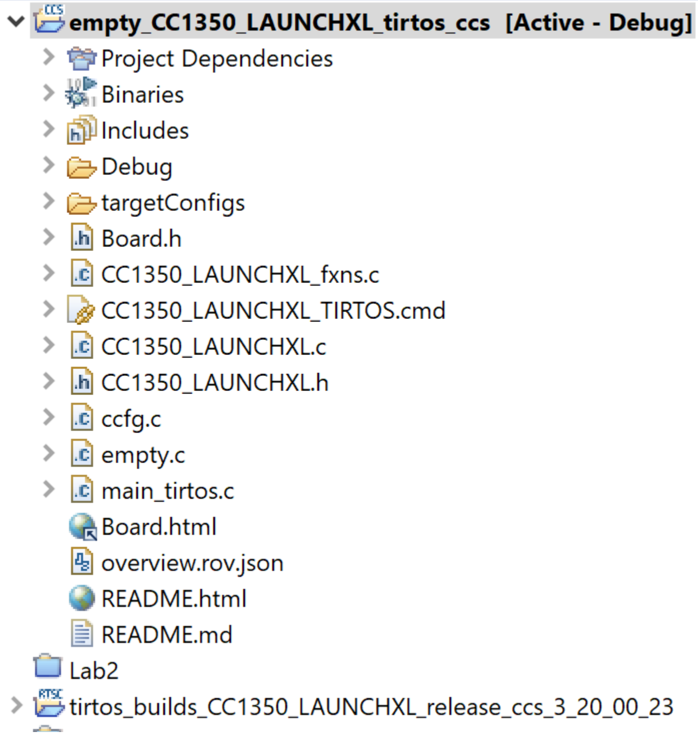
Tenniel Takenaka-Fuller
Github root directory: https://github.com/TennielTakenaka/sturdy-carnival/tree/master/Lab%204

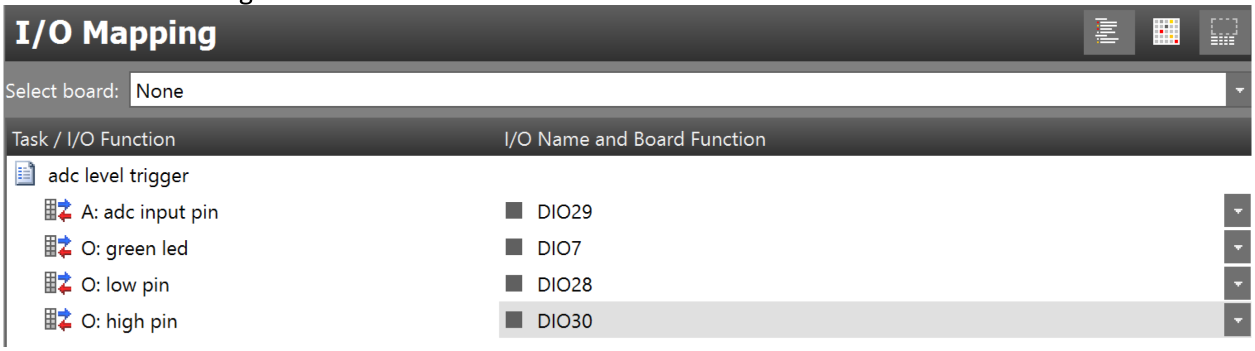
Date Submitted: **11/21/19**

Task 1: installed empty file & red light is flashing on tiva c



```
empty_CC1350_LAUNCHXL_tirtos_ccs  [Active - Debug]
    Project Dependencies
    Binaries
    Includes
    Debug
    targetConfigs
    Board.h
    CC1350_LAUNCHXL_fxns.c
    CC1350_LAUNCHXL_TIRTOS.cmd
    CC1350_LAUNCHXL.c
    CC1350_LAUNCHXL.h
    ccfg.c
    empty.c
    main_tirtos.c
    Board.html
    overview.rov.json
    README.html
    README.md
Lab2
tirtos_builds_CC1350_LAUNCHXL_release_ccs_3_20_00_23
```

