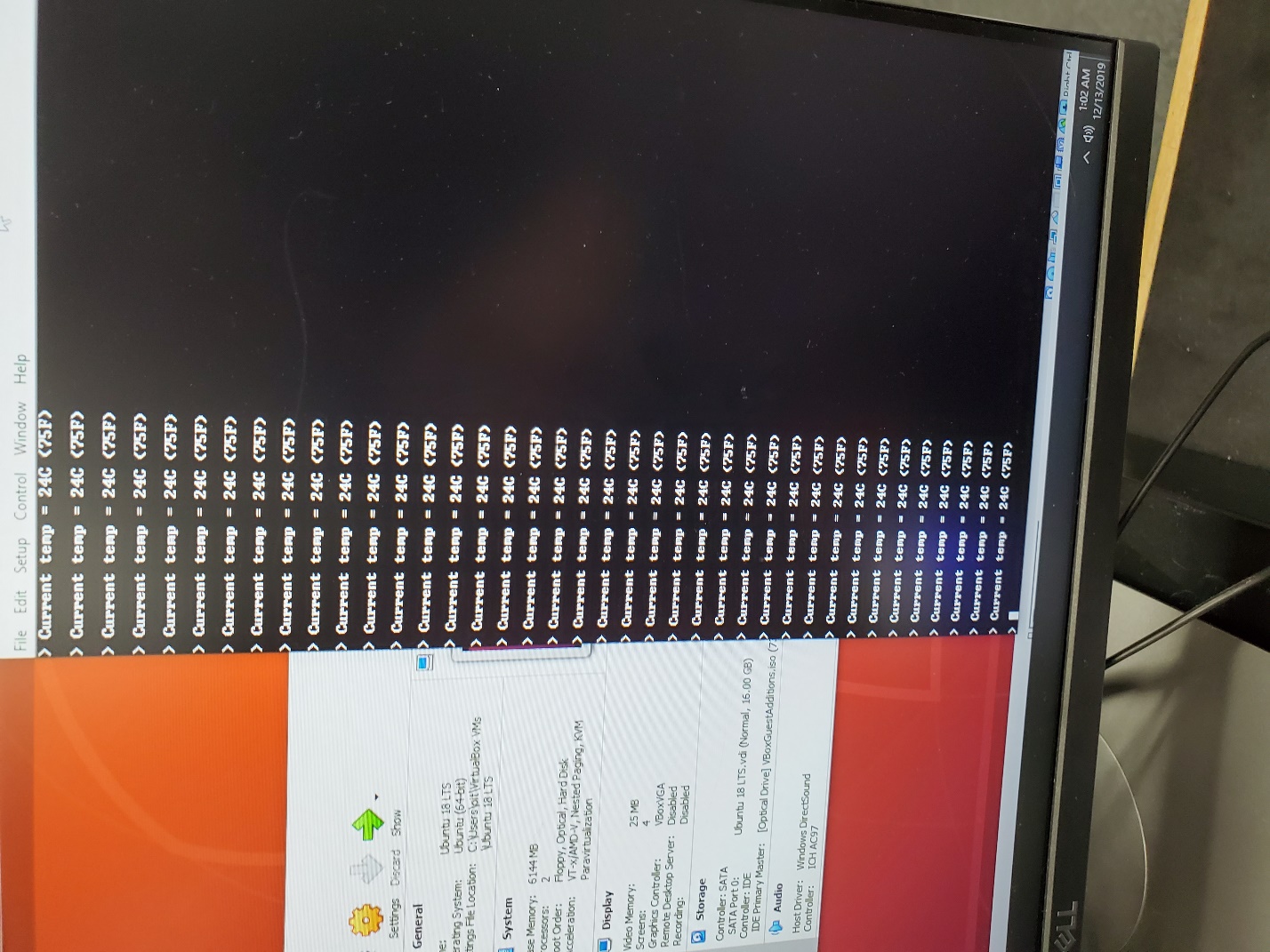
**FINAL PROJECT: Ivan Soto**

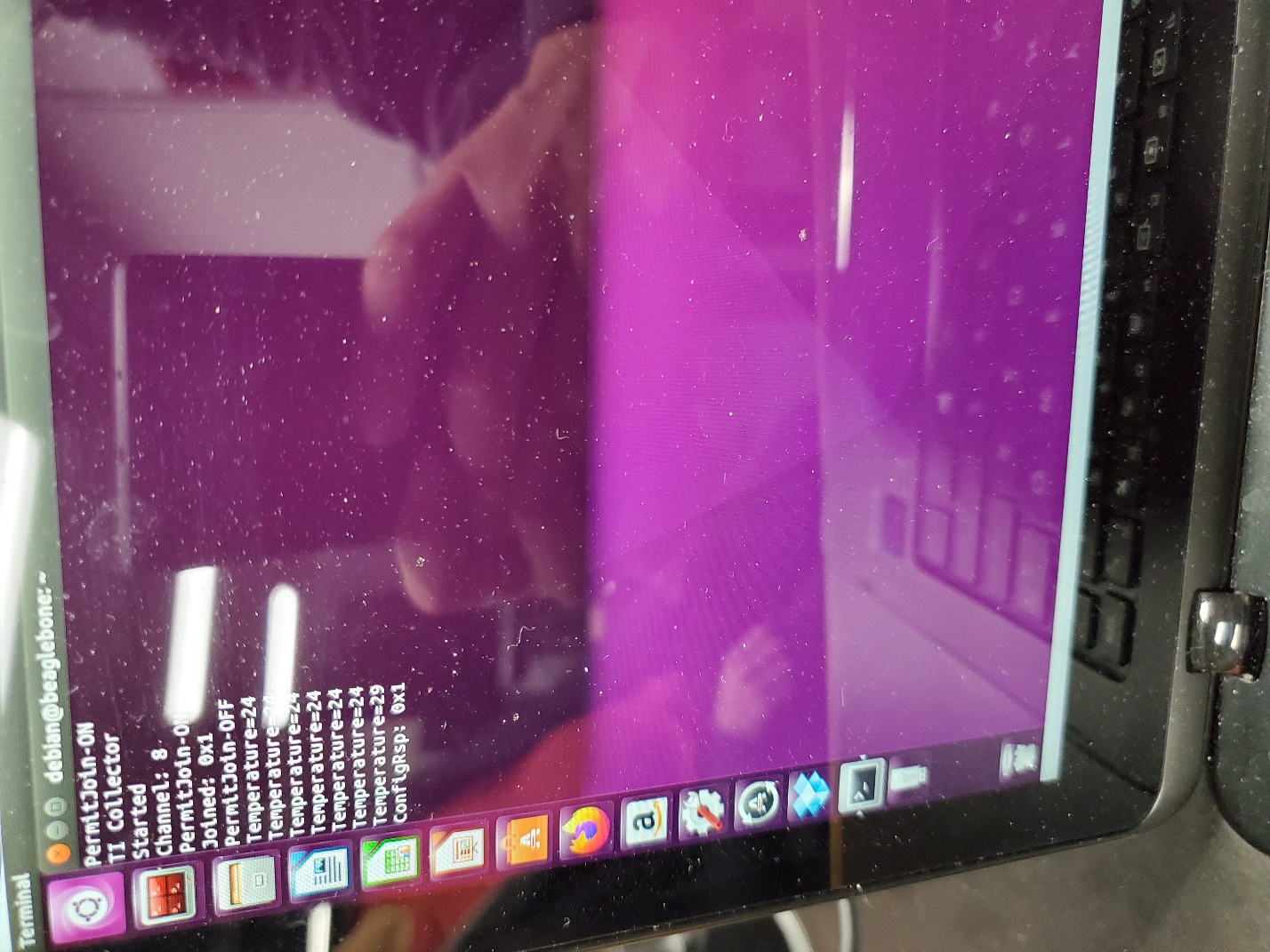
Youtube Links:

<https://www.youtube.com/watch?v=uJfJHFsUyqM>

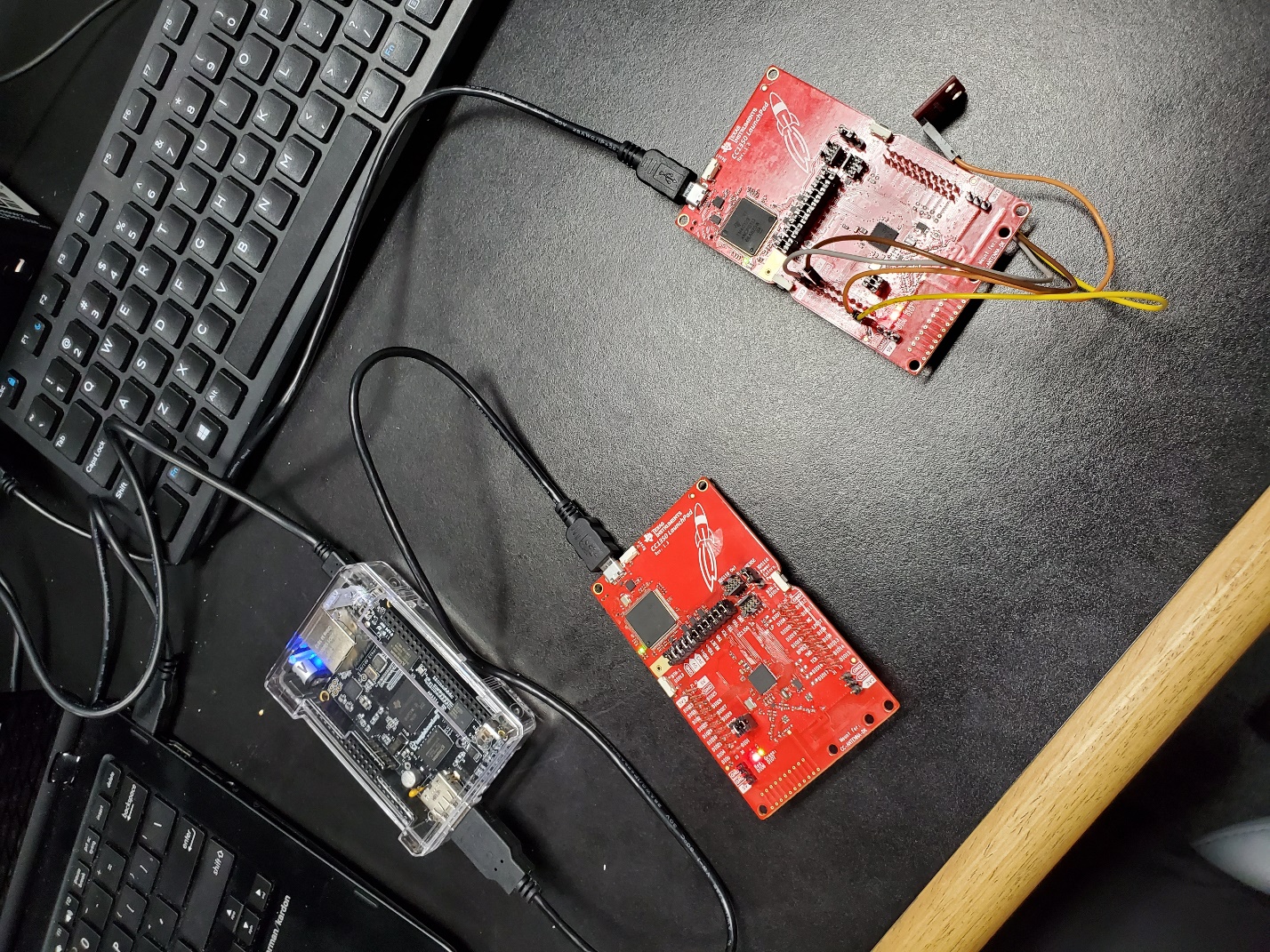
Partner: Jett Guerrero



Temperature being displayed by pressing ‘t’ .



This reading comes from the beaglebone black.

Project Setup

**Modified Code: In this lab, I was the collector.**

**// Insert code here**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@file config.h

@brief TI-15.4 Stack configuration parameters for Collector applications

Group: WCS LPC

Target Device: cc13x0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Copyright (c) 2016-2019, Texas Instruments Incorporated

All rights reserved.

Redistribution and use in source and binary forms, with or without

modification, are permitted provided that the following conditions

are met:

\* Redistributions of source code must retain the above copyright

notice, this list of conditions and the following disclaimer.

\* Redistributions in binary form must reproduce the above copyright

notice, this list of conditions and the following disclaimer in the

documentation and/or other materials provided with the distribution.

\* Neither the name of Texas Instruments Incorporated nor the names of

its contributors may be used to endorse or promote products derived

from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"

AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,

THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR

PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR

CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,

EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,

PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;

OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,

WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR

OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,

EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**#ifndef** CONFIG\_H

**#define** CONFIG\_H

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Includes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**#include** "api\_mac.h"

**#ifdef** \_\_cplusplus

**extern** "C"

{

**#endif**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Constants and definitions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* config parameters \*/

/\*! Security Enable - set to true to turn on security \*/

**#define** CONFIG\_SECURE **true**

/\*! PAN ID \*/

**#define** CONFIG\_PAN\_ID 0xFFFF

/\*! Coordinator short address \*/

**#define** CONFIG\_COORD\_SHORT\_ADDR 0xAABB

/\*! FH disabled as default \*/

**#define** CONFIG\_FH\_ENABLE **false**

/\*! maximum beacons possibly received \*/

**#define** CONFIG\_MAX\_BEACONS\_RECD 200

/\*! maximum devices in association table \*/

**#define** CONFIG\_MAX\_DEVICES 50

/\*!

Setting beacon order to 15 will disable the beacon, 8 is a good value for

beacon mode

\*/

**#define** CONFIG\_MAC\_BEACON\_ORDER 15

/\*!

Setting superframe order to 15 will disable the superframe, 8 is a good value

for beacon mode

\*/

**#define** CONFIG\_MAC\_SUPERFRAME\_ORDER 15

/\*! Setting for Phy ID \*/

**#define** CONFIG\_PHY\_ID (APIMAC\_STD\_US\_915\_PHY\_1)

**#if** ((CONFIG\_PHY\_ID >= APIMAC\_MRFSK\_STD\_PHY\_ID\_BEGIN) && (CONFIG\_PHY\_ID <= APIMAC\_MRFSK\_STD\_PHY\_ID\_END))

/\*! Setting for channel page \*/

**#define** CONFIG\_CHANNEL\_PAGE (APIMAC\_CHANNEL\_PAGE\_9)

**#elif** ((CONFIG\_PHY\_ID >= APIMAC\_MRFSK\_GENERIC\_PHY\_ID\_BEGIN) && (CONFIG\_PHY\_ID <= APIMAC\_MRFSK\_GENERIC\_PHY\_ID\_END))

/\*! Setting for channel page \*/

**#define** CONFIG\_CHANNEL\_PAGE (APIMAC\_CHANNEL\_PAGE\_10)

**#else**

**#error** "PHY ID is wrong."

**#endif**

**#if** (defined(CC1312R1\_LAUNCHXL))

**#if**((CONFIG\_PHY\_ID == APIMAC\_GENERIC\_CHINA\_433\_PHY\_128) || (CONFIG\_PHY\_ID == APIMAC\_GENERIC\_CHINA\_LRM\_433\_PHY\_130))

**#error** "Error: 433 MHz Operation is not supported on 1312 board!"

**#endif**

**#endif**

/\*! MAC Parameter \*/

/\*! Min BE - Minimum Backoff Exponent \*/

**#define** CONFIG\_MIN\_BE 3

/\*! Max BE - Maximum Backoff Exponent \*/

**#define** CONFIG\_MAX\_BE 5

/\*! MAC MAX CSMA Backoffs \*/

**#define** CONFIG\_MAC\_MAX\_CSMA\_BACKOFFS 4

/\*! macMaxFrameRetries - Maximum Frame Retries \*/

**#define** CONFIG\_MAX\_RETRIES 3

/\*! Application traffic profile \*/

**#if** (((CONFIG\_PHY\_ID >= APIMAC\_MRFSK\_STD\_PHY\_ID\_BEGIN) && (CONFIG\_PHY\_ID <= APIMAC\_MRFSK\_GENERIC\_PHY\_ID\_BEGIN)) || \

((CONFIG\_PHY\_ID >= APIMAC\_GENERIC\_US\_915\_PHY\_132) && (CONFIG\_PHY\_ID <= APIMAC\_GENERIC\_ETSI\_863\_PHY\_133)))

/\*!

Reporting Interval - in milliseconds to be set on connected devices using

configuration request messages

\*/ // used to be 90000 now 1000

**#define** CONFIG\_REPORTING\_INTERVAL 90000

/\*!

