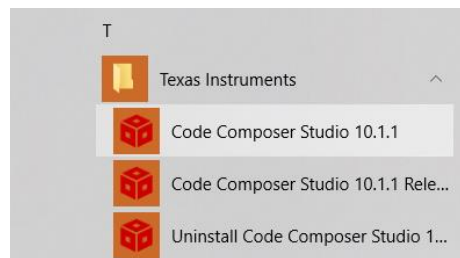




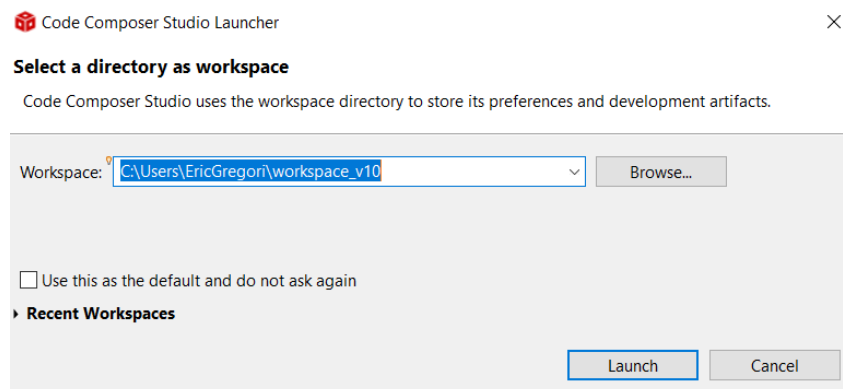
CS 350 Project Thermostat Lab Guide

It is important to note the distinct differences in the Cyber-Physical actions of a thermostat. When you select a button to increase or decrease the temperature in the room, what happens is the thermostat establishes a target temperature that is referred to as its 'set-point'. When the thermostat compares the temperature read from the sensor to the established set-point a decision is made. If the temperature is below the set-point, it turns the heater on. However, if the temperature is above the set-point, it turns the heater off.

1. Open TI Code Composer Studio (CCS). If you have not installed CCS, please install CCS by following installation guidance in Module One of your course. After CCS is installed you can open it by clicking **Code Composer Studio xxxx**, which can be found in the Texas Instruments folder. The CCS installer also added a Code Composer Studio icon on your desktop. It looks like a red die. You can click on that icon to open CCS as well.

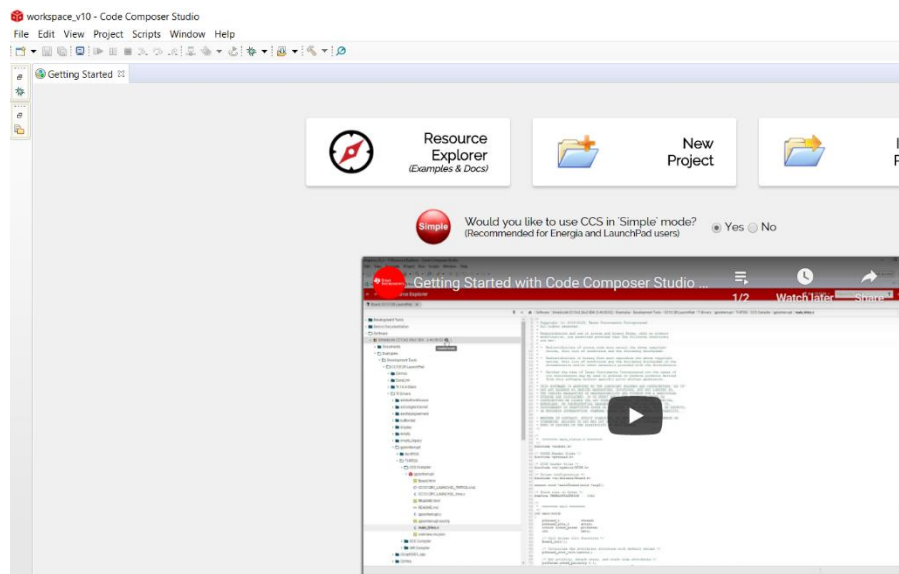
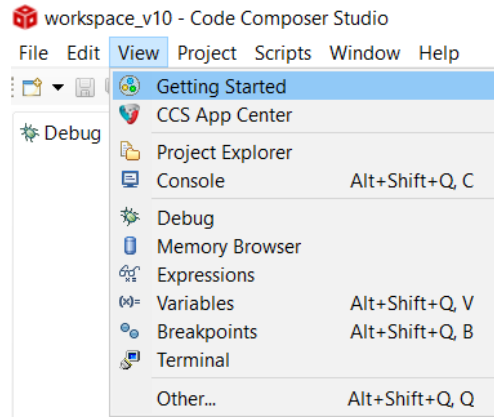


2. Select a workspace, then click **Launch**. It is very important that you take note of your workspace location. You will need the information later in order to turn in your project.