Task 2: Pin assignments in sensor control studio



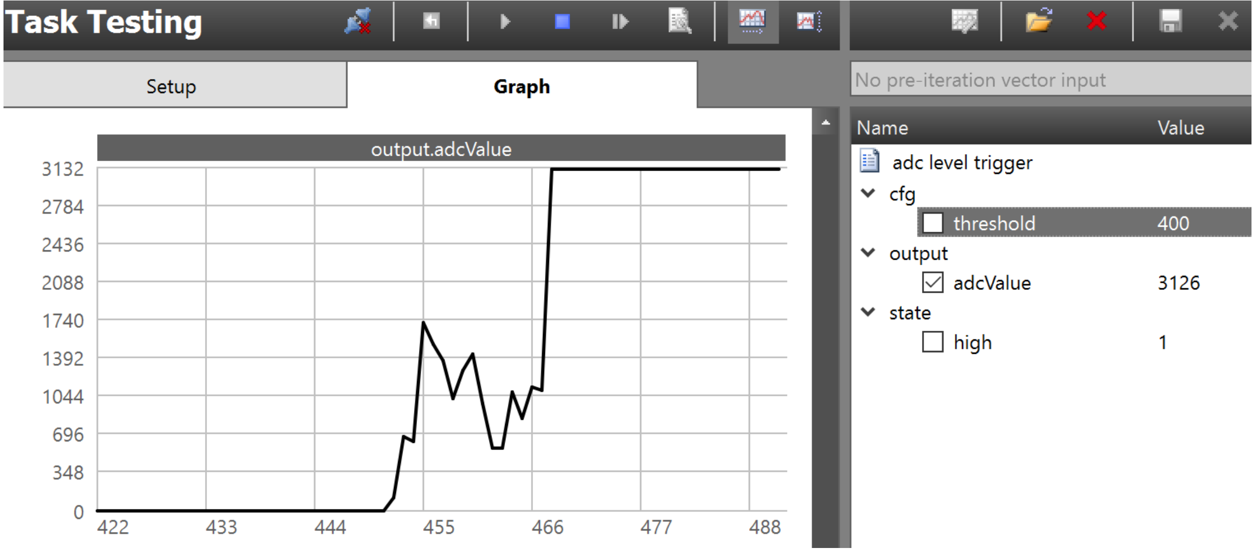
I/O Mapping

Select board: None

Task / I/O Function	I/O Name and Board Function
adc level trigger	
A: adc input pin	DIO29
O: green led	DIO7
O: low pin	DIO28
O: high pin	DIO30

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

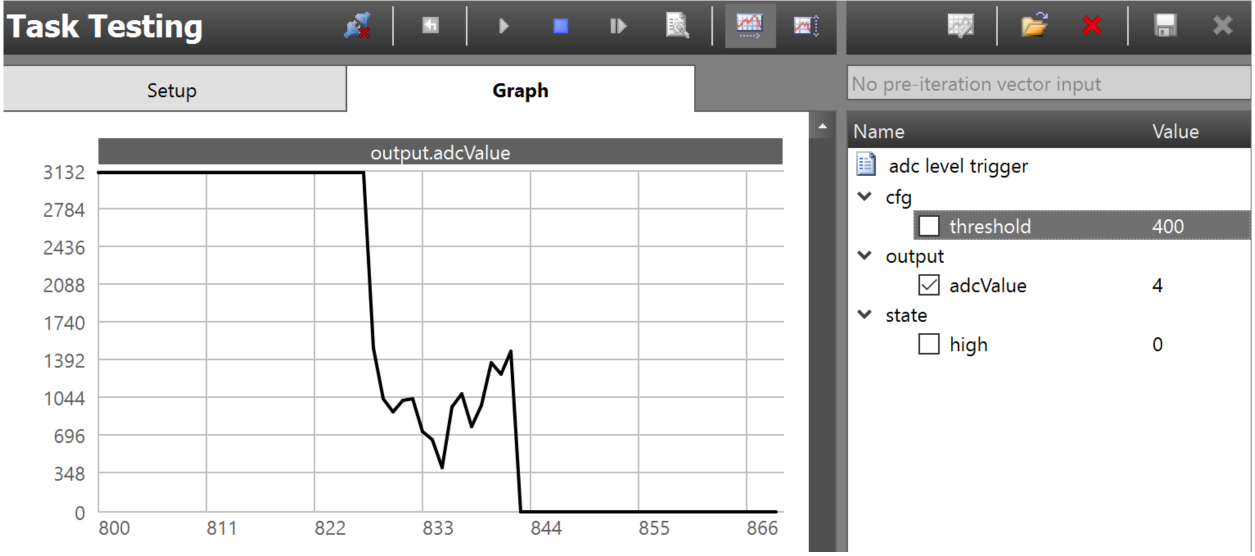
```
Task 3:
A picture of the graph when DIO29 and DIO30 are connected.
```