Polling interval in milliseconds to be set on connected devices using

configuration request messages. Must be greater than or equal to default

polling interval set on sensor devices

\*/ // used to be 6000 now 100

**#define** CONFIG\_POLLING\_INTERVAL 6000

/\*!

Time interval in ms between tracking message intervals

\*/

**#define** TRACKING\_DELAY\_TIME 60000

**#else**

/\*!

Reporting Interval - in milliseconds to be set on connected devices using

configuration request messages

\*/

**#define** CONFIG\_REPORTING\_INTERVAL 300000

/\*!

Polling interval in milliseconds to be set on connected devices using

configuration request messages. Must be greater than or equal to default

polling interval set on sensor devices

\*/

**#define** CONFIG\_POLLING\_INTERVAL 60000

/\*!

Time interval in ms between tracking message intervals

\*/

**#define** TRACKING\_DELAY\_TIME 300000

**#endif**

/\*! scan duration

\* scan type = MAC\_MPM\_SCAN\_NBPAN (see mac\_api.h):

\* scan duration = aBaseSlotDuration \* 2 \* CONFIG\_SCAN\_DURATION

\*

\* scan type = MAC\_MPM\_SCAN\_BPAN (see mac\_api.h):

\* scan duration = aBaseSuperframeDuration \* 2 \* CONFIG\_SCAN\_DURATION

\* other types

\* scan duration = aBaseSuperframeDuration \* (1 + 2 \* CONFIG\_SCAN\_DURATION)

\*/

**#define** CONFIG\_SCAN\_DURATION 5

/\*!

Range Extender Mode setting.

The following modes are available.

APIMAC\_NO\_EXTENDER - does not have PA/LNA

APIMAC\_HIGH\_GAIN\_MODE - high gain mode

To enable CC1190, use

#define CONFIG\_RANGE\_EXT\_MODE APIMAC\_HIGH\_GAIN\_MODE

\*/

**#define** CONFIG\_RANGE\_EXT\_MODE APIMAC\_NO\_EXTENDER

/\*! Setting Default Key\*/

**#define** KEY\_TABLE\_DEFAULT\_KEY {0x12, 0x34, 0x56, 0x78, 0x9a, 0xbc, 0xde, 0xf0,\

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}

/\*!

Channel mask used when CONFIG\_FH\_ENABLE is false.

Each bit indicates if the corresponding channel is to be scanned

First byte represents channels 0 to 7 and the last byte represents

channels 128 to 135.

For byte zero in the bit mask, LSB representing Ch0.

For byte 1, LSB represents Ch8 and so on.

e.g., 0x01 0x10 represents Ch0 and Ch12 are included.

The default of 0x0F represents channels 0-3 are selected.

APIMAC\_STD\_US\_915\_PHY\_1 (50kbps/2-FSK/915MHz band) has channels 0 - 128.

APIMAC\_STD\_ETSI\_863\_PHY\_3 (50kbps/2-FSK/863MHz band) has channels 0 - 33.

APIMAC\_GENERIC\_CHINA\_433\_PHY\_128 (50kbps/2-FSK/433MHz band) has channels 0 - 6.

\*/

**#define** CONFIG\_CHANNEL\_MASK { 0x00, 0x03, 0x00, 0x00, 0x00, 0x00, \

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \

0x00, 0x00, 0x00, 0x00, 0x00 }

/\*!

Channel mask used when CONFIG\_FH\_ENABLE is true.

Represents the list of channels on which the device can hop.

The actual sequence used shall be based on DH1CF function.

It is represented as a bit string with LSB representing Ch0.

e.g., 0x01 0x10 represents Ch0 and Ch12 are included.

\*/

**#define** CONFIG\_FH\_CHANNEL\_MASK { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, \

0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, \

0xFF, 0xFF, 0xFF, 0xFF, 0xFF }

/\*!

List of channels to target the Async frames

It is represented as a bit string with LSB representing Ch0

e.g., 0x01 0x10 represents Ch0 and Ch12 are included

It should cover all channels that could be used by a target device in its

hopping sequence. Channels marked beyond number of channels supported by

PHY Config will be excluded by stack. To avoid interference on a channel,

it should be removed from Async Mask and added to exclude channels

(CONFIG\_CHANNEL\_MASK).

\*/

**#define** FH\_ASYNC\_CHANNEL\_MASK { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, \

0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, \

0xFF, 0xFF, 0xFF, 0xFF, 0xFF }

/\* FH related config variables \*/

/\*!

The number of non sleepy channel hopping end devices to be supported.

It is to be noted that the total number of non sleepy devices supported

must be less than 50. Stack will allocate memory proportional

to the number of end devices requested.

\*/

**#define** FH\_NUM\_NON\_SLEEPY\_HOPPING\_NEIGHBORS 5

/\*!

The number of non sleepy fixed channel end devices to be supported.

It is to be noted that the total number of non sleepy devices supported

must be less than 50. Stack will allocate memory proportional

to the number of end devices requested.

\*/

**#define** FH\_NUM\_NON\_SLEEPY\_FIXED\_CHANNEL\_NEIGHBORS 5

/\*!

Dwell time: The duration for which the collector will

stay on a specific channel before hopping to next channel.

\*/

**#define** CONFIG\_DWELL\_TIME 250

/\*!

FH Application Broadcast Msg generation interval in ms.

Value should be set at least greater than 200 ms,

\*/

**#define** FH\_BROADCAST\_INTERVAL 10000

/\*! FH Broadcast dwell time. If set to 0, it shall disable broadcast hopping and

\* broadcast message transmissions in FH Mode \*/

**#define** FH\_BROADCAST\_DWELL\_TIME 100

**#if** (((CONFIG\_PHY\_ID >= APIMAC\_MRFSK\_STD\_PHY\_ID\_BEGIN) && (CONFIG\_PHY\_ID <= APIMAC\_MRFSK\_GENERIC\_PHY\_ID\_BEGIN)) || \

((CONFIG\_PHY\_ID >= APIMAC\_GENERIC\_US\_915\_PHY\_132) && (CONFIG\_PHY\_ID <= APIMAC\_GENERIC\_ETSI\_863\_PHY\_133)))

/\*!

The minimum trickle timer window for PAN Advertisement,

and PAN Configuration frame transmissions.

Recommended to set this to half of PAS/PCS MIN Timer

\*/

**#define** CONFIG\_TRICKLE\_MIN\_CLK\_DURATION 3000

/\*!

The maximum trickle timer window for PAN Advertisement,

and PAN Configuration frame transmissions.

\*/

**#define** CONFIG\_TRICKLE\_MAX\_CLK\_DURATION 6000

**#else**

/\*!

The minimum trickle timer window for PAN Advertisement,

and PAN Configuration frame transmissions.

Recommended to set this to half of PAS/PCS MIN Timer

\*/

**#define** CONFIG\_TRICKLE\_MIN\_CLK\_DURATION 30000

/\*!

The maximum trickle timer window for PAN Advertisement,

and PAN Configuration frame transmissions.

\*/

**#define** CONFIG\_TRICKLE\_MAX\_CLK\_DURATION 60000

**#endif**

/\*!

To enable Doubling of PA/PC trickle time,

useful when network has non sleepy nodes and

there is a requirement to use PA/PC to convey updated

PAN information. Note that when using option the CONFIG\_TRICKLE\_MIN\_CLK\_DURATION

and CONFIG\_TRICKLE\_MAX\_CLK\_DURATION should be set to a sufficiently large value.

Recommended values are 1 min and 16 min respectively.

\*/

**#define** CONFIG\_DOUBLE\_TRICKLE\_TIMER **false**

/\*! value for ApiMac\_FHAttribute\_netName \*/

**#define** CONFIG\_FH\_NETNAME {"FHTest"}

/\*!

Value for Transmit Power in dBm

For US and ETSI band, Default value is 10, allowed values are

-10, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 and 14dBm.

For China band, allowed values are 6, 10, 13, 14 and 15dBm.

For CC1190, allowed values are between 18, 23, 25, 26 and 27dBm.

When the nodes in the network are close to each other

lowering this value will help reduce saturation \*/

**#ifndef** DeviceFamily\_CC13X2

**#if** CONFIG\_RANGE\_EXT\_MODE

**#define** CONFIG\_TRANSMIT\_POWER 26

**#else**

**#if** ((CONFIG\_PHY\_ID == APIMAC\_GENERIC\_CHINA\_433\_PHY\_128) || (CONFIG\_PHY\_ID == APIMAC\_GENERIC\_CHINA\_LRM\_433\_PHY\_130))

**#define** CONFIG\_TRANSMIT\_POWER 14

**#else**

**#define** CONFIG\_TRANSMIT\_POWER 12

**#endif**

**#endif**

**#else** /\* DeviceFamily\_CC13X2 \*/

**#define** CONFIG\_TRANSMIT\_POWER 12

**#endif**

**#ifndef** DeviceFamily\_CC13X2

**#if** CONFIG\_RANGE\_EXT\_MODE

**#if** (CCFG\_FORCE\_VDDR\_HH == 1)

**#error** "CCFG\_FORCE\_VDDR\_HH should be 0"

**#endif**

**#else**

**#if** ((CONFIG\_PHY\_ID == APIMAC\_GENERIC\_CHINA\_433\_PHY\_128) || (CONFIG\_PHY\_ID == APIMAC\_GENERIC\_CHINA\_LRM\_433\_PHY\_130))

**#if** (CCFG\_FORCE\_VDDR\_HH == 0)

**#if** (CONFIG\_TRANSMIT\_POWER >= 15)

**#error** "CONFIG\_TRANSMIT\_POWER should be less than 15"

**#endif**

**#else**

**#if** (CONFIG\_TRANSMIT\_POWER < 15)

/\* In 433 MHz band when CCFG\_FORCE\_VDDR\_HH = 1, only possible value of transmit power is 15 \*/

**#error** "CONFIG\_TRANSMIT\_POWER should be 15"

**#endif**

**#endif**

**#else**

**#if** (CCFG\_FORCE\_VDDR\_HH == 0)

**#if** (CONFIG\_TRANSMIT\_POWER >= 14)

**#error** "CONFIG\_TRANSMIT\_POWER should be less than 14"

**#endif**

**#else**

**#if** (CONFIG\_TRANSMIT\_POWER < 14)

/\* In US and ETSI band when CCFG\_FORCE\_VDDR\_HH = 1, only possible value of transmit power is 14 \*/

**#error** "CONFIG\_TRANSMIT\_POWER should be 14"

**#endif**

**#endif**

**#endif**

**#endif**

**#else**

**#if** (CCFG\_FORCE\_VDDR\_HH == 1)

**#if** (CONFIG\_TRANSMIT\_POWER != 14)

/\* In US and ETSI band when CCFG\_FORCE\_VDDR\_HH = 1, only possible value of transmit power is 14 \*/

**#error** "CONFIG\_TRANSMIT\_POWER should be 14"

**#endif**

**#endif**

**#endif**

/\*!

\* Enable this mode for certfication.

\* For FH certification, CONFIG\_FH\_ENABLE should

\* also be enabled.

\*/

**#define** CERTIFICATION\_TEST\_MODE **false**

**#ifdef** POWER\_MEAS

/\*! Size of RAMP Data to be sent when POWER Test is enabled \*/

**#define** COLLECTOR\_TEST\_RAMP\_DATA\_SIZE 20

/\*!

Power profile to be used when Power MEAS is enabled.

Profile 1 - POLL\_ACK - Polling Only

Profile 2 - DATA\_ACK - 20 byte application data + ACK from sensor to collector

Profile 3 - POLL\_DATA - Poll + received Data from collector

Profile 4 - SLEEP - No Poll or Data. In Beacon mode, beacon RX would occur

\*/

**#define** POWER\_TEST\_PROFILE DATA\_ACK

**#endif**

/\* Check if all the necessary parameters have been set for FH mode \*/

**#if** CONFIG\_FH\_ENABLE

**#if** !defined(FEATURE\_ALL\_MODES) && !defined(FEATURE\_FREQ\_HOP\_MODE)

**#error** "Do you want to build image with frequency hopping mode? \

Define either FEATURE\_FREQ\_HOP\_MODE or FEATURE\_ALL\_MODES in features.h"

**#endif**

**#endif**

/\* Check if stack level security is enabled if application security is enabled \*/

**#if** CONFIG\_SECURE

**#if** !defined(FEATURE\_MAC\_SECURITY)

**#error** "Define FEATURE\_MAC\_SECURITY or FEATURE\_ALL\_MODES in features.h to \

be able to use security at application level"

**#endif**

**#endif**

/\* Set beacon order and superframe order to 15 for FH mode to avoid user error \*/

**#if** CONFIG\_FH\_ENABLE

**#if** (CONFIG\_MAC\_BEACON\_ORDER != 15) && (CONFIG\_MAC\_SUPERFRAME\_ORDER != 15)

**#error** "Do you want to build image with frequency hopping mode? \

If yes, CONFIG\_MAC\_BEACON\_ORDER and CONFIG\_MAC\_SUPERFRAME\_ORDER \

should both be set to 15"

**#endif**

**#endif**

**#ifdef** \_\_cplusplus

}

**#endif**

**#endif** /\* CONFIG\_H \*/

CODE FOR SENSOR:

File for config.h

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@file config.h

@brief TI-15.4 Stack configuration parameters for Sensor applications

Group: WCS LPC

Target Device: cc13x0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Copyright (c) 2016-2019, Texas Instruments Incorporated

All rights reserved.

Redistribution and use in source and binary forms, with or without

modification, are permitted provided that the following conditions

are met:

\* Redistributions of source code must retain the above copyright

notice, this list of conditions and the following disclaimer.

\* Redistributions in binary form must reproduce the above copyright

notice, this list of conditions and the following disclaimer in the

documentation and/or other materials provided with the distribution.

\* Neither the name of Texas Instruments Incorporated nor the names of

its contributors may be used to endorse or promote products derived

from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"

AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,

THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR

PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR

CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,

EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,

PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;

OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,

WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR

OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,

EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef CONFIG\_H

#define CONFIG\_H

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Includes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "api\_mac.h"

#ifdef \_\_cplusplus

extern "C"

{

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Constants and definitions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* config parameters \*/

/\*! Security Enable - set to true to turn on security \*/

#define CONFIG\_SECURE true

/\*! PAN ID \*/

#define CONFIG\_PAN\_ID 0xFFFF

/\*! FH disabled as default \*/

#define CONFIG\_FH\_ENABLE false

/\*! link quality \*/

#define CONFIG\_LINKQUALITY 1

/\*! percent filter \*/

#define CONFIG\_PERCENTFILTER 0xFF

/\*!

Beacon order, value of 15 indicates non beacon mode,

8 is a good value for beacon mode

\*/

#define CONFIG\_MAC\_BEACON\_ORDER 15

/\*!

Superframe order, value of 15 indicates non beacon mode,

8 is a good value for beacon mode

\*/

#define CONFIG\_MAC\_SUPERFRAME\_ORDER 15

/\*! Maximum number of message failure, to indicate sync loss \*/

#define CONFIG\_MAX\_DATA\_FAILURES 3

/\*!

Maximum number of attempts for association in FH mode

after reception of a PAN Config frame

\*/

#define CONFIG\_FH\_MAX\_ASSOCIATION\_ATTEMPTS 3

/\* Interval for scan backoff \*/

#define CONFIG\_SCAN\_BACKOFF\_INTERVAL 5000

/\* Interval for delay between orphan notifications \*/

#define CONFIG\_ORPHAN\_BACKOFF\_INTERVAL 300000

/\*! Setting for Phy ID \*/

#define CONFIG\_PHY\_ID (APIMAC\_STD\_US\_915\_PHY\_1)

/\*! MAC Parameter \*/

/\*! Min BE - Minimum Backoff Exponent \*/

#define CONFIG\_MIN\_BE 3

/\*! Max BE - Maximum Backoff Exponent \*/

#define CONFIG\_MAX\_BE 5

/\*! MAC MAX CSMA Backoffs \*/

#define CONFIG\_MAC\_MAX\_CSMA\_BACKOFFS 4

/\*! macMaxFrameRetries - Maximum Frame Retries \*/

#define CONFIG\_MAX\_RETRIES 3

#if ((CONFIG\_PHY\_ID >= APIMAC\_MRFSK\_STD\_PHY\_ID\_BEGIN) && (CONFIG\_PHY\_ID <= APIMAC\_MRFSK\_STD\_PHY\_ID\_END))

/\*! Setting for channel page \*/

#define CONFIG\_CHANNEL\_PAGE (APIMAC\_CHANNEL\_PAGE\_9)

#elif ((CONFIG\_PHY\_ID >= APIMAC\_MRFSK\_GENERIC\_PHY\_ID\_BEGIN) && (CONFIG\_PHY\_ID <= APIMAC\_MRFSK\_GENERIC\_PHY\_ID\_END))

/\*! Setting for channel page \*/

#define CONFIG\_CHANNEL\_PAGE (APIMAC\_CHANNEL\_PAGE\_10)

#else

#error "PHY ID is wrong."