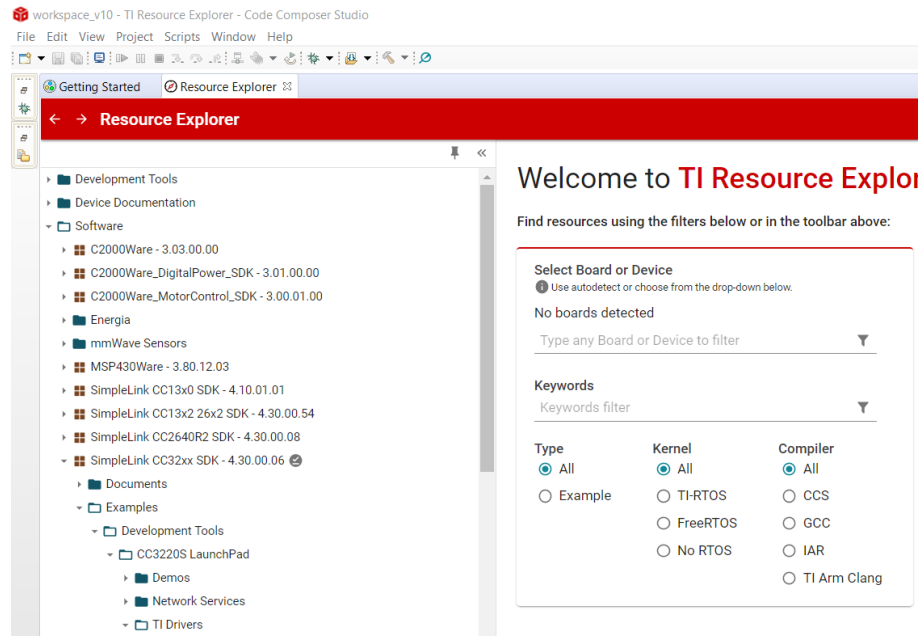






3. Open the Getting Started Window in CCS. CCS takes about 30 seconds to fully open (it's written in Java). You can review the loading process by looking at the lower right corner of the screen. You will see green progress bars popping up. After about 30 seconds, CCS should be fully loaded and you will not see any green progress bars.



4. Open the Resource Explorer by clicking on the Resource Explorer box (with the compass). In the Resource Explorer, find Software in the left pane. Click on the arrow to expand Software. Then, find the SimpleLink CC32xx SDK and expand it. Next, similarly locate and expand Examples, Development Tools, CC3220S Launchpad, and TI Drivers.

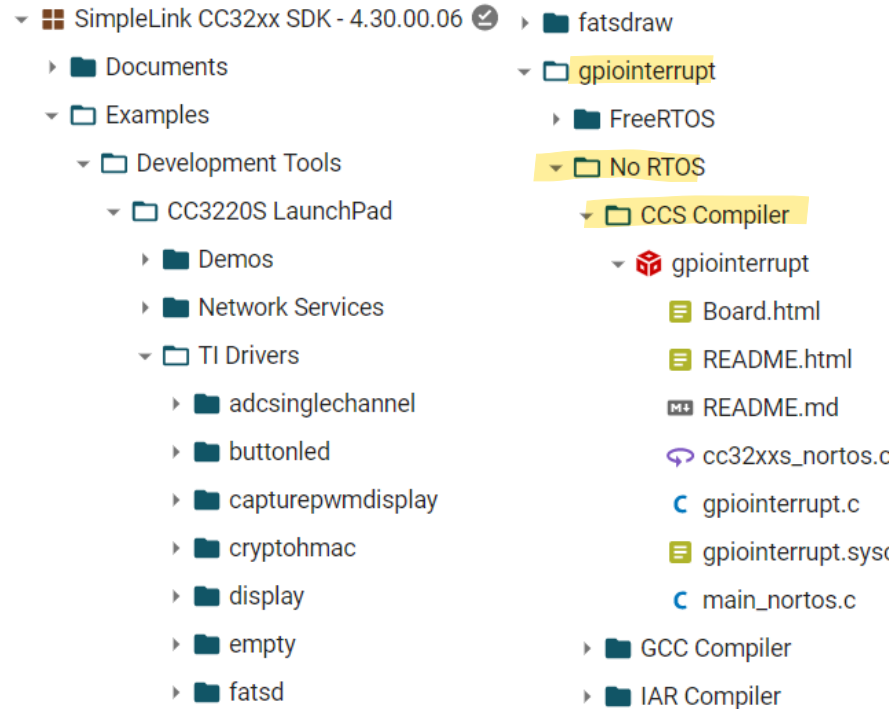


5. Verify you have the SDK installed. You should see a check mark by the SimpleLink CC32xx SDK.

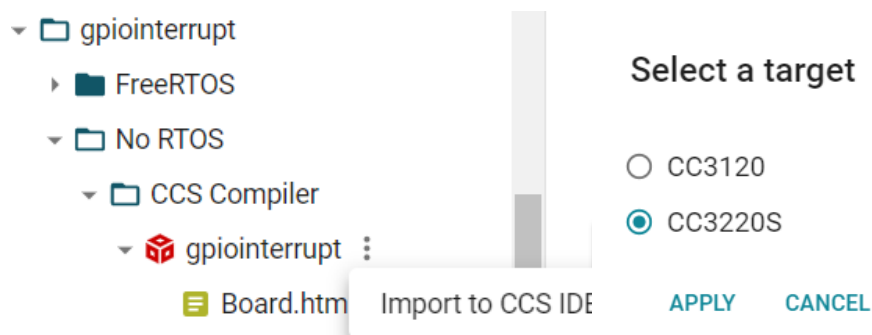
▶  SimpleLink CC32xx SDK - 4.30.00.06 

6. Import the gpiointerrupt project. You should already have imported the gpiointerrupt project during your work in a previous milestone, which would allow you to review the project from the Project Explorer window. If you still need to import the gpiointerrupt project, complete steps 7 through 9. If you already have the gpiointerrupt project, you may skip to step 10.

