 fd;
00407     unsigned char nodeid[32];
00408
00409     sprintf(&nodeid,"%d",NodeId );
00410
00411     fd = sl_FsOpen(FS_NODEID, SL_FS_OVERWRITE, 0);
00412     if (fd < 0){
00413         UART_PRINT("Error opening the file : %s\n\r", FS_NODEID);
00414     }
00415     else{
00416         RetVal = sl_FsWrite(fd, 0, nodeid, strlen(nodeid));
00417         if (RetVal <= 0){
00418             UART_PRINT("Writing error: %d\n\r" ,RetVal);
00419             return RetVal;
00420         }
00421         RetVal = sl_FsClose(fd, 0, 0, 0);
00422         if (RetVal < 0){
00423             UART_PRINT("Error closing the file : %s\n\r", FS_NODEID);
00424         }
00425     }
00426     return offset;
00427 }
```

3.21.2.19 writeSSID()

```
int writeSSID (
    unsigned char * SSID )
```

File System write SSID Function.

This function

1. Writes the SSID of Wifi network on FS_SSID file

Parameters

in	<i>unsigned</i>	char SSID Pointer to SSID
----	-----------------	---------------------------

Returns*_i32 offset***Definition at line 39 of file [file_system.c](#).**

```

00039
00040
00041     int RetVal = 0;
00042     _i32 offset = 0;
00043     _i32 fd;
00044     unsigned char ssid[13];
00045
00046     sprintf(&ssid,"%s",SSID );
00047
00048     fd = sl_FsOpen(FS_SSID, SL_FS_OVERWRITE, 0);
00049     if (fd < 0){
00050         UART_PRINT("Error opening the file : %s\n\r", FS_SSID);
00051     }
00052     else{
00053         RetVal = sl_FsWrite(fd, 0, ssid, strlen(ssid));
00054         if (RetVal <= 0){
00055             UART_PRINT("Writing error: %d\n\r",RetVal);
00056             return RetVal;
00057         }
00058
00059         RetVal = sl_FsClose(fd, 0, 0, 0);
00060         if (RetVal < 0){
00061             UART_PRINT("Error closing the file : %s\n\r", FS_SSID);
00062         }
00063     }
00064     return offset;
00065 }
```

3.21.2.20 writeUpCntr() *int writeUpCntr (*
uint32_t Upctr)

File System write Up Counter Function.

This function

1. Writes the value of UpCntr LoRa parameter on FS_UPCNTR file

Parameters

in	<i>uint32_t</i>	NFails UpCntr LoRa parameter value
----	-----------------	------------------------------------

Returns*_i32 offset***Definition at line 198 of file [file_system.c](#).**

```

00198
00199
00200     int RetVal = 0;
```

```

00201     _i32 offset = 0;
00202     _i32 fd;
00203     unsigned char upctr[32];
00204
00205     sprintf(&upctr, "%d", Upctr );
00206
00207     fd = sl_FsOpen(FS_UPCNTR, SL_FS_OVERWRITE, 0);
00208     if (fd < 0)
00209     {
00210         UART_PRINT("Error opening the file : %s\n\r", FS_UPCNTR);
00211     }
00212     else
00213     {
00214        RetVal = sl_FsWrite(fd, 0, upctr, strlen(upctr));
00215         if (RetVal <= 0)
00216         {
00217             UART_PRINT("Writing error: %d\n\r", RetVal);
00218             return RetVal;
00219         }
00220
00221        RetVal = sl_FsClose(fd, 0, 0, 0);
00222         if (RetVal < 0)
00223         {
00224             UART_PRINT("Error closing the file : %s\n\r", FS_UPCNTR);
00225         }
00226     }
00227     return offset;
00228 }
```

3.21.2.21 writeWakeUp() int writeWakeUp (uint16_t WakeUpInterval)

File System write WakeUp Function.

This function

1. Writes the wake up interval time of the node on FS_WAKEUP file

Parameters

in	<i>uint16_t</i>	WakeUpInterval Time(s) of the wake up interval
	<i>_t</i>	

Returns

_i32 offset

Definition at line 362 of file [file_system.c](#).

```

00362
00363
00364     int RetVal = 0;
00365     _i32 offset = 0;
00366     _i32 fd;
00367     unsigned char wakeup[32];
00368
00369     sprintf(&wakeup, "%d", WakeUpInterval );
00370
00371     fd = sl_FsOpen(FS_WAKEUP, SL_FS_OVERWRITE, 0);
00372     if (fd < 0){
00373         UART_PRINT("Error opening the file : %s\n\r", FS_WAKEUP);
00374     }
00375     else{
00376         RetVal = sl_FsWrite(fd, 0, wakeup, strlen(wakeup));
00377         if (RetVal <= 0){
00378             UART_PRINT("Writing error: %d\n\r", RetVal);
00379             return RetVal;
00380         }
00381     }
```

```

00381     RetVal = sl_FsClose(fd, 0, 0, 0);
00382     if (RetVal < 0){
00383         UART_PRINT("Error closing the file : %s\n\r", FS_WAKEUP);
00384     }
00385 }
00386 }
00387 return offset;
00388 }
```

3.21.3 Variable Documentation

3.21.3.1 uart0 UART_Handle uart0

Definition at line 65 of file [STARPORTS_App.c](#).

3.22 file_system.c

```

00001 //*****
00010 //***** INCLUDES *****
00011 //***** INCLUDES *****
00012 //***** INCLUDES *****
00013 #include <ti/devices/cc32xx/inc/hw_types.h>
00014 #include <ti/devices/cc32xx/driverlib/prcm.h>
00015 #include "hal_UART.h"
00016 #include "file_system.h"
00017 //*****
00018 //*****
00019 //***** GLOBALS *****
00020 //*****
00021 extern UART_Handle uart0;
00022 //*****
00023 //*****
00024 //***** FUNCTIONS *****
00025 //*****
00026 //*****
00027 //*****
00028 //*****
00029 //*****
00030 //*****
00031 //*****
00032 //*****
00033 //*****
00034 int writeSSID(unsigned char *SSID){
00040
00041     int RetVal = 0;
00042     _i32 offset = 0;
00043     _i32 fd;
00044     unsigned char ssid[13];
00045
00046     sprintf(&ssid,"%s",SSID );
00047
00048     fd = sl_FsOpen(FS_SSID, SL_FS_OVERWRITE, 0);
00049     if (fd < 0){
00050         UART_PRINT("Error opening the file : %s\n\r", FS_SSID);
00051     }
00052     else{
00053         RetVal = sl_FsWrite(fd, 0, ssid, strlen(ssid));
00054         if (RetVal <= 0){
00055             UART_PRINT("Writing error: %d\n\r", RetVal);
00056             return RetVal;
00057         }
00058
00059         RetVal = sl_FsClose(fd, 0, 0, 0);
00060         if (RetVal < 0){
00061             UART_PRINT("Error closing the file : %s\n\r", FS_SSID);
00062         }
00063     }
00064     return offset;
00065 }
00066 //*****
00067 //*****
00068 //*****
00069 //*****
00070 //*****
00071 //*****
00072 //*****
00073 //*****
00074 int writeNfails(uint16_t Nfails){
00080     int RetVal = 0;
```

```

00082     _i32 offset = 0;
00083     _i32 fd;
00084     unsigned char fails[32];
00085
00086     sprintf(&fails,"%d",Nfails );
00087
00088     fd = sl_FsOpen(FS_NFAILS, SL_FS_OVERWRITE, 0);
00089     if (fd < 0){
00090         UART_PRINT("Error opening the file : %s\n\r", FS_NFAILS);
00091     }
00092     else{
00093         RetVal = sl_FsWrite(fd, 0, fails, strlen(fails));
00094         if (RetVal <= 0){
00095             UART_PRINT("Writing error: %d\n\r" ,RetVal);
00096             return RetVal;
00097         }
00098        RetVal = sl_FsClose(fd, 0, 0, 0);
00099         if (RetVal < 0){
00100             UART_PRINT("Error closing the file : %s\n\r", FS_NFAILS);
00101         }
00102     }
00103     return offset;
00104 }
00105
00106 //*****
00107 //
00116 //
00117 //*****
00118 int writeNBoot(uint8_t FirstBoot){
00119
00120     int RetVal = 0;
00121     _i32 offset = 0;
00122     _i32 fd;
00123     unsigned char fboot[32];
00124
00125     sprintf(&fboot,"%d",FirstBoot );
00126
00127     fd = sl_FsOpen(FS_NBOOT, SL_FS_OVERWRITE, 0);
00128     if (fd < 0){
00129         UART_PRINT("Error opening the file : %s\n\r", FS_NBOOT);
00130     }
00131     else{
00132         RetVal = sl_FsWrite(fd, 0, fboot, strlen(fboot));
00133         if (RetVal <= 0){
00134             UART_PRINT("Writing error: %d\n\r" ,RetVal);
00135             return RetVal;
00136         }
00137
00138        RetVal = sl_FsClose(fd, 0, 0, 0);
00139         if (RetVal < 0){
00140             UART_PRINT("Error closing the file : %s\n\r", FS_NBOOT);
00141         }
00142     }
00143     return offset;
00144 }
00145
00146 //*****
00147 //
00156 //
00157 //*****
00158 int writeNCycles(uint8_t NCycles){
00159
00160     int RetVal = 0;
00161     _i32 offset = 0;
00162     _i32 fd;
00163     unsigned char ncycles[32];
00164
00165     sprintf(&ncycles,"%d",NCycles );
00166
00167     fd = sl_FsOpen(FS_NCYCLES, SL_FS_OVERWRITE, 0);
00168     if (fd < 0){
00169         UART_PRINT("Error opening the file : %s\n\r", FS_NCYCLES);
00170     }
00171     else{
00172         RetVal = sl_FsWrite(fd, 0, ncycles, strlen(ncycles));
00173         if (RetVal <= 0){
00174             UART_PRINT("Writing error: %d\n\r" ,RetVal);
00175             return RetVal;
00176         }
00177
00178        RetVal = sl_FsClose(fd, 0, 0, 0);
00179         if (RetVal < 0){
00180             UART_PRINT("Error closing the file : %s\n\r", FS_NCYCLES);
00181         }
00182     }
00183     return offset;
00184 }

```

```

00185 //*****
00186 //*****
00187 //
00196 //
00197 //*****
00198 int writeUpCntr(uint32_t Upctr) {
00199
00200     int RetVal = 0;
00201     _i32 offset = 0;
00202     _i32 fd;
00203     unsigned char upctr[32];
00204
00205     sprintf(&upctr, "%d", Upctr );
00206
00207     fd = sl_FsOpen(FS_UPCNTR, SL_FS_OVERWRITE, 0);
00208     if (fd < 0)
00209     {
00210         UART_PRINT("Error opening the file : %s\n\r", FS_UPCNTR);
00211     }
00212     else
00213     {
00214         RetVal = sl_FsWrite(fd, 0, upctr, strlen(upctr));
00215         if (RetVal <= 0)
00216         {
00217             UART_PRINT("Writing error: %d\n\r", RetVal);
00218             return RetVal;
00219         }
00220
00221         RetVal = sl_FsClose(fd, 0, 0, 0);
00222         if (RetVal < 0)
00223         {
00224             UART_PRINT("Error closing the file : %s\n\r", FS_UPCNTR);
00225         }
00226     }
00227     return offset;
00228 }
00229
00230 //*****
00231 //
00240 //
00241 //*****
00242 int writeDnCntr(uint32_t Dnctr) {
00243
00244     int RetVal = 0;
00245     _i32 offset = 0;
00246     _i32 fd;
00247     unsigned char dnctr[32];
00248
00249     sprintf(&dnctr, "%d", Dnctr );
00250
00251     fd = sl_FsOpen(FS_DNCNTR, SL_FS_OVERWRITE, 0);
00252     if (fd < 0){
00253         UART_PRINT("Error opening the file : %s\n\r", FS_DNCNTR);
00254     }
00255     else{
00256         RetVal = sl_FsWrite(fd, 0, dnctr, strlen(dnctr));
00257         if (RetVal <= 0){
00258             UART_PRINT("Writing error: %d\n\r", RetVal);
00259             return RetVal;
00260         }
00261
00262         RetVal = sl_FsClose(fd, 0, 0, 0);
00263         if (RetVal < 0){
00264             UART_PRINT("Error closing the file : %s\n\r", FS_DNCNTR);
00265         }
00266     }
00267     return offset;
00268 }
00269
00270 //*****
00271 //
00280 //
00281 //*****
00282 int writeMode(uint8_t Mode) {
00283
00284     int RetVal = 0;
00285     _i32 offset = 0;
00286     _i32 fd;
00287     unsigned char mode[32];
00288
00289     sprintf(&mode, "%d", Mode );
00290
00291     fd = sl_FsOpen(FS_MODE, SL_FS_OVERWRITE, 0);
00292     if (fd < 0){
00293         UART_PRINT("Error opening the file : %s\n\r", FS_MODE);
00294     }
00295     else{

```

```
00296     RetVal = sl_FsWrite(fd, 0, mode, strlen(mode));
00297     if (RetVal <= 0){
00298         UART_PRINT("Writing error: %d\n\r", RetVal);
00299         return RetVal;
00300     }
00301
00302     RetVal = sl_FsClose(fd, 0, 0, 0);
00303     if (RetVal < 0){
00304         UART_PRINT("Error closing the file : %s\n\r", FS_MODE);
00305     }
00306 }
00307 return offset;
00308 }
00309
00310 //*****
00311 //
00312 //
00313 //*****
00314 int writeChangeWakeUp(uint8_t ChangeWakeUp) {
00315
00316     int RetVal = 0;
00317     _i32 offset = 0;
00318     _i32 fd;
00319     unsigned char changewakeup[32];
00320
00321     sprintf(&changewakeup, "%d", ChangeWakeUp );
00322
00323     fd = sl_FsOpen(FS_CHANGEWAKEUP, SL_FS_OVERWRITE, 0);
00324     if (fd < 0){
00325         UART_PRINT("Error opening the file : %s\n\r", FS_CHANGEWAKEUP);
00326     }
00327     else{
00328         RetVal = sl_FsWrite(fd, 0, changewakeup, strlen(changewakeup));
00329         if (RetVal <= 0){
00330             UART_PRINT("Writing error: %d\n\r", RetVal);
00331             return RetVal;
00332         }
00333
00334         RetVal = sl_FsClose(fd, 0, 0, 0);
00335         if (RetVal < 0){
00336             UART_PRINT("Error closing the file : %s\n\r", FS_CHANGEWAKEUP);
00337         }
00338     }
00339 }
00340
00341 return offset;
00342 }
00343
00344 }
00345
00346 }
00347
00348 }
00349
00350 //*****
00351 //
00352 //
00353 //*****
00354 int writeWakeUp(uint16_t WakeUpInterval) {
00355
00356     int RetVal = 0;
00357     _i32 offset = 0;
00358     _i32 fd;
00359     unsigned char wakeup[32];
00360
00361     sprintf(&wakeup, "%d", WakeUpInterval );
00362
00363     fd = sl_FsOpen(FS_WAKEUP, SL_FS_OVERWRITE, 0);
00364     if (fd < 0){
00365         UART_PRINT("Error opening the file : %s\n\r", FS_WAKEUP);
00366     }
00367     else{
00368         RetVal = sl_FsWrite(fd, 0, wakeup, strlen(wakeup));
00369         if (RetVal <= 0){
00370             UART_PRINT("Writing error: %d\n\r", RetVal);
00371             return RetVal;
00372         }
00373
00374         RetVal = sl_FsClose(fd, 0, 0, 0);
00375         if (RetVal < 0){
00376             UART_PRINT("Error closing the file : %s\n\r", FS_WAKEUP);
00377         }
00378     }
00379 }
00380
00381 return offset;
00382 }
00383
00384 }
00385
00386 }
00387
00388 }
00389
00390 //*****
00391 //
00392 //
00393 //*****
00394 int writeNodeId(uint16_t NodeId) {
00395
00396     int RetVal = 0;
00397     _i32 offset = 0;
00398     _i32 fd;
```

```

00407     unsigned char nodeid[32];
00408
00409     sprintf(&nodeid,"%d",NodeId );
00410
00411     fd = sl_FsOpen(FS_NODEID, SL_FS_OVERWRITE, 0);
00412     if (fd < 0){
00413         UART_PRINT("Error opening the file : %s\n\r", FS_NODEID);
00414     }
00415     else{
00416         RetVal = sl_FsWrite(fd, 0, nodeid, strlen(nodeid));
00417         if (RetVal <= 0){
00418             UART_PRINT("Writing error: %d\n\r", RetVal);
00419             return RetVal;
00420         }
00421         RetVal = sl_FsClose(fd, 0, 0, 0);
00422         if (RetVal < 0){
00423             UART_PRINT("Error closing the file : %s\n\r", FS_NODEID);
00424         }
00425     }
00426     return offset;
00427 }
00428
00429 //*****
00430 //
00431 //
00432 //*****
00433
00434
00435
00436
00437
00438
00439
00440 //*****
00441 int st_readFileSSID(unsigned char *ssid){
00442     int RetVal = 0;
00443     int32_t fd;
00444
00445     fd = sl_FsOpen(FS_SSID, SL_FS_READ, 0);
00446     if (fd < 0) {
00447         UART_PRINT("\r\nError opening the file : %s\n\r", FS_SSID);
00448     }
00449
00450     RetVal = sl_FsRead(fd, 0, ssid, strlen(ssid));
00451
00452     if(RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE) {
00453         return RetVal;
00454     }else if (RetVal < 0) {
00455         UART_PRINT("sl_FsRead error: %d\n\r", RetVal);
00456         return RetVal;
00457     }
00458
00459     RetVal = sl_FsClose(fd, 0, 0, 0);
00460     if (RetVal < 0) {
00461         UART_PRINT("Error closing the file : %s\n\r", FS_SSID);
00462     }
00463
00464     return 0;
00465 }
00466
00467 //*****
00468 //
00469 //
00470 //*****
00471
00472
00473
00474
00475
00476 //*****
00477 int st_readFileNfails(){
00478
00479     int offset = 0;
00480     int RetVal = 0;
00481     _i8 buffer[MAX_FILE_SIZE];
00482     int32_t fd;
00483
00484     fd = sl_FsOpen(FS_NFAILS, SL_FS_READ, 0);
00485     if (fd < 0){
00486         UART_PRINT("\r\nError opening the file : %s\n\r", FS_NFAILS);
00487     }
00488
00489     while(RetVal=sl_FsRead(fd, offset, &buffer[0], MAX_FILE_SIZE)>1)
00490     {
00491         if(strlen(buffer)!=0){
00492             if(RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE){
00493                 break;
00494             }
00495             if(RetVal < 0){
00496                 UART_PRINT("sl_FsRead error: %d\n\r", RetVal);
00497                 return RetVal;
00498             }
00499             offset += strlen(buffer);
00500         }
00501     }
00502     UART_PRINT("READING NFAILS: %d \n\r", atoi(&buffer));
00503
00504     RetVal = sl_FsClose(fd, 0, 0, 0);
00505     if (RetVal < 0){
00506         UART_PRINT("Error closing the file : %s\n\r", FS_NFAILS);
00507     }

```

```
00508     return (atoi(&buffer));
00509 }
00510 */
00511 //*****
00512 /**
00513 */
00520 /**
00521 //*****
00522 int st_readFileNBoot(){
00523
00524     int offset = 0;
00525     int RetVal = 0;
00526     _i8 buffer[MAX_FILE_SIZE];
00527     int32_t fd;
00528
00529     fd = sl_FsOpen(FS_NBOOT, SL_FS_READ, 0);
00530     if (fd < 0){
00531         UART_PRINT("\r\nError opening the file : %s\r\n", FS_NBOOT);
00532     }
00533
00534     while(RetVal=sl_FsRead(fd, offset, &buffer[0], MAX_FILE_SIZE)>1){
00535         if(strlen(buffer)!=0){
00536             if(RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE){
00537                 break;
00538             }
00539             if(RetVal < 0){
00540                 UART_PRINT("sl_FsRead error: %d\r\n", RetVal);
00541                 return RetVal;
00542             }
00543             offset += strlen(buffer);
00544         }
00545     }
00546
00547     RetVal = sl_FsClose(fd, 0, 0, 0);
00548     if (RetVal < 0){
00549         UART_PRINT("Error closing the file : %s\r\n", FS_NBOOT);
00550     }
00551
00552     return (atoi(&buffer));
00553 }
00554
00555 //*****
00556 /**
00563 /**
00564 //*****
00565 int st_readFileNCycles(){
00566
00567     int offset = 0;
00568     int RetVal = 0;
00569     _i8 buffer[MAX_FILE_SIZE];
00570     int32_t fd;
00571
00572     fd = sl_FsOpen(FS_NCYCLES, SL_FS_READ, 0);
00573     if (fd < 0){
00574         UART_PRINT("\r\nError opening the file : %s\r\n", FS_NCYCLES);
00575     }
00576
00577     while(RetVal=sl_FsRead(fd, offset, &buffer[0], MAX_FILE_SIZE)>1){
00578         if(strlen(buffer)!=0){
00579             if(RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE){
00580                 break;
00581             }
00582             if(RetVal < 0){
00583                 UART_PRINT("sl_FsRead error: %d\r\n", RetVal);
00584                 return RetVal;
00585             }
00586             offset += strlen(buffer);
00587         }
00588     }
00589     UART_PRINT("READING NCYCLES: %d \r\n", atoi(&buffer));
00590
00591     RetVal = sl_FsClose(fd, 0, 0, 0);
00592     if (RetVal < 0){
00593         UART_PRINT("Error closing the file : %s\r\n", FS_NCYCLES);
00594     }
00595
00596     return (atoi(&buffer));
00597 }
00598
00599 //*****
00600 /**
00607 /**
00608 //*****
00609 int st_readFileUpCntr(){
00610
00611     int offset = 0;
00612     int RetVal = 0;
```

```
00613     _i8 buffer[MAX_FILE_SIZE];
00614     int32_t fd;
00615
00616     fd = sl_FsOpen(FS_UPCNTR, SL_FS_READ, 0);
00617     if (fd < 0) {
00618         UART_PRINT
```

00712 **break**

```

00811    RetVal = sl_FsClose(fd, 0, 0, 0);
00812     if (RetVal < 0){
00813         UART_PRINT("Error closing the file : %s\n\r", FS_WAKEUP);
00814     }
00815     return (atoi(&buffer));
00816 }
00817 }
00818
00819 //*****
00820 //
00821 //
00822 //*****
00823 int st_readFileNodeId(){
00824
00825     int offset = 0;
00826     int RetVal = 0;
00827     _i8 buffer[MAX_FILE_SIZE];
00828     int32_t fd;
00829
00830     fd = sl_FsOpen(FS_NODEID, SL_FS_READ, 0);
00831     if (fd < 0){
00832         UART_PRINT("\r\nError opening the file : %s\n\r", FS_NODEID);
00833     }
00834
00835     while(RetVal=sl_FsRead(fd, offset, &buffer[0], MAX_FILE_SIZE)>1){
00836         if(strlen(buffer)!=0){
00837             if(RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE){
00838                 break;
00839             }
00840             if(RetVal < 0){
00841                 UART_PRINT("sl_FsRead error: %d\n\r", RetVal);
00842                 return RetVal;
00843             }
00844             offset += strlen(buffer);
00845         }
00846     }
00847     UART_PRINT("READING NODEID: %d \n\r", atoi(&buffer));
00848
00849     RetVal = sl_FsClose(fd, 0, 0, 0);
00850     if (RetVal < 0){
00851         UART_PRINT("Error closing the file : %s\n\r", FS_NODEID);
00852     }
00853     return (atoi(&buffer));
00854 }
00855
00856
00857
00858
00859
00860
00861 }
00862
00863 //*****
00864 //
00865 //
00866 //*****
00867 int deleteFile(){
00868
00869     sl_FsDel(FS_UPCNTR, 0);
00870     st_listFiles(0);
00871     return 0;
00872 }
00873
00874
00875
00876
00877
00878
00879
00880 }
```

3.23 file_system.h File Reference

Functions to read and write data from file system files.

```
#include <ti/drivers/net/wifi/simplelink.h>
#include "hal_LORA.h"
```

Macros

- #define CERTIFICATE "dummy-trusted-cert"
- #define FS_CHANGEWAKEUP "changewakeup"
- #define FS_DNCNTR "dncntr"
- #define FS_MODE "mode"
- #define FS_NBOOT "nboot"
- #define FS_NCYCLES "ncycles"

- #define FS_NFAILS "nfails"
- #define FS_NODEID "nodeid"
- #define FS_PAYLOAD "payload"
- #define FS_SSID "ssid"
- #define FS_UPCNTR "upcntr"
- #define FS_WAKEUP "wake_up_time"
- #define MAX_FILE_ENTRIES 4
- #define MAX_FILE_SIZE 256

Functions

- int **deleteFile** ()
File System write NFails Function.
- int **st_readFileChangeWakeUp** ()
File System read ChangeWakeUp Function.
- int **st_readFileDnCntr** ()
File System read DnCntr Function.
- int **st_read FileMode** ()
File System read Mode Function.
- int **st_readFileNBoot** ()
File System read NBoot Function.
- int **st_readFileNCycles** ()
File System read NCycles Function.
- int **st_readFileNFails** ()
File System read NFails Function.
- int **st_readFileNodeld** ()
File System read Nodeld Function.
- int **st_readFileSSID** (unsigned char *ssid)
File System read SSID Function.
- int **st_readFileUpCntr** ()
File System read UpCntr Function.
- int **st_readFileWakeUp** ()
File System read WakeUp Interval Function.
- int **writeChangeWakeUp** (uint8_t ChangeWakeUp)
File System write ChangeWakeUp Function.
- int **writeDnCntr** (uint32_t Dnctr)
File System write NFails Function.
- int **writeMode** (uint8_t Mode)
File System write Mode Function.
- int **writeNBoot** (uint8_t FirstBoot)
File System write NBoot Function.
- int **writeNCycles** (uint8_t NCycles)
File System write NCycles Function.
- int **writeNFails** (uint16_t NFails)
File System write NFails Function.
- **writeNodeld** (uint16_t Nodeld)
File System write Nodeld Function.
- int **writeSSID** (unsigned char *SSID)
File System write SSID Function.
- int **writeUpCntr** (uint32_t Upctr)
File System write Up Counter Function.
- int **writeWakeUp** (uint16_t WakeUpInterval)
File System write WakeUp Function.

3.23.1 Detailed Description

Functions to read and write data from file system files.

Version

1.0

Date

07/05/2019

Author

A.Pz @tittle File system functions

Definition in file [file_system.h](#).

3.23.2 Macro Definition Documentation

3.23.2.1 CERTIFICATE `#define CERTIFICATE "dummy-trusted-cert"`

Definition at line [34](#) of file [file_system.h](#).

3.23.2.2 FS_CHANGEWAKEUP `#define FS_CHANGEWAKEUP "changewakeup"`

Definition at line [32](#) of file [file_system.h](#).

3.23.2.3 FS_DNCNTR `#define FS_DNCNTR "dncntr"`

Definition at line [30](#) of file [file_system.h](#).

3.23.2.4 FS_MODE `#define FS_MODE "mode"`

Definition at line [24](#) of file [file_system.h](#).

3.23.2.5 FS_NBOOT #define FS_NBOOT "nboot"

Definition at line 27 of file [file_system.h](#).

3.23.2.6 FS_NCYCLES #define FS_NCYCLES "ncycles"

Definition at line 25 of file [file_system.h](#).

3.23.2.7 FS_NFAILS #define FS_NFAILS "nfails"

Definition at line 28 of file [file_system.h](#).

3.23.2.8 FS_NODEID #define FS_NODEID "nodeid"

Definition at line 31 of file [file_system.h](#).

3.23.2.9 FS_PAYLOAD #define FS_PAYLOAD "payload"

Definition at line 33 of file [file_system.h](#).

3.23.2.10 FS_SSID #define FS_SSID "ssid"

Definition at line 26 of file [file_system.h](#).

3.23.2.11 FS_UPCNTR #define FS_UPCNTR "upcntr"

Definition at line 29 of file [file_system.h](#).

3.23.2.12 FS_WAKEUP #define FS_WAKEUP "wake_up_time"

Definition at line 23 of file [file_system.h](#).

3.23.2.13 MAX_FILE_ENTRIES #define MAX_FILE_ENTRIES 4

Definition at line 35 of file [file_system.h](#).

3.23.2.14 MAX_FILE_SIZE #define MAX_FILE_SIZE 256

Definition at line 22 of file [file_system.h](#).

3.23.3 Function Documentation

3.23.3.1 deleteFile() int deleteFile ()

File System write NFails Function.

This function

1. Writes the number of LoRa OTAA connection fails on NFails file

Parameters

in	NFails	Number of LoRa OTAA connection fails
----	--------	--------------------------------------

Returns

_i32 offset

Definition at line 875 of file [file_system.c](#).

```
00875          {  
00876  
00877     sl_FsDel(FS_UPCNTR, 0);  
00878     st_listFiles(0);  
00879     return 0;  
00880 }
```

3.23.3.2 st_readFileChangeWakeUp() int st_readFileChangeWakeUp ()

File System read ChangeWakeUp Function.

This function

1. Reads the ChangeWakeUp value of the node from FS_CHANGEWAKEUP file

Returns

0

Definition at line 742 of file file_system.c.

```

00742     {
00743
00744     int offset = 0;
00745     int RetVal = 0;
00746     _i8 buffer[MAX_FILE_SIZE];
00747     int32_t fd;
00748
00749     fd = sl_FsOpen(FS_CHANGEWAKEUP, SL_FS_READ, 0);
00750     if (fd < 0) {
00751         UART_PRINT("\r\nError opening the file : %s\n\r", FS_CHANGEWAKEUP);
00752     }
00753
00754     while (RetVal=sl_FsRead(fd, offset, &buffer[0], MAX_FILE_SIZE)>1) {
00755         if(strlen(buffer)!=0){
00756             if(RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE){
00757                 break;
00758             }
00759             if(RetVal < 0){
00760                 UART_PRINT("sl_FsRead error: %d\n\r", RetVal);
00761                 return RetVal;
00762             }
00763             offset += strlen(buffer);
00764         }
00765     }
00766     UART_PRINT("READING CHANGE WAKE UP: %d\n\r", (atoi(&buffer)));
00767
00768     RetVal = sl_FsClose(fd, 0, 0, 0);
00769     if (RetVal < 0) {
00770         UART_PRINT("Error closing the file : %s\n\r", FS_CHANGEWAKEUP);
00771     }
00772
00773     return (atoi(&buffer));
00774 }
```

3.23.3.3 st_readFileDnCntr() int st_readFileDnCntr ()

File System read DnCntr Function.

This function

1. Reads the Down Counter parameter of Lora GW from FS_DNCNTR file

Returns

0

Definition at line 653 of file file_system.c.

```

00653     {
00654
00655     int offset = 0;
00656     int RetVal = 0;
00657     _i8 buffer[MAX_FILE_SIZE];
00658     int32_t fd;
00659
00660     fd = sl_FsOpen(FS_DNCNTR, SL_FS_READ, 0);
00661     if (fd < 0) {
00662         UART_PRINT("\r\nError opening the file : %s\n\r", FS_DNCNTR);
00663     }
00664
00665     while (RetVal=sl_FsRead(fd, offset, &buffer[0], MAX_FILE_SIZE)>1) {
00666         if(strlen(buffer)!=0){
00667             if(RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE){
00668                 break;
00669             }
00670             if(RetVal < 0){
00671                 UART_PRINT("sl_FsRead error: %d\n\r", RetVal);
00672             }
00673         }
00674     }
00675 }
```

```

00672         return RetVal;
00673     }
00674     offset += strlen(buffer);
00675   }
00676 }
00678 }
00679 RetVal = sl_FsClose(fd, 0, 0, 0);
00680 if (RetVal < 0){
00681   UART_PRINT("Error closing the file : %s\n\r", FS_DNCNTR);
00682 }
00683
00684 return (atoi(&buffer));
00685 }

```

3.23.3.4 st_read FileMode() int st_read FileMode ()

File System read Mode Function.

This function

1. Reads the Mode of the node from FS_MODE file

Returns

0

Definition at line 697 of file [file_system.c](#).

```

00697
00698
00699   int offset = 0;
00700   int RetVal = 0;
00701   _i8 buffer[MAX_FILE_SIZE];
00702   int32_t fd;
00703
00704   fd = sl_FsOpen(FS_MODE, SL_FS_READ, 0);
00705   if (fd < 0){
00706     UART_PRINT("\r\nError opening the file : %s\n\r", FS_MODE);
00707   }
00708
00709   while(RetVal=sl_FsRead(fd, offset, &buffer[0], MAX_FILE_SIZE)>1){
00710     if(strlen(buffer)!=0){
00711       if(RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE){
00712         break;
00713       }
00714       if(RetVal < 0){
00715         UART_PRINT("sl_FsRead error: %d\n\r", RetVal);
00716         return RetVal;
00717       }
00718       offset += strlen(buffer);
00719     }
00720   }
00721
00722   UART_PRINT("READING MODE: %d\n\r", (atoi(&buffer)));
00723
00724   RetVal = sl_FsClose(fd, 0, 0, 0);
00725   if (RetVal < 0){
00726     UART_PRINT("Error closing the file : %s\n\r", FS_MODE);
00727   }
00728
00729   return (atoi(&buffer));
00730 }

```

3.23.3.5 st_readFileNBoot() int st_readFileNBoot ()

File System read NBoot Function.

This function

1. Reads the NBoot value from FS_NBOOT file. Is used to change to alternate of sending the sensors data

Returns

0

Definition at line 522 of file [file_system.c](#).

```
00522     {
00523
00524     int offset = 0;
00525     int RetVal = 0;
00526     _i8 buffer[MAX_FILE_SIZE];
00527     int32_t fd;
00528
00529     fd = sl_FsOpen(FS_NBOOT, SL_FS_READ, 0);
00530     if (fd < 0) {
00531         UART_PRINT("\r\nError opening the file : %s\r\n", FS_NBOOT);
00532     }
00533
00534     while (RetVal=sl_FsRead(fd, offset, &buffer[0], MAX_FILE_SIZE)>1) {
00535         if(strlen(buffer)!=0) {
00536             if(RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE) {
00537                 break;
00538             }
00539             if(RetVal < 0) {
00540                 UART_PRINT("sl_FsRead error: %d\r\n", RetVal);
00541                 return RetVal;
00542             }
00543             offset += strlen(buffer);
00544         }
00545     }
00546
00547     RetVal = sl_FsClose(fd, 0, 0, 0);
00548     if (RetVal < 0) {
00549         UART_PRINT("Error closing the file : %s\r\n", FS_NBOOT);
00550     }
00551
00552     return (atoi(&buffer));
00553 }
```

3.23.3.6 st_readFileNCycles() int st_readFileNCycles ()

File System read NCycles Function.

This function

1. Reads the number of cycles of the node value from FS_NCYCLES file.

Returns

0

Definition at line 565 of file file_system.c.

```

00565     {
00566
00567     int offset = 0;
00568     int RetVal = 0;
00569     _i8 buffer[MAX_FILE_SIZE];
00570     int32_t fd;
00571
00572     fd = sl_FsOpen(FS_NCYCLES, SL_FS_READ, 0);
00573     if (fd < 0) {
00574         UART_PRINT("\r\nError opening the file : %s\n\r", FS_NCYCLES);
00575     }
00576
00577     while (RetVal=sl_FsRead(fd, offset, &buffer[0], MAX_FILE_SIZE)>1) {
00578         if(strlen(buffer)!=0) {
00579             if(RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE) {
00580                 break;
00581             }
00582             if(RetVal < 0) {
00583                 UART_PRINT("sl_FsRead error: %d\n\r", RetVal);
00584                 return RetVal;
00585             }
00586             offset += strlen(buffer);
00587         }
00588     }
00589     UART_PRINT("READING NCYCLES: %d \n\r", atoi(&buffer));
00590
00591     RetVal = sl_FsClose(fd, 0, 0, 0);
00592     if (RetVal < 0) {
00593         UART_PRINT("Error closing the file : %s\n\r", FS_NCYCLES);
00594     }
00595
00596     return (atoi(&buffer));
00597 }
```

3.23.3.7 st_readFileNFails() int st_readFileNFails ()

File System read NFails Function.

This function

1. Reads the number of Lora OTAA connection fails from FS_NFAILS file. Is used to change to wifi if Lora OTAA fails n times.

Returns

0

Definition at line 477 of file file_system.c.

```

00477     {
00478
00479     int offset = 0;
00480     int RetVal = 0;
00481     _i8 buffer[MAX_FILE_SIZE];
00482     int32_t fd;
00483
00484     fd = sl_FsOpen(FS_NFAILS, SL_FS_READ, 0);
00485     if (fd < 0) {
00486         UART_PRINT("\r\nError opening the file : %s\n\r", FS_NFAILS);
00487     }
00488
00489     while (RetVal=sl_FsRead(fd, offset, &buffer[0], MAX_FILE_SIZE)>1)
00490     {
00491         if(strlen(buffer)!=0) {
00492             if(RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE) {
00493                 break;
00494             }
00495         }
00496     }
00497 }
```

```

00495     if(RetVal < 0){
00496         UART_PRINT("sl_FsRead error: %d\n\r", RetVal);
00497         return RetVal;
00498     }
00499     offset += strlen(buffer);
00500     }
00501 }
00502 UART_PRINT("READING NFAILS: %d \n\r", atoi(&buffer));
00503
00504 RetVal = sl_FsClose(fd, 0, 0, 0);
00505 if (RetVal < 0){
00506     UART_PRINT("Error closing the file : %s\n\r", FS_NFAILS);
00507 }
00508
00509 return (atoi(&buffer));
00510 }

```

3.23.3.8 st_readFileNodeId() int st_readFileNodeId ()

File System read Nodeld Function.

This function

1. Reads the Nodeld from FS_NODEID file

Returns

0

Definition at line 829 of file [file_system.c](#).

```

00829
00830
00831     int offset = 0;
00832     int RetVal = 0;
00833     _i8 buffer[MAX_FILE_SIZE];
00834     int32_t fd;
00835
00836     fd = sl_FsOpen(FS_NODEID, SL_FS_READ, 0);
00837     if (fd < 0){
00838         UART_PRINT("\r\nError opening the file : %s\n\r", FS_NODEID);
00839     }
00840
00841     while(RetVal=sl_FsRead(fd, offset, &buffer[0], MAX_FILE_SIZE)>1){
00842         if(strlen(buffer)!=0){
00843             if(RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE){
00844                 break;
00845             }
00846             if(RetVal < 0){
00847                 UART_PRINT("sl_FsRead error: %d\n\r", RetVal);
00848                 return RetVal;
00849             }
00850             offset += strlen(buffer);
00851         }
00852     }
00853
00854     UART_PRINT("READING NODEID: %d \n\r", atoi(&buffer));
00855
00856     RetVal = sl_FsClose(fd, 0, 0, 0);
00857     if (RetVal < 0){
00858         UART_PRINT("Error closing the file : %s\n\r", FS_NODEID);
00859     }
00860     return (atoi(&buffer));
00861 }

```

3.23.3.9 st_readFileSSID() int st_readFileSSID (unsigned char * ssid)

File System read SSID Function.

This function

1. Reads the wifi SSID from FS_SSID file

Parameters

<i>unsigned</i>	char pointer to ssid
-----------------	----------------------

Returns

0

Definition at line 441 of file [file_system.c](#).

```

00441     int RetVal = 0;
00442     int32_t fd;
00443
00444     fd = sl_FsOpen(FS_SSID, SL_FS_READ, 0);
00445     if (fd < 0) {
00446         UART_PRINT("\r\nError opening the file : %s\n\r", FS_SSID);
00447     }
00448
00449     RetVal = sl_FsRead(fd, 0, ssid, strlen(ssid));
00450
00451     if(RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE) {
00452         return RetVal;
00453     }else if (RetVal < 0) {
00454         UART_PRINT("sl_FsRead error: %d\n\r", RetVal);
00455         return RetVal;
00456     }
00457
00458     RetVal = sl_FsClose(fd, 0, 0, 0);
00459     if (RetVal < 0) {
00460         UART_PRINT("Error closing the file : %s\n\r", FS_SSID);
00461     }
00462
00463     return 0;
00464 }
```

3.23.3.10 st_readFileUpCntr() int st_readFileUpCntr ()

File System read UpCntr Function.

This function

1. Reads the Up Counter parameter of Lora GW from FS_UPCNTR file

Returns

0

Definition at line 609 of file [file_system.c](#).

```

00609     {
00610
00611     int offset = 0;
00612     int RetVal = 0;
00613     _i8 buffer[MAX_FILE_SIZE];
00614     int32_t fd;
00615
00616     fd = sl_FsOpen(FS_UPCNTR, SL_FS_READ, 0);
00617     if (fd < 0) {
00618         UART_PRINT("\r\nError opening the file : %s\n\r", FS_UPCNTR);
00619     }
00620
00621     while(RetVal=sl_FsRead(fd, offset, &buffer[0], MAX_FILE_SIZE)>1){
00622         if(strlen(buffer)!=0){
00623             if(RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE){
00624                 break;
00625             }
00626             if(RetVal < 0) {
```

```

00627         UART_PRINT("sl_FsRead error: %d\n\r", RetVal);
00628         return RetVal;
00629     }
00630     offset += strlen(buffer);
00631 }
00632 }
00633 }
00634
00635 RetVal = sl_FsClose(fd, 0, 0, 0);
00636 if (RetVal < 0){
00637     UART_PRINT("Error closing the file : %s\n\r", FS_UPCNTR);
00638 }
00639
00640 return (atoi(&buffer));
00641 }
```

3.23.3.11 st_readFileWakeUp() int st_readFileWakeUp ()

File System read WakeUp Interval Function.

This function

1. Reads the wake up interval value of the node from FS_WAKEUP file

Returns

0

Definition at line 786 of file [file_system.c](#).

```

00786     {
00787
00788     int offset = 0;
00789     int RetVal = 0;
00790     _i8 buffer[MAX_FILE_SIZE];
00791     int32_t fd;
00792
00793     fd = sl_FsOpen(FS_WAKEUP, SL_FS_READ, 0);
00794     if (fd < 0){
00795         UART_PRINT("\r\nError opening the file : %s\n\r", FS_WAKEUP);
00796     }
00797
00798     while(RetVal=sl_FsRead(fd, offset, &buffer[0], MAX_FILE_SIZE)>1){
00799         if(strlen(buffer) !=0){
00800             if(RetVal == SL_ERROR_FS_OFFSET_OUT_OF_RANGE){
00801                 break;
00802             }
00803             if(RetVal < 0){
00804                 UART_PRINT("sl_FsRead error: %d\n\r", RetVal);
00805                 return RetVal;
00806             }
00807             offset += strlen(buffer);
00808         }
00809     }
00810     UART_PRINT("\r\nREADING WAKEUPINTERVAL: %d\n\r", atoi(&buffer));
00811
00812     RetVal = sl_FsClose(fd, 0, 0, 0);
00813     if (RetVal < 0){
00814         UART_PRINT("Error closing the file : %s\n\r", FS_WAKEUP);
00815     }
00816     return (atoi(&buffer));
00817 }
```

3.23.3.12 writeChangeWakeUp() int writeChangeWakeUp (uint8_t ChangeWakeUp)

File System write ChangeWakeUp Function.

This function

1. Writes the ChangeWakeUp value on FS_CHANGEWAKEUP file. Is used to determine if is necessary to change the node wake up time.

Parameters

in	<i>uint8_t</i>	ChangeWakeUp
	<i>_t</i>	

Returns*_i32 offset***Definition at line 322 of file [file_system.c](#).**

```

00322
00323
00324     int RetVal = 0;
00325     _i32 offset = 0;
00326     _i32 fd;
00327     unsigned char changewakeup[32];
00328
00329     sprintf(&changewakeup,"%d",ChangeWakeUp );
00330
00331     fd = sl_FsOpen(FS_CHANGEWAKEUP, SL_FS_OVERWRITE, 0);
00332     if (fd < 0){
00333         UART_PRINT("Error opening the file : %s\n\r", FS_CHANGEWAKEUP);
00334     }
00335     else{
00336         RetVal = sl_FsWrite(fd, 0, changewakeup, strlen(changewakeup));
00337         if (RetVal <= 0){
00338             UART_PRINT("Writing error: %d\n\r" ,RetVal);
00339             return RetVal;
00340         }
00341
00342         RetVal = sl_FsClose(fd, 0, 0, 0);
00343         if (RetVal < 0){
00344             UART_PRINT("Error closing the file : %s\n\r", FS_CHANGEWAKEUP);
00345         }
00346     }
00347     return offset;
00348 }
```

3.23.3.13 writeDnCntr() *int writeDnCntr (**uint32_t Dnctr)*

File System write NFails Function.

This function

1. Writes the number of LoRa OTAA connection fails on NFails file

Parameters

in	<i>NFails</i>	Number of LoRa OTAA connection fails
----	---------------	--------------------------------------

Returns*_i32 offset***Definition at line 242 of file [file_system.c](#).**

```

00242
00243
00244     int RetVal = 0;
00245     _i32 offset = 0;
```

```

00246     _i32 fd;
00247     unsigned char dnctr[32];
00248
00249     sprintf(&dnctr, "%d", Dnctr);
00250
00251     fd = sl_FsOpen(FS_DNCNTR, SL_FS_OVERWRITE, 0);
00252     if (fd < 0) {
00253         UART_PRINT("Error opening the file : %s\n\r", FS_DNCNTR);
00254     }
00255     else{
00256         RetVal = sl_FsWrite(fd, 0, dnctr, strlen(dnctr));
00257         if (RetVal <= 0){
00258             UART_PRINT("Writing error: %d\n\r", RetVal);
00259             return RetVal;
00260         }
00261
00262         RetVal = sl_FsClose(fd, 0, 0, 0);
00263         if (RetVal < 0){
00264             UART_PRINT("Error closing the file : %s\n\r", FS_DNCNTR);
00265         }
00266     }
00267     return offset;
00268 }
```

3.23.3.14 writeMode() int writeMode (uint8_t Mode)

File System write Mode Function.

This function

1. Writes the working mode of the node (Lora or Wifi) on FS_MODE file

Parameters

in	Mode	working mode of the node (0 for LoRa, 2 for Wifi)
----	------	---

Returns

_i32 offset

Definition at line 282 of file [file_system.c](#).

```

00282
00283     {
00284     int RetVal = 0;
00285     _i32 offset = 0;
00286     _i32 fd;
00287     unsigned char mode[32];
00288
00289     sprintf(&mode, "%d", Mode );
00290
00291     fd = sl_FsOpen(FS_MODE, SL_FS_OVERWRITE, 0);
00292     if (fd < 0){
00293         UART_PRINT("Error opening the file : %s\n\r", FS_MODE);
00294     }
00295     else{
00296         RetVal = sl_FsWrite(fd, 0, mode, strlen(mode));
00297         if (RetVal <= 0){
00298             UART_PRINT("Writing error: %d\n\r", RetVal);
00299             return RetVal;
00300         }
00301
00302         RetVal = sl_FsClose(fd, 0, 0, 0);
00303         if (RetVal < 0){
00304             UART_PRINT("Error closing the file : %s\n\r", FS_MODE);
00305         }
00306     }
```

```
00307     return offset;
00308 }
```

3.23.3.15 writeNBoot() int writeNBoot (

uint8_t FirstBoot)

File System write NBoot Function.

This function

1. Writes the number of NBoot on FS_NBOOT file. This value is used to alternate the data sent between sensors.

Parameters

in	<i>uint8_t</i>	FirstBoot yes or no
----	----------------	---------------------

Returns

_i32 offset

Definition at line 118 of file [file_system.c](#).

```
00118 {
00119
00120     int RetVal = 0;
00121     _i32 offset = 0;
00122     _i32 fd;
00123     unsigned char fboot[32];
00124
00125     sprintf(&fboot,"%d",FirstBoot );
00126
00127     fd = sl_FsOpen(FS_NBOOT, SL_FS_OVERWRITE, 0);
00128     if (fd < 0){
00129         UART_PRINT("Error opening the file : %s\n\r", FS_NBOOT);
00130     }
00131     else{
00132         RetVal = sl_FsWrite(fd, 0, fboot, strlen(fboot));
00133         if (RetVal <= 0){
00134             UART_PRINT("Writing error: %d\n\r" ,RetVal);
00135             return RetVal;
00136         }
00137
00138         RetVal = sl_FsClose(fd, 0, 0, 0);
00139         if (RetVal < 0){
00140             UART_PRINT("Error closing the file : %s\n\r", FS_NBOOT);
00141         }
00142     }
00143     return offset;
00144 }
```

3.23.3.16 writeNCycles() int writeNCycles (

uint8_t NCycles)

File System write NCycles Function.

This function

1. Writes the number of cycles of the node on FS_NCYCLES file

Parameters

in	<i>uint8_t</i>	NCycles Number of cycles
	<i>_t</i>	

Returns*_i32 offset***Definition at line 158 of file file_system.c.**

```

00158
00159
00160     int RetVal = 0;
00161     _i32 offset = 0;
00162     _i32 fd;
00163     unsigned char ncycles[32];
00164
00165     sprintf(&ncycles, "%d", NCycles );
00166
00167     fd = sl_FsOpen(FS_NCYCLES, SL_FS_OVERWRITE, 0);
00168     if (fd < 0){
00169         UART_PRINT("Error opening the file : %s\n\r", FS_NCYCLES);
00170     }
00171     else{
00172         RetVal = sl_FsWrite(fd, 0, ncycles, strlen(ncycles));
00173         if (RetVal <= 0){
00174             UART_PRINT("Writing error: %d\n\r", RetVal);
00175             return RetVal;
00176         }
00177
00178         RetVal = sl_FsClose(fd, 0, 0, 0);
00179         if (RetVal < 0){
00180             UART_PRINT("Error closing the file : %s\n\r", FS_NCYCLES);
00181         }
00182     }
00183     return offset;
00184 }
```

3.23.3.17 writeNFails() *int writeNFails (**uint16_t NFails)*

File System write NFails Function.

This function

1. Writes the number of LoRa OTAA connection fails on NFails file

Parameters

in	<i>uint16_t</i>	FS_NFAILS Number of LoRa OTAA connection fails
	<i>_t</i>	

Returns*_i32 offset***Definition at line 79 of file file_system.c.**

```

00079
00080 {
```

```

00081     int RetVal = 0;
00082     _i32 offset = 0;
00083     _i32 fd;
00084     unsigned char fails[32];
00085
00086     sprintf(&fails,"%d",NFails );
00087
00088     fd = sl_FsOpen(FS_NFAILS, SL_FS_OVERWRITE, 0);
00089     if (fd < 0){
00090         UART_PRINT("Error opening the file : %s\n\r", FS_NFAILS);
00091     }
00092     else{
00093         RetVal = sl_FsWrite(fd, 0, fails, strlen(fails));
00094         if (RetVal <= 0){
00095             UART_PRINT("Writing error: %d\n\r", RetVal);
00096             return RetVal;
00097         }
00098         RetVal = sl_FsClose(fd, 0, 0, 0);
00099         if (RetVal < 0){
00100             UART_PRINT("Error closing the file : %s\n\r", FS_NFAILS);
00101         }
00102     }
00103     return offset;
00104 }
```

3.23.3.18 writeNodeld() writeNodeId(uint16_t Nodeld)

File System write Nodeld Function.

This function

1. Writes the Nodeld on Nodeld file

Parameters

in	<i>uint16_t</i>	Node Id
----	-----------------	---------

Returns

_i32 offset

Definition at line 402 of file [file_system.c](#).

```

00402
00403
00404     int RetVal = 0;
00405     _i32 offset = 0;
00406     _i32 fd;
00407     unsigned char nodeid[32];
00408
00409     sprintf(&nodeid,"%d",NodeId );
00410
00411     fd = sl_FsOpen(FS_NODEID, SL_FS_OVERWRITE, 0);
00412     if (fd < 0){
00413         UART_PRINT("Error opening the file : %s\n\r", FS_NODEID);
00414     }
00415     else{
00416         RetVal = sl_FsWrite(fd, 0, nodeid, strlen(nodeid));
00417         if (RetVal <= 0){
00418             UART_PRINT("Writing error: %d\n\r", RetVal);
00419             return RetVal;
00420         }
00421         RetVal = sl_FsClose(fd, 0, 0, 0);
00422         if (RetVal < 0){
00423             UART_PRINT("Error closing the file : %s\n\r", FS_NODEID);
00424         }
00425 }
```

```

00425     }
00426     return offset;
00427 }
```

3.23.3.19 writeSSID() int writeSSID (unsigned char * SSID)

File System write SSID Function.

This function

1. Writes the SSID of Wifi network on FS_SSID file

Parameters

in	<i>unsigned</i>	char SSID Pointer to SSID
----	-----------------	---------------------------

Returns

_i32 offset

Definition at line 39 of file [file_system.c](#).

```

00039
00040
00041     int RetVal = 0;
00042     _i32 offset = 0;
00043     _i32 fd;
00044     unsigned char ssid[13];
00045
00046     sprintf(&ssid,"%s",SSID );
00047
00048     fd = sl_FsOpen(FS_SSID, SL_FS_OVERWRITE, 0);
00049     if (fd < 0){
00050         UART_PRINT("Error opening the file : %s\n\r", FS_SSID);
00051     }
00052     else{
00053         RetVal = sl_FsWrite(fd, 0, ssid, strlen(ssid));
00054         if (RetVal <= 0){
00055             UART_PRINT("Writing error: %d\n\r" ,RetVal);
00056             return RetVal;
00057         }
00058
00059         RetVal = sl_FsClose(fd, 0, 0);
00060         if (RetVal < 0){
00061             UART_PRINT("Error closing the file : %s\n\r", FS_SSID);
00062         }
00063     }
00064     return offset;
00065 }
```

3.23.3.20 writeUpCntr() int writeUpCntr (uint32_t Upctr)

File System write Up Counter Function.

This function

1. Writes the value of UpCntr LoRa parameter on FS_UPCNTR file

Parameters

in	<i>uint32_t</i>	NFails UpCntr LoRa parameter value
----	-----------------	------------------------------------

Returns*_i32 offset*Definition at line 198 of file [file_system.c](#).

```

00198
00199
00200     int RetVal = 0;
00201     _i32 offset = 0;
00202     _i32 fd;
00203     unsigned char upctr[32];
00204
00205     sprintf(&upctr, "%d", Upctr );
00206
00207     fd = sl_FsOpen(FS_UPCNTR, SL_FS_OVERWRITE, 0);
00208     if (fd < 0)
00209     {
00210         UART_PRINT("Error opening the file : %s\n\r", FS_UPCNTR);
00211     }
00212     else
00213     {
00214         RetVal = sl_FsWrite(fd, 0, upctr, strlen(upctr));
00215         if (RetVal <= 0)
00216         {
00217             UART_PRINT("Writing error: %d\n\r", RetVal);
00218             return RetVal;
00219         }
00220
00221         RetVal = sl_FsClose(fd, 0, 0, 0);
00222         if (RetVal < 0)
00223         {
00224             UART_PRINT("Error closing the file : %s\n\r", FS_UPCNTR);
00225         }
00226     }
00227     return offset;
00228 }
```

3.23.3.21 writeWakeUp() `int writeWakeUp (`
 `uint16_t WakeUpInterval)`

File System write WakeUp Function.

This function

1. Writes the wake up interval time of the node on FS_WAKEUP file

Parameters

in	<i>uint16_t</i>	WakeUpInterval Time(s) of the wake up interval
----	-----------------	--

Returns*_i32 offset*Definition at line 362 of file [file_system.c](#).

```

00362
00363
00364     int RetVal = 0;
00365     _i32 offset = 0;
00366     _i32 fd;
00367     unsigned char wakeup[32];
00368
00369     sprintf(&wakeup, "%d", WakeUpInterval );
00370
00371     fd = sl_FsOpen(FS_WAKEUP, SL_FS_OVERWRITE, 0);
00372     if (fd < 0){
00373         UART_PRINT("Error opening the file : %s\n\r", FS_WAKEUP);
00374     }
00375     else{
00376         RetVal = sl_FsWrite(fd, 0, wakeup, strlen(wakeup));
00377         if (RetVal <= 0){
00378             UART_PRINT("Writing error: %d\n\r", RetVal);
00379             return RetVal;
00380         }
00381
00382         RetVal = sl_FsClose(fd, 0, 0, 0);
00383         if (RetVal < 0){
00384             UART_PRINT("Error closing the file : %s\n\r", FS_WAKEUP);
00385         }
00386     }
00387     return offset;
00388 }
```

3.24 file_system.h

```

00001
00010 #ifndef FILE_SYSTEM_H_
00011 #define FILE_SYSTEM_H_
00012
00013 //*****
00014 //           INCLUDES
00015 //*****
00016 #include <ti/drivers/net/wifi/simplelink.h>
00017 #include "hal_LORA.h"
00018
00019 //*****
00020 //           DEFINES
00021 //*****
00022 #define MAX_FILE_SIZE 256
00023 #define FS_WAKEUP          "wake_up_time"
00024 #define FS_MODE            "mode"
00025 #define FS_NCYCLES        "ncycles"
00026 #define FS_SSID            "ssid"
00027 #define FS_NBOOT           "nboot"
00028 #define FS_NFAILS          "nfails"
00029 #define FS_UPCNTR          "upcntr"
00030 #define FS_DNCNTR          "dhcntr"
00031 #define FS_NODEID          "nodeid"
00032 #define FS_CHANGEWAKEUP    "changewakeup"
00033 #define FS_PAYLOAD          "payload"
00034 #define CERTIFICATE        "dummy-trusted-cert"
00035 #define MAX_FILE_ENTRIES   4
00036
00037 //*****
00038 //           FUNCTION PROTOTYPES
00039 //*****
00040 int writeNFails(uint16_t NFails);
00041 int writeNBoot(uint8_t FirstBoot);
00042 int writeNCycles(uint8_t NCycles);
00043 int writeUpCntr(uint32_t Upctr);
00044 int writeDnCntr(uint32_t Dnctr);
00045 int writeChangeWakeUp(uint8_t ChangeWakeUp);
00046 int writeWakeUp(uint16_t WakeUpInterval);
00047 int writeMode(uint8_t Mode);
00048 int writeWakeUp(uint16_t WakeUpInterval);
00049 int writeNodeId(uint16_t NodeId);
00050 int writeSSID(unsigned char *SSID);
00051 int st_readFileNFails();
00052 int st_readFileNBoot();
00053 int st_readFileNCycles();
00054 int st_readFileUpCntr();
00055 int st_readFileDnCntr();
00056 int st_read FileMode();
00057 int st_readFileChangeWakeUp();
00058 int st_readFileWakeUp();
00059 int st_readFileNodeId();
00060 int st_readFileSSID(unsigned char *ssid);
00061 int deleteFile();
00062
00063 #endif /* FILE_SYSTEM_H_ */
```

3.25 hal_ADC.c File Reference

tr Function.50cJ0 g 0 G3.20 g 3.20 GJ/579 8.9663 Tf 0 -03.594 Td [(3.2.15)-1000DataFildc

Ptfunc579FunctionsADC50cnctions

Functions for ADC.50cJ0 g 0 G3.20 g 3.20 GJ/579 8.9663 Tf 0 -03.594 Td [Vestioe

Parameters

in	<i>uint8_t</i>	index
	<i>_t</i>	

Returns

ADC_Handle adc

Definition at line 34 of file [hal_ADC.c](#).

```

00034
00035
00036     ADC_Handle adc;
00037     ADC_Parms params;
00038
00039     ADC_Parms_init(&params);
00040     ADC_init();
00041     adc = ADC_open(index, &params);
00042
00043     return adc;
00044 }
```

3.26 hal_ADC.c

```

00001
00010 //*****
00011 //      INCLUDES
00012 //*****
00013 #include <ti/drivers/ADC.h>
00014 #include <stdint.h>
00015 #include <stddef.h>
00016 #include <stdbool.h>
00017
00018 //*****
00019 //      FUNCTIONS
00020 //*****
00021
00022 //*****
00023 //
00032 //
00033 //*****
00034 ADC_Handle Startup_ADC(uint8_t index) {
00035
00036     ADC_Handle adc;
00037     ADC_Parms params;
00038
00039     ADC_Parms_init(&params);
00040     ADC_init();
00041     adc = ADC_open(index, &params);
00042
00043     return adc;
00044 }
```

3.27 hal_ADC.h File Reference

Functions for ADC.

```
#include <stdint.h>
#include <ti/drivers/ADC.h>
```

Functions

- ADC_Handle [Startup_ADC](#) (uint8_t index)
ADC Start up Function.

3.27.1 Detailed Description

Functions for ADC.

Version

1.0

Date

07/05/2019

Author

A.Irizar @title ADC functions

Definition in file [hal_ADC.h](#).

3.27.2 Function Documentation

3.27.2.1 Startup_ADC()

```
ADC_HandleTypeDef Startup_ADC (
    uint8_t index )
```

ADC Start up Function.

This function

1. Handle for ADC

Parameters

in	<i>uint8_t</i>	index
	<i>t</i>	

Returns

ADC_HandleTypeDef adc

Definition at line 34 of file [hal_ADC.c](#).

```
00034
00035
00036     ADC_HandleTypeDef adc;
00037     ADC_Parmas params;
00038
00039     ADC_Parmas_init(&params);
00040     ADC_init();
00041     adc = ADC_open(index, &params);
00042
00043     return adc;
00044 }
```

3.28 hal_ADC.h

```

00001
00010 #ifndef HAL_ADC_H_
00011 #define HAL_ADC_H_
00012
00013 //*****
00014 //          INCLUDES
00015 //*****
00016 #include <stdint.h>
00017 #include <ti/drivers/ADC.h>
00018
00019 //*****
00020 //          FUNCTION PROTOTYPES
00021 //*****
00022 ADC_Handle Startup_ADC(uint8_t index);
00023
00024 #endif /* HAL_ADC_H_ */

```

3.29 hal_GPIO.c File Reference

Functions for GPIO utilities.

```
#include <unistd.h>
#include <ti/drivers/GPIO.h>
#include "Board.h"
```

Functions

- void [ADXL355_Disable](#) (void)
GPIO ADXL355 Disable Function.
- void [ADXL355_Enable](#) (void)
GPIO ADXL355 enable Function.
- void [ADXL355_SPI_Disable](#) (void)
GPIO ADXL355 SPI Disable Function.
- void [ADXL355_SPI_Enable](#) (void)
GPIO ADXL355 SPI enable Function.
- void [BME280_Disable](#) (void)
GPIO BME280 Disable Function.
- void [BME280_Enable](#) (void)
GPIO BME280 enable Function.
- void [BME280_SPI_Disable](#) (void)
GPIO BME280 SPI Disable Function.
- void [BME280_SPI_Enable](#) (void)
GPIO BME280 SPI enable Function.
- uint8_t [GPIO_Config](#) (void)
GPIO Configuration Function.
- uint8_t [I2C_As_GPIO_Low](#) (void)
GPIO I2C Low Function.
- void [LDC1000_Disable](#) (void)
GPIO LDC1000 Disable Function.
- void [LDC1000_Enable](#) (void)
GPIO LDC1000 enable Function.
- void [LDC1000_SPI_Disable](#) (void)
GPIO LDC1000 SPI Disable Function.
- void [LDC1000_SPI_Enable](#) (void)

- void [Node_Disable](#) (void)
GPIO Node Disable Function.
- void [Node_Enable](#) (void)
GPIO Node enable Function.
- void [RN2483_Clear](#) (void)
GPIO RN2483 Clear Function.
- void [RN2483_Set](#) (void)
GPIO RN2483 Set Function.
- uint8_t [SPI_As_GPIO_Low](#) (void)
GPIO SPI Low Function.
- void [SPI_CS_Disable](#) (void)
GPIO CS SPI Disable Function.

3.29.1 Detailed Description

Functions for GPIO utilities.

Version

1.0

Date

07/05/2019

Author

A.Irizar @tittle GPIO functions

Definition in file [hal_GPIO.c](#).

3.29.2 Function Documentation

3.29.2.1 ADXL355_Disable()

```
void ADXL355_Disable (
    void )
```

GPIO ADXL355 Disable Function.

This function

1. Configures ADXL355 GPIO pin as Low (Disable)

Parameters

in		
----	--	--

Definition at line 152 of file [hal_GPIO.c](#).

```
00152          {  
00153  
00154     GPIO_write(Board_EN_ADXL355,0);  
00155 }
```

3.29.2.2 ADXL355_Enable() void ADXL355_Enable (void)

GPIO ADXL355 enable Function.

This function

1. Configures ADXL355 GPIO pin as High (Enable)

Parameters

in		
----	--	--

Definition at line 135 of file [hal_GPIO.c](#).

```
00135          {  
00136  
00137     GPIO_write(Board_EN_ADXL355,1);  
00138 }
```

3.29.2.3 ADXL355_SPI_Disable() void ADXL355_SPI_Disable (void)

GPIO ADXL355 SPI Disable Function.

This function

1. Configures ADXL355 SPI GPIO pin as High (Disable)

Parameters

in		
----	--	--

Definition at line 288 of file [hal_GPIO.c](#).

```
00288          {  
00289  
00290     GPIO_write(Board_ADXL355_CS,1);  
00291 }
```

```
3.29.2.4 ADXL355_SPI_Enable() void ADXL355_SPI_Enable (
    void )
```

GPIO ADXL355 SPI enable Function.

This function

1. Configures ADXL355 SPI GPIO pin as Low (Enable)

Parameters

in		
----	--	--

Definition at line 271 of file [hal_GPIO.c](#).

```
00271 {
00272
00273     GPIO_write(Board_ADXL355_CS, 0);
00274 }
```

```
3.29.2.5 BME280_Disable() void BME280_Disable (
    void )
```

GPIO BME280 Disable Function.

This function

1. Configures BME280 GPIO pin as Low (Disable)

Parameters

in		
----	--	--

Definition at line 186 of file [hal_GPIO.c](#).

```
00186 {
00187
00188     GPIO_write(Board_EN_BME280, 0);
00189 }
```

```
3.29.2.6 BME280_Enable() void BME280_Enable (
    void )
```

GPIO BME280 enable Function.

This function

1. Configures BME280 GPIO pin as High (Enable)

Parameters

in		
----	--	--

Definition at line 169 of file [hal_GPIO.c](#).

```
00169      {
00170
00171     GPIO_write(Board_EN_BME280,1);
00172 }
```

3.29.2.7 BME280_SPI_Disable() void BME280_SPI_Disable (void)

GPIO BME280 SPI Disable Function.

This function

1. Configures BME280 SPI GPIO pin as High (Disable)

Parameters

in		
----	--	--

Definition at line 322 of file [hal_GPIO.c](#).

```
00322      {
00323
00324     GPIO_write(Board_BME280_CS,1);
00325 }
```

3.29.2.8 BME280_SPI_Enable() void BME280_SPI_Enable (void)

GPIO BME280 SPI enable Function.

This function

1. Configures BME280 SPI GPIO pin as Low (Enable)

Parameters

in		
----	--	--

Definition at line 305 of file [hal_GPIO.c](#).

```
00305      {
00306
00307     GPIO_write(Board_BME280_CS,0);
00308 }
```

3.29.2.9 GPIO_Config() uint8_t GPIO_Config (void)

GPIO Configuration Function.

This function

1. Configures CC3220SF GPIO pins

Parameters

in		
----	--	--

Definition at line 33 of file hal_GPIO.c.

```
00033           {
00034
00035     GPIO_setConfig(Board_RN2483_MCLR,      GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);          //MCLR Pin
00036     GPIO_setConfig(Board_ADXL355_CS,      GPIO_CFG_OUT_STD | GPIO_CFG_OUT_HIGH);           //Selects the
00037     SPI_device_to_interface (Active Low)
00038     GPIO_setConfig(Board_BME280_CS,        GPIO_CFG_OUT_STD | GPIO_CFG_OUT_HIGH);
00039     GPIO_setConfig(Board_LDC1000_CS,       GPIO_CFG_OUT_STD | GPIO_CFG_OUT_HIGH);
00040     GPIO_setConfig(Board_EN_ADXL355,       GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);            //Enable the
00041     individual LDO for sensor (active high)
00042     GPIO_setConfig(Board_EN_BME280,         GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
00043     GPIO_setConfig(Board_EN_LDC1000,        GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
00044     GPIO_setConfig(Board_DS1374_INTB,       GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);           //This
00045   connection no longer exists, so put it low
00046
00047   return 0;
00048 }
```

3.29.2.10 I2C_As_GPIO_Low() uint8_t I2C_As_GPIO_Low (void)

GPIO I2C Low Function.

This function

1. Configures CC3220SF I2C GPIO pin as Low

Parameters

in		
----	--	--

Definition at line 59 of file hal_GPIO.c.

```
00059           {
00060
00061     GPIO_setConfig(Board_SCL_ASGPIO,      GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);          //Important to
00062     usleep(2000);
00063     avoid an Start condition on RTCC (see datasheet DS1374)
00064     GPIO_setConfig(Board_SDA_ASGPIO,      GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
00065
00066 }
```

3.29.2.11 LDC1000_Disable() void LDC1000_Disable (void)

GPIO LDC1000 Disable Function.

This function

1. Configures LDC1000 GPIO pin as Low (Disable)

Parameters

in		
----	--	--

Definition at line 220 of file [hal_GPIO.c](#).

```
00220 {  
00221  
00222     GPIO_write(Board_EN_LDC1000,0);  
00223 }
```

3.29.2.12 LDC1000_Enable() void LDC1000_Enable (void)

GPIO LDC1000 enable Function.

This function

1. Configures LDC1000 GPIO pin as High (Enable)

Parameters

in		
----	--	--

Definition at line 203 of file [hal_GPIO.c](#).

```
00203 {  
00204  
00205     GPIO_write(Board_EN_LDC1000,1);  
00206 }
```

3.29.2.13 LDC1000_SPI_Disable() void LDC1000_SPI_Disable (void)

GPIO LDC1000 SPI Disable Function.

This function

1. Configures LDC1000 SPI GPIO pin as High (Disable)

Parameters

in		
----	--	--

Definition at line 356 of file [hal_GPIO.c](#).

```
00356 {  
00357  
00358     GPIO_write(Board_LDC1000_CS,1);  
00359 }
```

3.29.2.14 LDC1000_SPI_Enable() void LDC1000_SPI_Enable (
void)

GPIO LDC1000 SPI enable Function.

This function

1. Configures LDC1000 SPI GPIO pin as Low (Enable)

Parameters

in		
----	--	--

Definition at line 339 of file [hal_GPIO.c](#).

```
00339 {  
00340  
00341     GPIO_write(Board_LDC1000_CS,0);  
00342 }
```

3.29.2.15 Node_Disable() void Node_Disable (
void)

GPIO Node Disable Function.

This function

1. Configures Node GPIO pin as Low (Disable)

Parameters

in		
----	--	--

Definition at line 254 of file [hal_GPIO.c](#).

```
00254 {  
00255  
00256     GPIO_write(Board_EN_NODE,0);  
00257 }
```

```
3.29.2.16 Node_Enable() void Node_Enable (
    void )
```

GPIO [Node](#) enable Function.

This function

1. Configures [Node](#) GPIO pin as High (Enable)

Parameters

in		
----	--	--

Definition at line [237](#) of file [hal_GPIO.c](#).

```
00237     {
00238
00239     GPIO_write(Board_EN_NODE, 1);
00240 }
```

```
3.29.2.17 RN2483_Clear() void RN2483_Clear (
    void )
```

GPIO RN2483 Clear Function.

This function

1. Configures RN2483 GPIO pin as Low (clear)

Parameters

in		
----	--	--

Definition at line [101](#) of file [hal_GPIO.c](#).

```
00101     {
00102
00103     GPIO_write(Board_RN2483_MCLR, 0);
00104 }
```

```
3.29.2.18 RN2483_Set() void RN2483_Set (
    void )
```

GPIO RN2483 Set Function.

This function

1. Configures RN2483 GPIO pin as High (set)

Parameters

in		
----	--	--

Definition at line 118 of file hal_GPIO.c.

```
00118     {
00119
00120     GPIO_write(Board_RN2483_MCLR, 1);
00121 }
```

3.29.2.19 SPI_As_GPIO_Low() `uint8_t SPI_As_GPIO_Low (`
`void)`

GPIO SPI Low Function.

This function

1. Configures CC3220SF SPI GPIO pin as Low

Parameters

in		
----	--	--

Definition at line 80 of file hal_GPIO.c.

```
00080
00081
00082     GPIO_setConfig(Board_SCLK_ASGPIO,      GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
00083     GPIO_setConfig(Board_MOSI_ASGPIO,      GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
00084     GPIO_setConfig(Board_CS_ASGPIO,        GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
00085
00086     return 0;
00087 }
```

3.29.2.20 SPI_CS_Disable() `void SPI_CS_Disable (`
`void)`

GPIO CS SPI Disable Function.

This function

1. Configures CS SPI GPIO pin as High (Disable) for LDC1000, BME280 and ADXL355

Parameters

in		
----	--	--

Definition at line 373 of file hal_GPIO.c.