#endif

#if (defined(CC1312R1\_LAUNCHXL))

#if((CONFIG\_PHY\_ID == APIMAC\_GENERIC\_CHINA\_433\_PHY\_128) || (CONFIG\_PHY\_ID == APIMAC\_GENERIC\_CHINA\_LRM\_433\_PHY\_130))

#error "Error: 433 MHz Operation is not supported on 1312 board!"

#endif

#endif

/\*! scan duration in seconds\*/

#define CONFIG\_SCAN\_DURATION 5

/\*!

Coordinator Short Address When Operating with FH Enabled.

\*/

#define FH\_COORD\_SHORT\_ADDR 0xAABB

/\*!

Range Extender Mode setting.

The following modes are available.

APIMAC\_NO\_EXTENDER - does not have PA/LNA

APIMAC\_HIGH\_GAIN\_MODE - high gain mode

To enable CC1190, use

#define CONFIG\_RANGE\_EXT\_MODE APIMAC\_HIGH\_GAIN\_MODE

\*/

#define CONFIG\_RANGE\_EXT\_MODE APIMAC\_NO\_EXTENDER

/\*! Setting Default Key\*/

#define KEY\_TABLE\_DEFAULT\_KEY {0x12, 0x34, 0x56, 0x78, 0x9a, 0xbc, 0xde, 0xf0,\

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}

/\*!

Channel mask used when CONFIG\_FH\_ENABLE is false.

Each bit indicates if the corresponding channel is to be scanned

First byte represents channels 0 to 7 and the last byte represents

channels 128 to 135.

For byte zero in the bit mask, LSB representing Ch0.

For byte 1, LSB represents Ch8 and so on.

e.g., 0x01 0x10 represents Ch0 and Ch12 are included.

The default of 0x0F represents channels 0-3 are selected.

APIMAC\_STD\_US\_915\_PHY\_1 (50kbps/2-FSK/915MHz band) has channels 0 - 128.

APIMAC\_STD\_ETSI\_863\_PHY\_3 (50kbps/2-FSK/863MHz band) has channels 0 - 33.

APIMAC\_GENERIC\_CHINA\_433\_PHY\_128 (50kbps/2-FSK/433MHz band) has channels 0 - 6.

\*/

//#define CONFIG\_CHANNEL\_MASK { 0x0F, 0x00, 0x00, 0x00, 0x00, 0x00, \

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \

0x00, 0x00, 0x00, 0x00, 0x00 }

#define CONFIG\_CHANNEL\_MASK { 0x00, 0x03, 0x00, 0x00, 0x00, 0x00, \

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \

0x00, 0x00, 0x00, 0x00, 0x00,}

/\*!

Channel mask used when CONFIG\_FH\_ENABLE is true.

Represents the list of channels on which the device can hop.

When CONFIG\_RX\_ON\_IDLE is true, the actual sequence will

be based on DH1CF function. When it is set to false, the sequence

shall be a linear hopping over available channels in ascending order and

shall be used to change channel during the join phase.

It is represented as a bit string with LSB representing Ch0.

e.g., 0x01 0x10 represents Ch0 and Ch12 are included.

\*/

#define CONFIG\_FH\_CHANNEL\_MASK { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, \

0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, \

0x00, 0x00, 0x00, 0x00, 0x00,}

/\* FH related config variables \*/

/\*!

List of channels to target the Async frames

It is represented as a bit string with LSB representing Ch0

e.g., 0x01 0x10 represents Ch0 and Ch12 are included

It should cover all channels that could be used by a target device in its

hopping sequence. Channels marked beyond number of channels supported by

PHY Config will be excluded by stack. To avoid interference on a channel,

it should be removed from Async Mask and added to exclude channels

(CONFIG\_CHANNEL\_MASK).

\*/

#define FH\_ASYNC\_CHANNEL\_MASK { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, \

0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, \

0xFF, 0xFF, 0xFF, 0xFF, 0xFF }

/\*! Rx on when idle, false for sleepy device, true for non sleepy device \*/

#define CONFIG\_RX\_ON\_IDLE false

/\*!

The number of non sleepy channel hopping end devices to be supported.

It is to be noted that the total number of non sleepy devices supported

must be less than 50. Stack will allocate memory proportional

to the number of end devices requested.

\*/

#define FH\_NUM\_NON\_SLEEPY\_HOPPING\_NEIGHBORS 2

/\*!

The number of non sleepy fixed channel end devices to be supported.

It is to be noted that the total number of non sleepy devices supported

must be less than 50. Stack will allocate memory proportional

to the number of end devices requested.

\*/

#define FH\_NUM\_NON\_SLEEPY\_FIXED\_CHANNEL\_NEIGHBORS 2

/\*!

Dwell Time: The duration for which a non sleepy end device shall

stay on a specific channel before hopping to next channel.

\*/

#define CONFIG\_DWELL\_TIME 250

#if (((CONFIG\_PHY\_ID >= APIMAC\_MRFSK\_STD\_PHY\_ID\_BEGIN) && (CONFIG\_PHY\_ID <= APIMAC\_MRFSK\_GENERIC\_PHY\_ID\_BEGIN)) || \

((CONFIG\_PHY\_ID >= APIMAC\_GENERIC\_US\_915\_PHY\_132) && (CONFIG\_PHY\_ID <= APIMAC\_GENERIC\_ETSI\_863\_PHY\_133)))

/\*! Default Polling interval in milliseconds. It will get updated upon reception

of a config request message \*/

#define CONFIG\_POLLING\_INTERVAL 6000

/\*! PAN Advertisement Solicit trickle timer duration in milliseconds \*/

#define CONFIG\_PAN\_ADVERT\_SOLICIT\_CLK\_DURATION 6000

/\*! PAN Config Solicit trickle timer duration in milliseconds \*/

#define CONFIG\_PAN\_CONFIG\_SOLICIT\_CLK\_DURATION 6000

/\*! Default Reporting Interval - in milliseconds. It will get updated upon

reception of a config request message \*/

//#define CONFIG\_REPORTING\_INTERVAL 180000

#define CONFIG\_REPORTING\_INTERVAL 45000

//#define CONFIG\_REPORTING\_INTERVAL 500

#else

/\*! Default Polling interval in milliseconds. It will get updated upon reception

of a config request message \*/

#define CONFIG\_POLLING\_INTERVAL 60000

/\*! PAN Advertisement Solicit trickle timer duration in milliseconds \*/

#define CONFIG\_PAN\_ADVERT\_SOLICIT\_CLK\_DURATION 60000

/\*! PAN Config Solicit trickle timer duration in milliseconds \*/

#define CONFIG\_PAN\_CONFIG\_SOLICIT\_CLK\_DURATION 60000

/\*! Default Reporting Interval - in milliseconds. It will get updated upon

reception of a config request message \*/

#define CONFIG\_REPORTING\_INTERVAL 600000

#endif

/\*! FH Poll/Sensor msg start time randomization window \*/

#define CONFIG\_FH\_START\_POLL\_DATA\_RAND\_WINDOW 10000

/\*! If enabled, the periodic sensor message shall be sent as a fixed size

\* packet of specified size. If set to 0, the periodic sensor message shall be

\* of type sensor data specified in smsgs.h

\*/

#define SENSOR\_TEST\_RAMP\_DATA\_SIZE 0

/\*! value for ApiMac\_FHAttribute\_netName \*/

#define CONFIG\_FH\_NETNAME {"FHTest"}

/\*! Range Extender is not supported in uBLE project \*/

#ifdef FEATURE\_UBLE

#if CONFIG\_RANGE\_EXT\_MODE

#error "CONFIG\_RANGE\_EXT\_MODE should be APIMAC\_NO\_EXTENDER"

#endif

#endif

/\*!

Value for Transmit Power in dBm

For US and ETSI band, Default value is 10, allowed values are

-10, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 and 14dBm.

For China band, allowed values are 6, 10, 13, 14 and 15dBm.

For CC1190, allowed values are between 18, 23, 25, 26 and 27dBm.

When the nodes in the network are close to each other

lowering this value will help reduce saturation \*/

#ifndef DeviceFamily\_CC13X2

#if CONFIG\_RANGE\_EXT\_MODE

#define CONFIG\_TRANSMIT\_POWER 26

#else

#if ((CONFIG\_PHY\_ID == APIMAC\_GENERIC\_CHINA\_433\_PHY\_128) || (CONFIG\_PHY\_ID == APIMAC\_GENERIC\_CHINA\_LRM\_433\_PHY\_130))

#define CONFIG\_TRANSMIT\_POWER 14

#else

#define CONFIG\_TRANSMIT\_POWER 12

#endif

#endif

#else /\* DeviceFamily\_CC13X2 \*/

#define CONFIG\_TRANSMIT\_POWER 12

#endif

#ifndef DeviceFamily\_CC13X2

#if CONFIG\_RANGE\_EXT\_MODE

#if (CCFG\_FORCE\_VDDR\_HH == 1)

#error "CCFG\_FORCE\_VDDR\_HH should be 0"

#endif

#else

#if ((CONFIG\_PHY\_ID == APIMAC\_GENERIC\_CHINA\_433\_PHY\_128) || (CONFIG\_PHY\_ID == APIMAC\_GENERIC\_CHINA\_LRM\_433\_PHY\_130))

#if (CCFG\_FORCE\_VDDR\_HH == 0)

#if (CONFIG\_TRANSMIT\_POWER >= 15)

#error "CONFIG\_TRANSMIT\_POWER should be less than 15"

#endif

#else

#if (CONFIG\_TRANSMIT\_POWER < 15)

/\* In 433 MHz band when CCFG\_FORCE\_VDDR\_HH = 1, only possible value of transmit power is 15 \*/

#error "CONFIG\_TRANSMIT\_POWER should be 15"

#endif

#endif

#else

#if (CCFG\_FORCE\_VDDR\_HH == 0)

#if (CONFIG\_TRANSMIT\_POWER >= 14)

#error "CONFIG\_TRANSMIT\_POWER should be less than 14"

#endif

#else

#if (CONFIG\_TRANSMIT\_POWER < 14)

/\* In US and ETSI band when CCFG\_FORCE\_VDDR\_HH = 1, only possible value of transmit power is 14 \*/

#error "CONFIG\_TRANSMIT\_POWER should be 14"

#endif

#endif

#endif

#endif

#else

#if (CCFG\_FORCE\_VDDR\_HH == 1)

#if (CONFIG\_TRANSMIT\_POWER != 14)

/\* In US and ETSI band when CCFG\_FORCE\_VDDR\_HH = 1, only possible value of transmit power is 14 \*/

#error "CONFIG\_TRANSMIT\_POWER should be 14"

#endif

#endif

#endif

/\*!

\* Enable this mode for certfication.

\* For FH certification, CONFIG\_FH\_ENABLE should

\* also be enabled

\*/

#define CERTIFICATION\_TEST\_MODE false

#ifdef POWER\_MEAS

/\*!

Power profile to be used when Power MEAS is enabled.

Profile 1 - POLL\_ACK - Polling Only

Profile 2 - DATA\_ACK - 20 byte application data + ACK from sensor to collector

Profile 3 - POLL\_DATA - Poll + received Data from collector

Profile 4 - SLEEP - No Poll or Data. In Beacon mode, beacon RX would occur

\*/

#define POWER\_TEST\_PROFILE DATA\_ACK

#endif

/\* Check if all the necessary parameters have been set for FH mode \*/

#if CONFIG\_FH\_ENABLE

#if !defined(FEATURE\_ALL\_MODES) && !defined(FEATURE\_FREQ\_HOP\_MODE)

#error "Do you want to build image with frequency hopping mode? \

Define either FEATURE\_FREQ\_HOP\_MODE or FEATURE\_ALL\_MODES in features.h"

#endif

#endif

/\* Check if stack level security is enabled if application security is enabled \*/

#if CONFIG\_SECURE

#if !defined(FEATURE\_MAC\_SECURITY)

#error "Define FEATURE\_MAC\_SECURITY or FEATURE\_ALL\_MODES in features.h to \

be able to use security at application level"

#endif

#endif

/\* Set beacon order and superframe order to 15 for FH mode to avoid user error \*/

#if CONFIG\_FH\_ENABLE

#if (CONFIG\_MAC\_BEACON\_ORDER != 15) && (CONFIG\_MAC\_SUPERFRAME\_ORDER != 15)

#error "Do you want to build image with frequency hopping mode? \

If yes, CONFIG\_MAC\_BEACON\_ORDER and CONFIG\_MAC\_SUPERFRAME\_ORDER \

should both be set to 15"

#endif

#if (FH\_NUM\_NON\_SLEEPY\_HOPPING\_NEIGHBORS < 2) || (FH\_NUM\_NON\_SLEEPY\_FIXED\_CHANNEL\_NEIGHBORS < 2)

#error "You have an invalid value for FH neighbors. Set the values \

for FH\_NUM\_NON\_SLEEPY\_HOPPING\_NEIGHBORS and FH\_NUM\_NON\_SLEEPY\_FIXED\_CHANNEL\_NEIGHBORS to at least 2"

#endif

#endif

#ifdef \_\_cplusplus

}

#endif

#endif /\* CONFIG\_H \*/

**SENSOR.C**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@file sensor.c

@brief TIMAC 2.0 Sensor Example Application

Group: WCS LPC

Target Device: cc13x0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Copyright (c) 2016-2019, Texas Instruments Incorporated

All rights reserved.

Redistribution and use in source and binary forms, with or without

modification, are permitted provided that the following conditions

are met:

\* Redistributions of source code must retain the above copyright

notice, this list of conditions and the following disclaimer.

\* Redistributions in binary form must reproduce the above copyright

notice, this list of conditions and the following disclaimer in the

documentation and/or other materials provided with the distribution.

\* Neither the name of Texas Instruments Incorporated nor the names of

its contributors may be used to endorse or promote products derived

from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"

AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,

THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR

PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR

CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,

EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,

PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;

OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,

WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR

OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,

EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Includes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <string.h>

#include <stdint.h>

#include "mac\_util.h"

#include "api\_mac.h"

#include "jdllc.h"

#include "ssf.h"

#include "smsgs.h"

#include "sensor.h"

#include "config.h"

#include "board\_led.h"

#include "board\_lcd.h"

#ifdef FEATURE\_NATIVE\_OAD

#include "oad\_client.h"

#endif /\* FEATURE\_NATIVE\_OAD \*/

#ifdef OSAL\_PORT2TIRTOS

#include <ti/sysbios/knl/Clock.h>

#else

#include "icall.h"

#endif

#ifdef FEATURE\_SECURE\_COMMISSIONING

#include "sm\_ti154.h"

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Constants and definitions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#if !defined(CONFIG\_AUTO\_START)

#if defined(AUTO\_START)

#define CONFIG\_AUTO\_START 1

#else

#define CONFIG\_AUTO\_START 0

#endif

#endif

/\* default MSDU Handle rollover \*/

#define MSDU\_HANDLE\_MAX 0x1F

/\* App marker in MSDU handle \*/

#define APP\_MARKER\_MSDU\_HANDLE 0x80

/\* App Message Tracking Mask \*/

#define APP\_MASK\_MSDU\_HANDLE 0x60

/\* App Sensor Data marker for the MSDU handle \*/

#define APP\_SENSOR\_MSDU\_HANDLE 0x40

/\* App tracking response marker for the MSDU handle \*/

#define APP\_TRACKRSP\_MSDU\_HANDLE 0x20

/\* App config response marker for the MSDU handle \*/

#define APP\_CONFIGRSP\_MSDU\_HANDLE 0x60

/\* Reporting Interval Min and Max (in milliseconds) \*/

#define MIN\_REPORTING\_INTERVAL 1000

#define MAX\_REPORTING\_INTERVAL 360000

/\* Polling Interval Min and Max (in milliseconds) \*/

//#define MIN\_POLLING\_INTERVAL 1000

#define MIN\_POLLING\_INTERVAL 250

#define MAX\_POLLING\_INTERVAL 10000

/\* Blink Time for Identify LED Request (in milliseconds) \*/

#define IDENTIFY\_LED\_TIME 1000

/\* Inter packet interval in certification test mode \*/

#if CERTIFICATION\_TEST\_MODE

#if ((CONFIG\_PHY\_ID >= APIMAC\_MRFSK\_STD\_PHY\_ID\_BEGIN) && (CONFIG\_PHY\_ID <= APIMAC\_MRFSK\_GENERIC\_PHY\_ID\_BEGIN))

/\*! Regular Mode \*/

#define SENSOR\_TEST\_RAMP\_DATA\_SIZE 75

#define CERT\_MODE\_INTER\_PKT\_INTERVAL 50

#elif ((CONFIG\_PHY\_ID >= APIMAC\_MRFSK\_GENERIC\_PHY\_ID\_BEGIN + 1) && (CONFIG\_PHY\_ID <= APIMAC\_MRFSK\_GENERIC\_PHY\_ID\_END))

/\*! LRM Mode \*/

#define SENSOR\_TEST\_RAMP\_DATA\_SIZE 20

#define CERT\_MODE\_INTER\_PKT\_INTERVAL 300

#else

#error "PHY ID is wrong."