- Begin by expanding the SimpleLink CC32xx SDK, then the Examples folder, Development Tools, CC3220S LaunchPad, and finally TI Drivers. From here, locate the gpiointerrupt folder. Then the No RTOS sub-folder followed by the CCS Compiler. In here you will find the gpiointerrupt item illustrated with a red die icon.



- Click on the three dots icon next to gpiointerrupt then click **Import to CCS IDE**. From the Select a target window that appears, choose CC3220S and click **Apply** to load the example into the CCS workspace.



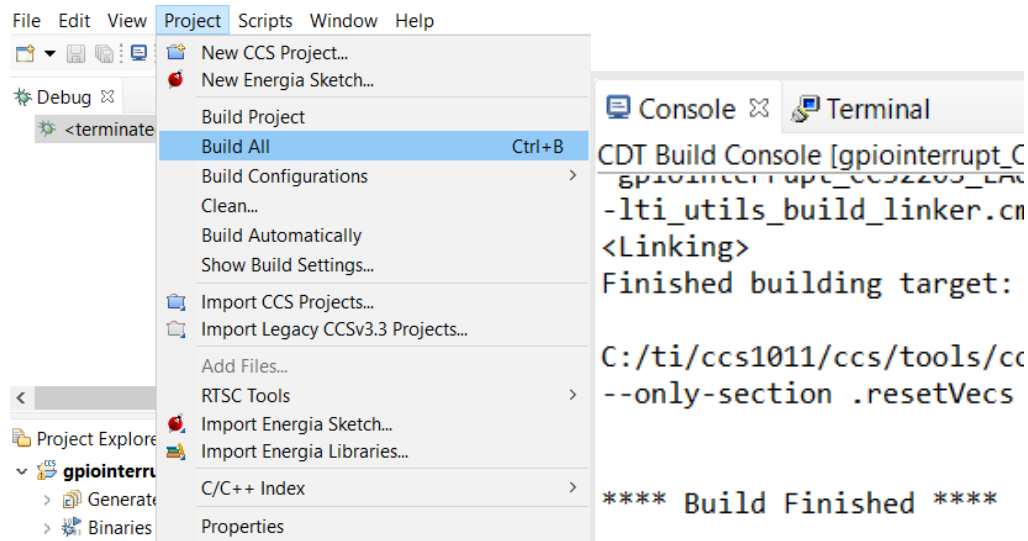


9. Review the project. Note that you will only need to change the contents of gpinterrupt.c for this lab.

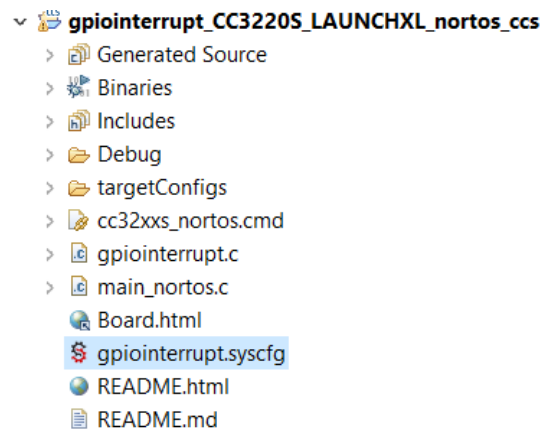
The screenshot shows the CCS IDE with the `gpinterrupt.c` file open. The code defines a `mainThread` function that initializes GPIO pins, configures the LED and button pins, and sets up interrupts. The Project Explorer on the left shows the project structure, including the `gpinterrupt.c` file.

```
73 void *mainThread(void *arg0)
74 {
75     /* Call driver init functions */
76     GPIO_init();
77
78     /* Configure the LED and button pins */
79     GPIO_setConfig(CONFIG_GPIO_LED_0, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
80     GPIO_setConfig(CONFIG_GPIO_LED_1, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
81     GPIO_setConfig(CONFIG_GPIO_BUTTON_0, GPIO_CFG_IN_PU | GPIO_CFG_IN_INT_FALLING);
82
83     /* Turn on user LED */
84     GPIO_write(CONFIG_GPIO_LED_0, CONFIG_GPIO_LED_ON);
85
86     /* Install Button callback */
87     GPIO_setCallback(CONFIG_GPIO_BUTTON_0, gpioButtonFxn0);
88
89     /* Enable interrupts */
90     GPIO_enableInt(CONFIG_GPIO_BUTTON_0);
91
92     /*
93      * If more than one input pin is available for your device, interrupts
94      * will be enabled on CONFIG_GPIO_BUTTON1.
95      */
96     if (CONFIG_GPIO_BUTTON_0 != CONFIG_GPIO_BUTTON_1) {
97         /* Configure BUTTON1 pin */
98         GPIO_setConfig(CONFIG_GPIO_BUTTON_1, GPIO_CFG_IN_PU | GPIO_CFG_IN_INT_FALLING);
99
100        /* Install Button callback */
101        GPIO_setCallback(CONFIG_GPIO_BUTTON_1, gpioButtonFxn1);
102        GPIO_enableInt(CONFIG_GPIO_BUTTON_1);
103    }
104 }
```

