Ali Asadi
Github root directory: https://github.com/aliasadivet/Lab_Submition-.git

CC1350

Lab05

Date Submitted: **10/10/2019**

Task 00: Execute provided code

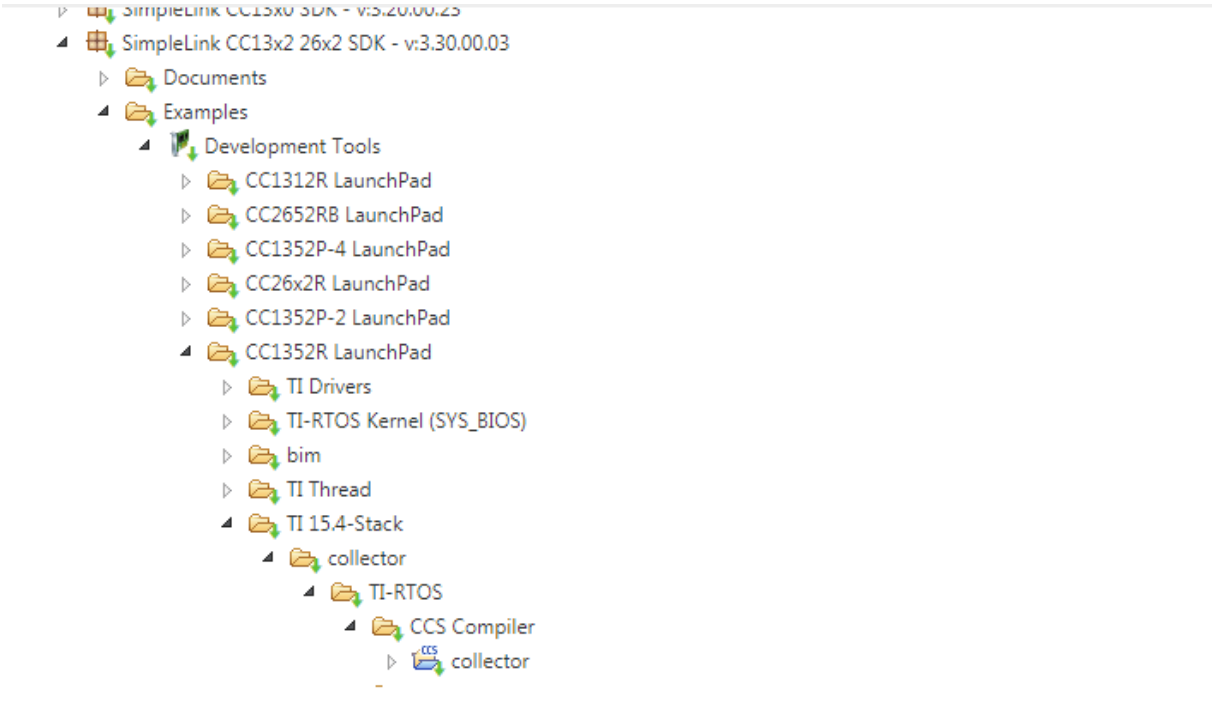
I have worked by myself. I have 2 boards, CC1350 and CC1352R
I used CC1350 as sensor and CC1352R as collector.

Part 1

Youtube Link: No submition required

Task 01:

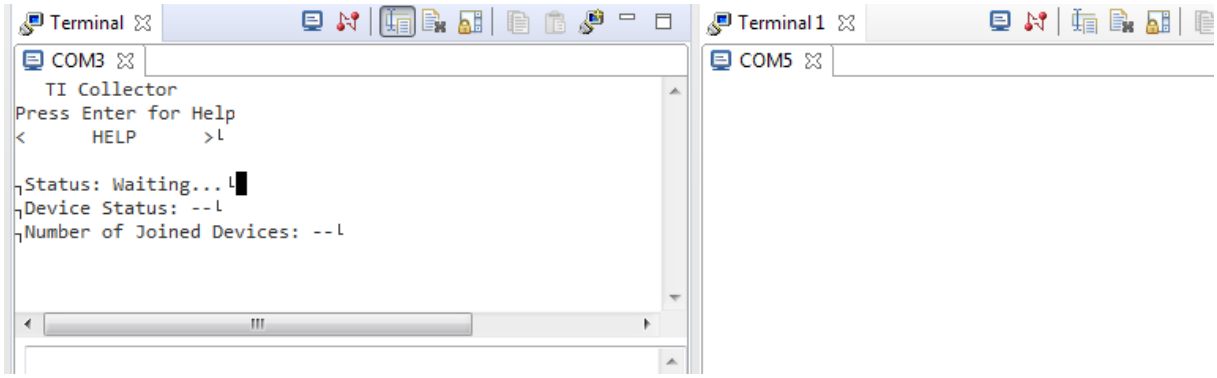
1. Import collector exp.