#endif

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Global variables

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Task pending events \*/

uint16\_t Sensor\_events = 0;

/\*! Sensor statistics \*/

Smsgs\_msgStatsField\_t Sensor\_msgStats =

{ 0 };

extern bool initBroadcastMsg;

extern bool parentFound;

#ifdef POWER\_MEAS

/\*! Power Meas Stats fields \*/

Smsgs\_powerMeastatsField\_t Sensor\_pwrMeasStats =

{ 0 };

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Local variables

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void \*sem;

/\*! Rejoined flag \*/

static bool rejoining = false;

/\*! Collector's address \*/

static ApiMac\_sAddr\_t collectorAddr = {0};

/\* Join Time Ticks (used for average join time calculations) \*/

static uint\_fast32\_t joinTimeTicks = 0;

/\* End to end delay statistics timestamp \*/

static uint32\_t startSensorMsgTimeStamp = 0;

/\*! Device's Outgoing MSDU Handle values \*/

STATIC uint8\_t deviceTxMsduHandle = 0;

STATIC Smsgs\_configReqMsg\_t configSettings;

#if !defined(OAD\_IMG\_A)

/\*!

Temp Sensor field - valid only if Smsgs\_dataFields\_tempSensor

is set in frameControl.

\*/

Smsgs\_tempSensorField\_t tempSensor =

{ 0 };

/\*!

Light Sensor field - valid only if Smsgs\_dataFields\_lightSensor

is set in frameControl.

\*/

STATIC Smsgs\_lightSensorField\_t lightSensor =

{ 0 };

/\*!

Humidity Sensor field - valid only if Smsgs\_dataFields\_humiditySensor

is set in frameControl.

\*/

STATIC Smsgs\_humiditySensorField\_t humiditySensor =

{ 0 };

#endif //OAD\_IMG\_A

STATIC Llc\_netInfo\_t parentInfo = {0};

STATIC uint16\_t lastRcvdBroadcastMsgId = 0;

#ifdef FEATURE\_SECURE\_COMMISSIONING

/\* variable to store the current setting of auto Request Pib attribute

\* before it gets modified by SM module, in beacon mode

\*/

static bool currAutoReq = 0;

#endif /\* FEATURE\_SECURE\_COMMISSIONING \*/

#ifdef OAD\_IMG\_A

static bool Oad\_hasSentResetRsp = false;

#endif /\* OAD\_IMG\_A \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Local function prototypes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void initializeClocks(void);

static void dataCnfCB(ApiMac\_mcpsDataCnf\_t \*pDataCnf);

static void dataIndCB(ApiMac\_mcpsDataInd\_t \*pDataInd);

static uint8\_t getMsduHandle(Smsgs\_cmdIds\_t msgType);

#if !defined(OAD\_IMG\_A)

static void processSensorMsgEvt(void);

static bool sendSensorMessage(ApiMac\_sAddr\_t \*pDstAddr,

Smsgs\_sensorMsg\_t \*pMsg);

static void readSensors(void);

#endif //OAD\_IMG\_A

#if SENSOR\_TEST\_RAMP\_DATA\_SIZE

static void processSensorRampMsgEvt(void);

#endif

static void processConfigRequest(ApiMac\_mcpsDataInd\_t \*pDataInd);

static void processBroadcastCtrlMsg(ApiMac\_mcpsDataInd\_t \*pDataInd);

static bool sendConfigRsp(ApiMac\_sAddr\_t \*pDstAddr, Smsgs\_configRspMsg\_t \*pMsg);

static uint16\_t validateFrameControl(uint16\_t frameControl);

static void jdllcJoinedCb(ApiMac\_deviceDescriptor\_t \*pDevInfo,

Llc\_netInfo\_t \*pStartedInfo);

static void jdllcDisassocIndCb(ApiMac\_sAddrExt\_t \*extAddress,

ApiMac\_disassocateReason\_t reason);

static void jdllcDisassocCnfCb(ApiMac\_sAddrExt\_t \*extAddress,

ApiMac\_status\_t status);

static void jdllcStateChangeCb(Jdllc\_states\_t state);

#ifdef FEATURE\_SECURE\_COMMISSIONING

/\* Security Manager callback functions \*/

static void smFailCMProcessCb(ApiMac\_deviceDescriptor\_t \*devInfo,

bool rxOnIdle, bool keyRefreshment);

static void smSuccessCMProcessCb(ApiMac\_deviceDescriptor\_t \*devInfo,

bool keyRefreshment);

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Callback tables

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*! API MAC Callback table \*/

STATIC ApiMac\_callbacks\_t Sensor\_macCallbacks =

{

/\*! Associate Indicated callback \*/

NULL,

/\*! Associate Confirmation callback \*/

NULL,

/\*! Disassociate Indication callback \*/

NULL,

/\*! Disassociate Confirmation callback \*/

NULL,

/\*! Beacon Notify Indication callback \*/

NULL,

/\*! Orphan Indication callback \*/

NULL,

/\*! Scan Confirmation callback \*/

NULL,

/\*! Start Confirmation callback \*/

NULL,

/\*! Sync Loss Indication callback \*/

NULL,

/\*! Poll Confirm callback \*/

NULL,

/\*! Comm Status Indication callback \*/

NULL,

/\*! Poll Indication Callback \*/

NULL,

/\*! Data Confirmation callback \*/

dataCnfCB,

/\*! Data Indication callback \*/

dataIndCB,

/\*! Purge Confirm callback \*/

NULL,

/\*! WiSUN Async Indication callback \*/

NULL,

/\*! WiSUN Async Confirmation callback \*/

NULL,

/\*! Unprocessed message callback \*/

NULL

};

STATIC Jdllc\_callbacks\_t jdllcCallbacks =

{

/\*! Network Joined Indication callback \*/

jdllcJoinedCb,

/\* Disassociation Indication callback \*/

jdllcDisassocIndCb,

/\* Disassociation Confirm callback \*/

jdllcDisassocCnfCb,

/\*! State Changed indication callback \*/

jdllcStateChangeCb

};

#ifdef FEATURE\_SECURE\_COMMISSIONING

STATIC SM\_callbacks\_t SMCallbacks =

{

/\*! Security authentication failed callback \*/

smFailCMProcessCb,

/\* Security authentication successful callback \*/

smSuccessCMProcessCb

};

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Public Functions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

Initialize this application.

Public function defined in sensor.h

\*/

#ifdef OSAL\_PORT2TIRTOS

void Sensor\_init(uint8\_t macTaskId)

#else

void Sensor\_init(void)

#endif

{

uint32\_t frameCounter = 0;

/\* Initialize the sensor's structures \*/

memset(&configSettings, 0, sizeof(Smsgs\_configReqMsg\_t));

#if defined(TEMP\_SENSOR)

configSettings.frameControl |= Smsgs\_dataFields\_tempSensor;

#endif

#if defined(LIGHT\_SENSOR)

configSettings.frameControl |= Smsgs\_dataFields\_lightSensor;

#endif

#if defined(HUMIDITY\_SENSOR)

configSettings.frameControl |= Smsgs\_dataFields\_humiditySensor;

#endif

configSettings.frameControl |= Smsgs\_dataFields\_msgStats;

configSettings.frameControl |= Smsgs\_dataFields\_configSettings;

if(!CERTIFICATION\_TEST\_MODE)

{

configSettings.reportingInterval = CONFIG\_REPORTING\_INTERVAL;

}

else

{

/\* start back to back data transmission at the earliest \*/

configSettings.reportingInterval = 100;

}

configSettings.pollingInterval = CONFIG\_POLLING\_INTERVAL;

/\* Initialize the MAC \*/

#ifdef OSAL\_PORT2TIRTOS

sem = ApiMac\_init(macTaskId, CONFIG\_FH\_ENABLE);

#else

sem = ApiMac\_init(CONFIG\_FH\_ENABLE);

#endif

/\* Initialize the Joining Device Logical Link Controller \*/

Jdllc\_init(&Sensor\_macCallbacks, &jdllcCallbacks);

/\* Register the MAC Callbacks \*/

ApiMac\_registerCallbacks(&Sensor\_macCallbacks);

/\* Initialize the platform specific functions \*/

Ssf\_init(sem);

#ifdef FEATURE\_SECURE\_COMMISSIONING

/\* Intialize the security manager and register callbacks \*/

SM\_init();

SM\_registerCallback(&SMCallbacks);

#endif /\* FEATURE\_SECURE\_COMMISSIONING \*/

ApiMac\_mlmeSetReqUint8(ApiMac\_attribute\_phyCurrentDescriptorId,

(uint8\_t)CONFIG\_PHY\_ID);

ApiMac\_mlmeSetReqUint8(ApiMac\_attribute\_channelPage,

(uint8\_t)CONFIG\_CHANNEL\_PAGE);

Ssf\_getFrameCounter(NULL, &frameCounter);

#ifdef FEATURE\_MAC\_SECURITY

/\* Initialize the MAC Security \*/

Jdllc\_securityInit(frameCounter);

#endif /\* FEATURE\_MAC\_SECURITY \*/

/\* Set the transmit power \*/

ApiMac\_mlmeSetReqUint8(ApiMac\_attribute\_phyTransmitPowerSigned,

(uint8\_t)CONFIG\_TRANSMIT\_POWER);

/\* Set Min BE \*/

ApiMac\_mlmeSetReqUint8(ApiMac\_attribute\_backoffExponent,

(uint8\_t)CONFIG\_MIN\_BE);

/\* Set Max BE \*/

ApiMac\_mlmeSetReqUint8(ApiMac\_attribute\_maxBackoffExponent,

(uint8\_t)CONFIG\_MAX\_BE);

/\* Set MAC MAX CSMA Backoffs \*/

ApiMac\_mlmeSetReqUint8(ApiMac\_attribute\_maxCsmaBackoffs,

(uint8\_t)CONFIG\_MAC\_MAX\_CSMA\_BACKOFFS);

/\* Set MAC MAX Frame Retries \*/

ApiMac\_mlmeSetReqUint8(ApiMac\_attribute\_maxFrameRetries,

(uint8\_t)CONFIG\_MAX\_RETRIES);

#ifdef FCS\_TYPE16

/\* Set the fcs type \*/

ApiMac\_mlmeSetReqBool(ApiMac\_attribute\_fcsType,

(bool)1);

#endif

/\* Initialize the app clocks \*/

initializeClocks();

if(CONFIG\_AUTO\_START)

{

/\* Start the device \*/

Util\_setEvent(&Sensor\_events, SENSOR\_START\_EVT);

}

}

/\*!

Application task processing.

Public function defined in sensor.h

\*/

void Sensor\_process(void)

{

/\* Start the collector device in the network \*/

if(Sensor\_events & SENSOR\_START\_EVT)

{

ApiMac\_deviceDescriptor\_t devInfo;

Llc\_netInfo\_t parentInfo;

if(Ssf\_getNetworkInfo(&devInfo, &parentInfo ) == true)

{

Ssf\_configSettings\_t configInfo;

#ifdef FEATURE\_MAC\_SECURITY

ApiMac\_status\_t stat;

#endif /\* FEATURE\_MAC\_SECURITY \*/

/\* Do we have config settings? \*/

if(Ssf\_getConfigInfo(&configInfo) == true)

{

/\* Save the config information \*/

configSettings.frameControl = configInfo.frameControl;

configSettings.reportingInterval = configInfo.reportingInterval;

configSettings.pollingInterval = configInfo.pollingInterval;

/\* Update the polling interval in the LLC \*/

Jdllc\_setPollRate(configSettings.pollingInterval);

}

/\* Initially, setup the parent as the collector \*/

if(parentInfo.fh == true && CONFIG\_RX\_ON\_IDLE)

{

collectorAddr.addrMode = ApiMac\_addrType\_extended;

memcpy(&collectorAddr.addr.extAddr,

parentInfo.devInfo.extAddress, APIMAC\_SADDR\_EXT\_LEN);

}

else

{

collectorAddr.addrMode = ApiMac\_addrType\_short;

collectorAddr.addr.shortAddr = parentInfo.devInfo.shortAddress;

}

#ifdef FEATURE\_MAC\_SECURITY

/\* Put the parent in the security device list \*/

stat = Jdllc\_addSecDevice(parentInfo.devInfo.panID,

parentInfo.devInfo.shortAddress,

&parentInfo.devInfo.extAddress, 0);

if(stat != ApiMac\_status\_success)

{

Ssf\_displayError("Auth Error: 0x", (uint8\_t)stat);

}

#endif /\* FEATURE\_MAC\_SECURITY \*/

#ifdef FEATURE\_SECURE\_COMMISSIONING

if(!CONFIG\_FH\_ENABLE)

{

nvDeviceKeyInfo\_t devKeyInfo;

SM\_seedKey\_Entry\_t \* pSeedKeyEnty;

if(Ssf\_getDeviceKeyInfo(&devKeyInfo) == TRUE)

{

/\* Update the seedKeyTable and MAC Key Table \*/

/\* Use its own ext address \*/

updateSeedKeyFromNV(&devInfo,&devKeyInfo);

pSeedKeyEnty = getEntryFromSeedKeyTable(devInfo.extAddress,devInfo.shortAddress);

/\* Do not change the order below to lines \*/

/\* Copy collector ext Address first \*/

memcpy(commissionDevInfo.extAddress, parentInfo.devInfo.extAddress, sizeof(ApiMac\_sAddrExt\_t));

addDeviceKey(pSeedKeyEnty,devKeyInfo.deviceKey, true);

LCD\_WRITE\_STRING("KeyInfo recovered", 6);

}

}

#endif

Jdllc\_rejoin(&devInfo, &parentInfo);

rejoining = true;

}

else

{

/\* Get Start Timestamp \*/

#ifdef OSAL\_PORT2TIRTOS

joinTimeTicks = Clock\_getTicks();

#else

joinTimeTicks = ICall\_getTicks();

#endif

Jdllc\_join();

}

/\* Clear the event \*/

Util\_clearEvent(&Sensor\_events, SENSOR\_START\_EVT);

}

if(Sensor\_events & EXT\_SENSOR\_READING\_TIMEOUT\_EVT)

{

/\* Process Sensor ReadingMessage Event \*/

processSensorMsgEvt();

/\* Clear the event \*/

Util\_clearEvent(&Sensor\_events, EXT\_SENSOR\_READING\_TIMEOUT\_EVT);

}

/\* Is it time to send the next sensor data message? \*/

if(Sensor\_events & SENSOR\_READING\_TIMEOUT\_EVT)

{

#if !defined(OAD\_IMG\_A)

/\* In certification test mode, back to back data shall be sent \*/

if(!CERTIFICATION\_TEST\_MODE)

{

/\* Setup for the next message \*/

Ssf\_setReadingClock(configSettings.reportingInterval);

}

#ifdef FEATURE\_SECURE\_COMMISSIONING

/\* if secure Commissioning feature is enabled, read

\* sensor data and send it only after the secure

\* commissioning process is done successfully.

\* else, do not read and send sensor data.