```
00373
00374
00375     GPIO_write(Board_LDC1000_CS, 1);
```

```

00376     GPIO_write(Board_BME280_CS,1);
00377     GPIO_write(Board_ADXL355_CS,1);
00378 }

```

3.30 hal_GPIO.c

```

00001
00010 //*****
00011 //          INCLUDES
00012 //*****
00013 #include <unistd.h>
00014 #include <ti/drivers/GPIO.h>
00015 #include "Board.h"
00016
00017 //*****
00018 //          FUNCTIONS
00019 //*****
00020
00021 //*****
00022 //
00031 //
00032 //*****
00033 uint8_t GPIO_Config(void) {
00034
00035     GPIO_setConfig(Board_RN2483_MCLR,      GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);           //MCLR Pin
00036     Configured, and set at Low
00036     GPIO_setConfig(Board_ADXL355_CS,      GPIO_CFG_OUT_STD | GPIO_CFG_OUT_HIGH);           //Selects the
00037     SPI device to interface (Active Low)
00037     GPIO_setConfig(Board_BME280_CS,      GPIO_CFG_OUT_STD | GPIO_CFG_OUT_HIGH);
00038     GPIO_setConfig(Board_LDC1000_CS,     GPIO_CFG_OUT_STD | GPIO_CFG_OUT_HIGH);
00039     GPIO_setConfig(Board_EN_ADXL355,    GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);           //Enable the
00039     individual LDO for sensor (active high)
00040     GPIO_setConfig(Board_EN_BME280,     GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
00041     GPIO_setConfig(Board_EN_LDC1000,   GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
00042     GPIO_setConfig(Board_DS1374_INTB,  GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);           //This
00042     connection no longer exists, so put it low
00043
00044     return 0;
00045 }
00046
00047 //*****
00048 //
00057 //
00058 //*****
00059 uint8_t I2C_As_GPIO_Low(void) {
00060
00061     GPIO_setConfig(Board_SCL_ASGPIO,    GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);           //Important to
00062     usleep(2000);
00063     avoid an Start condition on RTCC (see datasheet DS1374)
00063     GPIO_setConfig(Board_SDA_ASGPIO,    GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
00064
00065     return 0;
00066 }
00067
00068 //*****
00069 //
00078 //
00079 //*****
00080 uint8_t SPI_As_GPIO_Low(void) {
00081
00082     GPIO_setConfig(Board_SCLK_ASGPIO,   GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
00083     GPIO_setConfig(Board_MOSI_ASGPIO,   GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
00084     GPIO_setConfig(Board_CS_ASGPIO,    GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
00085
00086     return 0;
00087 }
00088
00089 //*****
00090 //
00099 //
00100 //*****
00101 void RN2483_Clear(void) {
00102
00103     GPIO_write(Board_RN2483_MCLR,0);
00104 }
00105
00106 //*****
00107 //
00116 //
00117 //*****
00118 void RN2483_Set(void) {
00119
00120     GPIO_write(Board_RN2483_MCLR,1);
00121 }

```

```

00122 //*****
00123 //*****
00124 //
00133 //
00134 //*****
00135 void ADXL355_Enable(void) {
00136     GPIO_write(Board_EN_ADXL355,1);
00138 }
00139
00140 //*****
00141 //
00150 //
00151 //*****
00152 void ADXL355_Disable(void) {
00153     GPIO_write(Board_EN_ADXL355,0);
00155 }
00156
00157 //*****
00158 //
00167 //
00168 //*****
00169 void BME280_Enable(void) {
00170     GPIO_write(Board_EN_BME280,1);
00172 }
00173
00174 //*****
00175 //
00184 //
00185 //*****
00186 void BME280_Disable(void) {
00187     GPIO_write(Board_EN_BME280,0);
00189 }
00190
00191 //*****
00192 //
00201 //
00202 //*****
00203 void LDC1000_Enable(void) {
00204
00205     GPIO_write(Board_EN_LDC1000,1);
00206 }
00207
00208 //*****
00209 //
00218 //
00219 //*****
00220 void LDC1000_Disable(void) {
00221
00222     GPIO_write(Board_EN_LDC1000,0);
00223 }
00224
00225 //*****
00226 //
00235 //
00236 //*****
00237 void Node_Enable(void) {
00238
00239     GPIO_write(Board_EN_NODE,1);
00240 }
00241
00242 //*****
00243 //
00252 //
00253 //*****
00254 void Node_Disable(void) {
00255
00256     GPIO_write(Board_EN_NODE,0);
00257 }
00258
00259 //*****
00260 //
00269 //
00270 //*****
00271 void ADXL355_SPI_Enable(void) {
00272
00273     GPIO_write(Board_ADXL355_CS,0);
00274 }
00275
00276 //*****
00277 //
00286 //
00287 //*****
00288 void ADXL355_SPI_Disable(void) {

```

```

00289     GPIO_write(Board_ADXL355_CS,1);
00290 }
00292
00293 //*****
00294 //
00295 //
00296 //*****
00297 void BME280_SPI_Enable(void) {
00298     GPIO_write(Board_BME280_CS,0);
00299 }
00300
00301 //*****
00302 //
00303 //
00304 //*****
00305 void BME280_SPI_Disable(void) {
00306     GPIO_write(Board_BME280_CS,1);
00307 }
00308
00309
00310 //*****
00311 //
00312 //
00313 //*****
00314 void LDC1000_SPI_Enable(void) {
00315     GPIO_write(Board_LDC1000_CS,0);
00316 }
00317
00318 //*****
00319 void LDC1000_SPI_Disable(void) {
00320     GPIO_write(Board_LDC1000_CS,1);
00321 }
00322
00323
00324 //*****
00325 void SPI_CS_Disable(void) {
00326     GPIO_write(Board_LDC1000_CS,1);
00327     GPIO_write(Board_BME280_CS,1);
00328     GPIO_write(Board_ADXL355_CS,1);
00329 }
00330
00331 //*****
00332 //
00333 //
00334 //*****
00335 void SPI_CS_Enable(void) {
00336     GPIO_write(Board_LDC1000_CS,0);
00337 }
00338
00339
00340 //*****
00341 void ADXL355_Disable(void) {
00342     GPIO_write(Board_ADXL355_CS,1);
00343 }
00344
00345 //*****
00346 void ADXL355_Enable(void) {
00347     GPIO_write(Board_ADXL355_CS,0);
00348 }
00349
00350
00351 //*****
00352 void ADXL355_SPI_Disable(void) {
00353     GPIO_write(Board_ADXL355_CS,1);
00354 }
00355
00356 //*****
00357 void ADXL355_SPI_Enable(void) {
00358     GPIO_write(Board_ADXL355_CS,0);
00359 }
00360
00361 //*****
00362 //
00363 //
00364 //*****
00365 void BME280_Disable(void) {
00366     GPIO_write(Board_BME280_CS,1);
00367 }
00368
00369
00370 //*****
00371 void BME280_Enable(void) {
00372     GPIO_write(Board_BME280_CS,0);
00373 }
00374
00375
00376 //*****
00377 void LDC1000_Disable(void) {
00378 }
00379
00380
00381 //*****
00382 
```

3.31 hal_GPIO.h File Reference

Functions for GPIO utilities.

Functions

- void [ADXL355_Disable](#) (void)
GPIO ADXL355 Disable Function.
- void [ADXL355_Enable](#) (void)
GPIO ADXL355 enable Function.
- void [ADXL355_SPI_Disable](#) (void)
GPIO ADXL355 SPI Disable Function.
- void [ADXL355_SPI_Enable](#) (void)
GPIO ADXL355 SPI enable Function.
- void [BME280_Disable](#) (void)
GPIO BME280 Disable Function.
- void [BME280_Enable](#) (void)
GPIO BME280 enable Function.
- void [BME280_SPI_Disable](#) (void)

GPIO BME280 SPI Disable Function.

- void **BME280_SPI_Enable** (void)
GPIO BME280 SPI enable Function.
- uint8_t **GPIO_Config** (void)
GPIO Configuration Function.
- uint8_t **I2C_As_GPIO_Low** (void)
GPIO I2C Low Function.
- void **LDC1000_Disable** (void)
GPIO LDC1000 Disable Function.
- void **LDC1000_Enable** (void)
GPIO LDC1000 enable Function.
- void **LDC1000_SPI_Disable** (void)
GPIO LDC1000 SPI Disable Function.
- void **LDC1000_SPI_Enable** (void)
GPIO LDC1000 SPI enable Function.
- void **Node_Disable** (void)
GPIO Node Disable Function.
- void **Node_Enable** (void)
GPIO Node enable Function.
- void **RN2483_Clear** (void)
GPIO RN2483 Clear Function.
- void **RN2483_Set** (void)
GPIO RN2483 Set Function.
- uint8_t **SPI_As_GPIO_Low** (void)
GPIO SPI Low Function.
- void **SPI_CS_Disable** (void)
GPIO CS SPI Disable Function.

3.31.1 Detailed Description

Functions for GPIO utilities.

Version

1.0

Date

07/05/2019

Author

A.Irizar @title GPIO functions

Definition in file [hal_GPIO.h](#).

3.31.2 Function Documentation

3.31.2.1 **ADXL355_Disable()** void ADXL355_Disable (void)

GPIO ADXL355 Disable Function.

This function

1. Configures ADXL355 GPIO pin as Low (Disable)

Parameters

in		
----	--	--

Definition at line 152 of file [hal_GPIO.c](#).

```
00152          {  
00153  
00154     GPIO_write(Board_EN_ADXL355,0);  
00155 }
```

3.31.2.2 ADXL355_Enable() void ADXL355_Enable (void)

GPIO ADXL355 enable Function.

This function

1. Configures ADXL355 GPIO pin as High (Enable)

Parameters

in		
----	--	--

Definition at line 135 of file [hal_GPIO.c](#).

```
00135          {  
00136  
00137     GPIO_write(Board_EN_ADXL355,1);  
00138 }
```

3.31.2.3 ADXL355_SPI_Disable() void ADXL355_SPI_Disable (void)

GPIO ADXL355 SPI Disable Function.

This function

1. Configures ADXL355 SPI GPIO pin as High (Disable)

Parameters

in		
----	--	--

Definition at line 288 of file [hal_GPIO.c](#).

```
00288          {  
00289  
00290     GPIO_write(Board_ADXL355_CS,1);  
00291 }
```

```
3.31.2.4 ADXL355_SPI_Enable() void ADXL355_SPI_Enable (
    void )
```

GPIO ADXL355 SPI enable Function.

This function

1. Configures ADXL355 SPI GPIO pin as Low (Enable)

Parameters

in		
----	--	--

Definition at line 271 of file [hal_GPIO.c](#).

```
00271 {
00272
00273     GPIO_write(Board_ADXL355_CS, 0);
00274 }
```

```
3.31.2.5 BME280_Disable() void BME280_Disable (
    void )
```

GPIO BME280 Disable Function.

This function

1. Configures BME280 GPIO pin as Low (Disable)

Parameters

in		
----	--	--

Definition at line 186 of file [hal_GPIO.c](#).

```
00186 {
00187
00188     GPIO_write(Board_EN_BME280, 0);
00189 }
```

```
3.31.2.6 BME280_Enable() void BME280_Enable (
    void )
```

GPIO BME280 enable Function.

This function

1. Configures BME280 GPIO pin as High (Enable)

Parameters

in		
----	--	--

Definition at line 169 of file [hal_GPIO.c](#).

```
00169      {
00170
00171     GPIO_write(Board_EN_BME280,1);
00172 }
```

3.31.2.7 BME280_SPI_Disable() void BME280_SPI_Disable (void)

GPIO BME280 SPI Disable Function.

This function

1. Configures BME280 SPI GPIO pin as High (Disable)

Parameters

in		
----	--	--

Definition at line 322 of file [hal_GPIO.c](#).

```
00322      {
00323
00324     GPIO_write(Board_BME280_CS,1);
00325 }
```

3.31.2.8 BME280_SPI_Enable() void BME280_SPI_Enable (void)

GPIO BME280 SPI enable Function.

This function

1. Configures BME280 SPI GPIO pin as Low (Enable)

Parameters

in		
----	--	--

Definition at line 305 of file [hal_GPIO.c](#).

```
00305      {
00306
00307     GPIO_write(Board_BME280_CS,0);
00308 }
```

3.31.2.9 GPIO_Config() uint8_t GPIO_Config (void)

GPIO Configuration Function.

This function

1. Configures CC3220SF GPIO pins

Parameters

in		
----	--	--

Definition at line 33 of file hal_GPIO.c.

```
00033           {
00034
00035     GPIO_setConfig(Board_RN2483_MCLR,      GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);          //MCLR Pin
00036     GPIO_setConfig(Board_ADXL355_CS,      GPIO_CFG_OUT_STD | GPIO_CFG_OUT_HIGH);           //Selects the
00037     SPI_device_to_interface (Active Low)
00038     GPIO_setConfig(Board_BME280_CS,        GPIO_CFG_OUT_STD | GPIO_CFG_OUT_HIGH);
00039     GPIO_setConfig(Board_LDC1000_CS,       GPIO_CFG_OUT_STD | GPIO_CFG_OUT_HIGH);
00040     GPIO_setConfig(Board_EN_ADXL355,       GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);            //Enable the
00041     individual LDO for sensor (active high)
00042     GPIO_setConfig(Board_EN_BME280,         GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
00043     GPIO_setConfig(Board_EN_LDC1000,        GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
00044     GPIO_setConfig(Board_DS1374_INTB,       GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);           //This
00045   connection no longer exists, so put it low
00046
00047   return 0;
00048 }
```

3.31.2.10 I2C_As_GPIO_Low() uint8_t I2C_As_GPIO_Low (void)

GPIO I2C Low Function.

This function

1. Configures CC3220SF I2C GPIO pin as Low

Parameters

in		
----	--	--

Definition at line 59 of file hal_GPIO.c.

```
00059           {
00060
00061     GPIO_setConfig(Board_SCL_ASGPIO,      GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);          //Important to
00062     usleep(2000);
00063     avoid an Start condition on RTCC (see datasheet DS1374)
00064     GPIO_setConfig(Board_SDA_ASGPIO,      GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
00065
00066 }
```

```
3.31.2.11 LDC1000_Disable() void LDC1000_Disable (
    void )
```

GPIO LDC1000 Disable Function.

This function

1. Configures LDC1000 GPIO pin as Low (Disable)

Parameters

in		
----	--	--

Definition at line 220 of file [hal_GPIO.c](#).

```
00220 {
00221
00222     GPIO_write(Board_EN_LDC1000,0);
00223 }
```

```
3.31.2.12 LDC1000_Enable() void LDC1000_Enable (
    void )
```

GPIO LDC1000 enable Function.

This function

1. Configures LDC1000 GPIO pin as High (Enable)

Parameters

in		
----	--	--

Definition at line 203 of file [hal_GPIO.c](#).

```
00203 {
00204
00205     GPIO_write(Board_EN_LDC1000,1);
00206 }
```

```
3.31.2.13 LDC1000_SPI_Disable() void LDC1000_SPI_Disable (
    void )
```

GPIO LDC1000 SPI Disable Function.

This function

1. Configures LDC1000 SPI GPIO pin as High (Disable)

Parameters

in		
----	--	--

Definition at line 356 of file [hal_GPIO.c](#).

```
00356 {  
00357  
00358     GPIO_write(Board_LDC1000_CS,1);  
00359 }
```

3.31.2.14 LDC1000_SPI_Enable() void LDC1000_SPI_Enable (void)

GPIO LDC1000 SPI enable Function.

This function

1. Configures LDC1000 SPI GPIO pin as Low (Enable)

Parameters

in		
----	--	--

Definition at line 339 of file [hal_GPIO.c](#).

```
00339 {  
00340  
00341     GPIO_write(Board_LDC1000_CS,0);  
00342 }
```

3.31.2.15 Node_Disable() void Node_Disable (void)

GPIO Node Disable Function.

This function

1. Configures Node GPIO pin as Low (Disable)

Parameters

in		
----	--	--

Definition at line 254 of file [hal_GPIO.c](#).

```
00254 {  
00255  
00256     GPIO_write(Board_EN_NODE,0);  
00257 }
```

```
3.31.2.16 Node_Enable() void Node_Enable (
    void )
```

GPIO [Node](#) enable Function.

This function

1. Configures [Node](#) GPIO pin as High (Enable)

Parameters

in		
----	--	--

Definition at line [237](#) of file [hal_GPIO.c](#).

```
00237     {
00238
00239     GPIO_write(Board_EN_NODE, 1);
00240 }
```

```
3.31.2.17 RN2483_Clear() void RN2483_Clear (
    void )
```

GPIO RN2483 Clear Function.

This function

1. Configures RN2483 GPIO pin as Low (clear)

Parameters

in		
----	--	--

Definition at line [101](#) of file [hal_GPIO.c](#).

```
00101     {
00102
00103     GPIO_write(Board_RN2483_MCLR, 0);
00104 }
```

```
3.31.2.18 RN2483_Set() void RN2483_Set (
    void )
```

GPIO RN2483 Set Function.

This function

1. Configures RN2483 GPIO pin as High (set)

Parameters

in		
----	--	--

Definition at line 118 of file hal_GPIO.c.

```
00118     {
00119
00120     GPIO_write(Board_RN2483_MCLR, 1);
00121 }
```

3.31.2.19 SPI_As_GPIO_Low() `uint8_t SPI_As_GPIO_Low (`
`void)`

GPIO SPI Low Function.

This function

1. Configures CC3220SF SPI GPIO pin as Low

Parameters

in		
----	--	--

Definition at line 80 of file hal_GPIO.c.

```
00080
00081
00082     GPIO_setConfig(Board_SCLK_ASGPIO,      GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
00083     GPIO_setConfig(Board_MOSI_ASGPIO,      GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
00084     GPIO_setConfig(Board_CS_ASGPIO,        GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
00085
00086     return 0;
00087 }
```

3.31.2.20 SPI_CS_Disable() `void SPI_CS_Disable (`
`void)`

GPIO CS SPI Disable Function.

This function

1. Configures CS SPI GPIO pin as High (Disable) for LDC1000, BME280 and ADXL355

Parameters

in		
----	--	--

Definition at line 373 of file hal_GPIO.c.

```
00373
00374
00375     GPIO_write(Board_LDC1000_CS, 1);
```

```
00376     GPIO_Write(Board_BME280_CS,1);
00377     GPIO_Write(Board_ADXL355_CS,1);
00378 }
```

3.32 hal_GPIO.h

```
00001
00010 #ifndef HAL_GPIO_H_
00011 #define HAL_GPIO_H_
00012
00013 //*****
00014 //          FUNCTION PROTOTYPES
00015 //*****
00016 uint8_t GPIO_Config(void);
00017 uint8_t I2C_As_GPIO_Low(void);
00018 uint8_t SPI_As_GPIO_Low(void);
00019 void RN2483_Clear(void);
00020 void RN2483_Set(void);
00021 void ADXL355_Enable(void);
00022 void ADXL355_Disable(void);
00023 void BME280_Enable(void);
00024 void BME280_Disable(void);
00025 void LDC1000_Enable(void);
00026 void LDC1000_Disable(void);
00027 void Node_Enable(void);
00028 void Node_Disable(void);
00029 void ADXL355_SPI_Enable(void);
00030 void ADXL355_SPI_Disable(void);
00031 void BME280_SPI_Enable(void);
00032 void BME280_SPI_Disable(void);
00033 void LDC1000_SPI_Enable(void);
00034 void LDC1000_SPI_Disable(void);
00035 void SPI_CS_Disable(void);
00036
00037 #endif /* HAL_GPIO_H_ */
```

3.33 hal_I2C.c File Reference

Functions for I2C protocol.

```
#include <ti/drivers/I2C.h>
#include "Board.h"
```

Functions

- `uint8_t I2C_read_8bits (I2C_Handle i2c, uint8_t Slave, uint8_t Address, uint8_t *RxBuffer, uint8_t nregs)`
I2C Read 8 bits Function.
- `uint8_t I2C_write_8bits (I2C_Handle i2c, uint8_t Slave, uint8_t Address, uint8_t Val)`
I2C Write 8 bits Function.
- `I2C_Handle Startup_I2C (uint_least8_t index)`
I2C Handle Function.

3.33.1 Detailed Description

Functions for I2C protocol.

Version

1.0

Date

07/05/2019

Author

A.Irizar @tittle I2C functions

Definition in file [hal_I2C.c](#).

3.33.2 Function Documentation

```
3.33.2.1 I2C_read_8bits() uint8_t I2C_read_8bits (
    I2C_Handle i2c,
    uint8_t Slave,
    uint8_t Address,
    uint8_t * RxBuffer,
    uint8_t nregs )
```

I2C Read 8 bits Function.

This function

1. Reads 8 bits from I2C register

Precondition

[Startup_I2C\(uint_least8_t index\)](#) must be called before

Parameters

in	<i>I2C_Handle</i>	i2c
in	<i>uint8_t</i>	Slave
in	<i>uint8_t</i>	Address
in	<i>uint8_t</i>	*RxBuffer Pointer to a buffer
in	<i>uint8_t</i>	nregs

Returns

bool TransferOK

Definition at line 65 of file [hal_I2C.c](#).

```
00065
00066 {
00067     I2C_Transaction i2cTransaction;
00068     uint8_t          TxBuffer;
00069     bool             TransferOK;
00070
00071     TxBuffer = Address;
00072
00073     i2cTransaction.slaveAddress = Slave;
00074     i2cTransaction.writeBuf = &TxBuffer;
00075     i2cTransaction.writeCount = 1;
00076     i2cTransaction.readBuf = RxBuffer;
00077     i2cTransaction.readCount = nregs;
00078
00079     TransferOK = I2C_transfer(i2c, &i2cTransaction);
00080
00081     return (TransferOK? 0 : 1);
00082 }
```

```
3.33.2.2 I2C_write_8bits() uint8_t I2C_write_8bits (
    I2C_Handle i2c,
    uint8_t Slave,
    uint8_t Address,
    uint8_t Val )
```

I2C Write 8 bits Function.

This function

1. Writes 8 bits to I2C register

Precondition

[Startup_I2C\(uint_least8_t index\)](#) must be called before

Parameters

in	<i>I2C_Handle</i>	i2c
in	<i>uint8_t</i>	Slave
in	<i>uint8_t</i>	Address
in	<i>uint8_t</i>	Val

Returns

bool TransferOK

Definition at line 100 of file [hal_I2C.c](#).

```
00100
00101
00102     I2C_Transaction i2cTransaction;
00103     uint8_t          TxBuffer[2];
00104     bool              TransferOK;
00105
00106     TxBuffer[0] = Address;
00107     TxBuffer[1] = Val;
00108
00109     i2cTransaction.slaveAddress = Slave;
00110     i2cTransaction.writeBuf = TxBuffer;
00111     i2cTransaction.writeCount = 2;
00112     i2cTransaction.readBuf = NULL;
00113     i2cTransaction.readCount = 0;
00114
00115     TransferOK = I2C_transfer(i2c, &i2cTransaction);
00116
00117     return (TransferOK? 0 : 1);
00118 }
```

```
3.33.2.3 Startup_I2C() I2C_Handle Startup_I2C (
    uint_least8_t index )
```

I2C Handle Function.

This function

1. Handle for I2C

Parameters

in	<i>uint_</i> ↔ <i>least8_t</i>	index
----	-----------------------------------	-------

Returns

I2C_Handle i2c

Definition at line 32 of file [hal_I2C.c](#).

```

00032
00033
00034     I2C_Params i2cParams;
00035     I2C_Handle i2c;
00036
00037     I2C_init();
00038     I2C_Params_init(&i2cParams);
00039
00040     i2cParams.transferMode = I2C_MODE_BLOCKING;
00041     i2cParams.bitRate = I2C_400kHz;
00042
00043     i2c = I2C_open(index, &i2cParams);
00044
00045     return i2c;
00046 }
```

3.34 hal_I2C.c

```

00001
00010 //*****
00011 //           INCLUDES
00012 //*****
00013 #include <ti/drivers/I2C.h>
00014 #include "Board.h"
00015
00016 //*****
00017 //           FUNCTIONS
00018 //*****
00019
00020 //*****
00021 //
00030 //
00031 //*****
00032 I2C_Handle Startup_I2C(uint_least8_t index) {
00033
00034     I2C_Params i2cParams;
00035     I2C_Handle i2c;
00036
00037     I2C_init();
00038     I2C_Params_init(&i2cParams);
00039
00040     i2cParams.transferMode = I2C_MODE_BLOCKING;
00041     i2cParams.bitRate = I2C_400kHz;
00042
00043     i2c = I2C_open(index, &i2cParams);
00044
00045     return i2c;
00046 }
00047
00048 //*****
00049 //
00063 //
00064 //*****
00065 uint8_t I2C_read_8bits(I2C_Handle i2c, uint8_t Slave, uint8_t Address, uint8_t *RxBuffer, uint8_t
nregs) {
00066
00067     I2C_Transaction i2cTransaction;
00068     uint8_t          TxBuffer;
00069     bool            TransferOK;
00070
00071     TxBuffer = Address;
00072
00073     i2cTransaction.slaveAddress = Slave;
00074     i2cTransaction.writeBuf = &TxBuffer;
00075     i2cTransaction.writeCount = 1;
00076     i2cTransaction.readBuf = RxBuffer;
00077     i2cTransaction.readCount = nregs;
```

```

00078     TransferOK = I2C_transfer(i2c, &i2cTransaction);
00080
00081     return (TransferOK? 0 : 1);
00082 }
00083
00084 //*****
00085 //
00086 //
00087 //*****
00088 //*****
00089 //*****
00100 uint8_t I2C_write_8bits(I2C_Handle i2c, uint8_t Slave, uint8_t Address, uint8_t Val) {
00101
00102     I2C_Transaction i2cTransaction;
00103     uint8_t TxBuffer[2];
00104     bool TransferOK;
00105
00106     TxBuffer[0] = Address;
00107     TxBuffer[1] = Val;
00108
00109     i2cTransaction.slaveAddress = Slave;
00110     i2cTransaction.writeBuf = TxBuffer;
00111     i2cTransaction.writeCount = 2;
00112     i2cTransaction.readBuf = NULL;
00113     i2cTransaction.readCount = 0;
00114
00115     TransferOK = I2C_transfer(i2c, &i2cTransaction);
00116
00117     return (TransferOK? 0 : 1);
00118 }
```

3.35 hal_I2C.h File Reference

Functions for I2C protocol.

```
#include <ti/drivers/I2C.h>
```

Functions

- `uint8_t I2C_read_8bits (I2C_Handle i2c, uint8_t Slave, uint8_t Address, uint8_t *RxBuffer, uint8_t nregs)`
I2C Read 8 bits Function.
- `uint8_t I2C_write_8bits (I2C_Handle i2c, uint8_t Slave, uint8_t Address, uint8_t Val)`
I2C Write 8 bits Function.
- `I2C_Handle Startup_I2C (uint_least8_t index)`
I2C Handle Function.

3.35.1 Detailed Description

Functions for I2C protocol.

Version

1.0

Date

07/05/2019

Author

A.Irizar @tittle I2C functions

Definition in file [hal_I2C.h](#).

3.35.2 Function Documentation

```
3.35.2.1 I2C_read_8bits() uint8_t I2C_read_8bits (
    I2C_Handle i2c,
    uint8_t Slave,
    uint8_t Address,
    uint8_t * RxBuffer,
    uint8_t nregs )
```

I2C Read 8 bits Function.

This function

1. Reads 8 bits from I2C register

Precondition

[Startup_I2C\(uint_least8_t index\)](#) must be called before

Parameters

in	<i>I2C_Handle</i>	i2c
in	<i>uint8_t</i>	Slave
in	<i>uint8_t</i>	Address
in	<i>uint8_t</i>	*RxBuffer Pointer to a buffer
in	<i>uint8_t</i>	nregs

Returns

bool TransferOK

Definition at line 65 of file [hal_I2C.c](#).

```
00065
00066 {
00067     I2C_Transaction i2cTransaction;
00068     uint8_t          TxBuffer;
00069     bool             TransferOK;
00070
00071     TxBuffer = Address;
00072
00073     i2cTransaction.slaveAddress = Slave;
00074     i2cTransaction.writeBuf = &TxBuffer;
00075     i2cTransaction.writeCount = 1;
00076     i2cTransaction.readBuf = RxBuffer;
00077     i2cTransaction.readCount = nregs;
00078
00079     TransferOK = I2C_transfer(i2c, &i2cTransaction);
00080
00081     return (TransferOK? 0 : 1);
00082 }
```

```
3.35.2.2 I2C_write_8bits() uint8_t I2C_write_8bits (
    I2C_Handle i2c,
    uint8_t Slave,
    uint8_t Address,
    uint8_t Val )
```

I2C Write 8 bits Function.

This function

1. Writes 8 bits to I2C register

Precondition

[Startup_I2C\(uint_least8_t index\)](#) must be called before

Parameters

in	<i>I2C_Handle</i>	i2c
in	<i>uint8_t</i>	Slave
in	<i>uint8_t</i>	Address
in	<i>uint8_t</i>	Val

Returns

bool TransferOK

Definition at line 100 of file [hal_I2C.c](#).

```
00100
00101
00102     I2C_Transaction i2cTransaction;
00103     uint8_t          TxBuffer[2];
00104     bool              TransferOK;
00105
00106     TxBuffer[0] = Address;
00107     TxBuffer[1] = Val;
00108
00109     i2cTransaction.slaveAddress = Slave;
00110     i2cTransaction.writeBuf = TxBuffer;
00111     i2cTransaction.writeCount = 2;
00112     i2cTransaction.readBuf = NULL;
00113     i2cTransaction.readCount = 0;
00114
00115     TransferOK = I2C_transfer(i2c, &i2cTransaction);
00116
00117     return (TransferOK? 0 : 1);
00118 }
```

```
3.35.2.3 Startup_I2C() I2C_Handle Startup_I2C (
    uint_least8_t index )
```

I2C Handle Function.

This function

1. Handle for I2C

Parameters

in	<i>uint_</i> ↔ <i>least8_t</i>	index
----	-----------------------------------	-------

Returns

I2C_Handle i2c

Definition at line 32 of file [hal_I2C.c](#).

```

00032
00033
00034     I2C_Params i2cParams;
00035     I2C_Handle i2c;
00036
00037     I2C_init();
00038     I2C_Params_init(&i2cParams);
00039
00040     i2cParams.transferMode = I2C_MODE_BLOCKING;
00041     i2cParams.bitRate = I2C_400kHz;
00042
00043     i2c = I2C_open(index, &i2cParams);
00044
00045     return i2c;
00046 }
```

3.36 hal_I2C.h

```

00001
00010 #ifndef HAL_I2C_H_
00011 #define HAL_I2C_H_
00012
00013 //*****
00014 //      INCLUDES
00015 //*****
00016 #include <ti/drivers/I2C.h>
00017
00018 //*****
00019 //      FUNCTION PROTOTYPES
00020 //*****
00021 I2C_Handle Startup_I2C(uint_least8_t index);
00022 uint8_t I2C_read_8bits(I2C_Handle i2c, uint8_t Slave, uint8_t Address, uint8_t *RxBuffer, uint8_t nregs);
00023 uint8_t I2C_write_8bits(I2C_Handle i2c, uint8_t Slave, uint8_t Address, uint8_t Val);
00024
00025 #endif /* HAL_I2C_H_ */
```

3.37 hal_LORA.c File Reference

Functions for LORA utilities.

```
#include <ti/drivers/UART.h>
#include <ti/drivers/GPIO.h>
#include <ti/drivers/net/wifi/simplelink.h>
#include "Board.h"
#include "hal_LORA.h"
#include "hal_UART.h"
#include "STARPORTS_App.h"
```

Functions

- void [GetLoraRxData](#) (unsigned char *buf, uint8_t size, struct [LoraNode](#) *MyLoraNode)
RN2483 Lora module Get Lora RX data Function.
- uint8_t [GetLoraServerParams](#) (uint8_t *bytes, uint8_t blen, struct [LoraNode](#) *MyLoraNode)
RN2483 Lora module Get Lora server parameters Function.
- uint8_t [hex2int](#) (unsigned char *hex, uint8_t hlen, uint8_t *bytes)
RN2483 Lora module Hex to int Function.
- uint8_t [Join_Abp_Lora](#) (UART_Handle uart)
RN2483 Lora module Join ABP Lora connection Function.
- uint8_t [Join_Otaa_Lora](#) (UART_Handle uart)
RN2483 Lora module join OTAA Lora connection Function.
- uint8_t [Mac_Adr_On](#) (UART_Handle uart)
RN2483 Lora module Mac ADR on Function.
- uint8_t [Mac_Ar_On](#) (UART_Handle uart)
RN2483 Lora module Mac AR on Function.
- uint8_t [Mac_Clear_Uptcr](#) (UART_Handle uart)
RN2483 Lora module Mac clear Up counter Function.
- uint8_t [Mac_Get_Appeui](#) (UART_Handle uart, struct [LoraNode](#) *MyLoraNode)
RN2483 Lora module Mac get Appeui Function.
- uint8_t [Mac_Get_Devaddr](#) (UART_Handle uart, struct [LoraNode](#) *MyLoraNode)
RN2483 Lora module Mac get Devaddr Function.
- uint8_t [Mac_Get_Deveui](#) (UART_Handle uart, struct [LoraNode](#) *MyLoraNode)
RN2483 Lora module Mac get Deveui Function.
- int [Mac_Get_Dnctr](#) (UART_Handle uart)
RN2483 Lora module Mac get Dnctr Function.
- int [Mac_Get_Uptcr](#) (UART_Handle uart)
RN2483 Lora module Mac get Uptcr Function.
- uint8_t [Mac_Save](#) (UART_Handle uart)
RN2483 Lora module Mac save Function.
- uint8_t [Mac_Set_Devaddr](#) (UART_Handle uart, struct [LoraNode](#) *MyLoraNode)
RN2483 Lora module Mac set Devaddr Function.
- uint8_t [Mac_Set_Dnctr](#) (UART_Handle uart, struct [LoraNode](#) *MyLoraNode)
RN2483 Lora module Mac set Dnctr Function.
- uint8_t [Mac_Set_Pwridx](#) (UART_Handle uart, uint8_t pwridx)
RN2483 Lora module Mac set Pwridx Function.
- uint8_t [Mac_Set_Rxdelay1](#) (UART_Handle uart, uint16_t delay)
RN2483 Lora module Mac set Rx delay 1 Function.
- uint8_t [Mac_Set_Uptcr](#) (UART_Handle uart, struct [LoraNode](#) *MyLoraNode)
RN2483 Lora module Mac set Uptcr Function.
- void [Reset_RN2483](#) (void)
RN2483 Lora module reset Function.
- uint8_t [Setup_Abp_Lora](#) (UART_Handle uart, struct [LoraNode](#) *MyLoraNode)
RN2483 Lora module Set up ABP Lora connection Function.
- uint8_t [Setup_Otaa_Lora](#) (UART_Handle uart, struct [LoraNode](#) *MyLoraNode)
RN2483 Lora module Set up OTAA Lora connection Function.
- uint8_t [Sys_FactoryReset](#) (UART_Handle uart)
RN2483 Lora module Sys Factory reset Function.
- uint8_t [Sys_Sleep](#) (UART_Handle uart, uint32_t sleep)
RN2483 Lora module Sys sleep Function.
- uint8_t [Try_Join_Lora_Gateway](#) (UART_Handle uart_dbg, UART_Handle uart_lora)

- RN2483 Lora module Try join Lora Gateway Function.
- `uint8_t Tx_Cnf_Lora` (`UART_Handle uart`, `struct LoraNode *MyLoraNode`, `uint8_t *mask`, `uint16_t *nodeId`)
RN2483 Lora module TX Cnf Function.
 - `uint8_t Tx_Uncnf_Lora` (`UART_Handle uart`, `struct LoraNode *MyLoraNode`, `uint8_t *mask`, `uint16_t *nodeId`)
RN2483 Lora module TX Uncnf Function.
 - `uint8_t Uint8Array2Char` (`uint8_t *DataPacket`, `uint8_t DataPacketLen`, `unsigned char *HexStr`)
RN2483 Lora module Uint8 array to char Function.

Variables

- `struct ADXL355_Data MyADXL`
- `struct BME280_Data MyBME`
- `struct LDC1000_Data MyLDC`
- `struct Node MyNode`
- `UART_Handle uart0`

3.37.1 Detailed Description

Functions for LORA utilities.

Version

1.0

Date

07/05/2019

Author

A.Irizar @tittle LORA functions

Definition in file [hal_LORA.c](#).

3.37.2 Function Documentation

3.37.2.1 GetLoraRxData() `void GetLoraRxData (`
`unsigned char * buf,`
`uint8_t size,`
`struct LoraNode * MyLoraNode)`

RN2483 Lora module Get Lora RX data Function.

This function

1. Gets data from Lora GW downlink

Parameters

in	<i>UART_Handle</i>	uart
	<i>struct</i>	LoraNode *MyLoraNode pointer to LoraNode struct

Returns

None

Definition at line 976 of file hal_LORA.c.

```

00976
00977
00978     uint8_t sz;
00979     uint8_t i, ini;
00980     const char space = ' ';
00981     const char cr = '\r';
00982     char *bufcp;
00983
00984     i=0;
00985     while (buf[i]!=space) {
00986         bufcp[i]=buf[i];
00987         i++;
00988     }
00989
00990     MyLoraNode->PortNoRx = atoi(bufcp);
00991
00992     i=i+1;
00993     ini = i;
00994     while (buf[i]!=cr) {
00995         *(MyLoraNode->DataRx+i-ini) = buf[i++];
00996     }
00997
00998     MyLoraNode->DataLenRx = (i-ini);
00999 }
```

3.37.2.2 GetLoraServerParams() `uint8_t GetLoraServerParams (`

```

    uint8_t * bytes,
    uint8_t blen,
    struct LoraNode * MyLoraNode )
```

RN2483 Lora module Get Lora server parameters Function.

This function

1. Gets Lora server parameters from Lora server

Parameters

in	<i>uint8_t</i>	*bytes pointer to bytes
in	<i>uint8_t</i>	blen
	<i>struct</i>	LoraNode *MyLoraNode pointer to LoraNode struct

Returns

0

Definition at line 1082 of file hal_LORA.c.

```

01082
01083
01084     uint8_t mask = bytes[0];
01085     int inext = 1;
01086
01087     if ((mask&0x01)!=0) {                                // WakeUpInterval
01088         MyNode.WakeUpInterval = (bytes[inext]«8) | bytes[inext+1];
01089         inext = inext+2;
01090         writeWakeUp(MyNode.WakeUpInterval);
01091         st_readFileWakeUp();
01092     }
01093
01094     if ((mask&0x02)!=0) {                                // Mode
01095         MyNode.Mode = (bytes[inext] & 0x07);
01096         inext = inext+1;
01097         writeMode(MyNode.Mode);
01098         st_read FileMode();
01099     }
01100
01101     if ((mask&0x04)!=0) {                                // SSID
01102         MyNode.SSID[5] = (char)(bytes[inext]);
01103         MyNode.SSID[4] = (char)(bytes[inext+1]);
01104         MyNode.SSID[3] = (char)(bytes[inext+2]);
01105         MyNode.SSID[2] = (char)(bytes[inext+3]);
01106         MyNode.SSID[1] = (char)(bytes[inext+4]);
01107         MyNode.SSID[0] = (char)(bytes[inext+5]);
01108         inext = inext + 6;
01109         writeSSID(MyNode.SSID);
01110         st_readFileSSID(&(MyNode.SSID));
01111     }
01112
01113     if ((mask&0x08)!=0) {                                // NCycles
01114         MyNode.NCycles = bytes[inext];
01115         inext = inext + 1;
01116         writeNCycles(MyNode.NCycles);
01117         st_readFileNCycles();
01118     }
01119
01120     if ((mask&0x10)!=0) {                                // Upcnt
01121         MyLoraNode->Upctr = (bytes[inext]«16) | (bytes[inext+1]«8) | (bytes[inext+2]);
01122         inext = inext + 3;
01123         writeUpCnt( MyLoraNode->Upctr );
01124         st_readFileUpCnt();
01125     }
01126
01127     if ((mask&0x20)!=0) {                                // ADXL355
01128         MyADXL.SampleRate = (bytes[inext]«8) | bytes[inext+1];
01129         MyADXL.NSamples = (bytes[inext+2]«8) | bytes[inext+3];
01130         inext = inext + 4;
01131     }
01132
01133     if ((mask&0x40)!=0) {                                // LDC1000
01134         MyLDC.NSamples = (bytes[inext]«8) | bytes[inext+1];
01135         inext = inext + 2;
01136     }
01137
01138     return 0;
01139 }
```

3.37.2.3 hex2int() uint8_t hex2int (

```

    unsigned char * hex,
    uint8_t hlen,
    uint8_t * bytes )
```

RN2483 Lora module Hex to int Function.

This function

1. Converts hexadecimal data to integer value

Parameters

in	<i>unsigned</i>	char *hex pointer to hex
in	<i>uint8_t</i>	hlen
	<i>uint8_t</i>	*bytes pointer to bytes

Returns

int k Returns the size of bytes array

Definition at line 1046 of file hal_LORA.c.

```

01046
01047
01048     uint32_t val = 0;
01049     uint8_t mybyte;
01050     int i;
01051     int k=0;
01052
01053     for (i=0;i<hlen;i++) {
01054         uint8_t mybyte = hex[i];
01055         current character then increment // get
01056         if (mybyte >= '0' && mybyte <= '9') mybyte = mybyte - '0';
01057         transform hex character to the 4bit equivalent number, using the ascii table indexes //
01058         else if (mybyte >= 'a' && mybyte <='f') mybyte = mybyte - 'a' + 10;
01059         else if (mybyte >= 'A' && mybyte <='F') mybyte = mybyte - 'A' + 10;
01060
01061         val = (val << 4) | (mybyte & 0xF);
01062         to make space for new digit, and add the 4 bits of the new digit // shift 4
01063         if ((i&0x01)==1) {
01064             bytes[k++]=val;
01065             val = 0;
01066         }
01067         return k;
01068     the size of bytes array
01069 }
```

3.37.2.4 Join_Abp_Lora() *uint8_t Join_Abp_Lora (*
 UART_Handle uart)

RN2483 Lora module Join ABP Lora connection Function.

This function

1. Function to join a lora connection using ABP (Authentification By Personalisation) in a RN2483.

Precondition[Setup_Abp_Lora\(\)](#) has been called**Parameters**

in	<i>UART_Handle</i>	uart
----	--------------------	------

ReturnsNone on Success / ERROR_ABP_JOIN on Fail according to [hal_LORA.h](#)

Definition at line 239 of file [hal_LORA.c](#).

```

00239
00240
00241     unsigned char Command[128];
00242     unsigned char buf[64];
00243     uint8_t sz;
00244
00245     strcpy(Command, "mac join abp\r\n");
00246     UART_write(uart, (const char *)Command, 14);
00247     sz = GetLine_UART(uart, buf);
00248     if (strncmp(buf, "ok", 2)==0) {
00249         sz = GetLine_UART(uart, buf);
00250         if (strncmp(buf, "denied", 6)==0) {
00251             return ERROR_ABP_DENIED;
00252         } else if (strncmp(buf, "accepted", 8)==0) {
00253             return SUCCESS_ABP_LORA;
00254         } else {
00255             return ERROR_ABP_JOIN;
00256         }
00257     } else {
00258         return ERROR_ABP_JOIN;
00259     }
00260 }
```

3.37.2.5 Join_Otaa_Lora() `uint8_t Join_Otaa_Lora (`
`UART_Handle uart)`

RN2483 Lora module join OTAA Lora connection Function.

This function

1. Function to join a lora connection using OTAA (Over the Air Activation) in a RN2483 based mote

Precondition

[Setup_Otaa_Lora\(\)](#) has been called

Parameters

in	<i>UART_Handle</i>	uart
----	--------------------	------

Returns

None on success / different ERROR Conditions according to [hal_LORA.h](#)

Definition at line 337 of file [hal_LORA.c](#).

```

00337
00338
00339     unsigned char Command[128];
00340     unsigned char buf[64];
00341     uint8_t sz;
00342
00343     memset(&buf, 0, sizeof(buf));
00344
00345     Mac_Adr_On(uart);
00346
00347     strcpy(Command, "mac join otaa\r\n");
00348     UART_write(uart, (const char *)Command, 15);
00349     sz = GetLine_UART(uart, buf);
00350     if (strncmp(buf, "ok", 2)==0) {
00351         sz = GetLine_UART(uart, buf);
00352         if (strncmp(buf, "denied", 6)==0) {
00353             return ERROR_OTAA_DENIED;
00354         } else if (strncmp(buf, "accepted", 8)==0) {
```

```

00355         return SUCCESS_OTAA_LORA;
00356     } else {
00357         return ERROR_OTAA_UNKNOWN;
00358     }
00359 } else if (strcmp(buf,"invalid_param",13)==0) {
00360     return ERROR_OTAA_INVALID_PARAM;
00361 } else if (strcmp(buf,"keys_not_init",13)==0) {
00362     return ERROR_OTAA_KEYS_NOT_INIT;
00363 } else if (strcmp(buf,"no_free_ch",10)==0) {
00364     return ERROR_OTAA_NO_FREE_CH;
00365 } else if (strcmp(buf,"silent",6)==0) {
00366     return ERROR_OTAA_SILENT;
00367 } else if (strcmp(buf,"busy",4)==0) {
00368     return ERROR_OTAA_BUSY;
00369 } else if (strcmp(buf,"mac_paused",10)==0) {
00370     return ERROR_OTAA_MAC_PAUSED;
00371 } else {
00372     return ERROR_OTAA_UNKNOWN;
00373 }
00374 }
```

3.37.2.6 Mac_Adr_On() uint8_t Mac_Adr_On (

UART_Handle *uart*)

RN2483 Lora module Mac ADR on Function.

This function

1. Sets adaptative data rate ON, on Lora RN2483 Module

Parameters

in	<i>UART_Handle</i>	<i>uart</i>
----	--------------------	-------------

Returns

0 on success / 1 on error

Definition at line 449 of file [hal_LORA.c](#).

```

00449
00450
00451     unsigned char Command[256];
00452     unsigned char buf[32];
00453     uint8_t sz;
00454
00455     memset(&buf, sizeof(buf));
00456     strcpy(Command,"mac set adr on\r\n");
00457     UART_write(uart, (const char *)Command, 16);
00458     sz = GetLine_UART(uart, buf);
00459     if (strcmp(buf,"ok",2)==0) {
00460         return 0;
00461     } else {
00462         return 1;
00463     }
00464 }
```

3.37.2.7 Mac_Ar_On() uint8_t Mac_Ar_On (

UART_Handle *uart*)

RN2483 Lora module Mac AR on Function.

This function

1. Sets automatic reply ON, on Lora RN2483 Module

Parameters

in	<i>UART_Handle</i>	uart
----	--------------------	------

Returns

0 on success / 1 on error

Definition at line 478 of file [hal_LORA.c](#).

```
00478
00479
00480     unsigned char Command[256];
00481     unsigned char buf[32];
00482     uint8_t sz;
00483
00484     memset (&buf, 0, sizeof(buf));
00485     strcpy(Command,"mac set ar on\r\n");
00486     UART_write(uart, (const char *)Command, 15);
00487     sz = GetLine_UART(uart, buf);
00488     if (strncmp(buf,"ok",2)==0) {
00489         return 0;
00490     } else {
00491         return 1;
00492     }
00493 }
```

3.37.2.8 Mac_Clear_Upctr() `uint8_t Mac_Clear_Upctr (`
 `UART_Handle uart)`

RN2483 Lora module Mac clear Up counter Function.

This function

1. Sets up counter parameter to 0 on Lora RN2483 Module

Parameters

in	<i>UART_Handle</i>	uart
----	--------------------	------

Returns

0 on success / 1 on fail

Definition at line 647 of file [hal_LORA.c](#).

```
00647
00648
00649     unsigned char Command[256];
00650     unsigned char buf[32];
00651     uint8_t sz;
00652
00653     strcpy(Command,"mac set upctr 0\r\n");
00654     UART_write(uart, (const char *)Command, 17);
00655     sz = GetLine_UART(uart, buf);
00656     if (strncmp(buf,"ok",2)==0) {
00657         return 0;
00658     } else {
00659         return 1;
00660     }
00661 }
```

```
3.37.2.9 Mac_Get_Appeui() uint8_t Mac_Get_Appeui (
    UART_Handle uart,
    struct LoraNode * MyLoraNode )
```

RN2483 Lora module Mac get Appeui Function.

This function

1. Gets Appeui for Lora RN2483 Module

Parameters

in	<i>UART_Handle</i>	uart
	<i>struct</i>	<i>LoraNode</i> *MyLoraNode pointer to <i>LoraNode</i> struct

Returns

0

Definition at line 129 of file [hal_LORA.c](#).

```
00129
00130     unsigned char Command[128];
00131     unsigned char buf[64];
00132     uint8_t sz;
00133
00134     memset(&buf, 0, sizeof(buf));
00135     strcpy(Command, "mac get appeui\r\n");
00136     UART_write(uart, (const char *) (Command), 16);
00137     sz = GetLine_UART(uart, buf);
00138     strncpy(MyLoraNode->AppEui, buf, 16);
00139
00140     return 0;
00141 }
```

```
3.37.2.10 Mac_Get_Devaddr() uint8_t Mac_Get_Devaddr (
    UART_Handle uart,
    struct LoraNode * MyLoraNode )
```

RN2483 Lora module Mac get Devaddr Function.

This function

1. Gets Devaddr for Lora RN2483 Module

Parameters

in	<i>UART_Handle</i>	uart
	<i>struct</i>	<i>LoraNode</i> *MyLoraNode pointer to <i>LoraNode</i> struct

Returns

0

Definition at line 156 of file [hal_LORA.c](#).

```

00156     unsigned char Command[256];
00157     unsigned char buf[32];
00158     uint8_t sz;
00159
00160     strcpy(Command, "mac get devaddr\r\n");
00161     UART_write(uart, (const char *) (Command), 17);
00162     sz = GetLine_UART(uart, buf);
00163     strncpy(MyLoraNode->DevAddr, buf, 8);
00164
00165     return 0;
00166 }
00167 }
```

3.37.2.11 Mac_Get_Deveui() `uint8_t Mac_Get_Deveui (`

`UART_Handle uart`
`struct LoraNode * MyLoraNode)`

RN2483 Lora module Mac get Deveui Function.

This function

1. Gets Deveui for Lora RN2483 Module

Parameters

in	<i>UART_Handle</i>	uart
	<i>struct LoraNode *</i>	MyLoraNode pointer to LoraNode struct

Returns

0

Definition at line 102 of file [hal_LORA.c](#).

```

00102
00103     unsigned char Command[128];
00104     unsigned char buf[64];
00105     uint8_t sz;
00106
00107     memset(&buf, 0, sizeof(buf));
00108     strcpy(Command, "mac get deveui\r\n");
00109     UART_write(uart, (const char *) (Command), 16);
00110     sz = GetLine_UART(uart, buf);
00111     strncpy(MyLoraNode->DevEui, buf, 16);
00112
00113     return 0;
00114 }
```

3.37.2.12 Mac_Get_Dnctr() `int Mac_Get_Dnctr (`

`UART_Handle uart)`

RN2483 Lora module Mac get Dnctr Function.

This function

1. Gets down counter parameter for Lora RN2483 Module

Parameters

in	<i>UART_Handle</i>	<i>uart</i>
----	--------------------	-------------

Returns

`atoi(buf)` value of the Dnctr parameter

Definition at line 592 of file hal_LORA.c.

```
00592
00593
00594     unsigned char Command[256];
00595     unsigned char buf[32];
00596     uint8_t sz;
00597
00598     strcpy(Command, "mac get dnctr\r\n");
00599     UART_write(uart, (const char *) (Command), 15);
00600     sz = GetLine_UART(uart, buf);
00601
00602     return atoi(buf);
00603 }
```

3.37.2.13 Mac_Get_Upctr() `int Mac_Get_Upctr (`
 `UART_Handle uart)`

RN2483 Lora module Mac get Upctr Function.

This function

1. Gets up counter parameter for Lora RN2483 Module

Parameters

in	<i>UART_Handle</i>	<i>uart</i>
----	--------------------	-------------

Returns

`atoi(buf)` value of the Upctr parameter

Definition at line 538 of file hal_LORA.c.

```
00538
00539
00540     unsigned char Command[256];
00541     unsigned char buf[32];
00542     uint8_t sz;
00543
00544     strcpy(Command, "mac get upctr\r\n");
00545     UART_write(uart, (const char *) (Command), 15);
00546     sz = GetLine_UART(uart, buf);
00547
00548     return atoi(buf);
00549 }
```

3.37.2.14 Mac_Save() `uint8_t Mac_Save (`
 `UART_Handle uart)`

RN2483 Lora module Mac save Function.

This function

1. Saves configuration parameters to Lora RN2483 Module

Parameters

in	<i>UART_Handle</i>	uart
----	--------------------	------

Returns

0 on success / 1 on error

Definition at line 417 of file [hal_LORA.c](#).

```
00417                                     {  
00418  
00419     unsigned char Command[256];  
00420     unsigned char buf[32];  
00421     uint8_t sz;  
00422  
00423     strcpy(Command, "mac save\r\n");  
00424  
00425     UART_write(uart, Command, 10);  
00426     sz = GetLine_UART(uart, buf);  
00427  
00428     if (strncmp(buf, "ok", 2)==0) {  
00429         return 0;  
00430     } else if (strncmp(buf, "invalid_param", 13)==0) {  
00431         return 1;  
00432     } else {  
00433         return 1;  
00434     }  
00435 }
```

3.37.2.15 Mac_Set_Devaddr() `uint8_t Mac_Set_Devaddr (`
 `UART_Handle uart,`
 `struct LoraNode * MyLoraNode)`

RN2483 Lora module Mac set Devaddr Function.

This function

1. Sets Devaddr on Lora RN2483 Module

Parameters

in	<i>UART_Handle</i>	uart
	<i>struct LoraNode *MyLoraNode</i>	pointer to <i>LoraNode</i> struct

Returns

SUCCESS_SET_DEVADDR / ERROR_SET_DEVADDR according to [hal_LORA.h](#)

Definition at line 508 of file [hal_LORA.c](#).

```

00508     unsigned char Command[256];
00509     unsigned char buf[32];
00510     uint8_t sz;
00511
00512     memset(&buf, 0, sizeof(buf));
00513     strcpy(Command, "mac set devaddr ");
00514     strncat(Command, MyLoraNode->DevAddr, sizeof(MyLoraNode->DevAddr));
00515     strcat(Command, "\r\n");
00516     UART_write(uart, (const char *)Command, 26);
00517     sz = GetLine_UART(uart, buf);
00518     if (strncmp(buf, "ok", 2) != 0) {
00519         return ERROR_SET_DEVADDR;
00520     } else {
00521         return SUCCESS_SET_DEVADDR;
00522     }
00523 }
00524 }
```

3.37.2.16 Mac_Set_Dnctr() `uint8_t Mac_Set_Dnctr (`
 `UART_Handle uart,`
 `struct LoraNode * MyLoraNode)`

RN2483 Lora module Mac set Dnctr Function.

This function

1. Sets down counter parameter on Lora RN2483 Module

Parameters

in	<i>UART_Handle</i>	uart
	<i>struct LoraNode *MyLoraNode</i>	pointer to <i>LoraNode</i> struct

Returns

ERROR_SET_DNCTR / SUCCESS_SET_DNCTR according to [hal_LORA.h](#)

Definition at line 618 of file [hal_LORA.c](#).

```

00618     unsigned char Command[256];
00619     unsigned char buf[32];
00620     uint8_t sz;
00621
00622     memset(&buf, 0, sizeof(buf));
00623     sprintf(Command, "mac set dnctr %d\r\n", MyLoraNode->Dnctr);
00624     UART_write(uart, (const char *)Command, strlen(Command));
00625     sz = GetLine_UART(uart, buf);
00626     if (strncmp(buf, "ok", 2) != 0) {
00627         return ERROR_SET_DNCTR;
00628     } else {
00629         return SUCCESS_SET_DNCTR;
00630     }
00631 }
00632 }
00633 }
```

3.37.2.17 Mac_Set_Pwridx() uint8_t Mac_Set_Pwridx (

```
UART_HandleTypeDef uart,
          uint8_t pwridx )
```

RN2483 Lora module Mac set Pwridx Function.

This function

1. Sets the output power to be used on next transmissions on Lora RN2483 Module

Parameters

in	<i>UART_HandleTypeDef</i>	uart
	<i>uint8_t</i>	pwridx decimal number representing the index output for the power value

Returns

`ERROR_SET_PWRIDX / SUCCESS_SET_PWRIDX` according to [hal_LORA.h](#)

Definition at line 676 of file [hal_LORA.c](#).

```
00676
00677     unsigned char Command[256];
00678     unsigned char buf[32];
00679     uint8_t sz;
00680
00681     memset(&buf, 0, sizeof(buf));
00682     sprintf(Command, "mac set pwridx %d\r\n", pwridx);
00683     UART_write(uart, (const char *)Command, strlen(Command));
00684     sz = GetLine_UART(uart, buf);
00685     if (strncmp(buf, "ok", 2) != 0) {
00686         return ERROR_SET_PWRIDX;
00687     } else {
00688         return SUCCESS_SET_PWRIDX;
00689     }
00690
00691 }
```

3.37.2.18 Mac_Set_Rxdelay1()

```
uint8_t Mac_Set_Rxdelay1 (
```

```
UART_HandleTypeDef uart,
                           uint16_t delay )
```

RN2483 Lora module Mac set Rx delay 1 Function.

This function

1. Sets RX delay 1 for Lora RN2483 Module

Parameters

in	<i>UART_HandleTypeDef</i>	uart
in	<i>uint16_t</i>	delay value of the delay

Returns

SUCCESS_RXDELAY1_SET / ERROR_RXDELAY1_SET according to [hal_LORA.h](#)

Definition at line 389 of file [hal_LORA.c](#).

```

00389
00390     unsigned char Command[128];
00391     unsigned char buf[64];
00392     uint8_t sz;
00393
00394     memset(&buf, 0, sizeof(buf));
00395     sprintf(Command, "mac set rxdelay1 %d\r\n", delay);
00396     UART_write(uart, (const char *)Command, strlen(Command));
00397     sz = GetLine_UART(uart, buf);
00398     if (strncmp(buf, "ok", 2) != 0) {
00399         return ERROR_RXDELAY1_SET;
00400     } else {
00401         return SUCCESS_RXDELAY1_SET;
00402     }
00403 }
```

3.37.2.19 Mac_Set_Upcstr() uint8_t Mac_Set_Upcstr (

```
    UART_Handle uart,
    struct LoraNode * MyLoraNode )
```

RN2483 Lora module Mac set Upctr Function.

This function

1. Sets up counter parameter on Lora RN2483 Module

Parameters

in	<i>UART_Handle</i>	uart
	<i>struct LoraNode</i> *	MyLoraNode pointer to LoraNode struct

Returns

SUCCESS_SET_UPCTR / ERROR_SET_UPCTR according to [hal_LORA.h](#)

Definition at line 564 of file [hal_LORA.c](#).

```

00564
00565     unsigned char Command[256];
00566     unsigned char buf[32];
00567     uint8_t sz;
00568
00569     memset(&buf, 0, sizeof(buf));
00570     sprintf(Command, "mac set upctr %d\r\n", MyLoraNode->Upctr);
00571     UART_write(uart, (const char *)Command, strlen(Command));
00572     sz = GetLine_UART(uart, buf);
00573     if (strncmp(buf, "ok", 2) != 0) {
00574         return ERROR_SET_UPCTR;
00575     } else {
00576         return SUCCESS_SET_UPCTR;
00577     }
00578 }
```

3.37.2.20 Reset_RN2483() void Reset_RN2483 (void)

RN2483 Lora module reset Function.

This function

1. Resets RN2483 Lora module

Parameters

in		
----	--	--

Activate MCLR Pin

Release MCLR Pin

Definition at line 52 of file [hal_LORA.c](#).

```
00052      {
00056     GPIO_write(Board_RN2483_MCLR, 0);
00057     usleep(50000);
00061     GPIO_write(Board_RN2483_MCLR, 1);
00062 }
```

3.37.2.21 Setup_Abp_Lora() uint8_t Setup_Abp_Lora (

```
    UART_HandleTypeDef uart,
    struct LoraNode * MyLoraNode )
```

RN2483 Lora module Set up ABP Lora connection Function.

This function

1. Function to setup a lora connection using ABP (Authentification By Personalisation) in a RN2483 based mote.
Uses constant values for device address, network key and appskey

Precondition

[Setup_Abp_Lora\(\)](#) has been called

Parameters

in	UART_HandleTypeDef	uart
	struct	LoraNode *MyLoraNode pointer to LoraNode struct

Returns

SUCCESS_ABPLORA Different SUCCESS/ERROR Conditions according to [hal_LORA.h](#)

Definition at line 183 of file [hal_LORA.c](#).

```

00183
00184
00185     unsigned char Command[128];
00186     unsigned char c, buf[64];
00187     uint8_t sz;
00188
00189     memset(&buf, 0, sizeof(buf));
00190
00191     strcpy(Command,"mac set devaddr ");
00192     strncat(Command, MyLoraNode->DevAddr,8);
00193     strncat(Command,"\\r\\n");
00194     UART_write(uart, (const char *)Command, 26);
00195     sz = GetLine\_UART(uart, buf);
00196     if (strncpy(buf,"ok",2)!=0) {
00197         return ERROR\_ABP\_DEVADDR;
00198     }
00199
00200     memset(&buf, 0, sizeof(buf));
00201
00202     strcpy(Command,"mac set nwkskey ");
00203     strncat(Command, MyLoraNode->NwksKey,32);
00204     strncat(Command,"\\r\\n");
00205     UART_write(uart, (const char *)Command, 50);
00206     sz = GetLine\_UART(uart, buf);
00207     if (strncpy(buf,"ok",2)!=0) {
00208         return ERROR\_ABP\_NWKSKEY;
00209     }
00210
00211     memset(&buf, 0, sizeof(buf));
00212     strcpy(Command,"mac set appskey ");
00213     strncat(Command, MyLoraNode->AppsKey,32);
00214     strncat(Command,"\\r\\n");
00215     UART_write(uart, (const char *)Command, 50);
00216     sz = GetLine\_UART(uart, buf);
00217     if (strncpy(buf,"ok",2)!=0) {
00218         return ERROR\_ABP\_APPSKEY;
00219     }
00220
00221     return SUCCESS\_ABP\_LORA;
00222
00223 }
```

3.37.2.22 Setup_Otaa_Lora() `uint8_t Setup_Otaa_Lora (`

```
        UART_Handle uart,
        struct LoraNode * MyLoraNode )
```

RN2483 Lora module Set up OTAA Lora connection Function.

This function

1. Function to setup a lora connection using OTAA (Over the Air Activation) in a RN2483 based mote

Precondition

`Setup_Otaa_Lora()` has been called

Parameters

in	<code>UART_Handle</code>	uart
	<code>struct LoraNode</code>	*MyLoraNode pointer to <code>LoraNode</code> struct

Returns

`SUCCESS_OTAA_LORA` Different SUCCESS/ERROR Conditions according to `hal_LORA.h`

Definition at line 277 of file [hal_LORA.c](#).

```

00277
00278
00279     unsigned char Command[128];
00280     unsigned char buf[64];
00281     uint8_t sz;
00282
00283     memset(&buf, 0, sizeof(buf));                                //Sys get HwEUI
00284     strcpy(Command,"sys get hweui\r\n");
00285     UART_write(uart, (const char *)Command, 15);
00286     sz = GetLine_UART(uart, buf);
00287     strncpy(MyLoraNode->DevEui,buf,16);
00288
00289     memset(&buf, 0, sizeof(buf));                                //Mac set DevEUI
00290     memset(&Command,0, sizeof(Command));
00291     strcpy(Command,"mac set deveui ");
00292     strncat(Command, MyLoraNode->DevEui,16);
00293     strcat(Command, "\r\n");
00294     UART_write(uart, (const char *)Command, 33);
00295     sz = GetLine_UART(uart, buf);
00296     if (strncpy(buf,"ok",2)!=0) {
00297         return ERROR_OTAA_DEVEUI;
00298     }
00299
00300     memset(&buf, 0, sizeof(buf));                                //Mac set AppEUI
00301     strcpy(Command,"mac set appeui ");
00302     strncat(Command, MyLoraNode->AppEui,16);
00303     strcat(Command, "\r\n");
00304     UART_write(uart, (const char *)Command, 33);
00305     sz = GetLine_UART(uart, buf);
00306     if (strncpy(buf,"ok",2)!=0) {
00307         return ERROR_OTAA_APPEUI;
00308     }
00309
00310     memset(&buf, 0, sizeof(buf));                                //Mac set Appkey
00311     strcpy(Command,"mac set appkey ");
00312     strncat(Command, MyLoraNode->AppKey,32);
00313     strcat(Command, "\r\n");
00314     UART_write(uart, (const char *)Command, 49);
00315     sz = GetLine_UART(uart, buf);
00316     if (strncpy(buf,"ok",2)!=0) {
00317         return ERROR_OTAA_APPKEY;
00318     }
00319
00320     return SUCCESS_OTAA_LORA;
00321 }
```

3.37.2.23 Sys_FactoryReset() `uint8_t Sys_FactoryReset (`
 `UART_Handle uart)`

RN2483 Lora module Sys Factory reset Function.

This function

1. Resets the RN2483 module to factory values

Parameters

in	<i>UART_Handle</i>	uart
----	--------------------	------

Returns

0

Definition at line 76 of file [hal_LORA.c](#).

```

00076
00077 {
```

```

00078     unsigned char Command[256];
00079     unsigned char buf[32];
00080     uint8_t sz;
00081
00082     strcpy(Command,"sys factoryRESET\r\n");
00083
00084     UART_write(uart, Command, strlen(Command));
00085     sz = GetLine_UART(uart, buf);
00086
00087     return 0;
00088 }
```

3.37.2.24 Sys_Sleep() uint8_t Sys_Sleep (

UART_Handle *uart*,
uint32_t *sleep*)

RN2483 Lora module Sys sleep Function.

This function

1. Puts the RN2483 module in sleep mode during sleep ms. This function returns when the device comes out of sleep mode

Precondition

[Sys_Sleep\(\)](#) has been called

Parameters

in	<i>UART_Handle</i>	<i>uart</i>
in	<i>uint32_t</i>	sleep Value of the sleep time in ms

Returns

0 on success / 1 on fail

Definition at line 943 of file [hal_LORA.c](#).

```

00943
00944
00945     unsigned char Command[256];
00946     unsigned char buf[32];
00947     uint8_t sz;
00948
00949     sprintf(Command,"sys sleep %d\r\n",sleep);
00950
00951     UART_write(uart, Command, strlen(Command));
00952     sz = GetLine_UART(uart, buf);
00953
00954     if (strncmp(buf,"ok",2)==0) {
00955         return 0;
00956     } else if (strncmp(buf,"invalid_param",13)==0) {
00957         return 1;
00958     } else {
00959         return 1;
00960     }
00961 }
```

3.37.2.25 Try_Join_Lora_Gateway() uint8_t Try_Join_Lora_Gateway (

```
UART_Handle uart_dbg,
UART_Handle uart_lora )
```

RN2483 Lora module Try join Lora Gateway Function.

This function

1. Does several tries to connect to GW using OTAA.

Parameters

in	<i>UART_Handle</i>	uart_dbg UART debug function
in	<i>UART_Handle</i>	uart

Returns

uint8_t ret

Definition at line 706 of file [hal_LORA.c](#).

```
00706
00707
00708     uint8_t nf;
00709     uint8_t wait;
00710     uint8_t ret;
00711     uint8_t sz;
00712     unsigned char Mess[32];
00713     unsigned char buf[256];
00714
00715
00716     nf=0;
00717     do {
00718         if (nf<3) {
00719             wait = 30;
00720         } else if (nf<6){
00721             wait = 60;
00722         } else {
00723             break;
00724         }
00725         ret = Join_Otaa_Lora(uart_lora);
00726         if (ret==SUCCESS_OTAA_LORA) {
00727             strcpy(Mess,"Join_Otaa_Lora() Success\r\n");
00728             UART_write(uart_dbg, Mess, 26);
00729         } else {
00730             strcpy(Mess,"Join_Otaa_Lora() Failed ");
00731             UART_write(uart_dbg, Mess, 24);
00732             Mess[0] = '('; Mess[1] = ret+48; Mess[2]=')';
00733             UART_write(uart_dbg, Mess,3);
00734             UART_write(uart_dbg, "\r\n",2);
00735             if (ret==ERROR_OTAA_DENIED) {
00736                 break;
00737             }
00738             nf++;
00739             sleep(wait);
00740         }
00741     } while (ret!=SUCCESS_OTAA_LORA);
00742
00743     return ret;
00744 }
```

3.37.2.26 Tx_Cnf_Lora() uint8_t Tx_Cnf_Lora (

```
UART_Handle uart,
struct LoraNode * MyLoraNode,
uint8_t * mask,
uint16_t * nodel d )
```

RN2483 Lora module TX Cnf Function.

This function

1. Transmits data using lora connection in a RN2483 based mote in confirmed mode

Precondition

[Tx_Cnf_Lora\(\)](#) has been called

Parameters

in	<i>UART_Handle</i>	uart
in	<i>struct</i>	LoraNode *MyLoraNode pointer to LoraNode struct
	<i>uint8_t</i>	*mask pointer to mask
	<i>uint16_t</i>	*nodeId pointer to nodeId

Returns

uint8_t ret Different SUCCESS / ERROR codes according to [hal_LORA.h](#)

Definition at line 853 of file [hal_LORA.c](#).