2. collector example terminal

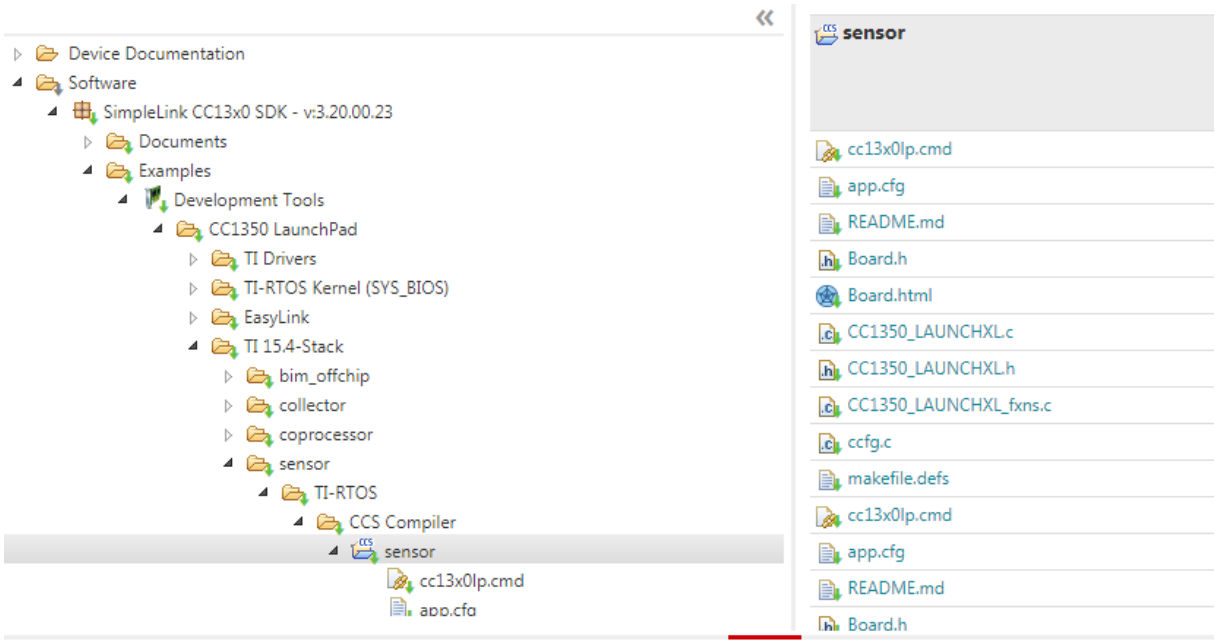
Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

Ali Asadi
Github root directory: https://github.com/aliasadivet/Lab_Submition-.git

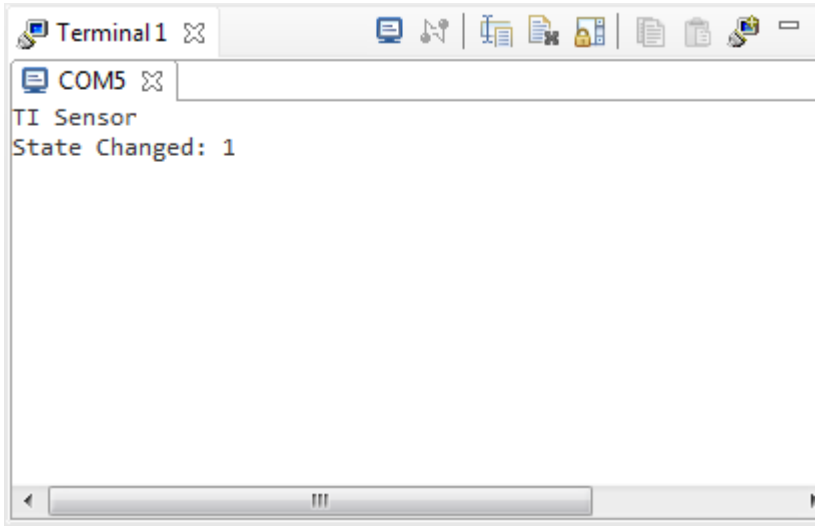


Task 02:

1. Import sensor exp.



2. sensor exp. terminal

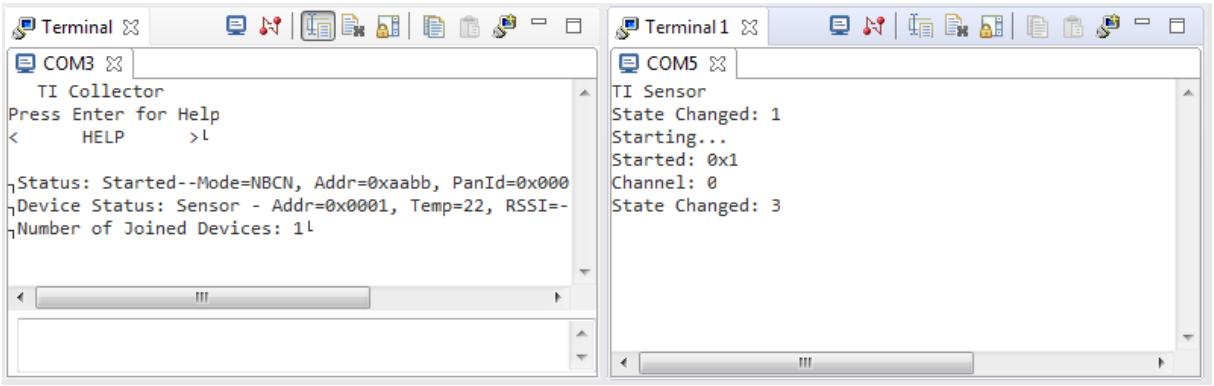


Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

Ali Asadi
Github root directory: https://github.com/aliasadivet/Lab_Submition-.git

Task 03:

Screenshot of sensor and collector terminals while they are joined.
Temperature of 22 Celsius has been collected.



Task 04:

Modifications on the code to update the sensor's reporting rate.

```
60
61 #define CONFIG_POLLING_INTERVAL 2000
62 #define CONFIG_REPORTING_INTERVAL 1000
63 /* Unused CONFIG_FH_NETNAME generated for code compilation */
64 #define CONFIG_FH_NETNAME {"FHTest"}
```

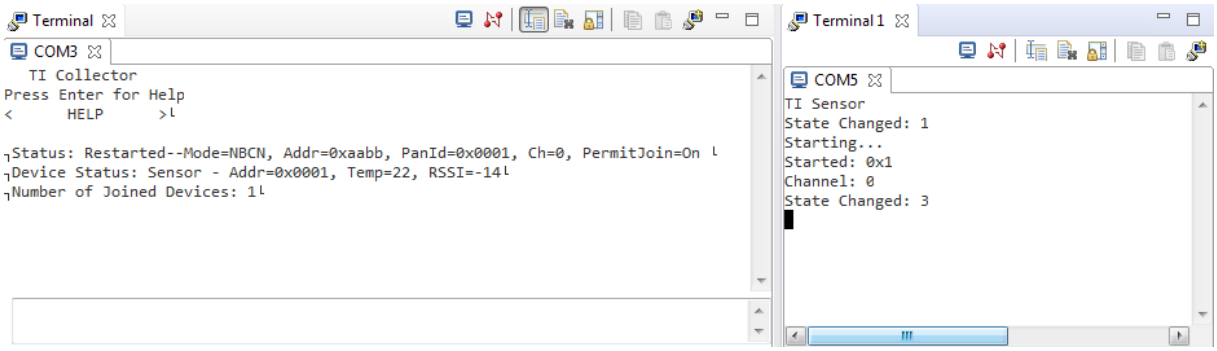
```
105
106 /* Reporting Interval Min and Max (in milliseconds) */
107 #define MIN_REPORTING_INTERVAL 500
108 #define MAX_REPORTING_INTERVAL 360000
109
110 /* Polling Interval Min and Max (in milliseconds) */
111 #define MIN_POLLING_INTERVAL 100
112 #define MAX_POLLING_INTERVAL 10000
113
114 /* Blink Time for Identify LED Request (in milliseconds) */
```

```
217  of a config request message */
218 #define CONFIG_POLLING_INTERVAL      100
219 /*! PAN Advertisement Solicit trickle timer duration in milliseconds */
220 #define CONFIG_PAN_ADVERT_SOLICIT_CLK_DURATION    6000
221 /*! PAN Config Solicit trickle timer duration in milliseconds */
222 #define CONFIG_PAN_CONFIG_SOLICIT_CLK_DURATION    6000
223 /*! Default Reporting Interval - in milliseconds. It will get updated upon
224  reception of a config request message */
225 #define CONFIG_REPORTING_INTERVAL  500
226 #else
```

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

Ali Asadi
Github root directory: https://github.com/aliasadivet/Lab_Submition-.git

Screenshot of updated temperature:



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

Ali Asadi
Github root directory: https://github.com/aliasadivet/Lab_Submition-.git

Part 2 (External Sensor _ si7021)

Youtube Link: T1: https://youtu.be/LzM6Ay7mSbU
T3: https://youtu.be/-BehxORYFmg

Task 01:

Modified code:

Temperature.c

```
*  ======== temperature.c ========
 */
#include <stdint.h>
#include <stdio.h>
#include <stddef.h>
#include <unistd.h>
#include <string.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/UART.h>
#include <ti/drivers/I2C.h>

/* Example/Board Header files */
#include "Board.h"

extern void itoa(int n, char s[]);

/* ======== 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 */

/*
```

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

```
 *  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;

/* 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.
 */
```