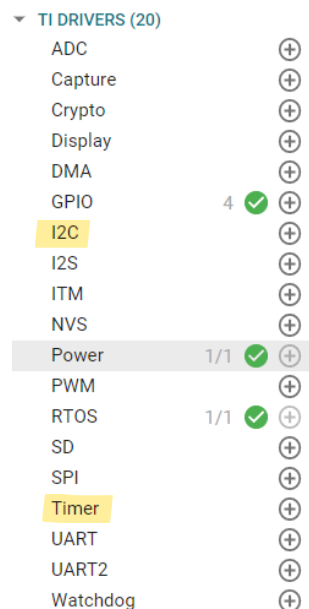
10. Next, build the code by selecting Project from the top menu options. Then click **Build All**.



11. Add the appropriate drivers using system config. TI provides a tool called system config for adding and configuring drivers in your project. To open the tool, double-click on the gpiointerrupt.syscfg file in the project.



12. Your goal is to add the **Timer** and **I2C** drivers, but if you already had the gpiointerrupt project imported, you may also already have the Timer driver imported from your previous milestone work. If that is the case, you will only need to add the **I2C driver** during the subsequent steps. Note that the I2C driver is connected to the temperature sensor. In the following example screenshot of the TI Drivers list, you can ignore the RTOS and POWER drivers. Both drivers are installed but disabled as part of the driver library default configuration. To add the Timer and I2C drivers click the plus signs next to Timer and I2C.





13. For this project we will need a timer that is 32 bits. The default will likely say 16 bits, and if that is the case, click the drop down next to Timer Type. Then select 32 bits.

Global Parameters Settings that affect all instances ^

Timer (1 Added) ⓘ + ADD REMOVE ALL

✓ CONFIG_TIMER_0 🗑

Name CONFIG_TIMER_0

Timer Type 32 Bits

Interrupt Priority 7 - Lowest Priority

PinMux Peripheral and Pin Configuration ^

Other Dependencies ^

14. Configure the I2C driver as shown in the following image. Under Use Hardware, LaunchPad I2C should be selected. Under SDA Pin, P02/10 (LaunchPad I2C) should be selected.

I2C (1 Added) ⓘ + ADD REMOVE ALL

✓ CONFIG_I2C_0 🗑

Name CONFIG_I2C_0

Use Hardware LaunchPad I2C

Maximum Bit Rate 0

Ignore Unused Address Conflicts ☐

Address Checks Fail

Speed Checks Fail

SCL Timeout 0x0

Interrupt Priority 7 - Lowest Priority

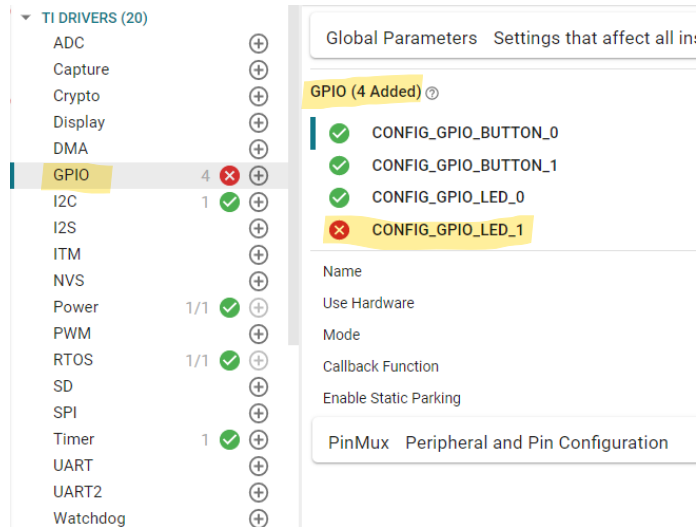
PinMux Peripheral and Pin Configuration ^

I2C Peripheral Any(I2C0)

SDA Pin P02/10 (LaunchPad I2C) 🔒

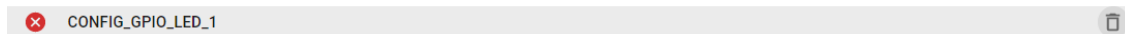
SCL Pin Any(P01/9 (LaunchPad I2C))

15. When you return to the TI Drivers list, you will notice you have a resource conflict, indicated by a red X on the screen. Unfortunately, this is a very common occurrence in embedded system design. The I2C peripheral uses the same pins as one of the LED GPIOs. We will need to **remove the driver** for **GPIO LED 1**, which is the green LED.

