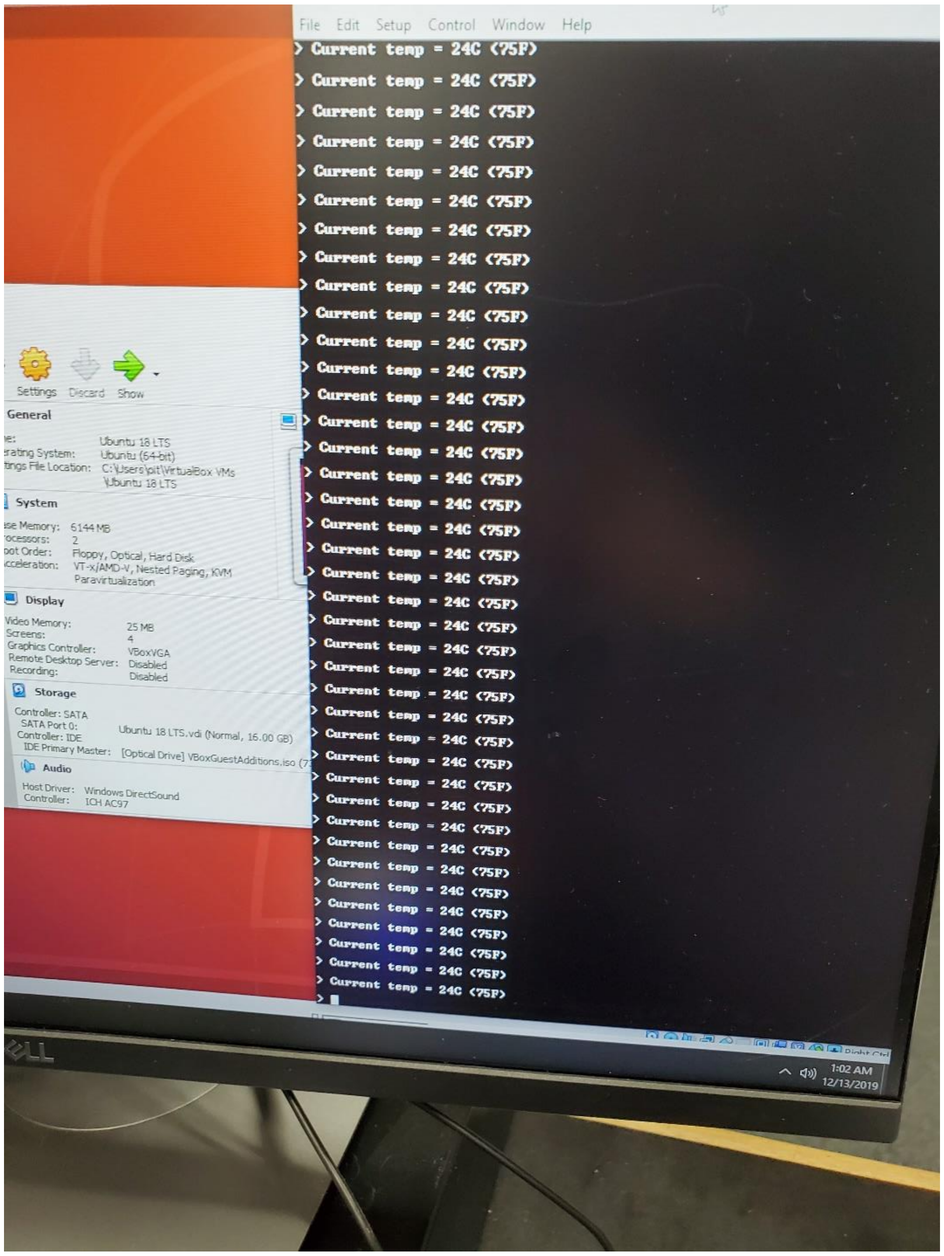
Ivan Soto
Github root directory: https://github.com/sotoi2/Class3.0.4