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

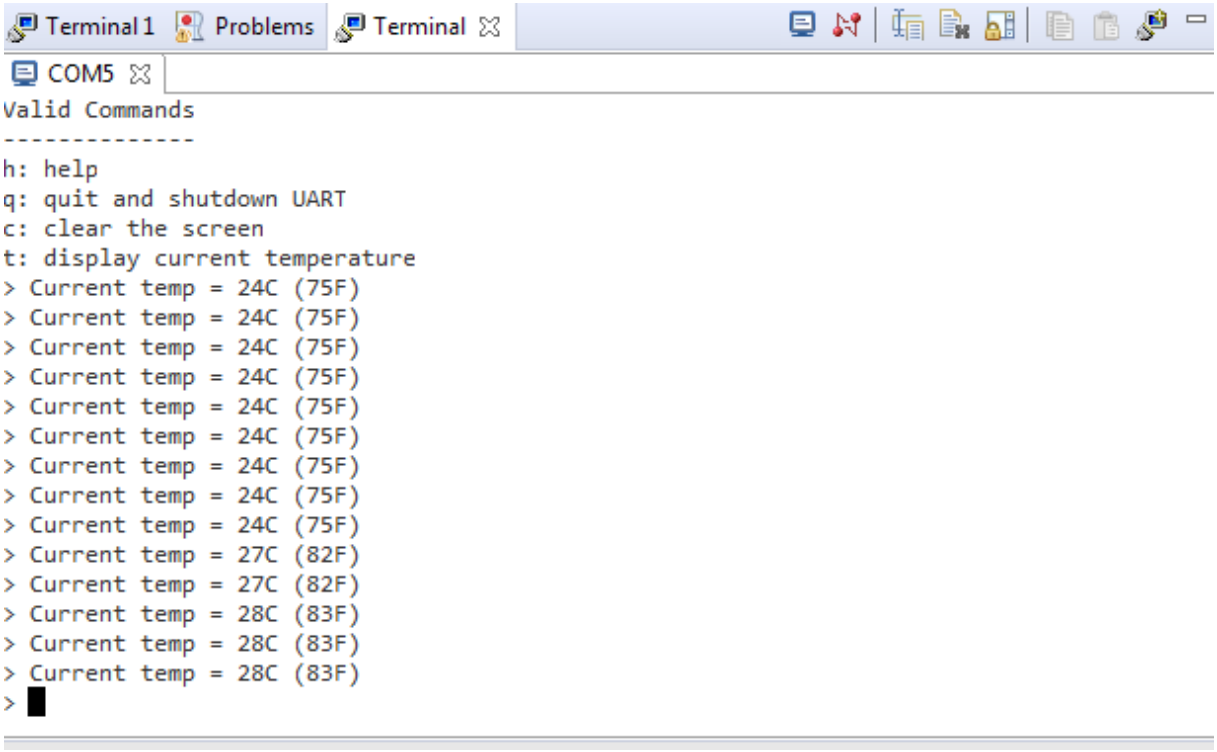
```
int setupTimer(sem_t *sem, timer_t *timerid, time_t sec, long nsec)
{
    struct sigevent   sev;
    struct itimerspec its;
    int               retc;

    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;

    /* 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
```

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

```
     */
    I2C_init();

    I2C_Params_init(&i2cParams);
    i2cParams.bitRate = I2C_100kHz;
    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;
    /* Try Si7021 */
    txBuffer[0] = Si7021_TMP_REG;
    i2cTransaction.slaveAddress = Si7021_ADDR;
    if (!I2C_transfer(i2c, &i2cTransaction)) {
    /* Could not resolve a sensor, error */
        //UART_write(0, "Error. No TMP sensor found!", strlen("Error. No TMP
sensor found!"));
        while(1);
    }
    else {
        //UART_write(0, "Detected Si7021 sensor.", strlen("Detected Si7021
sensor."));
    }

    while(1)
    {
        int     sample;
        float temperaturef = 0;
        /* Take 20 samples and print them out onto the console */
        for (sample = 0; sample < 20; sample++) {
            if (I2C_transfer(i2c, &i2cTransaction)) {
        /*
        * Extract degrees C from the received data;
        * see Si7021 datasheet
        */
                    int32_t temperature = (rxBuffer[0] << 8) | (rxBuffer[1]);
                    temperaturef += (((175.72 * temperature)/ 65536) - 46.85);
            }
        else {
            //UART_write(0, "I2C Bus fault.", strlen("I2C Bus fault."));
            }
        }
        pthread_mutex_lock(&temperatureMutex);
        temperatureC = temperaturef / 20.0 ;
        temperatureF = (temperatureC * 9.0/5.0) + 32;
        pthread_mutex_unlock(&temperatureMutex);
    }
}
```

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

Screenshot of sensor terminal



Task 02

Modified code:

`main_tirtos.c` (temperature)

```
/*
 *  ======== 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;
```

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

```
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_app(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 */
```

