**Date Submitted: 12/11/19**

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

**Task 01:**

A screenshot of a cell phone

Description automatically generated

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

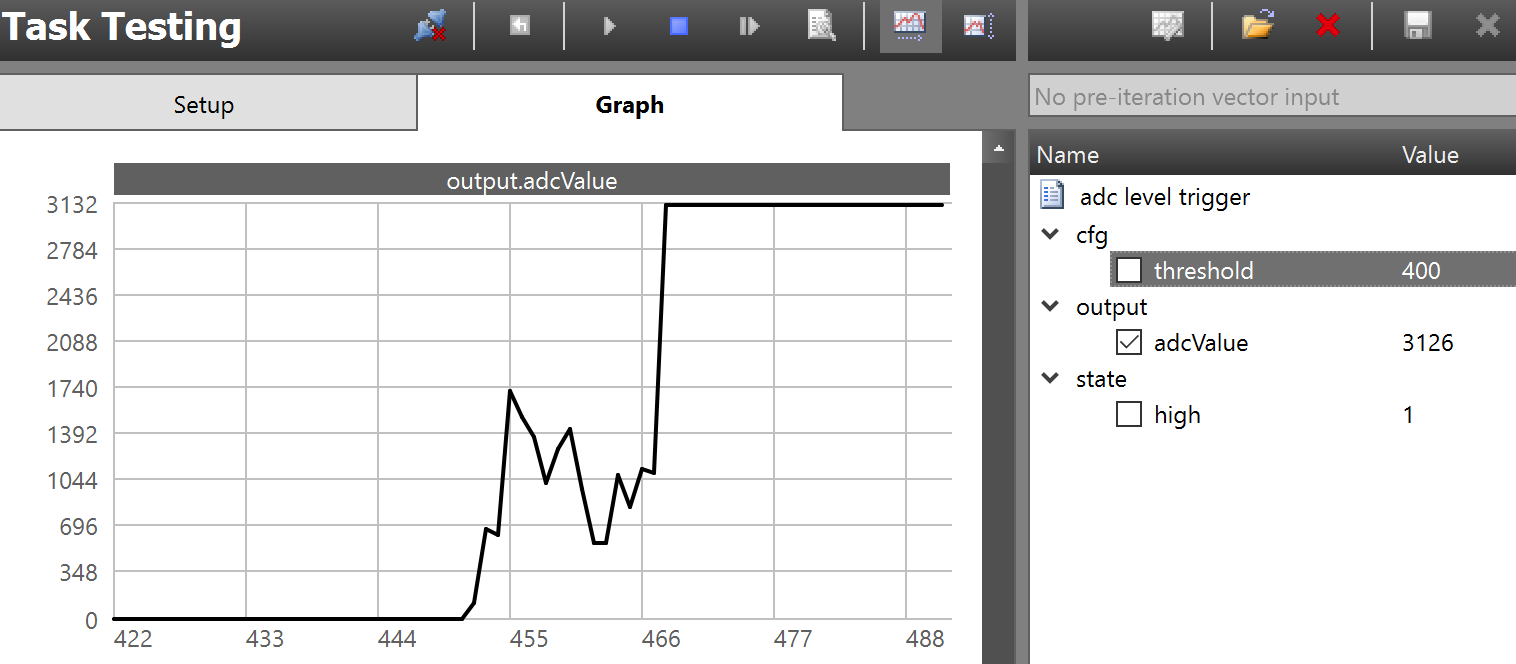
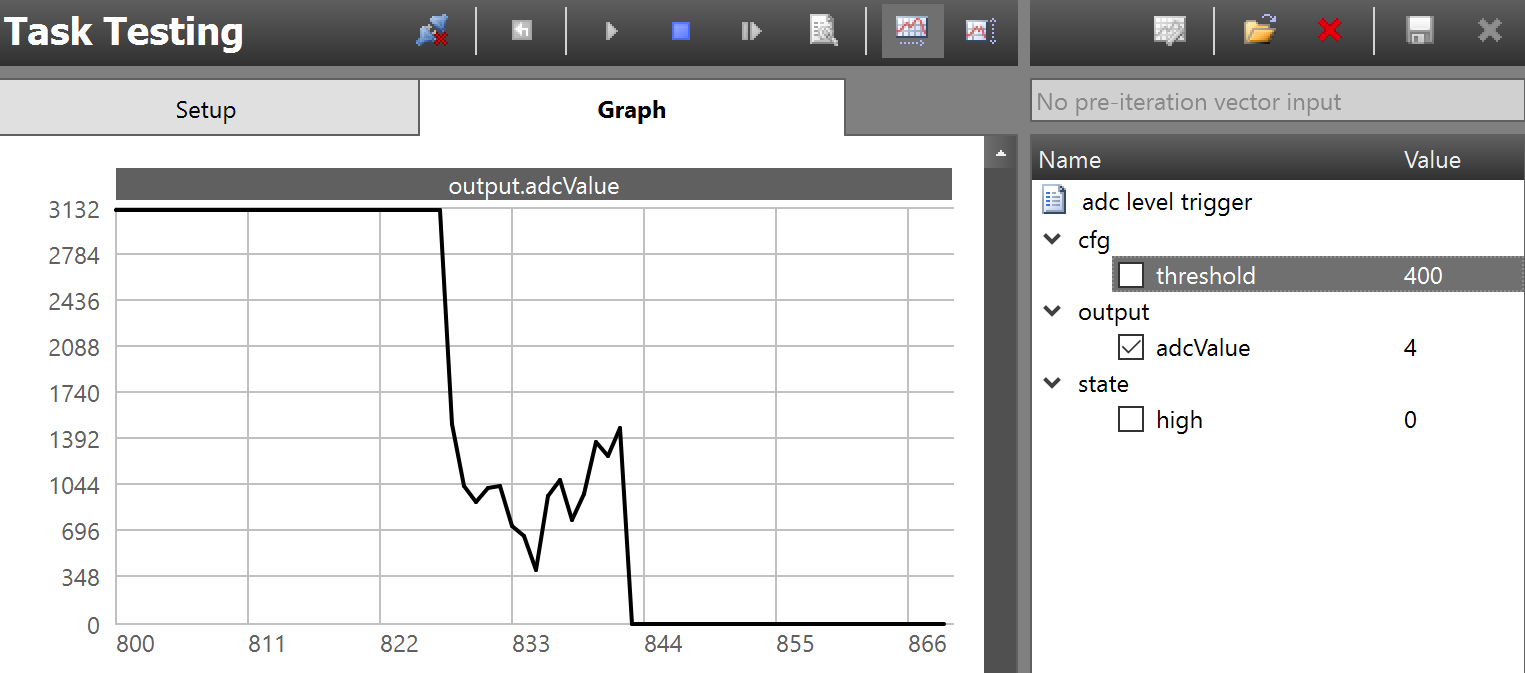
**Task 02:**

A screenshot of a cell phone

Description automatically generated

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

**Task 03:**

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

**Task 04:**

**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);**

**} 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:**

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

}

}