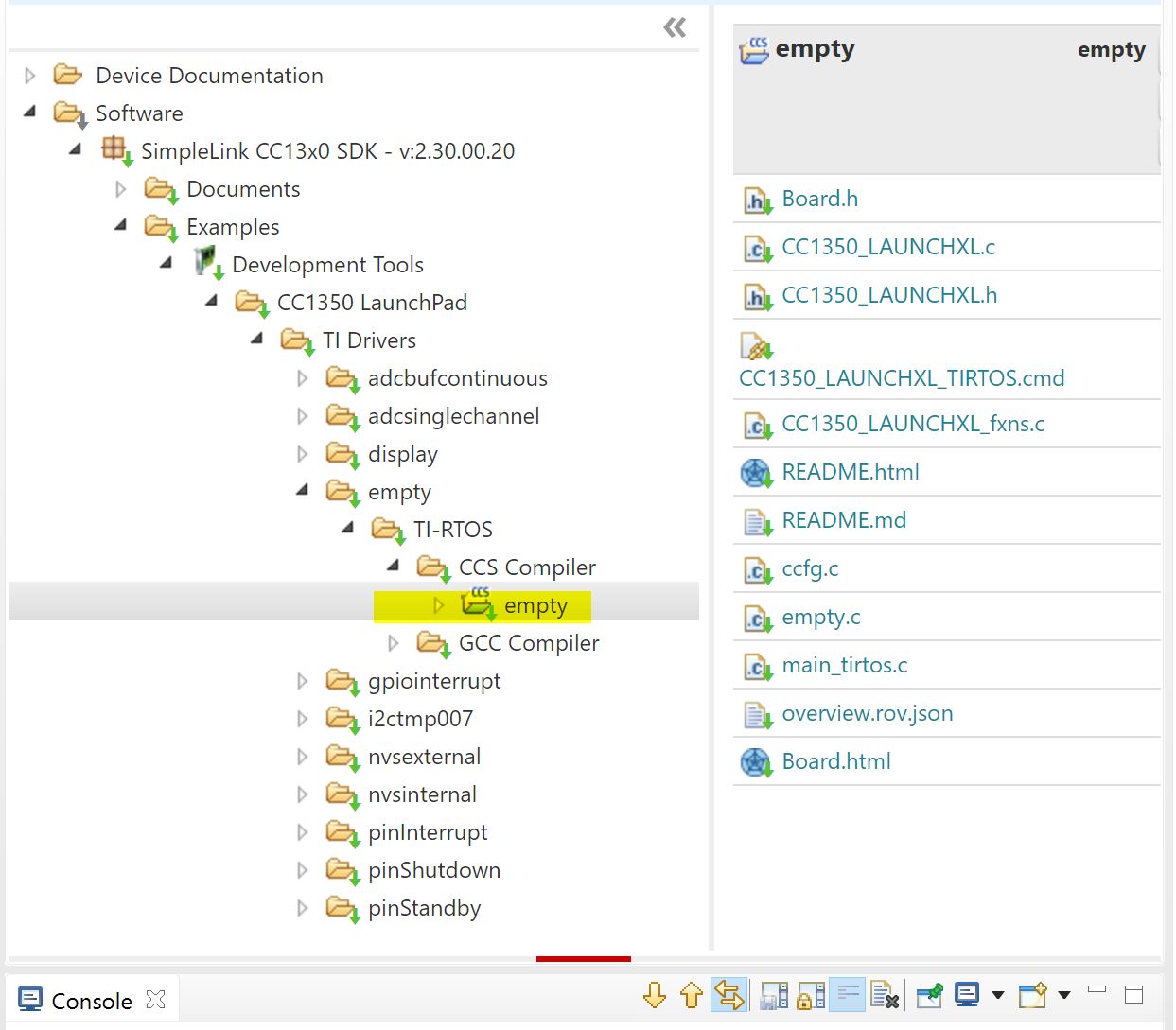
**Date Submitted: 11/16/18**

**Task 01:**

The purpose of this task is to import the example code provided within Resource Explorer. The project is called “empty” and this project is used to implement the GUI Composer on TI.



Provided Code:

/\*

\* Copyright (c) 2015-2017, 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.

\*/

/\*

\* ======== 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/SDSPI.h>

// #include <ti/drivers/SPI.h>

// #include <ti/drivers/UART.h>

// #include <ti/drivers/Watchdog.h>

/\* Board Header file \*/

#include "Board.h"

/\*

\* ======== mainThread ========

\*/

void \*mainThread(void \*arg0)

{

/\* 1 second delay \*/

uint32\_t time = 1;

/\* Call driver init functions \*/

GPIO\_init();

// I2C\_init();

// SDSPI\_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);