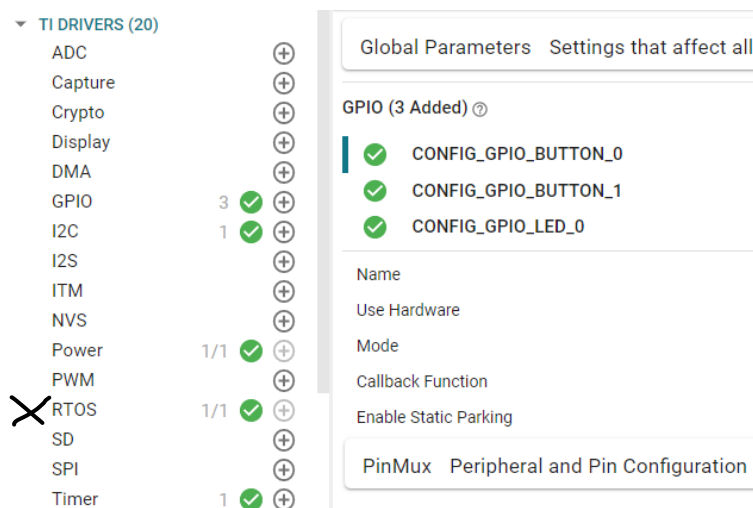


The screenshot shows the TI Drivers list on the left and the configuration panel on the right. In the TI Drivers list, the GPIO driver is highlighted with a red X next to it, indicating a resource conflict. The configuration panel on the right shows the GPIO (4 Added) settings. Under the Global Parameters section, the CONFIG_GPIO_LED_1 item is marked with a red X, indicating it is not added due to the conflict. The CONFIG_GPIO_LED_0 item is marked with a green checkmark, indicating it is added successfully.

16. Remove the driver by clicking on the trash can icon all the way to the right of the **CONFIG_GPIO_LED_1** item.



17. With the driver for LED 1 now removed, the TI Drivers list should display all green check marks.



The screenshot shows the TI Drivers list on the left and the configuration panel on the right. In the TI Drivers list, the GPIO driver now has a green checkmark next to it, indicating that the resource conflict has been resolved. The configuration panel on the right shows the GPIO (3 Added) settings. Under the Global Parameters section, the CONFIG_GPIO_LED_0 item is marked with a green checkmark, indicating it is added successfully. The CONFIG_GPIO_LED_1 item is no longer present in the list.



18. Add the UART by clicking the plus icon next to UART. Then configure the UART by selecting XDS110 UART under the Use Hardware option.

Capture		+
Crypto		+
Display		+
DMA		+
GPIO	3	✓ +
I2C	1	✓ +
I2S		+
ITM		+
NVS		+
Power	1/1	✓ +
PWM		+
RTOS	1/1	✓ +
SD		+
SPI		+
Timer	1	✓ +
UART	1	✓ +
UART2		+
Watchdog		+

UART (1 Added) ⓘ

✓ CONFIG_UART_0

Name: CONFIG_UART_0

Use Hardware: XDS110 UART

Data Direction: Send and Receive

Error Callback Function: Enter a function name to enable

Use DMA: ☐

Flow Control: ☐

Ring Buffer Size: 32

Interrupt Priority: 7 - Lowest Priority

PinMux Peripheral and Pin Configuration

Other Dependencies

19. Verify you have the GPIO, I2C, Timer, and UART drivers enabled. Do not make any changes to the Power and RTOS driver configurations. Then save the configuration and build the project following the same instructions as shown in step 10.

TI DRIVERS (20)

ADC		+
Capture		+
Crypto		+
Display		+
DMA		+
GPIO	3	✓ +
I2C	1	✓ +
I2S		+
ITM		+
NVS		+
Power	1/1	✓ +
PWM		+
RTOS	1/1	✓ +
SD		+
SPI		+
Timer	1	✓ +
UART	1	✓ +
UART2		+
Watchdog		+

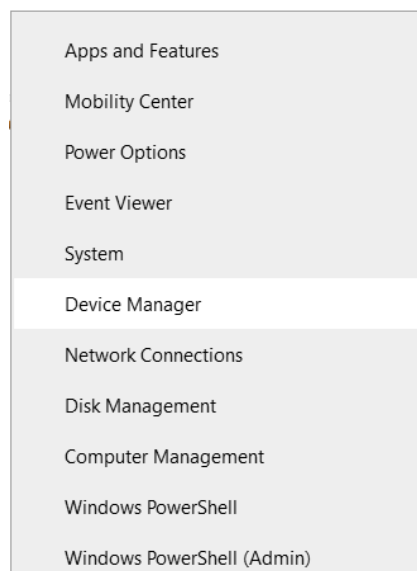
20. Note that **errors will appear when building the project**. This is because we removed the driver in sysconfig but we did not remove references to the driver in the code. To solve this issue, delete the **two lines referencing the LED 1 driver** in the gpiointerrupt.c file. Then build the project one more time. Now you will have a clean build!

```

64 void gpioButtonFxn1(uint_least8_t index)
65 {
66     /* Toggle an LED */
67     GPIO_toggle(CONFIG_GPIO_LED_1);
68 }
69
70 /*
71 * ===== mainThread =====
72 */
73 void *mainThread(void *arg0)
74 {
75     /* Call driver init functions */
76     GPIO_init();
77
78     /* Configure the LED and button pins */
79     GPIO_setConfig(CONFIG_GPIO_LED_0, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
80     GPIO_setConfig(CONFIG_GPIO_LED_1, GPIO_CFG_OUT_STD | GPIO_CFG_OUT_LOW);
81     GPIO_setConfig(CONFIG_GPIO_BUTTON_0, GPIO_CFG_IN_PU | GPIO_CFG_IN_INT_FALLING);
82

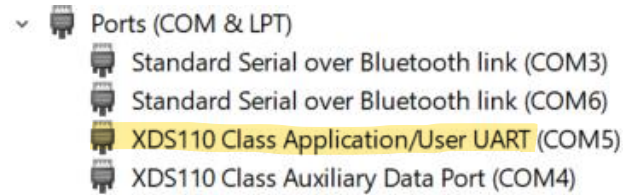
```

