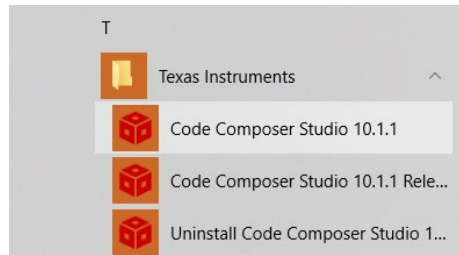


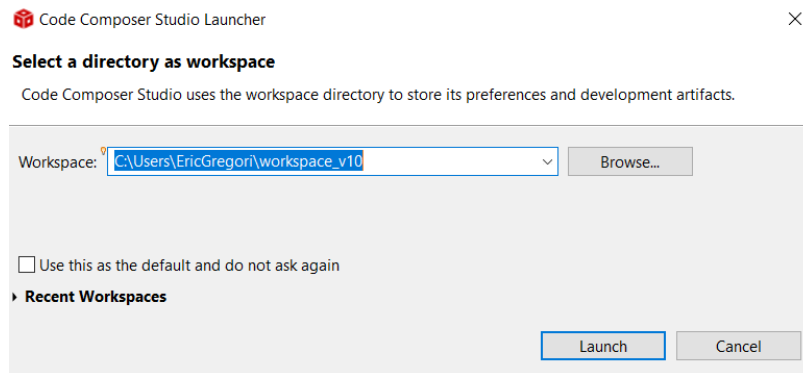


## CS 350 Milestone Three Timer Interrupt Lab Guide

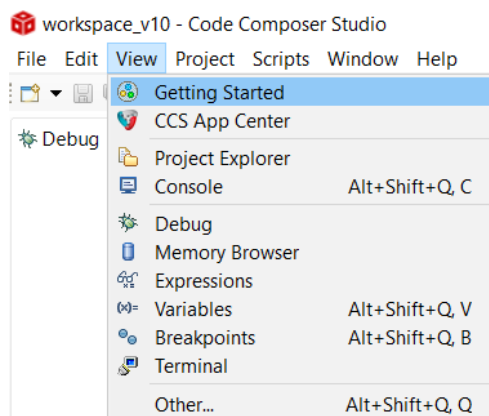
1. Open TI Code Composer Studio (CCS). If you have not installed CCS, please install CCS by following installation guidance in Module One of your course. After CCS is installed, you can open it by clicking **Code Composer Studio xxxx**, which can be found in the Texas Instruments folder. The CCS installer 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.

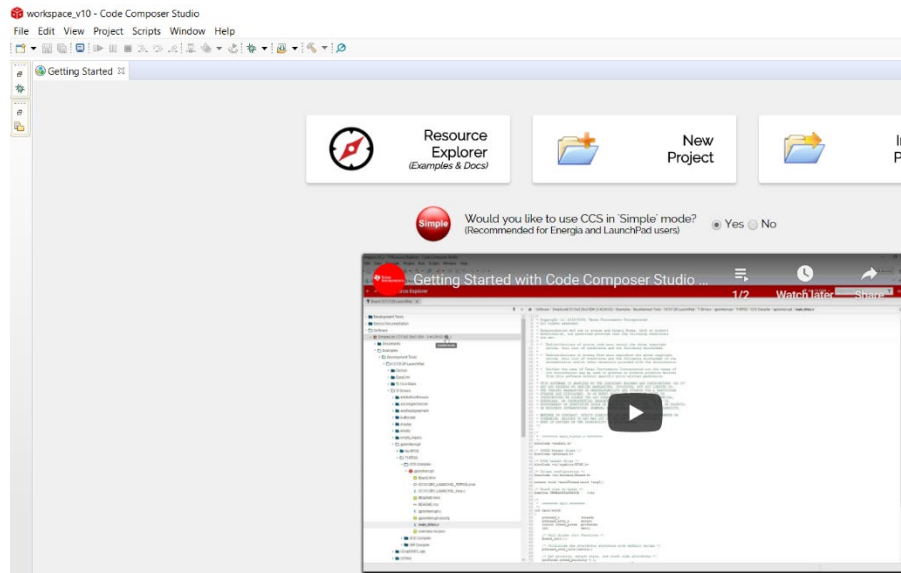


2. Select a workspace then click **Launch**. It is very important that you take note of your workspace location. You will need the information later in order to turn in your project.