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

```
        while (1) {}
    }

    /* Initialize the GPIO since multiple threads are using it */
    //GPIO_init();

    /* Start the TI-RTOS scheduler */
    //BIOS_start();

    return (0);
}
```

main.c

```
/*****************************************************************************
 Includes
 *****************************************************************************/

#include <xdc/std.h>

#include <xdc/runtime/Error.h>
#include <xdc/runtime/System.h>

#include <ti/sysbios/BIOS.h>
#include <ti/sysbios/knl/Task.h>

#include <ioc.h>

#include "sys_ctrl.h"

#include "Board.h"
#include "board_led.h"
#if defined(DeviceFamily_CC13X0) || defined(DeviceFamily_CC13X2)
#include "board_gpio.h"
#endif
#include <inc/hw_ccfg.h>
#include <inc/hw_ccfg_simple_struct.h>

/* Header files required for the temporary idle task function */
#include <ti/drivers/Power.h>
#include <ti/drivers/power/PowerCC26XX.h>
#include <aon_rtc.h>
#include <prcm.h>

/* Header files required to enable instruction fetch cache */
#include <vims.h>
#include <hw_memmap.h>

#include <ti/sysbios/hal/Hwi.h>


#include "cpu.h"

#ifdef NV_RESTORE
#include "macconfig.h"
#ifdef ONE_PAGE_NV
#include "nvocop.h"
```

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

```
#else
#include "nvoctp.h"
#endif
#endif

#include <string.h>
#ifdef OSAL_PORT2TIRTOS
#include "macTask.h"
#include"rom_jt.h"
#else
#include "api_mac.h"
#include "icall.h"
#endif

#include "ssf.h"
#include "uart_printf.h"

#include "sensor.h"
#if defined(ASSERT_LEDS)
#include "board_led.h"
#endif

#ifndef USE_DEFAULT_USER_CFG

#include "mac_user_config.h"

/* MAC user defined configuration */
macUserCfg_t macUser0Cfg[] = MAC_USER_CFG;

#endif /* USE_DEFAULT_USER_CFG */

/*********************************************************************
 Constants
 *********************************************************************/

/* Assert Reasons */
#define MAIN_ASSERT_ICALL        2
#define MAIN_ASSERT_MAC          3
#define MAIN_ASSERT_HWI_TIRTOS   4

#define MAX_ASSERT_TOGGLE_COUNT  500000

#define RFC_MODE_BLE                 PRCM_RFCMODESEL_CURR_MODE1
#define RFC_MODE_IEEE                PRCM_RFCMODESEL_CURR_MODE2
#define RFC_MODE_ANT                 PRCM_RFCMODESEL_CURR_MODE4
#define RFC_MODE_EVERYTHING_BUT_ANT  PRCM_RFCMODESEL_CURR_MODE5
#define RFC_MODE_EVERYTHING          PRCM_RFCMODESEL_CURR_MODE6

/* Extended Address offset in FCFG (LSB..MSB) */
#define EXTADDR_OFFSET 0x2F0

#define APP_TASK_PRIORITY   1
#if defined(DeviceFamily_CC13X2) || (DeviceFamily_CC26X2)
#define APP_TASK_STACK_SIZE 1536
#else
#define APP_TASK_STACK_SIZE 900
#endif

#define SET_RFC_MODE(mode) HWREG( PRCM_BASE + PRCM_O_RFCMODESEL ) = (mode)
```

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

```
/*********************************************************************
 External Variables
 *********************************************************************/

extern ApiMac_sAddrExt_t ApiMac_extAddr;
extern int main_app();
/*********************************************************************
 Global Variables
 *********************************************************************/
Task_Struct appTask;        /* not static so you can see in ROV */
static uint8_t appTaskStack[APP_TASK_STACK_SIZE];

#ifdef OSAL_PORT2TIRTOS
static uint8_t _macTaskId;
#endif

#ifdef NV_RESTORE
mac_Config_t Main_user1Cfg = { 0 };
#endif

#if defined(BOARD_DISPLAY_USE_UART)
UART_Params uartParams;
#endif

/*********************************************************************
 Local Variables
 *********************************************************************/
/* Used to check for a valid extended address */
static const uint8_t dummyExtAddr[] =
    { 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF };

#ifdef NV_RESTORE
#ifdef ONE_PAGE_NV
/* NVOCOP load API pointers */
static void NVOCOP_loadApiPtrs(NVINTF_nvFuncts_t *pfn)
{
    // Load caller's structure with pointers to the NV API functions
    pfn->initNV      = &NVOCOP_initNV;
    pfn->compactNV   = &NVOCOP_compactNV;
    pfn->createItem  = NULL;
    pfn->deleteItem  = &NVOCOP_deleteItem;
    pfn->readItem    = &NVOCOP_readItem;
    pfn->writeItem   = &NVOCOP_writeItem;
    pfn->writeItemEx = NULL;
    pfn->getItemLen  = NULL;
}
#endif
#endif

/*!
 * @brief       Fill in your own assert function.
 *
 * @param       assertReason - reason: MAIN_ASSERT_HWI_TIRTOS,
 *                                     MAIN_ASSERT_ICALL, or
 *                                     MAIN_ASSERT_MAC
 */
void Main_assertHandler(uint8_t assertReason)
{
```

