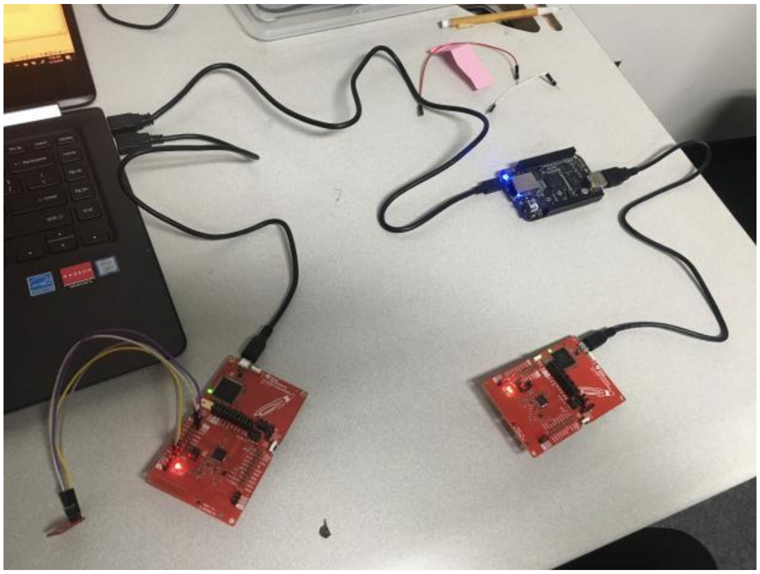
Chris Barr
Github root directory:
https://github.com/BarrChris/submissions_da_F19

Partner: Darryl Derico

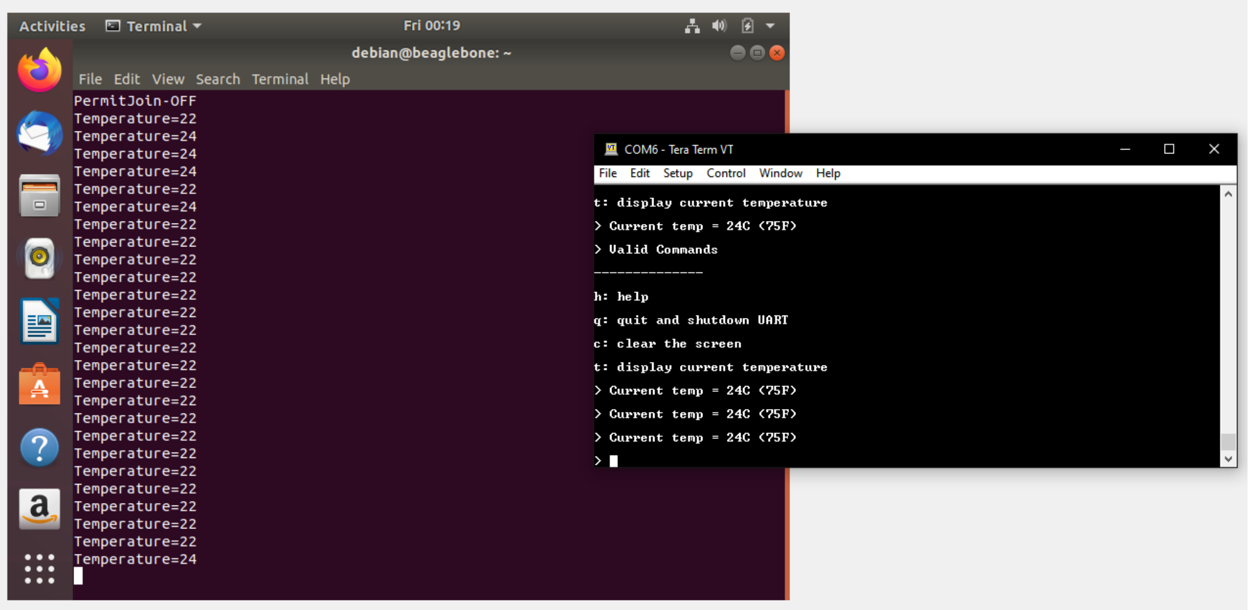
Video Link: https://www.youtube.com/watch?v=HAaFMS9DRKM

Code Editor/Compiler: Code Composer Studio
Serial Connection: PuTTy, Tera Term
Software: VM Virtual Box, Ubuntu

Final Setup



Final Result


Activities
Terminal
Fri 00:19
debian@beaglebone: ~
File Edit View Search Terminal Help
COM6 - Tera Term VT
File Edit Setup Control Window Help

```
PermitJoin-OFF
Temperature=22
Temperature=24
Temperature=24
Temperature=24
Temperature=22
Temperature=24
Temperature=22
Temperature=22
Temperature=22
Temperature=22
Temperature=22
Temperature=22
Temperature=22
Temperature=22
Temperature=22
Temperature=22
Temperature=22
Temperature=22
Temperature=22
Temperature=22
Temperature=22
Temperature=22
Temperature=22
Temperature=22
Temperature=22
Temperature=24
```

```
t: display current temperature
> Current temp = 24C (75F)
> Valid Commands
--------------
h: help
q: quit and shutdown UART
c: clear the screen
t: display current temperature
> Current temp = 24C (75F)
> Current temp = 24C (75F)
> Current temp = 24C (75F)
>
```

Chris Barr
Github root directory:
https://github.com/BarrChris/submissions_da_F19

Code:

```
/*
 *  ======== console.c ========
 */
#include <stdint.h>
#include <string.h>
#include <stdbool.h>

/* POSIX Header files */
#include <pthread.h>
#include <semaphore.h>

/* Driver Header files */
#include "smsgs.h"
#include "mac_util.h"
#include "api_mac.h"
#include "sensor.h"
extern Smsgs_tempSensorField_t tempSensor;

#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':
                tempSensor.objectTemp = localTemperatureC;
                tempSensor.ambienceTemp = localTemperatureC;
                Util_setEvent(&Sensor_events,
EXT_SENSOR_READING_TIMEOUT_EVT);
```

```
                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);
}


/*
 *  ======== 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"
```

Chris Barr
Github root directory:
https://github.com/BarrChris/submissions_da_F19

```
/* ======== 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);
//            }
//    }

        //------------------------------------------------------------------
--------
        /*
        // 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
             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 < 20; sample++)
         {
             if (I2C_transfer(i2c, &i2cTransaction))
             {
                 //
                 // Extract degrees C from the received data;
                 // see Si7021 datasheet
                 //
                 temperature = (rxBuffer[0] << 8) | (rxBuffer[1]);
                 temperaturef = (((175.72 * temperature)/ 65536) - 46.85);
                 Display_printf(display, 0, 0, "Sample %u: %d (C)", sample,
temperaturef);
             }

             else
             {
                 Display_printf(display, 0, 0, "I2C Bus fault.");
             }
         }
         */

         //------------------------------------------------------------------
---------

//        /* Block until the timer posts the semaphore. */
//        retc = sem_wait(&semTimer);
//        if (retc == -1) {
//            while (1);
//        }
//    }
}


/*
 *  ======== main_tirtos.c ========
 */
#include <stdint.h>

/* POSIX Header files */
#include <pthread.h>
```

Chris Barr
Github root directory:
https://github.com/BarrChris/submissions_da_F19

```
/* 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_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 */
        while (1) {}
    }

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

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

    return (0);
}
```