No pre-iteration vector input

Name	Value
adc level trigger	
cfg	
□ threshold	400
output	
☑ adcValue	3126
state	
□ high	1

```
A picture of the graph when DIO28 & DIO29 are connected. The green light also is on.
```



No pre-iteration vector input

Name	Value
adc level trigger	
cfg	
□ threshold	400
output	
☑ adcValue	4
state	
□ high	0

Task 04:

Youtube Link: https://youtu.be/HPP9XZqzmZg

Modified Code:

```
/*
 *   ======== empty.c ========
 */

/* For usleep() */
#include <unistd.h>
#include <stdint.h>
#include <stddef.h>
```

```
/* Driver Header files */
#include <ti/drivers/GPIO.h>
// #include <ti/drivers/I2C.h>
// #include <ti/drivers/SPI.h>
// #include <ti/drivers/UART.h>
// #include <ti/drivers/Watchdog.h>

/* Board Header file */
#include "Board.h"
#include "scif.h"

#define BV(x) (1 << (x))

void processTaskAlert(void) {
    //clear the alert interrupt source
    scifClearAlertIntSource();

    //do sc task processing here
    uint8_t high = scifTaskData.adcLevelTrigger.state.high; //fetch "state.high" from
sc
    GPIO_write(Board_GPIO_RLED, high); //set red led state equal to the state.high
variable

    //acknowledge the ALERT event
    scifAckAlertEvents();
}//processTaskAlert

void scCtrlReadyCallback(void) {

} //scCtrlReadyCallback

void scTaskAlertCallback(void) {

} //scTaskAlertCallback

/*
 *  ======== mainThread ========
 */
void *tirtosScThread(void *arg0)
{
    /* 1 second delay */
     // uint32_t time = 1;
      /* Call driver init functions */
    //Initialize the Sensor Controller
    scifOsalInit();
    scifOsalRegisterCtrlReadyCallback(scCtrlReadyCallback);
    scifOsalRegisterTaskAlertCallback(scTaskAlertCallback);
    scifInit(&scifDriverSetup);

    // Set the Sensor Controller task tick interval to 1 second
    uint32_t rtc_Hz = 1;  // 1Hz RTC
    scifStartRtcTicksNow(0x00010000/ rtc_Hz);

    //configure sensor controller tasks
    scifTaskData.adcLevelTrigger.cfg.threshold = 600;

    //start sensor controller task
    scifStartTasksNbl(BV(SCIF_ADC_LEVEL_TRIGGER_TASK_ID));
      GPIO_init();
      // I2C_init();
      // SPI_init();
      // UART_init();
      // Watchdog_init();

      /* Configure the LED pin */
      GPIO_setConfig(Board_GPIO_LED0, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);

      /* Turn on user LED */
      GPIO_write(Board_GPIO_LED0, Board_GPIO_LED_ON);

      while (1) {
         // sleep(time);
         // GPIO_toggle(Board_GPIO_LED0);
      }

}
```

```
//INITIALIZATION CODE in SENSOR CONTROL STUDIO
//Set 'DIO28' High
gpioSetOutput(AUXIO_O_HIGH);

//Set 'DIO30' Low
gpioClearOutput(AUXIO_O_LOW);

//Set ADC input
adcSelectGpioInput(AUXIO_A_ADC_INPUT);

//Schedule the first execution
fwScheduleTask(1);
```

```
//EXECUTION CODE IN SENSOR CONTROL STUDIO
//Enable the ADC
adcEnableSync(ADC_REF_FIXED, ADC_SAMPLE_TIME_2P7_US, ADC_TRIGGER_MANUAL);

//Sample the analog sensor
adcGenManualTrigger();
adcReadFifo(output.adcValue);

//Disable the ADC
adcDisable();

U16 oldState = state.high;

if(output.adcValue > cfg.threshold) {
    state.high = 1; //High input -> High state
    gpioClearOutput(AUXIO_O_GREEN_LED);
```