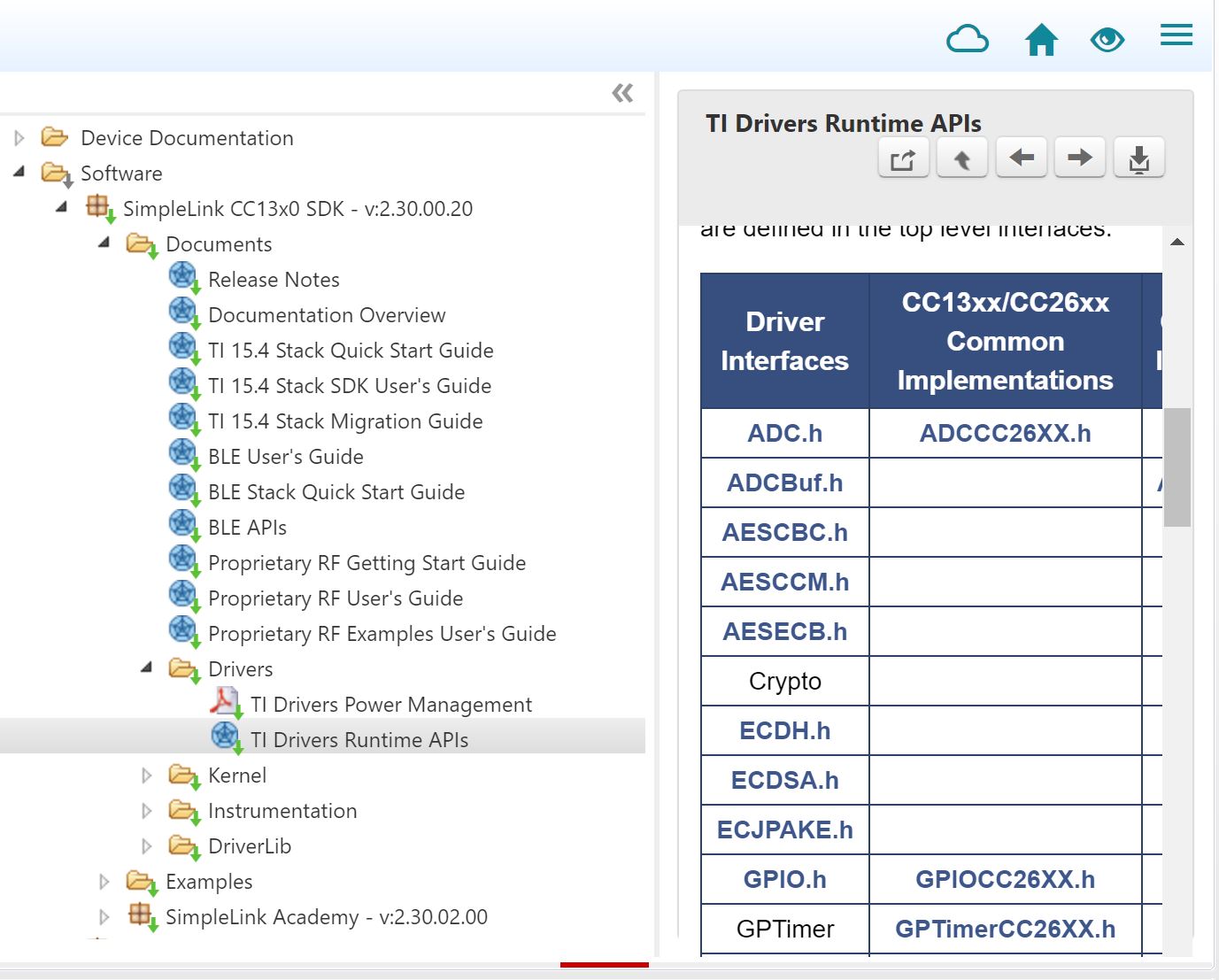
GPIO\_toggle(Board\_GPIO\_LED0);