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

```
#if defined(ASSERT_LEDS)
    int toggleCount = 0;
    bool toggle = true;

    Hwi_disable();

    while(1)
    {
        if(toggleCount == 0)
        {
            if(toggle == false)
            {
                Board_Led_control(board_led_type_LED1, board_led_state_OFF);
                Board_Led_control(board_led_type_LED2, board_led_state_OFF);
#if !defined(DeviceFamily_CC13X0) && !defined(DeviceFamily_CC26X0) &&
!defined(DeviceFamily_CC13X2) && !defined(DeviceFamily_CC26X2)
                Board_Led_control(board_led_type_LED3, board_led_state_OFF);
                Board_Led_control(board_led_type_LED4, board_led_state_OFF);
#endif
            }
            else if(toggle == true)
            {
                Board_Led_control(board_led_type_LED1, board_led_state_ON);
                Board_Led_control(board_led_type_LED2, board_led_state_ON);
#if !defined(DeviceFamily_CC13X0) && !defined(DeviceFamily_CC26X0) &&
!defined(DeviceFamily_CC13X2) && !defined(DeviceFamily_CC26X2)
                Board_Led_control(board_led_type_LED3, board_led_state_ON);
                Board_Led_control(board_led_type_LED4, board_led_state_ON);
#endif
            }
        }

        toggleCount++;
        if(toggleCount >= MAX_ASSERT_TOGGLE_COUNT)
        {
            toggleCount = 0;
            if(toggle == true)
            {
                toggle = false;
            }
            else
            {
                toggle = true;
            }
        }
    }
#else  /* ASSERT_LEDS */
    Ssf_assertInd(assertReason);

    /* Pull the plug and start over */
    SysCtrlSystemReset();
#endif /* !ASSERT_LEDS */
}


/*!
 * @brief       Main task function
 *
 * @param       a0 -
```

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

```
 * @param       a1 -
 */
Void appTaskFxn(UArg a0, UArg a1)
{
#ifdef TIMAC_AGAMA_FPGA
    /* FPGA build disables POWER constraints */
    Power_setConstraint(PowerCC26XX_IDLE_PD_DISALLOW);
    Power_setConstraint(PowerCC26XX_SB_DISALLOW);

    IOCPortConfigureSet(IOID_20, IOC_PORT_RFC_GPO0, IOC_STD_OUTPUT);
    IOCPortConfigureSet(IOID_18, IOC_PORT_RFC_GPI0, IOC_STD_INPUT);
    // configure RF Core SMI Command Link
    IOCPortConfigureSet(IOID_22, IOC_IOCFG0_PORT_ID_RFC_SMI_CL_OUT,
IOC_STD_OUTPUT);
    IOCPortConfigureSet(IOID_21, IOC_IOCFG0_PORT_ID_RFC_SMI_CL_IN, IOC_STD_INPUT);

#endif

#ifndef OSAL_PORT2TIRTOS
    /* Initialize ICall module */
    ICall_init();
#endif
    /* Copy the extended address from the CCFG area */
    memcpy(ApiMac_extAddr, (uint8_t *)&(__ccfg.CCFG_IEEE_MAC_0),
           (APIMAC_SADDR_EXT_LEN / 2));
    memcpy(ApiMac_extAddr + (APIMAC_SADDR_EXT_LEN / 2), (uint8_t
*)&(__ccfg.CCFG_IEEE_MAC_1),
           (APIMAC_SADDR_EXT_LEN / 2));

    /* Check to see if the CCFG IEEE is valid */
    if(memcmp(ApiMac_extAddr, dummyExtAddr, APIMAC_SADDR_EXT_LEN) == 0)
    {
        /* No, it isn't valid.  Get the Primary IEEE Address */
        memcpy(ApiMac_extAddr, (uint8_t *)(FCFG1_BASE + EXTADDR_OFFSET),
               (APIMAC_SADDR_EXT_LEN));
    }

#ifdef NV_RESTORE
    /* Setup the NV driver */
#ifdef ONE_PAGE_NV
    NVOCOP_loadApiPtrs(&Main_user1Cfg.nvFps);
#else
    NVOCTP_loadApiPtrs(&Main_user1Cfg.nvFps);
#endif

    if(Main_user1Cfg.nvFps.initNV)
    {
        Main_user1Cfg.nvFps.initNV( NULL);
    }
#endif

    /* Initialize the application */
#ifdef OSAL_PORT2TIRTOS
    Sensor_init(_macTaskId);
#else
    ICall_createRemoteTasks();

    /* Initialize the application */
    Sensor_init();
```