```

00853
00854
00855     unsigned char Command[256];
00856     unsigned char buf[32];
00857     unsigned char payload[40];
00858     uint8_t bytes[20];
00859     uint8_t PortNo;
00860     uint8_t sz;
00861     uint8_t ret;
00862     uint8_t LastAnswer = FALSE;
00863
00864     memset(bytes,0,sizeof(bytes));
00865
00866     Mac_Adr_On(uart);                                     // Set
00867     adaptive datarate ON
00868     Mac_Ar_On(uart);                                     // Set
00869     Automatic Retransmit ON
00870
00871     sprintf(Command,"mac tx cnf %d ", MyLoraNode->PortNoTx);
00872     strcat(Command,MyLoraNode->DataTx, MyLoraNode->DataLenTx);
00873     strcat(Command, "\r\n");
00874
00875     UART_write(uart, Command, strlen(Command));
00876
00877     *mask = 0;
00878     *nodeId = 0;
00879     while (LastAnswer==FALSE) {
00880         sz = GetLine_UART(uart, buf);
00881         if (strncmp(buf,"ok",2)==0) {
00882             ;
00883         } else if (strncmp(buf,"mac_tx_ok",9)==0) {
00884             LastAnswer=TRUE;
00885             ret = SUCCESS_TX_MAC_TX;
00886         } else if (strncmp(buf,"mac_rx %d %s\r\n",&PortNo,payload));
00887             sz = hex2int(payload, strlen(payload), bytes);
00888             GetLoraServerParams(bytes, sz, MyLoraNode);
00889             *mask = bytes[0];
00890             *nodeId = (bytes[1]«8) | bytes[2];
00891             ret = SUCCESS_TX_MAC_RX;
00892         } else if (strncmp(buf,"mac_err",7)==0) {
00893             LastAnswer=TRUE;
00894             ret = ERROR_TX_MAC_ERR;
00895         } else if (strncmp(buf,"invalid_data_len",16)==0) {
00896             LastAnswer=TRUE;
00897             ret = ERROR_TX_INVALID_DATA_LEN;
00898         } else if (strncmp(buf,"invalid_param",13)==0) {

```

```
00898     LastAnswer=TRUE;
00899     ret = ERROR_TX_INVALID_PARAM;
00900 } else if (strncmp(buf,"not_joined",10)==0) {
00901     LastAnswer=TRUE;
00902     ret = ERROR_TX_NOT_JOINEDret =00t-7800(Lee_ch{})]TJ0.880.50rg0.880.50RG[-600(else)]TJ0g0G04880.50rg0.880.50
00901     LastAnswer=TRUE;
00902     ret = ERROR_TX_NOT_JOINEDret =00t-7invalid_data_len{})]TJ0.880.50r60.880.50RG[-600(else)]TJ0g0G19880.50rg0.880.50
```

```

00769     uint8_t sz;
00770     uint8_t ret;
00771     uint8_t LastAnswer = FALSE;
00772
00773     memset(bytes, 0, sizeof(bytes));
00774
00775     sprintf(Command,"mac tx uncnf %d ", MyLoraNode->PortNoTx);
00776     strncat(Command,MyLoraNode->DataTx, MyLoraNode->DataLenTx);
00777     strncat(Command,"\\r\\n");
00778
00779     UART_write(uart, Command, strlen(Command));
00780
00781     *mask = 0;
00782     while (LastAnswer==FALSE) {
00783         sz = GetLine_UART(uart, buf);
00784         if (strncmp(buf,"ok",2)==0) {
00785             ;
00786         } else if (strncmp(buf,"mac_tx_ok",9)==0) {
00787             UART_PRINT("%s\\n",buf);
00788             LastAnswer=TRUE;
00789             ret = SUCCESS_TX_MAC_TX;
00790         } else if (strncmp(buf,"mac_rx ",7)==0) {
00791             sscanf(buf,"mac_rx %d %s\\r\\n",&PortNo,payload);
00792             UART_PRINT("%s\\n",buf);
00793             sz = hex2int(payload, strlen(payload), bytes);
00794             GetLoraServerParams(bytes, sz, MyLoraNode);
00795             *mask = bytes[0];
00796             *nodeId = (bytes[1]<<8) | bytes[2];
00797             LastAnswer=TRUE;
00798             ret = SUCCESS_TX_MAC_RX;
00799         } else if (strncmp(buf,"mac_err",7)==0) {
00800             LastAnswer=TRUE;
00801             ret = ERROR_TX_MAC_ERR;
00802         } else if (strncmp(buf,"invalid_data_len",16)==0) {
00803             LastAnswer=TRUE;
00804             ret = ERROR_TX_INVALID_DATA_LEN;
00805         } else if (strncmp(buf,"invalid_param",13)==0) {
00806             LastAnswer=TRUE;
00807             ret = ERROR_TX_INVALID_PARAM;
00808         } else if (strncmp(buf,"not_joined",10)==0) {
00809             LastAnswer=TRUE;
00810             ret = ERROR_TX_NOT_JOINED;
00811         } else if (strncmp(buf,"no_free_ch",10)==0) {
00812             LastAnswer=TRUE;
00813             ret = ERROR_TX_NO_FREE_CH;
00814         } else if (strncmp(buf,"silent",6)==0) {
00815             LastAnswer=TRUE;
00816             ret = ERROR_TX_SILENT;
00817         } else if (strncmp(buf,"frame_counter_err_rejoin_needed",31)==0) {
00818             LastAnswer=TRUE;
00819             ret = ERROR_TX_REJOIN_NEEDED;
00820         } else if (strncmp(buf,"busy",4)==0) {
00821             LastAnswer=TRUE;
00822             ret = ERROR_TX_BUSY;
00823         } else if (strncmp(buf,"mac_paused",10)==0) {
00824             LastAnswer=TRUE;
00825             ret = ERROR_TX_MAC_PAUSED;
00826         } else if (strncmp(buf,"invalid_data_len",16)==0) {
00827             LastAnswer=TRUE;
00828             ret = ERROR_TX_INVALID_DATA_LEN;
00829         } else {
00830             LastAnswer=TRUE;
00831             ret = ERROR_TX_UNKNOWN;
00832         }
00833     }
00834     return ret;
00835 }
```

3.37.2.28 Uint8Array2Char() uint8_t Uint8Array2Char (

```

        uint8_t * DataPacket,
        uint8_t DataPacketLen,
        unsigned char * HexStr )
```

RN2483 Lora module Uint8 array to char Function.

This function

1. Converts DataPacket (data from sensors) to hexadecimal value

Parameters

in	<i>uint8_t</i>	*DataPacket pointer to DataPacket struct
in	<i>uint8_t</i>	DataPacketLen
	<i>unsigned</i>	char *HexStr pointer to HexStr

3.37.3.5 uart0 UART_Handle uart0

Definition at line 65 of file [STARPORTS_App.c](#).

3.38 hal_LORA.c

```

00001 //*****
00010 //***** INCLUDES *****
00011 //***** INCLUDES *****
00012 //***** INCLUDES *****
00013 //#include <string.h>
00014 //#include <stdlib.h>
00015 //#include <stdio.h>
00016 //#include <unistd.h>
00017 #include <ti/drivers/UART.h>
00018 #include <ti/drivers/GPIO.h>
00019 #include <ti/drivers/net/wifi/simplelink.h>
00020 #include "Board.h"
00021 #include "hal_LORA.h"
00022 #include "hal_UART.h"
00023 //##include "file_system.h"
00024 #include "STARPORTS_App.h"
00025
00026
00027 //*****
00028 //***** GLOBALS *****
00029 //*****
00030 extern UART_Handle uart0;
00031 extern struct Node MyNode;
00032 extern struct ADXL355_Data MyADXL;
00033 extern struct BME280_Data MyBME;
00034 extern struct LDC1000_Data MyLDC;
00035
00036 //*****
00037 //***** FUNCTIONS *****
00038 //*****
00039
00040 //*****
00041 //
00050 //
00051 //*****
00052 void Reset_RN2483(void) {
00053     GPIO_write(Board_RN2483_MCLR, 0);
00054     usleep(50000);
00055     GPIO_write(Board_RN2483_MCLR, 1);
00056 }
00057
00058
00059
00060
00061
00062
00063
00064 //*****
00065 //
00066 //
00067 //*****
00068 uint8_t Sys_FactoryReset(UART_Handle uart) {
00069
00070     unsigned char Command[256];
00071     unsigned char buf[32];
00072     uint8_t sz;
00073
00074     strcpy(Command,"sys factoryRESET\r\n");
00075
00076     UART_write(uart, Command, strlen(Command));
00077     sz = GetLine_UART(uart, buf);
00078
00079     return 0;
00080 }
00081
00082
00083
00084
00085
00086
00087
00088
00089 //*****
00090 //
00091 //
00092 //*****
00093 uint8_t Mac_Get_Deveui(UART_Handle uart, struct LoraNode *MyLoraNode) {
00094
00095     unsigned char Command[128];
00096     unsigned char buf[64];
00097     uint8_t sz;
00098
00099     memset(&buf,0, sizeof(buf));
00100     strcpy(Command,"mac get deveui\r\n");
00101     UART_write(uart, (const char *) (Command), 16);
00102     sz = GetLine_UART(uart, buf);
00103     strncpy(MyLoraNode->DevEui,buf,16);
00104
00105     return 0;
00106 }
00107
00108
00109
00110
00111
00112
00113
00114
00115
00116 //*****

```

```

0017 //
0017 //
0018 //*****
0019 uint8_t Mac_Get_Appeui(UART_Handle uart, struct LoraNode *MyLoraNode) {
0020     unsigned char Command[128];
0021     unsigned char buf[64];
0022     uint8_t sz;
0023
0024     memset(&buf, 0, sizeof(buf));
0025     strcpy(Command,"mac get appeui\r\n");
0026     UART_write(uart, (const char *) (Command), 16);
0027     sz = GetLine_UART(uart, buf);
0028     strncpy(MyLoraNode->AppEui,buf,16);
0029
0030     return 0;
0031 }
0032
0033 //*****
0034 //*****
0035 //*****
0036 //*****
0037 //*****
0038 //*****
0039 //*****
0040 //*****
0041 //*****
0042 //*****
0043 //*****
0044 //*****
0045 //*****
0046 //*****
0047 //*****
0048 //*****
0049 //*****
0050 //*****
0051 //*****
0052 //*****
0053 //*****
0054 //*****
0055 //*****
0056 uint8_t Mac_Get_Devaddr(UART_Handle uart, struct LoraNode *MyLoraNode) {
0057     unsigned char Command[256];
0058     unsigned char buf[32];
0059     uint8_t sz;
0060
0061     strcpy(Command,"mac get devaddr\r\n");
0062     UART_write(uart, (const char *) (Command), 17);
0063     sz = GetLine_UART(uart, buf);
0064     strncpy(MyLoraNode->DevAddr,buf,8);
0065
0066     return 0;
0067 }
0068 //*****
0069 //*****
0070 //*****
0071 //*****
0072 //*****
0073 //*****
0074 //*****
0075 //*****
0076 //*****
0077 //*****
0078 //*****
0079 //*****
0080 //*****
0081 //*****
0082 //*****
0083 uint8_t Setup_Abp_Lora(UART_Handle uart, struct LoraNode *MyLoraNode) {
0084
0085     unsigned char Command[128];
0086     unsigned char c, buf[64];
0087     uint8_t sz;
0088
0089     memset(&buf, 0, sizeof(buf));
0090
0091     strcpy(Command,"mac set devaddr "); //Mac set Devaddr
0092     strncat(Command, MyLoraNode->DevAddr,8);
0093     strcat(Command,"\\r\\n");
0094     UART_write(uart, (const char *) Command, 26);
0095     sz = GetLine_UART(uart, buf);
0096     if (strncmp(buf,"ok",2)!=0) {
0097         return ERROR_ABP_DEVADDR;
0098     }
0099
0100     memset(&buf, 0, sizeof(buf));
0101
0102     strcpy(Command,"mac set nwkskey "); //Mac set NwSkey
0103     strncat(Command, MyLoraNode->NwksKey,32);
0104     strcat(Command,"\\r\\n");
0105     UART_write(uart, (const char *) Command, 50);
0106     sz = GetLine_UART(uart, buf);
0107     if (strncmp(buf,"ok",2)!=0) {
0108         return ERROR_ABP_NWKSKEY;
0109     }
0110
0111     memset(&buf, 0, sizeof(buf)); //Mac set AppSkey
0112     strcpy(Command,"mac set appskey ");
0113     strncat(Command, MyLoraNode->AppsKey,32);
0114     strcat(Command,"\\r\\n");
0115     UART_write(uart, (const char *) Command, 50);
0116     sz = GetLine_UART(uart, buf);
0117     if (strncmp(buf,"ok",2)!=0) {
0118         return ERROR_ABP_APPSKEY;
0119     }
0120
0121     return SUCCESS_ABP_LORA;
0122 }
0123
0124 //*****
0125 //*****
0126 //*****
0127 //*****
0128 //*****
0129 uint8_t Join_Abp_Lora(UART_Handle uart) {
0130
0131     unsigned char Command[128];
0132     unsigned char buf[64];

```

```

00243     uint8_t sz;
00244
00245     strcpy(Command,"mac join abp\r\n");
00246     UART_write(uart, (const char *)Command, 14);
00247     sz = GetLine_UART(uart, buf);
00248     if (strncmp(buf,"ok",2)==0) {
00249         sz = GetLine_UART(uart, buf);
00250         if (strncmp(buf,"denied",6)==0) {
00251             return ERROR_ABP_DENIED;
00252         } else if (strncmp(buf,"accepted",8)==0) {
00253             return SUCCESS_ABP_LORA;
00254         } else {
00255             return ERROR_ABP_JOIN;
00256         }
00257     } else {
00258         return ERROR_ABP_JOIN;
00259     }
00260 }
00261
00262 //*****
00263 //
00275 //
00276 //*****
00277 uint8_t Setup_Otaa_Lora(UART_Handle uart, struct LoraNode *MyLoraNode) {
00278
00279     unsigned char Command[128];
00280     unsigned char buf[64];
00281     uint8_t sz;
00282
00283     memset(&buf, 0, sizeof(buf));                                //Sys get HwEUI
00284     strcpy(Command,"sys get hwui\r\n");
00285     UART_write(uart, (const char *)Command, 15);
00286     sz = GetLine_UART(uart, buf);
00287     strncpy(MyLoraNode->DevEui,buf,16);
00288
00289     memset(&buf, 0, sizeof(buf));                                //Mac set DevEUI
00290     memset(&Command, 0, sizeof(Command));
00291     strcpy(Command,"mac set deveui ");
00292     strcat(Command, MyLoraNode->DevEui,16);
00293     strcat(Command, "\r\n");
00294     UART_write(uart, (const char *)Command, 33);
00295     sz = GetLine_UART(uart, buf);
00296     if (strncmp(buf,"ok",2)!=0) {
00297         return ERROR_OTAA_DEVEUI;
00298     }
00299
00300     memset(&buf, 0, sizeof(buf));                                //Mac set AppEUI
00301     strcpy(Command,"mac set appeui ");
00302     strcat(Command, MyLoraNode->AppEui,16);
00303     strcat(Command, "\r\n");
00304     UART_write(uart, (const char *)Command, 33);
00305     sz = GetLine_UART(uart, buf);
00306     if (strncmp(buf,"ok",2)!=0) {
00307         return ERROR_OTAA_APPEUI;
00308     }
00309
00310     memset(&buf, 0, sizeof(buf));                                //Mac set Appkey
00311     strcpy(Command,"mac set appkey ");
00312     strcat(Command, MyLoraNode->AppKey,32);
00313     strcat(Command, "\r\n");
00314     UART_write(uart, (const char *)Command, 49);
00315     sz = GetLine_UART(uart, buf);
00316     if (strncmp(buf,"ok",2)!=0) {
00317         return ERROR_OTAA_APPKEY;
00318     }
00319
00320     return SUCCESS_OTAA_LORA;
00321 }
00322
00323 //*****
00324 //
00335 //
00336 //*****
00337 uint8_t Join_Otaa_Lora(UART_Handle uart) {
00338
00339     unsigned char Command[128];
00340     unsigned char buf[64];
00341     uint8_t sz;
00342
00343     memset(&buf, 0, sizeof(buf));
00344
00345     Mac_Adr_On(uart);
00346
00347     strcpy(Command,"mac join otaa\r\n");
00348     UART_write(uart, (const char *)Command, 15);
00349     sz = GetLine_UART(uart, buf);
00350     if (strncmp(buf,"ok",2)==0) {

```

```

00351     sz = GetLine_UART(uart, buf);
00352     if (strncmp(buf,"denied",6)==0) {
00353         return ERROR_OTAA_DENIED;
00354     } else if (strncmp(buf,"accepted",8)==0) {
00355         return SUCCESS_OTAA_LORA;
00356     } else {
00357         return ERROR_OTAA_UNKNOWN;
00358     }
00359 } else if (strncmp(buf,"invalid_param",13)==0) {
00360     return ERROR_OTAA_INVALID_PARAM;
00361 } else if (strncmp(buf,"keys_not_init",13)==0) {
00362     return ERROR_OTAA_KEYS_NOT_INIT;
00363 } else if (strncmp(buf,"no_free_ch",10)==0) {
00364     return ERROR_OTAA_NO_FREE_CH;
00365 } else if (strncmp(buf,"silent",6)==0) {
00366     return ERROR_OTAA_SILENT;
00367 } else if (strncmp(buf,"busy",4)==0) {
00368     return ERROR_OTAA_BUSY;
00369 } else if (strncmp(buf,"mac_paused",10)==0) {
00370     return ERROR_OTAA_MAC_PAUSED;
00371 } else {
00372     return ERROR_OTAA_UNKNOWN;
00373 }
00374 }
00375
00376 //*****
00377 //
00387 //
00388 //*****
00389 uint8_t Mac_Set_Rxdelay1(UART_HandleTypeDef uart, uint16_t delay) {
00390     unsigned char Command[128];
00391     unsigned char buf[64];
00392     uint8_t sz;
00393
00394     memset(&buf, 0, sizeof(buf));
00395     sprintf(Command,"mac set rxdelay1 %d\r\n", delay);
00396     UART_write(uart, (const char *)Command, strlen(Command));
00397     sz = GetLine_UART(uart, buf);
00398     if (strncmp(buf,"ok",2)!=0) {
00399         return ERROR_RXDELAY1_SET;
00400     } else {
00401         return SUCCESS_RXDELAY1_SET;
00402     }
00403 }
00404
00405 //*****
00406 //
00415 //
00416 //*****
00417 uint8_t Mac_Save(UART_HandleTypeDef uart) {
00418
00419     unsigned char Command[256];
00420     unsigned char buf[32];
00421     uint8_t sz;
00422
00423     strcpy(Command,"mac save\r\n");
00424
00425     UART_write(uart, Command, 10);
00426     sz = GetLine_UART(uart, buf);
00427
00428     if (strncmp(buf,"ok",2)==0) {
00429         return 0;
00430     } else if (strncmp(buf,"invalid_param",13)==0) {
00431         return 1;
00432     } else {
00433         return 1;
00434     }
00435 }
00436
00437 //*****
00438 //
00447 //
00448 //*****
00449 uint8_t Mac_AdR_On(UART_HandleTypeDef uart) {
00450
00451     unsigned char Command[256];
00452     unsigned char buf[32];
00453     uint8_t sz;
00454
00455     memset(&buf, 0, sizeof(buf));
00456     strcpy(Command,"mac set adr on\r\n");
00457     UART_write(uart, (const char *)Command, 16);
00458     sz = GetLine_UART(uart, buf);
00459     if (strncmp(buf,"ok",2)==0) {
00460         return 0;
00461     } else {
00462         return 1;
00463     }

```

```
00463     }
00464 }
00465
00466 //*****
00467 //
00476 //
00477 //*****
00478 uint8_t Mac_Ar_On(UART_HandleTypeDef uart) {
00479
00480     unsigned char Command[256];
00481     unsigned char buf[32];
00482     uint8_t sz;
00483
00484     memset(&buf, sizeof(buf));
00485     strcpy(Command,"mac set ar on\r\n");
00486     UART_write(uart, (const char *)Command, 15);
00487     sz = GetLine_UART(uart, buf);
00488     if (strncmp(buf,"ok",2)==0) {
00489         return 0;
00490     } else {
00491         return 1;
00492     }
00493 }
00494
00495 //*****
00496 //
00506 //
00507 //*****
00508 uint8_t Mac_Set_Devaddr(UART_HandleTypeDef uart, struct LoraNode *MyLoraNode) {
00509     unsigned char Command[256];
00510     unsigned char buf[32];
00511     uint8_t sz;
00512
00513     memset(&buf, sizeof(buf));
00514     strcpy(Command,"mac set devaddr ");
00515     strncat(Command, MyLoraNode->DevAddr,sizeof(MyLoraNode->DevAddr));
00516     strcat(Command,"\\r\\n");
00517     UART_write(uart, (const char *)Command, 26);
00518     sz = GetLine_UART(uart, buf);
00519     if (strncmp(buf,"ok",2)!=0) {
00520         return ERROR_SET_DEVADDR;
00521     } else {
00522         return SUCCESS_SET_DEVADDR;
00523     }
00524 }
00525
00526 //*****
00527 //
00536 //
00537 //*****
00538 int Mac_Get_Uptcr(UART_HandleTypeDef uart) {
00539
00540     unsigned char Command[256];
00541     unsigned char buf[32];
00542     uint8_t sz;
00543
00544     strcpy(Command,"mac get upctr\\r\\n");
00545     UART_write(uart, (const char *)Command, 15);
00546     sz = GetLine_UART(uart, buf);
00547
00548     return atoi(buf);
00549 }
00550
00551 //*****
00552 //
00562 //
00563 //*****
00564 uint8_t Mac_Set_Uptcr(UART_HandleTypeDef uart, struct LoraNode *MyLoraNode) {
00565     unsigned char Command[256];
00566     unsigned char buf[32];
00567     uint8_t sz;
00568
00569     memset(&buf, sizeof(buf));
00570     sprintf(Command,"mac set upctr %d\\r\\n", MyLoraNode->Upctr);
00571     UART_write(uart, (const char *)Command, strlen(Command));
00572     sz = GetLine_UART(uart, buf);
00573     if (strncmp(buf,"ok",2)!=0) {
00574         return ERROR_SET_UPCTR;
00575     } else {
00576         return SUCCESS_SET_UPCTR;
00577     }
00578 }
00579
00580 //*****
00581 //
00590 //
00591 //*****
```

```

00592 int Mac_Get_Dnctr(UART_Handle uart) {
00593     unsigned char Command[256];
00595     unsigned char buf[32];
00596     uint8_t sz;
00597
00598     strcpy(Command,"mac get dnctr\r\n");
00599     UART_write(uart, (const char *)Command, 15);
00600     sz = GetLine_UART(uart, buf);
00601
00602     return atoi(buf);
00603 }
00604
00605 //*****
00606 //
00616 //
00617 //*****
00618 uint8_t Mac_Set_Dnctr(UART_Handle uart, struct LoraNode *MyLoraNode) {
00619     unsigned char Command[256];
00620     unsigned char buf[32];
00621     uint8_t sz;
00622
00623     memset(&buf,0, sizeof(buf));
00624     sprintf(Command,"mac set dnctr %d\r\n", MyLoraNode->Dnctr);
00625     UART_write(uart, (const char *)Command, strlen(Command));
00626     sz = GetLine_UART(uart, buf);
00627     if (strncmp(buf,"ok",2)!=0) {
00628         return ERROR_SET_DNCTR;
00629     } else {
00630         return SUCCESS_SET_DNCTR;
00631     }
00632
00633 }
00634
00635 //*****
00636 //
00645 //
00646 //*****
00647 uint8_t Mac_Clear_Upctr(UART_Handle uart) {
00648
00649     unsigned char Command[256];
00650     unsigned char buf[32];
00651     uint8_t sz;
00652
00653     strcpy(Command,"mac set upctr 0\r\n");
00654     UART_write(uart, (const char *)Command, 17);
00655     sz = GetLine_UART(uart, buf);
00656     if (strncmp(buf,"ok",2)==0) {
00657         return 0;
00658     } else {
00659         return 1;
00660     }
00661 }
00662
00663 //*****
00664 //
00674 //
00675 //*****
00676 uint8_t Mac_Set_Pwridx(UART_Handle uart, uint8_t pwridx) {
00677     unsigned char Command[256];
00678     unsigned char buf[32];
00679     uint8_t sz;
00680
00681     memset(&buf,0, sizeof(buf));
00682     sprintf(Command,"mac set pwridx %d\r\n", pwridx);
00683     UART_write(uart, (const char *)Command, strlen(Command));
00684     sz = GetLine_UART(uart, buf);
00685     if (strncmp(buf,"ok",2)!=0) {
00686         return ERROR_SET_PWRIDX;
00687     } else {
00688         return SUCCESS_SET_PWRIDX;
00689     }
00690
00691 }
00692
00693 //*****
00694 //
00704 //
00705 //*****
00706 uint8_t Try_Join_Lora_Gateway(UART_Handle uart_dbg, UART_Handle uart_lora) {
00707
00708     uint8_t nf;
00709     uint8_t wait;
00710     uint8_t ret;
00711     uint8_t sz;
00712     unsigned char Mess[32];
00713     unsigned char buf[256];

```

```

00714
00715
00716     nf=0;
00717     do {
00718         if (nf<3) {
00719             wait = 30;
00720         } else if (nf<6){
00721             wait = 60;
00722         } else {
00723             break;
00724         }
00725         ret = Join_Otaa_Lora(uart_lora);
00726         if (ret==SUCCESS_OTAA_LORA) {
00727             strcpy(Mess,"Join_Otaa_Lora() Success\r\n");
00728             UART_write(uart_dbg, Mess, 26);
00729         } else {
00730             strcpy(Mess,"Join_Otaa_Lora() Failed ");
00731             UART_write(uart_dbg, Mess, 24);
00732             Mess[0] = '('; Mess[1] = ret+48; Mess[2]=')';
00733             UART_write(uart_dbg, Mess,3);
00734             UART_write(uart_dbg, "\r\n",2);
00735             if (ret==ERROR_OTAA_DENIED) {
00736                 break;
00737             }
00738             nf++;
00739             sleep(wait);
00740         }
00741     } while (ret!=SUCCESS_OTAA_LORA);
00742
00743     return ret;
00744 }
00745
00746 //*****
00747 //
00759 //
00760 //*****
00761 uint8_t Tx_Uncnf_Lora(UART_Handle uart, struct LoraNode *MyLoraNode, uint8_t *mask, uint16_t *nodeId)
{
00762
00763     unsigned char Command[256];
00764     unsigned char buf[40];
00765     unsigned char payload[40];
00766     uint8_t bytes[20];
00767
00768     uint8_t PortNo;
00769     uint8_t sz;
00770     uint8_t ret;
00771     uint8_t LastAnswer = FALSE;
00772
00773     memset(bytes,0, sizeof(bytes));
00774
00775     sprintf(Command,"mac tx uncnf %d ", MyLoraNode->PortNoTx);
00776     strncat(Command,MyLoraNode->DataTx, MyLoraNode->DataLenTx);
00777     strncat(Command,"\r\n");
00778
00779     UART_write(uart, Command, strlen(Command));
00780
00781     *mask = 0;
00782     while (LastAnswer==FALSE) {
00783         sz = GetLine_UART(uart, buf);
00784         if (strncmp(buf,"ok",2)==0) {
00785             ;
00786         } else if (strncmp(buf,"mac_tx_ok",9)==0) {
00787             UART_PRINT("%s\r\n",buf);
00788             LastAnswer=TRUE;
00789             ret = SUCCESS_TX_MAC_TX;
00790         } else if (strncmp(buf,"mac_rx ",7)==0) {
00791             sscanf(buf,"%s %d %s\r\n",&PortNo,payload);
00792             UART_PRINT("%s\r\n",buf);
00793             sz = hex2int(payload, strlen(payload), bytes);
00794             GetLoraServerParams(bytes, sz, MyLoraNode);
00795             *mask = bytes[0];
00796             *nodeId = (bytes[1]<<8) | bytes[2];
00797             LastAnswer=TRUE;
00798             ret = SUCCESS_TX_MAC_RX;
00799         } else if (strncmp(buf,"mac_err",7)==0) {
00800             LastAnswer=TRUE;
00801             ret = ERROR_TX_MAC_ERR;
00802         } else if (strncmp(buf,"invalid_data_len",16)==0) {
00803             LastAnswer=TRUE;
00804             ret = ERROR_TX_INVALID_DATA_LEN;
00805         } else if (strncmp(buf,"invalid_param",13)==0) {
00806             LastAnswer=TRUE;
00807             ret = ERROR_TX_INVALID_PARAM;
00808         } else if (strncmp(buf,"not_joined",10)==0) {
00809             LastAnswer=TRUE;
00810             ret = ERROR_TX_NOT_JOINED;

```

```

00811     } else if (strncmp(buf, "no_free_ch", 10)==0) {
00812         LastAnswer=TRUE;
00813         ret = ERROR_TX_NO_FREE_CH;
00814     } else if (strncmp(buf, "silent", 6)==0) {
00815         LastAnswer=TRUE;
00816         ret = ERROR_TX_SILENT;
00817     } else if (strncmp(buf, "frame_counter_err_rejoin_needed", 31)==0) {
00818         LastAnswer=TRUE;
00819         ret = ERROR_TX_REJOIN_NEEDED;
00820     } else if (strncmp(buf, "busy", 4)==0) {
00821         LastAnswer=TRUE;
00822         ret = ERROR_TX_BUSY;
00823     } else if (strncmp(buf, "mac_paused", 10)==0) {
00824         LastAnswer=TRUE;
00825         ret = ERROR_TX_MAC_PAUSED;
00826     } else if (strncmp(buf, "invalid_data_len", 16)==0) {
00827         LastAnswer=TRUE;
00828         ret = ERROR_TX_INVALID_DATA_LEN;
00829     } else {
00830         LastAnswer=TRUE;
00831         ret = ERROR_TX_UNKNOWN;
00832     }
00833 }
00834 return ret;
00835 }
00836
00837 //*****
00838 //
00839 //
00840 //*****
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853 uint8_t Tx_Cnf_Lora(UART_HandleTypeDef uart, struct LoraNode *MyLoraNode, uint8_t *mask, uint16_t *nodeId) {
00854
00855     unsigned char Command[256];
00856     unsigned char buf[32];
00857     unsigned char payload[40];
00858     uint8_t bytes[20];
00859     uint8_t PortNo;
00860     uint8_t sz;
00861     uint8_t ret;
00862     uint8_t LastAnswer = FALSE;
00863
00864     memset(bytes, 0, sizeof(bytes));
00865
00866     Mac_Adr_On(uart); // Set
00867     adaptive datarate ON
00868     Mac_Ar_On(uart); // Set
00869     Automatic Retransmit ON
00870
00871     sprintf(Command,"mac tx cnf %d ", MyLoraNode->PortNoTx);
00872     strncat(Command,MyLoraNode->DataTx, MyLoraNode->DataLenTx);
00873     strncat(Command, "\r\n");
00874
00875     UART_write(uart, Command, strlen(Command));
00876
00877     *mask = 0;
00878     *nodeId = 0;
00879     while (LastAnswer==FALSE) {
00880         sz = GetLine_UART(uart, buf);
00881         if (strncmp(buf, "ok", 2)==0) {
00882             ;
00883         } else if (strncmp(buf, "mac_tx_ok", 9)==0) {
00884             LastAnswer=TRUE;
00885             ret = SUCCESS_TX_MAC_TX;
00886         } else if (strncmp(buf, "mac_rx %d %s\r\n", 7)==0) {
00887             sscanf(buf, "mac_rx %d %s\r\n", &PortNo, payload);
00888             sz = hex2int(payload, strlen(payload), bytes);
00889             GetLoraServerParams(bytes, sz, MyLoraNode);
00890             *mask = bytes[0];
00891             *nodeId = (bytes[1]«8) | bytes[2];
00892             ret = SUCCESS_TX_MAC_RX;
00893         } else if (strncmp(buf, "mac_err", 7)==0) {
00894             LastAnswer=TRUE;
00895             ret = ERROR_TX_MAC_ERR;
00896         } else if (strncmp(buf, "invalid_data_len", 16)==0) {
00897             LastAnswer=TRUE;
00898             ret = ERROR_TX_INVALID_DATA_LEN;
00899         } else if (strncmp(buf, "invalid_param", 13)==0) {
00900             LastAnswer=TRUE;
00901             ret = ERROR_TX_INVALID_PARAM;
00902         } else if (strncmp(buf, "not_joined", 10)==0) {
00903             LastAnswer=TRUE;
00904             ret = ERROR_TX_NOT_JOINED;
00905         } else if (strncmp(buf, "no_free_ch", 10)==0) {
00906             LastAnswer=TRUE;
00907             ret = ERROR_TX_NO_FREE_CH;
00908         } else if (strncmp(buf, "silent", 6)==0) {
00909             LastAnswer=TRUE;
00910         }
00911     }
00912 }
```

```

00908         ret = ERROR_TX_SILENT;
00909     } else if (strcmp(buf, "frame_counter_err_rejoin_needed", 31)==0) {
00910         LastAnswer=TRUE;
00911         ret = ERROR_TX_REJOIN_NEEDED;
00912     } else if (strcmp(buf, "busy", 4)==0) {
00913         LastAnswer=TRUE;
00914         ret = ERROR_TX_BUSY;
00915     } else if (strcmp(buf, "mac_paused", 10)==0) {
00916         LastAnswer=TRUE;
00917         ret = ERROR_TX_MAC_PAUSED;
00918     } else if (strcmp(buf, "invalid_data_len", 16)==0) {
00919         LastAnswer=TRUE;
00920         ret = ERROR_TX_INVALID_DATA_LEN;
00921     } else {
00922         LastAnswer=TRUE;
00923         ret = ERROR_TX_UNKNOWN;
00924     }
00925 }
00926 return ret;
00927 }
00928 //*****
00929 //*****
00930 //
00931 //
00932 //*****
00933 uint8_t Sys_Sleep(UART_HandleTypeDef uart, uint32_t sleep) {
00934
00935     unsigned char Command[256];
00936     unsigned char buf[32];
00937     uint8_t sz;
00938
00939     sprintf(Command,"sys sleep %d\r\n",sleep);
00940
00941     UART_write(uart, Command, strlen(Command));
00942     sz = GetLine_UART(uart, buf);
00943
00944     if (strcmp(buf, "ok", 2)==0) {
00945         return 0;
00946     } else if (strcmp(buf, "invalid_param", 13)==0) {
00947         return 1;
00948     } else {
00949         return 1;
00950     }
00951 }
00952
00953 //*****
00954 //*****
00955 //*****
00956 void GetLoraRxData(unsigned char *buf, uint8_t size, struct LoraNode *MyLoraNode) {
00957
00958     uint8_t sz;
00959     uint8_t i, ini;
00960     const char space = ' ';
00961     const char cr = '\r';
00962     char *bufcp;
00963
00964     i=0;
00965     while (buf[i]!=space) {
00966         bufcp[i]=buf[i];
00967         i++;
00968     }
00969
00970     MyLoraNode->PortNoRx = atoi(bufcp);
00971
00972     i=i+1;
00973     ini = i;
00974     while (buf[i]!=cr) {
00975         *(MyLoraNode->DataRx+i-ini) = buf[i++];
00976     }
00977
00978     MyLoraNode->DataLenRx = (i-ini);
00979 }
0100
0101 //*****
0102 //
0103 //
0104 //*****
0105 uint8_t Uint8Array2Char(uint8_t *DataPacket, uint8_t DataPacketLen, unsigned char *HexStr) {
0106
0107     uint8_t i, j, k;
0108     uint16_t a;
0109
0110     for (i=0;i<DataPacketLen;i++) {
0111         for (j=0;j<2;j++) {
0112             a = (DataPacket[i] >> (4-(j<<2))) & 0x0f;
0113             k = (i<<1)+j;
0114             HexStr[k] = ((a>>4)&0x0f)|0x30;
0115             if (a<0x10)
0116                 HexStr[k+1] = '0';
0117             else
0118                 HexStr[k+1] = ((a)&0x0f)|0x30;
0119         }
0120     }
0121
0122     return 0;
0123 }

```

```

01024         * (HexStr+k) = (a<10)? a+48 : a+55;
01025     }
01026     HexStr[k+1] = '\0';
01027     return k+1;
01028 }
01029
01030 }
01031
01032 //*****
01033 //
01044 //
01045 //*****
01046 uint8_t hex2int(unsigned char *hex, uint8_t hlen, uint8_t *bytes) {
01047
01048     uint32_t val = 0;
01049     uint8_t mybyte;
01050     int i;
01051     int k=0;
01052
01053     for (i=0;i<hlen;i++) {
01054         uint8_t mybyte = hex[i];
01055         current character then increment
01056         if (mybyte >= '0' && mybyte <= '9') mybyte = mybyte - '0';
01057         transform hex character to the 4bit equivalent number, using the ascii table indexes
01058         else if (mybyte >= 'a' && mybyte <='f') mybyte = mybyte - 'a' + 10;
01059         else if (mybyte >= 'A' && mybyte <='F') mybyte = mybyte - 'A' + 10;
01060
01061         val = (val << 4) | (mybyte & 0xF);
01062         to make space for new digit, and add the 4 bits of the new digit
01063         if ((i&0x01)==1) {
01064             bytes[k++]=val;
01065             val = 0;
01066         }
01067     }
01068 //*****
01069 //
01080 //
01081 //*****
01082 uint8_t GetLoraServerParams(uint8_t *bytes, uint8_t blen, struct LoraNode *MyLoraNode) {
01083
01084     uint8_t mask = bytes[0];
01085     int inext = 1;
01086
01087     if ((mask&0x01)!=0) { //WakeUpInterval
01088         MyNode.WakeUpInterval = (bytes[inext]<<8) | bytes[inext+1];
01089         inext = inext+2;
01090         writeWakeUp(MyNode.WakeUpInterval);
01091         st_readFileWakeUp();
01092     }
01093
01094     if ((mask&0x02)!=0) { //Mode
01095         MyNode.Mode = (bytes[inext] & 0x07);
01096         inext = inext+1;
01097         writeMode(MyNode.Mode);
01098         st_read FileMode();
01099     }
01100
01101     if ((mask&0x04)!=0) { //SSID
01102         MyNode.SSID[5] = (char)(bytes[inext]);
01103         MyNode.SSID[4] = (char)(bytes[inext+1]);
01104         MyNode.SSID[3] = (char)(bytes[inext+2]);
01105         MyNode.SSID[2] = (char)(bytes[inext+3]);
01106         MyNode.SSID[1] = (char)(bytes[inext+4]);
01107         MyNode.SSID[0] = (char)(bytes[inext+5]);
01108         inext = inext + 6;
01109         writeSSID(MyNode.SSID);
01110         st_readFileSSID(&(MyNode.SSID));
01111     }
01112
01113     if ((mask&0x08)!=0) { //Ncycles
01114         MyNode.NCycles = bytes[inext];
01115         inext = inext + 1;
01116         writeNCycles(MyNode.NCycles);
01117         st_readFileNCycles();
01118     }
01119
01120     if ((mask&0x10)!=0) { //Upcnt
01121         MyLoraNode->Upctr = (bytes[inext]<<16) | (bytes[inext+1]<<8) | (bytes[inext+2]);
01122         inext = inext + 3;
01123         writeUpCtr(MyLoraNode->Upctr);
01124         st_readFileUpCtr();
01125     }

```

```

01126
01127     if ((mask&0x20)!=0) {
01128         MyADXL.SampleRate = (bytes[inext]«8) | bytes[inext+1];
01129         MyADXL.NSamples = (bytes[inext+2]«8) | bytes[inext+3];
01130         inext = inext + 4;
01131     }
01132
01133     if ((mask&0x40)!=0) {
01134         MyLDC.NSamples = (bytes[inext]«8) | bytes[inext+1];
01135         inext = inext + 2;
01136     }
01137
01138     return 0;
01139 }
```

3.39 hal_LORA.h File Reference

Functions for LORA utilities.

```
#include <ti/drivers/UART.h>
```

Data Structures

- struct [LoraNode](#)
LORA node parameters. [More...](#)

Macros

- #define ERROR_APPSKEY 3
- #define ERROR_DENIED 6
- #define ERROR_DEVADDR 1
- #define ERROR_JOIN 5
- #define ERROR_NWKSKEY 2
- #define ERROR_SAVE 4
- #define ERROR_SET 14
- #define ERROR_APPEUI 2
- #define ERROR_APPKEY 3
- #define ERROR_BUSY 12
- #define ERROR_DENIED 6
- #define ERROR_DEVEUI 1
- #define ERROR_INVALID_PARAM 8
- #define ERROR_JOIN 5
- #define ERROR_KEYS_NOT_INIT 9
- #define ERROR_MAC_PAUSED 13
- #define ERROR_NO_FREE_CH 10
- #define ERROR_SAVE 4
- #define ERROR_SILENT 11
- #define ERROR_UNKNOWN 7
- #define ERROR_RXDELAY1_SET 1
- #define ERROR_SET_DEVADDR 1
- #define ERROR_SET_DNCTR 1
- #define ERROR_SET_NWSKEY 1
- #define ERROR_SET_PWRIDX 1
- #define ERROR_SET_UPCTR 1
- #define ERROR_TX_BUSY 2

- #define ERROR_TX_INVALID_DATA_LEN 6
- #define ERROR_TX_INVALID_PARAM 7
- #define ERROR_TX_MAC_ERR 1
- #define ERROR_TX_MAC_PAUSED 10
- #define ERROR_TX_NO_FREE_CH 4
- #define ERROR_TX_NOT_JOINED 3
- #define ERROR_TX_REJOIN_NEEDED 9
- #define ERROR_TX_SILENT 8
- #define ERROR_TX_UNKNOWN 5
- #define SUCCESS_ABP_LORA 0
- #define SUCCESS_OTAA_LORA 0
- #define SUCCESS_RXDELAY1_SET 0
- #define SUCCESS_SET_DEVADDR 0
- #define SUCCESS_SET_DNCTR 0
- #define SUCCESS_SET_NWSKEY 0
- #define SUCCESS_SET_PWRIDX 0
- #define SUCCESS_SET_UPCTR 0
- #define SUCCESS_TX_MAC_RX 11
- #define SUCCESS_TX_MAC_TX 0

Functions

- void [GetLoraRxData](#) (unsigned char *buf, uint8_t size, struct [LoraNode](#) *MyLoraNode)

RN2483 Lora module Get Lora RX data Function.
- uint8_t [GetLoraServerParams](#) (uint8_t *bytes, uint8_t blen, struct [LoraNode](#) *MyLoraNode)

RN2483 Lora module Get Lora server parameters Function.
- uint8_t [hex2int](#) (unsigned char *hex, uint8_t hlen, uint8_t *bytes)

RN2483 Lora module Hex to int Function.
- uint8_t [Join_Abp_Lora](#) (UART_Handle uart)

RN2483 Lora module Join ABP Lora connection Function.
- uint8_t [Join_Otaa_Lora](#) (UART_Handle uart)

RN2483 Lora module join OTAA Lora connection Function.
- uint8_t [Mac_Adr_On](#) (UART_Handle uart)

RN2483 Lora module Mac ADR on Function.
- uint8_t [Mac_Ar_On](#) (UART_Handle uart)

RN2483 Lora module Mac AR on Function.
- uint8_t [Mac_Clear_Upctr](#) (UART_Handle uart)

RN2483 Lora module Mac clear Up counter Function.
- uint8_t [Mac_Get_Appeui](#) (UART_Handle uart, struct [LoraNode](#) *MyLoraNode)

RN2483 Lora module Mac get Appeui Function.
- uint8_t [Mac_Get_Devaddr](#) (UART_Handle uart, struct [LoraNode](#) *MyLoraNode)

RN2483 Lora module Mac get Devaddr Function.
- uint8_t [Mac_Get_Deveui](#) (UART_Handle uart, struct [LoraNode](#) *MyLoraNode)

RN2483 Lora module Mac get Deveui Function.
- int [Mac_Get_Dnctr](#) (UART_Handle uart)

RN2483 Lora module Mac get Dnctr Function.
- int [Mac_Get_Upctr](#) (UART_Handle uart)

RN2483 Lora module Mac get Upctr Function.
- uint8_t [Mac_Save](#) (UART_Handle uart)

RN2483 Lora module Mac save Function.
- uint8_t [Mac_Set_Devaddr](#) (UART_Handle uart, struct [LoraNode](#) *MyLoraNode)

- `uint8_t Mac_Set_Dnctr (UART_Handle uart, struct LoraNode *MyLoraNode)`
RN2483 Lora module Mac set Devaddr Function.
- `uint8_t Mac_Set_Pwridx (UART_Handle uart, uint8_t pwridx)`
RN2483 Lora module Mac set Pwridx Function.
- `uint8_t Mac_Set_Rxdelay1 (UART_Handle uart, uint16_t delay)`
RN2483 Lora module Mac set Rx delay 1 Function.
- `uint8_t Mac_Set_Upctr (UART_Handle uart, struct LoraNode *MyLoraNode)`
RN2483 Lora module Mac set Upctr Function.
- `void Reset_RN2483 (void)`
RN2483 Lora module reset Function.
- `uint8_t Setup_Abp_Lora (UART_Handle uart, struct LoraNode *MyLoraNode)`
RN2483 Lora module Set up ABP Lora connection Function.
- `uint8_t Setup_Otaa_Lora (UART_Handle uart, struct LoraNode *MyLoraNode)`
RN2483 Lora module Set up OTAA Lora connection Function.
- `uint8_t Sys_FactoryReset (UART_Handle uart)`
RN2483 Lora module Sys Factory reset Function.
- `uint8_t Sys_Sleep (UART_Handle uart, uint32_t sleep)`
RN2483 Lora module Sys sleep Function.
- `uint8_t Try_Join_Lora_Gateway (UART_Handle uart_dbg, UART_Handle uart_lora)`
RN2483 Lora module Try join Lora Gateway Function.
- `uint8_t Tx_Cnf_Lora (UART_Handle uart, struct LoraNode *MyLoraNode, uint8_t *mask, uint16_t *nodeId)`
RN2483 Lora module TX Cnf Function.
- `uint8_t Tx_Uncnf_Lora (UART_Handle uart, struct LoraNode *MyLoraNode, uint8_t *mask, uint16_t *node←Id)`
RN2483 Lora module TX Uncnf Function.
- `uint8_t Uint8Array2Char (uint8_t *DataPacket, uint8_t DataPacketLen, unsigned char *HexStr)`
RN2483 Lora module Uint8 array to char Function.

3.39.1 Detailed Description

Functions for LORA utilities.

Version

1.0

Date

07/05/2019

Author

A.Irizar @tittle LORA functions

Definition in file [hal_LORA.h](#).

3.39.2 Data Structure Documentation

3.39.2.1 struct LoraNode LORA node parameters.

Definition at line 83 of file [hal_LORA.h](#).

Data Fields

unsigned char	AppEui[16]	Lora Node AppEui
unsigned char	AppKey[32]	Lora Node AppKey
unsigned char	AppsKey[32]	Lora Node AppsKey
uint8_t	DataLenRx	Size of Data received from downlink
uint8_t	DataLenTx	Size of data to transmit
unsigned char	DataRx[4]	Data received from downlink
unsigned char	DataTx[128]	Data to trasnmit
unsigned char	DevAddr[8]	Lora Node DevAddr
unsigned char	DevEui[16]	Lora Node DevEui
uint32_t	Dnctr	Down counter Lora parameter
unsigned char	NwksKey[32]	Lora Node NwksKey
uint8_t	PortNoRx	Rx number port
uint8_t	PortNoTx	Tx number port
uint32_t	Upctr	Up counter Lora parameter

3.39.3 Macro Definition Documentation**3.39.3.1 ERROR_ABP_APPSKEY** `#define ERROR_ABP_APPSKEY 3`

Definition at line [24](#) of file [hal_LORA.h](#).

3.39.3.2 ERROR_ABP_DENIED `#define ERROR_ABP_DENIED 6`

Definition at line [27](#) of file [hal_LORA.h](#).

3.39.3.3 ERROR_ABP_DEVADDR `#define ERROR_ABP_DEVADDR 1`

Definition at line [22](#) of file [hal_LORA.h](#).

3.39.3.4 ERROR_ABP_JOIN `#define ERROR_ABP_JOIN 5`

Definition at line [26](#) of file [hal_LORA.h](#).

3.39.3.5 ERROR_ABPNWKSKEY #define ERROR_ABPNWKSKEY 2

Definition at line 23 of file [hal_LORA.h](#).

3.39.3.6 ERROR_ABPSAVE #define ERROR_ABPSAVE 4

Definition at line 25 of file [hal_LORA.h](#).

3.39.3.7 ERRORADRSET #define ERRORADRSET 14

Definition at line 58 of file [hal_LORA.h](#).

3.39.3.8 ERROROTAAAPPEUI #define ERROROTAAAPPEUI 2

Definition at line 46 of file [hal_LORA.h](#).

3.39.3.9 ERROROTAAAPPKEY #define ERROROTAAAPPKEY 3

Definition at line 47 of file [hal_LORA.h](#).

3.39.3.10 ERROROTAA_BUSY #define ERROROTAA_BUSY 12

Definition at line 56 of file [hal_LORA.h](#).

3.39.3.11 ERROROTAA_DENIED #define ERROROTAA_DENIED 6

Definition at line 50 of file [hal_LORA.h](#).

3.39.3.12 ERROROTAADEVEUI #define ERROROTAADEVEUI 1

Definition at line 45 of file [hal_LORA.h](#).

3.39.3.13 ERROR_OTAA_INVALID_PARAM #define ERROR_OTAA_INVALID_PARAM 8

Definition at line 52 of file [hal_LORA.h](#).

3.39.3.14 ERROR_OTAA_JOIN #define ERROR_OTAA_JOIN 5

Definition at line 49 of file [hal_LORA.h](#).

3.39.3.15 ERROR_OTAA_KEYS_NOT_INIT #define ERROR_OTAA_KEYS_NOT_INIT 9

Definition at line 53 of file [hal_LORA.h](#).

3.39.3.16 ERROR_OTAA_MAC_PAUSED #define ERROR_OTAA_MAC_PAUSED 13

Definition at line 57 of file [hal_LORA.h](#).

3.39.3.17 ERROR_OTAA_NO_FREE_CH #define ERROR_OTAA_NO_FREE_CH 10

Definition at line 54 of file [hal_LORA.h](#).

3.39.3.18 ERROR_OTAA_SAVE #define ERROR_OTAA_SAVE 4

Definition at line 48 of file [hal_LORA.h](#).

3.39.3.19 ERROR_OTAA_SILENT #define ERROR_OTAA_SILENT 11

Definition at line 55 of file [hal_LORA.h](#).

3.39.3.20 ERROR_OTAA_UNKNOWN #define ERROR_OTAA_UNKNOWN 7

Definition at line 51 of file [hal_LORA.h](#).

3.39.3.21 ERROR_RXDELAY1_SET `#define ERROR_RXDELAY1_SET 1`

Definition at line 74 of file [hal_LORA.h](#).

3.39.3.22 ERROR_SET_DEVADDR `#define ERROR_SET_DEVADDR 1`

Definition at line 30 of file [hal_LORA.h](#).

3.39.3.23 ERROR_SET_DNCTR `#define ERROR_SET_DNCTR 1`

Definition at line 36 of file [hal_LORA.h](#).

3.39.3.24 ERROR_SET_NWSKEY `#define ERROR_SET_NWSKEY 1`

Definition at line 42 of file [hal_LORA.h](#).

3.39.3.25 ERROR_SET_PWRIDX `#define ERROR_SET_PWRIDX 1`

Definition at line 39 of file [hal_LORA.h](#).

3.39.3.26 ERROR_SET_UPCTR `#define ERROR_SET_UPCTR 1`

Definition at line 33 of file [hal_LORA.h](#).

3.39.3.27 ERROR_TX_BUSY `#define ERROR_TX_BUSY 2`

Definition at line 63 of file [hal_LORA.h](#).

3.39.3.28 ERROR_TX_INVALID_DATA_LEN `#define ERROR_TX_INVALID_DATA_LEN 6`

Definition at line 67 of file [hal_LORA.h](#).

3.39.3.29 ERROR_TX_INVALID_PARAM #define ERROR_TX_INVALID_PARAM 7

Definition at line 68 of file [hal_LORA.h](#).

3.39.3.30 ERROR_TX_MAC_ERR #define ERROR_TX_MAC_ERR 1

Definition at line 62 of file [hal_LORA.h](#).

3.39.3.31 ERROR_TX_MAC_PAUSED #define ERROR_TX_MAC_PAUSED 10

Definition at line 71 of file [hal_LORA.h](#).

3.39.3.32 ERROR_TX_NO_FREE_CH #define ERROR_TX_NO_FREE_CH 4

Definition at line 65 of file [hal_LORA.h](#).

3.39.3.33 ERROR_TX_NOT_JOINED #define ERROR_TX_NOT_JOINED 3

Definition at line 64 of file [hal_LORA.h](#).

3.39.3.34 ERROR_TX_REJOIN_NEEDED #define ERROR_TX_REJOIN_NEEDED 9

Definition at line 70 of file [hal_LORA.h](#).

3.39.3.35 ERROR_TX_SILENT #define ERROR_TX_SILENT 8

Definition at line 69 of file [hal_LORA.h](#).

3.39.3.36 ERROR_TX_UNKNOWN #define ERROR_TX_UNKNOWN 5

Definition at line 66 of file [hal_LORA.h](#).

3.39.3.37 SUCCESS_ABPLORA #define SUCCESS_ABPLORA 0

Definition at line 21 of file [hal_LORA.h](#).

3.39.3.38 SUCCESS_OTAA_LORA #define SUCCESS_OTAA_LORA 0

Definition at line 44 of file [hal_LORA.h](#).

3.39.3.39 SUCCESS_RXDELAY1_SET #define SUCCESS_RXDELAY1_SET 0

Definition at line 73 of file [hal_LORA.h](#).

3.39.3.40 SUCCESS_SET_DEVADDR #define SUCCESS_SET_DEVADDR 0

Definition at line 29 of file [hal_LORA.h](#).

3.39.3.41 SUCCESS_SET_DNCTR #define SUCCESS_SET_DNCTR 0

Definition at line 35 of file [hal_LORA.h](#).

3.39.3.42 SUCCESS_SET_NWSKEY #define SUCCESS_SET_NWSKEY 0

Definition at line 41 of file [hal_LORA.h](#).

3.39.3.43 SUCCESS_SET_PWRIDX #define SUCCESS_SET_PWRIDX 0

Definition at line 38 of file [hal_LORA.h](#).

3.39.3.44 SUCCESS_SET_UPCTR #define SUCCESS_SET_UPCTR 0

Definition at line 32 of file [hal_LORA.h](#).

3.39.3.45 SUCCESS_TX_MAC_RX #define SUCCESS_TX_MAC_RX 11

Definition at line 61 of file [hal_LORA.h](#).

3.39.3.46 SUCCESS_TX_MAC_TX #define SUCCESS_TX_MAC_TX 0

Definition at line 60 of file [hal_LORA.h](#).

3.39.4 Function Documentation

3.39.4.1 GetLoraRxData() void GetLoraRxData (

unsigned char * buf,	
uint8_t size,	
struct LoraNode * MyLoraNode)	

RN2483 Lora module Get Lora RX data Function.

This function

1. Gets data from Lora GW downlink

Parameters

in	<i>UART_Handle</i>	uart
	<i>struct LoraNode *MyLoraNode</i>	pointer to LoraNode struct

Returns

None

Definition at line 976 of file [hal_LORA.c](#).

```
00976
00977
00978     uint8_t sz;
00979     uint8_t i, ini;
00980     const char space = ' ';
00981     const char cr = '\r';
00982     char *bufcp;
00983
00984     i=0;
00985     while (buf[i]!=space) {
00986         bufcp[i]=buf[i];
00987         i++;
00988     }
00989     MyLoraNode->PortNoRx = atoi(bufcp);
00991
00992     i=i+1;
00993     ini = i;
00994     while (buf[i]!=cr) {
00995         *(MyLoraNode->DataRx+i-ini) = buf[i++];
00996     }
00997
00998     MyLoraNode->DataLenRx = (i-ini);
00999 }
```

```
3.39.4.2 GetLoraServerParams() uint8_t GetLoraServerParams (
    uint8_t * bytes,
    uint8_t blen,
    struct LoraNode * MyLoraNode )
```

RN2483 Lora module Get Lora server parameters Function.

This function

1. Gets Lora server parameters from Lora server

Parameters

in	<i>uint8_t</i>	*bytes pointer to bytes
in	<i>uint8_t</i>	blen
	<i>struct</i>	LoraNode *MyLoraNode pointer to LoraNode struct

Returns

0

Definition at line 1082 of file [hal_LORA.c](#).

```
01082
01083
01084     uint8_t mask = bytes[0];
01085     int inext = 1;
01086
01087     if ((mask&0x01)!=0) { // WakeUpInterval
01088         MyNode.WakeUpInterval = (bytes[inext]«8) | bytes[inext+1];
01089         inext = inext+2;
01090         writeWakeUp(MyNode.WakeUpInterval);
01091         st_readFileWakeUp();
01092     }
01093
01094     if ((mask&0x02)!=0) { // Mode
01095         MyNode.Mode = (bytes[inext] & 0x07);
01096         inext = inext+1;
01097         writeMode(MyNode.Mode);
01098         st_read FileMode();
01099     }
01100
01101     if ((mask&0x04)!=0) { // SSID
01102         MyNode.SSID[5] = (char)(bytes[inext]);
01103         MyNode.SSID[4] = (char)(bytes[inext+1]);
01104         MyNode.SSID[3] = (char)(bytes[inext+2]);
01105         MyNode.SSID[2] = (char)(bytes[inext+3]);
01106         MyNode.SSID[1] = (char)(bytes[inext+4]);
01107         MyNode.SSID[0] = (char)(bytes[inext+5]);
01108         inext = inext + 6;
01109         writeSSID(MyNode.SSID);
01110         st_readFileSSID(&(MyNode.SSID));
01111     }
01112
01113     if ((mask&0x08)!=0) { // NCycles
01114         MyNode.NCycles = bytes[inext];
01115         inext = inext + 1;
01116         writeNCycles(MyNode.NCycles);
01117         st_readFileNCycles();
01118     }
01119
01120     if ((mask&0x10)!=0) { // Upcnt
01121         MyLoraNode->Upctr = (bytes[inext]«16) | (bytes[inext+1]«8) | (bytes[inext+2]);
01122         inext = inext + 3;
01123         writeUpCtr(MyLoraNode->Upctr);
01124         st_readFileUpCtr();
01125     }
01126 }
```

```

01127     if ((mask&0x20)!=0) {
01128         MyADXL.SampleRate = (bytes[inext]«8) | bytes[inext+1];
01129         MyADXL.NSamples = (bytes[inext+2]«8) | bytes[inext+3];
01130         inext = inext + 4;
01131     }
01132
01133     if ((mask&0x40)!=0) {
01134         MyLDC.NSamples = (bytes[inext]«8) | bytes[inext+1];
01135         inext = inext + 2;
01136     }
01137
01138     return 0;
01139 }
```

3.39.4.3 hex2int() `uint8_t hex2int (`
 `unsigned char * hex,`
 `uint8_t hlen,`
 `uint8_t * bytes)`

RN2483 Lora module Hex to int Function.

This function

1. Converts hexadecimal data to integer value

Parameters

in	<i>unsigned</i>	char *hex pointer to hex
in	<i>uint8_t</i>	hlen
	<i>uint8_t</i>	*bytes pointer to bytes

Returns

int k Returns the size of bytes array

Definition at line 1046 of file [hal_LORA.c](#).

```

01046
01047
01048     uint32_t val = 0;
01049     uint8_t mybyte;
01050     int i;
01051     int k=0;
01052
01053     for (i=0;i<hlen;i++) {
01054         uint8_t mybyte = hex[i];
01055         current character then increment // get
01056         if (mybyte >= '0' && mybyte <= '9') mybyte = mybyte - '0';
01057         transform hex character to the 4bit equivalent number, using the ascii table indexes // 
01058         else if (mybyte >= 'a' && mybyte <= 'f') mybyte = mybyte - 'a' + 10;
01059         else if (mybyte >= 'A' && mybyte <= 'F') mybyte = mybyte - 'A' + 10;
01060
01061         val = (val << 4) | (mybyte & 0xF);
01062         to make space for new digit, and add the 4 bits of the new digit // shift 4
01063         if ((i&0x01)==1) {
01064             bytes[k++]=val;
01065             val = 0;
01066         }
01067     }
01068     return k;
01069     the size of bytes array // Returns
01070 }
```

```
3.39.4.4 Join_Abp_Lora() uint8_t Join_Abp_Lora (
    UART_Handle uart )
```

RN2483 Lora module Join ABP Lora connection Function.

This function

1. Function to join a lora connection using ABP (Authentification By Personalisation) in a RN2483.

Precondition

[Setup_Abp_Lora\(\)](#) has been called

Parameters

in	<i>UART_Handle</i>	uart
----	--------------------	------

Returns

None on Success / [ERROR_ABP_JOIN](#) on Fail according to [hal_LORA.h](#)

Definition at line 239 of file [hal_LORA.c](#).

```
00239
00240
00241     unsigned char Command[128];
00242     unsigned char buf[64];
00243     uint8_t sz;
00244
00245     strcpy(Command,"mac join abp\r\n");
00246     UART_write(uart, (const char *)Command, 14);
00247     sz = GetLine\_UART(uart, buf);
00248     if (strncmp(buf,"ok",2)==0) {
00249         sz = GetLine\_UART(uart, buf);
00250         if (strncmp(buf,"denied",6)==0) {
00251             return ERROR\_ABP\_DENIED;
00252         } else if (strncmp(buf,"accepted",8)==0) {
00253             return SUCCESS\_ABP\_LORA;
00254         } else {
00255             return ERROR\_ABP\_JOIN;
00256         }
00257     } else {
00258         return ERROR\_ABP\_JOIN;
00259     }
00260 }
```

```
3.39.4.5 Join_Otaa_Lora() uint8_t Join_Otaa_Lora (
    UART_Handle uart )
```

RN2483 Lora module join OTAA Lora connection Function.

This function

1. Function to join a lora connection using OTAA (Over the Air Activation) in a RN2483 based mote

Precondition

[Setup_Otaa_Lora\(\)](#) has been called

Parameters

in	<i>UART_Handle</i>	<i>uart</i>
----	--------------------	-------------

Returns

None on success / different ERROR Conditions according to [hal_LORA.h](#)

Definition at line 337 of file [hal_LORA.c](#).

```

00337
00338
00339     unsigned char Command[128];
00340     unsigned char buf[64];
00341     uint8_t sz;
00342
00343     memset(&buf, 0, sizeof(buf));
00344
00345     Mac_Adr_On(uart);
00346
00347     strcpy(Command, "mac join otaa\r\n");
00348     UART_write(uart, (const char *)Command, 15);
00349     sz = GetLine_UART(uart, buf);
00350     if (strncmp(buf, "ok", 2)==0) {
00351         sz = GetLine_UART(uart, buf);
00352         if (strncmp(buf, "denied", 6)==0) {
00353             return ERROR_OTAA_DENIED;
00354         } else if (strncmp(buf, "accepted", 8)==0) {
00355             return SUCCESS_OTAA_LORA;
00356         } else {
00357             return ERROR_OTAA_UNKNOWN;
00358         }
00359     } else if (strncmp(buf, "invalid_param", 13)==0) {
00360         return ERROR_OTAA_INVALID_PARAM;
00361     } else if (strncmp(buf, "keys_not_init", 13)==0) {
00362         return ERROR_OTAA_KEYS_NOT_INIT;
00363     } else if (strncmp(buf, "no_free_ch", 10)==0) {
00364         return ERROR_OTAA_NO_FREE_CH;
00365     } else if (strncmp(buf, "silent", 6)==0) {
00366         return ERROR_OTAA_SILENT;
00367     } else if (strncmp(buf, "busy", 4)==0) {
00368         return ERROR_OTAA_BUSY;
00369     } else if (strncmp(buf, "mac_paused", 10)==0) {
00370         return ERROR_OTAA_MAC_PAUSED;
00371     } else {
00372         return ERROR_OTAA_UNKNOWN;
00373     }
00374 }
```

3.39.4.6 Mac_Adr_On() `uint8_t Mac_Adr_On (`
`UART_Handle uart)`

RN2483 Lora module Mac ADR on Function.

This function

1. Sets adaptative data rate ON, on Lora RN2483 Module

Parameters

in	<i>UART_Handle</i>	<i>uart</i>
----	--------------------	-------------

Returns

0 on success / 1 on error

Definition at line 449 of file hal_LORA.c.

```
00449
00450
00451     unsigned char Command[256];
00452     unsigned char buf[32];
00453     uint8_t sz;
00454
00455     memset(&buf, sizeof(buf));
00456     strcpy(Command,"mac set adr on\r\n");
00457     UART_write(uart, (const char *)Command, 16);
00458     sz = GetLine_UART(uart, buf);
00459     if (strncmp(buf,"ok",2)==0) {
00460         return 0;
00461     } else {
00462         return 1;
00463     }
00464 }
```

3.39.4.7 Mac_Ar_On() `uint8_t Mac_Ar_On (`
`UART_Handle uart)`

RN2483 Lora module Mac AR on Function.

This function

1. Sets automatic reply ON, on Lora RN2483 Module

Parameters

in	<i>UART_Handle</i>	uart
----	--------------------	------

Returns

0 on success / 1 on error

Definition at line 478 of file hal_LORA.c.

```
00478
00479
00480     unsigned char Command[256];
00481     unsigned char buf[32];
00482     uint8_t sz;
00483
00484     memset(&buf, sizeof(buf));
00485     strcpy(Command,"mac set ar on\r\n");
00486     UART_write(uart, (const char *)Command, 15);
00487     sz = GetLine_UART(uart, buf);
00488     if (strncmp(buf,"ok",2)==0) {
00489         return 0;
00490     } else {
00491         return 1;
00492     }
00493 }
```

3.39.4.8 Mac_Clear_Upctr() `uint8_t Mac_Clear_Upctr (`
`UART_Handle uart)`

RN2483 Lora module Mac clear Up counter Function.

This function

1. Sets up counter parameter to 0 on Lora RN2483 Module

Parameters

in	<i>UART_Handle</i>	uart
----	--------------------	------

Returns

0 on success / 1 on fail

Definition at line 647 of file [hal_LORA.c](#).

```
00647
00648
00649     unsigned char Command[256];
00650     unsigned char buf[32];
00651     uint8_t sz;
00652
00653     strcpy(Command, "mac set upctr 0\r\n");
00654     UART_write(uart, (const char *) (Command), 17);
00655     sz = GetLine_UART(uart, buf);
00656     if (strncmp(buf, "ok", 2) == 0) {
00657         return 0;
00658     } else {
00659         return 1;
00660     }
00661 }
```

3.39.4.9 Mac_Get_Appeui() `uint8_t Mac_Get_Appeui (`
`UART_Handle uart,`
`struct LoraNode * MyLoraNode)`

RN2483 Lora module Mac get Appeui Function.

This function

1. Gets Appeui for Lora RN2483 Module

Parameters

in	<i>UART_Handle</i>	uart
	<i>struct LoraNode</i> *	MyLoraNode pointer to <i>LoraNode</i> struct

Returns

0

Definition at line 129 of file [hal_LORA.c](#).

```

00129
00130     unsigned char Command[128];
00131     unsigned char buf[64];
00132     uint8_t sz;
00133
00134     memset(&buf, 0, sizeof(buf));
00135     strcpy(Command, "mac get appeui\r\n");
00136     UART_write(uart, (const char *) (Command), 16);
00137     sz = GetLine_UART(uart, buf);
00138     strncpy(MyLoraNode->AppEui, buf, 16);
00139
00140     return 0;
00141 }
```

3.39.4.10 Mac_Get_Devaddr()

```
uint8_t Mac_Get_Devaddr (
    UART_HandleTypeDef uart,
    struct LoraNode * MyLoraNode )
```

RN2483 Lora module Mac get Devaddr Function.

This function

1. Gets Devaddr for Lora RN2483 Module

Parameters

in	<i>UART_HandleTypeDef</i>	uart
	<i>struct LoraNode *</i>	MyLoraNode pointer to <i>LoraNode</i> struct

Returns

0

Definition at line 156 of file [hal_LORA.c](#).

```

00156
00157     unsigned char Command[256];
00158     unsigned char buf[32];
00159     uint8_t sz;
00160
00161     strcpy(Command, "mac get devaddr\r\n");
00162     UART_write(uart, (const char *) (Command), 17);
00163     sz = GetLine_UART(uart, buf);
00164     strncpy(MyLoraNode->DevAddr, buf, 8);
00165
00166     return 0;
00167 }
```

3.39.4.11 Mac_Get_Deveui()

```
uint8_t Mac_Get_Deveui (
    UART_HandleTypeDef uart,
    struct LoraNode * MyLoraNode )
```

RN2483 Lora module Mac get Deveui Function.

This function

1. Gets Deveui for Lora RN2483 Module

Parameters

in	<i>UART_Handle</i>	uart
	<i>struct</i>	LoraNode *MyLoraNode pointer to LoraNode struct

Returns

0

Definition at line 102 of file [hal_LORA.c](#).

```
00102
00103     unsigned char Command[128];
00104     unsigned char buf[64];
00105     uint8_t sz;
00106
00107     memset(&buf, 0, sizeof(buf));
00108     strcpy(Command, "mac get deveui\r\n");
00109     UART_write(uart, (const char *) (Command), 16);
00110     sz = GetLine_UART(uart, buf);
00111     strncpy(MyLoraNode->DevEui, buf, 16);
00112
00113     return 0;
00114 }
```

3.39.4.12 Mac_Get_Dnctr() int Mac_Get_Dnctr (
 UART_Handle uart)

RN2483 Lora module Mac get Dnctr Function.

This function

1. Gets down counter parameter for Lora RN2483 Module

Parameters

in	<i>UART_Handle</i>	uart
----	--------------------	------

Returns

atoi(buf) value of the Dnctr parameter

Definition at line 592 of file [hal_LORA.c](#).

```
00592
00593
00594     unsigned char Command[256];
00595     unsigned char buf[32];
00596     uint8_t sz;
00597
00598     strcpy(Command, "mac get dnctr\r\n");
00599     UART_write(uart, (const char *) (Command), 15);
00600     sz = GetLine_UART(uart, buf);
00601
00602     return atoi(buf);
00603 }
```

3.39.4.13 Mac_Get_Upctr() `int Mac_Get_Upctr (`
`UART_Handle uart)`

RN2483 Lora module Mac get Upctr Function.

This function

1. Gets up counter parameter for Lora RN2483 Module

Parameters

in	<i>UART_Handle</i>	uart
----	--------------------	------

Returns

`atoi(buf)` value of the Upctr parameter

Definition at line 538 of file [hal_LORA.c](#).

```
00538
00539
00540     unsigned char Command[256];
00541     unsigned char buf[32];
00542     uint8_t sz;
00543
00544     strcpy(Command, "mac get upctr\r\n");
00545     UART_write(uart, (const char *) (Command), 15);
00546     sz = GetLine_UART(uart, buf);
00547
00548     return atoi(buf);
00549 }
```

3.39.4.14 Mac_Save() `uint8_t Mac_Save (`
`UART_Handle uart)`

RN2483 Lora module Mac save Function.

This function

1. Saves configuration parameters to Lora RN2483 Module

Parameters

in	<i>UART_Handle</i>	uart
----	--------------------	------

Returns

0 on success / 1 on error

Definition at line 417 of file [hal_LORA.c](#).

```
00417
00418
00419     unsigned char Command[256];
00420     unsigned char buf[32];
```

```

00421     uint8_t sz;
00422
00423     strcpy(Command,"mac save\r\n");
00424
00425     UART_write(uart, Command, 10);
00426     sz = GetLine_UART(uart, buf);
00427
00428     if (strncmp(buf,"ok",2)==0) {
00429         return 0;
00430     } else if (strncmp(buf,"invalid_param",13)==0) {
00431         return 1;
00432     } else {
00433         return 1;
00434     }
00435 }
```

3.39.4.15 Mac_Set_Devaddr() `uint8_t Mac_Set_Devaddr (`
 `UART_Handle uart,`
 `struct LoraNode * MyLoraNode)`

RN2483 Lora module Mac set Devaddr Function.

This function

1. Sets Devaddr on Lora RN2483 Module

Parameters

in	<code>UART_Handle</code>	uart
	<code>struct</code>	<code>LoraNode *</code> MyLoraNode pointer to <code>LoraNode</code> struct

Returns

`SUCCESS_SET_DEVADDR / ERROR_SET_DEVADDR` according to `hal_LORA.h`

Definition at line 508 of file `hal_LORA.c`.

```

00508
00509     unsigned char Command[256];
00510     unsigned char buf[32];
00511     uint8_t sz;
00512
00513     memset(&buf,0, sizeof(buf));
00514     strcpy(Command,"mac set devaddr ");
00515     strncat(Command, MyLoraNode->DevAddr,sizeof(MyLoraNode->DevAddr));
00516     strncat(Command,"\\r\\n");
00517     UART_write(uart, (const char *)Command, 26);
00518     sz = GetLine_UART(uart, buf);
00519     if (strncmp(buf,"ok",2)!=0) {
00520         return ERROR_SET_DEVADDR;
00521     } else {
00522         return SUCCESS_SET_DEVADDR;
00523     }
00524 }
```

3.39.4.16 Mac_Set_Dnctr() `uint8_t Mac_Set_Dnctr (`
 `UART_Handle uart,`
 `struct LoraNode * MyLoraNode)`

RN2483 Lora module Mac set Dnctr Function.

This function

1. Sets down counter parameter on Lora RN2483 Module

Parameters

in	<i>UART_Handle</i>	uart
	<i>struct</i>	LoraNode *MyLoraNode pointer to LoraNode struct

ReturnsERROR_SET_DNCTR / SUCCESS_SET_DNCTR according to [hal_LORA.h](#)**Definition at line 618 of file hal_LORA.c.**

```

00618
00619     unsigned char Command[256];
00620     unsigned char buf[32];
00621     uint8_t sz;
00622
00623     memset(&buf, 0, sizeof(buf));
00624     sprintf(Command, "mac set dnctr %d\r\n", MyLoraNode->Dnctr);
00625     UART_write(uart, (const char *)Command, strlen(Command));
00626     sz = GetLine_UART(uart, buf);
00627     if (strncmp(buf, "ok", 2) != 0) {
00628         return ERROR_SET_DNCTR;
00629     } else {
00630         return SUCCESS_SET_DNCTR;
00631     }
00632 }
00633 }
```

3.39.4.17 Mac_Set_Pwridx() `uint8_t Mac_Set_Pwridx (`

```
    UART_Handle uart,
    uint8_t pwridx )
```

RN2483 Lora module Mac set Pwridx Function.

This function

1. Sets the output power to be used on next transmissions on Lora RN2483 Module

Parameters

in	<i>UART_Handle</i>	uart
	<i>uint8_t</i>	pwridx decimal number representing the index output for the power value

ReturnsERROR_SET_PWRIDX / SUCCESS_SET_PWRIDX according to [hal_LORA.h](#)**Definition at line 676 of file hal_LORA.c.**

```

00676
00677     unsigned char Command[256];
00678     unsigned char buf[32];
00679     uint8_t sz;
00680
00681     memset(&buf, 0, sizeof(buf));
00682     sprintf(Command, "mac set pwridx %d\r\n", pwridx);
00683     UART_write(uart, (const char *)Command, strlen(Command));
00684     sz = GetLine_UART(uart, buf);
00685     if (strncmp(buf, "ok", 2) != 0) {
00686         return ERROR_SET_PWRIDX;
```

```

00687     } else {
00688         return SUCCESS_SET_PWRIDX;
00689     }
00690 }
00691 }
```

3.39.4.18 Mac_Set_Rxdelay1() uint8_t Mac_Set_Rxdelay1 (

```
UART_HandleTypeDef uart,
      uint16_t delay )
```

RN2483 Lora module Mac set Rx delay 1 Function.

This function

1. Sets RX delay 1 for Lora RN2483 Module

Parameters

in	<i>UART_HandleTypeDef</i>	uart
in	<i>uint16_t</i>	delay value of the delay

Returns

SUCCESS_RXDELAY1_SET / ERROR_RXDELAY1_SET according to [hal_LORA.h](#)

Definition at line 389 of file [hal_LORA.c](#).

```

00389
00390     unsigned char Command[128];
00391     unsigned char buf[64];
00392     uint8_t sz;
00393
00394     memset(&buf, sizeof(buf));
00395     sprintf(Command, "mac set rxdelay1 %d\r\n", delay);
00396     UART_write(uart, (const char *)Command, strlen(Command));
00397     sz = GetLine_UART(uart, buf);
00398     if (strncmp(buf, "ok", 2) != 0) {
00399         return ERROR_RXDELAY1_SET;
00400     } else {
00401         return SUCCESS_RXDELAY1_SET;
00402     }
00403 }
```

3.39.4.19 Mac_Set_Upcstr() uint8_t Mac_Set_Upcstr (

```
UART_HandleTypeDef uart,
      struct LoraNode * MyLoraNode )
```

RN2483 Lora module Mac set Upctr Function.

This function

1. Sets up counter parameter on Lora RN2483 Module

Parameters

in	<i>UART_Handle</i>	uart
	<i>struct</i>	LoraNode *MyLoraNode pointer to LoraNode struct

Returns

SUCCESS_SET_UPCTR / ERROR_SET_UPCTR according to [hal_LORA.h](#)

Definition at line 564 of file [hal_LORA.c](#).

```
00564
00565     unsigned char Command[256];
00566     unsigned char buf[32];
00567     uint8_t sz;
00568
00569     memset(&buf, 0, sizeof(buf));
00570     sprintf(Command, "mac set upctr %d\r\n", MyLoraNode->Upctr);
00571     UART_write(uart, (const char *)Command, strlen(Command));
00572     sz = GetLine_UART(uart, buf);
00573     if (strncmp(buf, "ok", 2) != 0) {
00574         return ERROR_SET_UPCTR;
00575     } else {
00576         return SUCCESS_SET_UPCTR;
00577     }
00578 }
```

3.39.4.20 Reset_RN2483() void Reset_RN2483 (

 void)

RN2483 Lora module reset Function.

This function

1. Resets RN2483 Lora module

Parameters

in		
----	--	--

Activate MCLR Pin

Release MCLR Pin

Definition at line 52 of file [hal_LORA.c](#).

```
00052
00053     {
00054     GPIO_write(Board_RN2483_MCLR, 0);
00055     usleep(50000);
00056     GPIO_write(Board_RN2483_MCLR, 1);
00057 }
```

3.39.4.21 Setup_Abp_Lora() uint8_t Setup_Abp_Lora (

UART_Handle *uart*,

struct LoraNode * *MyLoraNode*)

RN2483 Lora module Set up ABP Lora connection Function.

This function

Precondition

[Setup_Otaa_Lora\(\)](#) has been called

Parameters

in	<i>UART_Handle</i>	uart
	<i>struct</i>	LoraNode *MyLoraNode pointer to LoraNode struct

Returns

[SUCCESS_OTAA_LORA](#) Different SUCCESS/ERROR Conditions according to [hal_LORA.h](#)

Definition at line 277 of file [hal_LORA.c](#).

```

00277
00278
00279     unsigned char Command[128];
00280     unsigned char buf[64];
00281     uint8_t sz;
00282
00283     memset(&buf, sizeof(buf));
00284     strcpy(Command,"sys get hweui\r\n");
00285     UART_write(uart, (const char *)Command, 15);
00286     sz = GetLine\_UART(uart, buf);
00287     strncpy(MyLoraNode->DevEui,buf,16);
00288
00289     memset(&buf, sizeof(buf));
00290     memset(&Command, 0, sizeof(Command));
00291     strcpy(Command,"mac set deveui ");
00292     strncat(Command, MyLoraNode->DevEui,16);
00293     strcat(Command, "\r\n");
00294     UART_write(uart, (const char *)Command, 33);
00295     sz = GetLine\_UART(uart, buf);
00296     if (strncmp(buf,"ok",2)!=0) {
00297         return ERROR\_OTAA\_DEVEUI;
00298     }
00299
00300     memset(&buf, sizeof(buf));
00301     strcpy(Command,"mac set appeui ");
00302     strncat(Command, MyLoraNode->AppEui,16);
00303     strcat(Command, "\r\n");
00304     UART_write(uart, (const char *)Command, 33);
00305     sz = GetLine\_UART(uart, buf);
00306     if (strncmp(buf,"ok",2)!=0) {
00307         return ERROR\_OTAA\_APPEUI;
00308     }
00309
00310     memset(&buf, sizeof(buf));
00311     strcpy(Command,"mac set appkey ");
00312     strncat(Command, MyLoraNode->AppKey,32);
00313     strcat(Command, "\r\n");
00314     UART_write(uart, (const char *)Command, 49);
00315     sz = GetLine\_UART(uart, buf);
00316     if (strncmp(buf,"ok",2)!=0) {
00317         return ERROR\_OTAA\_APPKEY;
00318     }
00319
00320     return SUCCESS\_OTAA\_LORA;
00321 }
```

3.39.4.23 Sys_FactoryReset() `uint8_t Sys_FactoryReset (`
 `UART_Handle uart)`

RN2483 Lora module Sys Factory reset Function.

This function

1. Resets the RN2483 module to factory values

Parameters

in	<i>UART_Handle</i>	uart
----	--------------------	------

Returns

0

Definition at line 76 of file [hal_LORA.c](#).

```

00076
00077
00078     unsigned char Command[256];
00079     unsigned char buf[32];
00080     uint8_t sz;
00081
00082     strcpy(Command, "sys factoryRESET\r\n");
00083
00084     UART_write(uart, Command, strlen(Command));
00085     sz = GetLine_UART(uart, buf);
00086
00087     return 0;
00088 }
```

3.39.4.24 [Sys_Sleep\(\)](#) `uint8_t Sys_Sleep (`
 `UART_Handle uart,`
 `uint32_t sleep)`

RN2483 Lora module Sys sleep Function.

This function

1. Puts the RN2483 module in sleep mode during sleep ms. This function returns when the device comes out of sleep mode

Precondition

[Sys_Sleep\(\)](#) has been called

Parameters

in	<i>UART_Handle</i>	uart
in	<i>uint32_t</i>	sleep Value of the sleep time in ms

Returns

0 on success / 1 on fail

Definition at line 943 of file [hal_LORA.c](#).

```

00943
00944
00945     unsigned char Command[256];
00946     unsigned char buf[32];
00947     uint8_t sz;
00948
00949     sprintf(Command, "sys sleep %d\r\n", sleep);
```

```

00950
00951     UART_write(uart, Command, strlen(Command));
00952     sz = GetLine_UART(uart, buf);
00953
00954     if (strncmp(buf, "ok", 2)==0) {
00955         return 0;
00956     } else if (strncmp(buf, "invalid_param", 13)==0) {
00957         return 1;
00958     } else {
00959         return 1;
00960     }
00961 }
```

3.39.4.25 Try_Join_Lora_Gateway() uint8_t Try_Join_Lora_Gateway (

```
UART_Handle uart_dbg,
UART_Handle uart_lora )
```

RN2483 Lora module Try join Lora Gateway Function.

This function

1. Does several tries to connect to GW using OTAA.

Parameters

in	UART_Handle	uart_dbg UART debug function
in	UART_Handle	uart

Returns

uint8_t ret

Definition at line 706 of file hal_LORA.c.

```

00706
00707
00708     uint8_t nf;
00709     uint8_t wait;
00710     uint8_t ret;
00711     uint8_t sz;
00712     unsigned char Mess[32];
00713     unsigned char buf[256];
00714
00715
00716     nf=0;
00717     do {
00718         if (nf<3) {
00719             wait = 30;
00720         } else if (nf<6){
00721             wait = 60;
00722         } else {
00723             break;
00724         }
00725         ret = Join_Otaa_Lora(uart_lora);
00726         if (ret==SUCCESS_OTAA_LORA) {
00727             strcpy(Mess,"Join_Otaa_Lora() Success\r\n");
00728             UART_write(uart_dbg, Mess, 26);
00729         } else {
00730             strcpy(Mess,"Join_Otaa_Lora() Failed ");
00731             UART_write(uart_dbg, Mess, 24);
00732             Mess[0] = '('; Mess[1] = ret+48; Mess[2]= ')';
00733             UART_write(uart_dbg, Mess, 3);
00734             UART_write(uart_dbg, "\r\n", 2);
00735             if (ret==ERROR_OTAA_DENIED) {
00736                 break;
00737             }
00738         nf++;
```

```

00739         sleep(wait);
00740     }
00741 } while (ret!=SUCCESS_OTAA_LORA);
00742
00743 return ret;
00744 }
```

3.39.4.26 Tx_Cnf_Lora() uint8_t Tx_Cnf_Lora (

```

    UART_Handle uart,
    struct LoraNode * MyLoraNode,
    uint8_t * mask,
    uint16_t * nodeld )
```

RN2483 Lora module TX Cnf Function.

This function

1. Transmits data using lora connection in a RN2483 based mote in confirmed mode

Precondition

[Tx_Cnf_Lora\(\)](#) has been called

Parameters

in	<i>UART_Handle</i>	uart
in	<i>struct</i>	<i>LoraNode</i> *MyLoraNode pointer to <i>LoraNode</i> struct
	<i>uint8_t</i>	*mask pointer to mask
	<i>uint16_t</i>	*nodeld pointer to nodeld

Returns

uint8_t ret Different SUCCESS / ERROR codes according to [hal_LORA.h](#)

Definition at line 853 of file [hal_LORA.c](#).