\*/

if(SM\_Last\_State != SM\_CM\_InProgress)

{

#endif //FEATURE\_SECURE\_COMMISSIONING

#if SENSOR\_TEST\_RAMP\_DATA\_SIZE

processSensorRampMsgEvt();

#else

/\* Read sensors \*/

readSensors();

/\* Process Sensor Reading Message Event \*/

processSensorMsgEvt();

#endif //SENSOR\_TEST\_RAMP\_DATA\_SIZE

#ifdef FEATURE\_SECURE\_COMMISSIONING

}

#endif //FEATURE\_SECURE\_COMMISSIONING

#endif //OAD\_IMG\_A

/\* Clear the event \*/

Util\_clearEvent(&Sensor\_events, SENSOR\_READING\_TIMEOUT\_EVT);

}

#if defined(OAD\_IMG\_A)

if(Sensor\_events & SENSOR\_OAD\_SEND\_RESET\_RSP\_EVT )

{

/\* send OAD reset response \*/

if( false == Oad\_hasSentResetRsp)

{

OADProtocol\_Status\_t status;

status = OADProtocol\_sendOadResetRsp(&collectorAddr);

if(OADProtocol\_Status\_Success == status)

{

//notify to user

LCD\_WRITE\_STRING("Sent Reset Response", 6);

Oad\_hasSentResetRsp = true;

}

else

{

LCD\_WRITE\_STRING(" Failed to send Reset Response", 6);

}

}

/\* Clear the event \*/

Util\_clearEvent(&Sensor\_events, SENSOR\_OAD\_SEND\_RESET\_RSP\_EVT);

}

#endif //OAD\_IMG\_A

#ifdef DISPLAY\_PER\_STATS

if(Sensor\_events & SENSOR\_UPDATE\_STATS\_EVT)

{

Ssf\_displayPerStats(&Sensor\_msgStats);

/\* Clear the event \*/

Util\_clearEvent(&Sensor\_events, SENSOR\_UPDATE\_STATS\_EVT);

}

#endif /\* DISPLAY\_PER\_STATS \*/

/\* Process LLC Events \*/

Jdllc\_process();

/\* Allow the Specific functions to process \*/

Ssf\_processEvents();

#ifdef FEATURE\_SECURE\_COMMISSIONING

/\* Allow the security manager specific functions to process \*/

SM\_process();

#endif /\* FEATURE\_SECURE\_COMMISSIONING \*/

/\*

Don't process ApiMac messages until all of the sensor events

are processed.

\*/

#ifdef FEATURE\_SECURE\_COMMISSIONING

/\*only if there are no sensor events and security manager events to handle\*/

if((Sensor\_events == 0) && (SM\_events == 0))

#else

if(Sensor\_events == 0)

#endif

{

/\* Wait for response message or events \*/

ApiMac\_processIncoming();

}

}

/\*!

\* @brief Send MAC data request

\*

\* @param type - message type

\* @param pDstAddr - destination address

\* @param rxOnIdle - true if not a sleepy device

\* @param len - length of payload

\* @param pData - pointer to the buffer

\*

\* @return true if sent, false if not

\*/

bool Sensor\_sendMsg(Smsgs\_cmdIds\_t type, ApiMac\_sAddr\_t \*pDstAddr,

bool rxOnIdle, uint16\_t len, uint8\_t \*pData)

{

bool ret = false;

ApiMac\_mcpsDataReq\_t dataReq;

/\* Timestamp to compute end to end delay \*/

#ifdef OSAL\_PORT2TIRTOS

startSensorMsgTimeStamp = Clock\_getTicks();

#else

startSensorMsgTimeStamp = ICall\_getTicks();

#endif

/\* Fill the data request field \*/

memset(&dataReq, 0, sizeof(ApiMac\_mcpsDataReq\_t));

memcpy(&dataReq.dstAddr, pDstAddr, sizeof(ApiMac\_sAddr\_t));

if(pDstAddr->addrMode == ApiMac\_addrType\_extended)

{

dataReq.srcAddrMode = ApiMac\_addrType\_extended;

}

else

{

dataReq.srcAddrMode = ApiMac\_addrType\_short;

}

if(rejoining == true)

{

ApiMac\_mlmeGetReqUint16(ApiMac\_attribute\_panId,

&(parentInfo.devInfo.panID));

}

dataReq.dstPanId = parentInfo.devInfo.panID;

dataReq.msduHandle = getMsduHandle(type);

dataReq.txOptions.ack = true;

if(CERTIFICATION\_TEST\_MODE)

{

dataReq.txOptions.ack = false;

}

if(rxOnIdle == false)

{

dataReq.txOptions.indirect = true;

}

dataReq.msdu.len = len;

dataReq.msdu.p = pData;

#ifdef FEATURE\_MAC\_SECURITY

#ifdef FEATURE\_SECURE\_COMMISSIONING

{

extern ApiMac\_sAddrExt\_t ApiMac\_extAddr;

SM\_getSrcDeviceSecurityInfo(ApiMac\_extAddr, SM\_Sensor\_SAddress, &dataReq.sec);

}

#else

Jdllc\_securityFill(&dataReq.sec);

#endif /\* FEATURE\_SECURE\_COMMISSIONING \*/

#endif /\* FEATURE\_MAC\_SECURITY \*/

if(type == Smsgs\_cmdIds\_sensorData || type == Smsgs\_cmdIds\_rampdata)

{

Sensor\_msgStats.msgsAttempted++;

}

else if(type == Smsgs\_cmdIds\_trackingRsp)

{

Sensor\_msgStats.trackingResponseAttempts++;

}

else if(type == Smsgs\_cmdIds\_configRsp)

{

Sensor\_msgStats.configResponseAttempts++;

}

/\* Send the message \*/

if(ApiMac\_mcpsDataReq(&dataReq) == ApiMac\_status\_success)

{

ret = true;

}

else

{

/\* handle transaction overflow by retrying \*/

if(type == Smsgs\_cmdIds\_sensorData || type == Smsgs\_cmdIds\_rampdata)

{

Ssf\_setReadingClock(configSettings.reportingInterval);

Sensor\_msgStats.msgsAttempted++;

}

}

return (ret);

}

#ifdef IDENTIFY\_LED

/\*!

Send LED Identify Request to collector

Public function defined in sensor.h

\*/

void Sensor\_sendIdentifyLedRequest(void)

{

uint8\_t cmdBytes[SMSGS\_INDENTIFY\_LED\_REQUEST\_MSG\_LEN];

/\* send the response message directly \*/

cmdBytes[0] = (uint8\_t) Smsgs\_cmdIds\_IdentifyLedReq;

cmdBytes[1] = (uint8\_t) IDENTIFY\_LED\_TIME;

Sensor\_sendMsg(Smsgs\_cmdIds\_IdentifyLedReq,

&collectorAddr, true,

SMSGS\_INDENTIFY\_LED\_REQUEST\_MSG\_LEN,

cmdBytes);

}

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Local Functions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*!

\* @brief Initialize the clocks.

\*/

static void initializeClocks(void)

{

/\* Initialize the reading clock \*/

Ssf\_initializeReadingClock();

}

/\*!

\* @brief MAC Data Confirm callback.

\*

\* @param pDataCnf - pointer to the data confirm information

\*/

static void dataCnfCB(ApiMac\_mcpsDataCnf\_t \*pDataCnf)

{

uint16\_t endToEndDelay = 0;

/\* Record statistics \*/

if(pDataCnf->status == ApiMac\_status\_channelAccessFailure)

{

Sensor\_msgStats.channelAccessFailures++;

}

else if(pDataCnf->status == ApiMac\_status\_noAck)

{

Sensor\_msgStats.macAckFailures++;

#ifdef DISPLAY\_PER\_STATS

Util\_setEvent(&Sensor\_events, SENSOR\_UPDATE\_STATS\_EVT);

#endif

}

else if(pDataCnf->status != ApiMac\_status\_success)

{

Sensor\_msgStats.otherDataRequestFailures++;

#ifdef DISPLAY\_PER\_STATS

Util\_setEvent(&Sensor\_events, SENSOR\_UPDATE\_STATS\_EVT);

#endif

Ssf\_displayError("dataCnf: ", pDataCnf->status);

}

else if(pDataCnf->status == ApiMac\_status\_success)

{

Ssf\_updateFrameCounter(NULL, pDataCnf->frameCntr);

}

/\* Make sure the message came from the app \*/

if(pDataCnf->msduHandle & APP\_MARKER\_MSDU\_HANDLE)

{

/\* What message type was the original request? \*/

if((pDataCnf->msduHandle & APP\_MASK\_MSDU\_HANDLE)

== APP\_SENSOR\_MSDU\_HANDLE)

{

if(pDataCnf->status == ApiMac\_status\_success)

{

Sensor\_msgStats.msgsSent++;

#ifdef DISPLAY\_PER\_STATS

Util\_setEvent(&Sensor\_events, SENSOR\_UPDATE\_STATS\_EVT);

#endif

/\* Calculate end to end delay \*/

#ifdef OSAL\_PORT2TIRTOS

if(Clock\_getTicks() < startSensorMsgTimeStamp)

{

endToEndDelay = Clock\_getTicks() +

(0xFFFFFFFF-startSensorMsgTimeStamp);

}

else

{

endToEndDelay = Clock\_getTicks() - startSensorMsgTimeStamp;

}

#else

if(ICall\_getTicks() < startSensorMsgTimeStamp)

{

endToEndDelay = ICall\_getTicks() +

(0xFFFFFFFF-startSensorMsgTimeStamp);

}

else

{

endToEndDelay = ICall\_getTicks() - startSensorMsgTimeStamp;

}

#endif

endToEndDelay = endToEndDelay/TICKPERIOD\_MS\_US;

Sensor\_msgStats.worstCaseE2EDelay =

(Sensor\_msgStats.worstCaseE2EDelay > endToEndDelay) ?

Sensor\_msgStats.worstCaseE2EDelay:endToEndDelay;

Sensor\_msgStats.avgE2EDelay =

(((uint32\_t)Sensor\_msgStats.avgE2EDelay \*

(Sensor\_msgStats.msgsSent - 1)) + endToEndDelay)/

Sensor\_msgStats.msgsSent;

}

#if CERTIFICATION\_TEST\_MODE

{

/\* Setup for the next message \*/

Ssf\_setReadingClock(CERT\_MODE\_INTER\_PKT\_INTERVAL);

}

#endif

}

if((pDataCnf->msduHandle & APP\_MASK\_MSDU\_HANDLE)

== APP\_TRACKRSP\_MSDU\_HANDLE)

{

if(pDataCnf->status == ApiMac\_status\_success)

{

Sensor\_msgStats.trackingResponseSent++;

}

}

if((pDataCnf->msduHandle & APP\_MASK\_MSDU\_HANDLE)

== APP\_CONFIGRSP\_MSDU\_HANDLE)

{

if(pDataCnf->status == ApiMac\_status\_success)

{

Sensor\_msgStats.configResponseSent++;

}

}

}

}

/\*!

\* @brief MAC Data Indication callback.

\*

\* @param pDataInd - pointer to the data indication information

\*/

static void dataIndCB(ApiMac\_mcpsDataInd\_t \*pDataInd)

{

uint8\_t cmdBytes[SMSGS\_TOGGLE\_LED\_RESPONSE\_MSG\_LEN];

if((pDataInd != NULL) && (pDataInd->msdu.p != NULL)

&& (pDataInd->msdu.len > 0))

{

Smsgs\_cmdIds\_t cmdId = (Smsgs\_cmdIds\_t)\*(pDataInd->msdu.p);

#ifdef FEATURE\_MAC\_SECURITY

{

if(Jdllc\_securityCheck(&(pDataInd->sec)) == false)

{

/\* reject the message \*/

return;

}

}

#endif /\* FEATURE\_MAC\_SECURITY \*/

switch(cmdId)

{

case Smsgs\_cmdIds\_configReq:

processConfigRequest(pDataInd);

Sensor\_msgStats.configRequests++;

break;

case Smsgs\_cmdIds\_trackingReq:

/\* Make sure the message is the correct size \*/

if(pDataInd->msdu.len == SMSGS\_TRACKING\_REQUEST\_MSG\_LENGTH)

{

/\* Update stats \*/

Sensor\_msgStats.trackingRequests++;

/\* Indicate tracking message received \*/

Ssf\_trackingUpdate(&pDataInd->srcAddr);

/\* send the response message directly \*/

cmdBytes[0] = (uint8\_t) Smsgs\_cmdIds\_trackingRsp;

Sensor\_sendMsg(Smsgs\_cmdIds\_trackingRsp,

&pDataInd->srcAddr, true,

1, cmdBytes);

}

break;

case Smsgs\_cmdIds\_toggleLedReq:

/\* Make sure the message is the correct size \*/

if(pDataInd->msdu.len == SMSGS\_TOGGLE\_LED\_REQUEST\_MSG\_LEN)

{

/\* send the response message directly \*/

cmdBytes[0] = (uint8\_t) Smsgs\_cmdIds\_toggleLedRsp;

cmdBytes[1] = Ssf\_toggleLED();

Sensor\_sendMsg(Smsgs\_cmdIds\_toggleLedRsp,

&pDataInd->srcAddr, true,

SMSGS\_TOGGLE\_LED\_RESPONSE\_MSG\_LEN,

cmdBytes);

}

break;

case Smgs\_cmdIds\_broadcastCtrlMsg:

if(parentFound)

{

/\* Node has successfully associated with the network \*/

processBroadcastCtrlMsg(pDataInd);

}

break;

#ifdef POWER\_MEAS

case Smsgs\_cmdIds\_rampdata:

Sensor\_pwrMeasStats.rampDataRcvd++;

break;

#endif

#ifdef FEATURE\_NATIVE\_OAD

case Smsgs\_cmdIds\_oad:

//Index past the Smsgs\_cmdId

OADProtocol\_ParseIncoming((void\*) &(pDataInd->srcAddr), pDataInd->msdu.p + 1);

break;

#endif //FEATURE\_NATIVE\_OAD

#ifdef FEATURE\_SECURE\_COMMISSIONING

case Smgs\_cmdIds\_CommissionStart:

{

ApiMac\_sec\_t devSec;

extern ApiMac\_sAddrExt\_t ApiMac\_extAddr;

/\* Obtain MAC level security information. Use network key for SM \*/

Jdllc\_securityFill(&devSec);

uint8\_t \*pBuf = pDataInd->msdu.p;

pBuf += sizeof(Smsgs\_cmdIds\_t);

SMMsgs\_cmdIds\_t CMMsgId = (SMMsgs\_cmdIds\_t)Util\_buildUint16(pBuf[0], pBuf[1]);

/\* read the current value \*/

ApiMac\_mlmeGetReqBool(ApiMac\_attribute\_autoRequest, &currAutoReq);

/\* beacon-mode of operation and autoRequest is set to true \*/

if((CONFIG\_MAC\_BEACON\_ORDER != 15) && (currAutoReq == true))

{

/\* if false enable explicit polling \*/

ApiMac\_mlmeSetReqBool(ApiMac\_attribute\_autoRequest, false);

Util\_setEvent(&Jdllc\_events, JDLLC\_POLL\_EVT);

}

if ((SM\_Last\_State != SM\_CM\_InProgress) &&

(CMMsgId == SMMsgs\_cmdIds\_KeyRefreshRequest))

{

/\* Kick off key refreshment process after successful commissioning \*/

SM\_startKeyRefreshProcess(&parentInfo.devInfo, &devSec,

parentInfo.fh, true);

}

else

{

/\* Kick off commissioning process to obtain security information \*/

SM\_startCMProcess(&parentInfo.devInfo, &devSec, parentInfo.fh,

true, SM\_type\_device, SM\_SENSOR\_AUTH\_METHOD);

}

}

break;

case Smgs\_cmdIds\_CommissionMsg:

{

/\* Process Security manager commissioning data \*/

SM\_processCommData(pDataInd);

}

break;

#endif /\* FEATURE\_SECURE\_COMMISSIONING \*/

default:

/\* Should not receive other messages \*/

break;

}

}

}

/\*!

\* @brief Get the next MSDU Handle

\* <BR>

\* The MSDU handle has 3 parts:<BR>

\* - The MSBit(7), when set means the the application sent the

\* message

\* - Bit 6, when set means that the app message is a config request

\* - Bits 0-5, used as a message counter that rolls over.

\*

\* @param msgType - message command id needed

\*

\* @return msdu Handle

\*/

static uint8\_t getMsduHandle(Smsgs\_cmdIds\_t msgType)

{

uint8\_t msduHandle = deviceTxMsduHandle;

/\* Increment for the next msdu handle, or roll over \*/

if(deviceTxMsduHandle >= MSDU\_HANDLE\_MAX)

{

deviceTxMsduHandle = 0;

}

else

{

deviceTxMsduHandle++;

}

/\* Add the App specific bit \*/

msduHandle |= APP\_MARKER\_MSDU\_HANDLE;

/\* Add the message type bit \*/

if(msgType == Smsgs\_cmdIds\_sensorData || msgType == Smsgs\_cmdIds\_rampdata)

{

msduHandle |= APP\_SENSOR\_MSDU\_HANDLE;

}

else if(msgType == Smsgs\_cmdIds\_trackingRsp)

{

msduHandle |= APP\_TRACKRSP\_MSDU\_HANDLE;

}

else if(msgType == Smsgs\_cmdIds\_configRsp)

{

msduHandle |= APP\_CONFIGRSP\_MSDU\_HANDLE;

}

return (msduHandle);

}

/\*!