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

```
#endif

#ifdef FEATURE_BLE_OAD
    uint8_t blinkCnt = 8;

    /* blink to let user know sensor is initializing */
    while(blinkCnt)
    {
        Board_Led_toggle(board_led_type_LED1);
        Task_sleep(10000);
        blinkCnt--;
    }

    /* reset led back to known state */
    Board_Led_control(board_led_type_LED1, board_led_state_OFF);
#endif /* FEATURE_BLE_OAD */

    /* Kick off application - Forever loop */
    while(1)
    {
        Sensor_process();
    }
}

/*!
 * @brief       TIRTOS HWI Handler.  The name of this function is set to
 *              M3Hwi.excHandlerFunc in app.cfg, you can disable this by
 *              setting it to null.
 *
 * @param       excStack - TIROS variable
 * @param       lr - TIROS variable
 */
xdc_Void Main_excHandler(UInt *excStack, UInt lr)
{
    /* User defined function */
    Main_assertHandler(MAIN_ASSERT_HWI_TIRTOS);
}

/*!
 * @brief       HAL assert handler required by OSAL memory module.
 */
void halAssertHandler(void)
{
    /* User defined function */
    Main_assertHandler(MAIN_ASSERT_ICALL);
}

/*!
 * @brief       MAC HAL assert handler.
 */
void macHalAssertHandler(void)
{
    /* User defined function */
    Main_assertHandler(MAIN_ASSERT_MAC);
}

/*!
 * @brief       "main()" function - starting point
 */
```

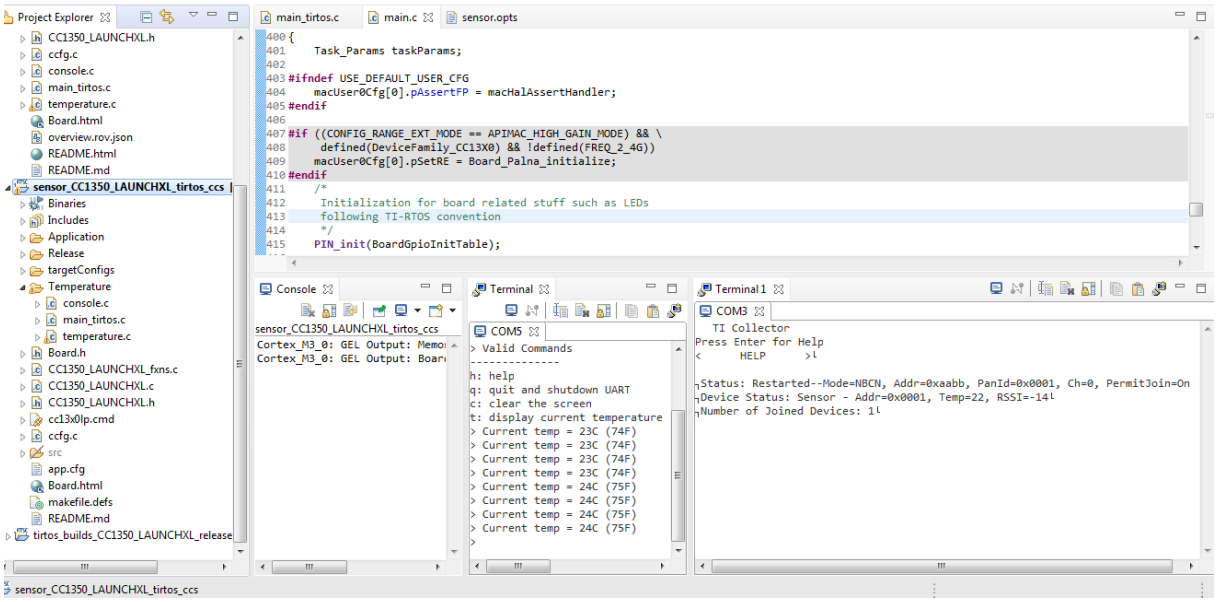
Grading scheme: 30% Coding, 30% Documentation, 40% Execution/Video.