```

00853
00854
00855     unsigned char Command[256];
00856     unsigned char buf[32];
00857     unsigned char payload[40];
00858     uint8_t bytes[20];
00859     uint8_t PortNo;
00860     uint8_t sz;
00861     uint8_t ret;
00862     uint8_t LastAnswer = FALSE;
00863
00864     memset(bytes,0,sizeof(bytes));
00865
00866     Mac_Adr_On(uart); // Set
00867     adaptive datarate ON
00868     Mac_Ar_On(uart); // Set
00869     Automatic Retransmit ON
00870
00871     sprintf(Command,"mac tx cnf %d ", MyLoraNode->PortNoTx);
00872     strncat(Command,MyLoraNode->DataTx, MyLoraNode->DataLenTx);
00873     strncat(Command,"\r\n");
00874
00875     UART_write(uart, Command, strlen(Command));
00876
00877     *mask = 0;
```

```

00876     *nodeId = 0;
00877     while (LastAnswer==FALSE) {
00878         sz = GetLine_UART(uart, buf);
00879         if (strncpy(buf, "ok", 2)==0) {
00880             ;
00881         } else if (strncpy(buf, "mac_tx_ok", 9)==0) {
00882             LastAnswer=TRUE;
00883             ret = SUCCESS_TX_MAC_TX;
00884         } else if (strncpy(buf, "mac_rx ", 7)==0) {
00885             sscanf(buf, "mac_rx %d %s\r\n", &PortNo, payload);
00886             sz = hex2int(payload, strlen(payload), bytes);
00887             GetLoraServerParams(bytes, sz, MyLoraNode);
00888             *mask = bytes[0];
00889             *nodeId = (bytes[1]«8) | bytes[2];
00890             ret = SUCCESS_TX_MAC_RX;
00891         } else if (strncpy(buf, "mac_err", 7)==0) {
00892             LastAnswer=TRUE;
00893             ret = ERROR_TX_MAC_ERR;
00894         } else if (strncpy(buf, "invalid_data_len", 16)==0) {
00895             LastAnswer=TRUE;
00896             ret = ERROR_TX_INVALID_DATA_LEN;
00897         } else if (strncpy(buf, "invalid_param", 13)==0) {
00898             LastAnswer=TRUE;
00899             ret = ERROR_TX_INVALID_PARAM;
00900         } else if (strncpy(buf, "not_joined", 10)==0) {
00901             LastAnswer=TRUE;
00902             ret = ERROR_TX_NOT_JOINED;
00903         } else if (strncpy(buf, "no_free_ch", 10)==0) {
00904             LastAnswer=TRUE;
00905             ret = ERROR_TX_NO_FREE_CH;
00906         } else if (strncpy(buf, "silent", 6)==0) {
00907             LastAnswer=TRUE;
00908             ret = ERROR_TX_SILENT;
00909         } else if (strncpy(buf, "frame_counter_err_rejoin_needed", 31)==0) {
00910             LastAnswer=TRUE;
00911             ret = ERROR_TX_REJOIN_NEEDED;
00912         } else if (strncpy(buf, "busy", 4)==0) {
00913             LastAnswer=TRUE;
00914             ret = ERROR_TX_BUSY;
00915         } else if (strncpy(buf, "mac_paused", 10)==0) {
00916             LastAnswer=TRUE;
00917             ret = ERROR_TX_MAC_PAUSED;
00918         } else if (strncpy(buf, "invalid_data_len", 16)==0) {
00919             LastAnswer=TRUE;
00920             ret = ERROR_TX_INVALID_DATA_LEN;
00921         } else {
00922             LastAnswer=TRUE;
00923             ret = ERROR_TX_UNKNOWN;
00924         }
00925     }
00926     return ret;
00927 }
```

3.39.4.27 Tx_Uncnf_Lora() uint8_t Tx_Uncnf_Lora (

- UART_Handle uart,
- struct LoraNode * MyLoraNode,
- uint8_t * mask,
- uint16_t * nodeld)

RN2483 Lora module TX Uncnf Function.

This function

1. Transmits data using lora connection in a RN2483 based mote in unconfirmed mode

Parameters

in	UART_Handle	uart
in	struct	LoraNode *MyLoraNode pointer to LoraNode struct
	uint8_t	*mask pointer to mask
	uint16_t	*nodeld pointer to nodeld

Returns

uint8_t ret Different SUCCESS / ERROR codes according to [hal_LORA.h](#)

Definition at line 761 of file [hal_LORA.c](#).

```

00761 {
00762     unsigned char Command[256];
00763     unsigned char buf[40];
00764     unsigned char payload[40];
00765     uint8_t bytes[20];
00766
00767     uint8_t PortNo;
00768     uint8_t sz;
00769     uint8_t ret;
00770     uint8_t LastAnswer = FALSE;
00771
00772     memset(bytes,0, sizeof(bytes));
00773
00774     sprintf(Command,"mac tx uncnf %d ", MyLoraNode->PortNoTx);
00775     strncat(Command,MyLoraNode->DataTx, MyLoraNode->DataLenTx);
00776     strncat(Command,"\\r\\n");
00777
00778     UART_write(uart, Command, strlen(Command));
00779
00780     *mask = 0;
00781     while (LastAnswer==FALSE) {
00782         sz = GetLine_UART(uart, buf);
00783         if (strncmp(buf,"ok",2)==0) {
00784             ;
00785         } else if (strncmp(buf,"mac_tx_ok",9)==0) {
00786             UART_PRINT("%s\\n",buf);
00787             LastAnswer=TRUE;
00788             ret = SUCCESS_TX_MAC_TX;
00789         } else if (strncmp(buf,"mac_rx ",7)==0) {
00790             sscanf(buf,"%d %s\\r\\n",&PortNo,payload);
00791             UART_PRINT("%s\\n",buf);
00792             sz = hex2int(payload, strlen(payload), bytes);
00793             GetLoraServerParams(bytes, sz, MyLoraNode);
00794             *mask = bytes[0];
00795             *nodeId = (bytes[1]«8) | bytes[2];
00796             LastAnswer=TRUE;
00797             ret = SUCCESS_TX_MAC_RX;
00798         } else if (strncmp(buf,"mac_err",7)==0) {
00799             LastAnswer=TRUE;
00800             ret = ERROR_TX_MAC_ERR;
00801         } else if (strncmp(buf,"invalid_data_len",16)==0) {
00802             LastAnswer=TRUE;
00803             ret = ERROR_TX_INVALID_DATA_LEN;
00804         } else if (strncmp(buf,"invalid_param",13)==0) {
00805             LastAnswer=TRUE;
00806             ret = ERROR_TX_INVALID_PARAM;
00807         } else if (strncmp(buf,"not_joined",10)==0) {
00808             LastAnswer=TRUE;
00809             ret = ERROR_TX_NOT_JOINED;
00810         } else if (strncmp(buf,"no_free_ch",10)==0) {
00811             LastAnswer=TRUE;
00812             ret = ERROR_TX_NO_FREE_CH;
00813         } else if (strncmp(buf,"silent",6)==0) {
00814             LastAnswer=TRUE;
00815             ret = ERROR_TX_SILENT;
00816         } else if (strncmp(buf,"frame_counter_err_rejoin_needed",31)==0) {
00817             LastAnswer=TRUE;
00818             ret = ERROR_TX_REJOIN_NEEDED;
00819         } else if (strncmp(buf,"busy",4)==0) {
00820             LastAnswer=TRUE;
00821             ret = ERROR_TX_BUSY;
00822         } else if (strncmp(buf,"mac_paused",10)==0) {
00823             LastAnswer=TRUE;
00824             ret = ERROR_TX_MAC_PAUSED;
00825         } else if (strncmp(buf,"invalid_data_len",16)==0) {
00826             LastAnswer=TRUE;
00827             ret = ERROR_TX_INVALID_DATA_LEN;
00828         } else {
00829             LastAnswer=TRUE;
00830             ret = ERROR_TX_UNKNOWN;
00831         }
00832     }
00833     return ret;
00834 }
```

3.39.4.28 Uint8Array2Char() `uint8_t Uint8Array2Char (`
`uint8_t * DataPacket,`
`uint8_t DataPacketLen,`
`unsigned char * HexStr)`

RN2483 Lora module Uint8 array to char Function.

This function

1. Converts DataPacket (data from sensors) to hexadecimal value

Parameters

in	<i>uint8_t</i>	*DataPacket pointer to DataPacket struct
in	<i>uint8_t</i>	DataPacketLen
	<i>unsigned</i>	char *HexStr pointer to HexStr

Returns

`uint8_t k+1`

Definition at line 1015 of file [hal_LORA.c](#).

```
01015
01016
01017     uint8_t i, j, k;
01018     uint16_t a;
01019
01020     for (i=0;i<DataPacketLen;i++) {
01021         for (j=0;j<2;j++) {
01022             a = (DataPacket[i] >> (4-(j<<2))) & 0x0f;
01023             k = (i<1)+j;
01024             *(HexStr+k) = (a<10)? a+48 : a+55;
01025         }
01026     }
01027     HexStr[k+1] = '\0';
01028
01029     return k+1;
01030 }
```

3.40 hal_LORA.h

```
00001
00010 #ifndef HAL_LORA_H_
00011 #define HAL_LORA_H_
00012
00013 //*****
00014 //           INCLUDES
00015 //*****
00016 #include <ti/drivers/UART.h>
00017
00018 //*****
00019 //           DEFINES
00020 //*****
00021 #define SUCCESS_ABP_LORA          0
00022 #define ERROR_ABP_DEVADDR         1
00023 #define ERROR_ABP_NWKSKEY         2
00024 #define ERROR_ABP_APPSKY          3
00025 #define ERROR_ABP_SAVE            4
00026 #define ERROR_ABP_JOIN             5
00027 #define ERROR_ABP_DENIED          6
00028
00029 #define SUCCESS_SET_DEVADDR        0
00030 #define ERROR_SET_DEVADDR         1
00031
00032 #define SUCCESS_SET_UPCTR          0
00033 #define ERROR_SET_UPCTR           1
00034
00035 #define SUCCESS_SET_DNCTR          0
```

```

00036 #define ERROR_SET_DNCTR 1
00037
00038 #define SUCCESS_SET_PWRIDX 0
00039 #define ERROR_SET_PWRIDX 1
00040
00041 #define SUCCESS_SET_NWSKEY 0
00042 #define ERROR_SET_NWSKEY 1
00043
00044 #define SUCCESS_OTAA_LORA 0
00045 #define ERROR_OTAA_DEVEUI 1
00046 #define ERROR_OTAA_APPEUI 2
00047 #define ERROR_OTAA_APPKEY 3
00048 #define ERROR_OTAA_SAVE 4
00049 #define ERROR_OTAA_JOIN 5
00050 #define ERROR_OTAA_DENIED 6
00051 #define ERROR_OTAA_UNKNOWN 7
00052 #define ERROR_OTAA_INVALID_PARAM 8
00053 #define ERROR_OTAA_KEYS_NOT_INIT 9
00054 #define ERROR_OTAA_NO_FREE_CH 10
00055 #define ERROR_OTAA_SILENT 11
00056 #define ERROR_OTAA_BUSY 12
00057 #define ERROR_OTAA_MAC_PAUSED 13
00058 #define ERROR_ADR_SET 14
00059
00060 #define SUCCESS_TX_MAC_TX 0
00061 #define SUCCESS_TX_MAC_RX 11
00062 #define ERROR_TX_MAC_ERR 1
00063 #define ERROR_TX_BUSY 2
00064 #define ERROR_TX_NOT_JOINED 3
00065 #define ERROR_TX_NO_FREE_CH 4
00066 #define ERROR_TX_UNKNOWN 5
00067 #define ERROR_TX_INVALID_DATA_LEN 6
00068 #define ERROR_TX_INVALID_PARAM 7
00069 #define ERROR_TX_SILENT 8
00070 #define ERROR_TX_REJOIN_NEEDED 9
00071 #define ERROR_TX_MAC_PAUSED 10
00072
00073 #define SUCCESS_RXDELAY1_SET 0
00074 #define ERROR_RXDELAY1_SET 1
00075
00076 //*****
00077 // TYPEDEFS
00078 //*****
00079
00083 struct LoraNode {
00084     unsigned char DevEui[16];
00085     unsigned char DevAddr[8];
00086     unsigned char AppEui[16];
00087     unsigned char AppKey[32];
00088     unsigned char NwksKey[32];
00089     unsigned char AppsKey[32];
00090     unsigned char DataTx[128];
00091     uint8_t DataLenTx;
00092     uint8_t PortNoTx;
00093     unsigned char DataRx[4];
00094     uint8_t DataLenRx;
00095     uint8_t PortNoRx;
00096     uint32_t Upctr;
00097     uint32_t Dnctr;
00098 };
00099
00100 //*****
00101 // FUNCTION PROTOTYPES
00102 //*****
00103 void Reset_RN2483(void);
00104 uint8_t Mac_Get_Deveui(UART_HandleTypeDef uart, struct LoraNode *MyLoraNode);
00105 uint8_t Mac_Get_Appeui(UART_HandleTypeDef uart, struct LoraNode *MyLoraNode);
00106 uint8_t Setup_Abp_Lora(UART_HandleTypeDef uart, struct LoraNode *MyLoraNode);
00107 uint8_t Join_Abp_Lora(UART_HandleTypeDef uart);
00108 uint8_t Setup_Otaa_Lora(UART_HandleTypeDef uart, struct LoraNode *MyLoraNode);
00109 uint8_t Join_Otaa_Lora(UART_HandleTypeDef uart);
00110 uint8_t Mac_Set_Rxdelay1(UART_HandleTypeDef uart, uint16_t delay);
00111 uint8_t Mac_Save(UART_HandleTypeDef uart);
00112 uint8_t Mac_AdR_On(UART_HandleTypeDef uart);
00113 uint8_t Mac_Ar_On(UART_HandleTypeDef uart);
00114 int Mac_Get_Upctr(UART_HandleTypeDef uart);
00115 uint8_t Mac_Set_Upctr(UART_HandleTypeDef uart, struct LoraNode *MyLoraNode);
00116 int Mac_Get_Dnctr(UART_HandleTypeDef uart);
00117 uint8_t Mac_Set_Dnctr(UART_HandleTypeDef uart, struct LoraNode *MyLoraNode);
00118 uint8_t Mac_Set_Pwridx(UART_HandleTypeDef uart, uint8_t pwridx);
00119 uint8_t Mac_Get_Devaddr(UART_HandleTypeDef uart, struct LoraNode *MyLoraNode);
00120 uint8_t Mac_Set_Devaddr(UART_HandleTypeDef uart, struct LoraNode *MyLoraNode);
00121 uint8_t Mac_Clear_Upctr(UART_HandleTypeDef uart);
00122 uint8_t Try_Join_Lora_Gateway(UART_HandleTypeDef uart_dbg, UART_HandleTypeDef uart_lora);
00123 uint8_t Tx_Uncnf_Lora(UART_HandleTypeDef uart, struct LoraNode *MyLoraNode, uint8_t *mask, uint16_t *nodeId);
00124 uint8_t Tx_Cnf_Lora(UART_HandleTypeDef uart, struct LoraNode *MyLoraNode, uint8_t *mask, uint16_t *nodeId);
00125 uint8_t Sys_Sleep(UART_HandleTypeDef uart, uint32_t sleep);

```

```
00126 uint8_t Sys_FactoryReset(UART_Handle uart);
00127 void GetLoraRxData(unsigned char *buf, uint8_t size, struct LoraNode *MyLoraNode);
00128 uint8_t Uint8Array2Char(uint8_t *DataPacket, uint8_t DataPacketLen, unsigned char *HexStr);
00129 uint8_t hex2int(unsigned char *hex, uint8_t hlen, uint8_t *bytes);
00130 uint8_t GetLoraServerParams(uint8_t *bytes, uint8_t blen, struct LoraNode *MyLoraNode);
00131
00132 #endif /* HAL_LORA_H */
```

3.41 hal_PWM.c File Reference

Functions to manage PWM (Pulse Width Modulated signals) on CC3220SF.

```
#include <stdbool.h>
#include <ti/drivers/PWM.h>
```

Functions

- PWM_Handle Config_PWM (uint_least8_t index)
PWM Configuration Function.

3.41.1 Detailed Description

Functions to manage PWM (Pulse Width Modulated signals) on CC3220SF.

Version

1.0

Date

07/05/2019

Author

A.Irizar @tittle PWM (Pulse Width Modulated signals) functions

Definition in file [hal_PWM.c](#).

3.41.2 Function Documentation

3.41.2.1 Config_PWM() PWM_Handle Config_PWM (

 uint_least8_t index)

PWM Configuration Function.

This function

1. Configures PWM on CC3220SF

Parameters

in	<i>uint_</i> ↔ <i>least8_t</i>	index
----	-----------------------------------	-------

Returns

PWM_Handle pwm

Definition at line 32 of file [hal_PWM.c](#).

```

00032
00033
00034     PWM_Handle pwm;
00035     PWM_Params params;
00036
00037     PWM_init();                                     // Call driver init functions
00038
00039     PWM_Params_init(&params);
00040     params.dutyUnits = PWM_DUTY_FRACTION;
00041     params.dutyValue = (PWM_DUTY_FRACTION_MAX>>1);
00042     params.periodUnits = PWM_PERIOD_HZ;
00043     params.periodValue = 8000000;
00044     pwm = PWM_open(index, &params);
00045
00046     return pwm;
00047 }
```

3.42 hal_PWM.c

```

00001 //*****
00010 //***** INCLUDES *****
00011 //***** FUNCTIONS *****
00012 //*****
00013 #include <stddef.h>
00014 #include <ti/drivers/PWM.h>
00015
00016 //*****
00017 //***** FUNCTIONS *****
00018 //*****
00019
00020 //*****
00021 //
00030 //
00031 //*****
00032 PWM_Handle Config_PWM(uint_least8_t index) {
00033
00034     PWM_Handle pwm;
00035     PWM_Params params;
00036
00037     PWM_init();                                     // Call driver init functions
00038
00039     PWM_Params_init(&params);
00040     params.dutyUnits = PWM_DUTY_FRACTION;
00041     params.dutyValue = (PWM_DUTY_FRACTION_MAX>>1);
00042     params.periodUnits = PWM_PERIOD_HZ;
00043     params.periodValue = 8000000;
00044     pwm = PWM_open(index, &params);
00045
00046     return pwm;
00047 }
```

3.43 hal_PWM.h File Reference

Functions to manage PWM (Pulse Width Modulated signals) on CC3220SF.

#include <ti/drivers/PWM.h>

Functions

- PWM_Handle [Config_PWM](#) (uint_least8_t index)
PWM Configuration Function.

3.43.1 Detailed Description

Functions to manage PWM (Pulse Width Modulated signals) on CC3220SF.

Version

1.0

Date

07/05/2019

Author

A.Irizar @tittle PWM (Pulse Width Modulated signals) functions

Definition in file [hal_PWM.h](#).

3.43.2 Function Documentation

3.43.2.1 Config_PWM()

```
PWM_Handle Config_PWM (
    uint_least8_t index )
```

PWM Configuration Function.

This function

1. Configures PWM on CC3220SF

Parameters

in	<i>uint_</i> ← <i>least8_t</i>	index
----	-----------------------------------	-------

Returns

PWM_Handle pwm

Definition at line 32 of file [hal_PWM.c](#).

00032

{

```

00033
00034     PWM_Handle pwm;
00035     PWM_Params params;
00036
00037     PWM_init();                                     // Call driver init functions
00038
00039     PWM_Params_init(&params);
00040     params.dutyUnits = PWM_DUTY_FRACTION;
00041     params.dutyValue = (PWM_DUTY_FRACTION_MAX>>1);
00042     params.periodUnits = PWM_PERIOD_HZ;
00043     params.periodValue = 8000000;
00044     pwm = PWM_Open(index, &params);
00045
00046     return pwm;
00047 }

```

3.44 hal_PWM.h

```

00001
00010 #ifndef HAL_PWM_H_
00011 #define HAL_PWM_H_
00012
00013 //***** INCLUDES *****
00014 //          INCLUDES
00015 //***** FUNCTION PROTOTYPES *****
00016 #include <ti/drivers/PWM.h>
00017
00018 //***** FUNCTION PROTOTYPES *****
00019 //          FUNCTION PROTOTYPES
00020 //***** FUNCTION PROTOTYPES *****
00021 PWM_Handle Config_PWM(uint_least8_t index);
00022
00023 #endif /* HAL_PWM_H_ */

```

3.45 hal_SPI.c File Reference

Functions to manage SPI on CC3220SF.

```
#include <stdbool.h>
#include <stdint.h>
#include <ti/drivers/SPI.h>
```

Functions

- uint8_t [SPI_read_16bits](#) (SPI_Handle [spi](#), uint8_t Address, uint8_t *RxBuffer)
SPI Read 16 bits Function.
- uint8_t [SPI_read_8bits](#) (SPI_Handle [spi](#), uint8_t Address, uint8_t *RxBuffer, uint8_t nregs, uint8_t rnwMSB)
SPI Read 8 bits Function.
- uint8_t [SPI_write_16bits](#) (SPI_Handle [spi](#), uint8_t Address, uint8_t Data)
SPI Write 16 bits Function.
- uint8_t [SPI_write_8bits](#) (SPI_Handle [spi](#), uint8_t Address, uint8_t Data, uint8_t rnwMSB)
SPI Write 8 bits Function.
- SPI_Handle [Startup_SPI](#) (uint_least8_t index, uint32_t dataSize, uint32_t dataRate)
SPI Start up Function.

3.45.1 Detailed Description

Functions to manage SPI on CC3220SF.

Version

1.0

Date

07/05/2019

Author

A.Irizar @title SPI CC3220SF functions

Definition in file [hal_SPI.c](#).

3.45.2 Function Documentation

```
3.45.2.1 SPI_read_16bits() uint8_t SPI_read_16bits (
    SPI_Handle spi,
    uint8_t Address,
    uint8_t * RxBuffer )
```

SPI Read 16 bits Function.

This function

1. Reads 16 bits from SPI

Parameters

in	<i>SPI_Handle</i>	spi
in	<i>uint8_t</i>	Address
in	<i>uint8_t</i>	*RxBuffer pointer to RxBuffer

Returns

bool transferOK

Definition at line 106 of file [hal_SPI.c](#).

```
00106
00107
00108     SPI_Transaction spiTransaction;
00109     uint16_t          TxBuffer[2];
00110
00111     bool            transferOK;
```

{

```

00112     TxBuffer[0] = ((Address & 0x7F) + 0x80)«8;
00113
00114     spiTransaction.count = 1;
00115     spiTransaction.txBuf = (void *) TxBuffer;
00116     spiTransaction.rxBuf = (void *) RxBuffer;
00117     transferOK = SPI_transfer(spi, &spiTransaction);
00118
00119     return (transferOK? 0 : 1);
00120 }
00121 }
```

3.45.2.2 SPI_read_8bits() uint8_t SPI_read_8bits (

```

    SPI_Handle spi,
    uint8_t Address,
    uint8_t * RxBuffer,
    uint8_t nregs,
    uint8_t rnwMSB )
```

SPI Read 8 bits Function.

This function

1. Reads 8 bits from SPI

Parameters

in	<i>SPI_Handle</i>	spi
in	<i>uint8_t</i>	Address
in	<i>uint8_t</i>	*RxBuffer pointer to RxBuffer
in	<i>uint8_t</i>	nregs Number of registers to read
in	<i>uint8_t</i>	rnwMSB

Returns

```
bool transferOK
```

Definition at line 67 of file [hal_SPI.c](#).

```

00067
00068     {
00069     SPI_Transaction spiTransaction;
00070     uint8_t TxBuffer[32];
00071     RxBuffer[0] = 0;
00072     RxBuffer[1] = 0;
00073
00074     bool transferOK;
00075
00076     if (rnwMSB==0) {
00077         TxBuffer[0] = ((Address«1) | 0x01);
00078     } else {
00079         TxBuffer[0] = (Address & 0x7F) | 0x80;
00080     }
00081
00082     TxBuffer[1] = 0x00;
00083
00084     spiTransaction.count = nregs+1;
00085     spiTransaction.txBuf = (void *)TxBuffer;
00086     spiTransaction.rxBuf = (void *)RxBuffer;
00087     transferOK = SPI_transfer(spi, &spiTransaction);
00088
00089     return (transferOK? 0 : 1);
00090 }
```

3.45.2.3 SPI_write_16bits() `uint8_t SPI_write_16bits (`
`SPI_Handle spi,`
`uint8_t Address,`
`uint8_t Data)`

SPI Write 16 bits Function.

This function

1. Writes 16 bits to SPI

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
in	<i>uint8_t</i>	<i>Address</i>
in	<i>uint8_t</i>	<i>Data Data to write</i>

Returns

`bool transferOK`

Definition at line 175 of file [hal_SPI.c](#).

```
00175
00176
00177     SPI_Transaction spiTransaction;
00178     uint16_t      TxBuffer[2];
00179     uint16_t      RxBuffer[2];
00180     bool          transferOK;
00181
00182     TxBuffer[0] = (Address << 8) + Data;
00183
00184     spiTransaction.count = 1;
00185     spiTransaction.txBuf = (void *) (TxBuffer);
00186     spiTransaction.rxBuf = (void *) (RxBuffer);
00187     transferOK = SPI_transfer(spi, &spiTransaction);
00188
00189     return (transferOK? 0 : 1);
00190 }
```

3.45.2.4 SPI_write_8bits() `uint8_t SPI_write_8bits (`
`SPI_Handle spi,`
`uint8_t Address,`
`uint8_t Data,`
`uint8_t rnwMSB)`

SPI Write 8 bits Function.

This function

1. Writes 8 bits to SPI

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
in	<i>uint8_t</i>	<i>Address</i>
in	<i>uint8_t</i>	

Returns

```
bool transferOK
```

Definition at line 138 of file [hal_SPI.c](#).

```
00138
00139
00140     SPI_Transaction spiTransaction;
00141     uint8_t          TxBuffer[2];
00142     uint8_t          RxBuffer[2];
00143     bool             transferOK;
00144
00145     if (rnwMSB==0) {
00146         TxBuffer[0] = (Address<<1);
00147     } else {
00148         TxBuffer[0] = (Address & 0x7F);
00149     }
00150
00151     TxBuffer[1] = Data;
00152
00153     spiTransaction.count = 2;
00154     spiTransaction.txBuf = (void *) (TxBuffer);
00155     spiTransaction.rxBuf = (void *) (RxBuffer);
00156     transferOK = SPI_transfer(spi, &spiTransaction);
00157
00158     return (transferOK? 0 : 1);
00159 }
```

3.45.2.5 Startup_SPI() SPI_Handle Startup_SPI (

uint_least8_t	<i>index</i> ,
uint32_t	<i>dataSize</i> ,
uint32_t	<i>dataRate</i>)

SPI Start up Function.

This function

1. Initiates SPI on CC3220SF

Parameters

in	<i>uint_</i> ↔ <i>least8_t</i>	<i>index</i>
in	<i>uint32_t</i>	<i>dataSize</i>
in	<i>uint32_t</i>	<i>dataRate</i>

Returns

```
SPI_Handle spi
```

Definition at line 35 of file [hal_SPI.c](#).

```
00035
00036
00037     SPI_Params spiParams;
00038     SPI_Handle spi;
00039
00040     SPI_Params_init(&spiParams);                                // Assumes SPI_init() has
00041     spiParams.mode = SPI_MASTER;
00042     spiParams.transferMode = SPI_MODE_BLOCKING;
00043     spiParams.frameFormat = SPI_POL0_PHA0;
00044     spiParams.dataSize = dataSize;
00045     spiParams.bitRate = dataRate;
```

```

00046     spi = SPI_open(index, &spiParams);
00047
00048     return spi;
00049 }

```

3.46 hal_SPI.c

```

00001
00010 //*****
00011 //           INCLUDES
00012 //*****
00013 #include <stddef.h>
00014 #include <stdint.h>
00015 #include <ti/drivers/SPI.h>
00016
00017 //*****
00018 //           FUNCTIONS
00019 //*****
00020
00021 //*****
00022 //
00033 //
00034 //*****
00035 SPI_Handle Startup_SPI(uint_least8_t index, uint32_t dataSize, uint32_t dataRate) {
00036
00037     SPI_Params spiParams;
00038     SPI_Handle spi;
00039
00040     SPI_Params_init(&spiParams);                                     // Assumes SPI_init() has
already been started
00041     spiParams.mode = SPI_MASTER;
00042     spiParams.transferMode = SPI_MODE_BLOCKING;
00043     spiParams.frameFormat = SPI_POL0_PHA0;
00044     spiParams.dataSize = dataSize;
00045     spiParams.bitRate = dataRate;
00046     spi = SPI_open(index, &spiParams);
00047
00048     return spi;
00049 }
00050
00051 //*****
00052 //
00065 //
00066 //*****
00067 uint8_t SPI_read_8bits(SPI_Handle spi, uint8_t Address, uint8_t *RxBuffer, uint8_t nregs, uint8_t rnwMSB) {
00068
00069     SPI_Transaction spiTransaction;
00070     uint8_t TxBuffer[32];
00071     RxBuffer[0] = 0;
00072     RxBuffer[1] = 0;
00073
00074     bool transferOK;
00075
00076     if (rnwMSB==0) {
00077         TxBuffer[0] = ((Address<<1) | 0x01);
00078     } else {
00079         TxBuffer[0] = (Address & 0x7F) | 0x80;
00080     }
00081
00082     TxBuffer[1] = 0x00;
00083
00084     spiTransaction.count = nregs+1;
00085     spiTransaction.txBuf = (void *)TxBuffer;
00086     spiTransaction.rxBuf = (void *)RxBuffer;
00087     transferOK = SPI_transfer(spi, &spiTransaction);
00088
00089     return (transferOK? 0 : 1);
00090 }
00091
00092 //*****
00093 //
00104 //
00105 //*****
00106 uint8_t SPI_read_16bits(SPI_Handle spi, uint8_t Address, uint8_t *RxBuffer) {
00107
00108     SPI_Transaction spiTransaction;
00109     uint16_t TxBuffer[2];
00110
00111     bool transferOK;
00112
00113     TxBuffer[0] = ((Address & 0x7F) + 0x80)<<8;
00114     spiTransaction.count = 1;

```

```

00116     spiTransaction.txBuf = (void *) TxBuffer;
00117     spiTransaction.rxBuf = (void *) RxBuffer;
00118     transferOK = SPI_transfer(spi, &spiTransaction);
00119
00120     return (transferOK? 0 : 1);
00121 }
00122
00123 //*****
00124 //
00136 //
00137 //*****
00138 uint8_t SPI_write_8bits(SPI_Handle spi, uint8_t Address, uint8_t Data, uint8_t rnwMSB) {
00139
00140     SPI_Transaction spiTransaction;
00141     uint8_t          TxBuffer[2];
00142     uint8_t          RxBuffer[2];
00143     bool            transferOK;
00144
00145     if (rnwMSB==0) {
00146         TxBuffer[0] = (Address<<1);
00147     } else {
00148         TxBuffer[0] = (Address & 0x7F);
00149     }
00150
00151     TxBuffer[1] = Data;
00152
00153     spiTransaction.count = 2;
00154     spiTransaction.txBuf = (void *) (TxBuffer);
00155     spiTransaction.rxBuf = (void *) (RxBuffer);
00156     transferOK = SPI_transfer(spi, &spiTransaction);
00157
00158     return (transferOK? 0 : 1);
00159 }
00160
00161 //*****
00162 //
00173 //
00174 //*****
00175 uint8_t SPI_write_16bits(SPI_Handle spi, uint8_t Address, uint8_t Data) {
00176
00177     SPI_Transaction spiTransaction;
00178     uint16_t        TxBuffer[2];
00179     uint16_t        RxBuffer[2];
00180     bool            transferOK;
00181
00182     TxBuffer[0] = (Address << 8) + Data;
00183
00184     spiTransaction.count = 1;
00185     spiTransaction.txBuf = (void *) (TxBuffer);
00186     spiTransaction.rxBuf = (void *) (RxBuffer);
00187     transferOK = SPI_transfer(spi, &spiTransaction);
00188
00189     return (transferOK? 0 : 1);
00190 }

```

3.47 hal_SPI.h File Reference

Functions to manage SPI on CC3220SF.

```
#include <stddef.h>
#include <stdint.h>
#include <ti/drivers/SPI.h>
```

Functions

- **uint8_t SPI_read_16bits (SPI_Handle spi, uint8_t Address, uint8_t *Data)**
SPI Read 16 bits Function.
- **uint8_t SPI_read_8bits (SPI_Handle spi, uint8_t Address, uint8_t *RxBuffer, uint8_t nregs, uint8_t rnwMSB)**
SPI Read 8 bits Function.
- **uint8_t SPI_write_16bits (SPI_Handle spi, uint8_t Address, uint8_t Data)**
SPI Write 16 bits Function.

- `uint8_t SPI_write_8bits (SPI_Handle spi, uint8_t Address, uint8_t Data, uint8_t rnwMSB)`
SPI Write 8 bits Function.
- `SPI_Handle Startup_SPI (uint_least8_t index, uint32_t dataSize, uint32_t dataRate)`
SPI Start up Function.

3.47.1 Detailed Description

Functions to manage SPI on CC3220SF.

Version

1.0

Date

07/05/2019

Author

A.Irizar @title SPI CC3220SF functions

Definition in file [hal_SPI.h](#).

3.47.2 Function Documentation

3.47.2.1 SPI_read_16bits() `uint8_t SPI_read_16bits (`
 `SPI_Handle spi,`
 `uint8_t Address,`
 `uint8_t * RxBuffer)`

SPI Read 16 bits Function.

This function

1. Reads 16 bits from SPI

Parameters

in	<i>SPI_Handle</i>	spi
in	<i>uint8_t</i>	Address
in	<i>uint8_t</i>	*RxBuffer pointer to RxBuffer

Returns

`bool transferOK`

Definition at line 106 of file [hal_SPI.c](#).

```

00106
00107
00108     SPI_Transaction spiTransaction;
00109     uint16_t         TxBuffer[2];
00110
00111     bool            transferOK;
00112
00113     TxBuffer[0] = ((Address & 0x7F) + 0x80)«8;
00114
00115     spiTransaction.count = 1;
00116     spiTransaction.txBuf = (void *) TxBuffer;
00117     spiTransaction.rxBuf = (void *) RxBuffer;
00118     transferOK = SPI_transfer(spi, &spiTransaction);
00119
00120     return (transferOK? 0 : 1);
00121 }
```

3.47.2.2 SPI_read_8bits() [uint8_t SPI_read_8bits \(](#)

```

    SPI_Handle spi,
    uint8_t Address,
    uint8_t * RxBuf,
    uint8_t nregs,
    uint8_t rnwMSB )
```

SPI Read 8 bits Function.

This function

1. Reads 8 bits from SPI

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
in	<i>uint8_t</i>	<i>Address</i>
in	<i>uint8_t</i>	* <i>RxBuf</i> pointer to <i>RxBuf</i>
in	<i>uint8_t</i>	<i>nregs</i> Number of registers to read
in	<i>uint8_t</i>	<i>rnwMSB</i>

Returns

bool transferOK

Definition at line 67 of file [hal_SPI.c](#).

```

00067
00068     {
00069     SPI_Transaction spiTransaction;
00070     uint8_t         TxBuffer[32];
00071     RxBuffer[0] = 0;
00072     RxBuffer[1] = 0;
00073
00074     bool            transferOK;
00075
00076     if (rnwMSB==0) {
00077         TxBuffer[0] = ((Address«1) | 0x01);
00078     } else {
00079         TxBuffer[0] = (Address & 0x7F) | 0x80;
00080     }
00081     TxBuffer[1] = 0x00;
00082
00083 }
```

```

00084     spiTransaction.count = nregs+1;
00085     spiTransaction.txBuf = (void *)TxBuffer;
00086     spiTransaction.rxBuf = (void *)RxBuffer;
00087     transferOK = SPI_transfer(spi, &spiTransaction);
00088
00089     return (transferOK? 0 : 1);
00090 }

```

3.47.2.3 SPI_write_16bits() `uint8_t SPI_write_16bits (`

```

    SPI_Handle spi,
    uint8_t Address,
    uint8_t Data )

```

SPI Write 16 bits Function.

This function

1. Writes 16 bits to SPI

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
in	<i>uint8_t</i>	<i>Address</i>
in	<i>uint8_t</i>	<i>Data Data to write</i>

Returns

`bool transferOK`

Definition at line 175 of file [hal_SPI.c](#).

```

00175
00176
00177     SPI_Transaction spiTransaction;
00178     uint16_t         TxBuffer[2];
00179     uint16_t         RxBuffer[2];
00180     bool             transferOK;
00181
00182     TxBuffer[0] = (Address << 8) + Data;
00183
00184     spiTransaction.count = 1;
00185     spiTransaction.txBuf = (void *) (TxBuffer);
00186     spiTransaction.rxBuf = (void *) (RxBuffer);
00187     transferOK = SPI_transfer(spi, &spiTransaction);
00188
00189     return (transferOK? 0 : 1);
00190 }

```

3.47.2.4 SPI_write_8bits() `uint8_t SPI_write_8bits (`

```

    SPI_Handle spi,
    uint8_t Address,
    uint8_t Data,
    uint8_t rnwMSB )

```

SPI Write 8 bits Function.

This function

\$PWIrites 8 bits to SPI

`SPI_Handle`

Parameters

in	<i>SPI_Handle</i>	spi
in	<i>uint8_t</i>	Address
in	<i>uint8_t</i>	Data Data to write
in	<i>uint8_t</i>	rnwMSB

Returns

bool transferOK

Definition at line 138 of file [hal_SPI.c](#).

```

00138
00139
00140     SPI_Transaction spiTransaction;
00141     uint8_t          TxBuffer[2];
00142     uint8_t          RxBuffer[2];
00143     bool            transferOK;
00144
00145     if (rnwMSB==0) {
00146         TxBuffer[0] = (Address<<1);
00147     } else {
00148         TxBuffer[0] = (Address & 0x7F);
00149     }
00150
00151     TxBuffer[1] = Data;
00152
00153     spiTransaction.count = 2;
00154     spiTransaction.txBuf = (void *) (TxBuffer);
00155     spiTransaction.rxBuf = (void *) (RxBuffer);
00156     transferOK = SPI_transfer(spi, &spiTransaction);
00157
00158     return (transferOK? 0 : 1);
00159 }
```

3.47.2.5 Startup_SPI() *SPI_Handle* Startup_SPI (

```

    uint_least8_t index,
    uint32_t      dataSize,
    uint32_t      dataRate )
```

SPI Start up Function.

This function

1. Initiates SPI on CC3220SF

Parameters

in	<i>uint_</i> ↔ <i>least8_t</i>	index
in	<i>uint32_t</i>	dataSize
in	<i>uint32_t</i>	dataRate

Returns*SPI_Handle* spi

Definition at line 35 of file [hal_SPI.c](#).

```

00035
00036
00037     SPI_Params spiParams;
00038     SPI_Handle spi;
00039
00040     SPI_Params_init(&spiParams);                                {
00041     // Assumes SPI_init() has
00042     // already been started
00043     spiParams.mode = SPI_MASTER;
00044     spiParams.transferMode = SPI_MODE_BLOCKING;
00045     spiParams.frameFormat = SPI_POL0_PH0;
00046     spiParams.dataSize = dataSize;
00047     spiParams.bitRate = dataRate;
00048     spi = SPI_open(index, &spiParams);
00049     return spi;
00049 }
```

3.48 hal_SPI.h

```

00001 #ifndef HAL_SPI_H_
00010 #define HAL_SPI_H_
00011
00012 //*****
00013 //***** INCLUDES *****
00014 //***** INCLUDES *****
00015 //***** FUNCTION PROTOTYPES *****
00016 #include <stddef.h>
00017 #include <stdint.h>
00018 #include <ti/drivers/SPI.h>
00019
00020 //*****
00021 //***** FUNCTION PROTOTYPES *****
00022 //*****
00023 SPI_Handle Startup_SPI(uint_least8_t index, uint32_t dataSize, uint32_t dataRate);
00024 uint8_t SPI_write_8bits(SPI_Handle spi, uint8_t Address, uint8_t Data, uint8_t rnwMSB);
00025 uint8_t SPI_read_8bits(SPI_Handle spi, uint8_t Address, uint8_t *RxBuffer, uint8_t nregs, uint8_t rnwMSB);
00026 uint8_t SPI_write_16bits(SPI_Handle spi, uint8_t Address, uint8_t Data);
00027 uint8_t SPI_read_16bits(SPI_Handle spi, uint8_t Address, uint8_t *Data);
00028
00029 #endif /* HAL_SPI_H_ */
```

3.49 hal_Timer.c File Reference

Functions for Timer.

```
#include <ti/drivers/Timer.h>
#include <ti/drivers/Power.h>
#include <stdint.h>
#include "STARPORTS_App.h"
```

Functions

- [Timer_Handle Startup_Continuous_Timer](#) (uint8_t index, uint32_t period)
Timer Start up continuous Function.
- [Timer_Handle Startup_Oneshot_Timer](#) (uint8_t index, uint32_t interval)
Timer Start up One shot Function.
- void [timer0Callback](#) (Timer_Handle myHandle)
Timer 0 callback Function.
- void [timer1Callback](#) (Timer_Handle myHandle)
Timer 1 callback Function.

Variables

- `uint8_t Timer0Event`
- `uint8_t Timer1Event`

3.49.1 Detailed Description

Functions for Timer.

Version

1.0

Date

07/05/2019

Author

A.Irizar @tittle Timer functions

Definition in file [hal_Timer.c](#).

3.49.2 Function Documentation

3.49.2.1 Startup_Continuous_Timer() `Timer_Handle Startup_Continuous_Timer (`
`uint8_t index,`
`uint32_t period)`

Timer Start up continuous Function.

This function

1. initiates Timer as continuous

Parameters

in	<code>uint8_t</code>	index
in	<code>uint32_t</code>	period

Returns

`Timer_Handle timer`

Definition at line 73 of file hal_Timer.c.

```
00073
00074
00075     Timer_Handle timer;
00076     Timer_Params params;
00077
00078     Timer_Params_init(&params);
00079     params.period = period;
00080     params.periodUnits = Timer_PERIOD_US;
00081     params.timerMode = Timer_CONTINUOUS_CALLBACK;
00082     params.timerCallback = timer0Callback;
00083
00084     timer = Timer_open(index, &params);
00085
00086     return timer;
00087 }
```

3.49.2.2 Startup_Oneshot_Timer() Timer_Handle Startup_Oneshot_Timer (

```
    uint8_t index,
    uint32_t interval )
```

Timer Start up One shot Function.

This function

1. initiates Timer as one shot

Parameters

in	<i>uint8_t</i>	index
in	<i>uint32_t</i>	interval

Returns

Timer_Handle timer

Definition at line 102 of file hal_Timer.c.

```
00102
00103
00104     Timer_Handle timer;
00105     Timer_Params params;
00106
00107     Timer_Params_init(&params);
00108     params.period = interval;
00109     params.periodUnits = Timer_PERIOD_US;
00110     params.timerMode = Timer_ONESHOT_CALLBACK;
00111     params.timerCallback = timer1Callback;
00112
00113     timer = Timer_open(index, &params);
00114
00115     return timer;
00116 }
```

3.49.2.3 timer0Callback() void timer0Callback (

```
    Timer_Handle myHandle )
```

Timer 0 callback Function.

This function

1. Is a callback for timer 0

Parameters

in	<i>Timer_Handle</i>	myHandle
----	---------------------	----------

Returns

None

Definition at line 40 of file [hal_Timer.c](#).

```
00040
00041     Timer0Event = 1;
00042 }
```

3.49.2.4 timer1Callback() `void timer1Callback (`
 `Timer_Handle myHandle)`

Timer 1 callback Function.

This function

1. Is a callback for timer 1

Parameters

in	<i>Timer_Handle</i>	myHandle
----	---------------------	----------

Returns

None

Definition at line 56 of file [hal_Timer.c](#).

```
00056
00057     Timer1Event = 1;
00058 }
```

3.49.3 Variable Documentation**3.49.3.1 Timer0Event** `uint8_t Timer0Event`

Definition at line 52 of file [STARPORTS_App.c](#).

3.49.3.2 Timer1Event `uint8_t Timer1Event`

Definition at line 53 of file [STARPORTS_App.c](#).

3.50 hal_Timer.c

```
00001
00010 //*****
00011 //           INCLUDES
00012 //*****
00013 #include <ti/drivers/Timer.h>
00014 #include <ti/drivers/Power.h>
00015 #include <stdint.h>
00016 #include "STARPORTS_App.h"
00017
00018 //*****
00019 //           GLOBALS
00020 //*****
00021 extern uint8_t Timer0Event;
00022 extern uint8_t Timer1Event;
00023
00024 //*****
00025 //           FUNCTIONS
00026 //*****
00027
00028 //*****
00029 //
00030 //
00031 //
00032 //
00033 //
00034 //
00035 //
00036 void timer0Callback(Timer_Handle myHandle) {
00037     Timer0Event = 1;
00038 }
00039
00040 void timer1Callback(Timer_Handle myHandle) {
00041     Timer1Event = 1;
00042 }
00043
00044 //*****
00045 //
00046 //
00047 //
00048 //
00049 //
00050 //
00051 //
00052 //
00053 //
00054 //
00055 //
00056 void timer1Callback(Timer_Handle myHandle) {
00057     Timer1Event = 1;
00058 }
00059
00060 //*****
00061 //
00062 //
00063 //
00064 //
00065 //
00066 //
00067 //
00068 //
00069 //
00070 //
00071 //
00072 //*****
00073 Timer_Handle Startup_Continuous_Timer(uint8_t index, uint32_t period) {
00074
00075     Timer_Handle timer;
00076     Timer_Parms params;
00077
00078     Timer_Parms_init(&params);
00079     params.period = period;
00080     params.periodUnits = Timer_PERIOD_US;
00081     params.timerMode = Timer_CONTINUOUS_CALLBACK;
00082     params.timerCallback = timer0Callback;
00083
00084     timer = Timer_open(index, &params);
00085
00086     return timer;
00087 }
00088
00089 //*****
00090 //
00091 //
00092 //
00093 //
00094 //
00095 //
00096 //
00097 //
00098 //
00099 //
00100 //
00101 //*****
00102 Timer_Handle Startup_Oneshot_Timer(uint8_t index, uint32_t interval) {
00103
00104     Timer_Handle timer;
00105     Timer_Parms params;
00106
00107     Timer_Parms_init(&params);
00108     params.period = interval;
00109     params.periodUnits = Timer_PERIOD_US;
00110     params.timerMode = Timer_ONESHOT_CALLBACK;
00111     params.timerCallback = timer1Callback;
00112
00113     timer = Timer_open(index, &params);
00114
00115     return timer;
00116 }
```

3.51 hal_Timer.h File Reference

Functions for Timer.

```
#include <stdint.h>
#include <ti/drivers/Timer.h>
```

Functions

- Timer_Handle [Startup_Continuous_Timer](#) (uint8_t index, uint32_t period)
Timer Start up continuous Function.
- Timer_Handle [Startup_Oneshot_Timer](#) (uint8_t index, uint32_t interval)
Timer Start up One shot Function.
- void [timer0Callback](#) (Timer_Handle myHandle)
Timer 0 callback Function.
- void [timer1Callback](#) (Timer_Handle myHandle)
Timer 1 callback Function.

3.51.1 Detailed Description

Functions for Timer.

Version

1.0

Date

07/05/2019

Author

A.Irizar @title Timer functions

Definition in file [hal_Timer.h](#).

3.51.2 Function Documentation

3.51.2.1 Startup_Continuous_Timer()

```
Timer_Handle Startup_Continuous_Timer (
    uint8_t index,
    uint32_t period )
```

Timer Start up continuous Function.

This function

1. initiates Timer as continuous

Parameters

in	<i>uint8_t</i> <i>_t</i>	index
in	<i>uint32_t</i> <i>_t</i>	period

Returns

Timer_Handle timer

Definition at line 73 of file [hal_Timer.c](#).

```

00073
00074
00075     Timer_Handle timer;
00076     Timer_Params params;
00077
00078     Timer_Params_init(&params);
00079     params.period = period;
00080     params.periodUnits = Timer_PERIOD_US;
00081     params.timerMode = Timer_CONTINUOUS_CALLBACK;
00082     params.timerCallback = timer0Callback;
00083
00084     timer = Timer_open(index, &params);
00085
00086     return timer;
00087 }
```

3.51.2.2 Startup_Oneshot_Timer() Timer_Handle Startup_Oneshot_Timer (

```

    uint8_t index,
    uint32_t interval )
```

Timer Start up One shot Function.

This function

1. initiates Timer as one shot

Parameters

in	<i>uint8_t</i> <i>_t</i>	index
in	<i>uint32_t</i> <i>_t</i>	interval

Returns

Timer_Handle timer

Definition at line 102 of file [hal_Timer.c](#).

```

00102
00103
00104     Timer_Handle timer;
00105     Timer_Params params;
00106
00107     Timer_Params_init(&params);
```

```
00108     params.period = interval;
00109     params.periodUnits = Timer_PERIOD_US;
00110     params.timerMode = Timer_ONESHOT_CALLBACK;
00111     params.timerCallback = timer1Callback;
00112
00113     timer = Timer_open(index, &params);
00114
00115     return timer;
00116 }
```

3.51.2.3 timer0Callback() `void timer0Callback (` `Timer_Handle myHandle)`

Timer 0 callback Function.

This function

1. Is a callback for timer 0

Parameters

in	<i>Timer_Handle</i>	myHandle
----	---------------------	----------

Returns

None

Definition at line 40 of file [hal_Timer.c](#).

```
00040
00041     Timer0Event = 1;
00042 }
```

3.51.2.4 timer1Callback() `void timer1Callback (` `Timer_Handle myHandle)`

Timer 1 callback Function.

This function

1. Is a callback for timer 1

Parameters

in	<i>Timer_Handle</i>	myHandle
----	---------------------	----------

Returns

None

Definition at line 56 of file [hal_Timer.c](#).

```
00056     Timer1Event = 1;
00058 }
```

3.52 hal_Timer.h

```
00001
00010 #ifndef HAL_TIMER_H_
00011 #define HAL_TIMER_H_
00012
00013 //*****
00014 //      INCLUDES
00015 //*****
00016 #include <stdint.h>
00017 #include <ti/drivers/Timer.h>
00018
00019 //*****
00020 //      FUNCTION PROTOTYPES
00021 //*****
00022 Timer_Handle Startup_Continuous_Timer(uint8_t index, uint32_t period);
00023 Timer_Handle Startup_Oneshot_Timer(uint8_t index, uint32_t interval);
00024 void timer0Callback(Timer_Handle myHandle);
00025 void timer1Callback(Timer_Handle myHandle);
00026
00027 #endif /* HAL_TIMER_H_ */
```

3.53 hal_TMP006.c File Reference

Functions to read data and configure TMP006 Temperature sensor.

```
#include <ti/drivers/I2C.h>
#include "hal_TMP006.h"
```

Functions

- `uint16_t GetID_TMP006 (I2C_Handle i2c, uint_least8_t Slave)`
TMP006 Get DevID Function.
- `int16_t GetTA_TMP006 (I2C_Handle i2c, uint_least8_t Slave)`
TMP006 Get Temperature Function.
- `uint8_t SetMOD_TMP006 (I2C_Handle i2c, uint_least8_t Slave, uint8_t cr)`
TMP006 Set Mode Function.

3.53.1 Detailed Description

Functions to read data and configure TMP006 Temperature sensor.

Version

1.0

Date

07/05/2019

Author

A.Irizar @title TMP006 Temperature sensor functions

Definition in file [hal_TMP006.c](#).

3.53.2 Function Documentation

```
3.53.2.1 GetID_TMP006() uint16_t GetID_TMP006 (
    I2C_Handle i2c,
    uint_least8_t Slave )
```

TMP006 Get DevID Function.

This function

1. Gets DevID value from TMP006

Parameters

in	<i>I2C_Handle</i>	i2c
in	<i>uint_</i> ↔ <i>least8_t</i>	Slave slave address

Returns

uint16_t DevId

Definition at line 117 of file [hal_TMP006.c](#).

```
00117
00118
00119     bool status;
00120     uint8_t writeBuffer;
00121     uint8_t readBuffer[2];
00122     I2C_Transaction i2cTransaction;
00123     uint16_t DevId;
00124
00125     writeBuffer = 0xFE;
00126
00127     i2cTransaction.slaveAddress = Slave;
00128     i2cTransaction.writeBuf = &writeBuffer;
00129     i2cTransaction.writeCount = 1;
00130     i2cTransaction.readBuf = readBuffer;
00131     i2cTransaction.readCount = 2;
00132     status = I2C_transfer(i2c, &i2cTransaction);
00133
00134     DevId = (readBuffer[0]«8) + readBuffer[1];
00135
00136     return DevId;
00137 }
```

```
3.53.2.2 GetTA_TMP006() int16_t GetTA_TMP006 (
    I2C_Handle i2c,
    uint_least8_t Slave )
```

TMP006 Get Temperature Function.

This function

1. Gets temperature value from TMP006

Parameters

in	<i>I2C_Handle</i>	i2c
in	<i>uint_↔ least8_t</i>	Slave slave address

Returns

int16_t Temp Temperature value

Definition at line 82 of file [hal_TMP006.c](#).

```

00082
00083
00084     bool status;
00085     uint8_t writeBuffer;
00086     uint8_t readBuffer[2];
00087     I2C_Transaction i2cTransaction;
00088     int16_t Temp;
00089
00090     writeBuffer = 0x01;
00091
00092     i2cTransaction.slaveAddress = Slave;
00093     i2cTransaction.writeBuf = &writeBuffer;
00094     i2cTransaction.writeCount = 1;
00095     i2cTransaction.readBuf = readBuffer;
00096     i2cTransaction.readCount = 2;
00097     status = I2C_transfer(i2c, &i2cTransaction);
00098
00099     Temp = (int16_t)((readBuffer[0] << 8) + readBuffer[1]) / 4;
00100
00101     return Temp;
00102 }
```

3.53.2.3 SetMOD_TMP006() *uint8_t SetMOD_TMP006 (*

```

    I2C_Handle i2c,
    uint_least8_t Slave,
    uint8_t cr )
```

TMP006 Set Mode Function.

This function

1. Sets TMP006 configuration as slave

Parameters

in	<i>I2C_Handle</i>	i2c
in	<i>uint_↔ least8_t</i>	Slave slave address
in	<i>uint8_t</i>	cr conversion rate

ReturnsSUCCESS_CONF_TMP006 / ERROR_WRONG_CR as described in [hal_TMP006.h](#)Definition at line 36 of file [hal_TMP006.c](#).

```

00036
00037
00038     uint8_t writeBuffer[3];
00039     I2C_Transaction i2cTransaction;
00040     bool status;
00041
00042     writeBuffer[0] = 0x02;
00043     if (cr==240)
00044         writeBuffer[1] = 0x70;
00045     else if (cr==120)
00046         writeBuffer[1] = 0x72;
00047     else if (cr==60)
00048         writeBuffer[1] = 0x74;
00049     else if (cr==30)
00050         writeBuffer[1] = 0x76;
00051     else if (cr==15)
00052         writeBuffer[1] = 0x78;
00053     else
00054         return ERROR_WRONG_CR;
00055
00056     writeBuffer[2] = 0x00;
00057
00058     i2cTransaction.slaveAddress = Slave;
00059     i2cTransaction.writeBuf = writeBuffer;
00060     i2cTransaction.writeCount = 3;
00061     i2cTransaction.readBuf = NULL;
00062     i2cTransaction.readCount = 0;
00063
00064     status = I2C_transfer(i2c, &i2cTransaction);
00065
00066     return SUCCESS_CONF_TMP006;
00067 }

```

3.54 hal_TMP006.c

```

00001
00010 //*****
00011 //          INCLUDES
00012 //*****
00013 //##include <stdint.h>
00014 //##include <stdbool.h>
00015 #include <ti/drivers/I2C.h>
00016 #include "hal_TMP006.h"
00017
00018 //*****
00019 //          FUNCTIONS
00020 //*****
00021
00022 //*****
00023 //
00034 //
00035 //*****
00036 uint8_t SetMOD_TMP006(I2C_Handle i2c, uint_least8_t Slave, uint8_t cr) {
00037
00038     uint8_t writeBuffer[3];
00039     I2C_Transaction i2cTransaction;
00040     bool status;
00041
00042     writeBuffer[0] = 0x02;
00043     if (cr==240)
00044         writeBuffer[1] = 0x70;
00045     else if (cr==120)
00046         writeBuffer[1] = 0x72;
00047     else if (cr==60)
00048         writeBuffer[1] = 0x74;
00049     else if (cr==30)
00050         writeBuffer[1] = 0x76;
00051     else if (cr==15)
00052         writeBuffer[1] = 0x78;
00053     else
00054         return ERROR_WRONG_CR;
00055
00056     writeBuffer[2] = 0x00;
00057
00058     i2cTransaction.slaveAddress = Slave;
00059     i2cTransaction.writeBuf = writeBuffer;
00060     i2cTransaction.writeCount = 3;
00061     i2cTransaction.readBuf = NULL;
00062     i2cTransaction.readCount = 0;
00063
00064     status = I2C_transfer(i2c, &i2cTransaction);
00065
00066     return SUCCESS_CONF_TMP006;
00067 }

```

```

00068
00069 //*****
00070 //
00080 //
00081 //*****
00082 int16_t GetTA_TMP006(I2C_Handle i2c, uint_least8_t Slave) {
00083
00084     bool status;
00085     uint8_t writeBuffer;
00086     uint8_t readBuffer[2];
00087     I2C_Transaction i2cTransaction;
00088     int16_t Temp;
00089
00090     writeBuffer = 0x01;
00091
00092     i2cTransaction.slaveAddress = Slave;
00093     i2cTransaction.writeBuf = &writeBuffer;
00094     i2cTransaction.writeCount = 1;
00095     i2cTransaction.readBuf = readBuffer;
00096     i2cTransaction.readCount = 2;
00097     status = I2C_transfer(i2c, &i2cTransaction);
00098
00099     Temp = (int16_t)((readBuffer[0] << 8) + readBuffer[1]) / 4;
00100
00101     return Temp;
00102 }
00103
00104 //*****
00105 //
00115 //
00116 //*****
00117 uint16_t GetID_TMP006(I2C_Handle i2c, uint_least8_t Slave) {
00118
00119     bool status;
00120     uint8_t writeBuffer;
00121     uint8_t readBuffer[2];
00122     I2C_Transaction i2cTransaction;
00123     uint16_t DevId;
00124
00125     writeBuffer = 0xFE;
00126
00127     i2cTransaction.slaveAddress = Slave;
00128     i2cTransaction.writeBuf = &writeBuffer;
00129     i2cTransaction.writeCount = 1;
00130     i2cTransaction.readBuf = readBuffer;
00131     i2cTransaction.readCount = 2;
00132     status = I2C_transfer(i2c, &i2cTransaction);
00133
00134     DevId = (readBuffer[0] << 8) + readBuffer[1];
00135
00136     return DevId;
00137 }
```

3.55 hal_TMP006.h File Reference

Functions to read data and configure TMP006 Temperature sensor.

```
#include <ti/drivers/I2C.h>
```

Macros

- #define ERROR_WRONG_CR 1
- #define SUCCESS_CONF_TMP006 0

Functions

- uint16_t **GetID_TMP006** (I2C_Handle i2c, uint_least8_t Slave)
TMP006 Get DevID Function.
- int16_t **GetTA_TMP006** (I2C_Handle i2c, uint_least8_t Slave)
TMP006 Get Temperature Function.
- uint8_t **SetMOD_TMP006** (I2C_Handle i2c, uint_least8_t Slave, uint8_t cr)
TMP006 Set Mode Function.

3.55.1 Detailed Description

Functions to read data and configure TMP006 Temperature sensor.

Version

1.0

Date

07/05/2019

Author

A.Irizar @title TMP006 Temperature sensor functions

Definition in file [hal_TMP006.h](#).

3.55.2 Macro Definition Documentation

3.55.2.1 ERROR_WRONG_CR #define ERROR_WRONG_CR 1

Definition at line [21](#) of file [hal_TMP006.h](#).

3.55.2.2 SUCCESS_CONF_TMP006 #define SUCCESS_CONF_TMP006 0

Definition at line [22](#) of file [hal_TMP006.h](#).

3.55.3 Function Documentation

3.55.3.1 GetID_TMP006() uint16_t GetID_TMP006 (

```
    I2C_Handle i2C,
    uint_least8_t Slave )
```

TMP006 Get DevID Function.

This function

1. Gets DevID value from TMP006

Parameters

in	<i>I2C_Handle</i>	i2c
in	<i>uint_<= least8_t</i>	Slave slave address

Returns

uint16_t DevId

Definition at line 117 of file [hal_TMP006.c](#).

```

00117
00118
00119     bool status;
00120     uint8_t writeBuffer;
00121     uint8_t readBuffer[2];
00122     I2C_Transaction i2cTransaction;
00123     uint16_t DevId;
00124
00125     writeBuffer = 0xFE;
00126
00127     i2cTransaction.slaveAddress = Slave;
00128     i2cTransaction.writeBuf = &writeBuffer;
00129     i2cTransaction.writeCount = 1;
00130     i2cTransaction.readBuf = readBuffer;
00131     i2cTransaction.readCount = 2;
00132     status = I2C_transfer(i2c, &i2cTransaction);
00133
00134     DevId = (readBuffer[0] << 8) + readBuffer[1];
00135
00136     return DevId;
00137 }
```

3.55.3.2 GetTA_TMP006() int16_t GetTA_TMP006 (
I2C_Handle i2c,
uint_least8_t Slave)

TMP006 Get Temperature Function.

This function

1. Gets temperature value from TMP006

Parameters

in	<i>I2C_Handle</i>	i2c
in	<i>uint_<= least8_t</i>	Slave slave address

Returns

int16_t Temp Temperature value

Definition at line 82 of file [hal_TMP006.c](#).

```

00082
00083
00084     bool status;
```

```

00085     uint8_t writeBuffer;
00086     uint8_t readBuffer[2];
00087     I2C_Transaction i2cTransaction;
00088     int16_t Temp;
00089
00090     writeBuffer = 0x01;
00091
00092     i2cTransaction.slaveAddress = Slave;
00093     i2cTransaction.writeBuf = &writeBuffer;
00094     i2cTransaction.writeCount = 1;
00095     i2cTransaction.readBuf = readBuffer;
00096     i2cTransaction.readCount = 2;
00097     status = I2C_transfer(i2c, &i2cTransaction);
00098
00099     Temp = (int16_t)((readBuffer[0] << 8) + readBuffer[1])) / 4;
00100
00101     return Temp;
00102 }
```

3.55.3.3 SetMOD_TMP006()

```

uint8_t SetMOD_TMP006 (
    I2C_Handle i2c,
    uint_least8_t Slave,
    uint8_t CR )
```

TMP006 Set Mode Function.

This function

1. Sets TMP006 configuration as slave

Parameters

in	<i>I2C_Handle</i>	i2c
in	<i>uint_</i> ← <i>least8_t</i>	Slave slave address
in	<i>uint8_t</i>	cr conversion rate

Returns

SUCCESS_CONF_TMP006 / ERROR_WRONG_CR as described in [hal_TMP006.h](#)

Definition at line 36 of file [hal_TMP006.c](#).

```

00036
00037
00038     uint8_t writeBuffer[3];
00039     I2C_Transaction i2cTransaction;
00040     bool status;
00041
00042     writeBuffer[0] = 0x02;
00043     if (cr==240)
00044         writeBuffer[1] = 0x70;
00045     else if (cr==120)
00046         writeBuffer[1] = 0x72;
00047     else if (cr==60)
00048         writeBuffer[1] = 0x74;
00049     else if (cr==30)
00050         writeBuffer[1] = 0x76;
00051     else if (cr==15)
00052         writeBuffer[1] = 0x78;
00053     else
00054         return ERROR_WRONG_CR;
00055
00056     writeBuffer[2] = 0x00;
00057 }
```

```

00058     i2cTransaction.slaveAddress = Slave;
00059     i2cTransaction.writeBuf = writeBuffer;
00060     i2cTransaction.writeCount = 3;
00061     i2cTransaction.readBuf = NULL;
00062     i2cTransaction.readCount = 0;
00063
00064     status = I2C_transfer(i2c, &i2cTransaction);
00065
00066     return SUCCESS_CONF_TMP006;
00067 }
```

3.56 hal_TMP006.h

```

00001
00010 #ifndef HAL_TMP006_H_
00011 #define HAL_TMP006_H_
00012
00013 //*****
00014 //           INCLUDES
00015 //*****
00016 #include <ti/drivers/I2C.h>
00017
00018 //*****
00019 //           DEFINES
00020 //*****
00021 #define ERROR_WRONG_CR    1
00022 #define SUCCESS_CONF_TMP006 0
00023
00024 //*****
00025 //           FUNCTION PROTOTYPES
00026 //*****
00027 uint8_t SetMOD_TMP006(I2C_Handle i2c, uint_least8_t Slave, uint8_t cr);
00028 int16_t GetTA_TMP006(I2C_Handle i2c, uint_least8_t Slave);
00029 uint16_t GetID_TMP006(I2C_Handle i2c, uint_least8_t Slave);
00030
00031 #endif /* HAL_TMP006_H_ */
```

3.57 hal_UART.c File Reference

Functions to manage UART on CC3220SF.

```
#include <stdarg.h>
#include <stdbool.h>
#include <ti/drivers/UART.h>
```

Functions

- **uint8_t GetLine_UART** (UART_Handle uart, unsigned char *buf)
UART Get line Function.
- **void Message** (const char *str)
UART Message Function.
- **void putch** (char ch)
UART Putch Function.
- **int Report** (const char *pcFormat,...)
UART Report Function.
- **UART_Handle Startup_UART** (uint_least8_t index, uint32_t baudRate)
UART Start up Function.
- **bool UART_resetInputBuffer** (UART_Handle hUart)
UART Reset input buffer Function.
- **int vsnprintf** (char *s, size_t n, const char *format, va_list arg)

Variables

- `UART_Handle uart0`

3.57.1 Detailed Description

Functions to manage UART on CC3220SF.

Version

1.0

Date

07/05/2019

Author

A.Irizar @tittle UART management functions

Definition in file [hal_UART.c](#).

3.57.2 Function Documentation

3.57.2.1 GetLine_UART() `uint8_t GetLine_UART (`
`UART_Handle uart,`
`unsigned char * buf)`

UART Get line Function.

This function

1. Gets a string of chars from the UART until a newline char is encountered

Parameters

in	<code>UART_Handle</code>	<code>uart</code>
in	<code>unsigned</code>	<code>char *buf</code> pointer to <code>unsigned char</code> where the line will be stored

Returns

`uint8_t sz`

Definition at line 108 of file [hal_UART.c](#).
00108 {

```

00109     int16_t i = 0;
00110     uint8_t sz;
00111     unsigned char c;
00112
00113     do {
00114         UART_read(uart, &c, 1);
00115         buf[i++] = c;
00116         if (i == 255) {
00117             i = 0;
00118             break;
00119         }
00120     } while (c != '\r');
00121     UART_read(uart, &c, 1); // Read final
00122
00123     sz = (i > 1) ? (i - 1) : 0;
00124     return sz;
00125 }
00126 }
```

3.57.2.2 Message() void Message (const char * str)

UART Message Function.

This function

1. Outputs a character string to the console

Parameters

in	const	char *str pointer to the string to be printed
----	-------	---

Returns

None

Definition at line 187 of file [hal_UART.c](#).

```

00188 {
00189 #ifdef UART_NONPOLLING
00190     UART_write(uart, str, strlen(str));
00191 #else
00192     UART_writePolling(uart0, str, strlen(str));
00193 #endif
00194 }
```

3.57.2.3 putch() void putch (char ch)

UART Putch Function.

This function

1. Outputs a character to the console

Parameters

in	char	ch character to be printed
----	------	----------------------------

Returns

None

Definition at line 208 of file [hal_UART.c](#).

```
00209 {
00210     UART_writePolling(uart0, &ch, 1);
00211 }
```

3.57.2.4 Report() int Report (
 const char * pcFormat,
 ...)

UART Report Function.

This function

1. Prints the formatted string on to the console

Parameters

in	const	char *pcFormat pointer to the character string
----	-------	--

Returns

int iRet count of characters printed

Definition at line 140 of file [hal_UART.c](#).

```
00141 {
00142     int iRet = 0;
00143     char      *pcBuff;
00144     char      *pcTemp;
00145     int iSize = 256;
00146     va_list list;
00147
00148     pcBuff = (char*)malloc(iSize);
00149     if(pcBuff == NULL){
00150         return(-1);
00151     }
00152     while(1){
00153         va_start(list,pcFormat);
00154         iRet = vsnprintf(pcBuff, iSize, pcFormat, list);
00155         va_end(list);
00156         if((iRet > -1) && (iRet < iSize)){
00157             break;
00158         }else{
00159             iSize *= 2;
00160             if((pcTemp = realloc(pcBuff, iSize)) == NULL){
00161                 Message("Could not reallocate memory\n\r");
00162                 iRet = -1;
00163                 break;
00164             }else{
00165                 pcBuff = pcTemp;
00166             }
00167     }
```

```
00168      }
00169      Message(pcBuff);
00170      free(pcBuff);
00171
00172      return(iRet);
00173 }
```

3.57.2.5 Startup_UART() `UART_Handle Startup_UART (`
 `uint_least8_t index,`
 `uint32_t baudRate)`

UART Start up Function.

This function

Parameters

in	<i>UART_Handle</i>	uart
----	--------------------	------

Returns

```
bool bUartBufferIsAvailable
```

Definition at line 79 of file [hal_UART.c](#).

```
00079
00080
00081     bool bUartBufferIsAvailable = false;
00082
00083     do{
00084         if (UART_control(hUart, UART_CMD_ISAVAILABLE, &bUartBufferIsAvailable) < 0){
00085             return false;
00086         }
00087         if (bUartBufferIsAvailable){
00088             UART_read(hUart, NULL, 1);
00089         }
00090     } while (bUartBufferIsAvailable);
00091
00092     return true;
00093 }
```

3.57.2.7 vsnprintf() `int vsnprintf (`

```
    char * s,
    size_t n,
    const char * format,
    va_list arg )
```

3.57.3 Variable Documentation**3.57.3.1 uart0** `UART_Handle uart0`

Definition at line 65 of file [STARPORTS_App.c](#).

3.58 hal_UART.c

```
00001 //*****
00010 //***** INCLUDES *****
00011 //***** INCLUDES *****
00012 //***** INCLUDES *****
00013 #include <stdarg.h>
00014 #include <stdbool.h>
00015 #include <ti/drivers/UART.h>
00016
00017 //*****
00018 //***** GLOBALS *****
00019 //*****
00020 extern int vsnprintf(char * s,
00021                      size_t n,
00022                      const char * format,
00023                      va_list arg);
00024
00025 extern UART_Handle uart0;
00026
00027 //*****
```

```

00028 //                                FUNCTIONS
00029 //*****
00030
00031 //*****
00032 //
00042 //
00043 //*****
00044 UART_Handle Startup_UART(uint_least8_t index, uint32_t baudRate) {
00045
00046     UART_Handle uart;
00047     UART_Params uartParams;
00048
00049     UART_init();
00050
00051     UART_Params_init(&uartParams);
00052     uartParams.writeDataMode = UART_DATA_BINARY;
00053     uartParams.readDataMode = UART_DATA_BINARY;
00054     uartParams.readReturnMode = UART_RETURN_FULL;
00055     uartParams.readMode      = UART_MODE_BLOCKING;
00056     uartParams.readEcho    = UART_ECHO_OFF;
00057     uartParams.baudRate   = baudRate;
00058
00059     uart = UART_open(index, &uartParams);
00060     if (uart == NULL) {                                         // UART_open()
00061         while (1);
00062     }
00063
00064     return uart;
00065 }
00066
00067 //*****
00068 //
00077 //
00078 //*****
00079 bool UART_resetInputBuffer(UART_Handle hUart) {
00080
00081     bool bUartBufferIsAvailable = false;
00082
00083     do{                                                 //while data
00084         if (UART_control(hUart, UART_CMD_ISAVAILABLE, &bUartBufferIsAvailable) < 0){
00085             return false;
00086         }
00087         if (bUartBufferIsAvailable){
00088             UART_read(hUart, NULL, 1);
00089         }
00090     } while (bUartBufferIsAvailable);
00091
00092     return true;
00093 }
00094
00095 //*****
00106 //
00107 //*****
00108 uint8_t GetLine_UART(UART_Handle uart, unsigned char *buf) {
00109     int16_t i = 0;
00110     uint8_t sz;
00111     unsigned char c;
00112
00113     do {
00114         UART_read(uart,&c,1);
00115         buf[i++]=c;
00116         if (i==255) {
00117             i = 0;
00118             break;
00119         }
00120     } while (c != '\r');
00121     UART_read(uart,&c,1);                                     // Read final
00122     '\n' character
00123     sz = (i>1)? (i-1) : 0;
00124     return sz;
00125
00126 }
00127
00128 //*****
00129 //
00138 //
00139 //*****
00140 int Report(const char *pcFormat,...)
00141 {
00142     int iRet = 0;
00143     char      *pcBuff;
00144     char      *pcTemp;
00145     int iSize = 256;

```

```
00146     va_list list;
00147
00148     pcBuff = (char*)malloc(iSize);
00149     if(pcBuff == NULL){
00150         return(-1);
00151     }
00152     while(1){
00153         va_start(list,pcFormat);
00154         iRet = vsnprintf(pcBuff, iSize, pcFormat, list);
0015500(list;Estart(list);) TJ0-7.97Td[(60150)] TJ0.880.50rg0.880.50RG[-5000(if)] TJ0g0G[(lloc(iRet)->iRet)-(-iRet= (iRet)-<uff,
70150
8015500({0150}) TJ0.880.50rg0.880.50R(elswhile)] TJ0g0G{
9 i051{*
lrg250..38050R("Could0(==)-not00(=)-rea51maate00(=)-memory00(=/F1/F436.9738242.691r.97Tn00(=/FT/F436.97384.0821r.97Tn00(=/F1/F436.97384.1841r.9
7.97Td[700151000(pclist);) TJ0-7.97Td[71051;

J0.380.250.125rg0.380.250.125RG[(char)] TJ0g0RG[-0501B51;
J001rg001R(ua_s0hile)] TJ0g0GBuff,
```

- void `putch` (char ch)
UART Putch Function.
- int `Report` (const char *pcFormat,...)
UART Report Function.
- UART_Handle `Startup_UART` (uint_least8_t index, uint32_t baudRate)
UART Start up Function.
- bool `UART_resetInputBuffer` (UART_Handle hUart)
UART Reset input buffer Function.

3.59.1 Detailed Description

Functions to manage UART on CC3220SF.

Version

1.0

Date

07/05/2019

Author

A.Irizar @tittle UART management functions

Definition in file [hal_UART.h](#).

3.59.2 Macro Definition Documentation

3.59.2.1 ERR_PRINT `#define ERR_PRINT(` `x)`

Value:

```
[%s]  \n\r", \
Report ("Error [%d] at line [%d] in function
x, __LINE__, \
__FUNCTION__)
```

Definition at line [23](#) of file [hal_UART.h](#).

3.59.2.2 UART_PRINT `#define UART_PRINT Report`

Definition at line [22](#) of file [hal_UART.h](#).

3.59.3 Function Documentation

3.59.3.1 GetLine_UART() `uint8_t GetLine_UART (` `UART_Handle uart,` `unsigned char * buf)`

UART Get line Function.

This function

1. Gets a string of chars from the UART until a newline char is encountered

Parameters

in	<i>UART_Handle</i>	uart
in	<i>unsigned</i>	char *buf pointer to unsigned char where the line will be stored

Returns

uint8_t sz

Definition at line 108 of file hal_UART.c.

```

00108
00109     int16_t i = 0;
00110     uint8_t sz;
00111     unsigned char c;
00112
00113     do {
00114         UART_read(uart,&c,1);
00115         buf[i++]=c;
00116         if (i==255) {
00117             i = 0;
00118             break;
00119         }
00120     } while (c !='\r');
00121     UART_read(uart,&c,1);
00122     // Read final
00123     '\n' character
00124     sz = (i>1)? (i-1) : 0;
00125
00126 }
```

3.59.3.2 Message() void Message (
 const char * str)

UART Message Function.

This function

1. Outputs a character string to the console

Parameters

in	<i>const</i>	char *str pointer to the string to be printed
----	--------------	---

Returns

None

Definition at line 187 of file hal_UART.c.

```

00188 {
00189 #ifdef UART_NONPOLLING
00190     UART_write(uart, str, strlen(str));
00191 #else
00192     UART_writePolling(uart0, str, strlen(str));
00193 #endif
00194 }
```

3.59.3.3 putch() void putch (char ch)

UART Putch Function.

This function

1. Outputs a character to the console

Parameters

in	char	ch character to be printed
----	------	----------------------------

Returns

None

Definition at line 208 of file [hal_UART.c](#).

```
00209 {
00210     UART_writePolling(uart0, &ch, 1);
00211 }
```

3.59.3.4 Report() int Report (const char * pcFormat, ...)

UART Report Function.

This function

1. Prints the formatted string on to the console

Parameters

in	const	char *pcFormat pointer to the character string
----	-------	--

Returns

int iRet count of characters printed

Definition at line 140 of file [hal_UART.c](#).