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

```
} else {
    state.high=0; //Low input->low state
    gpioSetOutput(AUXIO_O_GREEN_LED);
}

if(oldState!=state.high) {
    //signal the application processor
    fwGenAlertInterrupt();
}

//Schedule the next execution
fwScheduleTask(1);


------------------------------------------------------------------------------------
```

Task 05:

rfPacketTx.c

```
/***** Includes *****/
/* Standard C Libraries */
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <ti/sysbios/knl/Semaphore.h>


/* TI Drivers */
#include <ti/drivers/rf/RF.h>
#include <ti/drivers/PIN.h>
#include <ti/drivers/pin/PINCC26XX.h>

/* Driverlib Header files */
#include DeviceFamily_constructPath(driverlib/rf_prop_mailbox.h)

/* Board Header files */
#include "Board.h"
#include "smartrf_settings/smartrf_settings.h"
#include "scif.h"

#define BV(x) (1 << (x))

Semaphore_Struct semMainLoop;
Semaphore_Handle hSemMainLoop;
/***** Defines *****/

/* Do power measurement */
//#define POWER_MEASUREMENT

/* Packet TX Configuration */
#define PAYLOAD_LENGTH      30
#ifdef POWER_MEASUREMENT
#define PACKET_INTERVAL     5  /* For power measurement set packet interval to 5s */
#else
```

```
#define PACKET_INTERVAL     500000  /* Set packet interval to 500000us or 500ms */
#endif

/***** Prototypes *****/

/***** Variable declarations *****/
static RF_Object rfObject;
static RF_Handle rfHandle;

/* Pin driver handle */
static PIN_Handle ledPinHandle;
static PIN_State ledPinState;

static uint8_t packet[PAYLOAD_LENGTH];
static uint16_t seqNumber;

/*
 * Application LED pin configuration table:
 *   - All LEDs board LEDs are off.
 */

PIN_Config pinTable[] =
{
    Board_PIN_LED0 | PIN_GPIO_OUTPUT_EN | PIN_GPIO_LOW | PIN_PUSHPULL |
PIN_DRVSTR_MAX,
#ifdef POWER_MEASUREMENT
#if defined(Board_CC1350_LAUNCHXL)
    Board_DIO30_SWPWR | PIN_GPIO_OUTPUT_EN | PIN_GPIO_HIGH | PIN_PUSHPULL |
PIN_DRVSTR_MAX,
#endif
#endif
    PIN_TERMINATE
};

void scCtrlReadyCallback(void){
    //do nothing
} //scCtrlReadyCallback

void scTaskAlertCallback(void) {
    //signal main loop
    Semaphore_post(hSemMainLoop);
}//scTaskAlertCallback

void TxTask_init(void) {
    //main loop semaphore init
    Semaphore_Params semParams;
    Semaphore_Params_init(&semParams);
    semParams.mode = Semaphore_Mode_BINARY;
    Semaphore_construct(&semMainLoop, 0, &semParams);
    hSemMainLoop = Semaphore_handle(&semMainLoop);
}


/***** Function definitions *****/
```

```
void *mainThread(void *arg0)
{
    RF_Params rfParams;
    RF_Params_init(&rfParams);

    /* Open LED pins */
    ledPinHandle = PIN_open(&ledPinState, pinTable);
    if (ledPinHandle == NULL)
    {
        while(1);
    }

#ifdef POWER_MEASUREMENT
#if defined(Board_CC1350_LAUNCHXL)
    /* Route out PA active pin to Board_DIO30_SWPWR */
    PINCC26XX_setMux(ledPinHandle, Board_DIO30_SWPWR, PINCC26XX_MUX_RFC_GPO1);
#endif
#endif

    RF_cmdPropTx.pktLen = PAYLOAD_LENGTH;
    RF_cmdPropTx.pPkt = packet;
    RF_cmdPropTx.startTrigger.triggerType = TRIG_NOW;

    /* Request access to the radio */
#if defined(DeviceFamily_CC26X0R2)
    rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioSetup,
&rfParams);
#else