@brief Build and send fixed size ramp data

\*/

#if SENSOR\_TEST\_RAMP\_DATA\_SIZE

static void processSensorRampMsgEvt(void)

{

uint8\_t \*pMsgBuf;

uint16\_t index;

pMsgBuf = (uint8\_t \*)Ssf\_malloc(SENSOR\_TEST\_RAMP\_DATA\_SIZE);

if(pMsgBuf)

{

uint8\_t \*pBuf = pMsgBuf;

\*pBuf++ = (uint8\_t)Smsgs\_cmdIds\_rampdata;

for(index = 1; index < SENSOR\_TEST\_RAMP\_DATA\_SIZE; index++)

{

\*pBuf++ = (uint8\_t) (index & 0xFF);

}

#ifndef POWER\_MEAS

Board\_Led\_toggle(board\_led\_type\_LED2);

#endif

Sensor\_sendMsg(Smsgs\_cmdIds\_rampdata, &collectorAddr, true,

SENSOR\_TEST\_RAMP\_DATA\_SIZE, pMsgBuf);

Ssf\_free(pMsgBuf);

}

}

#endif

#if !defined(OAD\_IMG\_A)

/\*!

@brief Build and send sensor data message

\*/

static void processSensorMsgEvt(void)

{

Smsgs\_sensorMsg\_t sensor;

uint32\_t stat;

memset(&sensor, 0, sizeof(Smsgs\_sensorMsg\_t));

ApiMac\_mlmeGetReqUint32(ApiMac\_attribute\_diagRxSecureFail, &stat);

Sensor\_msgStats.rxDecryptFailures = (uint16\_t)stat;

ApiMac\_mlmeGetReqUint32(ApiMac\_attribute\_diagTxSecureFail, &stat);

Sensor\_msgStats.txEncryptFailures = (uint16\_t)stat;

ApiMac\_mlmeGetReqArray(ApiMac\_attribute\_extendedAddress,

sensor.extAddress);

/\* fill in the message \*/

sensor.frameControl = configSettings.frameControl;

if(sensor.frameControl & Smsgs\_dataFields\_tempSensor)

{

memcpy(&sensor.tempSensor, &tempSensor,

sizeof(Smsgs\_tempSensorField\_t));

}

if(sensor.frameControl & Smsgs\_dataFields\_lightSensor)

{

memcpy(&sensor.lightSensor, &lightSensor,

sizeof(Smsgs\_lightSensorField\_t));

}

if(sensor.frameControl & Smsgs\_dataFields\_humiditySensor)

{

memcpy(&sensor.humiditySensor, &humiditySensor,

sizeof(Smsgs\_humiditySensorField\_t));

}

if(sensor.frameControl & Smsgs\_dataFields\_msgStats)

{

memcpy(&sensor.msgStats, &Sensor\_msgStats,

sizeof(Smsgs\_msgStatsField\_t));

}

if(sensor.frameControl & Smsgs\_dataFields\_configSettings)

{

sensor.configSettings.pollingInterval = configSettings.pollingInterval;

sensor.configSettings.reportingInterval = configSettings

.reportingInterval;

}

/\* inform the user interface \*/

Ssf\_sensorReadingUpdate(&sensor);

/\* send the data to the collector \*/

sendSensorMessage(&collectorAddr, &sensor);

}

/\*!

\* @brief Manually read the sensors

\*/

static void readSensors(void)

{

#if defined(TEMP\_SENSOR)

/\* Read the temp sensor values \*/

tempSensor.ambienceTemp = Ssf\_readTempSensor();

tempSensor.objectTemp = tempSensor.ambienceTemp;

#endif

}

/\*!

\* @brief Build and send sensor data message

\*

\* @param pDstAddr - Where to send the message

\* @param pMsg - pointer to the sensor data

\*

\* @return true if message was sent, false if not

\*/

static bool sendSensorMessage(ApiMac\_sAddr\_t \*pDstAddr, Smsgs\_sensorMsg\_t \*pMsg)

{

bool ret = false;

uint8\_t \*pMsgBuf;

uint16\_t len = SMSGS\_BASIC\_SENSOR\_LEN;

/\* Figure out the length \*/

if(pMsg->frameControl & Smsgs\_dataFields\_tempSensor)

{

len += SMSGS\_SENSOR\_TEMP\_LEN;

}

if(pMsg->frameControl & Smsgs\_dataFields\_lightSensor)

{

len += SMSGS\_SENSOR\_LIGHT\_LEN;

}

if(pMsg->frameControl & Smsgs\_dataFields\_humiditySensor)

{

len += SMSGS\_SENSOR\_HUMIDITY\_LEN;

}

if(pMsg->frameControl & Smsgs\_dataFields\_msgStats)

{

//len += SMSGS\_SENSOR\_MSG\_STATS\_LEN;

len += sizeof(Smsgs\_msgStatsField\_t);

}

if(pMsg->frameControl & Smsgs\_dataFields\_configSettings)

{

len += SMSGS\_SENSOR\_CONFIG\_SETTINGS\_LEN;

}

pMsgBuf = (uint8\_t \*)Ssf\_malloc(len);

if(pMsgBuf)

{

uint8\_t \*pBuf = pMsgBuf;

\*pBuf++ = (uint8\_t)Smsgs\_cmdIds\_sensorData;

memcpy(pBuf, pMsg->extAddress, SMGS\_SENSOR\_EXTADDR\_LEN);

pBuf += SMGS\_SENSOR\_EXTADDR\_LEN;

pBuf = Util\_bufferUint16(pBuf,pMsg->frameControl);

/\* Buffer data in order of frameControl mask, starting with LSB \*/

if(pMsg->frameControl & Smsgs\_dataFields\_tempSensor)

{

pBuf = Util\_bufferUint16(pBuf, pMsg->tempSensor.ambienceTemp);

pBuf = Util\_bufferUint16(pBuf, pMsg->tempSensor.objectTemp);

}

if(pMsg->frameControl & Smsgs\_dataFields\_lightSensor)

{

pBuf = Util\_bufferUint16(pBuf, pMsg->lightSensor.rawData);

}

if(pMsg->frameControl & Smsgs\_dataFields\_humiditySensor)

{

pBuf = Util\_bufferUint16(pBuf, pMsg->humiditySensor.temp);

pBuf = Util\_bufferUint16(pBuf, pMsg->humiditySensor.humidity);

}

if(pMsg->frameControl & Smsgs\_dataFields\_msgStats)

{

pBuf = Util\_bufferUint16(pBuf, pMsg->msgStats.joinAttempts);

pBuf = Util\_bufferUint16(pBuf, pMsg->msgStats.joinFails);

pBuf = Util\_bufferUint16(pBuf, pMsg->msgStats.msgsAttempted);

pBuf = Util\_bufferUint16(pBuf, pMsg->msgStats.msgsSent);

pBuf = Util\_bufferUint16(pBuf, pMsg->msgStats.trackingRequests);

pBuf = Util\_bufferUint16(pBuf,

pMsg->msgStats.trackingResponseAttempts);

pBuf = Util\_bufferUint16(pBuf,

pMsg->msgStats.trackingResponseSent);

pBuf = Util\_bufferUint16(pBuf, pMsg->msgStats.configRequests);

pBuf = Util\_bufferUint16(pBuf,

pMsg->msgStats.configResponseAttempts);

pBuf = Util\_bufferUint16(pBuf,

pMsg->msgStats.configResponseSent);

pBuf = Util\_bufferUint16(pBuf,

pMsg->msgStats.channelAccessFailures);

pBuf = Util\_bufferUint16(pBuf, pMsg->msgStats.macAckFailures);

pBuf = Util\_bufferUint16(pBuf,

pMsg->msgStats.otherDataRequestFailures);

pBuf = Util\_bufferUint16(pBuf, pMsg->msgStats.syncLossIndications);

pBuf = Util\_bufferUint16(pBuf, pMsg->msgStats.rxDecryptFailures);

pBuf = Util\_bufferUint16(pBuf, pMsg->msgStats.txEncryptFailures);

pBuf = Util\_bufferUint16(pBuf, Ssf\_resetCount);

pBuf = Util\_bufferUint16(pBuf, Ssf\_resetReseason);

pBuf = Util\_bufferUint16(pBuf, pMsg->msgStats.joinTime);

pBuf = Util\_bufferUint16(pBuf, pMsg->msgStats.interimDelay);

pBuf = Util\_bufferUint16(pBuf, pMsg->msgStats.numBroadcastMsgRcvd);

pBuf = Util\_bufferUint16(pBuf, pMsg->msgStats.numBroadcastMsglost);

pBuf = Util\_bufferUint16(pBuf, pMsg->msgStats.avgE2EDelay);

pBuf = Util\_bufferUint16(pBuf, pMsg->msgStats.worstCaseE2EDelay);

}

if(pMsg->frameControl & Smsgs\_dataFields\_configSettings)

{

pBuf = Util\_bufferUint32(pBuf,

pMsg->configSettings.reportingInterval);

pBuf = Util\_bufferUint32(pBuf,

pMsg->configSettings.pollingInterval);

}

ret = Sensor\_sendMsg(Smsgs\_cmdIds\_sensorData, pDstAddr, true, len, pMsgBuf);

Ssf\_free(pMsgBuf);

}

return (ret);

}

#endif // !defined(OAD\_IMG\_A)

/\*!

\* @brief Process the Config Request message.

\*

\* @param pDataInd - pointer to the data indication information

\*/

static void processConfigRequest(ApiMac\_mcpsDataInd\_t \*pDataInd)

{

Smsgs\_statusValues\_t stat = Smsgs\_statusValues\_invalid;

Smsgs\_configRspMsg\_t configRsp;

memset(&configRsp, 0, sizeof(Smsgs\_configRspMsg\_t));

/\* Make sure the message is the correct size \*/

if(pDataInd->msdu.len == SMSGS\_CONFIG\_REQUEST\_MSG\_LENGTH)

{

uint8\_t \*pBuf = pDataInd->msdu.p;

uint16\_t frameControl;

uint32\_t reportingInterval;

uint32\_t pollingInterval;

/\* Parse the message \*/

configSettings.cmdId = (Smsgs\_cmdIds\_t)\*pBuf++;

frameControl = Util\_parseUint16(pBuf);

pBuf += 2;

reportingInterval = Util\_parseUint32(pBuf);

pBuf += 4;

pollingInterval = Util\_parseUint32(pBuf);

stat = Smsgs\_statusValues\_success;

collectorAddr.addrMode = pDataInd->srcAddr.addrMode;

if(collectorAddr.addrMode == ApiMac\_addrType\_short)

{

collectorAddr.addr.shortAddr = pDataInd->srcAddr.addr.shortAddr;

}

else

{

memcpy(collectorAddr.addr.extAddr, pDataInd->srcAddr.addr.extAddr,

(APIMAC\_SADDR\_EXT\_LEN));

}

configSettings.frameControl = validateFrameControl(frameControl);

if(configSettings.frameControl != frameControl)

{

stat = Smsgs\_statusValues\_partialSuccess;

}

configRsp.frameControl = configSettings.frameControl;

if((reportingInterval < MIN\_REPORTING\_INTERVAL)

|| (reportingInterval > MAX\_REPORTING\_INTERVAL))

{

stat = Smsgs\_statusValues\_partialSuccess;

}

else

{

#ifndef POWER\_MEAS

configSettings.reportingInterval = reportingInterval;

#endif

{

uint32\_t randomNum;

randomNum = ((ApiMac\_randomByte() << 16) +

(ApiMac\_randomByte() << 8) + ApiMac\_randomByte());

randomNum = (randomNum % reportingInterval) +

SENSOR\_MIN\_POLL\_TIME;

Ssf\_setReadingClock(randomNum);

}

}

configRsp.reportingInterval = configSettings.reportingInterval;

if((pollingInterval < MIN\_POLLING\_INTERVAL)

|| (pollingInterval > MAX\_POLLING\_INTERVAL))

{

stat = Smsgs\_statusValues\_partialSuccess;

}

else

{

configSettings.pollingInterval = pollingInterval;

Jdllc\_setPollRate(configSettings.pollingInterval);

}

configRsp.pollingInterval = configSettings.pollingInterval;

}

/\* Send the response message \*/

configRsp.cmdId = Smsgs\_cmdIds\_configRsp;

configRsp.status = stat;

/\* Update the user \*/

Ssf\_configurationUpdate(&configRsp);

/\* Response the the source device \*/

sendConfigRsp(&pDataInd->srcAddr, &configRsp);

}

/\*!

\* @brief Process the Broadcast Control Msg.

\*

\* @param pDataInd - pointer to the data indication information

\*/

static void processBroadcastCtrlMsg(ApiMac\_mcpsDataInd\_t \*pDataInd)

{

Smsgs\_broadcastcmdmsg\_t broadcastCmd;

memset(&broadcastCmd, 0, sizeof(Smsgs\_broadcastcmdmsg\_t));

/\* Make sure the message is the correct size \*/

if(pDataInd->msdu.len == SMSGS\_BROADCAST\_CMD\_LENGTH)

{

uint8\_t \*pBuf = pDataInd->msdu.p;

uint16\_t broadcastMsgId;

/\* Parse the message \*/

uint8\_t cmdId = (Smsgs\_cmdIds\_t)\*pBuf++;

broadcastMsgId = Util\_parseUint16(pBuf);

/\* Process Broadcast Command Message \*/

Sensor\_msgStats.numBroadcastMsgRcvd++;

if(!initBroadcastMsg)

{

/\* Not the first broadcast msg rcvdd after join or a rejoin\*/

if((broadcastMsgId - lastRcvdBroadcastMsgId) > 1)

{

Sensor\_msgStats.numBroadcastMsglost +=

((broadcastMsgId - lastRcvdBroadcastMsgId) -1);

}

}

lastRcvdBroadcastMsgId = broadcastMsgId;

/\*To handle the very first broadcast msg rcvdd after join or a rejoin\*/

initBroadcastMsg = false;

/\* Switch On or Off LED based on broadcast Msg Id \*/

if((broadcastMsgId % 2) == 0)

{

Ssf\_OnLED();

}

else

{

Ssf\_OffLED();

}

}

}

/\*!

\* @brief Build and send Config Response message

\*

\* @param pDstAddr - Where to send the message

\* @param pMsg - pointer to the Config Response

\*

\* @return true if message was sent, false if not

\*/

static bool sendConfigRsp(ApiMac\_sAddr\_t \*pDstAddr, Smsgs\_configRspMsg\_t \*pMsg)

{

uint8\_t msgBuf[SMSGS\_CONFIG\_RESPONSE\_MSG\_LENGTH];

uint8\_t \*pBuf = msgBuf;

\*pBuf++ = (uint8\_t) Smsgs\_cmdIds\_configRsp;

pBuf = Util\_bufferUint16(pBuf, pMsg->status);

pBuf = Util\_bufferUint16(pBuf, pMsg->frameControl);

pBuf = Util\_bufferUint32(pBuf, pMsg->reportingInterval);

pBuf = Util\_bufferUint32(pBuf, pMsg->pollingInterval);

return (Sensor\_sendMsg(Smsgs\_cmdIds\_configRsp, pDstAddr, true,

SMSGS\_CONFIG\_RESPONSE\_MSG\_LENGTH, msgBuf));

}

/\*!

\* @brief Filter the frameControl with readings supported by this device.

\*

\* @param frameControl - suggested frameControl

\*

\* @return new frame control settings supported

\*/

static uint16\_t validateFrameControl(uint16\_t frameControl)

{

uint16\_t newFrameControl = 0;

#if defined(TEMP\_SENSOR)

if(frameControl & Smsgs\_dataFields\_tempSensor)

{

newFrameControl |= Smsgs\_dataFields\_tempSensor;

}

#endif

#if defined(LIGHT\_SENSOR)

if(frameControl & Smsgs\_dataFields\_lightSensor)

{

newFrameControl |= Smsgs\_dataFields\_lightSensor;

}

#endif

#if defined(HUMIDITY\_SENSOR)

if(frameControl & Smsgs\_dataFields\_humiditySensor)

{

newFrameControl |= Smsgs\_dataFields\_humiditySensor;

}

#endif

if(frameControl & Smsgs\_dataFields\_msgStats)

{

newFrameControl |= Smsgs\_dataFields\_msgStats;

}

if(frameControl & Smsgs\_dataFields\_configSettings)

{

newFrameControl |= Smsgs\_dataFields\_configSettings;

}

return (newFrameControl);

}

/\*!

\* @brief The device joined callback.