```
00141 {
00142     int iRet = 0;
00143     char      *pcBuff;
00144     char      *pcTemp;
00145     int iSize = 256;
00146     va_list list;
00147
00148     pcBuff = (char*)malloc(iSize);
00149     if(pcBuff == NULL){
00150         return(-1);
00151     }
00152     while(1){
```

```

00153     va_start(list,pcFormat);
00154     iRet = vsnprintf(pcBuff, iSize, pcFormat, list);
00155     va_end(list);
00156     if((iRet > -1) && (iRet < iSize)){
00157         break;
00158     }else{
00159         iSize *= 2;
00160         if((pcTemp = realloc(pcBuff, iSize)) == NULL){
00161             Message("Could not reallocate memory\n\r");
00162             iRet = -1;
00163             break;
00164         }else{
00165             pcBuff = pcTemp;
00166         }
00167     }
00168 }
00169 Message(pcBuff);
00170 free(pcBuff);
00171
00172 return(iRet);
00173 }
```

3.59.3.5 Startup_UART()

```
UART_HandleTypeDef Startup_UART (
    uint_least8_t index,
    uint32_t baudRate )
```

UART Start up Function.

This function

1. Starts a UART peripheral

Parameters

in	<i>uint_</i> ← <i>least8_t</i>	index Logical peripheral number for the UART indexed into the UART_config table
in	<i>uint32_t</i>	baudRate UART speed data transfer, use one of the allowed ones

Returns

UART_HandleTypeDef uart A #UART_HandleTypeDef upon success. NULL if an error occurs, or if the indexed UART peripheral is already opened.

Definition at line 44 of file [hal_UART.c](#).

```

00044
00045
00046     UART_HandleTypeDef uart;
00047     UART_Params uartParams;
00048
00049     UART_init();
00050
00051     UART_Params_init(&uartParams);
00052     uartParams.writeDataMode = UART_DATA_BINARY;
00053     uartParams.readDataMode = UART_DATA_BINARY;
00054     uartParams.readReturnMode = UART_RETURN_FULL;
00055     uartParams.readMode      = UART_MODE_BLOCKING;
00056     uartParams.readEcho   = UART_ECHO_OFF;
00057     uartParams.baudRate = baudRate;
00058
00059     uart = UART_open(index, &uartParams);
00060     if (uart == NULL) {                                // UART_open()
00061         failed
00062         while (1);
00063     }
00064 }
```

```
00064     return uart;
00065 }
```

3.59.3.6 UART_resetInputBuffer()

```
bool UART_resetInputBuffer (
    UART_Handle huart )
```

UART Reset input buffer Function.

This function

1. Resets the input buffer of a UART peripheral

Parameters

in	<i>UART_Handle</i>	uart
----	--------------------	------

Returns

bool bUartBufferIsAvailable

Definition at line 79 of file [hal_UART.c](#).

```
00079
00080
00081     bool bUartBufferIsAvailable = false;
00082
00083     do{
00084         are presented in UART buffer we read each available byte to NULL pointer
00085         if (UART_control(huart, UART_CMD_ISAVAILABLE, &bUartBufferIsAvailable) < 0){
00086             return false;
00087         }
00088         if (bUartBufferIsAvailable){
00089             UART_read(huart, NULL, 1);
00090         }
00091     } while (bUartBufferIsAvailable);
00092
00093     return true;
00094 }
```

3.60 hal_UART.h

```
00001
00010 #ifndef HAL_UART_H_
00011 #define HAL_UART_H_
00012
00013 //*****
00014 // Includes
00015 //*****
00016 #include <stdbool.h>
00017 #include <ti/drivers/UART.h>
00018
00019 //*****
00020 // Defines
00021 //*****
00022 #define UART_PRINT Report
00023 #define ERR_PRINT(x) Report("Error [%d] at line [%d] in function [%s] \n\r", \
00024                                         x, __LINE__, \
00025                                         __FUNCTION__)
00026
00027 //*****
00028 //          FUNCTION PROTOTYPES
00029 //*****
00030 UART_Handle Startup_UART(uint_least8_t index, uint32_t baudRate);
00031 bool UART_resetInputBuffer(UART_Handle huart);
00032 uint8_t GetLine_UART(UART_Handle uart, unsigned char *buf);
00033 int Report(const char *pcFormat,...);
00034 void Message(const char *str);
00035 void putch(char ch);
00036
00037 #endif /* HAL_UART_H_ */
```

3.61 hal_WD.c File Reference

Functions to manage Watchdog.

```
#include <string.h>
#include <ti/drivers/I2C.h>
#include <ti/drivers/Watchdog.h>
#include "Board.h"
#include "hal_GPIO.h"
```

Functions

- Watchdog_Handle [Startup_Watchdog](#) (uint_least8_t index, uint32_t timeout)
Watchdog Startup Function.
- void [watchdogCallback](#) (uintptr_t watchdogHandle)
Watchdog callback Function.

Variables

- I2C_Handle [i2c](#)

3.61.1 Detailed Description

Functions to manage Watchdog.

Version

1.0

Date

07/05/2019

Author

A.Irizar @tittle Watchdog functions

Definition in file [hal_WD.c](#).

3.61.2 Function Documentation

```
3.61.2.1 Startup_Watchdog() Watchdog_Handle Startup_Watchdog (
    uint_least8_t index,
    uint32_t timeout )
```

Watchdog Startup Function.

This function

1. Starts up a watchdog instance

Parameters

in	<i>uint_</i> \leftrightarrow <i>least8_t</i>	index
in	<i>uint32_t</i>	timeout

Returns

Watchdog_Handle wd A #Watchdog_Handle upon success. NULL if an error occurs, or if the indexed UART peripheral is already opened.

Definition at line 60 of file [hal_WD.c](#).

```

00060
00061
00062     uint32_t reloadValue;
00063     Watchdog_Params wdParams;
00064     Watchdog_Handle wd;
00065
00066     Watchdog_init();
00067     Watchdog_Params_init(&wdParams);
00068     wdParams.callbackFxn = (Watchdog_Callback) watchdogCallback;
00069     wdParams.debugStallMode = Watchdog_DEBUG_STALL_ON;
00070     wdParams.resetMode = Watchdog_RESET_OFF;
00071
00072     wd = Watchdog_open(index, &wdParams);
00073     if (wd == NULL) {                                     // Error opening
00074         Watchdog
00075             while (1) {}
00076
00077     reloadValue = Watchdog_convertMsToTicks(wd, timeout);
00078
00079     if (reloadValue != 0) {
00080         Watchdog_setReload(wd, reloadValue);
00081     }
00082
00083     return wd;
00084 }
```

3.61.2.2 watchdogCallback() void watchdogCallback (

uintptr_t watchdogHandle)

Watchdog callback Function.

This function

1. Is a callback for watchdog.If the Watchdog Non-Maskable Interrupt (NMI) is called, set to '0' any output signal and disable the [Node](#)

Parameters

in	<i>uintptr_</i> \leftrightarrow <i>_t</i>	watchdogHandle
----	---	----------------

Returns

None

Definition at line 40 of file [hal_WD.c](#).

```

00040
00041
00042     I2C_close(i2c);
00043     I2C_As_GPIO_Low();                                // Puts SCL and SDA
00044     signals low to save power
00045     Node_Disable();                                 // Auto Shutdown
00045 }

```

3.61.3 Variable Documentation

3.61.3.1 i2c I2C_Handle i2c

Definition at line 66 of file [STARPORTS_App.c](#).

3.62 hal_WD.c

```

00001
00010 //*****
00011 //      INCLUDES
00012 //*****
00013 #include <string.h>
00014 #include <ti/drivers/I2C.h>
00015 #include <ti/drivers/Watchdog.h>
00016 #include "Board.h"
00017 #include "hal_GPIO.h"
00018
00019 //*****
00020 //      GLOBALS
00021 //*****
00022 extern I2C_Handle i2c;
00023
00024 //*****
00025 //      FUNCTIONS
00026 //*****
00027
00028 //*****
00029 //
00038 //
00039 //*****
00040 void watchdogCallback(uintptr_t watchdogHandle) {
00041
00042     I2C_close(i2c);
00043     I2C_As_GPIO_Low();                                // Puts SCL and SDA
00044     signals low to save power
00045     Node_Disable();                                 // Auto Shutdown
00045 }
00046
00047 //*****
00048 //
00058 //
00059 //*****
00060 Watchdog_Handle Startup_Watchdog(uint_least8_t index, uint32_t timeout) {
00061
00062     uint32_t reloadValue;
00063     Watchdog_Params wdParams;
00064     Watchdog_Handle wd;
00065
00066     Watchdog_init();
00067     Watchdog_Params_init(&wdParams);
00068     wdParams.callbackFxn = (Watchdog_Callback) watchdogCallback;
00069     wdParams.debugStallMode = Watchdog_DEBUG_STALL_ON;
00070     wdParams.resetMode = Watchdog_RESET_OFF;
00071
00072     wd = Watchdog_open(index, &wdParams);
00073     if (wd == NULL) {                                  // Error opening
00074         Watchdog
00074         while (1) {}
00075     }
00076
00077     reloadValue = Watchdog_convertMsToTicks(wd, timeout);
00078
00079     if (reloadValue != 0) {
00080         Watchdog_setReload(wd, reloadValue);
00081     }
00082
00083     return wd;
00084 }

```

3.63 hal_WD.h File Reference

Functions to manage Watchdog.

```
#include <stddef.h>
#include <stdint.h>
#include <ti/drivers/Watchdog.h>
```

Functions

- Watchdog_Handle [Startup_Watchdog](#) (uint_least8_t index, uint32_t timeout)
Watchdog Startup Function.

3.63.1 Detailed Description

Functions to manage Watchdog.

Version

1.0

Date

07/05/2019

Author

A.Irizar @tittle Watchdog functions

Definition in file [hal_WD.h](#).

3.63.2 Function Documentation

```
3.63.2.1 Startup_Watchdog() Watchdog_Handle Startup_Watchdog (
    uint_least8_t index,
    uint32_t timeout )
```

Watchdog Startup Function.

This function

1. Starts up a watchdog instance

Parameters

in	<i>uint_</i> ↔ <i>least8_t</i>	index
in	<i>uint32_t</i>	timeout

Returns

Watchdog_Handle wd A #Watchdog_Handle upon success. NULL if an error occurs, or if the indexed UART peripheral is already opened.

Definition at line 60 of file [hal_WD.c](#).

```

00060
00061
00062     uint32_t reloadValue;
00063     Watchdog_Params wdParams;
00064     Watchdog_Handle wd;
00065
00066     Watchdog_init();
00067     Watchdog_Params_init(&wdParams);
00068     wdParams.callbackFxn = (Watchdog_Callback) watchdogCallback;
00069     wdParams.debugStallMode = Watchdog_DEBUG_STALL_ON;
00070     wdParams.resetMode = Watchdog_RESET_OFF;
00071
00072     wd = Watchdog_open(index, &wdParams);
00073     if (wd == NULL) {                                     // Error opening
00074         Watchdog
00075             while (1) {}
00076
00077     reloadValue = Watchdog_convertMsToTicks(wd, timeout);
00078
00079     if (reloadValue != 0) {
00080         Watchdog_setReload(wd, reloadValue);
00081     }
00082
00083     return wd;
00084 }
```

3.64 hal_WD.h

```

00001
00010 #ifndef HAL_WD_H_
00011 #define HAL_WD_H_
00012
00013 //*****
00014 //           INCLUDES
00015 //*****
00016 #include <stddef.h>
00017 #include <stdint.h>
00018 #include <ti/drivers/Watchdog.h>
00019
00020 //*****
00021 //           FUNCTION PROTOTYPES
00022 //*****
00023 Watchdog_Handle Startup_Watchdog(uint_least8_t index, uint32_t timeout);
00024
00025 #endif /* HAL_WD_H_ */
```

3.65 LDC1000.c File Reference

Functions to read and write data to LDC1000 inductance sensor.

```
#include <unistd.h>
#include <math.h>
#include <ti/drivers/SPI.h>
#include "STARPORTS_App.h"
#include "LDC1000.h"
#include "hal_SPI.h"
#include "hal_UART.h"
```

Functions

- `uint8_t LDC1000_DevId (SPI_Handle spi)`
LDC1000 Get DevID Function.
- `void LDC1000_Get_Proximity_Frame (SPI_Handle spi, uint16_t Samples, int16_t *DataSensor)`
LDC1000 Get Proximity Frame Function.
- `uint32_t LDC1000_Read_Fcount (SPI_Handle spi)`
LDC1000 Read Fcount Function.
- `uint16_t LDC1000_Read_Proximity (SPI_Handle spi)`
LDC1000 Read Proximity Function.
- `uint8_t LDC1000_Read_Rp_Max (SPI_Handle spi)`
LDC1000 Read RP Max Function.
- `uint8_t LDC1000_Read_Rp_Min (SPI_Handle spi)`
LDC1000 Read RP Min Function.
- `uint8_t LDC1000_Read_Status (SPI_Handle spi)`
LDC1000 Read status Function.
- `void LDC1000_Write_Clk_Conf (SPI_Handle spi, uint8_t Val)`
LDC1000 Write clock configuration Function.
- `void LDC1000_Write_Conf (SPI_Handle spi, uint8_t Val)`
LDC1000 Write configuration Function.
- `void LDC1000_Write_Intb_Conf (SPI_Handle spi, uint8_t Val)`
LDC1000 Write Intb configuration Function.
- `void LDC1000_Write_Min_Freq (SPI_Handle spi, uint8_t Val)`
LDC1000 Write Min Freq Function.
- `void LDC1000_Write_Pow_Conf (SPI_Handle spi, uint8_t Val)`
LDC1000 Write Power configuration Function.
- `void LDC1000_Write_Rp_Max (SPI_Handle spi, uint8_t Val)`
LDC1000 Write RP Max Function.
- `void LDC1000_Write_Rp_Min (SPI_Handle spi, uint8_t Val)`
LDC1000 Write RP Min Function.
- `float RpCalc (uint32_t Proximity, float RpMax, float RpMin)`
LDC1000 Rp Calculator Function.

3.65.1 Detailed Description

Functions to read and write data to LDC1000 inductance sensor.

Version

1.0

Date

07/05/2019

Author

A.Irizar @tittle LDC1000 inductance sensor functions

Definition in file [LDC1000.c](#).

3.65.2 Function Documentation

3.65.2.1 LDC1000_DevId() `uint8_t LDC1000_DevId (`
`SPI_Handle spi)`

LDC1000 Get DevID Function.

This function

1. Gets DevID from LDC1000

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

`uint8_t RxBuffer DEV_ID value`

Definition at line 37 of file [LDC1000.c](#).

```
00037
00038
00039     uint8_t RxBuffer[2];
00040
00041     SPI_read_8bits(spi,DEV_ID,RxBuffer,1,RNW_MSB);
00042
00043     return RxBuffer[1];
00044 }
```

3.65.2.2 LDC1000_Get_Proximity_Frame() `void LDC1000_Get_Proximity_Frame (`
`SPI_Handle spi,`
`uint16_t Samples,`
`int16_t * DataSensor)`

LDC1000 Get Proximity Frame Function.

This function

1. Gets Proximity frame from LDC1000 register

Parameters

in	<i>SPI_Handle</i>	spi
in	<i>uint16_t</i>	Samples N samples to read
	<i>int16_t</i>	*DataSensor Pointer to DataSensor to save values

Returns

None

Definition at line 270 of file LDC1000.c.

```

00270
00271
00272     int i=0;
00273     uint32_t newProx, Proximity[512];
00274     float ProxMean;
00275     float ProxRms;
00276     uint8_t status;
00277
00278     ProxMean = 0;
00279     usleep(50000);
00280     while(i<Samples) {
00281         newProx = (uint32_t)LDC1000_Read_Proximity(spi);
00282         ProxMean += newProx;
00283         Proximity[i] = newProx;
00284         i++;
00285         usleep(250);
00286     }
00287     ProxMean = ProxMean/Samples;
00288
00289     ProxRms = 0.0;
00290     for (i=0;i<Samples;i++) {
00291         ProxRms += ( (Proximity[i]-ProxMean)*(Proximity[i]-ProxMean) );
00292     }
00293
00294     DataSensor[0] = (int16_t)ProxMean;
00295     DataSensor[1] = (int16_t)sqrt(ProxRms/Samples);
00296 }
```

3.65.2.3 LDC1000_Read_Fcount() uint32_t LDC1000_Read_Fcount (

SPI_Handle *spi*)

LDC1000 Read Fcount Function.

This function

1. Reads LDC1000 Fcount register

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
----	-------------------	------------

Returns

uint8_t RxBuffer Fcount register value

Definition at line 310 of file LDC1000.c.

```

00310
00311
00312     uint8_t RxBuffer[4];
00313
00314     SPI_read_8bits(spi,FCOUNT_LSB,RxBuffer,3,RNW_MSB);
00315
00316     return ( RxBuffer[1] + (RxBuffer[2]<<8) + (RxBuffer[3]<<16) );
00317 }
```

3.65.2.4 LDC1000_Read_Proximity() `uint16_t LDC1000_Read_Proximity (`
 `SPI_Handle spi)`

LDC1000 Read Proximity Function.

This function

1. Reads LDC1000 Proximity register

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

`uint8_t RxBuffer` Proximity register value

Definition at line 247 of file [LDC1000.c](#).

```
00247
00248
00249     uint8_t RxBuffer[3];
00250
00251     SPI_read_8bits(spi,PROX_LSB,RxBuffer,2,RNW_MSB);
00252
00253     return ( RxBuffer[1] + (uint16_t)(RxBuffer[2]«8) );
00254 }
```

3.65.2.5 LDC1000_Read_Rp_Max() `uint8_t LDC1000_Read_Rp_Max (`
 `SPI_Handle spi)`

LDC1000 Read RP Max Function.

This function

1. Reads LDC1000 RP max register

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

`uint8_t RxBuffer` RP max register value

Definition at line 76 of file [LDC1000.c](#).

```
00076
00077
00078     uint8_t RxBuffer[2];
00079
00080     SPI_read_8bits(spi,RP_MAX,RxBuffer,1,RNW_MSB);
00081
00082     return RxBuffer[1];
00083 }
```

3.65.2.6 LDC1000_Read_Rp_Min() `uint8_t LDC1000_Read_Rp_Min (SPI_Handle spi)`

LDC1000 Read RP Min Function.

This function

1. Reads LDC1000 RP min register

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

`uint8_t RxBuffer RP min register value`

Definition at line 115 of file [LDC1000.c](#).

```
00115                                     {  
00116  
00117     uint8_t RxBuffer[2];  
00118  
00119     SPI_read_8bits(spi,RP_MIN,RxBuffer,1,RNW_MSB);  
00120  
00121     return RxBuffer[1];  
00122 }
```

3.65.2.7 LDC1000_Read_Status() `uint8_t LDC1000_Read_Status (SPI_Handle spi)`

LDC1000 Read status Function.

This function

1. Reads LDC1000 status register

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

`uint8_t RxBuffer status register value`

Definition at line 226 of file [LDC1000.c](#).

```
00226                                     {  
00227  
00228     uint8_t RxBuffer[2];  
00229  
00230     SPI_read_8bits(spi,LDC_STATUS,RxBuffer,1,RNW_MSB);  
00231  
00232     return RxBuffer[1];  
00233 }
```

```
3.65.2.8 LDC1000_Write_Clk_Conf() void LDC1000_Write_Clk_Conf (
    SPI_HandleTypeDef spi,
    uint8_t Val )
```

LDC1000 Write clock configuration Function.

This function

1. Writes clock configuration to LDC1000

Parameters

in	<i>SPI_HandleTypeDef</i>	spi
in	<i>uint8_t</i>	Val

Returns

None

Definition at line 209 of file [LDC1000.c](#).

```
00209
00210
00211     SPI_write_8bits(spi,CLK_CONFIG,Val,RNW_MSB);
00212 }
```

```
3.65.2.9 LDC1000_Write_Conf() void LDC1000_Write_Conf (
    SPI_HandleTypeDef spi,
    uint8_t Val )
```

LDC1000 Write configuration Function.

This function

1. Writes configuration to LDC1000

Parameters

in	<i>SPI_HandleTypeDef</i>	spi
in	<i>uint8_t</i>	Val

Returns

None

Definition at line 155 of file [LDC1000.c](#).

```
00155
00156
00157     SPI_write_8bits(spi,LDC_CONFIG,Val,RNW_MSB);
00158 }
```

```
3.65.2.10 LDC1000_Write_Intb_Conf() void LDC1000_Write_Intb_Conf (
    SPI_Handle spi,
    uint8_t Val )
```

LDC1000 Write Intb configuration Function.

This function

1. Writes Intb configuration to LDC1000

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
in	<i>uint8_t</i>	<i>Val</i>

Returns

None

Definition at line 173 of file [LDC1000.c](#).

```
00173
00174
00175     SPI_write_8bits(spi,INTB_CONFIG,Val,RNW_MSB);
00176 }
```

```
3.65.2.11 LDC1000_Write_Min_Freq() void LDC1000_Write_Min_Freq (
    SPI_Handle spi,
    uint8_t Val )
```

LDC1000 Write Min Freq Function.

This function

1. Configures LDC1000 to min frequency

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
in	<i>uint8_t</i>	<i>Val</i>

Returns

None

Definition at line 137 of file [LDC1000.c](#).

```
00137
00138
00139     SPI_write_8bits(spi,MIN_FREQ,Val,RNW_MSB);
00140 }
```

3.65.2.12 LDC1000_Write_Pow_Conf() void LDC1000_Write_Pow_Conf (SPI_Handle *spi*, uint8_t *Val*)

LDC1000 Write Power configuration Function.

This function

1. Writes Power configuration to LDC1000

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
in	<i>uint8_t</i>	<i>Val</i>

Returns

None

Definition at line 191 of file [LDC1000.c](#).

```
00191
00192
00193     SPI_write_8bits(spi,POW_CONFIG,Val,RNW_MSB);
00194 }
```

3.65.2.13 LDC1000_Write_Rp_Max() void LDC1000_Write_Rp_Max (SPI_Handle *spi*, uint8_t *Val*)

LDC1000 Write RP Max Function.

This function

1. Configures LDC1000 to RP max

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
in	<i>uint8_t</i>	<i>Val</i>

Returns

None

Definition at line 59 of file [LDC1000.c](#).

```
00059
00060
00061     SPI_write_8bits(spi,RP_MAX,Val,RNW_MSB);
00062 }
```

```
3.65.2.14 LDC1000_Write_Rp_Min() void LDC1000_Write_Rp_Min (
    SPI_Handle spi,
    uint8_t Val )
```

LDC1000 Write RP Min Function.

This function

1. Configures LDC1000 to RP min

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
in	<i>uint8_t</i>	<i>Val</i>

Returns

None

Definition at line 98 of file [LDC1000.c](#).

```
00098
00099
00100     SPI_write_8bits(spi,RP_MIN,Val,RNW_MSB);
00101 }
```

```
3.65.2.15 RpCalc() float RpCalc (
    uint32_t Proximity,
    float RpMax,
    float RpMin )
```

LDC1000 Rp Calculator Function.

This function

1. Calculates the RP for LDC1000

Parameters

in	<i>uint32_t</i>	Proximity
in	<i>float</i>	RpMax
in	<i>float</i>	RpMin

Returns

float Rp

Definition at line 333 of file [LDC1000.c](#).

```
00333 }
```

```

00334     float Y = 1.0*Proximity/32768;
00335     float Rp;
00336
00337     Rp = RpMax * RpMin / (RpMin*(1-Y) + RpMax*Y);
00338
00339     return Rp;
00340
00341 }
```

3.66 LDC1000.c

```

00001 //*****
00010 //***** INCLUDES *****
00011 //***** INCLUDES *****
00012 //***** INCLUDES *****
00013 #include <unistd.h>
00014 #include <math.h>
00015 #include <ti/drivers/SPI.h>
00016 #include "STARPORTS_App.h"
00017 #include "LDC1000.h"
00018 #include "hal_SPI.h"
00019 #include "hal_UART.h"
00020
00021 //*****
00022 //***** FUNCTIONS *****
00023 //*****
00024
00025 //*****
00026 //
00027 //
00028 //
00029
00030 //*****
00031 //***** FUNCTIONS *****
00032 //*****
00033 //
00034
00035 //
00036 //*****
00037 uint8_t LDC1000_DevId(SPI_Handle spi) {
00038
00039     uint8_t RxBuffer[2];
00040
00041     SPI_read_8bits(spi,DEV_ID,RxBuffer,1,RNW_MSB);
00042
00043     return RxBuffer[1];
00044 }
00045
00046 //*****
00047 //
00048 //
00049 //*****
00050 void LDC1000_Write_Rp_Max(SPI_Handle spi, uint8_t Val) {
00051
00052     SPI_write_8bits(spi,RP_MAX,Val,RNW_MSB);
00053 }
00054
00055 //*****
00056 //
00057 //
00058 //*****
00059 void LDC1000_Read_Rp_Max(SPI_Handle spi) {
00060
00061     uint8_t RxBuffer[2];
00062
00063     SPI_read_8bits(spi,RP_MAX,RxBuffer,1,RNW_MSB);
00064
00065 //
00066 //
00067 //*****
00068 uint8_t LDC1000_Read_Rp_Min(SPI_Handle spi) {
00069
00070     uint8_t RxBuffer[2];
00071
00072     SPI_read_8bits(spi,RP_MIN,RxBuffer,1,RNW_MSB);
00073
00074     return RxBuffer[1];
00075 }
00076
00077
00078 //*****
00079 //
00080 //*****
00081 //
00082 //
00083 }
00084
00085 //*****
00086 //
00087 //
00088 //*****
00089 void LDC1000_Write_Rp_Min(SPI_Handle spi, uint8_t Val) {
00090
00091     SPI_write_8bits(spi,RP_MIN,Val,RNW_MSB);
00092 }
00093
00094 //*****
00095 //
00096 //
00097 //*****
00098 void LDC1000_Read_Rp_Min(SPI_Handle spi) {
00099
00100     uint8_t RxBuffer[2];
00101
00102     SPI_read_8bits(spi,RP_MIN,RxBuffer,1,RNW_MSB);
00103
00104 //
00105 //
00106 //*****
00107 uint8_t LDC1000_Read_Rp_Min(SPI_Handle spi) {
00108
00109     uint8_t RxBuffer[2];
00110
00111     SPI_read_8bits(spi,RP_MIN,RxBuffer,1,RNW_MSB);
00112
00113     return RxBuffer[1];
00114 }
00115
00116
00117
00118
00119
00120
00121
00122 }
00123
```

```

00124 //*****
00125 //
00135 //
00136 //*****
00137 void LDC1000_Write_Min_Freq(SPI_HandleTypeDef spi, uint8_t Val) {
00138     SPI_write_8bits(spi,MIN_FREQ,Val,RNW_MSB);
00140 }
00141 //
00142 //*****
00143 //
00153 //
00154 //*****
00155 void LDC1000_Write_Conf(SPI_HandleTypeDef spi, uint8_t Val) {
00156     SPI_write_8bits(spi,LDC_CONFIG,Val,RNW_MSB);
00158 }
00159 //
00160 //*****
00161 //
00171 //
00172 //*****
00173 void LDC1000_Write_Intb_Conf(SPI_HandleTypeDef spi, uint8_t Val) {
00174     SPI_write_8bits(spi,INTB_CONFIG,Val,RNW_MSB);
00176 }
00177 //
00178 //*****
00179 //
00189 //
00190 //*****
00191 void LDC1000_Write_Pow_Conf(SPI_HandleTypeDef spi, uint8_t Val) {
00192     SPI_write_8bits(spi,POW_CONFIG,Val,RNW_MSB);
00194 }
00195 //
00196 //*****
00197 //
00207 //
00208 //*****
00209 void LDC1000_Write_Clk_Conf(SPI_HandleTypeDef spi, uint8_t Val) {
00210     SPI_write_8bits(spi,CLK_CONFIG,Val,RNW_MSB);
00212 }
00213 //
00214 //*****
00215 //
00224 //
00225 //*****
00226 uint8_t LDC1000_Read_Status(SPI_HandleTypeDef spi) {
00227     uint8_t RxBuffer[2];
00229     SPI_read_8bits(spi,LDC_STATUS,RxBuffer,1,RNW_MSB);
00231     return RxBuffer[1];
00233 }
00234 //
00235 //*****
00236 //
00245 //
00246 //*****
00247 uint16_t LDC1000_Read_Proximity(SPI_HandleTypeDef spi) {
00248     uint8_t RxBuffer[3];
00250     SPI_read_8bits(spi,PROX_LSB,RxBuffer,2,RNW_MSB);
00252     return ( RxBuffer[1] + (uint16_t)(RxBuffer[2]<<8) );
00254 }
00255 //
00256 //*****
00257 //
00268 //
00269 //*****
00270 void LDC1000_Get_Proximity_Frame(SPI_HandleTypeDef spi, uint16_t Samples, int16_t *DataSensor) {
00271     int i=0;
00273     uint32_t newProx, Proximity[512];
00274     float ProxMean;
00275     float ProxRms;
00276     uint8_t status;
00277     ProxMean = 0;
00279     usleep(50000);
00280     while(i<Samples) {
00281         newProx = (uint32_t)LDC1000_Read_Proximity(spi);

```

```

00282     ProxMean += newProx;
00283     Proximity[i] = newProx;
00284     i++;
00285     usleep(250);
00286 }
00287 ProxMean = ProxMean/Samples;
00288
00289 ProxRms = 0.0;
00290 for (i=0;i<Samples;i++) {
00291     ProxRms += ( (Proximity[i]-ProxMean)*(Proximity[i]-ProxMean) );
00292 }
00293
00294 DataSensor[0] = (int16_t)ProxMean;
00295 DataSensor[1] = (int16_t)sqrt(ProxRms/Samples);
00296 }
00297
00298 //*****
00299 //
00308 //
00309 //*****
00310 uint32_t LDC1000_Read_Fcount(SPI_Handle spi) {
00311
00312     uint8_t RxBuffer[4];
00313
00314     SPI_read_8bits(spi,FCOUNT_LSB,RxBuffer,3,RNW_MSB);
00315
00316     return ( RxBuffer[1] + (RxBuffer[2]<<8) + (RxBuffer[3]<<16) );
00317 }
00318
00319 //*****
00320 //
00331 //
00332 //*****
00333 float RpCalc(uint32_t Proximity, float RpMax, float RpMin) {
00334
00335     float Y = 1.0*Proximity/32768;
00336     float Rp;
00337
00338     Rp = RpMax * RpMin / (RpMin*(1-Y) + RpMax*Y);
00339
00340     return Rp;
00341 }

```

3.67 LDC1000.h File Reference

Functions to read and write data to LDC1000 inductance sensor.

```
#include <stdbool.h>
#include <ti/drivers/SPI.h>
#include <ti/drivers/I2C.h>
```

Enumerations

- enum `LDC1000_amplitude_t` { `AMP_1V` = 0x00, `AMP_2V` = (0x01<<3), `AMP_4V` = (0x02<<3) }
LDC1000 CONFIG REGISTERS.
- enum `LDC1000_clkpd_t` { `CLK_ON` = 0x00, `CLK_OFF` = 0x01 }
LDC1000 Clock enum.
- enum `LDC1000_clksel_t` { `XIN` = 0x00, `XIN_XOUT` = 0x02 }
LDC1000 clock selection enum.
- enum `LDC1000_intb_conf_t` { `INTB_DIS` = 0x00, `WAKEUP_EN` = 0x01, `COMP_OUT` = 0x02, `DRDY_EN` = 0x04 }
LDC1000 Internal configuration enum.
- enum `LDC1000_pow_conf_t` { `ACTIVE_MODE` = 0x01, `STBY_MODE` = 0x00 }
LDC1000 Power configuration enum.

- enum `LDC1000_register_t`{
 `DEV_ID` = 0x00, `RP_MAX` = 0x01, `RP_MIN` = 0x02, `MIN_FREQ` = 0x03,
 `LDC_CONFIG` = 0x04, `CLK_CONFIG` = 0x05, `THR_HIGH_LSB` = 0x06, `THR_HIGH_MSB` = 0x07,
 `THR_LOW_LSB` = 0x08, `THR_LOW_MSB` = 0x09, `INTB_CONFIG` = 0x0A, `POW_CONFIG` = 0x0B,
 `LDC_STATUS` = 0x20, `PROX_LSB` = 0x21, `PROX_MSB` = 0x22, `FCOUNT_LSB` = 0x23,
 `FCOUNT_XSB` = 0x24, `FCOUNT_MSB` = 0x25 }
- LDC1000 registers.*
- enum `LDC1000_resp_time_t`{
 `RESP_TIME_0192` = 0x02, `RESP_TIME_0384` = 0x03, `RESP_TIME_0768` = 0x04, `RESP_TIME_1536` = 0x05,
 `RESP_TIME_3072` = 0x06, `RESP_TIME_6144` = 0x07 }
- LDC1000 Response time enum.*
- enum `LDC1000_rp_max_t`{
 `RPMAX3936` = 0x00, `RPMAX3141` = 0x01, `RPMAX2243` = 0x02, `RPMAX1745` = 0x03,
 `RPMAX1308` = 0x04, `RPMAX0981` = 0x05, `RPMAX0747` = 0x06, `RPMAX0581` = 0x07,
 `RPMAX0436` = 0x08, `RPMAX0349` = 0x09, `RPMAX0249` = 0x0a, `RPMAX0193` = 0x0b,
 `RPMAX0145` = 0x0c, `RPMAX0109` = 0x0d, `RPMAX0083` = 0x0e, `RPMAX0064` = 0x0f,
 `RPMAX0048` = 0x10, `RPMAX0038` = 0x11, `RPMAX0027` = 0x12, `RPMAX0021` = 0x13,
 `RPMAX0016` = 0x14, `RPMAX0012` = 0x15, `RPMAX0009` = 0x16, `RPMAX0007` = 0x17,
 `RPMAX0005` = 0x18, `RPMAX0004` = 0x19, `RPMAX0003` = 0x1a, `RPMAX0002` = 0x1b,
 `RPMAX0001p7` = 0x1c, `RPMAX0001p3` = 0x1d, `RPMAX0001p0` = 0x1e, `RPMAX0000` = 0x1f }
- LDC1000 RP_MAX REGISTERS.*
- enum `LDC1000_rp_min_t`{
 `RPMIN3926` = 0x20, `RPMIN3141` = 0x21, `RPMIN2243` = 0x22, `RPMIN1745` = 0x23,
 `RPMIN1308` = 0x24, `RPMIN0981` = 0x25, `RPMIN0747` = 0x26, `RPMIN0581` = 0x27,
 `RPMIN0436` = 0x28, `RPMIN0349` = 0x29, `RPMIN0249` = 0x2a, `RPMIN0193` = 0x2b,
 `RPMIN0145`

- LDC1000 Write clock configuration Function.*
- void [LDC1000_Write_Conf](#) (SPI_Handle *spi*, uint8_t *Val*)
LDC1000 Write configuration Function.
 - void [LDC1000_Write_Intb_Conf](#) (SPI_Handle *spi*, uint8_t *Val*)
LDC1000 Write Intb configuration Function.
 - void [LDC1000_Write_Min_Freq](#) (SPI_Handle *spi*, uint8_t *Val*)
LDC1000 Write Min Freq Function.
 - void [LDC1000_Write_Pow_Conf](#) (SPI_Handle *spi*, uint8_t *Val*)
LDC1000 Write Power configuration Function.
 - void [LDC1000_Write_Rp_Max](#) (SPI_Handle *spi*, uint8_t *Val*)
LDC1000 Write RP Max Function.
 - void [LDC1000_Write_Rp_Min](#) (SPI_Handle *spi*, uint8_t *Val*)
LDC1000 Write RP Min Function.
 - float [RpCalc](#) (uint32_t *Proximity*, float *RpMax*, float *RpMin*)
LDC1000 Rp Calculator Function.

3.67.1 Detailed Description

Functions to read and write data to LDC1000 inductance sensor.

Version

1.0

Date

07/05/2019

Author

A.Irizar @tittle LDC1000 inductance sensor functions

Definition in file [LDC1000.h](#).

3.67.2 Enumeration Type Documentation

3.67.2.1 [LDC1000_amplitude_t](#) enum [LDC1000_amplitude_t](#)

LDC1000 CONFIG REGISTERS.

Enumerator

AMP_1V	Configuration reg
AMP_2V	Configuration reg
AMP_4V	Configuration reg

Definition at line 127 of file [LDC1000.h](#).

```
00127      {
00128      AMP_1V = 0x00,
00129      AMP_2V = (0x01«3),
00130      AMP_4V = (0x02«3)
00131 } LDC1000_amplitude_t;
```

3.67.2.2 LDC1000_clkpd_t enum [LDC1000_clkpd_t](#)

LDC1000 Clock enum.

Enumerator

CLK_ON	Clock
CLK_OFF	Clock

Definition at line 156 of file [LDC1000.h](#).

```
00156      {
00157      CLK_ON = 0x00,
00158      CLK_OFF = 0x01,
00159 } LDC1000_clkpd_t;
```

3.67.2.3 LDC1000_clksel_t enum [LDC1000_clksel_t](#)

LDC1000 clock selection enum.

Enumerator

XIN	clock selection
XIN_XOUT	clock selection

Definition at line 148 of file [LDC1000.h](#).

```
00148      {
00149      XIN = 0x00,
00150      XIN_XOUT = 0x02,
00151 } LDC1000_clksel_t;
```

3.67.2.4 LDC1000_intb_conf_t enum [LDC1000_intb_conf_t](#)

LDC1000 Internal configuration enum.

Enumerator

INTB_DIS	INTB DIS
WAKEUP_EN	WAKEUP EN
COMP_OUT	COMP OUT
DRDY_EN	DRDY EN

Definition at line 164 of file [LDC1000.h](#).

```
00164      {
00165      INTB_DIS = 0x00,
00166      WAKEUP_EN = 0x01,
00167      COMP_OUT = 0x02,
00168      DRDY_EN = 0x04
00169 } LDC1000_intb_conf_t;
```

3.67.2.5 LDC1000_pow_conf_t enum LDC1000_pow_conf_t

LDC1000 Power configuration enum.

Enumerator

ACTIVE_MODE	ACTIVE MODE
STBY_MODE	STBY MODE

Definition at line 174 of file [LDC1000.h](#).

```
00174      {
00175      ACTIVE_MODE = 0x01,
00176      STBY_MODE = 0x00
00177 } LDC1000_pow_conf_t;
```

3.67.2.6 LDC1000_register_t enum LDC1000_register_t

LDC1000 registers.

Enumerator

DEV_ID	DEV ID reg
RP_MAX	RP MAX reg
RP_MIN	RP MIN reg
MIN_FREQ	MIN FREQ reg
LDC_CONFIG	LDC CONFIG reg
CLK_CONFIG	CLK CONFIG reg
THR_HIGH_LSB	THR HIGH LSB reg
THR_HIGH_MSB	THR HIGH MSB reg
THR_LOW_LSB	THR LOW LSB reg
THR_LOW_MSB	THR LOW MSB reg
INTB_CONFIG	INTB CONFIG reg
POW_CONFIG	POW CONFIG reg
LDC_STATUS	LDC STATUS reg
PROX_LSB	PROX LSB reg
PROX_MSB	PROX MSB reg
FCOUNT_LSB	FCOUNT LSB reg
FCOUNT_XSB	FCOUNT XSB reg
FCOUNT_MSB	FCOUNT MSB reg

Definition at line 27 of file [LDC1000.h](#).

```

00027      {
00028      DEV_ID = 0x00,
00029      RP_MAX = 0x01,
00030      RP_MIN = 0x02,
00031      MIN_FREQ = 0x03,
00032      LDC_CONFIG = 0x04,
00033      CLK_CONFIG = 0x05,
00034      THR_HIGH_LSB = 0x06,
00035      THR_HIGH_MSB = 0x07,
00036      THR_LOW_LSB = 0x08,
00037      THR_LOW_MSB = 0x09,
00038      INTB_CONFIG = 0x0A,
00039      POW_CONFIG = 0x0B,
00040      LDC_STATUS = 0x20,
00041      PROX_LSB = 0x21,
00042      PROX_MSB = 0x22,
00043      FCOUNT_LSB = 0x23,
00044      FCOUNT_XSB = 0x24,
00045      FCOUNT_MSB = 0x25
00046 } LDC1000_register_t;

```

3.67.2.7 LDC1000_resp_time_t enum LDC1000_resp_time_t

LDC1000 Response time enum.

Enumerator

RESP_TIME_0192	RESP TIME 0192
RESP_TIME_0384	RESP TIME 0384
RESP_TIME_0768	RESP TIME 0768
RESP_TIME_1536	RESP TIME 0768
RESP_TIME_3072	RESP TIME 0768
RESP_TIME_6144	RESP TIME 0768

Definition at line 136 of file [LDC1000.h](#).

```

00136      {
00137      RESP_TIME_0192 = 0x02,
00138      RESP_TIME_0384 = 0x03,
00139      RESP_TIME_0768 = 0x04,
00140      RESP_TIME_1536 = 0x05,
00141      RESP_TIME_3072 = 0x06,
00142      RESP_TIME_6144 = 0x07
00143 } LDC1000_resp_time_t;

```

3.67.2.8 LDC1000_rp_max_t enum LDC1000_rp_max_t

LDC1000 RP_MAX REGISTERS.

Enumerator

RPMAX3936	RP MAX reg
RPMAX3141	RP MAX reg
RPMAX2243	RP MAX reg
RPMAX1745	RP MAX reg
RPMAX1308	RP MAX reg
RPMAX0981	RP MAX reg
RPMAX0747	RP MAX reg

Enumerator

RPMAX0581	RP MAX reg
RPMAX0436	RP MAX reg
RPMAX0349	RP MAX reg
RPMAX0249	RP MAX reg
RPMAX0193	RP MAX reg
RPMAX0145	RP MAX reg
RPMAX0109	RP MAX reg
RPMAX0083	RP MAX reg
RPMAX0064	RP MAX reg
RPMAX0048	RP MAX reg
RPMAX0038	RP MAX reg
RPMAX0027	RP MAX reg
RPMAX0021	RP MAX reg
RPMAX0016	RP MAX reg
RPMAX0012	RP MAX reg
RPMAX0009	RP MAX reg
RPMAX0007	RP MAX reg
RPMAX0005	RP MAX reg
RPMAX0004	RP MAX reg
RPMAX0003	RP MAX reg
RPMAX0002	RP MAX reg
RPMAX0001p7	RP MAX reg
RPMAX0001p3	RP MAX reg
RPMAX0001p0	RP MAX reg
RPMAX0000	RP MAX reg

Definition at line 51 of file [LDC1000.h](#).

```

00051      {
00052      RPMAX3936 = 0x00,
00053      RPMAX3141 = 0x01,
00054      RPMAX2243 = 0x02,
00055      RPMAX1745 = 0x03,
00056      RPMAX1308 = 0x04,
00057      RPMAX0981 = 0x05,
00058      RPMAX0747 = 0x06,
00059      RPMAX0581 = 0x07,
00060      RPMAX0436 = 0x08,
00061      RPMAX0349 = 0x09,
00062      RPMAX0249 = 0x0a,
00063      RPMAX0193 = 0x0b,
00064      RPMAX0145 = 0x0c,
00065      RPMAX0109 = 0x0d,
00066      RPMAX0083 = 0x0e,
00067      RPMAX0064 = 0x0f,
00068      RPMAX0048 = 0x10,
00069      RPMAX0038 = 0x11,
00070      RPMAX0027 = 0x12,
00071      RPMAX0021 = 0x13,
00072      RPMAX0016 = 0x14,
00073      RPMAX0012 = 0x15,
00074      RPMAX0009 = 0x16,
00075      RPMAX0007 = 0x17,
00076      RPMAX0005 = 0x18,
00077      RPMAX0004 = 0x19,
00078      RPMAX0003 = 0x1a,
00079      RPMAX0002 = 0x1b,
00080      RPMAX0001p7 = 0x1c,
00081      RPMAX0001p3 = 0x1d,
00082      RPMAX0001p0 = 0x1e,
00083      RPMAX0000 = 0x1f
00084 } LDC1000_rp_max_t;

```

3.67.2.9 LDC1000_rp_min_t enum [LDC1000_rp_min_t](#)

LDC1000 RP_MIN REGISTERS.

Enumerator

RPMIN3926	RP MIN reg
RPMIN3141	RP MIN reg
RPMIN2243	RP MIN reg
RPMIN1745	RP MIN reg
RPMIN1308	RP MIN reg
RPMIN0981	RP MIN reg
RPMIN0747	RP MIN reg
RPMIN0581	RP MIN reg
RPMIN0436	RP MIN reg
RPMIN0349	RP MIN reg
RPMIN0249	RP MIN reg
RPMIN0193	RP MIN reg
RPMIN0145	RP MIN reg
RPMIN0109	RP MIN reg
RPMIN0083	RP MIN reg
RPMIN0064	RP MIN reg
RPMIN0048	RP MIN reg
RPMIN0038	RP MIN reg
RPMIN0027	RP MIN reg
RPMIN0021	RP MIN reg
RPMIN0016	RP MIN reg
RPMIN0012	RP MIN reg
RPMIN0009	RP MIN reg
RPMIN0007	RP MIN reg
RPMIN0005	RP MIN reg
RPMIN0004	RP MIN reg
RPMIN0003	RP MIN reg
RPMIN0002	RP MIN reg
RPMIN0001p7	RP MIN reg
RPMIN0001p3	RP MIN reg
RPMIN0001p0	RP MIN reg
RPMIN0000	RP MIN reg

Definition at line 89 of file [LDC1000.h](#).

```
00089 {
00090     RPMIN3926 = 0x20,
00091     RPMIN3141 = 0x21,
00092     RPMIN2243 = 0x22,
00093     RPMIN1745 = 0x23,
00094     RPMIN1308 = 0x24,
00095     RPMIN0981 = 0x25,
00096     RPMIN0747 = 0x26,
00097     RPMIN0581 = 0x27,
00098     RPMIN0436 = 0x28,
00099     RPMIN0349 = 0x29,
00100     RPMIN0249 = 0x2a,
00101     RPMIN0193 = 0x2b,
00102     RPMIN0145 = 0x2c,
00103     RPMIN0109 = 0x2d,
00104     RPMIN0083 = 0x2e,
```

```

00105     RPMIN0064 = 0x2f,
00106     RPMIN0048 = 0x30,
00107     RPMIN0038 = 0x31,
00108     RPMIN0027 = 0x32,
00109     RPMIN0021 = 0x33,
00110     RPMIN0016 = 0x34,
00111     RPMIN0012 = 0x35,
00112     RPMIN0009 = 0x36,
00113     RPMIN0007 = 0x37,
00114     RPMIN0005 = 0x38,
00115     RPMIN0004 = 0x39,
00116     RPMIN0003 = 0x3a,
00117     RPMIN0002 = 0x3b,
00118     RPMIN0001p7 = 0x3c,
00119     RPMIN0001p3 = 0x3d,
00120     RPMIN0001p0 = 0x3e,
00121     RPMIN0000 = 0x3f
00122 } LDC1000_rp_min_t;

```

3.67.2.10 LDC1000_status_t enum LDC1000_status_t

LDC1000 Status enum.

Enumerator

OSC_STATUS_MASK	OSC STATUS MASK
DRDYB_MASK	DRDYB MASK
WAKEUP_DIS_MASK	WAKEUP DIS MASK
COMPB_MASK	COMPB MASK

Definition at line 182 of file [LDC1000.h](#).

```

00182     {
00183     OSC_STATUS_MASK = 0x80,
00184     DRDYB_MASK = 0x40,
00185     WAKEUP_DIS_MASK = 0x20,
00186     COMPB_MASK = 0x10
00187 } LDC1000_status_t;

```

3.67.3 Function Documentation

3.67.3.1 LDC1000_DevId() uint8_t LDC1000_DevId (SPI_HandleTypeDef spi)

LDC1000 Get DevID Function.

This function

1. Gets DevID from LDC1000

Parameters

in	SPI_HandleTypeDef	spi
----	-------------------	-----

Returns

```
uint8_t RxBuffer DEV_ID value
```

Definition at line 37 of file [LDC1000.c](#).

```
00037
00038
00039     uint8_t RxBuffer[2];
00040
00041     SPI_read_8bits(spi,DEV_ID,RxBuffer,1,RNW_MSB);
00042
00043     return RxBuffer[1];
00044 }
```

3.67.3.2 LDC1000_Get_Proximity_Frame() void LDC1000_Get_Proximity_Frame (

```
SPI_Handle spi,
uint16_t Samples,
int16_t * DataSensor )
```

LDC1000 Get Proximity Frame Function.

This function

1. Gets Proximity frame from LDC1000 register

Parameters

in	<i>SPI_Handle</i>	spi
in	<i>uint16_t</i>	Samples N samples to read
	<i>int16_t</i>	*DataSensor Pointer to DataSensor to save values

Returns

None

Definition at line 270 of file [LDC1000.c](#).

```
00270
00271
00272     int i=0;
00273     uint32_t newProx, Proximity[512];
00274     float ProxMean;
00275     float ProxRms;
00276     uint8_t status;
00277
00278     ProxMean = 0;
00279     usleep(50000);
00280     while(i<Samples) {
00281         newProx = (uint32_t)LDC1000_Read_Proximity(spi);
00282         ProxMean += newProx;
00283         Proximity[i] = newProx;
00284         i++;
00285         usleep(250);
00286     }
00287     ProxMean = ProxMean/Samples;
00288
00289     ProxRms = 0.0;
00290     for (i=0;i<Samples;i++) {
00291         ProxRms += ( (Proximity[i]-ProxMean)*(Proximity[i]-ProxMean) );
00292     }
00293
00294     DataSensor[0] = (int16_t)ProxMean;
00295     DataSensor[1] = (int16_t)sqrt(ProxRms/Samples);
00296 }
```

3.67.3.3 LDC1000_Read_Fcount() `uint32_t LDC1000_Read_Fcount (`
 `SPI_Handle spi)`

LDC1000 Read Fcount Function.

This function

1. Reads LDC1000 Fcount register

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

`uint8_t RxBuffer Fcount register value`

Definition at line 310 of file [LDC1000.c](#).

```
00310                                     {  
00311  
00312     uint8_t RxBuffer[4];  
00313  
00314     SPI_read_8bits(spi,FCOUNT_LSB,RxBuffer,3,RNW_MSB);  
00315  
00316     return ( RxBuffer[1] + (RxBuffer[2]«8) + (RxBuffer[3]«16) );  
00317 }
```

3.67.3.4 LDC1000_Read_Proximity() `uint16_t LDC1000_Read_Proximity (`
 `SPI_Handle spi)`

LDC1000 Read Proximity Function.

This function

1. Reads LDC1000 Proximity register

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

`uint8_t RxBuffer Proximity register value`

Definition at line 247 of file [LDC1000.c](#).

```
00247                                     {  
00248  
00249     uint8_t RxBuffer[3];  
00250  
00251     SPI_read_8bits(spi,PROX_LSB,RxBuffer,2,RNW_MSB);  
00252  
00253     return ( RxBuffer[1] + (uint16_t)(RxBuffer[2]«8) );  
00254 }
```

3.67.3.5 LDC1000_Read_Rp_Max() `uint8_t LDC1000_Read_Rp_Max (`
 `SPI_Handle spi)`

LDC1000 Read RP Max Function.

This function

1. Reads LDC1000 RP max register

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

`uint8_t RxBuffer RP max register value`

Definition at line 76 of file [LDC1000.c](#).

```
00076                                     {  
00077  
00078     uint8_t RxBuffer[2];  
00079  
00080     SPI_read_8bits(spi,RP_MAX,RxBuffer,1,RNW_MSB);  
00081  
00082     return RxBuffer[1];  
00083 }
```

3.67.3.6 LDC1000_Read_Rp_Min() `uint8_t LDC1000_Read_Rp_Min (`
 `SPI_Handle spi)`

LDC1000 Read RP Min Function.

This function

1. Reads LDC1000 RP min register

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

`uint8_t RxBuffer RP min register value`

Definition at line 115 of file [LDC1000.c](#).

```
00115                                     {  
00116  
00117     uint8_t RxBuffer[2];  
00118  
00119     SPI_read_8bits(spi,RP_MIN,RxBuffer,1,RNW_MSB);  
00120  
00121     return RxBuffer[1];  
00122 }
```

3.67.3.7 LDC1000_Read_Status() `uint8_t LDC1000_Read_Status (`
 `SPI_Handle spi)`

LDC1000 Read status Function.

This function

1. Reads LDC1000 status register

Parameters

in	<i>SPI_Handle</i>	spi
----	-------------------	-----

Returns

`uint8_t RxBuffer status register value`

Definition at line 226 of file [LDC1000.c](#).

```
00226                                     {  
00227  
00228     uint8_t RxBuffer[2];  
00229  
00230     SPI_read_8bits(spi,LDC_STATUS,RxBuffer,1,RNW_MSB);  
00231  
00232     return RxBuffer[1];  
00233 }
```

3.67.3.8 LDC1000_Write_Clk_Conf() `void LDC1000_Write_Clk_Conf (`
 `SPI_Handle spi,`
 `uint8_t Val)`

LDC1000 Write clock configuration Function.

This function

1. Writes clock configuration to LDC1000

Parameters

in	<i>SPI_Handle</i>	spi
in	<i>uint8_t</i>	Val

Returns

None

Definition at line 209 of file [LDC1000.c](#).

```
00209                                     {  
00210  
00211     SPI_write_8bits(spi,CLK_CONFIG,Val,RNW_MSB);  
00212 }
```

```
3.67.3.9 LDC1000_Write_Conf() void LDC1000_Write_Conf (
    SPI_Handle spi,
    uint8_t Val )
```

LDC1000 Write configuration Function.

This function

1. Writes configuration to LDC1000

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
in	<i>uint8_t</i>	<i>Val</i>

Returns

None

Definition at line 155 of file [LDC1000.c](#).

```
00155
00156
00157     SPI_write_8bits(spi,LDC_CONFIG,Val,RNW_MSB);
00158 }
```

```
3.67.3.10 LDC1000_Write_Intb_Conf() void LDC1000_Write_Intb_Conf (
    SPI_Handle spi,
    uint8_t Val )
```

LDC1000 Write Intb configuration Function.

This function

1. Writes Intb configuration to LDC1000

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
in	<i>uint8_t</i>	<i>Val</i>

Returns

None

Definition at line 173 of file [LDC1000.c](#).

```
00173
00174
00175     SPI_write_8bits(spi,INTB_CONFIG,Val,RNW_MSB);
00176 }
```

```
3.67.3.11 LDC1000_Write_Min_Freq() void LDC1000_Write_Min_Freq (
    SPI_Handle spi,
    uint8_t Val )
```

LDC1000 Write Min Freq Function.

This function

1. Configures LDC1000 to min frequency

Parameters

in	<i>SPI_Handle</i>	spi
in	<i>uint8_t</i>	Val

Returns

None

Definition at line 137 of file [LDC1000.c](#).

```
00137
00138
00139     SPI_write_8bits(spi,MIN_FREQ,Val,RNW_MSB);
00140 }
```

```
3.67.3.12 LDC1000_Write_Pow_Conf() void LDC1000_Write_Pow_Conf (
    SPI_Handle spi,
    uint8_t Val )
```

LDC1000 Write Power configuration Function.

This function

1. Writes Power configuration to LDC1000

Parameters

in	<i>SPI_Handle</i>	spi
in	<i>uint8_t</i>	Val

Returns

None

Definition at line 191 of file [LDC1000.c](#).

```
00191
00192
00193     SPI_write_8bits(spi,POW_CONFIG,Val,RNW_MSB);
00194 }
```

```
3.67.3.13 LDC1000_Write_Rp_Max() void LDC1000_Write_Rp_Max (
    SPI_Handle spi,
    uint8_t Val )
```

LDC1000 Write RP Max Function.

This function

1. Configures LDC1000 to RP max

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
in	<i>uint8_t</i>	<i>Val</i>

Returns

None

Definition at line 59 of file [LDC1000.c](#).

```
00059
00060
00061     SPI_write_8bits(spi,RP_MAX,Val,RNW_MSB);
00062 }
```

```
3.67.3.14 LDC1000_Write_Rp_Min() void LDC1000_Write_Rp_Min (
    SPI_Handle spi,
    uint8_t Val )
```

LDC1000 Write RP Min Function.

This function

1. Configures LDC1000 to RP min

Parameters

in	<i>SPI_Handle</i>	<i>spi</i>
in	<i>uint8_t</i>	<i>Val</i>

Returns

None

Definition at line 98 of file [LDC1000.c](#).

```
00098
00099
00100     SPI_write_8bits(spi,RP_MIN,Val,RNW_MSB);
00101 }
```

3.67.3.15 RpCalc()

```
float RpCalc (
    uint32_t Proximity,
    float RpMax,
    float RpMin )
```

LDC1000 Rp Calculator Function.

This function

1. Calculates the RP for LDC1000

Parameters

in	<i>uint32_t</i>	Proximity
in	<i>float</i>	RpMax
in	<i>float</i>	RpMin

Returns

float Rp

Definition at line 333 of file [LDC1000.c](#).

```
00333
00334
00335     float Y = 1.0*Proximity/32768;
00336     float Rp;
00337
00338     Rp = RpMax * RpMin / (RpMin*(1-Y) + RpMax*Y);
00339
00340     return Rp;
00341 }
```

3.68 LDC1000.h

```
00001
00010 #ifndef LDC1000_H_
00011 #define LDC1000_H_
00012
00013 //*****
00014 //           INCLUDES
00015 //*****
00016 #include <stdbool.h>
00017 #include <ti/drivers/SPI.h>
00018 #include <ti/drivers/I2C.h>
00019
00020 //*****
00021 //           TYPEDEFS
00022 //*****
00023
00027 typedef enum {
00028     DEV_ID = 0x00,
00029     RP_MAX = 0x01,
00030     RP_MIN = 0x02,
00031     MIN_FREQ = 0x03,
00032     LDC_CONFIG = 0x04,
00033     CLK_CONFIG = 0x05,
00034     THR_HIGH_LSB = 0x06,
00035     THR_HIGH_MSB = 0x07,
00036     THR_LOW_LSB = 0x08,
00037     THR_LOW_MSB = 0x09,
00038     INTB_CONFIG = 0x0A,
00039     POW_CONFIG = 0x0B,
00040     LDC_STATUS = 0x20,
00041     PROX_LSB = 0x21,
00042     PROX_MSB = 0x22,
00043     FCOUNT_LSB = 0x23,
00044     FCOUNT_XSB = 0x24,
```

```
00045     FCOUNT_MSB = 0x25
00046 } LDC1000_register_t;
00047
00051 typedef enum {
00052     RPMAX3936 = 0x00,
00053     RPMAX3141 = 0x01,
00054     RPMAX2243 = 0x02,
00055     RPMAX1745 = 0x03,
00056     RPMAX1308 = 0x04,
00057     RPMAX0981 = 0x05,
00058     RPMAX0747 = 0x06,
00059     RPMAX0581 = 0x07,
00060     RPMAX0436 = 0x08,
00061     RPMAX0349 = 0x09,
00062     RPMAX0249 = 0x0a,
00063     RPMAX0193 = 0x0b,
00064     RPMAX0145 = 0x0c,
00065     RPMAX0109 = 0x0d,
00066     RPMAX0083 = 0x0e,
00067     RPMAX0064 = 0x0f,
00068     RPMAX0048 = 0x10,
00069     RPMAX0038 = 0x11,
00070     RPMAX0027 = 0x12,
00071     RPMAX0021 = 0x13,
00072     RPMAX0016 = 0x14,
00073     RPMAX0012 = 0x15,
00074     RPMAX0009 = 0x16,
00075     RPMAX0007 = 0x17,
00076     RPMAX0005 = 0x18,
00077     RPMAX0004 = 0x19,
00078     RPMAX0003 = 0x1a,
00079     RPMAX0002 = 0x1b,
00080     RPMAX0001p7 = 0x1c,
00081     RPMAX0001p3 = 0x1d,
00082     RPMAX0001p0 = 0x1e,
00083     RPMAX0000 = 0x1f
00084 } LDC1000_rp_max_t;
00085
00089 typedef enum {
00090     RPMIN3926 = 0x20,
00091     RPMIN3141 = 0x21,
00092     RPMIN2243 = 0x22,
00093     RPMIN1745 = 0x23,
00094     RPMIN1308 = 0x24,
00095     RPMIN0981 = 0x25,
00096     RPMIN0747 = 0x26,
00097     RPMIN0581 = 0x27,
00098     RPMIN0436 = 0x28,
00099     RPMIN0349 = 0x29,
00100    RPMIN0249 = 0x2a,
00101    RPMIN0193 = 0x2b,
00102    RPMIN0145 = 0x2c,
00103    RPMIN0109 = 0x2d,
00104    RPMIN0083 = 0x2e,
00105    RPMIN0064 = 0x2f,
00106    RPMIN0048 = 0x30,
00107    RPMIN0038 = 0x31,
00108    RPMIN0027 = 0x32,
00109    RPMIN0021 = 0x33,
00110    RPMIN0016 = 0x34,
00111    RPMIN0012 = 0x35,
00112    RPMIN0009 = 0x36,
00113    RPMIN0007 = 0x37,
00114    RPMIN0005 = 0x38,
00115    RPMIN0004 = 0x39,
00116    RPMIN0003 = 0x3a,
00117    RPMIN0002 = 0x3b,
00118    RPMIN0001p7 = 0x3c,
00119    RPMIN0001p3 = 0x3d,
00120    RPMIN0001p0 = 0x3e,
00121    RPMIN0000 = 0x3f
00122 } LDC1000_rp_min_t;
00123
00127 typedef enum {
00128     AMP_1V = 0x00,
00129     AMP_2V = (0x01<<3),
00130     AMP_4V = (0x02<<3)
00131 } LDC1000_amplitude_t;
00132
00136 typedef enum {
00137     RESP_TIME_0192 = 0x02,
00138     RESP_TIME_0384 = 0x03,
00139     RESP_TIME_0768 = 0x04,
00140     RESP_TIME_1536 = 0x05,
00141     RESP_TIME_3072 = 0x06,
00142     RESP_TIME_6144 = 0x07
00143 } LDC1000_resp_time_t;
```

```

00144
00148 typedef enum {
00149     XIN = 0x00,
00150     XIN_XOUT = 0x02,
00151 } LDC1000_clksel_t;
00152
00156 typedef enum {
00157     CLK_ON = 0x00,
00158     CLK_OFF = 0x01,
00159 } LDC1000_clkpdt;
00160
00164 typedef enum {
00165     INTB_DIS = 0x00,
00166     WAKEUP_EN = 0x01,
00167     COMP_OUT = 0x02,
00168     DRDY_EN = 0x04
00169 } LDC1000_intb_conf_t;
00170
00174 typedef enum {
00175     ACTIVE_MODE = 0x01,
00176     STBY_MODE = 0x00
00177 } LDC1000_pow_conf_t;
00178
00182 typedef enum {
00183     OSC_STATUS_MASK = 0x80,
00184     DRDYB_MASK = 0x40,
00185     WAKEUP_DIS_MASK = 0x20,
00186     COMPB_MASK = 0x10
00187 } LDC1000_status_t;
00188
00189 //*****
00190 //          FUNCTION PROTOTYPES
00191 //*****
00192 uint8_t LDC1000_DevId(SPI_Handle spi);
00193 void LDC1000_Write_Rp_Max(SPI_Handle spi, uint8_t Val);
00194 uint8_t LDC1000_Read_Rp_Max(SPI_Handle spi);
00195 void LDC1000_Write_Rp_Min(SPI_Handle spi, uint8_t Val);
00196 uint8_t LDC1000_Read_Rp_Min(SPI_Handle spi);
00197 void LDC1000_Write_Min_Freq(SPI_Handle spi, uint8_t Val);
00198 void LDC1000_Write_Conf(SPI_Handle spi, uint8_t Val);
00199 void LDC1000_Write_Intb_Conf(SPI_Handle spi, uint8_t Val);
00200 void LDC1000_Write_Clk_Conf(SPI_Handle spi, uint8_t Val);
00201 void LDC1000_Write_Pow_Conf(SPI_Handle spi, uint8_t Val);
00202 uint8_t LDC1000_Read_Status(SPI_Handle spi);
00203 int16_t LDC1000_Read_Proximity(SPI_Handle spi);
00204 void LDC1000_Get_Proximity_Frame(SPI_Handle spi, uint16_t Samples, int16_t *DataSensor);
00205 uint32_t LDC1000_Read_Fcount(SPI_Handle spi);
00206 float RpCalc(uint32_t Proximity, float RpMax, float RpMin);
00207
00208 #endif /* LDC1000_H_ */

```

3.69 main_nortos.c File Reference

Main no Rtos.

```
#include <stdint.h>
#include <stddef.h>
#include <NoRTOS.h>
#include "Board.h"
```

Functions

- int **main** (void)

Main No Rtos Function.

- void * **mainThread** (void *arg0)

3.69.1 Detailed Description

Main no Rtos.

Version

1.0

Date

07/05/2019

Author

A.Irizar @title main no Rtos

Definition in file [main_nortos.c](#).

3.69.2 Function Documentation

3.69.2.1 main()

```
int main (
```

```
    void )
```

Main No Rtos Function.

This function

1. Initiates STARPORTS App

Parameters

in		
----	--	--

Definition at line 35 of file [main_nortos.c](#).

```
00036 {  
00037     Board_init();           // Call driver init functions  
00038     NoRTOS_start();         // Start NoRTOS  
00039     mainThread(NULL);       // Call mainThread function  
00040  
00041     while (1) {}  
00042  
00043 }
```

3.69.2.2 mainThread()

```
void* mainThread (
```

```
    void * arg0 )
```

Definition at line 111 of file [STARPORTS_App.c](#).

```

00112 {
00113     UART_Handle uart1;                                     //uart1 communicates
00114     with RN2483 Lora module                                //struct of parameters
00115     struct LoraNode MyLoraNode;
00116     of the node
00117     uint8_t DataPacket[256];                                //packet with sensors
00118     data
00119     uint8_t DataPacketLen;                                 //length of Datapacket
00120     unsigned char buf[256];
00121     uint8_t ret;
00122     uint8_t sz;
00123     uint8_t NextStep = CONTINUE;                           //controls node OFF
00124     uint8_t mask;                                         //mask to change node
00125     parameters
00126     uint16_t nodeId;                                      //node ID var
00127     int16_t      sockId;                                    //wifi var
00128     int16_t      status = -1;                             //wifi var
00129     GPIO_setConfig(Board_EN_NODE, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_HIGH); //config node enable
00130     GPIO_Node_Enable();                                  //Quicky enable de
00131     node (the jumper can be removed)
00132     Watchdog_init();                                   //Inits watchdog
00133     instance
00134     ret = GPIO_Config();                               //Configure the rest
00135     of GPIO pins
00136     wd = Startup_Watchdog(Board_WATCHDOG0, TIMEOUT_MS); //Open a Watchdog
00137     driver instance
00138     SPI_init();                                       //initiates SPI before
00139     sl_Start;                                         //needed to work with
00140     sl_Start(0, 0, 0);
00141     File System
00142     uart0 = Startup_UART(Board_UART0, 115200);        //initiates uart
00143
00144 #ifdef DEBUG
00145     UART_write(uart0, "\r\nInitiating Test of STARPORTS...\r\n", 35); //debug
00146 #endif
00147
00148     /***** Begin Read Configuration Files *****/
00149     MyLoraNode.PortNoTx = 1;
00150     MyNode.WakeUpInterval = st_readFileWakeUp();          //Get
00151     MyNode.WakeUpInterval --> Read WakeUp_Time File
00152     MyNode.Mode = st_read FileMode();                     //Get MyNode.Mode -->
00153     Read FileMode
00154     MyNode.NCycles = st_read FileModeNCycles();           //Get MyNode.NCycles
00155     --> Read FileMode
00156     MyNode.NBoot = st_read FileModeNBoot();               //Get MyNode.FirstBoot
00157     --> Read FirstBoot File: Yes (1) No (0)
00158     MyNode.NFails = st_read FileModeNFails();             //Get MyNode.NFails
00159     --> Number of failed attempts to Wireless Conn.
00160     MyNode.ChangeWakeUp = st_read FileModeChangeWakeUp(); //Get
00161     MyNode.ChangeWakeUp --> Read change wake up interval or not
00162     //MyNode.NodeId = st_read FileModeNodeId();           //Get MyNodeId
00163     //st_read FileModeSSID(&(MyNode.SSID));              //Get MyNode.SSID[]
00164     --> Read SSID File
00165     /***** Ends Reading Configuration Files *****/
00166
00167     if (MyNode.NFails>=4) {                                //if NFails >= 4
00168 #ifdef DEBUG
00169     UART_PRINT("NFails > 4, SETTING NODE IN WIFI MODE\r\n");
00170 #endif
00171     st_read FileModeSSID(&(MyNode.SSID));
00172     wlanConf();
00173     wlanConnectFromFile(MyNode.SSID);
00174
00175     Host
00176
00177     MyNode.NBoot=0;                                         //set NBoot = 0
00178     writeNBoot(MyNode.NBoot);                            //write to file NBoot
00179     MyNode.NFails=0;                                       //set NFails = 0
00180     writeNFails(MyNode.NFails);                          //write to file NFails
00181 }
00182
00183     /***** Begin Configure Peripherals *****/
00184     RN2483_Clear();                                     //Reset the RN2483
00185     i2c = Startup_I2C(Board_I2C0);                      //I2C interface
00186     started
00187     /***** End Configure Peripherals *****/
00188
00189     /***** Begin Setting Node WakeUp_Time in RTC */
00190     if (DS1374_Read_Ctrl(i2c)==0x06|| MyNode.ChangeWakeUp == 1 ) { //if DS1374 is powerup
00191     or MyNode.ChangeWakeUp == 1
00192     DS1374_Clear_AF(i2c);                                //clear AF register of
00193     DS1374
00194     DS1374_Write_WdAlmb(i2c, MyNode.WakeUpInterval);    //configures the RTC
00195     DS1374_Write_Ctrl(i2c);                            //write Control

```

```

00177     register of DS1374
00178 } else {
00179 powerup
00180     DS1374_Clear_AF(i2c);
00181 DS1374
00182 }
00183
00184 /***** Begin Configuration and Setup Wireless Connectivity *****/
00185     uart1 = Startup_UART(Board_UART1, 57600);
00186 (connected to RN2483)
00187
00188 if (MyNode.Mode==MODE_NORMAL_LORA) {
00189     RN2483_Set();
00190 releases RN2483
00191     sz = GetLine_UART(uart1, buf);
00192 initialization text of RN2483
00193
00194     if (sz==0) {
00195 large
00196     }
00197
00198     MyLoraNode.Upctr = st_readFileUpCntr();
00199     Mac_Set_Upctr(uart1,&MyLoraNode);
00200     MyLoraNode.Dnctr = st_readFileDnCntr();
00201     Mac_Set_Dnctr(uart1,&MyLoraNode);
00202     Mac_Ar_On(uart1);
00203 Retransmit ON
00204     // Mac_Set_Pwridx(uart1, 1);
00205 default is 1, the maximum 14dBm
00206     // Mac_Adr_On(uart1);
00207 datarate ON (useful when using OTAA)
00208
00209     ret = Join_Abp_Lora(uart1);
00210     if (ret==SUCCESS_ABP_LORA){
00211 Success
00212         MyNode.NFails=0;
00213         writeNFails(MyNode.NFails);
00214     } else {
00215 #ifdef DEBUG
00216         UART_write(uarto0,"Join_Abp_Lora() Failed",22);
00217         UART_write(uarto0, "\r\n", 2);
00218 #endif
00219         MyNode.NFails++;
00220         writeNFails(MyNode.NFails);
00221         NextStep=SHUTDOWN;
00222     }
00223     } else if (MyNode.Mode==MODE_NORMAL_WIFI){
00224         st_readFileSSID(&(MyNode.SSID));
00225         ret = wlanConf();
00226         ret = wlanConnectFromFile(MyNode.SSID);
00227 file
00228
00229     if (ret==SUCCESS_CONNECT_WIFI) {
00230 success
00231         MyNode.NFails=0;
00232         writeNFails(MyNode.NFails);
00233         UART_PRINT("\n\rCONN OK!\n\r");
00234     } else {
00235 fail
00236         MyNode.NFails++;
00237         writeNFails(MyNode.NFails);
00238         UART_PRINT("\n\rCONN KO!\n\r");
00239         NextStep=SHUTDOWN;
00240     }
00241 /***** End of Configuration and Setup Wireless Connectivity *****/
00242     if (NextStep==SHUTDOWN) {
00243 shutdown
00244         I2C_close(i2c);
00245         I2C_As_GPIO_Low();
00246 signals low to save power
00247         Node_Disable();
00248     }
00249
00250 /***** Begin Reading Data from Sensors *****/
00251     SPI_CS_Disable();
00252 High
00253     spi = Startup_SPI(Board_SPI_MASTER, 8, 5000000);
00254 at 5 Mbps, 8-bits, CPOL=0, PHA=0
00255     Timer_init();
00256 Timestep
00257     DataPacketLen = GetSensorData(DataPacket);
00258 /***** End Reading Data from Sensors *****/
00259
00260     SPI_close(spi);
00261 related peripherals
00262
00263
00264
00265
00266
00267
00268
00269
00270
00271
00272
00273
00274
00275
00276
00277
00278
00279
00280
00281
00282
00283
00284
00285
00286
00287
00288
00289
00290
00291
00292
00293
00294
00295
00296
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01922
01923
01924
01925
01926
01927

```

```

    the SPI but just puts the CS signal to '0'
00244    not really shutting down (a high value in some SPI signals
00245    active)
00246        LDC1000_SPI_Enable();
00247        ADXL355_SPI_Enable();
00248        BME280_SPI_Enable();
00249        SPI_As_GPIO_Low();
00250        signals to '0' to save power
00251        I2C_Close(i2c);
00252        I2C_As_GPIO_Low();
00253        signals low to save power
00254
00255    MyLoraNode.DataLenTx = Uint8Array2Char(DataPacket, DataPacketLen,
00256    &(MyLoraNode.DataTx)); //initiates data to send
00257
00258 #ifdef DEBUG
00259     UART_Write(uart0, &(MyLoraNode.DataTx), MyLoraNode.DataLenTx); //debug
00260     UART_Write(uart0, "\r\n", 2); //debug
00261 #endif
00262
00263     if (MyNode.Mode==MODE_NORMAL_LORA) { //if Mode = Lora
00264         ret = Tx_Uncnf_Lora(uart1, &MyLoraNode, &mask, &nodeId); //Transmit Data,
00265         several tries?
00266         if (ret==SUCCESS_TX_MAC_TX) { //if tx = success
00267             MyLoraNode.Upctr = Mac_Get_Upctr(uart1); //Get upctr from
00268             RN2483 writeUpCntr(MyLoraNode.Upctr); //write upctr to file
00269             MyLoraNode.Dnctr = Mac_Get_Dnctr(uart1); //Get dnctr from
00270             RN2483 writeDnCntr(MyLoraNode.Dnctr); //write dnctr to file
00271             //writeNFails(0); //if rx = success
00272             } else if (ret == SUCCESS_TX_MAC_RX) { //Get upctr from
00273                 MyLoraNode.Upctr = Mac_Get_Upctr(uart1); //*****SI HAGO ESTE
00274                 RN2483 writeUpCntr(MyLoraNode.Upctr); //Get dnctr from
00275                 WRITE MACHACO EL VALOR DE DOWNLINK PARA ESTE PARAM. //*****SI HAGO ESTE
00276                 MyLoraNode.Dnctr = Mac_Get_Dnctr(uart1); //Get dnctr from
00277                 RN2483 writeDnCntr(MyLoraNode.Dnctr); //Get dnctr from
00278                 WRITE MACHACO EL VALOR DE DOWNLINK PARA ESTE PARAM.
00279                 /*if (nodeId==MyNode.NodeId) { // Write New Configuration Data to Files
00280                     if (mask&0x01!=0) { // writeWakeUp(MyNode.WakeUpInterval);
00281                         }
00282                     if (mask&0x02!=0) { // writeFirstBoot(MyNode.FirstBoot);
00283                         // writeMode(MyNode.Mode);
00284                     }
00285                     if (mask&0x04!=0) { // writeSSID(&(MyNode.SSID));
00286                         }
00287                     if (mask&0x08!=0) { // writeNCycles(MyNode.NCycles);
00288                         }
00289                     if (mask&0x10!=0) { // writeUpCntr(MyLoraNode.Upctr);
00290                         }
00291                     }*/
00292                 } //*/ //Nfails = 0 and write
00293                 //writeNFails(0); //if tx = fail
00294             to Nfails
00295             } else {
00296 #ifdef DEBUG
00297                 UART_Write(uart0, "Tx_Uncnf_Lora() Failed", 22); //debug
00298                 UART_Write(uart0, "\r\n", 2); //debug
00299                 //MyNode.Nfails++;
00300                 //writeNFails(MyNode.Nfails); //Nfails ++
00301             }
00302         }
00303         else if (MyNode.Mode==MODE_NORMAL_WIFI) { //if Mode = wifi
00304             prepareDataFrame(PORT, DEST_IP_ADDR); //prepare destination
00305             frame sockId = sl_Socket(SL_AF_INET, SL_SOCKET_DGRAM, 0); //creates a socket
00306             if(sockId < 0){ //if socket = fail
00307                 UART_PRINT("error UDP %d\r\n",sockId); //debug
00308             }
00309             status = sl_SendTo(sockId, MyLoraNode.DataTx, MyLoraNode.DataLenTx, 0, (SlSockAddr_t *) &PowerMeasure_CB.ipV4Addr, sizeof(SlSockAddrIn_t)); //send data
00310             if(status < 0){ //if send = fail
00311                 status = sl_Close(sockId); //close socket
00312                 ASSERT_ON_ERROR(sockId); //debug error
00313                 UART_PRINT("\r\nERROR SENDING PACKET: %s\r\n", status); //debug error
00314                 Node_Disable(); //Auto Shutdown
00315             } else{ //if send = success

```

```

00316     status = sl_Close(sockId);                                //close socket
00317     UART_PRINT("PACKET SEND: %s\n\r", MyLoraNode.DataTx);    //debug ok
00318 }
00319 }
00320
00321     writeNBoot(1-MyNode.NBoot);                            //change NBoot and
00322     write to NBoot file
00323 #ifdef DEBUG
00324     UART_write(uart0, "End of STARPORTS Measures\r\n", 27); //debug
00325 #endif
00326
00327     UART_close(uart1);                                     //close uart1
00328     UART_close(uart0);                                     //close uart0
00329     Node_Disable();                                       //Auto shutdown
00330 }
```

3.70 main_nortos.c

```

00001 //*****
00010 //***** INCLUDES *****
00011 //***** INCLUDES *****
00012 //***** INCLUDES *****
00013 #include <stdint.h>
00014 #include <stddef.h>
00015 #include <NoRTOS.h>
00016 #include "Board.h"
00017
00018 //*****
00019 //***** GLOBALS *****
00020 //*****
00021 extern void *mainThread(void *arg0);
00022
00023 //*****
00024 //*****
00033 //*****
00034 //*****
00035 int main(void)
00036 {
00037     Board_init();                                         // Call driver init functions
00038     NoRTOS_start();                                      // Start NoRTOS
00039     mainThread(NULL);                                    // Call mainThread function
00040
00041     while (1) {};
00042
00043 }
```

3.71 Sensors.c File Reference

Functions to get data from sensors and packet it to send later.