FINAL PROJECT: Ivan Soto

`Youtube Links:`
https://www.youtube.com/watch?v=uJfJHFsUyqM

Partner: Jett Guerrero



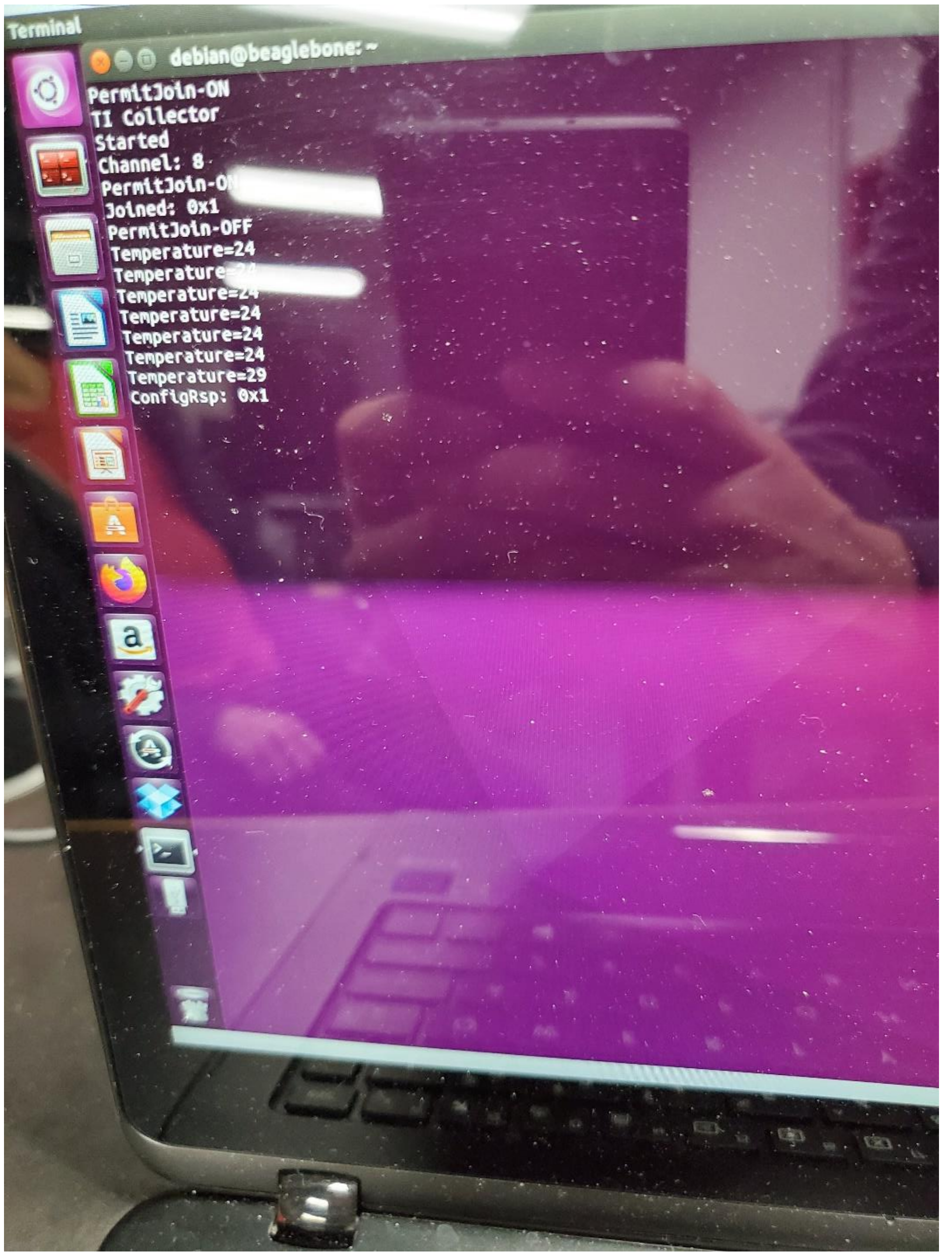
```
General
ne:                    Ubuntu 18 LTS
erating System:        Ubuntu (64-bit)
ttings File Location:  C:\Users\pit\VirtualBox VMs
                       \Ubuntu 18 LTS
System
se Memory:     6144 MB
rocessors:     2
oot Order:     Floppy, Optical, Hard Disk
cceleration:   VT-x/AMD-V, Nested Paging, KVM
               Paravirtualization
Display
Video Memory:            25 MB
Screens:                 4
Graphics Controller:     VBoxVGA
Remote Desktop Server:   Disabled
Recording:               Disabled
Storage
Controller: SATA
 SATA Port 0:        Ubuntu 18 LTS.vdi (Normal, 16.00 GB)
Controller: IDE
 IDE Primary Master: [Optical Drive] VBoxGuestAdditions.iso (7
Audio
Host Driver:  Windows DirectSound
Controller:   ICH AC97
```

```
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> 
```

Temperature being displayed by pressing ‘t’ .

<u>Grading scheme:</u> 30% Coding, 30% Documentation, 40% Execution/Video.