\*

\* @param pDevInfo - This device's information

\* @param pParentInfo - This is the parent's information

\*/

static void jdllcJoinedCb(ApiMac\_deviceDescriptor\_t \*pDevInfo,

Llc\_netInfo\_t \*pParentInfo)

{

uint32\_t randomNum = 0;

/\* Copy the parent information \*/

memcpy(&parentInfo, pParentInfo, sizeof(Llc\_netInfo\_t));

/\* Set the collector's address as the parent's address \*/

if (pParentInfo->fh && CONFIG\_RX\_ON\_IDLE)

{

collectorAddr.addrMode = ApiMac\_addrType\_extended;

memcpy(collectorAddr.addr.extAddr, pParentInfo->devInfo.extAddress,

(APIMAC\_SADDR\_EXT\_LEN));

}

else

{

collectorAddr.addrMode = ApiMac\_addrType\_short;

collectorAddr.addr.shortAddr = pParentInfo->devInfo.shortAddress;

}

/\* Start the reporting timer \*/

if(CONFIG\_FH\_ENABLE)

{

randomNum = ((ApiMac\_randomByte() << 16) +

(ApiMac\_randomByte() << 8) + ApiMac\_randomByte());

randomNum = (randomNum % configSettings.reportingInterval) +

SENSOR\_MIN\_POLL\_TIME;

Ssf\_setReadingClock(randomNum);

}

else

{

uint32\_t randomNum;

randomNum = ((ApiMac\_randomByte() << 16) +

(ApiMac\_randomByte() << 8) + ApiMac\_randomByte());

randomNum = (randomNum % configSettings.reportingInterval ) +

SENSOR\_MIN\_POLL\_TIME;

Ssf\_setReadingClock(randomNum);

}

/\* Inform the user of the joined information \*/

Ssf\_networkUpdate(rejoining, pDevInfo, pParentInfo);

#ifdef FEATURE\_SECURE\_COMMISSIONING

SM\_Sensor\_SAddress = pDevInfo->shortAddress;

#endif

if((rejoining == false) && (pParentInfo->fh == false))

{

#ifdef FEATURE\_MAC\_SECURITY

ApiMac\_status\_t stat;

/\* Add the parent to the security device list \*/

stat = Jdllc\_addSecDevice(pParentInfo->devInfo.panID,

pParentInfo->devInfo.shortAddress,

&pParentInfo->devInfo.extAddress, 0);

if(stat != ApiMac\_status\_success)

{

Ssf\_displayError("Auth Error: 0x", (uint8\_t)stat);

}

#endif /\* FEATURE\_MAC\_SECURITY \*/

}

#if (CONFIG\_MAC\_SUPERFRAME\_ORDER != 15) && defined(MAC\_NO\_AUTO\_REQ)

/\*

\* Set MAC Auto Request to false to enable multiple poll requests

\* per beacon interval

\*/

ApiMac\_mlmeSetReqBool(ApiMac\_attribute\_autoRequest, false);

#endif

#ifdef OSAL\_PORT2TIRTOS

/\* Calculate Join Time \*/

if(Clock\_getTicks() < joinTimeTicks)

{

joinTimeTicks = Clock\_getTicks() + (0xFFFFFFFF-joinTimeTicks);

}

else

{

joinTimeTicks = Clock\_getTicks() - joinTimeTicks;

}

#else

/\* Calculate Join Time \*/

if(ICall\_getTicks() < joinTimeTicks)

{

joinTimeTicks = ICall\_getTicks() + (0xFFFFFFFF-joinTimeTicks);

}

else

{

joinTimeTicks = ICall\_getTicks() - joinTimeTicks;

}

#endif

Sensor\_msgStats.joinTime = joinTimeTicks / TICKPERIOD\_MS\_US;

#ifdef DISPLAY\_PER\_STATS

/\* clear the stats used for PER so that we start out at a

\* zeroed state

\*/

Sensor\_msgStats.macAckFailures = 0;

Sensor\_msgStats.otherDataRequestFailures = 0;

Sensor\_msgStats.msgsSent = 0;

#endif

}

/\*!

\* @brief Disassociation indication callback.

\*

\* @param pExtAddress - extended address

\* @param reason - reason for disassociation

\*/

static void jdllcDisassocIndCb(ApiMac\_sAddrExt\_t \*pExtAddress,

ApiMac\_disassocateReason\_t reason)

{

/\* Stop the reporting timer \*/

Ssf\_setReadingClock(0);

Ssf\_clearNetworkInfo();

#ifdef FEATURE\_SECURE\_COMMISSIONING

SM\_removeEntryFromSeedKeyTable(pExtAddress);

ApiMac\_secDeleteDevice(pExtAddress);

Ssf\_clearDeviceKeyInfo();

#endif

#ifdef FEATURE\_NATIVE\_OAD

/\* OAD abort with no auto resume \*/

OADClient\_abort(false);

#endif //FEATURE\_NATIVE\_OAD

}

/\*!

\* @brief Disassociation confirm callback to an application intiated

\* disassociation request.

\*

\* @param pExtAddress - extended address

\* @param status - status of disassociation

\*/

static void jdllcDisassocCnfCb(ApiMac\_sAddrExt\_t \*pExtAddress,

ApiMac\_status\_t status)

{

/\* Stop the reporting timer \*/

Ssf\_setReadingClock(0);

Ssf\_clearNetworkInfo();

#ifdef FEATURE\_SECURE\_COMMISSIONING

SM\_removeEntryFromSeedKeyTable(pExtAddress);

ApiMac\_secDeleteDevice(pExtAddress);

Ssf\_clearDeviceKeyInfo();

#endif

#ifdef FEATURE\_NATIVE\_OAD

/\* OAD abort with no auto resume \*/

OADClient\_abort(false);

#endif //FEATURE\_NATIVE\_OAD

}

/\*!

\* @brief JDLLC state change callback.

\*

\* @param state - new state

\*/

static void jdllcStateChangeCb(Jdllc\_states\_t state)

{

#ifdef FEATURE\_NATIVE\_OAD

if( (state == Jdllc\_states\_joined) || (state == Jdllc\_states\_rejoined))

{

#if (CONFIG\_MAC\_SUPERFRAME\_ORDER == 15)

/\* resume an OAD that may have aborted \*/

OADClient\_resume(30000);

#else

/\* resume an OAD that may have aborted \*/

OADClient\_resume(60000);

#endif

}

else if(state == Jdllc\_states\_orphan)

{

/\* OAD abort with no auto resume \*/

OADClient\_abort(false);

}

#endif /\* FEATURE\_NATIVE\_OAD \*/

Ssf\_stateChangeUpdate(state);

#ifdef OAD\_IMG\_A

if( (state == Jdllc\_states\_joined) || (state == Jdllc\_states\_rejoined))

{

Util\_setEvent(&Sensor\_events, SENSOR\_OAD\_SEND\_RESET\_RSP\_EVT);

}

#endif

}

#ifdef FEATURE\_SECURE\_COMMISSIONING

/\*!

\* @brief Security manager failure processing function

\*/

void smFailCMProcessCb(ApiMac\_deviceDescriptor\_t \*devInfo, bool rxOnIdle, bool keyRefreshment)

{

/\* restore, write back current Pib value for auto request attribute \*/

ApiMac\_mlmeSetReqBool(ApiMac\_attribute\_autoRequest, currAutoReq);

if (keyRefreshment == true)

{

LCD\_WRITE\_STRING\_VALUE("Key Refresh Failed: 0x", SM\_Sensor\_SAddress, 16, 4);

}

else

{

LCD\_WRITE\_STRING\_VALUE("Commissioning Failed: 0x", SM\_Sensor\_SAddress, 16, 4);

}

}

/\*!

\* @brief Security manager success processing function

\*/

void smSuccessCMProcessCb(ApiMac\_deviceDescriptor\_t \*devInfo, bool keyRefreshment)

{

/\* restore, write back current Pib value for auto request attribute \*/

ApiMac\_mlmeSetReqBool(ApiMac\_attribute\_autoRequest, currAutoReq);

if (keyRefreshment == true)

{

LCD\_WRITE\_STRING\_VALUE("Key Refreshed: 0x", SM\_Sensor\_SAddress, 16, 4);

}

else

{

LCD\_WRITE\_STRING\_VALUE("Commissioned: 0x", SM\_Sensor\_SAddress, 16, 4);

}

}

#endif /\* FEATURE\_SECURE\_COMMISSIONING \*/

**------------------------------------------------------------------------------------**

**Modified Code:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**CCFG.C FOR SENSOR:**

/\*

\* Copyright (c) 2015-2017, Texas Instruments Incorporated

\* All rights reserved.

\*

\* Redistribution and use in source and binary forms, with or without

\* modification, are permitted provided that the following conditions

\* are met:

\*

\* \* Redistributions of source code must retain the above copyright

\* notice, this list of conditions and the following disclaimer.

\*

\* \* Redistributions in binary form must reproduce the above copyright

\* notice, this list of conditions and the following disclaimer in the

\* documentation and/or other materials provided with the distribution.

\*

\* \* Neither the name of Texas Instruments Incorporated nor the names of

\* its contributors may be used to endorse or promote products derived

\* from this software without specific prior written permission.

\*

\* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"

\* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,

\* THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR

\* PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR

\* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,

\* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,

\* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;

\* OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,

\* WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR

\* OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,

\* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*/

/\*

\* ======== ccfg.c ========

\* Customer Configuration for CC26xx and CC13xx devices. This file is used to

\* configure Boot ROM, start-up code, and SW radio behaviour.

\*

\* By default, driverlib startup\_files/ccfg.c settings are used. However, if

\* changes are required there are two means to do so:

\*

\* 1. Remove this file and copy driverlib's startup\_files/ccfg.c file in

\* its place. Make all changes to the file. Changes made are local to

\* the project and will not affect other projects.

\*

\* 2. Perform changes to driverlib startup\_files/ccfg.c file. Changes

\* made to this file will be applied to all projects. This file must

\* remain unmodified.

\*/

#include <ti/devices/DeviceFamily.h>

#include DeviceFamily\_constructPath(startup\_files/ccfg.c)