```
#include <ti/drivers/I2C.h>
#include <ti/drivers/UART.h>
#include <ti/drivers/ADC.h>
#include <ti/drivers/SPI.h>
#include <ti/drivers/PWM.h>
#include <ti/drivers/Timer.h>
#include <ti/drivers/Watchdog.h>
#include "Board.h"
#include "hal_UART.h"
#include "ADXL355.h"
#include "BME280.h"
#include "LDC1000.h"
#include "STARPORTS_App.h"
```

Functions

- `uint8_t Add_s16Data2Packet (uint8_t *DataPacket, uint8_t DataPacketLen, uint16_t SensorId, int16_t *DataSensor, uint8_t DataSensorLen)`
Sensors Add s16 data to packet Function.
- `uint8_t Add_s32Data2Packet (uint8_t *DataPacket, uint8_t DataPacketLen, uint16_t SensorId, int32_t *DataSensor, uint8_t DataSensorLen)`
Sensors Add s32 data to packet Function.
- `uint8_t GetSensorData (uint8_t *DataPacket)`
Sensors Get sensors data Function.
- `uint8_t IniPacket (uint8_t *DataPacket, uint16_t Nodeld)`
Sensors Ini packet Function.

Variables

- `I2C_Handle i2c`
- `struct ADXL355_Data MyADXL`
- `struct BME280_Data MyBME`
- `struct LDC1000_Data MyLDC`
- `struct Node MyNode`
- `struct TMP006_Data MyTMP006`
- `struct Vbat_Data MyVbat`
- `SPI_Handle spi`
- `uint8_t Timer1Event`
- `UART_Handle uart0`
- `Watchdog_Handle wd`

3.71.1 Detailed Description

Functions to get data from sensors and packet it to send later.

Version

1.0

Date

07/05/2019

Author

A.Irizar @tittle Sensors functions

Definition in file [Sensors.c](#).

3.71.2 Function Documentation

```
3.71.2.1 Add_s16Data2Packet() uint8_t Add_s16Data2Packet (
    uint8_t * DataPacket,
    uint8_t DataPacketLen,
    uint16_t SensorId,
    int16_t * DataSensor,
    uint8_t DataSensorLen )
```

Sensors Add s16 data to packet Function.

This function

1. Adds DataSensor to DataPacket

Parameters

in	<i>uint8_t</i>	*DataPacket Pointer to DataPacket
in	<i>uint8_t</i>	DataPacketLen
in	<i>uint16_t</i>	SensorId
in	<i>int16_t</i>	*DataSensor Pointer to DataSensor
in	<i>uint8_t</i>	DataSensorLen

Returns

uint8_t n new length of Datapacket

Definition at line 92 of file [Sensors.c](#).

```
00092
00093
00094     uint8_t i, j, n, a;
00095
00096     n=DataPacketLen;
00097
00098     DataPacket[n++] = (SensorId & 0xFF00) >> 8;
00099     DataPacket[n++] = (SensorId & 0x00FF);
00100
00101     for (i=0; i<DataSensorLen;i++) {
00102         for (j=0;j<2;j++) {
00103             a = (DataSensor[i] >> (8-(j<<3))) & 0x00FF;
00104             DataPacket[n++] = a;
00105         }
00106     }
00107
00108     return n;
00109 }
```

```
3.71.2.2 Add_s32Data2Packet() uint8_t Add_s32Data2Packet (
    uint8_t * DataPacket,
    uint8_t DataPacketLen,
    uint16_t SensorId,
    int32_t * DataSensor,
    uint8_t DataSensorLen )
```

Sensors Add s32 data to packet Function.

This function

1. Adds DataSensor to DataPacket

Parameters

in	<i>uint8_t</i>	*DataPacket Pointer to DataPacket
in	<i>uint8_t</i>	DataPacketLen
in	<i>uint16_t</i>	SensorId
in	<i>int32_t</i>	*DataSensor Pointer to DataSensor
in	<i>uint8_t</i>	DataSensorLen

Returns

uint8_t n new length of Datapacket

Definition at line 127 of file [Sensors.c](#).

```
00127     {
00128     uint8_t i, j, n, a;
00129     n=DataPacketLen;
00130     DataPacket[n++] = (SensorId & 0xFF00) >> 8;
00131     DataPacket[n++] = (SensorId & 0x00FF);
00132
00133     for (i=0; i<DataSensorLen; i++) {
00134         for (j=0; j<4; j++) {
00135             a = (DataSensor[i] >> (24-(j<3))) & 0x00FF;
00136             DataPacket[n++] = a;
00137         }
00138     }
00139     return n;
00140 }
```

3.71.2.3 GetSensorData() *uint8_t* GetSensorData (
uint8_t * *DataPacket*)

Sensors Get sensors data Function.

This function

1. Reads data from TMP006, ADC, ADXL355, BME280 and LDC1000 sensors and packet them in *DataPacket, alternating data from sensors due to MyNode.NBoot

Parameters

in	<i>uint8_t</i>	*DataPacket Pointer to DataPacket
----	----------------	-----------------------------------

Returns

uint8_t DataPacketLen length of Datapacket

Gets ADC Data (Normally VBat Value)and add data to Datapacket

Gets ADXL355 accelerometer Data and add data to Datapacket if N.Boot = 0 or mode = NORMAL WIFI

Gets BME280 sensor Data and add data to Datapacket if N.Boot = 1 or mode = NORMAL WIFI

Gets LDC1000 sensor Data and add data to Datapacket

Definition at line 160 of file Sensors.c.

```

00160
00161
00162     ADC_Handle adc;
00163     //PWM_Handle pwm;
00164     Timer_Handle timer;
00165
00166     int16_t s16DataSensor[32];
00167     int32_t s32DataSensor[32];
00168     uint8_t DataPacketLen;
00169     uint32_t interval = 10000;
00170     //uint8_t cr;
00171
00172     uint16_t DevId;
00173
00174     int i=0;
00175         // Variables for Vbat Sensor (ADC)
00176     uint16_t adcVal=0;
00177         // Variables for Vbat Sensor (ADC)
00178     uint16_t adcValMean=0;
00179         // Variables for Vbat Sensor (ADC)
00180
00181     uint32_t PressureUn;
00182         // Variables for BME280 Sensor
00183     uint32_t TemperatureUn;
00184         // Variables for BME280 Sensor
00185     uint16_t HumidityUn;
00186         // Variables for BME280 Sensor
00187     struct bme280_calib_data *MyCalib = malloc(sizeof *MyCalib);
00188         // Variables for BME280 Sensor
00189
00190     DataPacketLen = IniPacket(DataPacket, MyNode.NodeId);
00191         // Init Data Packet to Send.If Remove NodeId from the packet header, comment the line
00192     //DataPacketLen = 0;
00193         // and uncomment this one
00194
00195     timer = Startup_Oneshot_Timer(Board_TIMER1, interval);
00196         // Init Timer to take several measures at exactly the same interval
00197     adc = Startup_ADC(Board_ADC0);
00198         // Init CC3220 Internal ADC
00199     if ((adc != NULL) & (timer!=NULL)) {
00200         do {
00201             if (Timer_start(timer)==Timer_STATUS_ERROR) break;
00202             ADC_convert(adc, &adcVal);
00203             Timer1Event = 0;
00204             adcValMean += adcVal;
00205             i++;
00206             while (!Timer1Event) {};
00207         } while (i<=MyVbat.NSamples);
00208         //Watchdog_clear(wd);
00209         ADC_close(adc);
00210         Timer_stop(timer);
00211         Timer_close(timer);
00212
00213 #ifdef DEBUG
00214     UART_write uart0, "3\r\n", 3;
00215 #endif
00216     s16DataSensor[0] = (int16_t)(adcValMean/MyVbat.NSamples);
00217     DataPacketLen = Add_s16Data2Packet(DataPacket, DataPacketLen, MyVbat.SensorId, s16DataSensor,
00218     1);           //Add ADC Data to Packet
00219
00220     if (spi != NULL) {
00221         ADXL355_Enable();
00222         ADXL355_SPI_Enable();
00223         usleep(100);
00224         DevId = ADXL355_DevId(spi);
00225         if (DevId==ADXL355_ID) {
00226             ADXL355_Reset(spi);
00227             ADXL355_Range_Conf(spi, I2C_HIGH_SPEED | RANGE2G);

```

```

00222         ADXL355_Filter_Conf(spi, HPFOFF | ODR250HZ);
00223         ADXL355_Power_Conf(spi, DRDY_OFF | TEMP_OFF | MEASUREMENT);
00224         usleep(20000);
00225         if (MyNode.NBoot==0 || MyNode.Mode== MODE_NORMAL_WIFI) {
00226             ADXL355_Get_Accel_Frame(spi, MyADXL.NSamples, s32DataSensor);
00227             //Get Accelerometer Data
00228             DataPacketLen = Add_s32Data2Packet(DataPacket, DataPacketLen, MyADXL.SensorId,
00229             s32DataSensor, 6); //Add ADXL355 Data to Packet
00230         }
00231 #ifdef DEBUG
00230         UART_write(uart0, "4\r\n", 3);
00231 #endif
00232     }
00233     //Watchdog_clear(wd);
00234     ADXL355_SPI_Disable();
00235     ADXL355_Disable();
00236
00237     BME280_Enable();
00238     BME280_SPI_Enable();
00239     usleep(100);
00240     DevId = (uint16_t)BME280_DevId(spi);
00241     if (DevId==BME280_ID) {
00242         BME280_Reset(spi);
00243         usleep(2000);
00244         if (MyNode.NBoot==1 || MyNode.Mode== MODE_NORMAL_WIFI) {
00245             UART_PRINT("BME280\r\n");
00246             BME280_Read_Calib(spi, MyCalib);
00247             BME280_Write_Ctrl_Hum(spi, OSRS_HX1);
00248             BME280_Write_Ctrl_Meas(spi, OSRS_TX1 | OSRS_RX1 | FORCED);
00249             usleep(10000);
00250             while (BME280_Read_Status(spi)==MEASURING) {}
00251             PressureUn = BME280_Read_Pressure(spi);
00252             HumidityUn = BME280_Read_Humidity(spi);
00253             TemperatureUn = BME280_Read_Temperature(spi);
00254             s32DataSensor[2] = (int32_t)compensate_temperature(TemperatureUn, MyCalib);
00255             s32DataSensor[1] = compensate_humidity(HumidityUn, MyCalib);
00256             s32DataSensor[0] = compensate_pressure(PressureUn, MyCalib);
00257
00258             DataPacketLen = Add_s32Data2Packet(DataPacket, DataPacketLen, MyBME.SensorId,
00259             s32DataSensor, 3); //Add BME280 Data to Packet
00260         }
00261     }
00262 #ifdef DEBUG
00262         UART_write(uart0, "5\r\n ", 3);
00263 #endif
00264     free(MyCalib);
00265     //Watchdog_clear(wd);
00266     BME280_SPI_Disable();
00267     BME280_Disable();
00268
00269 #ifdef LDC1000
00270     LDC1000_Enable();
00271     LDC1000_SPI_Enable();
00272     //pwm = Config_PWM(Board_PWM0);
00273     //PWM_start(pwm);
00274     //Generate CLK Signal of LDC1000 (if necessary) */
00275     usleep(1000);
00276     DevId = (uint16_t)LDC1000_DevId(spi);
00277     if (DevId==LDC1000_ID) {
00278         LDC1000_Write_Pow_Conf(spi, STBY_MODE);
00279         LDC1000_Write_Rp_Max(spi, RPMAX0007);
00280         LDC1000_Write_Rp_Min(spi, RPMIN0001p3);
00281         LDC1000_Write_Min_Freq(spi, 127);
00282         //Val = round ( 68.94 x log10(172e3/2500) )
00283         LDC1000_Write_Conf(spi, AMP_4V | RESP_TIME_6144);
00284         LDC1000_Write_Intb_Conf(spi, INTB_DIS);
00285         LDC1000_Write_Clk_Conf(spi, XIN | CLK_OFF);
00286         LDC1000_Write_Pow_Conf(spi, ACTIVE_MODE);
00287         LDC1000_Get_Proximity_Frame(spi, MyLDC.NSamples, s16DataSensor);
00288
00289         DataPacketLen = Add_s16Data2Packet(DataPacket, DataPacketLen, MyLDC.SensorId,
00290         s16DataSensor, 2); //Add LDC1000 Data to Packet
00291
00292 #ifdef DEBUG
00292         UART_write(uart0, "6 ", 2);
00293 #endif
00294     }
00295     //PWM_stop(pwm);
00296     //PWM_close(pwm);
00297     //Watchdog_clear(wd);
00298     LDC1000_SPI_Disable();
00299     LDC1000_Disable();
00300
00301 #endif
00302     }
00303     return DataPacketLen;
00304 }
00305 }
00306 }
```

```
3.71.2.4 IniPacket() uint8_t IniPacket (
    uint8_t * DataPacket,
    uint16_t Nodeld )
```

Sensors Ini packet Function.

This function

1. Initiates datapacket to send with sensors data

Parameters

in	<i>uint8_t</i>	*DataPacket Pointer to DataPacket
in	<i>uint16_t</i>	Nodeld

Returns

2

Definition at line 68 of file [Sensors.c](#).

```
00068
00069
00070     DataPacket[0] = (NodeId & 0xFF00) >> 8;
00071     DataPacket[1] = (NodeId & 0x00FF);
00072
00073     return 2;
00074 }
```

3.71.3 Variable Documentation

3.71.3.1 i2c I2C_Handle i2c

Definition at line 66 of file [STARPORTS_App.c](#).

3.71.3.2 MyADXL struct ADXL355_Data MyADXL

Definition at line 58 of file [STARPORTS_App.c](#).

3.71.3.3 MyBME struct BME280_Data MyBME

Definition at line 59 of file [STARPORTS_App.c](#).

3.71.3.4 MyLDC struct `LDC1000_Data` MyLDC

Definition at line 60 of file [STARPORTS_App.c](#).

3.71.3.5 MyNode struct `Node` MyNode

Definition at line 56 of file [STARPORTS_App.c](#).

3.71.3.6 MyTMP006 struct `TMP006_Data` MyTMP006

Definition at line 57 of file [STARPORTS_App.c](#).

3.71.3.7 MyVbat struct `Vbat_Data` MyVbat

Definition at line 61 of file [STARPORTS_App.c](#).

3.71.3.8 spi SPI_Handle spi

Definition at line 67 of file [STARPORTS_App.c](#).

3.71.3.9 Timer1Event uint8_t Timer1Event

Definition at line 53 of file [STARPORTS_App.c](#).

3.71.3.10 uart0 UART_Handle uart0

Definition at line 65 of file [STARPORTS_App.c](#).

3.71.3.11 wd Watchdog_Handle wd

Definition at line 68 of file [STARPORTS_App.c](#).

3.72 Sensors.c

```

00001
00010 //*****
00011 //           INCLUDES
00012 //*****
00013 //#include <stdlib.h>
00014 //#include <unistd.h>
00015 #include <ti/drivers/I2C.h>
00016 #include <ti/drivers/UART.h>
00017 #include <ti/drivers/ADC.h>
00018 #include <ti/drivers/SPI.h>
00019 #include <ti/drivers/PWM.h>
00020 #include <ti/drivers/Timer.h>
00021 #include <ti/drivers/Watchdog.h>
00022 #include "Board.h"
00023 #include "hal_UART.h"
00024 //#include "hal_PWM.h"
00025 //#include "hal_I2C.h"
00026 //#include "hal_Timer.h"
00027 //#include "hal_ADC.h"
00028 //#include "hal_SPI.h"
00029 //#include "hal_GPIO.h"
00030 //#include "hal_TMP006.h"
00031 #include "ADXL355.h"
00032 #include "BME280.h"
00033 #include "LDC1000.h"
00034 #include "STARPORTS_App.h"
00035
00036 //*****
00037 //           GLOBALS
00038 //*****
00039 extern uint8_t Timer1Event;
00040 extern UART_Handle uart0;
00041 extern I2C_Handle i2c;
00042 extern SPI_Handle spi;
00043 extern Watchdog_Handle wd;
00044 extern struct Node MyNode;
00045 extern struct TMP006_Data MyTMP006;
00046 extern struct ADXL355_Data MyADXL;
00047 extern struct BME280_Data MyBME;
00048 extern struct LDC1000_Data MyLDC;
00049 extern struct Vbat_Data MyVbat;
00050
00051 //*****
00052 //           FUNCTIONS
00053 //*****
00054
00055 //*****
00056 //
00066 //
00067 //*****
00068 uint8_t IniPacket(uint8_t *DataPacket, uint16_t NodeId) {
00069
00070     DataPacket[0] = (NodeId & 0xFF00) » 8;
00071     DataPacket[1] = (NodeId & 0x00FF);
00072
00073     return 2;
00074 }
00075
00076 //*****
00077 //
00090 //
00091 //*****
00092 uint8_t Add_s16Data2Packet(uint8_t *DataPacket, uint8_t DataPacketLen, uint16_t SensorId, int16_t
    *DataSensor, uint8_t DataSensorLen) {
00093
00094     uint8_t i, j, n, a;
00095
00096     n=DataPacketLen;
00097
00098     DataPacket[n++] = (SensorId & 0xFF00) » 8;
00099     DataPacket[n++] = (SensorId & 0x00FF);
00100
00101     for (i=0; i<DataSensorLen;i++) {
00102         for (j=0;j<2;j++) {
00103             a = (DataSensor[i] » (8-(j«3))) & 0x00FF;
00104             DataPacket[n++] = a;
00105         }
00106     }
00107
00108     return n;
00109 }
00110
00111 //*****
00112 //
00125 //

```

```

00126 //*****
00127 uint8_t Add_s32Data2Packet(uint8_t *DataPacket, uint8_t DataPacketLen, uint16_t SensorId, int32_t
00128     *DataSensor, uint8_t DataSensorLen) {
00129     uint8_t i, j, n, a;
00130
00131     n=DataPacketLen;
00132
00133     DataPacket[n++] = (SensorId & 0xFF00) >> 8;
00134     DataPacket[n++] = (SensorId & 0x00FF);
00135
00136     for (i=0; i<DataSensorLen;i++) {
00137         for (j=0;j<4;j++) {
00138             a = (DataSensor[i] >> (24-(j<3))) & 0x00FF;
00139             DataPacket[n++] = a;
00140         }
00141     }
00142
00143     return n;
00144 }
00145
00146 //*****
00147 //
00158 //
00159 //*****
00160 uint8_t GetSensorData(uint8_t *DataPacket) {
00161
00162     ADC_Handle adc;
00163     //PWM_Handle pwm;
00164     Timer_Handle timer;
00165
00166     int16_t s16DataSensor[32];
00167     int32_t s32DataSensor[32];
00168     uint8_t DataPacketLen;
00169     uint32_t interval = 10000;
00170     //uint8_t cr;
00171
00172     uint16_t DevId;
00173
00174     int i=0;
00175         // Variables for Vbat Sensor (ADC)
00176     uint16_t adcVal=0;
00177         // Variables for Vbat Sensor (ADC)
00178     uint16_t adcValMean=0;
00179         // Variables for Vbat Sensor (ADC)
00180
00181     uint32_t PressureUn;
00182         // Variables for BME280 Sensor
00183     uint32_t TemperatureUn;
00184         // Variables for BME280 Sensor
00185     uint16_t HumidityUn;
00186         // Variables for BME280 Sensor
00187     struct bme280_calib_data *MyCalib = malloc(sizeof *MyCalib);
00188         // Variables for BME280 Sensor
00189
00190     DataPacketLen = IniPacket(DataPacket, MyNode.NodeId);
00191         // Init Data Packet to Send.If Remove NodeId from the packet header, comment the line
00192     //DataPacketLen = 0;
00193         // and uncomment this one
00194
00195     timer = Startup_Oneshot_Timer(Board_TIMER1, interval);
00196         // Init Timer to take several measures at exactly the same interval
00197     adc = Startup_ADC(Board_ADC0);
00198         // Init CC3220 Internal ADC
00199     if ((adc != NULL) & (timer!=NULL)) {
00200         do {
00201             if (Timer_start(timer)==Timer_STATUS_ERROR) break;
00202             ADC_convert(adc, &adcVal);
00203             Timer1Event = 0;
00204             adcValMean += adcVal;
00205             i++;
00206             while (!Timer1Event) {};
00207             } while (i<=MyVbat.NSamples);
00208             //Watchdog_clear(wd);
00209             ADC_close(adc);
00210             Timer_stop(timer);
00211             Timer_close(timer);
00212 #ifdef DEBUG
00213             UART_write(uart0, "3\r\n",3);
00214 #endif
00215             s16DataSensor[0] = (int16_t)(adcValMean/MyVbat.Nsamples);
00216             DataPacketLen = Add_s16Data2Packet(DataPacket, DataPacketLen, MyVbat.SensorId, s16DataSensor,
00217             1);           //Add ADC Data to Packet
00218     }
00219
00220     if (spi != NULL) {
00221         ADXL355_Enable();

```

```

00216     ADXL355_SPI_Enable();
00217     usleep(100);
00218     DevId = ADXL355_DevId(spi);
00219     if (DevId==ADXL355_ID) {
00220         ADXL355_Reset(spi);
00221         ADXL355_Range_Conf(spi, I2C_HIGH_SPEED | RANGE2G);
00222         ADXL355_Filter_Conf(spi, HPFOFF | ODR250HZ);
00223         ADXL355_Power_Conf(spi, DRDY_OFF | TEMP_OFF | MEASUREMENT);
00224         usleep(20000);
00225         if (MyNode.NBoot==0 || MyNode.Mode== MODE_NORMAL_WIFI) {
00226             ADXL355_Get_Accel_Frame(spi, MyADXL.NSamples, s32DataSensor);
00227             //Get Accelerometer Data
00228             DataPacketLen = Add_s32Data2Packet(DataPacket, DataPacketLen, MyADXL.SensorId,
00229             s32DataSensor, 6); //Add ADXL355 Data to Packet
00230         }
00231 #ifdef DEBUG
00232         UART_write(uart0, "4\r\n",3);
00233     }
00234     //Watchdog_clear(wd);
00235     ADXL355_SPI_Disable();
00236     ADXL355_Disable();
00237
00238     BME280_Enable();
00239     BME280_SPI_Enable();
00240     usleep(100);
00241     DevId = (uint16_t)BME280_DevId(spi);
00242     if (DevId==BME280_ID) {
00243         BME280_Reset(spi);
00244         usleep(2000);
00245         if (MyNode.NBoot==1 || MyNode.Mode== MODE_NORMAL_WIFI) {
00246             UART_PRINT("BME280\r\n");
00247             BME280_Read_Calib(spi, MyCalib);
00248             BME280_Write_Ctrl_Hum(spi, OSRS_HX1);
00249             BME280_Write_Ctrl_Meas(spi, OSRS_TX1 | OSRS_PX1 | FORCED);
00250             usleep(10000);
00251             while (BME280_Read_Status(spi)==MEASURING) {}
00252             PressureUn = BME280_Read_Pressure(spi);
00253             HumidityUn = BME280_Read_Humidity(spi);
00254             TemperatureUn = BME280_Read_Temperature(spi);
00255             s32DataSensor[2] = (int32_t)compensate_temperature(TemperatureUn, MyCalib);
00256             s32DataSensor[1] = compensate_humidity(HumidityUn, MyCalib);
00257             s32DataSensor[0] = compensate_pressure(PressureUn, MyCalib);
00258
00259             DataPacketLen = Add_s32Data2Packet(DataPacket, DataPacketLen, MyBME.SensorId,
00260             s32DataSensor, 3); //Add BME280 Data to Packet
00261         }
00262     }
00263 #ifdef DEBUG
00264         UART_write(uart0, "5\r\n ",3);
00265     }
00266     free(MyCalib);
00267     //Watchdog_clear(wd);
00268     BME280_SPI_Disable();
00269     BME280_Disable();
00270
00271 #ifdef LDC1000
00272     LDC1000_Enable();
00273     LDC1000_SPI_Enable();
00274     //pwm = Config_PWM(Board_PWM0);
00275     //PWM_start(pwm);
00276     //Generate CLK Signal of LDC1000 (if necessary) */
00277     usleep(1000);
00278     DevId = (uint16_t)LDC1000_DevId(spi);
00279     if (DevId==LDC1000_ID) {
00280         LDC1000_Write_Pow_Conf(spi, STBY_MODE);
00281         LDC1000_Write_Rp_Max(spi, RPMAX0007);
00282         LDC1000_Write_Rp_Min(spi, RPMIN0001p3);
00283         LDC1000_Write_Min_Freq(spi, 127);
00284         //Val = round ( 68.94 x log10(172e3/2500) )
00285         LDC1000_Write_Conf(spi, AMP_4V | RESP_TIME_6144);
00286         LDC1000_Write_Intb_Conf(spi, INTB_DIS);
00287         LDC1000_Write_Clk_Conf(spi, XIN | CLK_OFF);
00288         LDC1000_Write_Pow_Conf(spi, ACTIVE_MODE);
00289         LDC1000_Get_Proximity_Frame(spi, MyLDC.NSamples, s16DataSensor);
00290
00291         DataPacketLen = Add_s16Data2Packet(DataPacket, DataPacketLen, MyLDC.SensorId,
00292         s16DataSensor, 2); //Add LDC1000 Data to Packet
00293
00294 #ifdef DEBUG
00295         UART_write(uart0, "6 ",2);
00296     }
00297 #endif
00298     //PWM_stop(pwm);
00299     //PWM_close(pwm);
00300     //Watchdog_clear(wd);
00301     LDC1000_SPI_Disable();
00302

```

```
00303     LDC1000_Disable();  
00304 #endif  
00305 }  
00306 return DataPacketLen;  
00307 }
```

3.73 Sensors.h File Reference

Functions to get data from sensors and packet it to send later.

Functions

- `uint8_t Add_s16Data2Packet (uint8_t *DataPacket, uint8_t DataPacketLen, uint16_t SensorId, int16_t *DataSensor, uint8_t DataSensorLen)`
Sensors Add s16 data to packet Function.
- `uint8_t Add_s32Data2Packet (uint8_t *DataPacket, uint8_t DataPacketLen, uint16_t SensorId, int32_t *DataSensor, uint8_t DataSensorLen)`
Sensors Add s32 data to packet Function.
- `uint8_t GetSensorData (uint8_t *DataPacket)`
Sensors Get sensors data Function.
- `uint8_t IniPacket (uint8_t *DataPacket, uint16_t Nodeld)`
Sensors Ini packet Function.

3.73.1 Detailed Description

Functions to get data from sensors and packet it to send later.

Version

1.0

Date

07/05/2019

Author

A.Irizar @title Sensors functions

Definition in file [Sensors.h](#).

3.73.2 Function Documentation

3.73.2.1 Add_s16Data2Packet() `uint8_t Add_s16Data2Packet (`
 `uint8_t * DataPacket,`
 `uint8_t DataPacketLen,`
 `uint16_t SensorId,`
 `int16_t * DataSensor,`
 `uint8_t DataSensorLen)`

Sensors Add s16 data to packet Function.

This function

1. Adds DataSensor to DataPacket

Parameters

in	<i>uint8_t</i>	*DataPacket Pointer to DataPacket
in	<i>uint8_t</i>	DataPacketLen
in	<i>uint16_t</i>	SensorId
in	<i>int16_t</i>	*DataSensor Pointer to DataSensor
in	<i>uint8_t</i>	DataSensorLen

Returns

uint8_t n new length of Datapacket

Definition at line 92 of file [Sensors.c](#).

```
00092
00093
00094     uint8_t i, j, n, a;
00095
00096     n=DataPacketLen;
00097
00098     DataPacket[n++] = (SensorId & 0xFF00) >> 8;
00099     DataPacket[n++] = (SensorId & 0x00FF);
00100
00101     for (i=0; i<DataSensorLen;i++) {
00102         for (j=0;j<2;j++) {
00103             a = (DataSensor[i] >> (8-(j<<3))) & 0x00FF;
00104             DataPacket[n++] = a;
00105         }
00106     }
00107
00108     return n;
00109 }
```

3.73.2.2 Add_s32Data2Packet() *uint8_t* Add_s32Data2Packet (

```
    uint8_t * DataPacket,
    uint8_t DataPacketLen,
    uint16_t SensorId,
    int32_t * DataSensor,
    uint8_t DataSensorLen )
```

Sensors Add s32 data to packet Function.

This function

1. Adds DataSensor to DataPacket

Parameters

in	<i>uint8_t</i>	*DataPacket Pointer to DataPacket
in	<i>uint8_t</i>	DataPacketLen
in	<i>uint16_t</i>	SensorId
Generated by <i>Int32Gen</i>	<i>uint8_t</i>	*DataSensor Pointer to DataSensor
in	<i>uint8_t</i>	DataSensorLen

Returns

`uint8_t n` new length of Datapacket

Definition at line 127 of file [Sensors.c](#).

```
00127
00128     {
00129     uint8_t i, j, n, a;
00130
00131     n=DataPacketLen;
00132
00133     DataPacket[n++] = (SensorId & 0xFF00) >> 8;
00134     DataPacket[n++] = (SensorId & 0x00FF);
00135
00136     for (i=0; i<DataSensorLen;i++) {
00137         for (j=0;j<4;j++) {
00138             a = (DataSensor[i] >> (24-(j<3))) & 0x00FF;
00139             DataPacket[n++] = a;
00140         }
00141     }
00142
00143     return n;
00144 }
```

3.73.2.3 GetSensorData() `uint8_t GetSensorData (`
`uint8_t * DataPacket)`

Sensors Get sensors data Function.

This function

1. Reads data from TMP006, ADC, ADXL355, BME280 and LDC1000 sensors and packet them in *DataPacket, alternating data from sensors due to MyNode.NBoot

Parameters

in	<code>uint8_t</code>	*DataPacket Pointer to DataPacket
----	----------------------	-----------------------------------

Returns

`uint8_t DataPacketLen` length of Datapacket

Gets ADC Data (Normally VBat Value)and add data to Datapacket

Gets ADXL355 accelerometer Data and add data to Datapacket if N.Boot = 0 or mode = NORMAL WIFI

Gets BME280 sensor Data and add data to Datapacket if N.Boot = 1 or mode = NORMAL WIFI

Gets LDC1000 sensor Data and add data to Datapacket

Definition at line 160 of file [Sensors.c](#).

```
00160
00161
00162     ADC_Handle adc;
00163     //PWM_Handle pwm;
00164     Timer_Handle timer;
00165
00166     int16_t s16DataSensor[32];
00167     int32_t s32DataSensor[32];
```

```

00168     uint8_t DataPacketLen;
00169     uint32_t interval = 10000;
00170     //uint8_t cr;
00171
00172     uint16_t DevId;
00173
00174     int i=0;
00175         // Variables for Vbat Sensor (ADC)
00176     uint16_t adcVal=0;
00177         // Variables for Vbat Sensor (ADC)
00178     uint16_t adcValMean=0;
00179         // Variables for Vbat Sensor (ADC)
00180
00181     uint32_t PressureUn;
00182         // Variables for BME280 Sensor
00183     uint32_t TemperatureUn;
00184         // Variables for BME280 Sensor
00185     uint16_t HumidityUn;
00186         // Variables for BME280 Sensor
00187     struct bme280_calib_data *MyCalib = malloc(sizeof *MyCalib);
00188         // Variables for BME280 Sensor
00189
00190     DataPacketLen = IniPacket(DataPacket, MyNode.NodeId);
00191         // Init Data Packet to Send.If Remove NodeId from the packet header, comment the line
00192     //DataPacketLen = 0;
00193         // and uncomment this one
00194
00195     timer = Startup_Oneshot_Timer(Board_TIMER1, interval);
00196         // Init Timer to take several measures at exactly the same interval
00197     adc = Startup_ADC(Board_ADC0);
00198         // Init CC3220 Internal ADC
00199     if ((adc != NULL) & (timer!=NULL)) {
00200         do {
00201             if (Timer_start(timer)==Timer_STATUS_ERROR) break;
00202             ADC_convert(adc, &adcVal);
00203             Timer1Event = 0;
00204             adcValMean += adcVal;
00205             i++;
00206             while (!Timer1Event) {};
00207         } while (i<=MyVbat.NSamples);
00208         //Watchdog_clear(wd);
00209         ADC_close(adc);
00210         Timer_stop(timer);
00211         Timer_close(timer);
00212 #ifdef DEBUG
00213         UART_write(uart0, "3\r\n",3);
00214 #endif
00215         s16DataSensor[0] = (int16_t)(adcValMean/MyVbat.NSamples);
00216         DataPacketLen = Add_s16Data2Packet(DataPacket, DataPacketLen, MyVbat.SensorId, s16DataSensor,
00217         1);           //Add ADC Data to Packet
00218
00219     if (spi != NULL) {
00220         ADXL355_Enable();
00221         ADXL355_SPI_Enable();
00222         usleep(100);
00223         DevId = ADXL355_DevId(spi);
00224         if (DevId==ADXL355_ID) {
00225             ADXL355_Reset(spi);
00226             ADXL355_Range_Conf(spi, I2C_HIGH_SPEED | RANGE2G);
00227             ADXL355_Filter_Conf(spi, HPFOFF | ODR250HZ);
00228             ADXL355_Power_Conf(spi, DRDY_OFF | TEMP_OFF | MEASUREMENT);
00229             usleep(20000);
00230             if (MyNode.NBoot==0 || MyNode.Mode== MODE_NORMAL_WIFI) {
00231                 ADXL355_Get_Accel_Frame(spi, MyADXL.NSamples, s32DataSensor);
00232                 //Get Accelerometer Data
00233                 DataPacketLen = Add_s32Data2Packet(DataPacket, DataPacketLen, MyADXL.SensorId,
00234                 s32DataSensor, 6); //Add ADXL355 Data to Packet
00235             }
00236 #ifdef DEBUG
00237             UART_write(uart0, "4\r\n",3);
00238 #endif
00239         }
00240         //Watchdog_clear(wd);
00241         ADXL355_SPI_Disable();
00242         ADXL355_Disable();
00243
00244         BME280_Enable();
00245         BME280_SPI_Enable();
00246         usleep(100);
00247         DevId = (uint16_t)BME280_DevId(spi);
00248         if (DevId==BME280_ID) {
00249             BME280_Reset(spi);
00250             usleep(2000);
00251             if (MyNode.NBoot==1 || MyNode.Mode== MODE_NORMAL_WIFI) {
00252                 UART_PRINT("BME280\r\n");
00253                 BME280_Read_Calib(spi, MyCalib);
00254             }
00255         }
00256     }
00257 }
```

```

00250     BME280_Write_Ctrl_Hum(spi, OSRS_HX1);
00251     BME280_Write_Ctrl_Meas(spi, OSRS_TX1 | OSRS_PX1 | FORCED);
00252     usleep(10000);
00253     while (BME280_Read_Status(spi)==MEASURING) {}
00254     PressureUn = BME280_Read_Pressure(spi);
00255     HumidityUn = BME280_Read_Humidity(spi);
00256     TemperatureUn = BME280_Read_Temperature(spi);
00257     s32DataSensor[2] = (int32_t) compensate_temperature(TemperatureUn, MyCalib);
00258     s32DataSensor[1] = compensate_humidity(HumidityUn, MyCalib);
00259     s32DataSensor[0] = compensate_pressure(PressureUn, MyCalib);
00260
00261     DataPacketLen = Add_s32Data2Packet(DataPacket, DataPacketLen, MyBME.SensorId,
00262     s32DataSensor, 3); //Add BME280 Data to Packet
00263 #ifdef DEBUG
00264     UART_write(uart0, "5\r\n ",3);
00265 #endif
00266 }
00267 free(MyCalib);
00268 //Watchdog_clear(wd);
00269 BME280_SPI_Disable();
00270 BME280_Disable();
00271
00272 #ifdef LDC1000
00273     LDC1000_Enable();
00274     LDC1000_SPI_Enable();
00275     //pwm = Config_PWM(Board_PWM0);
00276     //PWM_Start(pwm);
00277     //Generate CLK Signal of LDC1000 (if necessary) */
00278     usleep(1000);
00279     DevId = (uint16_t)LDC1000_DevId(spi);
00280     if (DevId==LDC1000_ID) {
00281         LDC1000_Write_Pow_Conf(spi, STBY_MODE);
00282         LDC1000_Write_Rp_Max(spi, RPMAX0007);
00283         LDC1000_Write_Rp_Min(spi, RPMIN0001p3);
00284         LDC1000_Write_Min_Freq(spi, 127);
00285         //Val = round ( 68.94 * log10(172e3/2500) )
00286         LDC1000_Write_Conf(spi, AMP_4V | RESP_TIME_6144);
00287         LDC1000_Write_Intb_Conf(spi, INTB_DIS);
00288         LDC1000_Write_Clk_Conf(spi, XIN | CLK_OFF);
00289         LDC1000_Write_Pow_Conf(spi, ACTIVE_MODE);
00290         LDC1000_Get_Proximity_Frame(spi, MyLDC.NSamples, s16DataSensor);
00291
00292     DataPacketLen = Add_s16Data2Packet(DataPacket, DataPacketLen, MyLDC.SensorId,
00293     s16DataSensor, 2); //Add LDC1000 Data to Packet
00294 #ifdef DEBUG
00295     UART_write(uart0, "6 ",2);
00296 #endif
00297 }
00298 //PWM_Stop(pwm);
00299 //PWM_Close(pwm);
00300 //Watchdog_Clear(wd);
00301 LDC1000_SPI_Disable();
00302 LDC1000_Disable();
00303
00304 #endif
00305 }
00306 return DataPacketLen;
00307 }

```

3.73.2.4 IniPacket() uint8_t IniPacket (

```

        uint8_t * DataPacket,
        uint16_t NodeId )

```

Sensors Ini packet Function.

This function

1. Initiates datapacket to send with sensors data

Parameters

in	<i>uint8_t</i>	*DataPacket Pointer to DataPacket
in	<i>uint16_t</i>	NodeId

Returns

2

Definition at line 68 of file [Sensors.c](#).

```

00068
00069
00070     DataPacket[0] = (NodeId & 0xFF00) >> 8;
00071     DataPacket[1] = (NodeId & 0x00FF);
00072
00073     return 2;
00074 }
```

3.74 Sensors.h

```

00001
00010 #ifndef SENSORS_H_
00011 #define SENSORS_H_
00012
00013 //*****
00014 // FUNCTION PROTOTYPES
00015 //*****
00016 uint8_t GetSensorData(uint8_t *DataPacket);
00017 uint8_t IniPacket(uint8_t *DataPacket, uint16_t NodeId);
00018 uint8_t Add_s16Data2Packet(uint8_t *DataPacket, uint8_t DataPacketLen, uint16_t SensorId, int16_t
    *DataSensor, uint8_t DataSensorLen);
00019 uint8_t Add_s32Data2Packet(uint8_t *DataPacket, uint8_t DataPacketLen, uint16_t SensorId, int32_t
    *DataSensor, uint8_t DataSensorLen);
00020
00021 #endif /* SENSORS_H_ */
```

3.75 STARPORTS_App.c File Reference

Starports Main program flow.

```

#include <ti/drivers/UART.h>
#include <ti/drivers/I2C.h>
#include <ti/drivers/SPI.h>
#include <ti/drivers/GPIO.h>
#include <ti/drivers/Watchdog.h>
#include <ti/drivers/net/wifi/simplelink.h>
#include <ti/devices/cc32xx/inc/hw_types.h>
#include <ti/devices/cc32xx/driverlib/prcm.h>
#include "hal_UART.h"
#include "wifi.h"
#include "file_system.h"
#include "Board.h"
#include "STARPORTS_App.h"
```

Macros

- `#define TIMEOUT_MS 40000`

Functions

- void * `mainThread` (void *arg0)
- void `WakeupFromLPDS` (uint_least8_t)

Variables

- I2C_Handle `i2c`
- uint8_t `LPDSOut` = 0
- struct `ADXL355_Data` `MyADXL` = {`ADXL355_ID`,500,256}
- struct `BME280_Data` `MyBME` = {`BME280_ID`}
- struct `LDC1000_Data` `MyLDC` = {`LDC1000_ID`,16}
- struct `Node` `MyNode` = {`NODEID`,1200,`MODE_NORMAL_LORA`,16,"Orange-D62D"}
- struct `TMP006_Data` `MyTMP006` = {`TMP006_ID`}
- struct `Vbat_Data` `MyVbat` = {`VBAT_ID`,16}
- SPI_Handle `spi`
- uint8_t `Timer0Event` = 0
- uint8_t `Timer1Event` = 0
- UART_Handle `uart0`
- Watchdog_Handle `wd`

3.75.1 Detailed Description

Starports Main program flow.

Version

1.0

Date

07/05/2019

Author

A.Irizar

Definition in file [STARPORTS_App.c](#).

3.75.2 Macro Definition Documentation

3.75.2.1 `TIMEOUT_MS` #define TIMEOUT_MS 40000

Definition at line 47 of file [STARPORTS_App.c](#).

3.75.3 Function Documentation

3.75.3.1 mainThread() void* mainThread (void * arg0)

Definition at line 111 of file [STARPORTS_App.c](#).

```

00112 {
00113     UART_Handle uart1;                                //uart1 communicates
00114     with RN2483 Lora module                         //struct of parameters
00115     struct LoraNode MyLoraNode;
00116     of the node
00117     uint8_t DataPacket[256];                          //packet with sensors
00118     data
00119     uint8_t DataPacketLen;                           //lenght of Datapacket
00120     unsigned char buf[256];
00121     uint8_t ret;
00122     uint8_t sz;
00123     uint8_t NextStep = CONTINUE;                    //controls node OFF
00124     uint8_t mask;                                    //mask to change node
00125     parameters
00126     uint16_t nodeId;                               //node ID var
00127     int16_t      sockId;                           //wifi var
00128     int16_t      status = -1;                      //wifi var
00129     GPIO_setConfig(Board_EN_NODE, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_HIGH); //config node enable
00130     GPIO_Node_Enable();                            //Quicky enable de
00131     node (the jumper an be removed)
00132     Watchdog_init();                             //Inits watchdog
00133     instance
00134     ret = GPIO_Config();                          //Configure the rest
00135     of GPIO pins
00136     wd = Startup_Watchdog(Board_WATCHDOG0, TIMEOUT_MS); //Open a Watchdog
00137     driver instance
00138     SPI_init();                                  //initiates SPI before
00139     sl_Start();
00140     sl_Start(0, 0, 0);                           //needed to work with
00141     File System
00142     uart0 = Startup_UART(Board_UART0, 115200); //initiates uart
00143 #ifdef DEBUG
00144     UART_write(uart0, "\r\nInitiating Test of STARPORTS...\r\n", 35); //debug
00145 #endif
00146     /***** Begin Read Configuration Files *****/
00147     MyLoraNode.PortNoTx = 1;                      //Get
00148     MyNode.WakeUpInterval = st_readFileWakeUp();   //Get
00149     MyNode.WakeUpInterval --> Read WakeUp_Time File
00150     MyNode.Mode = st_read FileMode();             //Get MyNode.Mode -->
00151     Read File Mode
00152     MyNode.NCycles = st_readFileNCycles();        //Get MyNode.NCycles
00153     --> Read File Ncycles
00154     MyNode.NBoot = st_readFileNBoot();            //Get MyNode.FirstBoot
00155     --> Read FirstBoot File: Yes (1) No (0)
00156     MyNode.NFails = st_readFileNFails();          //Get MyNode.NFails
00157     --> Number of failed attempts to Wireless Conn.
00158     MyNode.ChangeWakeUp = st_readFileChangeWakeUp(); //Get
00159     MyNode.ChangeWakeUp --> Read change wake up interval or not
00160     //MyNode.NodeId = st_readFileNodeId();          //Get MyNodeId
00161     //st_readFileSSID(&(MyNode.SSID));           //Get MyNode.SSID[]
00162     --> Read SSID File
00163     /***** Ends Reading Configuration Files *****/
00164     if (MyNode.NFails>=4) {                        //if NFails >= 4
00165 #ifdef DEBUG
00166     UART_PRINT("NFails > 4, SETTING NODE IN WIFI MODE\r\n");
00167 #endif
00168     st_readFileSSID(&(MyNode.SSID));              //Read SSID from file
00169     wlanConf();                                 //Setup Node as WiFi
00170     wlanConnectFromFile(MyNode.SSID);            //connect to Known
00171     Host
00172     MyNode.NBoot=0;                             //set NBoot = 0
00173     writeNBoot(MyNode.NBoot);                  //write to file NBoot
00174     MyNode.NFails=0;                           //set NFails = 0
00175     writeNFails(MyNode.NFails);                //write to file NFails
00176 }
```

```

00166     /***** Begin Configure Peripherals *****/
00167     RN2483_Clear();                                     //Reset the RN2483
00168     i2c = Startup_I2C(Board_I2C0);                   //I2C interface
00169     started
00170
00171     /***** End Configure Peripherals *****/
00172
00173     /***** Begin Setting Node WakeUp_Time in RTC */
00174     if (DS1374_Read_Ctrl(i2c)==0x06|| MyNode.ChangeWakeUp == 1 ) {           //if DS1374 is powerup
00175         or MyNode.ChangeWakeUp == 1
00176         DS1374_Clear_AF(i2c);                                //clear AF register of
00177         DS1374
00178         DS1374_Write_WdAlmb(i2c, MyNode.WakeUpInterval);    //configures the RTC
00179         DS1374_Write_Ctrl(i2c);                            //write Control
00180         register of DS1374
00181     } else {                                            //if DS1374 was not
00182         powerup
00183         DS1374_Clear_AF(i2c);                                //clear AF register of
00184         DS1374
00185     }
00186
00187     /***** Begin Configuration and Setup Wireless Connectivity *****/
00188     uart1 = Startup_UART(Board_UART1, 57600);          //initiates UART1
00189     (connected to RN2483)
00190
00191     if (MyNode.Mode==MODE_NORMAL_LORA) {
00192         RN2483_Set();                                    //if Mode = Lora
00193         releases RN2483                                //Set /MCLR Pin to 1
00194         sz = GetLine_UART(uart1, buf);                  //collects
00195         initialization text of RN2483
00196
00197         if (sz==0) {                                    //Buffer size is too
00198             large
00199         }
00200
00201         MyLoraNode.Upctr = st_readFileUpCntr();        //Read upctr from file
00202         Mac_Set_Upctr(uart1,&MyLoraNode);            //set upctr on node
00203         MyLoraNode.Dnctr = st_readFileDnCntr();        //Read dnctr from file
00204         Mac_Set_Dnctr(uart1,&MyLoraNode);            //set dnctr on node
00205         Mac_Ar_On(uart1);                            //Set Automatic
00206
00207         Retransmit ON
00208         // Mac_Set_Pwridx(uart1, 1);
00209         default is 1, the maximum 14dBm
00210         // Mac_Adr_On(uart1);
00211         datarate ON (useful when using OTAA)
00212
00213         ret = Join_Abp_Lora(uart1);                  //Join ABP
00214         if (ret==SUCCESS_ABP_LORA){                   //if Join ABP =
00215             Success
00216             MyNode.NFails=0;
00217             writeNFails(MyNode.NFails);                //set NFails = 0
00218             } else {                                  //if Join ABP = Fail
00219             } #ifdef DEBUG
00220             UART_write(uart0,"Join_Abp_Lora() Failed",22); //debug
00221             UART_write(uart0, "\r\n", 2);               //debug
00222             #endif
00223             MyNode.NFails++;
00224             writeNFails(MyNode.NFails);                //Nfails ++
00225             NextStep=SHUTDOWN;                         //Write to NFails File
00226             //shutdown node
00227         } else if (MyNode.Mode==MODE_NORMAL_WIFI){
00228             st_readfileSSID(&(MyNode.SSID));          //if Mode = wifi
00229             ret = wlanConf();                          //Read SSID from file
00230             ret = wlanConnectFromFile(MyNode.SSID);    //Setup WiFi
00231             file
00232             if (ret==SUCCESS_CONNECT_WIFI) {           //Connect WiFi from
00233                 success
00234                 MyNode.NFails=0;
00235                 writeNFails(MyNode.NFails);            //set NFails = 0
00236                 UART_PRINT("\n\rCONN OK!\n\r");
00237                 } else {
00238                     fail
00239                     MyNode.NFails++;
00240                     writeNFails(MyNode.NFails);            //if connect wifi =
00241                     UART_PRINT("\n\rCONN KO!\n\r");
00242                     NextStep=SHUTDOWN;                    //Write to NFails File
00243                     //shutdown node
00244                 }
00245             }
00246             /***** End of Configuration and Setup Wireless Connectivity *****/
00247             if (NextStep==SHUTDOWN) {                   //if nextstep =
00248                 shutdown
00249                 I2C_Close(i2c);                      //close i2c
00250                 I2C_As_GPIO_Low();                  //Puts SCL and SDA
00251                 signals low to save power
00252                 Node_Disable();                    //Auto Shutdown
00253             }

```

```

00234
00235     /***** Begin Reading Data from Sensors *****/
00236     SPI_CS_Disable();                                     //Put all CS to Logic
00237     High
00238         spi = Startup_SPI(Board_SPI_MASTER, 8, 5000000);          //Configure SPI Master
00239         at 5 Mbps, 8-bits, CPOL=0, PHA=0
00240         Timer_init();                                         //Start Timer for ADC
00241         Timestep
00242             DataPacketLen = GetSensorData(DataPacket);           //get sensors data
00243             /***** End Reading Data from Sensors *****/
00244
00245         SPI_Close(spi);                                       //Close all sensor
00246         related peripherals                                         //Not really enable
00247
00248         the SPI but just puts the CS signal to '0'           //to prevent PCB from
00249
00250         not really shutting down (a high value in some SPI signals
00251
00252         active)                                              //keeps sensor chips
00253             LDC1000_SPI_Enable();                                //ldc spi enable
00254             ADXL355_SPI_Enable();                                //adxl spi enable
00255             BME280_SPI_Enable();                                //bme spi enable
00256             SPI_As_GPIO_Low();                                 //Puts all rest of SPI
00257             signals to '0' to save power                      //close i2c
00258             I2C_Close(i2c);                                    //Puts SCL and SDA
00259             I2C_As_GPIO_Low();                                signals low to save power
00260
00261             MyLoraNode.DataLenTx = Uint8Array2Char(DataPacket, DataPacketLen,
00262             &(MyLoraNode.DataTx)); //initiates data to send
00263
00264 #ifdef DEBUG
00265     UART_Write(uart0, &(MyLoraNode.DataTx), MyLoraNode.DataLenTx);      //debug
00266     UART_Write(uart0, "\r\n", 2);                                         //debug
00267 #endif
00268
00269     if (MyNode.Mode==MODE_NORMAL_LORA) {                                //if Mode = Lora
00270         ret = Tx_Uncnf_Lora(uart1, &MyLoraNode, &mask, &nodeId);
00271         several tries?
00272             if (ret==SUCCESS_TX_MAC_TX) {                                //if tx = success
00273                 MyLoraNode.Upctr = Mac_Get_Upctr(uart1);                //Get upctr from
00274
00275             RN2483
00276                 writeUpCntr(MyLoraNode.Upctr);                            //write upctr to file
00277                 MyLoraNode.Dnctr = Mac_Get_Dnctr(uart1);                //Get dnctr from
00278
00279             RN2483
00280                 writeDnCntr(MyLoraNode.Dnctr);                            //write dnctr to file
00281                 //writeNFails(0);
00282             } else if (ret == SUCCESS_TX_MAC_RX) {                          //if rx = success
00283                 MyLoraNode.Upctr = Mac_Get_Upctr(uart1);                //Get upctr from
00284
00285             RN2483
00286                 writeUpCntr(MyLoraNode.Upctr);                            //*****SI HAGO ESTE
00287                 WRITE_MACHACO_EL_VALOR_DE_DOWNLINK PARA ESTE PARAM.        //Get dnctr from
00288                 MyLoraNode.Dnctr = Mac_Get_Dnctr(uart1);                //*****
00289             RN2483
00290                 writeDnCntr(MyLoraNode.Dnctr);                            //*****SI HAGO ESTE
00291                 WRITE_MACHACO_EL_VALOR_DE_DOWNLINK PARA ESTE PARAM.
00292
00293             /*if (nodeId==MyNode.NodeId) {                                //Nfails = 0 and write
00294                 // Write New Configuration Data to Files
00295                 if (mask&0x01!=0) {
00296                     writeWakeUp(MyNode.WakeUpInterval);
00297                 }
00298                 if (mask&0x02!=0) {
00299                     // writeFirstBoot(MyNode.FirstBoot);
00300                     // writeMode(MyNode.Mode);
00301                 }
00302             }*/
00303             //writeNFails(0);                                           //Nfails = 0 and write
00304             to Nfails
00305             } else {                                                 //if tx = fail
00306             #ifdef DEBUG
00307                 UART_Write(uart0, "Tx_Uncnf_Lora() Failed",22);       //debug
00308                 UART_Write(uart0, "\r\n", 2);                           //debug
00309             #endif
00310             //MyNode.Nfails++;                                         //Nfails ++
00311             //writeNFails(MyNode.Nfails);                            //write to Nfails file
00312         }
00313     } else if (MyNode.Mode==MODE_NORMAL_WIFI) {                         //if Mode = wifi

```

```

00304     prepareDataFrame(PORT, DEST_IP_ADDR); //prepare destination
00305     sockId = sl_Socket(SL_AF_INET,SL_SOCKET_DGRAM, 0); //creates a socket
00306     if(sockId < 0){ //if socket = fail
00307         UART_PRINT("error UDP %d\n\r",sockId); //debug
00308     }
00309     status = sl_SendTo(sockId, MyLoraNode.DataTx, MyLoraNode.DataLenTx, 0,(SlSockAddr_t *) &PowerMeasure_CB.ipV4Addr,sizeof(SlSockAddrIn_t)); //send data
00310     if(status < 0 ){ //if send = fail
00311         status = sl_Close(sockId); //close socket
00312         ASSERT_ON_ERROR(sockId); //debug error
00313         UART_PRINT("\n\rERROR SENDING PACKET: %s\n\r", status); //debug error
00314         Node_Disable(); //Auto Shutdown
00315     }else{ //if send = success
00316         status = sl_Close(sockId); //close socket
00317         UART_PRINT("PACKET SEND: %s\n\r", MyLoraNode.DataTx); //debug ok
00318     }
00319 }
00320
00321     writeNBoot(1-MyNode.NBoot); //change NBoot and
00322     write to NBoot file
00323 #ifdef DEBUG
00324     UART_write(uart0, "End of STARPORTS Measures\r\n", 27); //debug
00325 #endif
00326
00327     UART_close(uart1); //close uart1
00328     UART_close(uart0); //close uart0
00329     Node_Disable(); //Auto shutdown
00330 }

```

3.75.3.2 WakeupFromLPDS() void WakeupFromLPDS (uint_least8_t var)

Definition at line 332 of file [STARPORTS_App.c](#).

```

00332 {
00333     LPDSOut = 1;
00334 }
```

3.75.4 Variable Documentation

3.75.4.1 i2c I2C_Handle i2c

Definition at line 66 of file [STARPORTS_App.c](#).

3.75.4.2 LPDSOut uint8_t LPDSOut = 0

Definition at line 54 of file [STARPORTS_App.c](#).

3.75.4.3 MyADXL struct ADXL355_Data MyADXL = {ADXL355_ID,500,256}

Definition at line 58 of file [STARPORTS_App.c](#).

3.75.4.4 MyBME struct `BME280_Data` MyBME = {`BME280_ID`}

Definition at line 59 of file [STARPORTS_App.c](#).

3.75.4.5 MyLDC struct `LDC1000_Data` MyLDC = {`LDC1000_ID`,16}

Definition at line 60 of file [STARPORTS_App.c](#).

3.75.4.6 MyNode struct `Node` MyNode = {`NODEID`,1200,`MODE_NORMAL_LORA`,16,"Orange-D62D"}

Definition at line 56 of file [STARPORTS_App.c](#).

3.75.4.7 MyTMP006 struct `TMP006_Data` MyTMP006 = {`TMP006_ID`}

Definition at line 57 of file [STARPORTS_App.c](#).

3.75.4.8 MyVbat struct `Vbat_Data` MyVbat = {`VBAT_ID`,16}

Definition at line 61 of file [STARPORTS_App.c](#).

3.75.4.9 spi SPI_Handle spi

Definition at line 67 of file [STARPORTS_App.c](#).

3.75.4.10 Timer0Event uint8_t Timer0Event = 0

Definition at line 52 of file [STARPORTS_App.c](#).

3.75.4.11 Timer1Event uint8_t Timer1Event = 0

Definition at line 53 of file [STARPORTS_App.c](#).

3.75.4.12 uart0 UART_Handle uart0

Definition at line 65 of file [STARPORTS_App.c](#).

3.75.4.13 wd Watchdog_Handle wd

Definition at line 68 of file [STARPORTS_App.c](#).

3.76 STARPORTS_App.c

```

00001
00009 //*****
00010 //           INCLUDES
00011 //*****
00012 ##include <string.h>
00013 ##include <stdlib.h>
00014 ##include <stdio.h>
00015 ##include <stdint.h>
00016 ##include <stddef.h>
00017 ##include <stdbool.h>
00018 ##include <unistd.h>
00019 #include <ti/drivers/UART.h>
00020 #include <ti/drivers/I2C.h>
00021 #include <ti/drivers/SPI.h>
00022 ##include <ti/drivers/Timer.h>
00023 ##include <ti/drivers/Power.h>
00024 #include <ti/drivers/GPIO.h>
00025 #include <ti/drivers/Watchdog.h>
00026 ##include <ti/drivers/net/wifi/wlan.h>
00027 #include <ti/drivers/net/wifi/simplelink.h>
00028 #include <ti/devices/cc32xx/inc/hw_types.h>
00029 #include <ti/devices/cc32xx/driverlib/prcm.h>
00030 #include "hal_UART.h"
00031 ##include "hal_Timer.h"
00032 ##include "hal_GPIO.h"
00033 ##include "hal_WD.h"
00034 ##include "hal_I2C.h"
00035 ##include "hal_SPI.h"
00036 #include "wifi.h"
00037 #include "file_system.h"
00038 ##include "DS1374.h"
00039 ##include "Sensors.h"
00040 ##include "hal_LORA.h"
00041 #include "Board.h"
00042 #include "STARPORTS_App.h"
00043
00044 //*****
00045 //           DEFINES
00046 //*****
00047 #define TIMEOUT_MS 40000                                // 40 seconds Timeout
00048
00049 //*****
00050 //           GLOBALS
00051 //*****
00052 uint8_t Timer0Event = 0;
00053 uint8_t Timer1Event = 0;
00054 uint8_t LPDSOut = 0;
00055
00056 struct Node MyNode = {NODEID,1200,MODE_NORMAL_LORA,16,"Orange-D62D"};
00057 struct TMP006_Data MyTMP006 = {TMP006_ID};
00058 struct ADXL355_Data MyADXL = {ADXL355_ID,500,256};
00059 struct BME280_Data MyBME = {BME280_ID};
00060 struct LDC1000_Data MyLDC = {LDC1000_ID,16};
00061 struct Vbat_Data MyVbat = {VBAT_ID,16};
00062
00063 void WakeupFromLPDS(uint_least8_t);
00064
00065 UART_Handle uart0;                                     // uart0 is global to
00066                         debug messages. Can be removed in final deployment
00067 I2C_Handle i2c;
00068 SPI_Handle spi;
00069 Watchdog_Handle wd;
00070 //*****
00071 //
00109 //
00110 //*****

```

```

0011 void *mainThread(void *arg0)
0012 {
0013     UART_Handle uart1;                                //uart1 communicates
0014     with RN2483 Lora module                         //struct of parameters
0015     struct LoraNode MyLoraNode;
0016     of the node
0017     uint8_t DataPacket[256];                          //packet with sensors
0018     data
0019     uint8_t DataPacketLen;                           //length of Datapacket
0020     unsigned char buf[256];                          //uart 1 buffer
0021     uint8_t ret;
0022     uint8_t sz;
0023     uint8_t NextStep = CONTINUE;                    //controls node OFF
0024     uint8_t mask;                                  //mask to change node
0025     parameters
0026     uint16_t nodeId;                               //node ID var
0027     int16_t      sockId;                           //wifi var
0028     int16_t      status = -1;                      //wifi var
0029     GPIO_setConfig(Board_EN_NODE, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_HIGH); //config node enable
0030     GPIO
0031     Node_Enable();                                //Quicky enable de
0032     node (the jumper can be removed)
0033     Watchdog_init();                            //Inits watchdog
0034     instance
0035     ret = GPIO_Config();                         //Configure the rest
0036     of GPIO pins
0037     wd = Startup_Watchdog(Board_WATCHDOGO, TIMEOUT_MS); //Open a Watchdog
0038     driver instance
0039     SPI_init();                                 //initiates SPI before
0040     sl_Start
0041     sl_Start(0, 0, 0);                          //needed to work with
0042     File System
0043     uart0 = Startup_UART(Board_UART0, 115200); //initiates uart0
0044
0045 #ifdef DEBUG
0046     UART_write(uart0, "\r\nInitiating Test of STARPORTS...\r\n", 35); //debug
0047 #endif
0048
0049 /***** Begin Read Configuration Files *****/
0050 MyLoraNode.PortNoTx = 1;                         //Get
0051 MyNode.WakeUpInterval = st_readFileWakeUp();       //Get
0052 MyNode.WakeUpInterval --> Read WakeUp_Time File //Get MyNode.Mode -->
0053 MyNode.Mode = st_read FileMode();                 //Get MyNode.Mode
0054 MyNode.NCycles = st_readFileNCycles();            //Get MyNode.NCycles
0055 --> Read File Ncycles
0056 MyNode.NBoot = st_readFileNBoot();                //Get MyNode.FirstBoot
0057 --> Read FirstBoot File: Yes (1) No (0)
0058 MyNode.NFails = st_readFileNFails();              //Get MyNode.NFails
0059 --> Number of failed attempts to Wireless Conn.
0060 MyNode.ChangeWakeUp = st_readFileChangeWakeUp();   //Get
0061 MyNode.ChangeWakeUp --> Read change wake up interval or not
0062 //MyNode.NodeId = st_readFileNodeId();             //Get MyNodeId
0063 //st_readFileSSID(&(MyNode.SSID));               //Get MyNode.SSID[]
0064 --> Read SSID File
0065 /***** Ends Reading Configuration Files *****/
0066
0067 if (MyNode.NFails>=4) {                           //if NFails >= 4
0068 #ifdef DEBUG
0069     UART_PRINT("NFails > 4, SETTING NODE IN WIFI MODE\r\n");
0070 #endif
0071     st_readFileSSID(&(MyNode.SSID));           //Read SSID from file
0072     wlanConf();                                //Setup Node as WiFi
0073     wlanConnectFromFile(MyNode.SSID);          //connect to Known
0074
0075 Host
0076
0077     MyNode.NBoot=0;                            //set NBoot = 0
0078     writeNBoot(MyNode.NBoot);                  //write to file NBoot
0079     MyNode.NFails=0;                           //set NFails = 0
0080     writeNFails(MyNode.NFails);                //write to file NFails
0081 }
0082
0083 /***** Begin Configure Peripherals *****/
0084 RN2483_Clear();                                //Reset the RN2483
0085 i2c = Startup_I2C(Board_I2C0);                //I2C interface
0086 started
0087 /***** End Configure Peripherals *****/
0088
0089 /***** Begin Setting Node WakeUp_Time in RTC */
0090 if (DS1374_Read_Ctrl(i2c)==0x06|| MyNode.ChangeWakeUp == 1 ) { //if DS1374 is powerup
0091 or MyNode.ChangeWakeUp == 1
0092     DS1374_Clear_AF(i2c);                     //clear AF register of
0093 DS1374
0094     DS1374_Write_WdAlmb(i2c, MyNode.WakeUpInterval); //configures the RTC

```

```

00176     DS1374_Write_Ctrl(i2c);
00177     register of DS1374
00178   } else {
00179     powerup
00180     DS1374
00181   }
00182   **** Begin Configuration and Setup Wireless Connectivity ****
00183   uart1 = Startup_UART(Board_UART1, 57600);
00184   (connected to RN2483)
00185   if (MyNode.Mode==MODE_NORMAL_LORA) {
00186     RN2483_Set();
00187     releases RN2483
00188     sz = GetLine_UART(uart1, buf);
00189     initialization text of RN2483
00190     if (sz==0) {
00191       large
00192     }
00193     MyLoraNode.Upctr = st_readFileUpCntr();
00194     Mac_Set_Upctr(uart1,&MyLoraNode);
00195     MyLoraNode.Dnctr = st_readFileDnCntr();
00196     Mac_Set_Dnctr(uart1,&MyLoraNode);
00197     Mac_Ar_On(uart1);
00198     Retransmit ON
00199     // Mac_Set_Pwridx(uart1, 1);
00200     default is 1, the maximum 14dBm
00201     // Mac_Adr_On(uart1);
00202     datarate ON (useful when using OTAA)
00203     ret = Join_Abp_Lora(uart1);
00204     if (ret==SUCCESS_ABP_LORA){
00205       Success
00206       MyNode.NFails=0;
00207       writeNFails(MyNode.NFails);
00208     } else {
00209       #ifdef DEBUG
00210         UART_write(uart0,"Join_Abp_Lora() Failed",22);
00211         UART_write(uart0, "\r\n", 2);
00212       #endif
00213       MyNode.NFails++;
00214       writeNFails(MyNode.NFails);
00215       NextStep=SHUTDOWN;
00216     }
00217   } else if (MyNode.Mode==MODE_NORMAL_WIFI){
00218     st_readfileSSID(&(MyNode.SSID));
00219     ret = wlanConf();
00220     ret = wlanConnectFromFile((MyNode.SSID));
00221     file
00222     if (ret==SUCCESS_CONNECT_WIFI) {
00223       success
00224       MyNode.NFails=0;
00225       writeNFails(MyNode.NFails);
00226       UART_PRINT("\n\rCONN OK!\n\r");
00227     } else {
00228       fail
00229       MyNode.NFails++;
00230       writeNFails(MyNode.NFails);
00231       UART_PRINT("\n\rCONN KO!\n\r");
00232       NextStep=SHUTDOWN;
00233     }
00234   }
00235   **** End of Configuration and Setup Wireless Connectivity ****
00236   SPI_CS_Disable();
00237   High
00238   spi = Startup_SPI(Board_SPI_MASTER, 8, 5000000);
00239   at 5 Mbps, 8-bits, CPOL=0, PHA=0
00240   Timer_init();
00241   Timestep
00242   DataPacketLen = GetSensorData(DataPacket);
00243   **** End Reading Data from Sensors ****
00244   SPI_close(spi);
00245   related peripherals

```

```

00243     the SPI but just puts the CS signal to '0'                                //Not really enable
00244     not really shutting down (a high value in some SPI signals           //to prevent PCB from
00245     active)                                                               //keeps sensor chips
00246         LDC1000_SPI_Enable();                                              //ldc spi enable
00247         ADXL355_SPI_Enable();                                              //adxl spi enable
00248         BME280_SPI_Enable();                                              //bme spi enable
00249         SPI_As_GPIO_Low();                                                 //Puts all rest of SPI
00250         signals to '0' to save power                                         //close i2c
00251             I2C_Close(i2c);                                                 //Puts SCL and SDA
00252             I2C_As_GPIO_Low();
00253             signals low to save power
00254
00255     MyLoraNode.DataLenTx = Uint8Array2Char(DataPacket, DataPacketLen,
00256     &(MyLoraNode.DataTx)); //initiates data to send
00257
00258 #ifdef DEBUG
00259     UART_Write(uart0, &(MyLoraNode.DataTx), MyLoraNode.DataLenTx);        //debug
00260     UART_Write(uart0, "\r\n", 2);                                           //debug
00261 #endif
00262
00263     if (MyNode.Mode==MODE_NORMAL_LORA) {                                       //if Mode = Lora
00264         ret = Tx_Uncnf_Lora(uart1, &MyLoraNode, &mask, &nodeId);          //Transmit Data,
00265         several tries?
00266         if (ret==SUCCESS_TX_MAC_TX) {                                         //if tx = success
00267             MyLoraNode.Upctr = Mac_Get_Upctr(uart1);                         //Get upctr from
00268             RN2483
00269             writeUpCntr(MyLoraNode.Upctr);                                     //write upctr to file
00270             MyLoraNode.Dnctr = Mac_Get_Dnctr(uart1);                         //Get dnctr from
00271             RN2483
00272             writeDnCntr(MyLoraNode.Dnctr);                                     //write dnctr to file
00273             //writeNfails(0);
00274         } else if (ret == SUCCESS_TX_MAC_RX) {                                 //if rx = success
00275             MyLoraNode.Upctr = Mac_Get_Upctr(uart1);                         //Get upctr from
00276             RN2483
00277             writeUpCntr(MyLoraNode.Upctr);                                     //*****SI HAGO ESTE
00278             WRITE_MACHACO_EL_VALOR_DE_DOWNLINK PARA ESTE PARAM.            //Get dnctr from
00279             MyLoraNode.Dnctr = Mac_Get_Dnctr(uart1);                         //*****
00280             RN2483
00281             writeDnCntr(MyLoraNode.Dnctr);                                     //*****SI HAGO ESTE
00282             WRITE_MACHACO_EL_VALOR_DE_UPLINK PARA ESTE PARAM.
00283             /*if (nodeId==MyNode.NodeId) {                                         //Write New Configuration Data to Files
00284                 if (mask&0x01!=0) {                                           
00285                     writeWakeUp(MyNode.WakeUpInterval);                      //if tx = fail
00286                 }
00287                 if (mask&0x02!=0) {                                           
00288                     // writeFirstBoot(MyNode.FirstBoot);                      //if tx = fail
00289                     // writeMode(MyNode.Mode);                           //if tx = fail
00290                 }
00291                 if (mask&0x04!=0) {                                           
00292                     writeSSID(&(MyNode.SSID));                           //if tx = fail
00293                 }
00294                 if (mask&0x08!=0) {                                           
00295                     writeNCycles(MyNode.NCycles);                          //if tx = fail
00296                 }
00297                 if (mask&0x10!=0) {                                           
00298                     writeUpCntr(MyLoraNode.Upctr);                         //if tx = fail
00299                 }
00300             }*/
00301             /*/
00302             //writeNfails(0);                                              //Nfails = 0 and write
00303         } else {                                                       //if tx = fail
00304             #ifdef DEBUG
00305                 UART_Write(uart0, "Tx_Uncnf_Lora() Failed",22);        //debug
00306                 UART_Write(uart0, "\r\n", 2);                           //debug
00307             #endif
00308             //MyNode.Nfails++;                                         //Nfails ++
00309             //writeNfails(MyNode.Nfails);                                //write to Nfails file
00310         }
00311     } else if (MyNode.Mode==MODE_NORMAL_WIFI) {                               //if Mode = wifi
00312         prepareDataFrame(PORT, DEST_IP_ADDR);
00313         frame
00314             sockId = sl_Socket(SL_AF_INET,SL_SOCKET_DGRAM, 0);           //creates a socket
00315             if(sockId < 0){                                               //if socket = fail
00316                 UART_PRINT("error UDP %d\r\n",sockId);                  //debug
00317             }
00318             status = sl_SendTo(sockId, MyLoraNode.DataTx, MyLoraNode.DataLenTx, 0,(SlSockAddr_t
00319             *)&PowerMeasure_CB.ipV4Addr,sizeof(SlSockAddrIn_t)); //send data
00320             if(status < 0){                                              //if send = fail
00321                 status = sl_Close(sockId);                            //close socket
00322                 ASSERT_ON_ERROR(sockId);                            //debug error
00323                 UART_PRINT("\r\nERROR SENDING PACKET: %s\r\n", status); //debug error
00324                 Node_Disable();                                    //Auto Shutdown

```

```

00315     }else{
00316         status = sl_Close(sockId);
00317         UART_PRINT("PACKET SEND: %s\n\r", MyLoraNode.DataTx);
00318     }
00319 }
00320
00321     writeNBoot(l-MyNode.NBoot); //change NBoot and
00322     write to NBoot file
00323 #ifdef DEBUG
00324     UART_write(uart0, "End of STARPORTS Measures\r\n", 27); //debug
00325 #endif
00326
00327     UART_close(uart1); //close uart1
00328     UART_close(uart0); //close uart0
00329     Node_Disable(); //Auto shutdown
00330 }
00331
00332 void WakeupFromLPDS(uint_least8_t var) {
00333     LPDSOut = 1;
00334 }
```

3.77 STARPORTS_App.h File Reference

Starports Main program flow.

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

Data Structures

- struct [ADXL355_Data](#)
STARPORTS ADXL355 Data structure. [More...](#)
- struct [BME280_Data](#)
STARPORTS BME280 Data structure. [More...](#)
- struct [Keller_Data](#)
STARPORTS Keller Data structure. [More...](#)
- struct [LDC1000_Data](#)
STARPORTS LDC1000 Data structure. [More...](#)
- struct [Node](#)
STARPORTS Node parameters structure. [More...](#)
- struct [TMP006_Data](#)
STARPORTS TMP006 Data structure. [More...](#)
- struct [Vbat_Data](#)
STARPORTS Battery Data structure. [More...](#)

Macros

- #define [ADXL355_ID](#) 0xAD1D
- #define [APPEUI](#) "70B3D57ED0019CD5"
- #define [APPKEY](#) "E506C0086DA5C31C301FCAC72BC808F6"
- #define [APPSKEY](#) "56297CFF77806B54D335DE9F94745DEB"
- #define [BME280_ID](#) 0x0060
- #define [CONTINUE](#) 0
- #define [DEVADDR](#) "2601267B"
- #define [DEVEUI](#) "0004A30B00EB15E8"
- #define [LDC1000_ID](#) 0x0080
- #define [MODE_AP_WIFI](#) 0x04

- #define MODE_NORMAL_LORA 0x00
- #define MODE_NORMAL_WIFI 0x02
- #define MODE_STORAGE_LORA 0x01
- #define MODE_STORAGE_WIFI 0x03
- #define NODEID 0x7abc
- #define NUMBER_SENSORS 6
- #define NWKSKEY "1AE937D87A65EDE42B1FCE61BCF277B9"
- #define RNW_LSB 0
- #define RNW_MSB 1
- #define SHUTDOWN 1
- #define TMP006_ID 0x5449
- #define VBAT_ID 0x0055

Enumerations

- enum Node_states_t {
POWERUP, GET_PARAMS, CONFIG_LORA, CONFIG_WIFI,
GET_SENSOR_DATA, SEND_DATA_LORA, SEND_DATA_WIFI, STORE_DATA,
GET_NEW_PARAMS, STORE_NEW_PARAMS }

STARPORTS *Node states* enum.

Variables

- uint8_t NodeState

3.77.1 Detailed Description

Starports Main program flow.

Version

1.0

Date

07/05/2019

Author

A.Irizar @tittle Starports Main

Definition in file [STARPORTS_App.h](#).

3.77.2 Data Structure Documentation

3.77.2.1 struct ADXL355_Data

 STARPORTS ADXL355 Data structure.

Definition at line 99 of file [STARPORTS_App.h](#).

Data Fields

uint16_t	NSamples	Number of Samples to Process
uint16_t	SampleRate	Sample rate
uint16_t	SensorId	ADXL355 Dev ID

3.77.2.2 struct BME280_Data STARPORTS BME280 Data structure.

Definition at line 108 of file [STARPORTS_App.h](#).

Data Fields

uint16_t	SensorId	BME280 Dev ID
----------	----------	---------------

3.77.2.3 struct Keller_Data STARPORTS Keller Data structure.

Definition at line 131 of file [STARPORTS_App.h](#).

Data Fields

uint16_t	SensorId	Keller sensor ID
----------	----------	------------------

3.77.2.4 struct LDC1000_Data STARPORTS LDC1000 Data structure.

Definition at line 115 of file [STARPORTS_App.h](#).

Data Fields

uint16_t	NSamples	Number of Samples to Process
uint16_t	SensorId	LDC1000 Dev ID

3.77.2.5 struct Node STARPORTS Node parameters structure.

Definition at line 74 of file [STARPORTS_App.h](#).

Data Fields

uint8_t	ChangeWakeUp	Value to determine if it neccesary to change wakr up interval on RTC
uint8_t	Mode	Mode (wifi/lora)
uint8_t	NBoot	N boot - to alternate data from sensors
uint16_t	NCycles	Number of cycles
uint16_t	NFails	Number of Lora connection fails
uint16_t	NodeID	Node ID
unsigned char	SSID[13]	Wifi SSID
uint16_t	WakeUpInterval	Wake up interval(s)

3.77.2.6 struct TMP006_Data STARPORTS TMP006 Data structure.

Definition at line 92 of file [STARPORTS_App.h](#).

Data Fields

uint16_t	SensorId	TMP006 Dev ID
----------	----------	---------------

3.77.2.7 struct Vbat_Data STARPORTS Battery Data structure.

Definition at line 123 of file [STARPORTS_App.h](#).

Data Fields

uint16_t	NSamples	Number of Samples to Process
uint16_t	SensorId	Battery sensor ID

3.77.3 Macro Definition Documentation**3.77.3.1 ADXL355_ID** #define ADXL355_ID 0xAD1D

Definition at line 22 of file [STARPORTS_App.h](#).

3.77.3.2 APPEUI #define APPEUI "70B3D57ED0019CD5"

Definition at line 43 of file [STARPORTS_App.h](#).

3.77.3.3 APPKEY #define APPKEY "E506C0086DA5C31C301FCAC72BC808F6"

Definition at line 44 of file [STARPORTS_App.h](#).

3.77.3.4 APPSKEY #define APPSKEY "56297CFF77806B54D335DE9F94745DEB"

Definition at line 47 of file [STARPORTS_App.h](#).

3.77.3.5 BME280_ID #define BME280_ID 0x0060

Definition at line 23 of file [STARPORTS_App.h](#).

3.77.3.6 CONTINUE #define CONTINUE 0

Definition at line 35 of file [STARPORTS_App.h](#).

3.77.3.7 DEVADDR #define DEVADDR "2601267B"

Definition at line 45 of file [STARPORTS_App.h](#).

3.77.3.8 DEVEUI #define DEVEUI "0004A30B00EB15E8"

Definition at line 42 of file [STARPORTS_App.h](#).

3.77.3.9 LDC1000_ID #define LDC1000_ID 0x0080

Definition at line 24 of file [STARPORTS_App.h](#).

3.77.3.10 MODE_AP_WIFI #define MODE_AP_WIFI 0x04

Definition at line 32 of file [STARPORTS_App.h](#).

3.77.3.11 MODE_NORMAL_LORA #define MODE_NORMAL_LORA 0x00

Definition at line 28 of file [STARPORTS_App.h](#).

3.77.3.12 MODE_NORMAL_WIFI #define MODE_NORMAL_WIFI 0x02

Definition at line 30 of file [STARPORTS_App.h](#).

3.77.3.13 MODE_STORAGE_LORA #define MODE_STORAGE_LORA 0x01

Definition at line 29 of file [STARPORTS_App.h](#).

3.77.3.14 MODE_STORAGE_WIFI #define MODE_STORAGE_WIFI 0x03

Definition at line 31 of file [STARPORTS_App.h](#).

3.77.3.15 NODEID #define NODEID 0x7abc

Definition at line 21 of file [STARPORTS_App.h](#).

3.77.3.16 NUMBER_SENSORS #define NUMBER_SENSORS 6

Definition at line 40 of file [STARPORTS_App.h](#).

3.77.3.17 NWKSKEY #define NWKSKEY "1AE937D87A65EDE42B1FCE61BCF277B9"

Definition at line 46 of file [STARPORTS_App.h](#).

3.77.3.18 RNW_LSB #define RNW_LSB 0

Definition at line 37 of file [STARPORTS_App.h](#).

3.77.3.19 RNW_MSB #define RNW_MSB 1

Definition at line 38 of file [STARPORTS_App.h](#).

3.77.3.20 SHUTDOWN #define SHUTDOWN 1

Definition at line 34 of file [STARPORTS_App.h](#).

3.77.3.21 TMP006_ID #define TMP006_ID 0x5449

Definition at line 26 of file [STARPORTS_App.h](#).

3.77.3.22 VBAT_ID #define VBAT_ID 0x0055

Definition at line 25 of file [STARPORTS_App.h](#).

3.77.4 Enumeration Type Documentation

3.77.4.1 Node_states_t enum [Node_states_t](#)

STARPORTS Node states enum.

Enumerator

POWERUP	POWERUP status
GET_PARAMS	GET_PARAMS status
CONFIG_LORA	CONFIG_LORA status
CONFIG_WIFI	CONFIG_WIFI status
GET_SENSOR_DATA	GET_SENSOR_DATA status
SEND_DATA_LORA	SEND_DATA_LORA status
SEND_DATA_WIFI	SEND_DATA_WIFI status
STORE_DATA	STORE_DATA status
GET_NEW_PARAMS	GET_NEW_PARAMS status
STORE_NEW_PARAMS	STORE_NEW_PARAMS status

Definition at line 58 of file [STARPORTS_App.h](#).