21. Open a **serial console** to communicate with the board. The USB connection to the board supports both the debug command protocol and a virtual com port for serial communications. In order to determine which com port the board is assigned to, open the **Windows Device Manager** by right-clicking on the Windows start icon in the lower left corner. In the device manager select **Ports(COM & LPT)**.

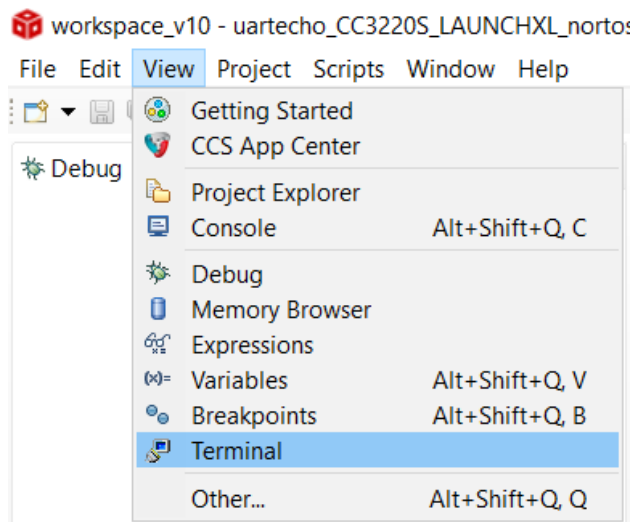




22. The USB connection to the TI board is called **XDS110**. For this lab, look for **XDS110 Class Application/User UART**. In the provided example, the UART interface is COM5. However, it may be a different COM port on your computer. Make note of the COMx port assigned to XDS110 Class Application/User UART for your PC.



23. In CCS, select the **View** tab to open the **Terminal** window.

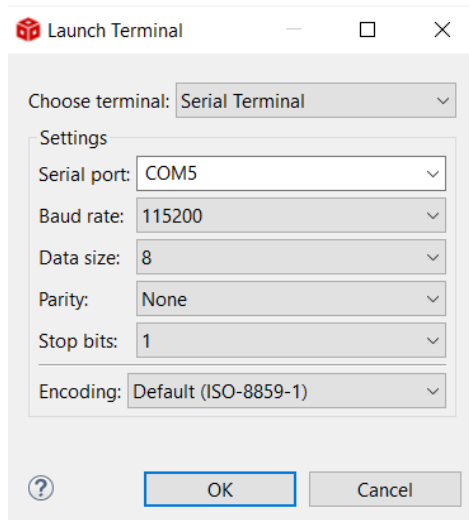


24. From the Terminal window, select the Open a Terminal button which appears with a blue monitor icon (shown first on the left of the row of icons in the upper right corner).





25. The Launch Terminal window will open. From here, choose **Serial Terminal**. Then select the COM port from earlier. Remember that while this example uses COM5, yours will likely be different. Every other field will remain as its default. Once complete, click **Ok**.



26. Integrate the driver code into gpiointerrupt.c. These drivers are installed with the Simplelink SDK. You can locate these drivers at the following locations:

- C:\ti\simplelink_cc32xx_sdk_4_30_00_06\source\ti\drivers
- C:\ti\simplelink_cc32xx_sdk_4_30_00_06\examples\nortos\CC3220S_LAUNCHXL\drivers
- C:\ti\simplelink_cc32xx_sdk_4_30_00_06\examples\rtos\CC3220S_LAUNCHXL\drivers

27. For the I2C driver, copy the following code into CCS. You will need to call initUART() before calling initI2C(). Note that initI2C() initializes the I2C peripheral and readTemp() uses the I2C peripheral to read the temperature sensor.