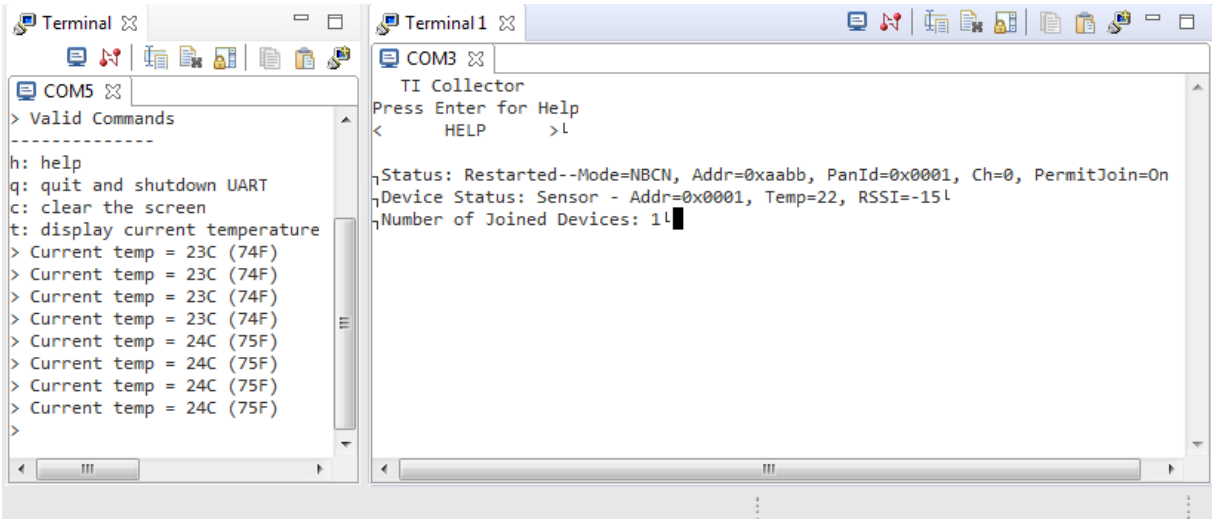
```
int main(void)
{
    Task_Params taskParams;

#ifndef USE_DEFAULT_USER_CFG
    macUser0Cfg[0].pAssertFP = macHalAssertHandler;
#endif

#if ((CONFIG_RANGE_EXT_MODE == APIMAC_HIGH_GAIN_MODE) && \
     defined(DeviceFamily_CC13X0) && !defined(FREQ_2_4G))
    macUser0Cfg[0].pSetRE = Board_Palna_initialize;
#endif
    /*
     Initialization for board related stuff such as LEDs
     following TI-RTOS convention
     */
    PIN_init(BoardGpioInitTable);

#ifdef FEATURE_BLE_OAD
    /* If FEATURE_BLE_OAD is enabled, look for a left button
     *  press on reset. This indicates to revert to some
     *  factory image
     */
    if(!PIN_getInputValue(Board_PIN_BUTTON0))
    {
        OAD_markSwitch();
    }
#endif /* FEATURE_BLE_OAD */

#if defined(POWER_MEAS)
    /* Disable external flash for power measurements */
    Board_shutDownExtFlash();
#endif

#if defined(FEATURE_BLE_OAD) || defined(FEATURE_NATIVE_OAD)
    SPI_init();
#endif

#ifndef POWER_MEAS
#if defined(BOARD_DISPLAY_USE_UART)
    /* Enable System_printf(..) UART output */
    UART_init();
    UART_Params_init(&uartParams);
#ifndef TIMAC_AGAMA_FPGA
    uartParams.baudRate = 115200;
#else
    uartParams.baudRate = 460800;
#endif
    UartPrintf_init(UART_open(Board_UART0, &uartParams));
#endif /* BOARD_DISPLAY_USE_UART */
#endif

#ifdef OSAL_PORT2TIRTOS
    _macTaskId = macTaskInit(macUser0Cfg);
#endif

    /* Configure task. */
    Task_Params_init(&taskParams);
    taskParams.stack = appTaskStack;
```

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

```
    taskParams.stackSize = APP_TASK_STACK_SIZE;
    taskParams.priority = APP_TASK_PRIORITY;
    Task_construct(&appTask, appTaskFxn, &taskParams, NULL);

#ifdef DEBUG_SW_TRACE
    IOCPortConfigureSet(IOID_8, IOC_PORT_RFC_TRC, IOC_STD_OUTPUT
                        | IOC_CURRENT_4MA | IOC_SLEW_ENABLE);
#endif /* DEBUG_SW_TRACE */
    main_app();
    BIOS_start(); /* enable interrupts and start SYS/BIOS */

    return (0);
}
```

Screensots of collected temperature:





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

Ali Asadi
Github root directory: https://github.com/aliasadivet/Lab_Submition-.git

Task 03

Modified codes:

temperature.c

```
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)temperatureC;
                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));
                tempSensor.objectTemp = localTemperatureC;
                tempSensor.ambienceTemp = localTemperatureC;
                Util_setEvent(&Sensor_events, EXT_SENSOR_READING_TIMEOUT_EVT);
                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));
```

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

```
                break;
            }
        }
    }
```

sensor.c

```
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)
```

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

```
            {
                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);
    }

    /* Is it time to send the next sensor data message? */
    if(Sensor_events & EXT_SENSOR_READING_TIMEOUT_EVT)
    {
     /* Process Sensor Reading Message Event */
     processSensorMsgEvt();
     /* Clear the event */
     Util_clearEvent(&Sensor_events, EXT_SENSOR_READING_TIMEOUT_EVT);
    }

    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);
        }
```

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

```
#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)
```

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

```
    {
        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();
    }
}
```

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