Ivan Soto
Github root directory: https://github.com/sotoi2/Class3.0.4

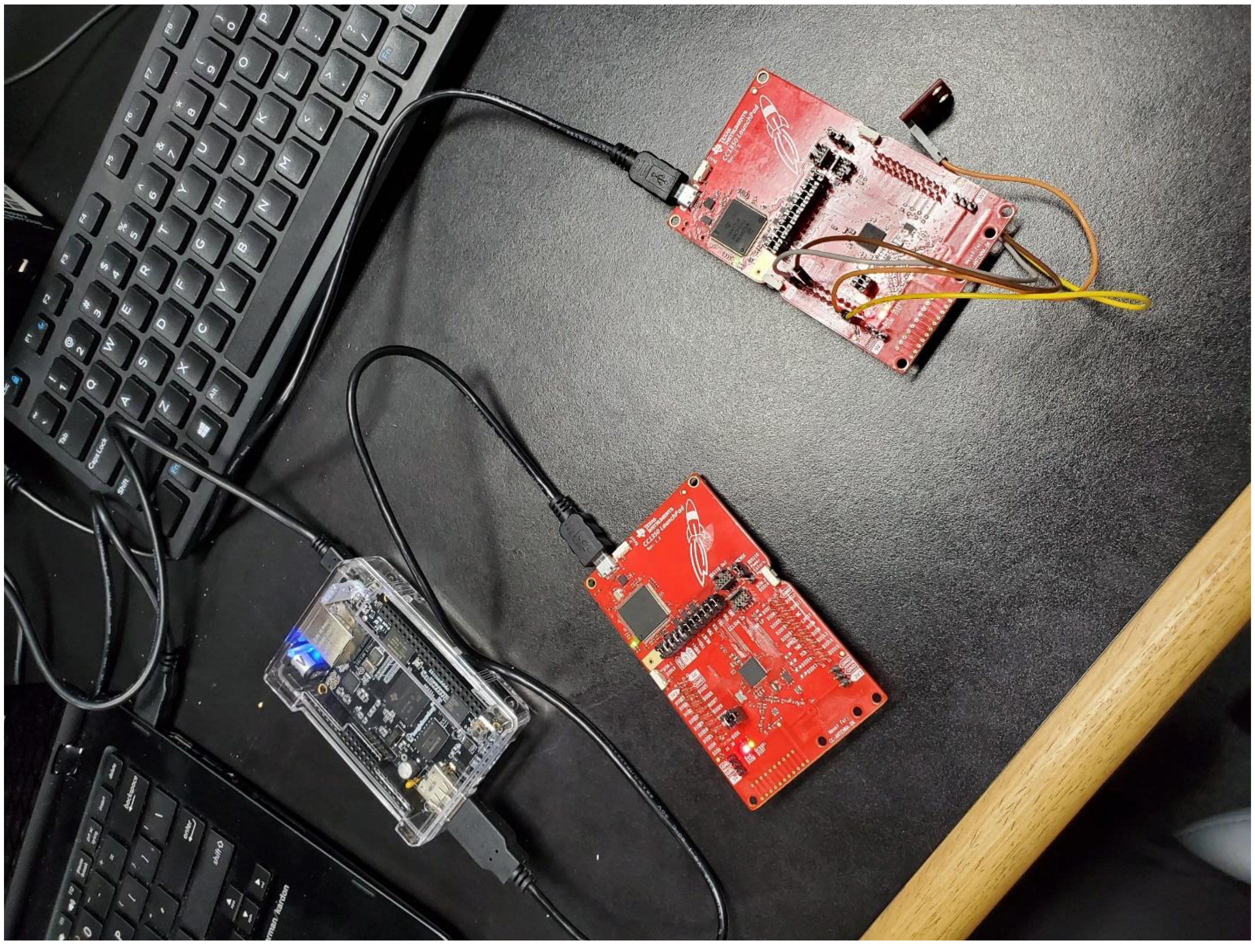


```
PermitJoin-ON
TI Collector
Started
Channel: 8
PermitJoin-ON
Joined: 0x1
PermitJoin-OFF
Temperature=24
Temperature=24
Temperature=24
Temperature=24
Temperature=24
Temperature=24
Temperature=29
ConfigRsp: 0x1
```

This reading comes from the beaglebone black.

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

Ivan Soto
Github root directory: https://github.com/sotoi2/Class3.0.4


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
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
 ******************************************************************************

 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);
                    }
                }
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
                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
}
```

```c
/*!
 * @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);
    }
```

Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

```
    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:

```
 * 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);
}
```