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

// I2C Global Variables
static const struct {
    uint8_t address;
    uint8_t resultReg;
} sensors[3] = {
    { 0x48, 0x0000, "11X" },
    { 0x49, 0x0000, "116" },
    { 0x41, 0x0001, "006" }
};

uint8_t txBuffer[1];
uint8_t rxBuffer[2];
I2C_Transaction i2cTransaction;

// Driver Handles - Global variables
```



```
I2C_Handle          i2c;

// Make sure you call initUART() before calling this function.
void initI2C(void)
{
    int8_t          i, found;
    I2C_Params      i2cParams;

    DISPLAY(snprintf(output, 64, "Initializing I2C Driver - "))

    // Init the driver
    I2C_init();

    // Configure the driver
    I2C_Params_init(&i2cParams);
    i2cParams.bitRate = I2C_400kHz;

    // Open the driver
    i2c = I2C_open(CONFIG_I2C_0, &i2cParams);
    if (i2c == NULL)
    {
        DISPLAY(snprintf(output, 64, "Failed\n\r"))
        while (1);
    }

    DISPLAY(snprintf(output, 32, "Passed\n\r"))

    // Boards were shipped with different sensors.
    // Welcome to the world of embedded systems.
    // Try to determine which sensor we have.
    // Scan through the possible sensor addresses

    /* Common I2C transaction setup */
    i2cTransaction.writeBuf  = txBuffer;
    i2cTransaction.writeCount = 1;
    i2cTransaction.readBuf   = rxBuffer;
    i2cTransaction.readCount = 0;

    found = false;
    for (i=0; i<3; ++i)
    {
        i2cTransaction.slaveAddress = sensors[i].address;
        txBuffer[0] = sensors[i].resultReg;

        DISPLAY(snprintf(output, 64, "Is this %s? ", sensors[i].id))
        if (I2C_transfer(i2c, &i2cTransaction))
        {
            DISPLAY(snprintf(output, 64, "Found\n\r"))
            found = true;
            break;
        }
    }
}
```



```
    }
    DISPLAY(snprintf(output, 64, "No\n\r"))
}

if(found)
{
    DISPLAY(snprintf(output, 64, "Detected TMP%s I2C address:
%x\n\r", sensors[i].id, i2cTransaction.slaveAddress))
}
else
{
    DISPLAY(snprintf(output, 64, "Temperature sensor not found,
contact professor\n\r"))
}
}

int16_t readTemp(void)
{
    Int    j;
    int16_t temperature = 0;

    i2cTransaction.readCount = 2;
    if (I2C_transfer(i2c, &i2cTransaction))
    {
        /*
        * Extract degrees C from the received data;
        * see TMP sensor datasheet
        */
        temperature = (rxBuffer[0] << 8) | (rxBuffer[1]);
        temperature *= 0.0078125;

        /*
        * If the MSB is set '1', then we have a 2's complement
        * negative value which needs to be sign extended
        */
        if (rxBuffer[0] & 0x80)
        {
            temperature |= 0xF000;
        }
    }
    else
    {
        DISPLAY(snprintf(output, 64, "Error reading temperature sensor
(%d)\n\r", i2cTransaction.status))
        DISPLAY(snprintf(output, 64, "Please power cycle your board by
unplugging USB and plugging back in.\n\r"))
    }

    return temperature;
}
```



```
}
```

28. For the UART driver, copy the following code into CCS.

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

#define DISPLAY(x) UART_write(uart, &output, x);

// UART Global Variables
Char      output[64];
Int       bytesToSend;

// Driver Handles - Global variables
UART_Handle  uart;

void initUART(void)
{
    UART_Params uartParams;

    // Init the driver
    UART_init();

    // Configure the driver
    UART_Params_init(&uartParams);
    uartParams.writeDataMode = UART_DATA_BINARY;
    uartParams.readDataMode = UART_DATA_BINARY;
    uartParams.readReturnMode = UART_RETURN_FULL;
    uartParams.baudRate = 115200;

    // Open the driver
    uart = UART_open(CONFIG_UART_0, &uartParams);

    if (uart == NULL) {
        /* UART_open() failed */
        while (1);
    }
}
```

29. For the Timer driver, copy the following code into CCS. This should appear familiar to you because of your practice with this course's resources. Note that the driver defaults to calling timerCallback() every 1000000 microseconds (1 second). You will need to modify this to match your work.

```
#include <ti/drivers/Timer.h>

// Driver Handles - Global variables
```



```
Timer_Handle timer0;

volatile unsigned char TimerFlag = 0;
void timerCallback(Timer_Handle myHandle, int_fast16_t status)
{
    TimerFlag = 1;
}

void initTimer(void)
{
    Timer_Params    params;

    // Init the driver
    Timer_init();

    // Configure the driver
    Timer_Params_init(&params);
    params.period = 1000000;
    params.periodUnits = Timer_PERIOD_US;
    params.timerMode = Timer_CONTINUOUS_CALLBACK;
    params.timerCallback = timerCallback;

    // Open the driver
    timer0 = Timer_open(CONFIG_TIMER_0, &params);

    if (timer0 == NULL) {
        /* Failed to initialized timer */
        while (1) {}
    }

    if (Timer_start(timer0) == Timer_STATUS_ERROR) {
        /* Failed to start timer */
        while (1) {}
    }
}
```