}

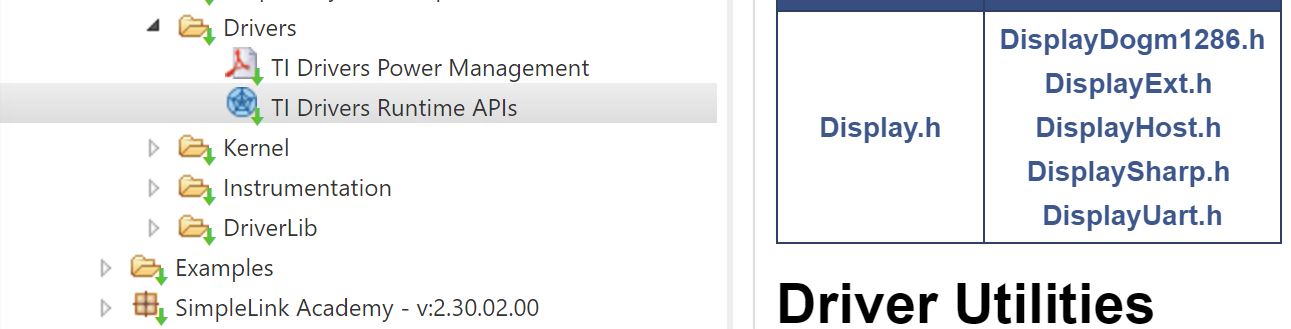
}

**Task 02, 03, and 04:**

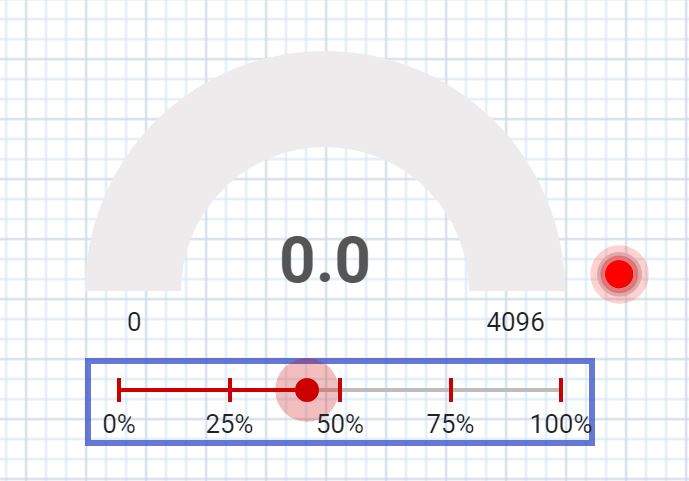
Task 02: The purpose of this task is to modify the existing project that was imported in the previous task. Its purpose is to turn on an LED only when the ADC value has met the threshold value. This can be done within the GUI Composer.



Task 03: The purpose of this task is to modify the code to add a serial UART transmission to report specific ADC readings. We can do this by adding Display.h using the API guide.



Task 04: Lastly, in this task we would like to include the GPIO interrupts to the final program. This is important in using GPIO inputs and outputs in regards to the GUI Composer.



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/ADC.h>

**#include** <ti/display/Display.h>

// #include <ti/drivers/I2C.h>

// #include <ti/drivers/SDSPI.h>

// #include <ti/drivers/SPI.h>

// #include <ti/drivers/UART.h>

// #include <ti/drivers/Watchdog.h>

/\* Board Header file \*/

**#include** "Board.h"

/\* GLOBAL VARIABLES FOR GUI COMPOSER \*/

uint16\_t adcValue = 0;

uint16\_t threshold = 100;

uint16\_t trigger = 0;

/\*

\* ======== gpioButtonFxn0 ========

\* Callback function for the GPIO interrupt on Board\_GPIO\_BUTTON0.

\*/

**void** **gpioButtonFxn0**(uint\_least8\_t index)

{

/\* Clear the GPIO interrupt and decrement threshold \*/

**if**(threshold < 250){ // Ensure threshold doesn't go below zero

threshold = 0;

} **else** {

threshold -= 250; // decrement by 250

}

}

/\*

\* ======== gpioButtonFxn1 ========

\* Callback function for the GPIO interrupt on Board\_GPIO\_BUTTON1.

\* This may not be used for all boards.

\*/

**void** **gpioButtonFxn1**(uint\_least8\_t index)

{

/\* Clear the GPIO interrupt and increment threshold \*/

**if**(threshold > 4096){ // Ensure threshold doesn't go above max ADC range

threshold = 4096;

} **else** {

threshold += 250; // increment by 250

}

}

/\*

\* ======== mainThread ========

\*/

**void** \***mainThread**(**void** \*arg0)

{

/\* ~10 loops/second \*/

uint32\_t time = 100000;

/\* Call driver init functions \*/

**GPIO\_init**();

**ADC\_init**();

// I2C\_init();

// SDSPI\_init();

// SPI\_init();

// UART\_init();

// Watchdog\_init();

/\* Open Display Driver \*/

Display\_Handle displayHandle;

Display\_Params displayParams;

Display\_Params\_init(&displayParams);

displayHandle = Display\_open(Display\_Type\_UART, NULL);

/\* Open ADC Driver \*/

ADC\_Handle adc;

ADC\_Params params;

**ADC\_Params\_init**(&params);

adc = **ADC\_open**(Board\_ADC0, &params);

**if** (adc == NULL) {

// Error initializing ADC channel 0

**while** (1);

}

/\* install Button callback \*/

**GPIO\_setCallback**(Board\_GPIO\_BUTTON0, gpioButtonFxn0);

**GPIO\_setCallback**(Board\_GPIO\_BUTTON1, gpioButtonFxn1);

/\* Enable interrupts \*/

**GPIO\_enableInt**(Board\_GPIO\_BUTTON0);

**GPIO\_enableInt**(Board\_GPIO\_BUTTON1);

**while** (1) {

int\_fast16\_t res;

res = **ADC\_convert**(adc, &adcValue);

**if** (res == ADC\_STATUS\_SUCCESS) {

Display\_printf(displayHandle, 1, 0, "ADC Reading %d", adcValue);

**if**(adcValue >= threshold){

**GPIO\_write**(Board\_GPIO\_LED0, Board\_GPIO\_LED\_ON);

trigger = 1;

} **else**{

**GPIO\_write**(Board\_GPIO\_LED0, Board\_GPIO\_LED\_OFF);

trigger = 0;

}

}

**usleep**(time);

}

}