    rfHandle = RF_open(&rfObject, &RF_prop, (RF_RadioSetup*)&RF_cmdPropRadioDivSetup,
&rfParams);
#endif// DeviceFamily_CC26X0R2

    /* Set the frequency */
    RF_postCmd(rfHandle, (RF_Op*)&RF_cmdFs, RF_PriorityNormal, NULL, 0);

    while(1)
    {
        /* Create packet with incrementing sequence number and random payload */
        packet[0] = (uint8_t)(seqNumber >> 8);
        packet[1] = (uint8_t)(seqNumber++);
        uint8_t i;
        for (i = 2; i < PAYLOAD_LENGTH; i++)
        {
            packet[i] = rand();
        }

        /* Send packet */
        RF_EventMask terminationReason = RF_runCmd(rfHandle, (RF_Op*)&RF_cmdPropTx,
                                                   RF_PriorityNormal, NULL, 0);

        switch(terminationReason)
        {
            case RF_EventLastCmdDone:
                // A stand-alone radio operation command or the last radio
                // operation command in a chain finished.
                break;
            case RF_EventCmdCancelled:
                // Command cancelled before it was started; it can be caused
            // by RF_cancelCmd() or RF_flushCmd().
                break;
            case RF_EventCmdAborted:
                // Abrupt command termination caused by RF_cancelCmd() or
                // RF_flushCmd().
                break;
            case RF_EventCmdStopped:
                // Graceful command termination caused by RF_cancelCmd() or
                // RF_flushCmd().
                break;
            default:
                // Uncaught error event
                while(1);
        }

        uint32_t cmdStatus = ((volatile RF_Op*)&RF_cmdPropTx)->status;
        switch(cmdStatus)
        {
            case PROP_DONE_OK:
                // Packet transmitted successfully
                break;
            case PROP_DONE_STOPPED:
                // received CMD_STOP while transmitting packet and finished
                // transmitting packet
                break;
            case PROP_DONE_ABORT:
                // Received CMD_ABORT while transmitting packet
                break;
            case PROP_ERROR_PAR:
                // Observed illegal parameter
                break;
            case PROP_ERROR_NO_SETUP:
                // Command sent without setting up the radio in a supported
                // mode using CMD_PROP_RADIO_SETUP or CMD_RADIO_SETUP
                break;
            case PROP_ERROR_NO_FS:
                // Command sent without the synthesizer being programmed
                break;
            case PROP_ERROR_TXUNF:
                // TX underflow observed during operation
                break;
            default:
                // Uncaught error event - these could come from the
                // pool of states defined in rf_mailbox.h
                while(1);
        }

#ifndef POWER_MEASUREMENT
        PIN_setOutputValue(ledPinHandle,
Board_PIN_LED1,!PIN_getOutputValue(Board_PIN_LED1));
#endif
        /* Power down the radio */
        RF_yield(rfHandle);

#ifdef POWER_MEASUREMENT
        /* Sleep for PACKET_INTERVAL s */
        sleep(PACKET_INTERVAL);
#else
        /* Sleep for PACKET_INTERVAL us */
        usleep(PACKET_INTERVAL);
#endif

    }
}
```

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