30. Modify the provided gpiointerrupt.c code. Your code needs to check the buttons every 200ms, check the temperature every 500ms, and update the LED and report to the server every second (via the UART). If you push a button, it increases or decreases the temperature set-point by 1 degree every 200ms. If the temperature is greater than the set-point, the LED should turn off. If the temperature is less than the set-point, the LED should turn on (the LED controls a heater). You can simulate a heating or cooling room by putting your finger on the temperature sensor. The output to the server (via UART) should be formatted as <AA,BB,S,CCCC>. This can be broken down as follows:

- AA = ASCII decimal value of room temperature (00 - 99) degrees Celsius
- BB = ASCII decimal value of set-point temperature (00-99) degrees Celsius
- S = '0' if heat is off, '1' if heat is on



- CCCC = decimal count of seconds since board has been reset
- <%02d,%02d,%d,%04d> = temperature, set-point, heat, seconds

31. To accomplish this work, you will need to use a task scheduler, as described in your course resources. Note that some additional content from your course will be useful in helping you complete this project.

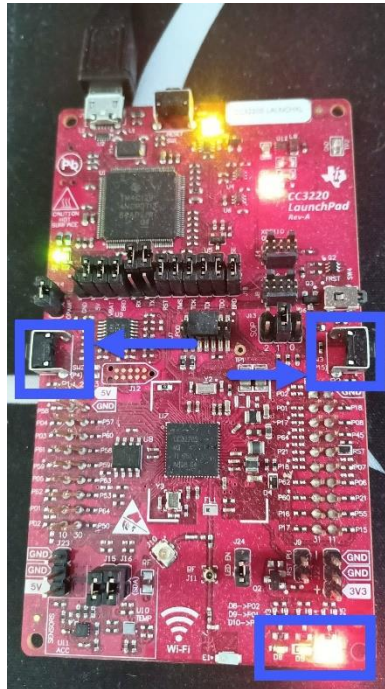
- Milestone Two covered how to turn an LED on or off.
- Milestone Three covered how to change the period for the timer interrupt.
- Milestone Three also showed you how to use the button interrupt callbacks.
- zyBooks activities covered how to create a task scheduler.

32. Consider the following code to help you get started.

```
initUART(); // The UART must be initialized before calling initI2C()
initI2C();
initTimer();

// Loop Forever
// The student should add flags (similar to the timer flag) to the button handlers.
while (1)
{
    // Every 200ms check the button flags
    // Every 500ms read the temperature and update the LED
    // Every second output the following to the UART
    // "<%02d,%02d,%d,%04d>, temperature, setpoint, heat, seconds"
    DISPLAY( snprintf(output, 64, "<%02d,%02d,%d,%04d>\n\r", temperature, setpoint, heat, seconds))
    // Refer to ZyBooks - "Converting different-period tasks to C"
    // Remember to configure the timer period
    while (!TimerFlag){} // Wait for timer period
    TimerFlag = 0;       // Lower flag raised by timer
    ++timer;
}
```




33. Make sure you pay attention to the button you use for this work. The button next to the USB cable is the reset button and should not be pressed. Instead select either of the buttons located opposite each other near the middle of the board to use for the set-point. The LEDs are located in the corner diagonally opposite the USB cable.







34. The silver square next to the word TEMP is the temperature sensor. This is positioned in the lower left corner of the board, when it is oriented so the USB is in the upper left.



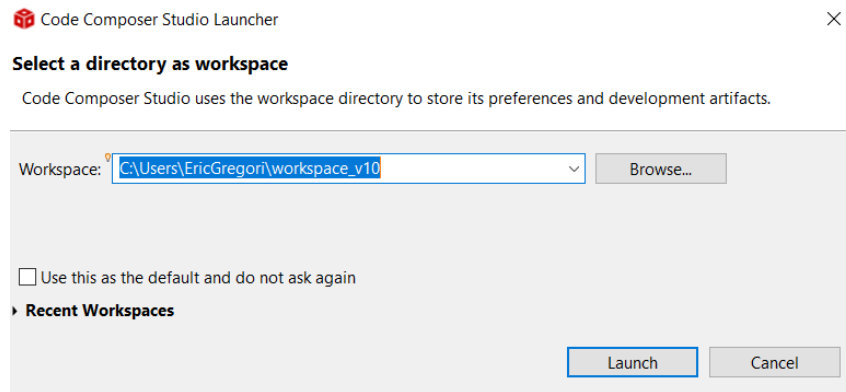
35. As you work, remember: the easiest method to execute your code is to use the debug icon (green bug). Clicking the debug icon will load the code into the board and put the tool into debug mode. Once in debug mode, six additional icons become live.

-  Debug your code
-  Execute your code
-  Exit debug mode



-  Step into
-  Step over
-  Step out
-  Halt the code

36. Remember that part of your submission will also involve creating a video that demonstrates the functionality you have constructed. Make sure you push the button during the video so you can show what happens.
37. You will also need to zip your workspace and submit the ZIP file. Your workspace is the directory you selected when opening CCS. In the provided example, you would zip the workspace_v10 folder and submit the resulting ZIP file.



38. Congratulations! You have now completed this lab guide. Remember to refer to the Project Guidelines and Rubric in your course to ensure you have all the necessary components for your required submission to Brightspace.