**MAIN\_TIRTOS.C:**

**/\***

**\* Copyright (c) 2016-2019, Texas Instruments Incorporated**

**\* All rights reserved.**

**\***

**\* Redistribution and use in source and binary forms, with or without**

**\* modification, are permitted provided that the following conditions**

**\* are met:**

**\***

**\* \* Redistributions of source code must retain the above copyright**

**\* notice, this list of conditions and the following disclaimer.**

**\***

**\* \* Redistributions in binary form must reproduce the above copyright**

**\* notice, this list of conditions and the following disclaimer in the**

**\* documentation and/or other materials provided with the distribution.**

**\***

**\* \* Neither the name of Texas Instruments Incorporated nor the names of**

**\* its contributors may be used to endorse or promote products derived**

**\* from this software without specific prior written permission.**

**\***

**\* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"**

**\* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,**

**\* THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR**

**\* PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR**

**\* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,**

**\* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,**

**\* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;**

**\* OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,**

**\* WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR**

**\* OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,**

**\* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.**

**\*/**

**/\***

**\* ======== main\_tirtos.c ========**

**\*/**

**#include <stdint.h>**

**/\* POSIX Header files \*/**

**#include <pthread.h>**

**/\* RTOS header files \*/**

**#include <ti/sysbios/BIOS.h>**

**/\* Driver header files \*/**

**#include <ti/drivers/GPIO.h>**

**/\* Example/Board Header files \*/**

**#include <ti/drivers/Board.h>**

**/\* Mutex to protect the reading/writing of the temperature variables \*/**

**pthread\_mutex\_t temperatureMutex;**

**extern void \*temperatureThread(void \*arg0);**

**extern void \*consoleThread(void \*arg0);**

**/\* Stack size in bytes. Large enough in case debug kernel is used. \*/**

**#define THREADSTACKSIZE 1024**

**/\***

**\* ======== main ========**

**\*/**

**int main(void)**

**{**

**pthread\_t thread;**

**pthread\_attr\_t attrs;**

**struct sched\_param priParam;**

**int retc;**

**/\* Call driver init functions \*/**

**Board\_init();**

**/\* Initialize the attributes structure with default values \*/**

**pthread\_attr\_init(&attrs);**

**/\* Set priority, detach state, and stack size attributes \*/**

**priParam.sched\_priority = 1;**

**retc = pthread\_attr\_setschedparam(&attrs, &priParam);**

**retc |= pthread\_attr\_setdetachstate(&attrs, PTHREAD\_CREATE\_DETACHED);**

**retc |= pthread\_attr\_setstacksize(&attrs, THREADSTACKSIZE);**

**if (retc != 0) {**

**/\* failed to set attributes \*/**

**while (1) {}**

**}**

**retc = pthread\_create(&thread, &attrs, consoleThread, NULL);**

**if (retc != 0) {**

**/\* pthread\_create() failed \*/**

**while (1) {}**

**}**

**/\***

**\* Let's make the temperature thread a higher priority .**

**\* Higher number means higher priority in TI-RTOS.**

**\*/**

**priParam.sched\_priority = 2;**

**retc = pthread\_attr\_setschedparam(&attrs, &priParam);**

**if (retc != 0) {**

**/\* failed to set priority \*/**

**while (1) {}**

**}**

**retc = pthread\_create(&thread, &attrs, temperatureThread, NULL);**

**if (retc != 0) {**

**/\* pthread\_create() failed \*/**

**while (1) {}**

**}**

**/\* Create a mutex that will protect temperature variables \*/**

**retc = pthread\_mutex\_init(&temperatureMutex, NULL);**

**if (retc != 0) {**

**/\* pthread\_mutex\_init() failed \*/**

**while (1) {}**

**}**

**/\* Initialize the GPIO since multiple threads are using it \*/**

**GPIO\_init();**

**/\* Start the TI-RTOS scheduler \*/**

**BIOS\_start();**

**return (0);**

**}**

**TEMPERATURE.C**

**/\***

**\* Copyright (c) 2016-2019, Texas Instruments Incorporated**

**\* All rights reserved.**

**\***

**\* Redistribution and use in source and binary forms, with or without**

**\* modification, are permitted provided that the following conditions**

**\* are met:**

**\***

**\* \* Redistributions of source code must retain the above copyright**

**\* notice, this list of conditions and the following disclaimer.**

**\***

**\* \* Redistributions in binary form must reproduce the above copyright**

**\* notice, this list of conditions and the following disclaimer in the**

**\* documentation and/or other materials provided with the distribution.**

**\***

**\* \* Neither the name of Texas Instruments Incorporated nor the names of**

**\* its contributors may be used to endorse or promote products derived**

**\* from this software without specific prior written permission.**

**\***

**\* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"**

**\* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,**

**\* THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR**

**\* PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR**

**\* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,**

**\* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,**

**\* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;**

**\* OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,**

**\* WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR**

**\* OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,**

**\* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.**

**\*/**

**/\***

**\* ======== temperature.c ========**

**\*/**

**#include <stdint.h>**

**#include <stddef.h>**

**#include <unistd.h>**

**#include <ti/display/Display.h>**

**/\* POSIX Header files \*/**

**#include <pthread.h>**

**#include <semaphore.h>**

**#include <signal.h>**

**#include <time.h>**

**/\* Driver Header files \*/**

**#include <ti/drivers/GPIO.h>**

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

**/\* Example/Board Header files \*/**

**#include "Board.h"**

**/\* ======== Si7021 Registers ======== \*/**

**#define Si7021\_TMP\_REG 0xE3**

**#define Si7021\_HUM\_REG 0xE5**

**#define Si7021\_ADDR 0x40**

**/\***

**\* ======== HIGH\_TEMP ========**

**\* Send alert when this temperature (in Celsius) is exceeded**

**\*/**

**#define HIGH\_TEMP 30**

**/\***

**\* ======== TMP Registers ========**

**\*/**

**#define TMP006\_REG 0x0001 /\* Die Temp Result Register for TMP006 \*/**

**#define TMP116\_REG 0x0000 /\* Die Temp Result Register for TMP116 \*/**

**/\***

**\* The CC32XX LaunchPads come with an on-board TMP006 or TMP116 temperature**

**\* sensor depending on the revision. Newer revisions come with the TMP116.**

**\* The Build Automation Sensors (BOOSTXL-BASSENSORS) BoosterPack**

**\* contains a TMP116.**

**\***

**\* We are using the DIE temperature because it's cool!**

**\***

**\* Additionally: no calibration is being done on the TMPxxx device to simplify**

**\* the example code.**

**\*/**

**#define TMP006\_ADDR 0x41;**

**#define TMP116\_BP\_ADDR 0x48;**

**#define TMP116\_LP\_ADDR 0x49;**

**/\* Temperature written by the temperature thread and read by console thread \*/**

**volatile float temperatureC;**

**volatile float temperatureF;**

**volatile float temperaturef;**

**volatile float temperature;**

**volatile float temp;**

**volatile float sample;**

**Display\_Handle display;**

**/\* Mutex to protect the reading/writing of the temperature variables \*/**

**extern pthread\_mutex\_t temperatureMutex;**

**/\***

**\* ======== clearAlert ========**

**\* Clear the LED**

**\*/**

**//static void clearAlert(float temperature)**

**//{**

**// GPIO\_write(Board\_GPIO\_LED0, Board\_GPIO\_LED\_OFF);**

**//}**

**/\***

**\* ======== sendAlert ========**

**\* Okay, just light a LED in this example, but with the SimpleLink SDK,**

**\* you could send it out over the radio to something cool!**

**\*/**

**//static void sendAlert(float temperature)**

**//{**

**// GPIO\_write(Board\_GPIO\_LED0, Board\_GPIO\_LED\_ON);**

**//}**

**/\***

**\* ======== postSem ========**

**\* Function called when the timer (created in setupTimer) expires.**

**\*/**

**static void postSem(union sigval val)**

**{**

**sem\_t \*sem = (sem\_t\*)(val.sival\_ptr);**

**sem\_post(sem);**

**}**

**/\***

**\* ======== setupTimer ========**

**\* Create a timer that will expire at the period specified by the**

**\* time arguments. When the timer expires, the passed in semaphore**

**\* will be posted by the postSem function.**

**\***

**\* A non-zero return indicates a failure.**

**\*/**

**int setupTimer(sem\_t \*sem, timer\_t \*timerid, time\_t sec, long nsec)**

**{**

**struct sigevent sev;**

**struct itimerspec its;**

**int retc;**

**retc = sem\_init(sem, 0, 0);**

**if (retc != 0) {**

**return(retc);**

**}**

**/\* Create the timer that wakes up the thread that will pend on the sem. \*/**

**sev.sigev\_notify = SIGEV\_SIGNAL;**

**sev.sigev\_value.sival\_ptr = sem;**

**sev.sigev\_notify\_function = &postSem;**

**sev.sigev\_notify\_attributes = NULL;**

**retc = timer\_create(CLOCK\_MONOTONIC, &sev, timerid);**

**if (retc != 0) {**

**return(retc);**

**}**

**/\* Set the timer to go off at the specified period \*/**

**its.it\_interval.tv\_sec = sec;**

**its.it\_interval.tv\_nsec = nsec;**

**its.it\_value.tv\_sec = sec;**

**its.it\_value.tv\_nsec = nsec;**

**retc = timer\_settime(\*timerid, 0, &its, NULL);**

**if (retc != 0) {**

**timer\_delete(\*timerid);**

**return(retc);**

**}**

**return(0);**

**}**

**/\***

**\* ======== temperatureThread ========**

**\* This thread reads the temperature every second via I2C and sends an**

**\* alert if it goes above HIGH\_TEMP.**

**\*/**

**void \*temperatureThread(void \*arg0)**

**{**

**uint8\_t txBuffer[1];**

**uint8\_t rxBuffer[2];**

**I2C\_Handle i2c;**

**I2C\_Params i2cParams;**

**I2C\_Transaction i2cTransaction;**

**sem\_t semTimer;**

**// timer\_t timerid;**

**// int retc;**

**/\* Configure the LED and if applicable, the TMP116\_EN pin \*/**

**GPIO\_setConfig(Board\_GPIO\_LED0, GPIO\_CFG\_OUT\_STD | GPIO\_CFG\_OUT\_LOW);**

**#ifdef Board\_GPIO\_TMP116\_EN**

**GPIO\_setConfig(Board\_GPIO\_TMP116\_EN, GPIO\_CFG\_OUT\_STD | GPIO\_CFG\_OUT\_HIGH);**

**/\* 1.5 ms reset time for the TMP116 \*/**

**sleep(1);**

**#endif**

**/\***

**\* Create/Open the I2C that talks to the TMP sensor**

**\*/**

**I2C\_init();**

**Display\_init();**

**I2C\_Params\_init(&i2cParams);**

**i2cParams.bitRate = I2C\_400kHz;**

**i2c = I2C\_open(Board\_I2C\_TMP, &i2cParams);**

**if (i2c == NULL) {**

**while (1);**

**}**

**/\* Common I2C transaction setup \*/**

**i2cTransaction.writeBuf = txBuffer;**

**i2cTransaction.writeCount = 1;**

**i2cTransaction.readBuf = rxBuffer;**

**i2cTransaction.readCount = 2;**

**/\***

**\* Determine which I2C sensor is present.**

**\* We will prefer sensors in this order: TMP116 (on BoosterPacks),**

**\* TMP116 (on-board CC32XX LaunchPads), and last TMP006**

**\* (on older CC32XX LaunchPads).**

**\*/**

**/\***

**// Try TMP116 values**

**txBuffer[0] = TMP116\_REG;**

**i2cTransaction.slaveAddress = TMP116\_BP\_ADDR;**

**if (!I2C\_transfer(i2c, &i2cTransaction)) {**

**// Not BP TMP116, try LP TMP116**

**i2cTransaction.slaveAddress = TMP116\_LP\_ADDR;**

**if (!I2C\_transfer(i2c, &i2cTransaction)) {**

**// Not a TMP116 try TMP006**

**txBuffer[0] = TMP006\_REG;**

**i2cTransaction.slaveAddress = TMP006\_ADDR;**

**if (!I2C\_transfer(i2c, &i2cTransaction)) {**

**// Could not resolve a sensor, error**

**while(1);**

**}**

**}**

**}**

**\*/**

**// Try Si7021**

**txBuffer[0] = Si7021\_TMP\_REG;**

**i2cTransaction.slaveAddress = Si7021\_ADDR;**

**if (!I2C\_transfer(i2c, &i2cTransaction))**

**{**

**// Could not resolve a sensor, error**

**Display\_printf(display, 0, 0, "Error. No TMP sensor found!");**

**while(1);**

**}**

**else**

**{**

**Display\_printf(display, 0, 0, "Detected Si7021 sensor.");**

**}**

**// Take 20 samples and print them out onto the console**

**for (sample = 0; sample < 100; sample++)**

**{**

**if (I2C\_transfer(i2c, &i2cTransaction))**

**{**

**//**

**// Extract degrees C from the received data;**

**// see Si7021 datasheet**

**//**

**temp = (rxBuffer[0] << 8) | (rxBuffer[1]);**

**temperature = (((175.72 \* temp)/ 65536) - 46.85); // celsius**

**temperaturef = (temperature \* (1.8)) + 32; //farenheit**

**Display\_printf(display, 0, 0, "Sample %u: %d (C)", sample, temperaturef);**

**}**

**else**

**{**

**Display\_printf(display, 0, 0, "I2C Bus fault.");**

**}**

**}**

**/\***

**\* The temperature thread blocks on the semTimer semaphore, which the**

**\* timerId timer will post every second. The timer is created in the**

**\* setupTimer function. It's returned so the thread could change the**

**\* period or delete it if desired.**

**\*/**

**// retc = setupTimer(&semTimer, &timerid, 1, 0);**

**// if (retc != 0) {**

**// while (1);**

**// }**

**// while (1)**

**// {**

**// if (I2C\_transfer(i2c, &i2cTransaction)) {**

**// /\***

**// \* Extract degrees C from the received data; see sensor datasheet.**

**// \* Make sure we are updating the global temperature variables**

**// \* in a thread-safe manner.**

**// \*/**

**// pthread\_mutex\_lock(&temperatureMutex);**

**// temperatureC = (rxBuffer[0] << 6) | (rxBuffer[1] >> 2);**

**// temperatureC \*= 0.03125;**

**// temperatureF = temperatureC \* 9 / 5 + 32;**

**// pthread\_mutex\_unlock(&temperatureMutex);**

**//**

**// /\* Send an alert if the temperature is too high!! \*/**

**// if ((int)temperatureC >= HIGH\_TEMP) {**

**// sendAlert(temperatureC);**

**// }**

**// else {**

**// clearAlert(temperatureC);**

**// }**

**// }**

**// /\* Block until the timer posts the semaphore. \*/**

**// retc = sem\_wait(&semTimer);**

**// if (retc == -1) {**

**// while (1);**

**// }**

**// }**

**}**

**CONSOLE.C**

**/\***

**\* Copyright (c) 2016-2019, Texas Instruments Incorporated**

**\* All rights reserved.**

**\***

**\* Redistribution and use in source and binary forms, with or without**

**\* modification, are permitted provided that the following conditions**

**\* are met:**

**\***

**\* \* Redistributions of source code must retain the above copyright**

**\* notice, this list of conditions and the following disclaimer.**

**\***

**\* \* Redistributions in binary form must reproduce the above copyright**

**\* notice, this list of conditions and the following disclaimer in the**

**\* documentation and/or other materials provided with the distribution.**

**\***

**\* \* Neither the name of Texas Instruments Incorporated nor the names of**

**\* its contributors may be used to endorse or promote products derived**

**\* from this software without specific prior written permission.**

**\***

**\* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"**

**\* AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,**

**\* THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR**

**\* PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR**

**\* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,**

**\* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,**

**\* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;**

**\* OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,**

**\* WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR**

**\* OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,**

**\* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.**

**\*/**

**/\***

**\* ======== console.c ========**

**\*/**

**#include <stdint.h>**

**#include <string.h>**

**#include <stdbool.h>**

**/\* POSIX Header files \*/**

**#include <pthread.h>**

**#include <semaphore.h>**

**/\* Driver Header files \*/**

**#include <ti/drivers/GPIO.h>**

**#include <ti/drivers/UART.h>**

**#ifdef CC32XX**

**#include <ti/drivers/Power.h>**

**#include <ti/drivers/power/PowerCC32XX.h>**

**#endif**

**/\* Example/Board Header files \*/**

**#include "Board.h"**

**/\* Console display strings \*/**

**const char consoleDisplay[] = "\fConsole (h for help)\r\n";**

**const char helpPrompt[] = "Valid Commands\r\n" \**

**"--------------\r\n" \**

**"h: help\r\n" \**

**"q: quit and shutdown UART\r\n" \**

**"c: clear the screen\r\n" \**

**"t: display current temperature\r\n";**

**const char byeDisplay[] = "Bye! Hit button1 to start UART again\r\n";**

**const char tempStartDisplay[] = "Current temp = ";**

**const char tempMidDisplay[] = "C (";**

**const char tempEndDisplay[] = "F)\r\n";**

**const char cleanDisplay[] = "\f";**

**const char userPrompt[] = "> ";**

**const char readErrDisplay[] = "Problem read UART.\r\n";**

**/\* Used to determine whether to have the thread block \*/**

**volatile bool uartEnabled = true;**

**sem\_t semConsole;**

**/\* Temperature written by the temperature thread and read by console thread \*/**

**extern volatile float temperature;**

**extern volatile float temperaturef;**

**/\* Mutex to protect the reading/writing of the float temperature \*/**

**extern pthread\_mutex\_t temperatureMutex;**

**/\* Used itoa instead of sprintf to help minimize the size of the stack \*/**

**static void itoa(int n, char s[]);**

**/\***

**\* ======== gpioButtonFxn ========**

**\* Callback function for the GPIO interrupt on Board\_GPIO\_BUTTON1.**

**\* There is no debounce logic here since we are just looking for**

**\* a button push. The uartEnabled variable protects use against any**

**\* additional interrupts cased by the bouncing of the button.**

**\*/**

**void gpioButtonFxn(uint\_least8\_t index)**

**{**

**/\* If disabled, enable and post the semaphore \*/**

**if (uartEnabled == false) {**

**uartEnabled = true;**

**sem\_post(&semConsole);**

**}**

**}**

**/\***

**\* ======== simpleConsole ========**

**\* Handle the user input. Currently this console does not handle**

**\* user back-spaces or other "hard" characters.**

**\*/**

**void simpleConsole(UART\_Handle uart)**

**{**

**char cmd;**

**int status;**

**char tempStr[8];**

**int localTemperatureC;**

**int localTemperatureF;**

**UART\_write(uart, consoleDisplay, sizeof(consoleDisplay));**

**/\* Loop until read fails or user quits \*/**

**while (1) {**

**UART\_write(uart, userPrompt, sizeof(userPrompt));**

**status = UART\_read(uart, &cmd, sizeof(cmd));**

**if (status == 0) {**

**UART\_write(uart, readErrDisplay, sizeof(readErrDisplay));**

**cmd = 'q';**

**}**

**switch (cmd) {**

**case 't':**

**UART\_write(uart, tempStartDisplay, sizeof(tempStartDisplay));**

**/\***

**\* Make sure we are accessing the global float temperature variables**

**\* in a thread-safe manner.**

**\*/**

**pthread\_mutex\_lock(&temperatureMutex);**

**localTemperatureC = (int)temperature;**

**localTemperatureF = (int)temperaturef;**

**pthread\_mutex\_unlock(&temperatureMutex);**

**itoa((int)localTemperatureC, tempStr);**

**UART\_write(uart, tempStr, strlen(tempStr));**

**UART\_write(uart, tempMidDisplay, sizeof(tempMidDisplay));**

**itoa((int)localTemperatureF, tempStr);**

**UART\_write(uart, tempStr, strlen(tempStr));**

**UART\_write(uart, tempEndDisplay, sizeof(tempEndDisplay));**

**break;**

**case 'c':**

**UART\_write(uart, cleanDisplay, sizeof(cleanDisplay));**

**break;**

**case 'q':**

**UART\_write(uart, byeDisplay, sizeof(byeDisplay));**

**return;**

**case 'h':**

**default:**

**UART\_write(uart, helpPrompt, sizeof(helpPrompt));**

**break;**

**}**

**}**

**}**

**/\***

**\* ======== consoleThread ========**

**\*/**

**void \*consoleThread(void \*arg0)**

**{**

**UART\_Params uartParams;**

**UART\_Handle uart;**

**int retc;**

**#ifdef CC32XX**

**/\***

**\* The CC3220 examples by default do not have power management enabled.**

**\* This allows a better debug experience. With the power management**

**\* enabled, if the device goes into a low power mode the emulation**

**\* session is lost.**

**\* Let's enable it and also configure the button to wake us up.**

**\*/**

**PowerCC32XX\_Wakeup wakeup;**

**PowerCC32XX\_getWakeup(&wakeup);**

**wakeup.wakeupGPIOFxnLPDS = gpioButtonFxn;**

**PowerCC32XX\_configureWakeup(&wakeup);**

**Power\_enablePolicy();**

**#endif**

**/\* Configure the button pin \*/**

**GPIO\_setConfig(Board\_GPIO\_BUTTON1, GPIO\_CFG\_IN\_PU | GPIO\_CFG\_IN\_INT\_FALLING);**

**/\* install Button callback and enable it \*/**

**GPIO\_setCallback(Board\_GPIO\_BUTTON1, gpioButtonFxn);**

**GPIO\_enableInt(Board\_GPIO\_BUTTON1);**

**retc = sem\_init(&semConsole, 0, 0);**

**if (retc == -1) {**

**while (1);**

**}**

**UART\_init();**

**/\***

**\* Initialize the UART parameters outside the loop. Let's keep**

**\* most of the defaults (e.g. baudrate = 115200) and only change the**

**\* following.**

**\*/**

**UART\_Params\_init(&uartParams);**

**uartParams.writeDataMode = UART\_DATA\_BINARY;**

**uartParams.readDataMode = UART\_DATA\_BINARY;**

**uartParams.readReturnMode = UART\_RETURN\_FULL;**

**/\* Loop forever to start the console \*/**

**while (1) {**

**if (uartEnabled == false) {**

**retc = sem\_wait(&semConsole);**

**if (retc == -1) {**

**while (1);**

**}**

**}**

**/\* Create a UART for the console \*/**

**uart = UART\_open(Board\_UART0, &uartParams);**

**if (uart == NULL) {**

**while (1);**

**}**

**simpleConsole(uart);**

**/\***

**\* Since we returned from the console, we need to close the UART.**

**\* The Power Manager will go into a lower power mode when the UART**

**\* is closed.**

**\*/**

**UART\_close(uart);**

**uartEnabled = false;**

**}**

**}**

**/\***

**\* The following function is from good old K & R.**

**\*/**

**static void reverse(char s[])**

**{**

**int i, j;**

**char c;**

**for (i = 0, j = strlen(s)-1; i<j; i++, j--) {**

**c = s[i];**

**s[i] = s[j];**

**s[j] = c;**

**}**

**}**

**/\***

**\* The following function is from good old K & R.**

**\*/**

**static void itoa(int n, char s[])**

**{**

**int i, sign;**

**if ((sign = n) < 0) /\* record sign \*/**

**n = -n; /\* make n positive \*/**

**i = 0;**

**do { /\* generate digits in reverse order \*/**

**s[i++] = n % 10 + '0'; /\* get next digit \*/**

**} while ((n /= 10) > 0); /\* delete it \*/**

**if (sign < 0)**

**s[i++] = '-';**

**s[i] = '\0';**

**reverse(s);**

**}**