```

00058     {
00059     POWERUP,
00060     GET_PARAMS,
00061     CONFIG_LORA,
00062     CONFIG_WIFI,
00063     GET_SENSOR_DATA,
00064     SEND_DATA_LORA,
00065     SEND_DATA_WIFI,
00066     STORE_DATA,
00067     GET_NEW_PARAMS,
00068     STORE_NEW_PARAMS
00069 } Node_states_t;

```

3.77.5 Variable Documentation**3.77.5.1 NodeState** uint8_t NodeStateDefinition at line 49 of file [STARPORTS_App.h](#).**3.78 STARPORTS_App.h**

```

00001
00010 #ifndef STARPORTS_APP_H_
00011 #define STARPORTS_APP_H_
00012
00013 //*****
00014 //           INCLUDES
00015 //*****
00016 #include <stdint.h>
00017
00018 //*****
00019 //           DEFINES
00020 //*****
00021 #define NODEID          0x7abc
00022 #define ADXL355_ID        0xAD1D
00023 #define BME280_ID         0x0060
00024 #define LDC1000_ID        0x0080
00025 #define VBAT_ID          0x0055
00026 #define TMP006_ID         0x5449
00027
00028 #define MODE_NORMAL_LORA   0x00
00029 #define MODE_STORAGE_LORA  0x01
00030 #define MODE_NORMAL_WIFI   0x02
00031 #define MODE_STORAGE_WIFI  0x03
00032 #define MODE_AP_WIFI        0x04
00033
00034 #define SHUTDOWN           1
00035 #define CONTINUE           0
00036

```

```

00037 #define RNW_LSB          0
00038 #define RNW_MSB          1
00039
00040 #define NUMBER_SENSORS   6
00041
00042 #define DEVEUI           "0004A30B00EB15E8"
00043 #define APPEUI           "70B3D57ED0019CD5"
00044 #define APPKEY            "E506C0086DA5C31C301FCAC72BC808F6"
00045 #define DEVADDR           "2601267B"
00046 #define NWKSKEY           "1AE937D87A65EDE42B1FCE61BCF277B9"
00047 #define APPSKEY           "56297CFF77806B54D335DE9F94745DEB"
00048
00049 uint8_t NodeState;
00050
00051 //***** TYPEDEFS *****
00052 //***** TYPEDEFS *****
00053 //***** TYPEDEFS *****
00054
00055 typedef enum {
00056     POWERUP,
00057     GET_PARAMS,
00058     CONFIG_LORA,
00059     CONFIG_WIFI,
00060     GET_SENSOR_DATA,
00061     SEND_DATA_LORA,
00062     SEND_DATA_WIFI,
00063     STORE_DATA,
00064     GET_NEW_PARAMS,
00065     STORE_NEW_PARAMS
00066 } Node_states_t;
00067
00068 struct Node {
00069     uint16_t NodeId;
00070     uint16_t WakeUpInterval;
00071     uint8_t Mode;
00072     uint16_t NCycles;
00073     unsigned char SSID[13];
00074     uint16_t NFails;
00075     uint8_t NBoot;
00076     uint8_t NSensors;           /***< DEVID_AD reg*/
00077     uint16_t *SensorIdPtr;     /***< DEVID_AD reg*/
00078     uint8_t *SensorDataLen;    /***< DEVID_AD reg*/
00079     int16_t **SensorDataPtr;   /***< DEVID_AD reg*/
00080     uint8_t ChangeWakeUp;
00081 };
00082
00083 struct TMP006_Data {
00084     uint16_t SensorId;
00085 };
00086
00087 struct ADXL355_Data {
00088     uint16_t SensorId;
00089     uint16_t SampleRate;
00090     uint16_t NSamples;
00091 };
00092
00093 struct BME280_Data {
00094     uint16_t SensorId;
00095 };
00096
00097 struct LDC1000_Data {
00098     uint16_t SensorId;
00099     uint16_t NSamples;
00100 };
00101
00102 struct Vbat_Data {
00103     uint16_t SensorId;
00104     uint16_t NSamples;
00105 };
00106
00107 struct Keller_Data {
00108     uint16_t SensorId;
00109 };
00110
00111
00112
00113
00114
00115
00116
00117
00118
00119
00120
00121
00122
00123
00124
00125
00126
00127
00128
00129
00130
00131
00132
00133
00134
00135 #endif /* STARPORTS_APP_H_ */

```

3.79 wifi.c File Reference

Functions for CC3220SF wifi management.

```
#include <ti/drivers/UART.h>
#include <ti/drivers/net/wifi/simplelink.h>
```

```
#include "wifi.h"
#include "hal_UART.h"
```

Functions

- void [prepareDataFrame](#) (uint16_t port, uint32_t ipAddr)
SimpleLink Prepare data frame Function.
- int16_t [sendUdpClient](#) (uint16_t port)
SimpleLink Send UDP Client Function.
- void [SimpleLinkFatalErrorHandler](#) (SIDeviceFatal_t *slFatalErrorEvent)
SimpleLink EventHandler Function.
- void [SimpleLinkGeneralEventHandler](#) (SIDeviceEvent_t *pDevEvent)
SimpleLink EventHandler Function.
- void [SimpleLinkHttpServerEventHandler](#) (SINetAppHttpServerEvent_t *pHttpEvent, SINetAppHttpServerResponse_t *pHttpResponse)
SimpleLink EventHandler Function.
- void [SimpleLinkNetAppEventHandler](#) (SINetAppEvent_t *pNetAppEvent)
SimpleLink EventHandler Function.
- void [SimpleLinkNetAppRequestEventHandler](#) (SINetAppRequest_t *pNetAppRequest, SINetAppResponse_t *pNetAppResponse)
SimpleLink EventHandler Function.
- void [SimpleLinkNetAppRequestMemFreeEventHandler](#) (uint8_t *buffer)
SimpleLink EventHandler Function.
- void [SimpleLinkSocketTriggerEventHandler](#) (SISockTriggerEvent_t *pSITriggerEvent)
SimpleLink EventHandler Function.
- void [SimpleLinkSockEventHandler](#) (SISockEvent_t *pSock)
SimpleLink EventHandler Function.
- void [SimpleLinkWlanEventHandler](#) (SIWlanEvent_t *pWlanEvent)
SimpleLink EventHandler Function.
- int32_t [wlanConf](#) (void)
SimpleLink wlan configuration Function.
- int32_t [wlanConnect](#) (void)
SimpleLink wlan connect Function.
- int32_t [wlanConnectFromFile](#) (unsigned char *ssid)
SimpleLink wlan connect from file Function.

3.79.1 Detailed Description

Functions for CC3220SF wifi management.

Version

1.0

Date

07/05/2019

Author

A.Irizar @tittle CC3220SF wifi functions

Definition in file [wifi.c](#).

3.79.2 Function Documentation

3.79.2.1 `prepareDataFrame()` void prepareDataFrame (

in	<i>uint16_t</i>	port
		<i>_t</i>

in	<i>uint32_t</i>	<i>ipAddr</i>
		<i>_t</i>

SimpleLink Prepare data frame Function.

This function

1. Prepares the connection parameters for WIFI on CC3220SF

Parameters

in	<i>uint16_t</i>	port
in	<i>uint32_t</i>	<i>ipAddr</i>

Returns

None

Definition at line 349 of file [wifi.c](#).

```
00350 {  
00351     uint16_t idx;  
00352  
00353     for(idx = 0;idx < FRAME_LENGTH;idx++)  
00354     {  
00355         PowerMeasure_CB.frameData[idx] = (signed char)(idx % 255);  
00356     }  
00357     PowerMeasure_CB.ipV4Addr.sin_family = SL_AF_INET;  
00358     PowerMeasure_CB.ipV4Addr.sin_port = sl_Htons(port);  
00359     PowerMeasure_CB.ipV4Addr.sin_addr.s_addr = sl_Htonl(ipAddr);  
00360 }
```

3.79.2.2 `sendUdpClient()` int16_t sendUdpClient (

in	<i>uint16_t</i>	port
		<i>_t</i>

SimpleLink Send UDP Client Function.

This function

1. Sends data over UDP protocol

Parameters

in	<i>uint16_t</i>	port
		<i>_t</i>

Returns

0

Definition at line 374 of file wifi.c.

```

00375 {
00376     int16_t          sockId;
00377     int16_t          idx = 0;
00378     int32_t          status = -1;
00379
00380     sockId = sl_Socket(SL_AF_INET, SL SOCK_DGRAM, 0);
00381
00382     if(sockId > 0){
00383         UART_PRINT("SOCKET UDP");
00384     }else{
00385         UART_PRINT("error UDP");
00386     }
00387
00388     while (idx < NUM_OF_PKT){
00389         status = sl_SendTo(sockId, PowerMeasure_CB.frameData ,FRAME_LENGTH , 0,(SlSockAddr_t
00390 *) &PowerMeasure_CB.ipV4Addr, sizeof(SlSockAddrIn_t));
00391         if(status <= 0 ){
00392             status = sl_Close(sockId);
00393             ASSERT_ON_ERROR(sockId);
00394         }
00395         idx++;
00396     }
00397     return(0);
00398 }
```

3.79.2.3 SimpleLinkFatalErrorHandler() void SimpleLinkFatalErrorHandler (SlDeviceFatal_t * sl Fatal ErrorEvent)

SimpleLink EventHandler Function.

This function

1. Is EventHandler for simplelink

Parameters**Definition at line 157 of file wifi.c.**

```

00158 {
00159     switch (slFatalErrorEvent->Id){
00160         case SL_DEVICE_EVENT_FATAL_DEVICE_ABORT:{
00161             UART_PRINT(
00162                 "[ERROR] - FATAL ERROR: Abort NWP event "
00163                 "detected: AbortType=%d, AbortData=0x%x\n\r",
00164                 slFatalErrorEvent->Data.DeviceAssert.Code,
00165                 slFatalErrorEvent->Data.DeviceAssert.Value);
00166         }
00167         break;
00168
00169         case SL_DEVICE_EVENT_FATAL_DRIVER_ABORT:{
00170             UART_PRINT("[ERROR] - FATAL ERROR: Driver Abort detected. \n\r");
00171         }
00172         break;
00173
00174         case SL_DEVICE_EVENT_FATAL_NO_CMD_ACK:{
00175             UART_PRINT(
00176                 "[ERROR] - FATAL ERROR: "
00177                 "No Cmd Ack detected [cmd opcode = 0x%x] \n\r",
00178                 slFatalErrorEvent->Data.NoCmdAck.Code);
00179         }
00180         break;
00181 }
```

```
00182     case SL_DEVICE_EVENT_FATAL_SYNC_LOSS:{  
00183         UART_PRINT("[ERROR] - FATAL ERROR: Sync loss detected \n\r");  
00184     }  
00185     break;  
00186  
00187     case SL_DEVICE_EVENT_FATAL_CMD_TIMEOUT:{  
00188         UART_PRINT(  
00189             "[ERROR] - FATAL ERROR: Async event timeout "  
00190             "detected [event opcode =0x%x] \n\r",  
00191             slFatalErrorEvent->Data.CmdTimeout.Code);  
00192     }  
00193     break;  
00194  
00195     default:  
00196         UART_PRINT("[ERROR] - FATAL ERROR:  
00197             Unspecified error detected \n\r");  
00198     break;  
00199 }  
00200 }
```

3.79.2.4 SimpleLinkGeneralEventHandler()

```
void SimpleLinkGeneralEventHandler (
```

```
    SlDeviceEvent_t * pDevEvent )
```

SimpleLink EventHandler Function.

This function

1. Is EventHandler for simplelink

Parameters



Definition at line 104 of file wifi.c.

```
00105 {  
00106     UART_PRINT("[GENERAL EVENT] - ID=[%d] Sender=[%d]\n\n",  
00107             pDevEvent->Data.Error.Code,  
00108             pDevEvent->Data.Error.Source);  
00109 }
```

3.79.2.5 SimpleLinkHttpServerEventHandler()

```
void SimpleLinkHttpServerEventHandler (
```

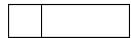
```
    SlNetAppHttpServerEvent_t * pHtpEvent,  
    SlNetAppHttpServerResponse_t * pHtpResponse )
```

SimpleLink EventHandler Function.

This function

1. Is EventHandler for simplelink

Parameters



Definition at line 62 of file [wifi.c](#).
00062 { }

3.79.2.6 SimpleLinkNetAppEventHandler() void SimpleLinkNetAppEventHandler (S1NetAppEvent_t * pNetAppEvent)

SimpleLink EventHandler Function.

This function

1. Is EventHandler for simplelink

Parameters



Definition at line 214 of file [wifi.c](#).

```
00215 {
00216     switch(pNetAppEvent->Id) {
00217         case SL_NETAPP_EVENT_IPV4_ACQUIRED:{
00218             SET_STATUS_BIT(PowerMeasure_CB.siStatus, STATUS_BIT_IP_ACQUIRED);
00219         }
00220         break;
00221
00222     default:{
00223         UART_PRINT("[NETAPP EVENT] Unexpected event [0x%x] \n\r",
00224                     pNetAppEvent->Id);
00225     }
00226     break;
00227 }
00228 }
```

3.79.2.7 SimpleLinkNetAppRequestEventHandler() void SimpleLinkNetAppRequestEventHandler (S1NetAppRequest_t * pNetAppRequest,
S1NetAppResponse_t * pNetAppResponse)

SimpleLink EventHandler Function.

This function

1. Is EventHandler for simplelink

Parameters



Definition at line 48 of file [wifi.c](#).
00048 { }

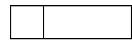
```
3.79.2.8 SimpleLinkNetAppRequestMemFreeEventHandler() void SimpleLinkNetAppRequestMemFree<-->
EventHandler (
    uint8_t * buffer )
```

SimpleLink EventHandler Function.

This function

1. Is EventHandler for simplelink

Parameters



Definition at line 34 of file [wifi.c](#).
00034 { }

```
3.79.2.9 SimpleLinkSocketTriggerEventHandler() void SimpleLinkSocketTriggerEventHandler (
    SlSockTriggerEvent_t * pSlTriggerEvent )
```

SimpleLink EventHandler Function.

This function

1. Is EventHandler for simplelink

Parameters



Definition at line 90 of file [wifi.c](#).
00090 { }

```
3.79.2.10 SimpleLinkSockEventHandler() void SimpleLinkSockEventHandler (
    SlSockEvent_t * pSock )
```

SimpleLink EventHandler Function.

This function

1. Is EventHandler for simplelink

Parameters



Definition at line 76 of file [wifi.c](#).
 00076 { }

3.79.2.11 SimpleLinkWlanEventHandler() void SimpleLinkWlanEventHandler (SlWlanEvent_t * pWlanEvent)

SimpleLink EventHandler Function.

This function

1. Is EventHandler for simplelink

Parameters



Definition at line 123 of file [wifi.c](#).

```
00124 {
00125     switch(pWlanEvent->Id) {
00126         case SL_WLAN_EVENT_CONNECT:{
00127             SET_STATUS_BIT(PowerMeasure_CB.slStatus, STATUS_BIT_CONNECTION);
00128         }
00129         break;
00130
00131         case SL_WLAN_EVENT_DISCONNECT:{
00132             CLR_STATUS_BIT(PowerMeasure_CB.slStatus, STATUS_BIT_CONNECTION);
00133             CLR_STATUS_BIT(PowerMeasure_CB.slStatus, STATUS_BIT_IP_ACQUIRED);
00134         }
00135         break;
00136
00137         default:{
00138             UART_PRINT("[WLAN EVENT] Unexpected event [0x%x]\n\r",
00139                         pWlanEvent->Id);
00140         }
00141         break;
00142     }
00143 }
```

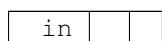
3.79.2.12 wlanConf() int32_t wlanConf (void)

SimpleLink wlan configuration Function.

This function

1. Configures wifi for CC3220SF as station

Parameters



Definition at line 242 of file [wifi.c](#).
 00242 {
 00243 }

```

00244     sl_Start(0, 0, 0);
00245     //START DEVICE
00246     sl_WlanSetMode(ROLE_STA);
00247     //SET ROLE AS STATION
00248     sl_WlanPolicySet(SL_WLAN_POLICY_PM, SL_WLAN_NORMAL_POLICY, NULL, 0);
00249     //Set PowerManagement policy to normal
00250     sl_WlanPolicySet(SL_WLAN_POLICY_CONNECTION, SL_WLAN_CONNECTION_POLICY(1, 0, 0, 0), NULL, 0);
00251     //Connection policy Auto
00252     sl_NetCfgSet(SL_NETCFG_IPV4_STA_ADDR_MODE, SL_NETCFG_ADDR_DHCP, 0, 0);
00253     //set IP DHCP
00254
00255     return(0);
00256 }

```

3.79.2.13 wlanConnect() int32_t wlanConnect (

void)

SimpleLink wlan connect Function.

This function

1. Connects to a WIFI AP.

Parameters

in		
----	--	--

Definition at line 265 of file [wifi.c](#).

```

00266 {
00267     SlWlanSecParams_t secParams = {0};
00268
00269     secParams.Key = (signed char*)SECURITY_KEY;
00270     secParams.KeyLen = strlen(SECURITY_KEY);
00271     secParams.Type = SECURITY_TYPE;
00272
00273     sl_WlanConnect((signed char*)SSID_NAME, strlen(SSID_NAME), 0, &secParams, 0);
00274     UART_PRINT("Trying to connect to AP : %s\n\r", SSID_NAME);
00275
00276     sl_Task(NULL);
00277
00278     while ((!IS_CONNECTED(PowerMeasure_CB.slStatus)) || (!IS_IP_ACQUIRED(PowerMeasure_CB.slStatus))) {
00279         sl_Task(NULL);
00280     }
00281     return(0);
00282 }

```

3.79.2.14 wlanConnectFromFile() int32_t wlanConnectFromFile (

unsigned char * ssid)

SimpleLink wlan connect from file Function.

This function

1. Connects to a WIFI AP reading the SSID from file

Parameters

in	unsigned	char *ssid Pointer to ssid
----	----------	----------------------------

Returns

SUCCESS_CONNECT_WIFI / ERROR_CONNECT_WIFI as described in [wifi.h](#)

Definition at line 296 of file wifi.c.

```

00297 {
00298     uint8_t ret;
00299     SlWlanSecParams_t secParams = {0};
00300
00301     secParams.Key = (signed char*)SECURITY_KEY_FILE_CONNECT;
00302     secParams.KeyLen = strlen(SECURITY_KEY_FILE_CONNECT);
00303     secParams.Type = SECURITY_TYPE;
00304
00305     unsigned char FileSSID[13];
00306
00307     sprintf(&FileSSID,"%s",ssid );
00308
00309     ret = sl_WlanConnect((signed char*)FileSSID, strlen(FileSSID), 0, &secParams, 0);
00310     UART_PRINT("WIFI NETWORK: %s\n\r", FileSSID);
00311
00312     sl_Task(NULL);
00313
00314     int tries=0;
00315     while((!IS_CONNECTED(PowerMeasure_CB.slStatus)) || (!IS_IP_ACQUIRED(PowerMeasure_CB.slStatus))){
00316         sl_Task(NULL);
00317
00318         if((!IS_CONNECTED(PowerMeasure_CB.slStatus)) || (!IS_IP_ACQUIRED(PowerMeasure_CB.slStatus))){
00319             tries++;
00320             UART_PRINT("%");
00321             }if(tries==20000){
00322                 Node_Disable();
00323                 return ERROR_CONNECT_WIFI;
00324             }
00325         }
00326
00327         if (ret!=0) {
00328             Node_Disable();
00329             return ERROR_CONNECT_WIFI;
00330         }else{
00331             return SUCCESS_CONNECT_WIFI;
00332         }
00333     return(0);
00334 }
```

3.80 wifi.c

```

00001 //*****
00010 //***** INCLUDES *****
00011 //***** INCLUDES *****
00012 //***** INCLUDES *****
00013 #include <ti/drivers/UART.h>
00014 #include <ti/drivers/net/wifi/simplelink.h>
00015 #include "wifi.h"
00016 #include "hal_UART.h"
00017
00018 //*****
00019 //***** FUNCTIONS *****
00020 //*****
00021
00022 //*****
00023 //
00032 //
00033 //*****
00034 void SimpleLinkNetAppRequestMemFreeEventHandler (uint8_t *buffer){}
00035
00036 //*****
00037 //
00046 //
00047 //*****
00048 void SimpleLinkNetAppRequestEventHandler(SlNetAppRequest_t *pNetAppRequest, SlNetAppResponse_t
    *pNetAppResponse){}
00049
00050 //*****
00051 //
00060 //
00061 //*****
00062 void SimpleLinkHttpServerEventHandler( SlNetAppHttpServerEvent_t *pHttpEvent,
    SlNetAppHttpServerResponse_t *pHttpResponse){}
00063
00064 //*****
00065 //
00074 //
```

```
00075 //*****
00076 void SimpleLinkSockEventHandler(SlSockEvent_t *pSock){}
00077
00078 //*****
00079 //
00080 //
00081 //*****
00082 void SimpleLinkSocketTriggerEventHandler(SlSockTriggerEvent_t *pSlTriggerEvent){}
00083
00084 //*****
00085 //
00086 //
00087 //*****
00088 void SimpleLinkGeneralEventHandler(SlDeviceEvent_t *pDevEvent)
00089 {
00090     UART_PRINT("[GENERAL EVENT] - ID=%d Sender=%d\n\n",
00091                 pDevEvent->Data.Error.Code,
00092                 pDevEvent->Data.Error.Source);
00093 }
00094
00095 //*****
00096 //
00097 //
00098 //*****
00099 void SimpleLinkWlanEventHandler(SlWlanEvent_t *pWlanEvent)
00100 {
00101     switch(pWlanEvent->Id) {
00102         case SL_WLAN_EVENT_CONNECT:{
00103             SET_STATUS_BIT(PowerMeasure_CB.slStatus, STATUS_BIT_CONNECTION);
00104         }
00105         break;
00106
00107         case SL_WLAN_EVENT_DISCONNECT:{
00108             CLR_STATUS_BIT(PowerMeasure_CB.slStatus, STATUS_BIT_CONNECTION);
00109             CLR_STATUS_BIT(PowerMeasure_CB.slStatus, STATUS_BIT_IP_ACQUIRED);
00110         }
00111         break;
00112     }
00113     default:{
00114         >Data.Error.Source);
00115     }
00116 }
```

```

00194     default:
00195         UART_PRINT("[ERROR] - FATAL ERROR:"
00196                     " Unspecified error detected \n\r");
00197     break;
00198 }
00199 }
00200 }
00201
00202 //*****
00203 //
00212 //
00213 //*****
00214 void SimpleLinkNetAppEventHandler(SlNetAppEvent_t *pNetAppEvent)
00215 {
00216     switch(pNetAppEvent->Id) {
00217         case SL_NETAPP_EVENT_IPV4_ACQUIRED:{  

00218             SET_STATUS_BIT(PowerMeasure_CB.slStatus, STATUS_BIT_IP_ACQUIRED);
00219         }
00220         break;
00221
00222         default:{  

00223             UART_PRINT("[NETAPP EVENT] Unexpected event [0x%x] \n\r",
00224                         pNetAppEvent->Id);
00225         }
00226         break;
00227     }
00228 }
00229
00230 //*****
00231 //
00240 //
00241 //*****
00242 int32_t wlanConf(void){
00243
00244     sl_Start(0, 0, 0);
00245     //START DEVICE
00246     sl_WlanSetMode(Role_STA);
00247     //SET ROLE AS STATION
00248     sl_WlanPolicySet(SL_WLAN_POLICY_PM, SL_WLAN_NORMAL_POLICY, NULL, 0);
00249     //Set PowerManagement policy to normal
00250     sl_WlanPolicySet(SL_WLAN_POLICY_CONNECTION, SL_WLAN_CONNECTION_POLICY(1, 0, 0, 0), NULL, 0);
00251     //Connection policy Auto
00252     sl_NetCfgSet(SL_NETCFG_IPV4_STA_ADDR_MODE, SL_NETCFG_ADDR_DHCP, 0, 0);
00253     //set IP DHCP
00254
00255     return(0);
00256 }
00257
00258 //*****
00259 //
00260 //
00261 //*****
00262 int32_t wlanConnect(void)
00263 {
00264     SlWlanSecParams_t secParams = {0};
00265
00266     secParams.Key = (signed char*)SECURITY_KEY;
00267     secParams.KeyLen = strlen(SECURITY_KEY);
00268     secParams.Type = SECURITY_TYPE;
00269
00270     sl_WlanConnect((signed char*)SSID_NAME, strlen(SSID_NAME), 0, &secParams, 0);
00271     UART_PRINT("Trying to connect to AP : %s\n\r", SSID_NAME);
00272
00273     sl_Task(NULL);
00274
00275     while(!IS_CONNECTED(PowerMeasure_CB.slStatus)) || (!IS_IP_ACQUIRED(PowerMeasure_CB.slStatus))){
00276         sl_Task(NULL);
00277     }
00278
00279     return(0);
00280 }
00281
00282 }
00283
00284 //*****
00285 //
00286 //
00287 //*****
00288 int32_t wlanConnectFromFile(unsigned char *ssid)
00289 {
00290     uint8_t ret;
00291     SlWlanSecParams_t secParams = {0};
00292
00293     secParams.Key = (signed char*)SECURITY_KEY_FILE_CONNECT;
00294     secParams.KeyLen = strlen(SECURITY_KEY_FILE_CONNECT);
00295     secParams.Type = SECURITY_TYPE;
00296
00297     unsigned char FileSSID[13];
00298     sprintf(&FileSSID,"%s",ssid );
00299
00300
00301
00302
00303
00304
00305
00306
00307

```

```

00308
00309     ret = sl_WlanConnect((signed char*)FileSSID, strlen(FileSSID), 0, &secParams, 0);
00310     UART_PRINT("WIFI NETWORK: %s\n\r", FileSSID);
00311
00312     sl_Task(NULL);
00313
00314     int tries=0;
00315     while((!IS_CONNECTED(PowerMeasure_CB.slStatus)) || (!IS_IP_ACQUIRED(PowerMeasure_CB.slStatus))){
00316         sl_Task(NULL);
00317
00318         if((!IS_CONNECTED(PowerMeasure_CB.slStatus)) || (!IS_IP_ACQUIRED(PowerMeasure_CB.slStatus))){
00319             tries++;
00320             UART_PRINT("*");
00321             }if(tries==20000){
00322                 Node_Disable();
00323                 return ERROR_CONNECT_WIFI;
00324             }
00325         }
00326
00327         if (ret!=0){
00328             Node_Disable();
00329             return ERROR_CONNECT_WIFI;
00330         }else{
00331             return SUCCESS_CONNECT_WIFI;
00332         }
00333     }
00334 }
00335
00336 //*****
00337 //
00347 //
00348 //*****
00349 void prepareDataFrame(uint16_t port,uint32_t ipAddr)
00350 {
00351     uint16_t idx;
00352
00353     for(idx = 0;idx < FRAME_LENGTH;idx++)
00354     {
00355         PowerMeasure_CB.frameData[idx] = (signed char)(idx % 255);
00356     }
00357     PowerMeasure_CB.ipV4Addr.sin_family = SL_AF_INET;
00358     PowerMeasure_CB.ipV4Addr.sin_port = sl_Htons(port);
00359     PowerMeasure_CB.ipV4Addr.sin_addr.s_addr = sl_Htonl(ipAddr);
00360 }
00361
00362 //*****
00363 //
00372 //
00373 //*****
00374 int16_t sendUpdClient(uint16_t port)
00375 {
00376     int16_t      sockId;
00377     int16_t      idx = 0;
00378     int32_t      status = -1;
00379
00380     sockId = sl_Socket(SL_AF_INET,SL_SOCKET_DGRAM, 0);
00381
00382     if(sockId > 0){
00383         UART_PRINT("SOCKET UDP");
00384     }else{
00385         UART_PRINT("error UDP");
00386     }
00387
00388     while (idx < NUM_OF_PKT){
00389         status = sl_SendTo(sockId,PowerMeasure_CB.frameData ,FRAME_LENGTH , 0,(SlSockAddrIn_t *)
00390 * )&PowerMeasure_CB.ipV4Addr,sizeof(SlSockAddrIn_t));
00391         if(status <= 0 ){
00392             status = sl_Close(sockId);
00393             ASSERT_ON_ERROR(sockId);
00394         }
00395         idx++;
00396     }
00397
00398     return(0);
00398 }

```

3.81 wifi.h File Reference

Functions for CC3220SF wifi management.

Data Structures

- struct `_PowerMeasure_ControlBlock_t_`
Wifi Control block. [More...](#)

Macros

- `#define ASSERT_ON_ERROR(error_code)`
- `#define CLR_STATUS_BIT(status_variable, bit) status_variable &= ~(1 << (bit))`
- `#define DEST_IP_ADDR SL_IPV4_VAL(192,168,1,160)`
- `#define ERROR_CONNECT_WIFI 1`
- `#define FRAME_LENGTH (1000)`
- `#define GET_STATUS_BIT(status_variable, bit)`
- `#define IS_CONNECTED(status_variable)`
- `#define IS_IP_ACQUIRED(status_variable)`
- `#define LOOP_FOREVER()`
- `#define NUM_OF_PKT (10)`
- `#define PORT (9000)`
- `#define PORT_UDP (9999)`
- `#define SECURITY_KEY "363232EE"`
- `#define SECURITY_KEY_FILE_CONNECT "363232EE"`
- `#define SECURITY_TYPE SL_WLAN_SEC_TYPE_WPA_WPA2`
- `#define SERIALWIFI_PORT (9000)`
- `#define SET_STATUS_BIT(status_variable, bit) status_variable |= (1 << (bit))`
- `#define SL_SOCKET_ERROR`
- `#define SSID_NAME "Orange-D62D"`
- `#define SUCCESS_CONNECT_WIFI 0`

Enumerations

- enum `e_StatusBits` { `STATUS_BIT_CONNECTION, STATUS_BIT_IP_ACQUIRED` }
Wifi Connection.

Functions

- void `prepareDataFrame` (uint16_t Port, uint32_t IpAddr)
SimpleLink Prepare data frame Function.
- void `SimpleLinkFatalErrorHandler` (SIDeviceFatal_t *slFatalErrorEvent)
SimpleLink EventHandler Function.
- void `SimpleLinkGeneralEventHandler` (SIDeviceEvent_t *pDevEvent)
SimpleLink EventHandler Function.
- void `SimpleLinkHttpServerEventHandler` (SINetAppHttpServerEvent_t *pHttpEvent, SINetAppHttpServerResponse_t *pHttpResponse)
SimpleLink EventHandler Function.
- void `SimpleLinkNetAppEventHandler` (SINetAppEvent_t *pNetAppEvent)
SimpleLink EventHandler Function.
- void `SimpleLinkNetAppRequestEventHandler` (SINetAppRequest_t *pNetAppRequest, SINetAppResponse_t *pNetAppResponse)
SimpleLink EventHandler Function.
- void `SimpleLinkNetAppRequestMemFreeEventHandler` (uint8_t *buffer)
SimpleLink EventHandler Function.

- void [SimpleLinkSocketTriggerEventHandler](#) (SISockTriggerEvent_t *pSITriggerEvent)
SimpleLink EventHandler Function.
- void [SimpleLinkSockEventHandler](#) (SISockEvent_t *pSock)
SimpleLink EventHandler Function.
- void [SimpleLinkWlanEventHandler](#) (SIWlanEvent_t *pWlanEvent)
SimpleLink EventHandler Function.
- int32_t [wlanConf](#) (void)
SimpleLink wlan configuration Function.
- int32_t [wlanConnect](#) (void)
SimpleLink wlan connect Function.
- int32_t [wlanConnectFromFile](#) (unsigned char *ssid)
SimpleLink wlan connect from file Function.

Variables

- PowerMeasure_ControlBlock [PowerMeasure_CB](#)

3.81.1 Detailed Description

Functions for CC3220SF wifi management.

Version

1.0

Date

07/05/2019

Author

A.Irizar @tittle CC3220SF wifi functions

Definition in file [wifi.h](#).

3.81.2 Data Structure Documentation

3.81.2.1 struct _PowerMeasure_ControlBlock_t_ Wifi Control block.

Definition at line [76](#) of file [wifi.h](#).

Data Fields

signed char	frameData[FRAME_LENGTH]	Wifi connection data frame
SISockAddrIn_t	ipV4Addr	IP connection control block
uint32_t	sIStatus	SimpleLink Status

3.81.3 Macro Definition Documentation

3.81.3.1 ASSERT_ON_ERROR `#define ASSERT_ON_ERROR(`
`error_code)`

Value:

```
{ \
    if(error_code < 0) \
    { \
        ERR_PRINT(error_code); \
        return error_code; \
    } \
}
```

Definition at line 48 of file [wifi.h](#).

3.81.3.2 CLR_STATUS_BIT `#define CLR_STATUS_BIT(`
`status_variable,`
`bit) status_variable &= ~(1 << (bit))`

Definition at line 36 of file [wifi.h](#).

3.81.3.3 DEST_IP_ADDR `#define DEST_IP_ADDR SL_IPV4_VAL(192, 168, 1, 160)`

Definition at line 17 of file [wifi.h](#).

3.81.3.4 ERROR_CONNECT_WIFI `#define ERROR_CONNECT_WIFI 1`

Definition at line 24 of file [wifi.h](#).

3.81.3.5 FRAME_LENGTH `#define FRAME_LENGTH (1000)`

Definition at line 19 of file [wifi.h](#).

3.81.3.6 GET_STATUS_BIT `#define GET_STATUS_BIT(`
`status_variable,`
`bit)`

Value:

```
(0 != \
(status_variable & (1 << (bit))))
```

Definition at line 37 of file [wifi.h](#).

3.81.3.7 IS_CONNECTED #define IS_CONNECTED(
 status_variable)

Value:

```
GET_STATUS_BIT( \  
    status_variable, \  
    STATUS_BIT_CONNECTION)
```

Definition at line 40 of file [wifi.h](#).

3.81.3.8 IS_IP_ACQUIRED #define IS_IP_ACQUIRED(
 status_variable)

Value:

```
GET_STATUS_BIT( \  
    status_variable, \  
    STATUS_BIT_IP_ACQUIRED)
```

Definition at line 44 of file [wifi.h](#).

3.81.3.9 LOOP_FOREVER #define LOOP_FOREVER()

Value:

```
{ \  
    while(1) {; } \  
}
```

Definition at line 31 of file [wifi.h](#).

3.81.3.10 NUM_OF_PKT #define NUM_OF_PKT (10)

Definition at line 18 of file [wifi.h](#).

3.81.3.11 PORT #define PORT (9000)

Definition at line 16 of file [wifi.h](#).

3.81.3.12 PORT_UDP #define PORT_UDP (9999)

Definition at line 20 of file [wifi.h](#).

3.81.3.13 SECURITY_KEY #define SECURITY_KEY "363232EE"

Password of the secured AP

Definition at line 28 of file [wifi.h](#).

3.81.3.14 SECURITY_KEY_FILE_CONNECT #define SECURITY_KEY_FILE_CONNECT "363232EE"

Password of the secured AP

Definition at line 29 of file [wifi.h](#).

3.81.3.15 SECURITY_TYPE #define SECURITY_TYPE SL_WLAN_SEC_TYPE_WPA_WPA2

Security type (OPEN or WEP or WPA)

Definition at line 27 of file [wifi.h](#).

3.81.3.16 SERIALWIFI_PORT #define SERIALWIFI_PORT (9000)

Definition at line 21 of file [wifi.h](#).

3.81.3.17 SET_STATUS_BIT #define SET_STATUS_BIT(

```
    status_variable,  
    bit ) status_variable |= (1 << (bit))
```

Definition at line 35 of file [wifi.h](#).

3.81.3.18 SL_SOCKET_ERROR #define SL_SOCKET_ERROR

Value:

```
( \\\n    "Socket error, please refer \"SOCKET ERRORS CODES\" section in errors.h")
```

Definition at line 57 of file [wifi.h](#).

3.81.3.19 SSID_NAME #define SSID_NAME "Orange-D62D"

AP SSID

Definition at line 26 of file [wifi.h](#).

3.81.3.20 SUCCESS_CONNECT_WIFI #define SUCCESS_CONNECT_WIFI 0

Definition at line 23 of file [wifi.h](#).

3.81.4 Enumeration Type Documentation

3.81.4.1 e_StatusBits enum e_StatusBits

Wifi Connection.

Enumerator

<code>STATUS_BIT_CONNECTION</code>	If this bit is set: the device is connected to the AP or client is connected to device (AP)
<code>STATUS_BIT_IP_ACQUIRED</code>	If this bit is set: the device has acquired an IP

Definition at line 67 of file `wifi.h`.

```
00068 {
00069     STATUS_BIT_CONNECTION,
00070     STATUS_BIT_IP_ACQUIRED,
00071 }e_StatusBits;
```

3.81.5 Function Documentation

3.81.5.1 `prepareDataFrame()` `void prepareDataFrame (`
 `uint16_t port,`
 `uint32_t ipAddr)`

SimpleLink Prepare data frame Function.

This function

1. Prepares the connection parameters for WIFI on CC3220SF

Parameters

in	<code>uint16_t</code>	port
in	<code>uint32_t</code>	ipAddr

Returns

None

Definition at line 349 of file `wifi.c`.

```
00350 {
00351     uint16_t idx;
00352
00353     for(idx = 0;idx < FRAME_LENGTH;idx++)
00354     {
00355         PowerMeasure_CB.frameData[idx] = (signed char)(idx % 255);
00356     }
00357     PowerMeasure_CB.ipV4Addr.sin_family = SL_AF_INET;
00358     PowerMeasure_CB.ipV4Addr.sin_port = sl_Htons(port);
00359     PowerMeasure_CB.ipV4Addr.sin_addr.s_addr = sl_Htonl(ipAddr);
00360 }
```

```
3.81.5.2 SimpleLinkFatalErrorHandler() void SimpleLinkFatalErrorHandler (
    SlDeviceFatal_t * sl Fatal ErrorEvent )
```

SimpleLink EventHandler Function.

This function

1. Is EventHandler for simplelink

Parameters



Definition at line 157 of file [wifi.c](#).

```
00158 {
00159     switch (slFatalErrorEvent->Id){
00160         case SL_DEVICE_EVENT_FATAL_DEVICE_ABORT:{
00161             UART_PRINT(
00162                 "[ERROR] - FATAL ERROR: Abort NWP event "
00163                 "detected: AbortType=%d, AbortData=0x%x\n\r",
00164                 slFatalErrorEvent->Data.DeviceAssert.Code,
00165                 slFatalErrorEvent->Data.DeviceAssert.Value);
00166         }
00167         break;
00168
00169         case SL_DEVICE_EVENT_FATAL_DRIVER_ABORT:{
00170             UART_PRINT("[ERROR] - FATAL ERROR: Driver Abort detected. \n\r");
00171         }
00172         break;
00173
00174         case SL_DEVICE_EVENT_FATAL_NO_CMD_ACK:{
00175             UART_PRINT(
00176                 "[ERROR] - FATAL ERROR: "
00177                 "No Cmd Ack detected [cmd opcode = 0x%x] \n\r",
00178                 slFatalErrorEvent->Data.NoCmdAck.Code);
00179         }
00180         break;
00181
00182         case SL_DEVICE_EVENT_FATAL_SYNC_LOSS:{
00183             UART_PRINT("[ERROR] - FATAL ERROR: Sync loss detected \n\r");
00184         }
00185         break;
00186
00187         case SL_DEVICE_EVENT_FATAL_CMD_TIMEOUT:{
00188             UART_PRINT(
00189                 "[ERROR] - FATAL ERROR: Async event timeout "
00190                 "detected [event opcode =0x%x] \n\r",
00191                 slFatalErrorEvent->Data.CmdTimeout.Code);
00192         }
00193         break;
00194
00195         default:
00196             UART_PRINT("[ERROR] - FATAL ERROR:"
00197                         " Unspecified error detected \n\r");
00198         break;
00199     }
00200 }
```

```
3.81.5.3 SimpleLinkGeneralEventHandler() void SimpleLinkGeneralEventHandler (
    SlDeviceEvent_t * pDevEvent )
```

SimpleLink EventHandler Function.

This function

1. Is EventHandler for simplelink

Parameters

Definition at line 104 of file [wifi.c](#).

```
00105 {  
00106     UART_PRINT("[GENERAL EVENT] - ID=%d Sender=%d\n\n",  
00107         pDevEvent->Data.Error.Code,  
00108         pDevEvent->Data.Error.Source);  
00109 }
```

3.81.5.4 SimpleLinkHttpServerEventHandler() void SimpleLinkHttpServerEventHandler (SlNetAppHttpServerEvent_t * pHttEvent, SlNetAppHttpServerResponse_t * pHttResponse)

SimpleLink EventHandler Function.

This function

1. Is EventHandler for simplelink

Parameters

Definition at line 62 of file [wifi.c](#).

```
00062 {}
```

3.81.5.5 SimpleLinkNetAppEventHandler() void SimpleLinkNetAppEventHandler (SlNetAppEvent_t * pNetAppEvent)

SimpleLink EventHandler Function.

This function

1. Is EventHandler for simplelink

Parameters

Definition at line 214 of file [wifi.c](#).

```
00215 {  
00216     switch(pNetAppEvent->Id){  
00217         case SL_NETAPP_EVENT_IPV4_ACQUIRED:{  
00218             SET_STATUS_BIT(PowerMeasure_CB.slStatus, STATUS_BIT_IP_ACQUIRED);  
00219         }  
00220         break;  
00221 }
```

```

00222     default: {
00223         UART_PRINT("[NETAPP EVENT] Unexpected event [0x%lx] \n\r",
00224             pNetAppEvent->Id);
00225     }
00226     break;
00227 }
00228 }
```

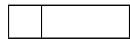
3.81.5.6 SimpleLinkNetAppRequestEventHandler() void SimpleLinkNetAppRequestEventHandler (S1NetAppRequest_t * pNetAppRequest, S1NetAppResponse_t * pNetAppResponse)

SimpleLink EventHandler Function.

This function

1. Is EventHandler for simplelink

Parameters



Definition at line 48 of file [wifi.c](#).

```
00048 { }
```

3.81.5.7 SimpleLinkNetAppRequestMemFreeEventHandler() void SimpleLinkNetAppRequestMemFreeEventHandler (uint8_t * buffer)

SimpleLink EventHandler Function.

This function

1. Is EventHandler for simplelink

Parameters



Definition at line 34 of file [wifi.c](#).

```
00034 { }
```

3.81.5.8 SimpleLinkSocketTriggerEventHandler() void SimpleLinkSocketTriggerEventHandler (S1SockTriggerEvent_t * pS1TriggerEvent)

SimpleLink EventHandler Function.

This function

1. Is EventHandler for simplelink

Parameters

Definition at line 90 of file [wifi.c](#).

```
00090 { }
```

3.81.5.9 SimpleLinkSockEventHandler() void SimpleLinkSockEventHandler (

```
    SlSockEvent_t * pSock )
```

SimpleLink EventHandler Function.

This function

1. Is EventHandler for simplelink

Parameters

Definition at line 76 of file [wifi.c](#).

```
00076 { }
```

3.81.5.10 SimpleLinkWlanEventHandler() void SimpleLinkWlanEventHandler (

```
    SlWlanEvent_t * pWlanEvent )
```

SimpleLink EventHandler Function.

This function

1. Is EventHandler for simplelink

Parameters

Definition at line 123 of file [wifi.c](#).

```
00124 {  
00125     switch (pWlanEvent->Id) {  
00126         case SL_WLAN_EVENT_CONNECT:{  
00127             SET_STATUS_BIT(PowerMeasure_CB.slStatus, STATUS_BIT_CONNECTION);  
00128         }  
00129         break;  
00130     case SL_WLAN_EVENT_DISCONNECT:{  
00131         CLR_STATUS_BIT(PowerMeasure_CB.slStatus, STATUS_BIT_CONNECTION);  
00132     }
```

```

00133     CLR_STATUS_BIT(PowerMeasure_CB.siStatus, STATUS_BIT_IP_ACQUIRED);
00134 }
00135 break;
00136
00137 default:{
00138     UART_PRINT("[WLAN EVENT] Unexpected event [0x%x]\n\r",
00139                 pWlanEvent->Id);
00140 }
00141 break;
00142 }
00143 }
```

3.81.5.11 wlanConf() int32_t wlanConf (void)

SimpleLink wlan configuration Function.

This function

1. Configures wifi for CC3220SF as station

Parameters

in		
----	--	--

Definition at line 242 of file [wifi.c](#).

```

00242 {
00243
00244     sl_Start(0, 0, 0);
00245     //START DEVICE
00246     sl_WlanSetMode(ROLE_STA);
00247     //SET ROLE AS STATION
00248     sl_WlanPolicySet(SL_WLAN_POLICY_PM , SL_WLAN_NORMAL_POLICY, NULL, 0);
00249     //Set PowerManagement policy to normal
00250     sl_WlanPolicySet(SL_WLAN_POLICY_CONNECTION, SL_WLAN_CONNECTION_POLICY(1, 0, 0, 0), NULL, 0);
00251     //Connection policy Auto
00252     sl_NetCfgSet(SL_NETCFG_IPV4_STA_ADDR_MODE, SL_NETCFG_ADDR_DHCP, 0, 0);
00253     //set IP DHCP
00254
00255     return(0);
00256 }
```

3.81.5.12 wlanConnect() int32_t wlanConnect (void)

SimpleLink wlan connect Function.

This function

1. Connects to a WIFI AP.

Parameters

in		
----	--	--

Definition at line 265 of file [wifi.c](#).

```
00266 {
00267     SlWlanSecParams_t secParams = {0};
00268
00269     secParams.Key = (signed char*)SECURITY_KEY;
00270     secParams.KeyLen = strlen(SECURITY_KEY);
00271     secParams.Type = SECURITY_TYPE;
00272
00273     sl_WlanConnect((signed char*)SSID_NAME, strlen(SSID_NAME), 0, &secParams, 0);
00274     UART_PRINT("Trying to connect to AP : %s\n\r", SSID_NAME);
00275
00276     sl_Task(NULL);
00277
00278     while((!IS_CONNECTED(PowerMeasure_CB.slStatus)) || (!IS_IP_ACQUIRED(PowerMeasure_CB.slStatus))) {
00279         sl_Task(NULL);
00280     }
00281     return(0);
00282 }
```

3.81.5.13 wlanConnectFromFile() int32_t wlanConnectFromFile (unsigned char * ssid)

SimpleLink wlan connect from file Function.

This function

1. Connects to a WIFI AP reading the SSID from file

Parameters

in	<i>unsigned</i>	char *ssid Pointer to ssid
----	-----------------	----------------------------

Returns

SUCCESS_CONNECT_WIFI / ERROR_CONNECT_WIFI as described in [wifi.h](#)

Definition at line 296 of file [wifi.c](#).

```
00297 {
00298     uint8_t ret;
00299     SlWlanSecParams_t secParams = {0};
00300
00301     secParams.Key = (signed char*)SECURITY_KEY_FILE_CONNECT;
00302     secParams.KeyLen = strlen(SECURITY_KEY_FILE_CONNECT);
00303     secParams.Type = SECURITY_TYPE;
00304
00305     unsigned char FileSSID[13];
00306
00307     sprintf(&FileSSID,"%s",ssid );
00308
00309     ret = sl_WlanConnect((signed char*)FileSSID, strlen(FileSSID), 0, &secParams, 0);
00310     UART_PRINT("WIFI NETWORK: %s\n\r", FileSSID);
00311
00312     sl_Task(NULL);
00313
00314     int tries=0;
00315     while((!IS_CONNECTED(PowerMeasure_CB.slStatus)) || (!IS_IP_ACQUIRED(PowerMeasure_CB.slStatus))) {
00316         sl_Task(NULL);
00317
00318         if((!IS_CONNECTED(PowerMeasure_CB.slStatus)) || (!IS_IP_ACQUIRED(PowerMeasure_CB.slStatus))) {
00319             tries++;
00320             UART_PRINT("*");
00321         }if(tries==20000) {
00322             Node_Disable();
00323             return ERROR_CONNECT_WIFI;
00324         }
00325     }
00326 }
```

```

00327     if (ret!=0) {
00328         Node_Disable();
00329         return ERROR_CONNECT_WIFI;
00330     }else{
00331         return SUCCESS_CONNECT_WIFI;
00332     }
00333     return(0);
00334 }
```

3.81.6 Variable Documentation

3.81.6.1 PowerMeasure_CB PowerMeasure_ControlBlock PowerMeasure_CB

Definition at line 86 of file [wifi.h](#).

3.82 wifi.h

```

00001
00010 #ifndef WIFI_H_
00011 #define WIFI_H_
00012
00013 //*****
00014 //      DEFINES
00015 //*****
00016 #define PORT          (9000)
00017 #define DEST_IP_ADDR   SL_IPV4_VAL(192,168,1,160)
00018 #define NUM_OF_PKT     (10)
00019 #define FRAME_LENGTH   (1000)
00020 #define PORT_UDP       (9999)
00021 #define SERIALWIFI_PORT (9000)
00022
00023 #define SUCCESS_CONNECT_WIFI 0
00024 #define ERROR_CONNECT_WIFI   1
00025
00026 #define SSID_NAME          "Orange-D62D"
00027 #define SECURITY_TYPE      SL_WLAN_SEC_TYPE_WPA_WPA2
00028 #define SECURITY_KEY        "363232EE"
00029 #define SECURITY_KEY_FILE_CONNECT "363232EE"
00031 #define LOOP_FOREVER() \
00032     { \
00033     while(1) {; } \
00034     }
00035 #define SET_STATUS_BIT(status_variable, bit) status_variable |= (1 << (bit))
00036 #define CLR_STATUS_BIT(status_variable, bit) status_variable &= ~(1 << (bit))
00037 #define GET_STATUS_BIT(status_variable, bit) (0 != \
00038                                         (status_variable & (1 << (bit))))
00039
00040 #define IS_CONNECTED(status_variable)      GET_STATUS_BIT( \
00041             status_variable, \
00042             STATUS_BIT_CONNECTION)
00043
00044 #define IS_IP_ACQUIRED(status_variable)    GET_STATUS_BIT( \
00045             status_variable, \
00046             STATUS_BIT_IP_ACQUIRED)
00047
00048 #define ASSERT_ON_ERROR(error_code) \
00049     { \
00050     if(error_code < 0) \
00051     { \
00052         ERR_PRINT(error_code); \
00053         return error_code; \
00054     } \
00055 }
00056
00057 #define SL_SOCKET_ERROR      ( \
00058     "Socket error, please refer \"SOCKET ERRORS CODES\" section in errors.h")
00059
00060 //*****
00061 //      TYPEDEFS
00062 //*****
00063
00067 typedef enum
00068 {
00069     STATUS_BIT_CONNECTION,
```

```
00070     STATUS_BIT_IP_ACQUIRED,
00071 }e_StatusBits;
00072
00076 typedef struct _PowerMeasure_ControlBlock_t_
00077 {
00078     uint32_t          slStatus;                                //SimpleLink Status
00079     signed char       frameData[FRAME_LENGTH];
00080     SlSockAddrIn_t   ipV4Addr;
00081 }PowerMeasure_ControlBlock;
00082
00083 //*****
00084 //          GLOBALS
00085 //*****
00086 PowerMeasure_ControlBlock  PowerMeasure_CB;
00087
00088 //*****
00089 //          FUNCTION PROTOTYPES
00090 //*****
00091 int32_t wlanConnect(void);
00092 int32_t wlanConnectFromFile(unsigned char *ssid);
00093 int32_t wlanConf(void);
00094 void SimpleLinkNetAppRequestMemFreeEventHandler (uint8_t *buffer);
00095 void SimpleLinkNetAppRequestEventHandler(SlNetAppRequest_t *pNetAppRequest, SlNetAppResponse_t
    *pNetAppResponse);
00096 void SimpleLinkWlanEventHandler(SlWlanEvent_t *pWlanEvent);
00097 void SimpleLinkFatalErrorHandler(SlDeviceFatal_t *slFatalErrorEvent);
00098 void SimpleLinkNetAppEventHandler(SlNetAppEvent_t *pNetAppEvent);
00099 void SimpleLinkHttpServerEventHandler( SlNetAppHttpServerEvent_t *pHttpEvent,
    SlNetAppHttpServerResponse_t *pHttpResponse);
00100 void SimpleLinkGeneralEventHandler(SlDeviceEvent_t *pDevEvent);
00101 void SimpleLinkSockEventHandler(SlSockEvent_t *pSock);
00102 void SimpleLinkSocketTriggerEventHandler(SlSockTriggerEvent_t *pSlTriggerEvent);
00103 void prepareDataFrame(uint16_t Port, uint32_t IpAddr);
00104
00105 #endif /* WIFI_H_ */
```

Index

_PowerMeasure_ControlBlock_t, 410
ACT_COUNT
 ADXL355.h, 27
ACT_EN
 ADXL355.h, 27
ACT_THRESH_H
 ADXL355.h, 27
ACT_THRESH_L
 ADXL355.h, 27
ACT_X
 ADXL355.h, 24
ACT_Y
 ADXL355.h, 24
ACT_Z
 ADXL355.h, 24
ACTIVE_MODE
 LDC1000.h, 342
ACTIVITY
 ADXL355.h, 28
ADC_config
 CC3220SF_STARPORTS.c, 94
ADC_count
 CC3220SF_STARPORTS.c, 94
adcCC3220SHWAttrs
 CC3220SF_STARPORTS.c, 94
adcCC3220SObjects
 CC3220SF_STARPORTS.c, 95
Add_s16Data2Packet
 Sensors.c, 362
 Sensors.h, 372
Add_s32Data2Packet
 Sensors.c, 363
 Sensors.h, 373
ADXL355.c, 4
 ADXL355_Data_Rdy, 5
 ADXL355_DevId, 6
 ADXL355_Fifo_Data, 6
 ADXL355_Fifo_Entries, 7
 ADXL355_Fifo_Full, 8
 ADXL355_Fifo_Samples, 8
 ADXL355_Filter_Conf, 10
 ADXL355_Get_Accel_Frame, 10
 ADXL355_PartId, 11
 ADXL355_Power_Conf, 12
 ADXL355_Range_Conf, 12
 ADXL355_Reset, 13
 ADXL355_Temp, 13
 ADXL355_XData, 14
 ADXL355_YData, 14
 ADXL355_ZData, 16
 u20_to_s32, 16
ADXL355.h, 20
 ACT_COUNT, 27
 ACT_EN, 27
 ACT_THRESH_H, 27
 ACT_THRESH_L, 27
 ACT_X, 24
 ACT_Y, 24
 ACT_Z, 24
 ACTIVITY, 28
 ADXL355_DEVICE_AD, 23
 ADXL355_act_ctl_t, 23
 ADXL355_Data_Rdy, 29
 ADXL355_DevId, 30
 ADXL355_Fifo_Data, 30
 ADXL355_Fifo_Entries, 31
 ADXL355_Fifo_Full, 32
 ADXL355_Fifo_Samples, 32
 ADXL355_Filter_Conf, 33
 ADXL355_filter_ctl_t, 24
 ADXL355_Get_Accel_Frame, 33
 ADXL355_i2c_speed_t, 25
 ADXL355_int_ctl_t, 25
 ADXL355_int_pol_t, 25
 ADXL355_intmap_ctl_t, 26
 ADXL355_modes_t, 26
 ADXL355_PartId, 34
 ADXL355_Power_Conf, 35
 ADXL355_Range_Conf, 35
 ADXL355_range_ctl_t, 26
 ADXL355_register_t, 27
 ADXL355_RESET, 23
 ADXL355_Reset, 36
 ADXL355_RESET_REG, 28
 ADXL355_STATUS, 27
 ADXL355_status_t, 28
 ADXL355_sync_ctl_t, 29
 ADXL355_Temp, 36
 ADXL355_XData, 37
 ADXL355_YData, 37
 ADXL355_ZData, 38
 axis355_sens, 39
 calib_data, 39
 DATA_RDY, 28
 DEVID_AD, 27
 DEVID_MST, 27
 DRDY_OFF, 26
 EXT_CLK, 29
 EXT_SYNC_INT, 29
 EXT_SYNC_NO_INT, 29
 FIFO_DATA, 27
 FIFO_ENTRIES, 27
 FIFO_FULL, 28
 FIFO_OVR, 28
 FIFO_SAMPLES, 27
 FILTER, 27
 FULL_EN, 26
 HPF02, 24
 HPF09, 24

HPF15, 24
HPF247, 24
HPF3, 24
HPF62, 24
HPFOFF, 24
I2C_FAST_MODE, 25
I2C_HIGH_SPEED, 25
INT_MAP, 28
INT_POL_HIGH, 25
INT_POL_LOW, 25
INT_SYNC, 29
MEASUREMENT, 26
NVM_BUSY, 28
ODR1000HZ, 24
ODR125Hz, 24
ODR15Hz, 24
ODR2000HZ, 24
ODR250HZ, 24
ODR31Hz, 24
ODR3HZ, 24
ODR4000HZ, 24
ODR500HZ, 24
ODR62HZ, 24
ODR7Hz, 24
OFFSET_X_H, 27
OFFSET_X_L, 27
OFFSET_Y_H, 27
OFFSET_Y_L, 27
OFFSET_Z_H, 27
OFFSET_Z_L, 27
OVR_EN, 26
PARTID, 27
POWER_CTL, 28
RANGE, 28
RANGE2G, 27
RANGE4G, 27
RANGE8G, 27
RDY_EN, 26
REVID, 27
SELF_TEST, 28
ST1, 25
ST2, 25
STANDBY, 26
SYNC, 28
TEMP1, 27
TEMP2, 27
TEMP_OFF, 26
u20_to_s32, 38
XDATA1, 27
XDATA2, 27
XDATA3, 27
YDATA1, 27
YDATA2, 27
YDATA3, 27
ZDATA1, 27
ZDATA2, 27
ZDATA3, 27
ADXL355__DEVICE_AD
ADXL355.h, 23
ADXL355_act_ctl_t
ADXL355.h, 23
ADXL355_calibdata_t, 23
ADXL355_Data, 389
ADXL355_Data_Rdy
ADXL355.c, 5
ADXL355.h, 29
ADXL355_DevId
ADXL355.c, 6
ADXL355.h, 30
ADXL355_Disable
hal_GPIO.c, 182
hal_GPIO.h, 194
ADXL355_Enable
hal_GPIO.c, 183
hal_GPIO.h, 195
ADXL355_Fifo_Data
ADXL355.c, 6
ADXL355.h, 30
ADXL355_Fifo_Entries
ADXL355.c, 7
ADXL355.h, 31
ADXL355_Fifo_Full
ADXL355.c, 8
ADXL355.h, 32
ADXL355_Fifo_Samples
ADXL355.c, 8
ADXL355.h, 32
ADXL355_Filter_Conf
ADXL355.c, 10
ADXL355.h, 33
ADXL355_filter_ctl_t
ADXL355.h, 24
ADXL355_Get_Accel_Frame
ADXL355.c, 10
ADXL355.h, 33
ADXL355_i2c_speed_t
ADXL355.h, 25
ADXL355_ID
STARPORTS_App.h, 391
ADXL355_int_ctl_t
ADXL355.h, 25
ADXL355_int_pol_t
ADXL355.h, 25
ADXL355_intmap_ctl_t
ADXL355.h, 26
ADXL355_modes_t
ADXL355.h, 26
ADXL355_PartId
ADXL355.c, 11
ADXL355.h, 34
ADXL355_Power_Conf
ADXL355.c, 12
ADXL355.h, 35
ADXL355_Range_Conf
ADXL355.c, 12
ADXL355.h, 35

ADXL355_range_ctl_t
 ADXL355.h, 26

ADXL355_register_t
 ADXL355.h, 27

ADXL355_RESET
 ADXL355.h, 23

ADXL355_Reset
 ADXL355.c, 13
 ADXL355.h, 36

ADXL355_RESET_REG
 ADXL355.h, 28

ADXL355_SPI_Disable
 hal_GPIO.c, 183
 hal_GPIO.h, 195

ADXL355_SPI_Enable
 hal_GPIO.c, 183
 hal_GPIO.h, 195

ADXL355_STATUS
 ADXL355.h, 27

ADXL355_status_t
 ADXL355.h, 28

ADXL355_sync_ctl_t
 ADXL355.h, 29

ADXL355_Temp
 ADXL355.c, 13
 ADXL355.h, 36

ADXL355_XData
 ADXL355.c, 14
 ADXL355.h, 37

ADXL355_YData
 ADXL355.c, 14
 ADXL355.h, 37

ADXL355_ZData
 ADXL355.c, 16
 ADXL355.h, 38

AMP_1V
 LDC1000.h, 340

AMP_2V
 LDC1000.h, 340

AMP_4V
 LDC1000.h, 340

APPEUI
 STARPORTS_App.h, 391

APPKEY
 STARPORTS_App.h, 391

APPSKEY
 STARPORTS_App.h, 391

ASSERT_ON_ERROR
 wifi.h, 411

axis355_sens
 ADXL355.h, 39

BME280.c, 41

 BME280_DevId, 43

 BME280_Read_Calib, 43

 BME280_Read_Config, 44

 BME280_Read_Ctrl_Hum, 44

 BME280_Read_Ctrl_Meas, 45

 BME280_Read_Humidity, 45

 BME280_Read_Pressure, 46

 BME280_Read_Status, 47

 BME280_Read_Temperature, 47

 BME280_Reset, 48

 BME280_Write_Config, 48

 BME280_Write_Ctrl_Hum, 49

 BME280_Write_Ctrl_Meas, 49

 compensate_humidity, 50

 compensate_pressure, 51

 compensate_temperature, 52

 BME280.h, 56

 BME280_DEVICE_AD, 59

 BME280_DevId, 64

 BME280_filter_t, 60

 BME280_modes_t, 61

 BME280_osrs_h_t, 61

 BME280_osrs_p_t, 62

 BME280_osrs_t_t, 62

 BME280_Read_Calib, 65

 BME280_Read_Config, 66

 BME280_Read_Ctrl_Hum, 66

 BME280_Read_Ctrl_Meas, 67

 BME280_Read_Humidity, 67

 BME280_Read_Pressure, 68

 BME280_Read_Status, 68

 BME280_Read_Temperature, 69

 BME280_register_t, 62

 BME280_RESET, 59

 BME280_Reset, 69

 BME280_RESET_REG, 63

 BME280_spi3w_t, 63

 BME280_STATUS, 63

 BME280_status_t, 63

 BME280_t_sb_t, 64

 BME280_Write_Config, 70

 BME280_Write_Ctrl_Hum, 70

 BME280_Write_Ctrl_Meas, 71

 CALIB00, 63

 CALIB00_NDATA, 59

 CALIB26, 63

 CALIB26_NDATA, 59

 compensate_humidity, 71

 compensate_pressure, 72

 compensate_temperature, 73

 CONFIG, 63

 CTRL_HUM, 63

 CTRL_MEAS, 63

 DEVID, 63

 FILTER_16, 61

 FILTER_2, 61

 FILTER_4, 61

 FILTER_8, 61

 FILTER_OFF, 61

 FORCED, 61

 HUM_LSB, 63

 HUM_MSB, 63

 MEASURING, 64

 NORMAL, 61

OSRS_HX1, 61
 OSRS_HX16, 61
 OSRS_HX2, 61
 OSRS_HX4, 61
 OSRS_HX8, 61
 OSRS_PX1, 62
 OSRS_PX16, 62
 OSRS_PX2, 62
 OSRS_PX4, 62
 OSRS_PX8, 62
 OSRS_TX1, 62
 OSRS_TX16, 62
 OSRS_TX2, 62
 OSRS_TX4, 62
 OSRS_TX8, 62
 PRESS_LSB, 63
 PRESS_MSB, 63
 PRESS_XLSB, 63
 SLEEP, 61
 SPI3W_OFF, 63
 SPI3W_ON, 63
 T_SB_0p5, 64
 T_SB_10, 64
 T_SB_1000, 64
 T_SB_125, 64
 T_SB_20, 64
 T_SB_250, 64
 T_SB_500, 64
 T_SB_62p5, 64
 TEMP_LSB, 63
 TEMP_MSB, 63
 TEMP_XLSB, 63
 UPDATE, 64
 bme280_calib_data, 58
 BME280_Data, 390
 BME280_DEVICE_AD
 BME280.h, 59
 BME280_DevId
 BME280.c, 43
 BME280.h, 64
 BME280_Disable
 hal_GPIO.c, 184
 hal_GPIO.h, 196
 BME280_Enable
 hal_GPIO.c, 184
 hal_GPIO.h, 196
 BME280_filter_t
 BME280.h, 60
 BME280_ID
 STARPORTS_App.h, 391
 BME280_modes_t
 BME280.h, 61
 BME280_osrs_h_t
 BME280.h, 61
 BME280_osrs_p_t
 BME280.h, 62
 BME280_osrs_t_t
 BME280.h, 62
 BME280_Read_Calib
 BME280.c, 43
 BME280.h, 65
 BME280_Read_Config
 BME280.c, 44
 BME280.h, 66
 BME280_Read_Ctrl_Hum
 BME280.c, 44
 BME280.h, 66
 BME280_Read_Ctrl_Meas
 BME280.c, 45
 BME280.h, 67
 BME280_Read_Humidity
 BME280.c, 45
 BME280.h, 67
 BME280_Read_Pressure
 BME280.c, 46
 BME280.h, 68
 BME280_Read_Status
 BME280.c, 47
 BME280.h, 68
 BME280_Read_Temperature
 BME280.c, 47
 BME280.h, 69
 BME280_register_t
 BME280.h, 62
 BME280_RESET
 BME280.h, 59
 BME280_Reset
 BME280.c, 48
 BME280.h, 69
 BME280_RESET_REG
 BME280.h, 63
 BME280_spi3w_t
 BME280.h, 63
 BME280_SPI_Disable
 hal_GPIO.c, 185
 hal_GPIO.h, 197
 BME280_SPI_Enable
 hal_GPIO.c, 185
 hal_GPIO.h, 197
 BME280_STATUS
 BME280.h, 63
 BME280_status_t
 BME280.h, 63
 BME280_t_sb_t
 BME280.h, 64
 BME280_Write_Config
 BME280.c, 48
 BME280.h, 70
 BME280_Write_Ctrl_Hum
 BME280.c, 49
 BME280.h, 70
 BME280_Write_Ctrl_Meas
 BME280.c, 49
 BME280.h, 71
 Board.h, 76
 Board_ADC0, 77

Board_ADXL355_CS, 77
Board_BME280_CS, 77
Board_CAPTURE0, 77
Board_CAPTURE1, 77
Board_CC3220SF_STARPORTS, 77
Board_CRYPTO0, 78
Board_CS_ASGPIO, 78
Board_DS1374_INTB, 78
Board_EN_ADXL355, 78
Board_EN_BME280, 78
Board_EN_LDC1000, 78
Board_EN_NODE, 78
Board_GPIO_LED0, 78
Board_GPIO_LED_OFF, 79
Board_GPIO_LED_ON, 79
Board_I2C0, 79
Board_I2C_TMP, 79
Board_I2C_TMP006_ADDR, 79
Board_I2S0, 79
Board_init, 79
Board_initGeneral, 79
Board_LDC1000_CS, 80
Board_MOSI_ASGPIO, 80
Board_PWM0, 80
Board_PWM1, 80
Board_RN2483_MCLR, 80
Board_SCL_ASGPIO, 80
Board_SCLK_ASGPIO, 80
Board_SD0, 80
Board_SDA_ASGPIO, 81
Board_SDFFatFS0, 81
Board_SPI0, 81
Board_SPI_MASTER, 81
Board_SPI_MASTER_READY, 81
Board_SPI_SLAVE, 81
Board_SPI_SLAVE_READY, 81
Board_TIMER0, 81
Board_TIMER1, 82
Board_TIMER2, 82
Board_TMP006_ADDR, 82
Board_UART0, 82
Board_UART1, 82
Board_WATCHDOG0, 82
Board_ADC0
 Board.h, 77
Board_ADXL355_CS
 Board.h, 77
Board_BME280_CS
 Board.h, 77
Board_CAPTURE0
 Board.h, 77
Board_CAPTURE1
 Board.h, 77
Board_CC3220SF_STARPORTS
 Board.h, 77
Board_CRYPTO0
 Board.h, 78
Board_CS_ASGPIO
 Board.h, 78
Board_DS1374_INTB
 Board.h, 78
Board_EN_ADXL355
 Board.h, 78
Board_EN_BME280
 Board.h, 78
Board_EN_LDC1000
 Board.h, 78
Board_EN_NODE
 Board.h, 78
Board_GPIO_LED0
 Board.h, 78
Board_GPIO_LED_OFF
 Board.h, 79
Board_GPIO_LED_ON
 Board.h, 79
Board_I2C0
 Board.h, 79
Board_I2C_TMP
 Board.h, 79
Board_I2C_TMP006_ADDR
 Board.h, 79
Board_I2S0
 Board.h, 79
Board_init
 Board.h, 79
Board_initGeneral
 Board.h, 79
Board_LDC1000_CS
 Board.h, 80
Board_MOSI_ASGPIO
 Board.h, 80
Board_PWM0
 Board.h, 80
Board_PWM1
 Board.h, 80
Board_RN2483_MCLR
 Board.h, 80
Board_SCL_ASGPIO
 Board.h, 80
Board_SCLK_ASGPIO
 Board.h, 80
Board_SD0
 Board.h, 80
Board_SDA_ASGPIO
 Board.h, 81
Board_SDFFatFS0
 Board.h, 81
Board_SPI0
 Board.h, 81
Board_SPI_MASTER
 Board.h, 81
Board_SPI_MASTER_READY
 Board.h, 81
Board_SPI_SLAVE
 Board.h, 81
Board_SPI_SLAVE_READY
 Board.h, 81

Board.h, 81
 Board_TIMER0
 Board.h, 81
 Board_TIMER1
 Board.h, 82
 Board_TIMER2
 Board.h, 82
 Board_TMP006_ADDR
 Board.h, 82
 Board_UART0
 Board.h, 82
 Board_UART1
 Board.h, 82
 Board_WATCHDOG0
 Board.h, 82

 CALIB00
 BME280.h, 63
 CALIB00_NDATA
 BME280.h, 59
 CALIB26
 BME280.h, 63
 CALIB26_NDATA
 BME280.h, 59
 calib_data
 ADXL355.h, 39
 Capture_config
 CC3220SF_STARPORTS.c, 95
 Capture_count
 CC3220SF_STARPORTS.c, 95
 captureCC3220SFHWAttrs
 CC3220SF_STARPORTS.c, 95
 captureCC3220SFObjets
 CC3220SF_STARPORTS.c, 95
 CC3220SF_LAUNCHXL.h, 84
 CC3220SF_LAUNCHXL_ADC0, 85
 CC3220SF_LAUNCHXL_ADC1, 85
 CC3220SF_LAUNCHXL_ADCCOUNT, 85
 CC3220SF_LAUNCHXL_ADCName, 85
 CC3220SF_LAUNCHXL_CAPTURE0, 86
 CC3220SF_LAUNCHXL_CAPTURE1, 86
 CC3220SF_LAUNCHXL_CAPTURECOUNT, 86
 CC3220SF_LAUNCHXL_CaptureName, 85
 CC3220SF_LAUNCHXL_CRYPTO0, 86
 CC3220SF_LAUNCHXL_CRYPTOCOUNT, 86
 CC3220SF_LAUNCHXL_CryptoName, 86
 CC3220SF_LAUNCHXL_GPIO_LED_D10, 86
 CC3220SF_LAUNCHXL_GPIO_LED_OFF, 85
 CC3220SF_LAUNCHXL_GPIO_LED_ON, 85
 CC3220SF_LAUNCHXL_GPIO_SW2, 86
 CC3220SF_LAUNCHXL_GPIO_SW3, 86
 CC3220SF_LAUNCHXL_GPIO_TMP116_EN, 86
 CC3220SF_LAUNCHXL_GPIOCOUNT, 86
 CC3220SF_LAUNCHXL_GPIOName, 86
 CC3220SF_LAUNCHXL_I2C0, 87
 CC3220SF_LAUNCHXL_I2CCOUNT, 87
 CC3220SF_LAUNCHXL_I2CName, 87
 CC3220SF_LAUNCHXL_I2S0, 87
 CC3220SF_LAUNCHXL_I2SCOUNT, 87
 CC3220SF_LAUNCHXL_I2SName, 87
 CC3220SF_LAUNCHXL_initGeneral, 90
 CC3220SF_LAUNCHXL_LCD_CS, 86
 CC3220SF_LAUNCHXL_LCD_ENABLE, 86
 CC3220SF_LAUNCHXL_LCD_POWER, 86
 CC3220SF_LAUNCHXL_PWM6, 87
 CC3220SF_LAUNCHXL_PWM7, 87
 CC3220SF_LAUNCHXL_PWMCOUNT, 87
 CC3220SF_LAUNCHXL_PWMName, 87
 CC3220SF_LAUNCHXL_SD0, 88
 CC3220SF_LAUNCHXL_SDCOUNT, 88
 CC3220SF_LAUNCHXL_SDFatFS0, 88
 CC3220SF_LAUNCHXL_SDFatFSCOUNT, 88
 CC3220SF_LAUNCHXL_SDFatFSName, 88
 CC3220SF_LAUNCHXL_SDName, 88
 CC3220SF_LAUNCHXL_SPI0, 88
 CC3220SF_LAUNCHXL_SPI1, 88
 CC3220SF_LAUNCHXL_SPI_MASTER_READY, 86
 CC3220SF_LAUNCHXL_SPI_SLAVE_READY, 86
 CC3220SF_LAUNCHXL_SPICOUNT, 88
 CC3220SF_LAUNCHXL_SPIName, 88
 CC3220SF_LAUNCHXL_TIMER0, 89
 CC3220SF_LAUNCHXL_TIMER1, 89
 CC3220SF_LAUNCHXL_TIMER2, 89
 CC3220SF_LAUNCHXL_TIMERCOUNT, 89
 CC3220SF_LAUNCHXL_TimerName, 89
 CC3220SF_LAUNCHXL_UART0, 89
 CC3220SF_LAUNCHXL_UART1, 89
 CC3220SF_LAUNCHXL_UARTCOUNT, 89
 CC3220SF_LAUNCHXL_UARTName, 89
 CC3220SF_LAUNCHXL_WATCHDOG0, 89
 CC3220SF_LAUNCHXL_WATCHDOGCOUNT, 89
 CC3220SF_LAUNCHXL_WatchdogName, 89
 CC3220SF_LAUNCHXL_ADC0
 CC3220SF_LAUNCHXL.h, 85
 CC3220SF_LAUNCHXL_ADC1
 CC3220SF_LAUNCHXL.h, 85
 CC3220SF_LAUNCHXL_ADCCOUNT
 CC3220SF_LAUNCHXL.h, 85
 CC3220SF_LAUNCHXL_ADCName
 CC3220SF_LAUNCHXL.h, 85
 CC3220SF_LAUNCHXL_CAPTURE0
 CC3220SF_LAUNCHXL.h, 86
 CC3220SF_LAUNCHXL_CAPTURE1
 CC3220SF_LAUNCHXL.h, 86
 CC3220SF_LAUNCHXL_CAPTURECOUNT
 CC3220SF_LAUNCHXL.h, 86
 CC3220SF_LAUNCHXL_CaptureName
 CC3220SF_LAUNCHXL.h, 85
 CC3220SF_LAUNCHXL_CRYPTO0
 CC3220SF_LAUNCHXL.h, 86
 CC3220SF_LAUNCHXL_CRYPTOCOUNT
 CC3220SF_LAUNCHXL.h, 86
 CC3220SF_LAUNCHXL_CryptoName
 CC3220SF_LAUNCHXL.h, 86
 CC3220SF_LAUNCHXL_GPIO_LED_D10
 CC3220SF_LAUNCHXL.h, 86

CC3220SF_LAUNCHXL_GPIO_LED_OFF
 CC3220SF_LAUNCHXL.h, 85
CC3220SF_LAUNCHXL_GPIO_LED_ON
 CC3220SF_LAUNCHXL.h, 85
CC3220SF_LAUNCHXL_GPIO_SW2
 CC3220SF_LAUNCHXL.h, 86
CC3220SF_LAUNCHXL_GPIO_SW3
 CC3220SF_LAUNCHXL.h, 86
CC3220SF_LAUNCHXL_GPIO_TMP116_EN
 CC3220SF_LAUNCHXL.h, 86
CC3220SF_LAUNCHXL_GPIOCOUNT
 CC3220SF_LAUNCHXL.h, 86
CC3220SF_LAUNCHXL_GPIOName
 CC3220SF_LAUNCHXL.h, 86
CC3220SF_LAUNCHXL_I2C0
 CC3220SF_LAUNCHXL.h, 87
CC3220SF_LAUNCHXL_I2CCOUNT
 CC3220SF_LAUNCHXL.h, 87
CC3220SF_LAUNCHXL_I2CName
 CC3220SF_LAUNCHXL.h, 87
CC3220SF_LAUNCHXL_I2S0
 CC3220SF_LAUNCHXL.h, 87
CC3220SF_LAUNCHXL_I2SCOUNT
 CC3220SF_LAUNCHXL.h, 87
CC3220SF_LAUNCHXL_I2SName
 CC3220SF_LAUNCHXL.h, 87
CC3220SF_LAUNCHXL_initGeneral
 CC3220SF_LAUNCHXL.h, 90
CC3220SF_LAUNCHXL_LCD_CS
 CC3220SF_LAUNCHXL.h, 86
CC3220SF_LAUNCHXL_LCD_ENABLE
 CC3220SF_LAUNCHXL.h, 86
CC3220SF_LAUNCHXL_LCD_POWER
 CC3220SF_LAUNCHXL.h, 86
CC3220SF_LAUNCHXL_PWM6
 CC3220SF_LAUNCHXL.h, 87
CC3220SF_LAUNCHXL_PWM7
 CC3220SF_LAUNCHXL.h, 87
CC3220SF_LAUNCHXL_PWMCOUNT
 CC3220SF_LAUNCHXL.h, 87
CC3220SF_LAUNCHXL_PWMName
 CC3220SF_LAUNCHXL.h, 87
CC3220SF_LAUNCHXL_SD0
 CC3220SF_LAUNCHXL.h, 88
CC3220SF_LAUNCHXL_SDCount
 CC3220SF_LAUNCHXL.h, 88
CC3220SF_LAUNCHXL_SDFatFS0
 CC3220SF_LAUNCHXL.h, 88
CC3220SF_LAUNCHXL_SDFatFSCOUNT
 CC3220SF_LAUNCHXL.h, 88
CC3220SF_LAUNCHXL_SDFatFSName
 CC3220SF_LAUNCHXL.h, 88
CC3220SF_LAUNCHXL_SDName
 CC3220SF_LAUNCHXL.h, 88
CC3220SF_LAUNCHXL_SPI0
 CC3220SF_LAUNCHXL.h, 88
CC3220SF_LAUNCHXL_SPI1
 CC3220SF_LAUNCHXL.h, 88
CC3220SF_LAUNCHXL_SPI_MASTER_READY
 CC3220SF_LAUNCHXL.h, 86
CC3220SF_LAUNCHXL_SPI_SLAVE_READY
 CC3220SF_LAUNCHXL.h, 86
CC3220SF_LAUNCHXL_SPICOUNT
 CC3220SF_LAUNCHXL.h, 88
CC3220SF_LAUNCHXL_SPIName
 CC3220SF_LAUNCHXL.h, 88
CC3220SF_LAUNCHXL_TIMER0
 CC3220SF_LAUNCHXL.h, 89
CC3220SF_LAUNCHXL_TIMER1
 CC3220SF_LAUNCHXL.h, 89
CC3220SF_LAUNCHXL_TIMER2
 CC3220SF_LAUNCHXL.h, 89
CC3220SF_LAUNCHXL_TIMERCOUNT
 CC3220SF_LAUNCHXL.h, 89
CC3220SF_LAUNCHXL_TimerName
 CC3220SF_LAUNCHXL.h, 89
CC3220SF_LAUNCHXL_UART0
 CC3220SF_LAUNCHXL.h, 89
CC3220SF_LAUNCHXL_UART1
 CC3220SF_LAUNCHXL.h, 89
CC3220SF_LAUNCHXL_UARTCOUNT
 CC3220SF_LAUNCHXL.h, 89
CC3220SF_LAUNCHXL_UARTName
 CC3220SF_LAUNCHXL.h, 89
CC3220SF_LAUNCHXL_WATCHDOG0
 CC3220SF_LAUNCHXL.h, 89
CC3220SF_LAUNCHXL_WATCHDOGCOUNT
 CC3220SF_LAUNCHXL.h, 89
CC3220SF_LAUNCHXL_WatchdogName
 CC3220SF_LAUNCHXL.h, 89
CC3220SF_STARPORTS.c, 92
 ADC_config, 94
 ADC_count, 94
 adcCC3220SHWAttrs, 94
 adcCC3220SObjects, 95
 Capture_config, 95
 Capture_count, 95
 captureCC3220SFHWAttrs, 95
 captureCC3220SFObjects, 95
 CC3220SF_STARPORTS_initGeneral, 94
 cryptoCC3220SObjects, 96
 CryptoCC32XX_config, 96
 CryptoCC32XX_count, 96
 gpioCallbackFunctions, 96
 GPIOCC32XX_config, 96
 gpioPinConfigs, 96
 I2C_config, 97
 I2C_count, 97
 i2cCC3220SHWAttrs, 97
 i2cCC3220SObjects, 97
 parkInfo, 98
 PowerCC32XX_config, 98
 PWM_config, 98
 PWM_count, 98
 pwmTimerCC3220SHWAttrs, 98
 pwmTimerCC3220SObjects, 99

SPI_config, 99
 SPI_count, 99
 spiCC3220SDMAHWAAttrs, 99
 spiCC3220SDMAObjects, 99
 spiCC3220SDMAscratchBuf, 99
 TI_DRIVERS_UART_DMA, 93
 Timer_config, 100
 Timer_count, 100
 timerCC3220SFHWAAttrs, 100
 timerCC3220SFObjets, 100
 UART_config, 101
 UART_count, 101
 uartCC3220SHWAAttrs, 101
 uartCC3220SObjects, 101
 uartCC3220SRingBuffer, 102
 udmaCC3220SHWAAttrs, 102
 udmaCC3220SObject, 102
 UDMACC32XX_config, 102
 WakeupFromLPDS, 94
 Watchdog_config, 102
 Watchdog_count, 102
 watchdogCC3220SHWAAttrs, 103
 watchdogCC3220SObjects, 103
CC3220SF_STARPORTS.h, 111
 CC3220SF_STARPORTS_ADC0, 114
 CC3220SF_STARPORTS_ADCCOUNT, 114
 CC3220SF_STARPORTS_ADCName, 113
 CC3220SF_STARPORTS_ADXL355_CS, 114
 CC3220SF_STARPORTS_BME280_CS, 114
 CC3220SF_STARPORTS_CAPTURE0, 114
 CC3220SF_STARPORTS_CAPTURE1, 114
 CC3220SF_STARPORTS_CAPTURECOUNT, 114
 CC3220SF_STARPORTS_CaptureName, 114
 CC3220SF_STARPORTS_CRYPTO0, 114
 CC3220SF_STARPORTS_CRYPTOCOUNT, 114
 CC3220SF_STARPORTS_CryptoName, 114
 CC3220SF_STARPORTS_CS, 115
 CC3220SF_STARPORTS_DS1374_INTB, 115
 CC3220SF_STARPORTS_EN_ADXL355, 115
 CC3220SF_STARPORTS_EN_BME280, 115
 CC3220SF_STARPORTS_EN_LDC1000, 115
 CC3220SF_STARPORTS_EN_NODE, 115
 CC3220SF_STARPORTS_GPIO_LED_OFF, 112
 CC3220SF_STARPORTS_GPIO_LED_ON, 112
 CC3220SF_STARPORTS_GPIOCOUNT, 115
 CC3220SF_STARPORTS_GPIOName, 114
 CC3220SF_STARPORTS_I2C0, 115
 CC3220SF_STARPORTS_I2CCOUNT, 115
 CC3220SF_STARPORTS_I2CName, 115
 CC3220SF_STARPORTS_initGeneral, 117
 CC3220SF_STARPORTS_LDC1000_CS, 115
 CC3220SF_STARPORTS_MOSI, 115
 CC3220SF_STARPORTS_PWM5, 115
 CC3220SF_STARPORTS_PWMCOUNT, 115
 CC3220SF_STARPORTS_PWMName, 115
 CC3220SF_STARPORTS_RN2483_MCLR, 114
 CC3220SF_STARPORTS_SCL, 115
 CC3220SF_STARPORTS_SCLK, 115
 CC3220SF_STARPORTS_SD0, 116
 CC3220SF_STARPORTS_SDA, 115
 CC3220SF_STARPORTS_SDCOUNT, 116
 CC3220SF_STARPORTS_SDFatFS0, 116
 CC3220SF_STARPORTS_SDFatFSCOUNT, 116
 CC3220SF_STARPORTS_SDFatFSName, 116
 CC3220SF_STARPORTS_SDName, 116
 CC3220SF_STARPORTS_SPI0, 116
 CC3220SF_STARPORTS_SPI1, 116
 CC3220SF_STARPORTS_SPICOUNT, 116
 CC3220SF_STARPORTS_SPIName, 116
 CC3220SF_STARPORTS_TIMER0, 117
 CC3220SF_STARPORTS_TIMER1, 117
 CC3220SF_STARPORTS_TIMER2, 117
 CC3220SF_STARPORTS_TIMERCOUNT, 117
 CC3220SF_STARPORTS_TimerName, 116
 CC3220SF_STARPORTS_UART0, 117
 CC3220SF_STARPORTS_UART1, 117
 CC3220SF_STARPORTS_UARTCOUNT, 117
 CC3220SF_STARPORTS_UARTName, 117
 CC3220SF_STARPORTS_WATCHDOG0, 117
 CC3220SF_STARPORTS_WATCHDOGCOUNT, 117
 CC3220SF_STARPORTS_WatchdogName, 117
CC3220SF_STARPORTS_ADC0
 CC3220SF_STARPORTS.h, 114
CC3220SF_STARPORTS_ADCCOUNT
 CC3220SF_STARPORTS.h, 114
CC3220SF_STARPORTS_ADCName
 CC3220SF_STARPORTS.h, 113
CC3220SF_STARPORTS_ADXL355_CS
 CC3220SF_STARPORTS.h, 114
CC3220SF_STARPORTS_BME280_CS
 CC3220SF_STARPORTS.h, 114
CC3220SF_STARPORTS_CAPTURE0
 CC3220SF_STARPORTS.h, 114
CC3220SF_STARPORTS_CAPTURE1
 CC3220SF_STARPORTS.h, 114
CC3220SF_STARPORTS_CAPTURECOUNT
 CC3220SF_STARPORTS.h, 114
CC3220SF_STARPORTS_CaptureName
 CC3220SF_STARPORTS.h, 114
CC3220SF_STARPORTS_CRYPTO0
 CC3220SF_STARPORTS.h, 114
CC3220SF_STARPORTS_CRYPTOCOUNT
 CC3220SF_STARPORTS.h, 114
CC3220SF_STARPORTS_CryptoName
 CC3220SF_STARPORTS.h, 114
CC3220SF_STARPORTS_CS
 CC3220SF_STARPORTS.h, 115
CC3220SF_STARPORTS_DS1374_INTB
 CC3220SF_STARPORTS.h, 115
CC3220SF_STARPORTS_EN_ADXL355
 CC3220SF_STARPORTS.h, 115
CC3220SF_STARPORTS_EN_BME280
 CC3220SF_STARPORTS.h, 115
CC3220SF_STARPORTS_EN_LDC1000
 CC3220SF_STARPORTS.h, 115

CC3220SF_STARPORTS_EN_NODE
 CC3220SF_STARPORTS.h, 115
CC3220SF_STARPORTS_GPIO_LED_OFF
 CC3220SF_STARPORTS.h, 112
CC3220SF_STARPORTS_GPIO_LED_ON
 CC3220SF_STARPORTS.h, 112
CC3220SF_STARPORTS_GPIOCOUNT
 CC3220SF_STARPORTS.h, 115
CC3220SF_STARPORTS_GPIOName
 CC3220SF_STARPORTS.h, 114
CC3220SF_STARPORTS_I2C0
 CC3220SF_STARPORTS.h, 115
CC3220SF_STARPORTS_I2CCOUNT
 CC3220SF_STARPORTS.h, 115
CC3220SF_STARPORTS_I2CName
 CC3220SF_STARPORTS.h, 115
CC3220SF_STARPORTS_initGeneral
 CC3220SF_STARPORTS.c, 94
 CC3220SF_STARPORTS.h, 117
CC3220SF_STARPORTS_LDC1000_CS
 CC3220SF_STARPORTS.h, 115
CC3220SF_STARPORTS_MOSI
 CC3220SF_STARPORTS.h, 115
CC3220SF_STARPORTS_PWM5
 CC3220SF_STARPORTS.h, 115
CC3220SF_STARPORTS_PWMCOUNT
 CC3220SF_STARPORTS.h, 115
CC3220SF_STARPORTS_PWMName
 CC3220SF_STARPORTS.h, 115
CC3220SF_STARPORTS_RN2483_MCLR
 CC3220SF_STARPORTS.h, 114
CC3220SF_STARPORTS_SCL
 CC3220SF_STARPORTS.h, 115
CC3220SF_STARPORTS_SCLK
 CC3220SF_STARPORTS.h, 115
CC3220SF_STARPORTS_SD0
 CC3220SF_STARPORTS.h, 116
CC3220SF_STARPORTS_SDA
 CC3220SF_STARPORTS.h, 115
CC3220SF_STARPORTS_SDCount
 CC3220SF_STARPORTS.h, 116
CC3220SF_STARPORTS_SDFatFS0
 CC3220SF_STARPORTS.h, 116
CC3220SF_STARPORTS_SDFatFSCount
 CC3220SF_STARPORTS.h, 116
CC3220SF_STARPORTS_SDFatFSName
 CC3220SF_STARPORTS.h, 116
CC3220SF_STARPORTS_SDName
 CC3220SF_STARPORTS.h, 116
CC3220SF_STARPORTS_SPI0
 CC3220SF_STARPORTS.h, 116
CC3220SF_STARPORTS_SPI1
 CC3220SF_STARPORTS.h, 116
CC3220SF_STARPORTS_SPICOUNT
 CC3220SF_STARPORTS.h, 116
CC3220SF_STARPORTS_SPIName
 CC3220SF_STARPORTS.h, 116
CC3220SF_STARPORTS_TIMER0
 CC3220SF_STARPORTS.h, 117
CC3220SF_STARPORTS_TIMER1
 CC3220SF_STARPORTS.h, 117
CC3220SF_STARPORTS_TIMER2
 CC3220SF_STARPORTS.h, 117
CC3220SF_STARPORTS_TIMERCOUNT
 CC3220SF_STARPORTS.h, 117
CC3220SF_STARPORTS_TimerName
 CC3220SF_STARPORTS.h, 116
CC3220SF_STARPORTS_UART0
 CC3220SF_STARPORTS.h, 117
CC3220SF_STARPORTS_UART1
 CC3220SF_STARPORTS.h, 117
CC3220SF_STARPORTS_UARTCOUNT
 CC3220SF_STARPORTS.h, 117
CC3220SF_STARPORTS_UARTName
 CC3220SF_STARPORTS.h, 117
CC3220SF_STARPORTS_WATCHDOG0
 CC3220SF_STARPORTS.h, 117
CC3220SF_STARPORTS_WATCHDOGCOUNT
 CC3220SF_STARPORTS.h, 117
CC3220SF_STARPORTS_WatchdogName
 CC3220SF_STARPORTS.h, 117
CERTIFICATE
 file_system.h, 160
CLK_CONFIG
 LDC1000.h, 342
CLK_OFF
 LDC1000.h, 341
CLK_ON
 LDC1000.h, 341
CLR_STATUS_BIT
 wifi.h, 411
COMP_OUT
 LDC1000.h, 341
COMPB_MASK
 LDC1000.h, 346
compensate_humidity
 BME280.c, 50
 BME280.h, 71
compensate_pressure
 BME280.c, 51
 BME280.h, 72
compensate_temperature
 BME280.c, 52
 BME280.h, 73
CONFIG
 BME280.h, 63
CONFIG_LORA
 STARPORTS_App.h, 395
Config_PWM
 hal_PWM.c, 278
 hal_PWM.h, 280
CONFIG_WIFI
 STARPORTS_App.h, 395
CONTINUE
 STARPORTS_App.h, 392
cryptoCC3220SObjects

CC3220SF_STARPORTS.c, 96
 CryptoCC32XX_config
 CC3220SF_STARPORTS.c, 96
 CryptoCC32XX_count
 CC3220SF_STARPORTS.c, 96
 CTRL_HUM
 BME280.h, 63
 CTRL_MEAS
 BME280.h, 63
 DATA_RDY
 ADXL355.h, 28
 deleteFile
 file_system.c, 134
 file_system.h, 162
 DEST_IP_ADDR
 wifi.h, 411
 DEV_ID
 LDC1000.h, 342
 DEVADDR
 STARPORTS_App.h, 392
 DEVEUI
 STARPORTS_App.h, 392
 DEVID
 BME280.h, 63
 DEVID_AD
 ADXL355.h, 27
 DEVID_MST
 ADXL355.h, 27
 DRDY_EN
 LDC1000.h, 341
 DRDY_OFF
 ADXL355.h, 26
 DRDYB_MASK
 LDC1000.h, 346
 DS1374.c, 119
 DS1374_Clear_AF, 120
 DS1374_Read_Ctrl, 121
 DS1374_Read_Status, 121
 DS1374_Read_Tod, 122
 DS1374_Read_WdAlmb, 122
 DS1374_Write_Ctrl, 123
 DS1374_Write_Tod, 123
 DS1374_Write_WdAlmb, 124
 DS1374.h, 126
 DS1374_Clear_AF, 128
 DS1374_CTRL, 127
 DS1374_Read_Ctrl, 128
 DS1374_Read_Status, 129
 DS1374_Read_Tod, 129
 DS1374_Read_WdAlmb, 130
 DS1374_register_t, 127
 DS1374_STATUS, 127
 DS1374_TC, 127
 DS1374_Write_Ctrl, 130
 DS1374_Write_Tod, 131
 DS1374_Write_WdAlmb, 131
 TOD0, 127
 TOD1, 127
 TOD2, 127
 TOD3, 127
 WDALMB0, 127
 WDALMB1, 127
 WDALMB2, 127
 DS1374_Clear_AF
 DS1374.c, 120
 DS1374.h, 128
 DS1374_CTRL
 DS1374.h, 127
 DS1374_Read_Ctrl
 DS1374.c, 121
 DS1374.h, 128
 DS1374_Read_Status
 DS1374.c, 121
 DS1374.h, 129
 DS1374_Read_Tod
 DS1374.c, 122
 DS1374.h, 129
 DS1374_Read_WdAlmb
 DS1374.c, 122
 DS1374.h, 130
 DS1374_register_t
 DS1374.h, 127
 DS1374_STATUS
 DS1374.h, 127
 DS1374_TC
 DS1374.h, 127
 DS1374_Write_Ctrl
 DS1374.c, 123
 DS1374.h, 130
 DS1374_Write_Tod
 DS1374.c, 123
 DS1374.h, 131
 DS1374_Write_WdAlmb
 DS1374.c, 124
 DS1374.h, 131
 e_StatusBits
 wifi.h, 413
 ERR_PRINT
 hal_UART.h, 317
 ERROR_ABP_APPSKY
 hal_LORA.h, 248
 ERROR_ABP_DENIED
 hal_LORA.h, 248
 ERROR_ABP_DEVADDR
 hal_LORA.h, 248
 ERROR_ABP_JOIN
 hal_LORA.h, 248
 ERROR_ABP_NWKSKEY
 hal_LORA.h, 248
 ERROR_ABP_SAVE
 hal_LORA.h, 249
 ERROR_ADR_SET
 hal_LORA.h, 249
 ERROR_CONNECT_WIFI
 wifi.h, 411
 ERROR_OTAA_APPEUI

hal_LORA.h, 249
ERROR_OTAA_APPKEY
 hal_LORA.h, 249
ERROR_OTAA_BUSY
 hal_LORA.h, 249
ERROR_OTAA_DENIED
 hal_LORA.h, 249
ERROR_OTAA_DEVEUI
 hal_LORA.h, 249
ERROR_OTAA_INVALID_PARAM
 hal_LORA.h, 249
ERROR_OTAA_JOIN
 hal_LORA.h, 250
ERROR_OTAA_KEYS_NOT_INIT
 hal_LORA.h, 250
ERROR_OTAA_MAC_PAUSED
 hal_LORA.h, 250
ERROR_OTAA_NO_FREE_CH
 hal_LORA.h, 250
ERROR_OTAA_SAVE
 hal_LORA.h, 250
ERROR_OTAA_SILENT
 hal_LORA.h, 250
ERROR_OTAA_UNKNOWN
 hal_LORA.h, 250
ERROR_RXDELAY1_SET
 hal_LORA.h, 250
ERROR_SET_DEVADDR
 hal_LORA.h, 251
ERROR_SET_DNCTR
 hal_LORA.h, 251
ERROR_SET_NWSKEY
 hal_LORA.h, 251
ERROR_SET_PWRIDX
 hal_LORA.h, 251
ERROR_SET_UPCTR
 hal_LORA.h, 251
ERROR_TX_BUSY
 hal_LORA.h, 251
ERROR_TX_INVALID_DATA_LEN
 hal_LORA.h, 251
ERROR_TX_INVALID_PARAM
 hal_LORA.h, 251
ERROR_TX_MAC_ERR
 hal_LORA.h, 252
ERROR_TX_MAC_PAUSED
 hal_LORA.h, 252
ERROR_TX_NO_FREE_CH
 hal_LORA.h, 252
ERROR_TX_NOT_JOINED
 hal_LORA.h, 252
ERROR_TX_REJOIN_NEEDED
 hal_LORA.h, 252
ERROR_TX_SILENT
 hal_LORA.h, 252
ERROR_TX_UNKNOWN
 hal_LORA.h, 252
ERROR_WRONG_CR
 hal_TMP006.h, 306
EXT_CLK
 ADXL355.h, 29
EXT_SYNC_INT
 ADXL355.h, 29
EXT_SYNC_NO_INT
 ADXL355.h, 29
FCOUNT_LSB
 LDC1000.h, 342
FCOUNT_MSB
 LDC1000.h, 342
FCOUNT_XSB
 LDC1000.h, 342
FIFO_DATA
 ADXL355.h, 27
FIFO_ENTRIES
 ADXL355.h, 27
FIFO_FULL
 ADXL355.h, 28
FIFO_OVR
 ADXL355.h, 28
FIFO_SAMPLES
 ADXL355.h, 27
file_system.c, 132
 deleteFile, 134
 st_readFileChangeWakeUp, 134
 st_readFileDnCntr, 135
 st_read FileMode, 136
 st_readFileNBoot, 137
 st_readFileNCycles, 137
 st_readFileNFails, 138
 st_readFileNodeId, 139
 st_readFileSSID, 140
 st_readFileUpCntr, 140
 st_readFileWakeUp, 141
 uart0, 150
 writeChangeWakeUp, 142
 writeDnCntr, 143
 writeMode, 143
 writeNBoot, 144
 writeNCycles, 145
 writeNFails, 146
 writeNodeId, 147
 writeSSID, 147
 writeUpCntr, 148
 writeWakeUp, 149
file_system.h, 158
 CERTIFICATE, 160
 deleteFile, 162
 FS_CHANGEWAKEUP, 160
 FS_DNCNTR, 160
 FS_MODE, 160
 FS_NBOOT, 160
 FS_NCYCLES, 161
 FS_NFAILS, 161
 FS_NODEID, 161
 FS_PAYLOAD, 161
 FS_SSID, 161

FS_UPCNTR, 161
FS_WAKEUP, 161
MAX_FILE_ENTRIES, 161
MAX_FILE_SIZE, 162
st_readFileChangeWakeUp, 162
st_readFileDnCntr, 163
st_read FileMode, 164
st_readFileNBoot, 164
st_readFileNCycles, 165
st_readFileNFails, 166
st_readFileNodeID, 167
st_readFileSSID, 167
st_readFileUpCntr, 168
st_readFileWakeUp, 169
writeChangeWakeUp, 169
writeDnCntr, 170
writeMode, 171
writeNBoot, 172
writeNCycles, 172
writeNFails, 173
writeNodeID, 174
writeSSID, 175
writeUpCntr, 175
writeWakeUp, 176
FILTER
 ADXL355.h, 27
FILTER_16
 BME280.h, 61
FILTER_2
 BME280.h, 61
FILTER_4
 BME280.h, 61
FILTER_8
 BME280.h, 61
FILTER_OFF
 BME280.h, 61
FORCED
 BME280.h, 61
FRAME_LENGTH
 wifi.h, 411
FS_CHANGEWAKEUP
 file_system.h, 160
FS_DNCNTR
 file_system.h, 160
FS_MODE
 file_system.h, 160
FS_NBOOT
 file_system.h, 160
FS_NCYCLES
 file_system.h, 161
FS_NFAILS
 file_system.h, 161
FS_NODEID
 file_system.h, 161
FS_PAYLOAD
 file_system.h, 161
FS_SSID
 file_system.h, 161
FS_UPCNTR
 file_system.h, 161
FS_WAKEUP
 file_system.h, 161
FULL_EN
 ADXL355.h, 26
GET_NEW_PARAMS
 STARPORTS_App.h, 395
GET_PARAMS
 STARPORTS_App.h, 395
GET_SENSOR_DATA
 STARPORTS_App.h, 395
GET_STATUS_BIT
 wifi.h, 411
GetID_TMP006
 hal_TMP006.c, 302
 hal_TMP006.h, 306
GetLine_UART
 hal_UART.c, 310
 hal_UART.h, 317
GetLoraRxData
 hal_LORA.c, 212
 hal_LORA.h, 254
GetLoraServerParams
 hal_LORA.c, 213
 hal_LORA.h, 254
GetSensorData
 Sensors.c, 364
 Sensors.h, 374
GetTA_TMP006
 hal_TMP006.c, 302
 hal_TMP006.h, 307
GPIO_Config
 hal_GPIO.c, 185
 hal_GPIO.h, 197
gpioCallbackFunctions
 CC3220SF_STARPORTS.c, 96
GPIOCC32XX_config
 CC3220SF_STARPORTS.c, 96
gpioPinConfigs
 CC3220SF_STARPORTS.c, 96
hal_ADC.c, 178
 Startup_ADC, 178
hal_ADC.h, 179
 Startup_ADC, 180
hal_GPIO.c, 181
 ADXL355_Disable, 182
 ADXL355_Enable, 183
 ADXL355_SPI_Disable, 183
 ADXL355_SPI_Enable, 183
 BME280_Disable, 184
 BME280_Enable, 184
 BME280_SPI_Disable, 185
 BME280_SPI_Enable, 185
 GPIO_Config, 185
 I2C_As_GPIO_Low, 186
 LDC1000_Disable, 186

LDC1000_Enable, 187
LDC1000_SPI_Disable, 187
LDC1000_SPI_Enable, 188
Node_Disable, 188
Node_Enable, 188
RN2483_Clear, 189
RN2483_Set, 189
SPI_As_GPIO_Low, 190
SPI_CS_Disable, 190
hal_GPIO.h, 193
ADXL355_Disable, 194
ADXL355_Enable, 195
ADXL355_SPI_Disable, 195
ADXL355_SPI_Enable, 195
BME280_Disable, 196
BME280_Enable, 196
BME280_SPI_Disable, 197
BME280_SPI_Enable, 197
GPIO_Config, 197
I2C_As_GPIO_Low, 198
LDC1000_Disable, 198
LDC1000_Enable, 199
LDC1000_SPI_Disable, 199
LDC1000_SPI_Enable, 200
Node_Disable, 200
Node_Enable, 200
RN2483_Clear, 201
RN2483_Set, 201
SPI_As_GPIO_Low, 202
SPI_CS_Disable, 202
hal_I2C.c, 203
I2C_read_8bits, 204
I2C_write_8bits, 204
Startup_I2C, 205
hal_I2C.h, 207
I2C_read_8bits, 208
I2C_write_8bits, 208
Startup_I2C, 209
hal_LORA.c, 210
GetLoraRxData, 212
GetLoraServerParams, 213
hex2int, 214
Join_Abp_Lora, 215
Join_Otaa_Lora, 216
Mac_AdR_On, 217
Mac_Ar_On, 217
Mac_Clear_Upctr, 218
Mac_Get_Appeui, 218
Mac_Get_Devaddr, 219
Mac_Get_Deveui, 220
Mac_Get_Dnctr, 220
Mac_Get_Upctr, 221
Mac_Save, 221
Mac_Set_Devaddr, 222
Mac_Set_Dnctr, 223
Mac_Set_Pwridx, 223
Mac_Set_Rxdelay1, 224
Mac_Set_Upctr, 225
MyADXL, 234
MyBME, 234
MyLDC, 234
MyNode, 234
Reset_RN2483, 225
Setup_Abp_Lora, 226
Setup_Otaa_Lora, 227
Sys_FactoryReset, 228
Sys_Sleep, 229
Try_Join_Lora_Gateway, 229
Tx_Cnf_Lora, 230
Tx_Uncnf_Lora, 232
uart0, 234
Uint8Array2Char, 233
hal_LORA.h, 245
ERROR_ABP_APPSKEY, 248
ERROR_ABP_DENIED, 248
ERROR_ABP_DEVADDR, 248
ERROR_ABP_JOIN, 248
ERROR_ABP_NWKSKEY, 248
ERROR_ABP_SAVE, 249
ERROR_ADR_SET, 249
ERROR_OTAA_APPEUI, 249
ERROR_OTAA_APPKEY, 249
ERROR_OTAA_BUSY, 249
ERROR_OTAA_DENIED, 249
ERROR_OTAA_DEVEUI, 249
ERROR_OTAA_INVALID_PARAM, 249
ERROR_OTAA_JOIN, 250
ERROR_OTAA_KEYS_NOT_INIT, 250
ERROR_OTAA_MAC_PAUSED, 250
ERROR_OTAA_NO_FREE_CH, 250
ERROR_OTAA_SAVE, 250
ERROR_OTAA_SILENT, 250
ERROR_OTAA_UNKNOWN, 250
ERROR_RXDELAY1_SET, 250
ERROR_SET_DEVADDR, 251
ERROR_SET_DNCTR, 251
ERROR_SET_NWSKEY, 251
ERROR_SET_PWRIDX, 251
ERROR_SET_UPCTR, 251
ERROR_TX_BUSY, 251
ERROR_TX_INVALID_DATA_LEN, 251
ERROR_TX_INVALID_PARAM, 251
ERROR_TX_MAC_ERR, 252
ERROR_TX_MAC_PAUSED, 252
ERROR_TX_NO_FREE_CH, 252
ERROR_TX_NOT_JOINED, 252
ERROR_TX_REJOIN_NEEDED, 252
ERROR_TX_SILENT, 252
ERROR_TX_UNKNOWN, 252
GetLoraRxData, 254
GetLoraServerParams, 254
hex2int, 256
Join_Abp_Lora, 256
Join_Otaa_Lora, 257
Mac_AdR_On, 258
Mac_Ar_On, 259

Mac_Clear_Upctr, 259
 Mac_Get_Appeui, 260
 Mac_Get_Devaddr, 261
 Mac_Get_Deveui, 261
 Mac_Get_Dnctr, 262
 Mac_Get_Upctr, 262
 Mac_Save, 263
 Mac_Set_Devaddr, 264
 Mac_Set_Dnctr, 264
 Mac_Set_Pwridx, 266
 Mac_Set_Rxdelay1, 267
 Mac_Set_Upctr, 267
 Reset_RN2483, 268
 Setup_Abp_Lora, 268
 Setup_Otaa_Lora, 269
 SUCCESS_ABP_LORA, 252
 SUCCESS_OTAA_LORA, 253
 SUCCESS_RXDELAY1_SET, 253
 SUCCESS_SET_DEVADDR, 253
 SUCCESS_SET_DNCTR, 253
 SUCCESS_SET_NWSKEY, 253
 SUCCESS_SET_PWRIDX, 253
 SUCCESS_SET_UPCTR, 253
 SUCCESS_TX_MAC_RX, 253
 SUCCESS_TX_MAC_TX, 254
 Sys_FactoryReset, 270
 Sys_Sleep, 271
 Try_Join_Lora_Gateway, 272
 Tx_Cnf_Lora, 273
 Tx_Uncnf_Lora, 274
 Uint8Array2Char, 275
 hal_PWM.c, 278
 Config_PWM, 278
 hal_PWM.h, 279
 Config_PWM, 280
 hal_SPI.c, 281
 SPI_read_16bits, 282
 SPI_read_8bits, 283
 SPI_write_16bits, 283
 SPI_write_8bits, 284
 Startup_SPI, 285
 hal_SPI.h, 287
 SPI_read_16bits, 288
 SPI_read_8bits, 289
 SPI_write_16bits, 290
 SPI_write_8bits, 290
 Startup_SPI, 291
 hal_Timer.c, 292
 Startup_Continuous_Timer, 293
 Startup_Oneshot_Timer, 294
 timer0Callback, 294
 Timer0Event, 296
 timer1Callback, 296
 Timer1Event, 296
 hal_Timer.h, 297
 Startup_Continuous_Timer, 298
 Startup_Oneshot_Timer, 299
 timer0Callback, 300
 timer1Callback, 300
 hal_TMP006.c, 301
 GetID_TMP006, 302
 GetTA_TMP006, 302
 SetMOD_TMP006, 303
 hal_TMP006.h, 305
 ERROR_WRONG_CR, 306
 GetID_TMP006, 306
 GetTA_TMP006, 307
 SetMOD_TMP006, 308
 SUCCESS_CONF_TMP006, 306
 hal_UART.c, 309
 GetLine_UART, 310
 Message, 311
 putch, 311
 Report, 312
 Startup_UART, 313
 uart0, 314
 UART_resetInputBuffer, 313
 vsnprintf, 314
 hal_UART.h, 316
 ERR_PRINT, 317
 GetLine_UART, 317
 Message, 318
 putch, 318
 Report, 319
 Startup_UART, 320
 UART_PRINT, 317
 UART_resetInputBuffer, 321
 hal_WD.c, 322
 i2c, 324
 Startup_Watchdog, 322
 watchdogCallback, 323
 hal_WD.h, 325
 Startup_Watchdog, 325
 hex2int
 hal_LORA.c, 214
 hal_LORA.h, 256
 HPF02
 ADXL355.h, 24
 HPF09
 ADXL355.h, 24
 HPF15
 ADXL355.h, 24
 HPF247
 ADXL355.h, 24
 HPF3
 ADXL355.h, 24
 HPF62
 ADXL355.h, 24
 HPFOFF
 ADXL355.h, 24
 HUM_LSB
 BME280.h, 63
 HUM_MSB
 BME280.h, 63
 i2c
 hal_WD.c, 324

Sensors.c, 367
STARPORTS_App.c, 382
I2C_As_GPIO_Low
 hal_GPIO.c, 186
 hal_GPIO.h, 198
I2C_config
 CC3220SF_STARPORTS.c, 97
I2C_count
 CC3220SF_STARPORTS.c, 97
I2C_FAST_MODE
 ADXL355.h, 25
I2C_HIGH_SPEED
 ADXL355.h, 25
I2C_read_8bits
 hal_I2C.c, 204
 hal_I2C.h, 208
I2C_write_8bits
 hal_I2C.c, 204
 hal_I2C.h, 208
i2cCC3220SHWAttrs
 CC3220SF_STARPORTS.c, 97
i2cCC3220SObjects
 CC3220SF_STARPORTS.c, 97
IniPacket
 Sensors.c, 367
 Sensors.h, 376
INT_MAP
 ADXL355.h, 28
INT_POL_HIGH
 ADXL355.h, 25
INT_POL_LOW
 ADXL355.h, 25
INT_SYNC
 ADXL355.h, 29
INTB_CONFIG
 LDC1000.h, 342
INTB_DIS
 LDC1000.h, 341
IS_CONNECTED
 wifi.h, 411
IS_IP_ACQUIRED
 wifi.h, 412
Join_Abp_Lora
 hal_LORA.c, 215
 hal_LORA.h, 256
Join_Otaa_Lora
 hal_LORA.c, 216
 hal_LORA.h, 257
Keller_Data, 390
LDC1000.c, 326
 LDC1000_DevId, 328
 LDC1000_Get_Proximity_Frame, 328
 LDC1000_Read_Fcount, 329
 LDC1000_Read_Proximity, 329
 LDC1000_Read_Rp_Max, 330
 LDC1000_Read_Rp_Min, 330
 LDC1000_Read_Status, 331
 LDC1000_Write_Clk_Conf, 331
 LDC1000_Write_Conf, 332
 LDC1000_Write_Intb_Conf, 332
 LDC1000_Write_Min_Freq, 333
 LDC1000_Write_Pow_Conf, 333
 LDC1000_Write_Rp_Max, 334
 LDC1000_Write_Rp_Min, 334
 RpCalc, 335
 LDC1000.h, 338
 ACTIVE_MODE, 342
 AMP_1V, 340
 AMP_2V, 340
 AMP_4V, 340
 CLK_CONFIG, 342
 CLK_OFF, 341
 CLK_ON, 341
 COMP_OUT, 341
 COMPB_MASK, 346
 DEV_ID, 342
 DRDY_EN, 341
 DRDYB_MASK, 346
 FCOUNT_LSB, 342
 FCOUNT_MSB, 342
 FCOUNT_XSB, 342
 INTB_CONFIG, 342
 INTB_DIS, 341
 LDC1000_amplitude_t, 340
 LDC1000_clkpdt_t, 341
 LDC1000_clkSEL_t, 341
 LDC1000_DevId, 346
 LDC1000_Get_Proximity_Frame, 347
 LDC1000_intb_conf_t, 341
 LDC1000_pow_conf_t, 342
 LDC1000_Read_Fcount, 347
 LDC1000_Read_Proximity, 348
 LDC1000_Read_Rp_Max, 348
 LDC1000_Read_Rp_Min, 349
 LDC1000_Read_Status, 349
 LDC1000_register_t, 342
 LDC1000_resp_time_t, 343
 LDC1000_rp_max_t, 343
 LDC1000_rp_min_t, 344
 LDC1000_status_t, 346
 LDC1000_Write_Clk_Conf, 350
 LDC1000_Write_Conf, 350
 LDC1000_Write_Intb_Conf, 351
 LDC1000_Write_Min_Freq, 351
 LDC1000_Write_Pow_Conf, 352
 LDC1000_Write_Rp_Max, 352
 LDC1000_Write_Rp_Min, 353
 LDC_CONFIG, 342
 LDC_STATUS, 342
 MIN_FREQ, 342
 OSC_STATUS_MASK, 346
 POW_CONFIG, 342
 PROX_LSB, 342
 PROX_MSB, 342

RESP_TIME_0192, 343
RESP_TIME_0384, 343
RESP_TIME_0768, 343
RESP_TIME_1536, 343
RESP_TIME_3072, 343
RESP_TIME_6144, 343
RP_MAX, 342
RP_MIN, 342
RpCalc, 353
RPMAX0000, 344
RPMAX0001p0, 344
RPMAX0001p3, 344
RPMAX0001p7, 344
RPMAX0002, 344
RPMAX0003, 344
RPMAX0004, 344
RPMAX0005, 344
RPMAX0007, 344
RPMAX0009, 344
RPMAX0012, 344
RPMAX0016, 344
RPMAX0021, 344
RPMAX0027, 344
RPMAX0038, 344
RPMAX0048, 344
RPMAX0064, 344
RPMAX0083, 344
RPMAX0109, 344
RPMAX0145, 344
RPMAX0193, 344
RPMAX0249, 344
RPMAX0349, 344
RPMAX0436, 344
RPMAX0581, 345
RPMIN0083, 345
RPMIN0109, 345
RPMIN0145, 345
RPMIN0193, 345
RPMIN0249, 345
RPMIN0349, 345
RPMIN0436, 345
RPMIN0581, 345
RPMIN0747, 345
RPMIN0981, 345
RPMIN1308, 345
RPMIN1745, 345
RPMIN2243, 345
RPMIN3141, 345
RPMIN3926, 345
STBY_MODE, 342
THR_HIGH_LSB, 342
THR_HIGH_MSB, 342
THR_LOW_LSB, 342
THR_LOW_MSB, 342
WAKEUP_DIS_MASK, 346
WAKEUP_EN, 341
XIN, 341
XIN_XOUT, 341
LDC1000_amplitude_t
 LDC1000.h, 340
LDC1000_clkpd_t
 LDC1000.h, 341
LDC1000_clksel_t
 LDC1000.h, 341
LDC1000_Disable
 hal_GPIO.c, 186
 hal_GPIO.h, 198
LDC1000_Enable
 hal_GPIO.c, 187
 hal_GPIO.h, 199
LDC1000_Get_Proximity_Frame
 LDC1000.c, 328
 LDC1000.h, 347
LDC1000_ID
 STARPORTS_App.h, 392
LDC1000_intb_conf_t
 LDC1000.h, 341
LDC1000_pow_conf_t
 LDC1000.h, 342
LDC1000_Read_Fcount
 LDC1000.c, 329
 LDC1000.h, 347
LDC1000_Read_Proximity
 LDC1000.c, 329
 LDC1000.h, 348
LDC1000_Read_Rp_Max
 LDC1000.c, 330
 LDC1000.h, 348

LDC1000_Read_Rp_Min
 LDC1000.c, 330
 LDC1000.h, 349

LDC1000_Read_Status
 LDC1000.c, 331
 LDC1000.h, 349

LDC1000_register_t
 LDC1000.h, 342

LDC1000_resp_time_t
 LDC1000.h, 343

LDC1000_rp_max_t
 LDC1000.h, 343

LDC1000_rp_min_t
 LDC1000.h, 344

LDC1000_SPI_Disable
 hal_GPIO.c, 187
 hal_GPIO.h, 199

LDC1000_SPI_Enable
 hal_GPIO.c, 188
 hal_GPIO.h, 200

LDC1000_status_t
 LDC1000.h, 346

LDC1000_Write_Clk_Conf
 LDC1000.c, 331
 LDC1000.h, 350

LDC1000_Write_Conf
 LDC1000.c, 332
 LDC1000.h, 350

LDC1000_Write_Intb_Conf
 LDC1000.c, 332
 LDC1000.h, 351

LDC1000_Write_Min_Freq
 LDC1000.c, 333
 LDC1000.h, 351

LDC1000_Write_Pow_Conf
 LDC1000.c, 333
 LDC1000.h, 352

LDC1000_Write_Rp_Max
 LDC1000.c, 334
 LDC1000.h, 352

LDC1000_Write_Rp_Min
 LDC1000.c, 334
 LDC1000.h, 353

LDC_CONFIG
 LDC1000.h, 342

LDC_STATUS
 LDC1000.h, 342

LOOP_FOREVER
 wifi.h, 412

LoraNode, 247

LPDSOut
 STARPORTS_App.c, 382

Mac_Adr_On
 hal_LORA.c, 217
 hal_LORA.h, 258

Mac_Ar_On
 hal_LORA.c, 217
 hal_LORA.h, 259

Mac_Clear_Upctr
 hal_LORA.c, 218
 hal_LORA.h, 259

Mac_Get_Appeui
 hal_LORA.c, 218
 hal_LORA.h, 260

Mac_Get_Devaddr
 hal_LORA.c, 219
 hal_LORA.h, 261

Mac_Get_Deveui
 hal_LORA.c, 220
 hal_LORA.h, 261

Mac_Get_Dnctr
 hal_LORA.c, 220
 hal_LORA.h, 262

Mac_Get_Upctr
 hal_LORA.c, 221
 hal_LORA.h, 262

Mac_Save
 hal_LORA.c, 221
 hal_LORA.h, 263

Mac_Set_Devaddr
 hal_LORA.c, 222
 hal_LORA.h, 264

Mac_Set_Dnctr
 hal_LORA.c, 223
 hal_LORA.h, 264

Mac_Set_Pwridx
 hal_LORA.c, 223
 hal_LORA.h, 266

Mac_Set_Rxdelay1
 hal_LORA.c, 224
 hal_LORA.h, 267

Mac_Set_Upctr
 hal_LORA.c, 225
 hal_LORA.h, 267

main
 main_nortos.c, 357

main_nortos.c, 356
 main, 357
 mainThread, 357

mainThread
 main_nortos.c, 357
 STARPORTS_App.c, 379

MAX_FILE_ENTRIES
 file_system.h, 161

MAX_FILE_SIZE
 file_system.h, 162

MEASUREMENT
 ADXL355.h, 26

MEASURING
 BME280.h, 64

Message
 hal_UART.c, 311
 hal_UART.h, 318

MIN_FREQ
 LDC1000.h, 342

MODE_AP_WIFI

STARPORTS_App.h, 392
 MODE_NORMAL_LORA
 STARPORTS_App.h, 392
 MODE_NORMAL_WIFI
 STARPORTS_App.h, 392
 MODE_STORAGE_LORA
 STARPORTS_App.h, 392
 MODE_STORAGE_WIFI
 STARPORTS_App.h, 393
 MyADXL
 hal_LORA.c, 234
 Sensors.c, 367
 STARPORTS_App.c, 382
 MyBME
 hal_LORA.c, 234
 Sensors.c, 367
 STARPORTS_App.c, 382
 MyLDC
 hal_LORA.c, 234
 Sensors.c, 367
 STARPORTS_App.c, 383
 MyNode
 hal_LORA.c, 234
 Sensors.c, 368
 STARPORTS_App.c, 383
 MyTMP006
 Sensors.c, 368
 STARPORTS_App.c, 383
 MyVbat
 Sensors.c, 368
 STARPORTS_App.c, 383
 Node, 390
 Node_Disable
 hal_GPIO.c, 188
 hal_GPIO.h, 200
 Node_Enable
 hal_GPIO.c, 188
 hal_GPIO.h, 200
 Node_states_t
 STARPORTS_App.h, 394
 NODEID
 STARPORTS_App.h, 393
 NodeState
 STARPORTS_App.h, 395
 NORMAL
 BME280.h, 61
 NUM_OF_PKT
 wifi.h, 412
 NUMBER_SENSORS
 STARPORTS_App.h, 393
 NVM_BUSY
 ADXL355.h, 28
 NWKSKEY
 STARPORTS_App.h, 393
 ODR1000HZ
 ADXL355.h, 24
 ODR125Hz
 ADXL355.h, 24
 ODR15Hz
 ADXL355.h, 24
 ODR2000HZ
 ADXL355.h, 24
 ODR250HZ
 ADXL355.h, 24
 ODR31Hz
 ADXL355.h, 24
 ODR3HZ
 ADXL355.h, 24
 ODR4000HZ
 ADXL355.h, 24
 ODR500HZ
 ADXL355.h, 24
 ODR62HZ
 ADXL355.h, 24
 ODR7Hz
 ADXL355.h, 24
 OFFSET_X_H
 ADXL355.h, 27
 OFFSET_X_L
 ADXL355.h, 27
 OFFSET_Y_H
 ADXL355.h, 27
 OFFSET_Y_L
 ADXL355.h, 27
 OFFSET_Z_H
 ADXL355.h, 27
 OFFSET_Z_L
 ADXL355.h, 27
 OSC_STATUS_MASK
 LDC1000.h, 346
 OSRS_HX1
 BME280.h, 61
 OSRS_HX16
 BME280.h, 61
 OSRS_HX2
 BME280.h, 61
 OSRS_HX4
 BME280.h, 61
 OSRS_HX8
 BME280.h, 61
 OSRS_PX1
 BME280.h, 62
 OSRS_PX16
 BME280.h, 62
 OSRS_PX2
 BME280.h, 62
 OSRS_PX4
 BME280.h, 62
 OSRS_PX8
 BME280.h, 62
 OSRS_TX1
 BME280.h, 62
 OSRS_TX16
 BME280.h, 62
 OSRS_TX2

BME280.h, 62
OSRS_TX4
 BME280.h, 62
OSRS_TX8
 BME280.h, 62
OVR_EN
 ADXL355.h, 26

parkInfo
 CC3220SF_STARPORTS.c, 98
PARTID
 ADXL355.h, 27
PORT
 wifi.h, 412
PORT_UDP
 wifi.h, 412
POW_CONFIG
 LDC1000.h, 342
POWER_CTL
 ADXL355.h, 28
PowerCC32XX_config
 CC3220SF_STARPORTS.c, 98
PowerMeasure_CB
 wifi.h, 421
POWERUP
 STARPORTS_App.h, 395
prepareDataFrame
 wifi.c, 398
 wifi.h, 414
PRESS_LSB
 BME280.h, 63
PRESS_MSB
 BME280.h, 63
PRESS_XLSB
 BME280.h, 63
PROX_LSB
 LDC1000.h, 342
PROX_MSB
 LDC1000.h, 342
putch
 hal_UART.c, 311
 hal_UART.h, 318
PWM_config
 CC3220SF_STARPORTS.c, 98
PWM_count
 CC3220SF_STARPORTS.c, 98
pwmTimerCC3220SHWAttrs
 CC3220SF_STARPORTS.c, 98
pwmTimerCC3220SObjects
 CC3220SF_STARPORTS.c, 99

RANGE
 ADXL355.h, 28
RANGE2G
 ADXL355.h, 27
RANGE4G
 ADXL355.h, 27
RANGE8G
 ADXL355.h, 27

RDY_EN
 ADXL355.h, 26
Report
 hal_UART.c, 312
 hal_UART.h, 319
Reset_RN2483
 hal_LORA.c, 225
 hal_LORA.h, 268
RESP_TIME_0192
 LDC1000.h, 343
RESP_TIME_0384
 LDC1000.h, 343
RESP_TIME_0768
 LDC1000.h, 343
RESP_TIME_1536
 LDC1000.h, 343
RESP_TIME_3072
 LDC1000.h, 343
RESP_TIME_6144
 LDC1000.h, 343
REVID
 ADXL355.h, 27
RN2483_Clear
 hal_GPIO.c, 189
 hal_GPIO.h, 201
RN2483_Set
 hal_GPIO.c, 189
 hal_GPIO.h, 201
RNW_LSB
 STARPORTS_App.h, 393
RNW_MSB
 STARPORTS_App.h, 393
RP_MAX
 LDC1000.h, 342
RP_MIN
 LDC1000.h, 342
RpCalc
 LDC1000.c, 335
 LDC1000.h, 353
RPMAX0000
 LDC1000.h, 344
RPMAX0001p0
 LDC1000.h, 344
RPMAX0001p3
 LDC1000.h, 344
RPMAX0001p7
 LDC1000.h, 344
RPMAX0002
 LDC1000.h, 344
RPMAX0003
 LDC1000.h, 344
RPMAX0004
 LDC1000.h, 344
RPMAX0005
 LDC1000.h, 344
RPMAX0007
 LDC1000.h, 344
RPMAX0009

LDC1000.h, 344
RPMAX0012
 LDC1000.h, 344
RPMAX0016
 LDC1000.h, 344
RPMAX0021
 LDC1000.h, 344
RPMAX0027
 LDC1000.h, 344
RPMAX0038
 LDC1000.h, 344
RPMAX0048
 LDC1000.h, 344
RPMAX0064
 LDC1000.h, 344
RPMAX0083
 LDC1000.h, 344
RPMAX0109
 LDC1000.h, 344
RPMAX0145
 LDC1000.h, 344
RPMAX0193
 LDC1000.h, 344
RPMAX0249
 LDC1000.h, 344
RPMAX0349
 LDC1000.h, 344
RPMAX0436
 LDC1000.h, 344
RPMAX0581
 LDC1000.h, 344
RPMAX0747
 LDC1000.h, 343
RPMAX0981
 LDC1000.h, 343
RPMAX1308
 LDC1000.h, 343
RPMAX1745
 LDC1000.h, 343
RPMAX2243
 LDC1000.h, 343
RPMAX3141
 LDC1000.h, 343
RPMAX3936
 LDC1000.h, 343
RPMIN0000
 LDC1000.h, 345
RPMIN0001p0
 LDC1000.h, 345
RPMIN0001p3
 LDC1000.h, 345
RPMIN0001p7
 LDC1000.h, 345
RPMIN0002
 LDC1000.h, 345
RPMIN0003
 LDC1000.h, 345
RPMIN0004
 LDC1000.h, 345
 RPMIN0005
 LDC1000.h, 345
 RPMIN0007
 LDC1000.h, 345
 RPMIN0009
 LDC1000.h, 345
 RPMIN0012
 LDC1000.h, 345
 RPMIN0016
 LDC1000.h, 345
 RPMIN0021
 LDC1000.h, 345
 RPMIN0027
 LDC1000.h, 345
 RPMIN0038
 LDC1000.h, 345
 RPMIN0048
 LDC1000.h, 345
 RPMIN0064
 LDC1000.h, 345
 RPMIN0083
 LDC1000.h, 345
 RPMIN0109
 LDC1000.h, 345
 RPMIN0145
 LDC1000.h, 345
 RPMIN0249
 LDC1000.h, 345
 RPMIN0349
 LDC1000.h, 345
 RPMIN0436
 LDC1000.h, 345
 RPMIN0581
 LDC1000.h, 345
 RPMIN0747
 LDC1000.h, 345
 RPMIN0981
 LDC1000.h, 345
 RPMIN1308
 LDC1000.h, 345
 RPMIN1745
 LDC1000.h, 345
 RPMIN2243
 LDC1000.h, 345
 RPMIN3141
 LDC1000.h, 345
 RPMIN3926
 LDC1000.h, 345
 SECURITY_KEY
 wifi.h, 412
 SECURITY_KEY_FILE_CONNECT
 wifi.h, 413
 SECURITY_TYPE
 wifi.h, 413
 SELF_TEST

ADXL355.h, 28
SEND_DATA_LORA
 STARPORTS_App.h, 395
SEND_DATA_WIFI
 STARPORTS_App.h, 395
sendUdpClient
 wifi.c, 398
Sensors.c, 361
 Add_s16Data2Packet, 362
 Add_s32Data2Packet, 363
 GetSensorData, 364
 i2c, 367
 IniPacket, 367
 MyADXL, 367
 MyBME, 367
 MyLDC, 367
 MyNode, 368
 MyTMP006, 368
 MyVbat, 368
 spi, 368
 Timer1Event, 368
 uart0, 368
 wd, 368
Sensors.h, 372
 Add_s16Data2Packet, 372
 Add_s32Data2Packet, 373
 GetSensorData, 374
 IniPacket, 376
SERIALWIFI_PORT
 wifi.h, 413
SET_STATUS_BIT
 wifi.h, 413
SetMOD_TMP006
 hal_TMP006.c, 303
 hal_TMP006.h, 308
Setup_Abp_Lora
 hal_LORA.c, 226
 hal_LORA.h, 268
Setup_Otaa_Lora
 hal_LORA.c, 227
 hal_LORA.h, 269
SHUTDOWN
 STARPORTS_App.h, 393
SimpleLinkFatalErrorHandler
 wifi.c, 399
 wifi.h, 414
SimpleLinkGeneralEventHandler
 wifi.c, 400
 wifi.h, 415
SimpleLinkHttpServerEventHandler
 wifi.c, 400
 wifi.h, 416
SimpleLinkNetAppEventHandler
 wifi.c, 401
 wifi.h, 416
SimpleLinkNetAppRequestEventHandler
 wifi.c, 401
 wifi.h, 417
SimpleLinkNetAppRequestMemFreeEventHandler
 wifi.c, 401
 wifi.h, 417
SimpleLinkSocketTriggerEventHandler
 wifi.c, 402
 wifi.h, 417
SimpleLinkSockEventHandler
 wifi.c, 402
 wifi.h, 418
SimpleLinkWlanEventHandler
 wifi.c, 403
 wifi.h, 418
SL_SOCKET_ERROR
 wifi.h, 413
SLEEP
 BME280.h, 61
spi
 Sensors.c, 368
 STARPORTS_App.c, 383
SPI3W_OFF
 BME280.h, 63
SPI3W_ON
 BME280.h, 63
SPI_As_GPIO_Low
 hal_GPIO.c, 190
 hal_GPIO.h, 202
SPI_config
 CC3220SF_STARPORTS.c, 99
SPI_count
 CC3220SF_STARPORTS.c, 99
SPI_CS_Disable
 hal_GPIO.c, 190
 hal_GPIO.h, 202
SPI_read_16bits
 hal_SPI.c, 282
 hal_SPI.h, 288
SPI_read_8bits
 hal_SPI.c, 283
 hal_SPI.h, 289
SPI_write_16bits
 hal_SPI.c, 283
 hal_SPI.h, 290
SPI_write_8bits
 hal_SPI.c, 284
 hal_SPI.h, 290
spiCC3220SDMAHWAttrs
 CC3220SF_STARPORTS.c, 99
spiCC3220SDMAObjects
 CC3220SF_STARPORTS.c, 99
spiCC3220SDMAscratchBuf
 CC3220SF_STARPORTS.c, 99
SSID_NAME
 wifi.h, 413
ST1
 ADXL355.h, 25
ST2
 ADXL355.h, 25
st_readFileChangeWakeUp

file_system.c, 134
 file_system.h, 162
 st_readFileDnCntr
 file_system.c, 135
 file_system.h, 163
 st_read FileMode
 file_system.c, 136
 file_system.h, 164
 st_readFileNBoot
 file_system.c, 137
 file_system.h, 164
 st_readFileNCycles
 file_system.c, 137
 file_system.h, 165
 st_readFileNFails
 file_system.c, 138
 file_system.h, 166
 st_readFileNodeId
 file_system.c, 139
 file_system.h, 167
 st_readFileSSID
 file_system.c, 140
 file_system.h, 167
 st_readFileUpCntr
 file_system.c, 140
 file_system.h, 168
 st_readFileWakeUp
 file_system.c, 141
 file_system.h, 169
 STANDBY
 ADXL355.h, 26
 STARPORTS_App.c, 377
 i2c, 382
 LPDSOut, 382
 mainThread, 379
 MyADXL, 382
 MyBME, 382
 MyLDC, 383
 MyNode, 383
 MyTMP006, 383
 MyVbat, 383
 spi, 383
 TIMEOUT_MS, 378
 Timer0Event, 383
 Timer1Event, 383
 uart0, 383
 WakeupFromLPDS, 382
 wd, 384
 STARPORTS_App.h, 388
 ADXL355_ID, 391
 APPEUI, 391
 APPKEY, 391
 APPSKEY, 391
 BME280_ID, 391
 CONFIG_LORA, 395
 CONFIG_WIFI, 395
 CONTINUE, 392
 DEVADDR, 392
 DEVEUI, 392
 GET_NEW_PARAMS, 395
 GET_PARAMS, 395
 GET_SENSOR_DATA, 395
 LDC1000_ID, 392
 MODE_AP_WIFI, 392
 MODE_NORMAL_LORA, 392
 MODE_NORMAL_WIFI, 392
 MODE_STORAGE_LORA, 392
 MODE_STORAGE_WIFI, 393
 Node_states_t, 394
 NODEID, 393
 NodeState, 395
 NUMBER_SENSORS, 393
 NWKSKEY, 393
 POWERUP, 395
 RNW_LSB, 393
 RNW_MSB, 393
 SEND_DATA_LORA, 395
 SEND_DATA_WIFI, 395
 SHUTDOWN, 393
 STORE_DATA, 395
 STORE_NEW_PARAMS, 395
 TMP006_ID, 393
 VBAT_ID, 394
 Startup_ADC
 hal_ADC.c, 178
 hal_ADC.h, 180
 Startup_Continuous_Timer
 hal_Timer.c, 293
 hal_Timer.h, 298
 Startup_I2C
 hal_I2C.c, 205
 hal_I2C.h, 209
 Startup_Oneshot_Timer
 hal_Timer.c, 294
 hal_Timer.h, 299
 Startup_SPI
 hal_SPI.c, 285
 hal_SPI.h, 291
 Startup_UART
 hal_UART.c, 313
 hal_UART.h, 320
 Startup_Watchdog
 hal_WD.c, 322
 hal_WD.h, 325
 STATUS_BIT_CONNECTION
 wifi.h, 414
 STATUS_BIT_IP_ACQUIRED
 wifi.h, 414
 STBY_MODE
 LDC1000.h, 342
 STORE_DATA
 STARPORTS_App.h, 395
 STORE_NEW_PARAMS
 STARPORTS_App.h, 395
 SUCCESS_ABPLORA
 hal_LORA.h, 252

SUCCESS_CONF_TMP006
 hal_TMP006.h, 306

SUCCESS_CONNECT_WIFI
 wifi.h, 413

SUCCESS_OTAA_LORA
 hal_LORA.h, 253

SUCCESS_RXDELAY1_SET
 hal_LORA.h, 253

SUCCESS_SET_DEVADDR
 hal_LORA.h, 253

SUCCESS_SET_DNCTR
 hal_LORA.h, 253

SUCCESS_SET_NWSKEY
 hal_LORA.h, 253

SUCCESS_SET_PWRIDX
 hal_LORA.h, 253

SUCCESS_SET_UPCTR
 hal_LORA.h, 253

SUCCESS_TX_MAC_RX
 hal_LORA.h, 253

SUCCESS_TX_MAC_TX
 hal_LORA.h, 254

SYNC
 ADXL355.h, 28

Sys_FactoryReset
 hal_LORA.c, 228
 hal_LORA.h, 270

Sys_Sleep
 hal_LORA.c, 229
 hal_LORA.h, 271

T_SB_0p5
 BME280.h, 64

T_SB_10
 BME280.h, 64

T_SB_1000
 BME280.h, 64

T_SB_125
 BME280.h, 64

T_SB_20
 BME280.h, 64

T_SB_250
 BME280.h, 64

T_SB_500
 BME280.h, 64

T_SB_62p5
 BME280.h, 64

TEMP1
 ADXL355.h, 27

TEMP2
 ADXL355.h, 27

TEMP_LSB
 BME280.h, 63

TEMP_MSB
 BME280.h, 63

TEMP_OFF
 ADXL355.h, 26

TEMP_XLSB
 BME280.h, 63

THR_HIGH_LSB
 LDC1000.h, 342

THR_HIGH_MSB
 LDC1000.h, 342

THR_LOW_LSB
 LDC1000.h, 342

THR_LOW_MSB
 LDC1000.h, 342

TI_DRIVERS_UART_DMA
 CC3220SF_STARPORTS.c, 93

TIMEOUT_MS
 STARPORTS_App.c, 378

timer0Callback
 hal_Timer.c, 294
 hal_Timer.h, 300

Timer0Event
 hal_Timer.c, 296
 STARPORTS_App.c, 383

timer1Callback
 hal_Timer.c, 296
 hal_Timer.h, 300

Timer1Event
 hal_Timer.c, 296
 Sensors.c, 368
 STARPORTS_App.c, 383

Timer_config
 CC3220SF_STARPORTS.c, 100

Timer_count
 CC3220SF_STARPORTS.c, 100

timerCC3220SFHWAttrs
 CC3220SF_STARPORTS.c, 100

timerCC3220SFObjets
 CC3220SF_STARPORTS.c, 100

TMP006_Data, 391

TMP006_ID
 STARPORTS_App.h, 393

TOD0
 DS1374.h, 127

TOD1
 DS1374.h, 127

TOD2
 DS1374.h, 127

TOD3
 DS1374.h, 127

Try_Join_Lora_Gateway
 hal_LORA.c, 229
 hal_LORA.h, 272

Tx_Cnf_Lora
 hal_LORA.c, 230
 hal_LORA.h, 273

Tx_Uncnf_Lora
 hal_LORA.c, 232
 hal_LORA.h, 274

u20_to_s32
 ADXL355.c, 16
 ADXL355.h, 38

uart0
 file_system.c, 150

hal_LORA.c, 234
 hal_UART.c, 314
 Sensors.c, 368
 STARPORTS_App.c, 383
UART_config
 CC3220SF_STARPORTS.c, 101
UART_count
 CC3220SF_STARPORTS.c, 101
UART_PRINT
 hal_UART.h, 317
UART_resetInputBuffer
 hal_UART.c, 313
 hal_UART.h, 321
uartCC3220SHWAttrs
 CC3220SF_STARPORTS.c, 101
uartCC3220SObjects
 CC3220SF_STARPORTS.c, 101
uartCC3220SRingBuffer
 CC3220SF_STARPORTS.c, 102
udmaCC3220SHWAttrs
 CC3220SF_STARPORTS.c, 102
udmaCC3220SObject
 CC3220SF_STARPORTS.c, 102
UDMACC32XX_config
 CC3220SF_STARPORTS.c, 102
Uint8Array2Char
 hal_LORA.c, 233
 hal_LORA.h, 275
UPDATE
 BME280.h, 64

Vbat_Data, 391
VBAT_ID
 STARPORTS_App.h, 394
vsnprintf
 hal_UART.c, 314

WAKEUP_DIS_MASK
 LDC1000.h, 346
WAKEUP_EN
 LDC1000.h, 341
WakeupFromLPDS
 CC3220SF_STARPORTS.c, 94
 STARPORTS_App.c, 382
Watchdog_config
 CC3220SF_STARPORTS.c, 102
Watchdog_count
 CC3220SF_STARPORTS.c, 102
watchdogCallback
 hal_WD.c, 323
watchdogCC3220SHWAttrs
 CC3220SF_STARPORTS.c, 103
watchdogCC3220SObjects
 CC3220SF_STARPORTS.c, 103
wd
 Sensors.c, 368
 STARPORTS_App.c, 384
WDALMB0
 DS1374.h, 127

WDALMB1
 DS1374.h, 127
WDALMB2
 DS1374.h, 127
wifi.c, 396
 prepareDataFrame, 398
 sendUdpClient, 398
 SimpleLinkFatalErrorHandler, 399
 SimpleLinkGeneralEventHandler, 400
 SimpleLinkHttpServerEventHandler, 400
 SimpleLinkNetAppEventHandler, 401
 SimpleLinkNetAppRequestEventHandler, 401
 SimpleLinkNetAppRequestMemFreeEventHandler, 401
 SimpleLinkSocketTriggerEventHandler, 402
 SimpleLinkSockEventHandler, 402
 SimpleLinkWlanEventHandler, 403
 wlanConf, 403
 wlanConnect, 404
 wlanConnectFromFile, 404
wifi.h, 408
 ASSERT_ON_ERROR, 411
 CLR_STATUS_BIT, 411
 DEST_IP_ADDR, 411
 e_StatusBits, 413
 ERROR_CONNECT_WIFI, 411
 FRAME_LENGTH, 411
 GET_STATUS_BIT, 411
 IS_CONNECTED, 411
 IS_IP_ACQUIRED, 412
 LOOP_FOREVER, 412
 NUM_OF_PKT, 412
 PORT, 412
 PORT_UDP, 412
 PowerMeasure_CB, 421
 prepareDataFrame, 414
 SECURITY_KEY, 412
 SECURITY_KEY_FILE_CONNECT, 413
 SECURITY_TYPE, 413
 SERIALWIFI_PORT, 413
 SET_STATUS_BIT, 413
 SimpleLinkFatalErrorHandler, 414
 SimpleLinkGeneralEventHandler, 415
 SimpleLinkHttpServerEventHandler, 416
 SimpleLinkNetAppEventHandler, 416
 SimpleLinkNetAppRequestEventHandler, 417
 SimpleLinkNetAppRequestMemFreeEventHandler, 417
 SimpleLinkSocketTriggerEventHandler, 417
 SimpleLinkSockEventHandler, 418
 SimpleLinkWlanEventHandler, 418
 SL_SOCKET_ERROR, 413
 SSID_NAME, 413
 STATUS_BIT_CONNECTION, 414
 STATUS_BIT_IP_ACQUIRED, 414
 SUCCESS_CONNECT_WIFI, 413
 wlanConf, 419
 wlanConnect, 419

wlanConnectFromFile, [420](#)
wlanConf
 wifi.c, [403](#)
 wifi.h, [419](#)
wlanConnect
 wifi.c, [404](#)
 wifi.h, [419](#)
wlanConnectFromFile
 wifi.c, [404](#)
 wifi.h, [420](#)
writeChangeWakeUp
 file_system.c, [142](#)
 file_system.h, [169](#)
writeDnCntr
 file_system.c, [143](#)
 file_system.h, [170](#)
writeMode
 file_system.c, [143](#)
 file_system.h, [171](#)
writeNBoot
 file_system.c, [144](#)
 file_system.h, [172](#)
writeNCycles
 file_system.c, [145](#)
 file_system.h, [172](#)
writeNFails
 file_system.c, [146](#)
 file_system.h, [173](#)
writeNodeld
 file_system.c, [147](#)
 file_system.h, [174](#)
writeSSID
 file_system.c, [147](#)
 file_system.h, [175](#)
writeUpCntr
 file_system.c, [148](#)
 file_system.h, [175](#)
writeWakeUp
 file_system.c, [149](#)
 file_system.h, [176](#)

XDATA1
 ADXL355.h, [27](#)
XDATA2
 ADXL355.h, [27](#)
XDATA3
 ADXL355.h, [27](#)
XIN
 LDC1000.h, [341](#)
XIN_XOUT
 LDC1000.h, [341](#)

YDATA1
 ADXL355.h, [27](#)
YDATA2
 ADXL355.h, [27](#)
YDATA3
 ADXL355.h, [27](#)