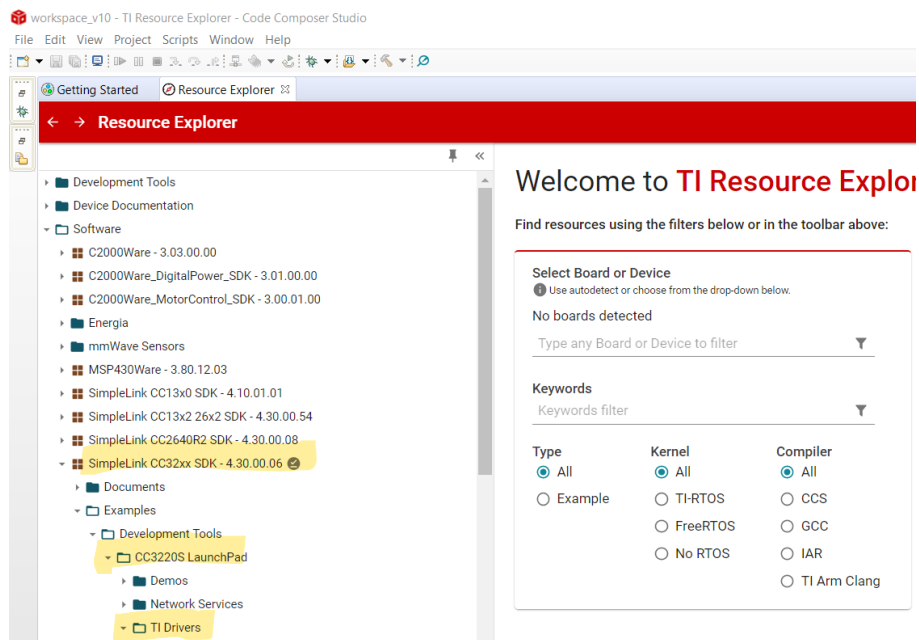


3. Open the Getting Started window in CCS. CCS takes about 30 seconds to fully open (it's written in Java). You can review the loading process by looking at the lower right corner of the screen. You will see green progress bars popping up. After about 30 seconds, CCS should be fully loaded and you will 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.

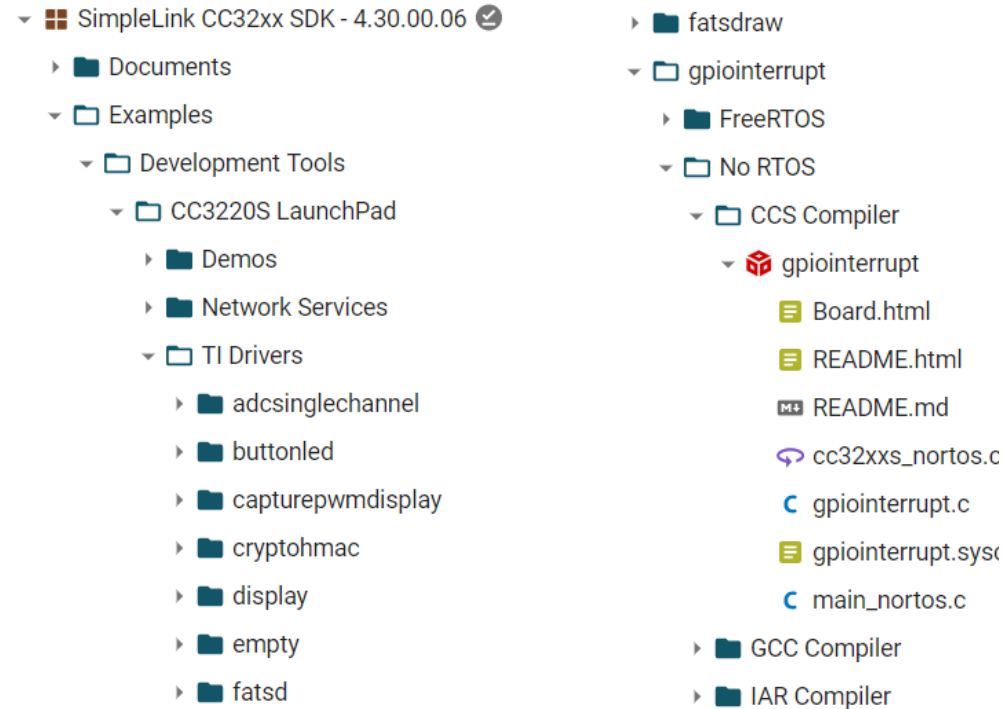


5. You should already have the SDK installed, which is indicated by a check mark by the SimpleLink CC32xx SDK.

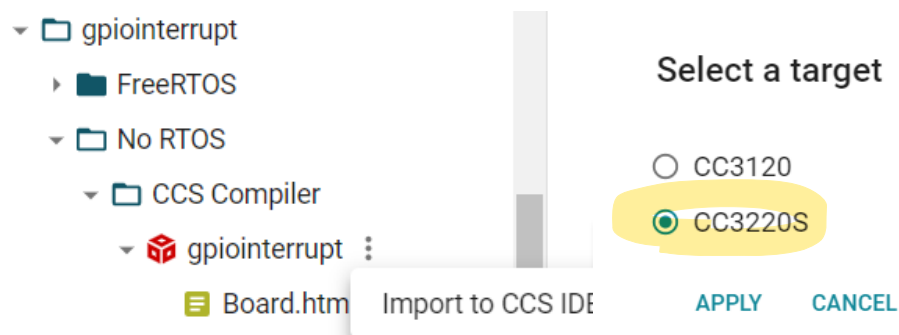
▶  SimpleLink CC32xx SDK - 4.30.00.06 

7.10.00.13

- Import the **gpiointerrupt** project. To accomplish this, 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.



~



- Review the project. Note that you will only need to change the contents of `gpiointerrupt.c` for this lab.

```

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 }

```

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

The 'Project' menu is open, and 'Build All' (Ctrl+B) is selected. The 'Console' window shows the following output:

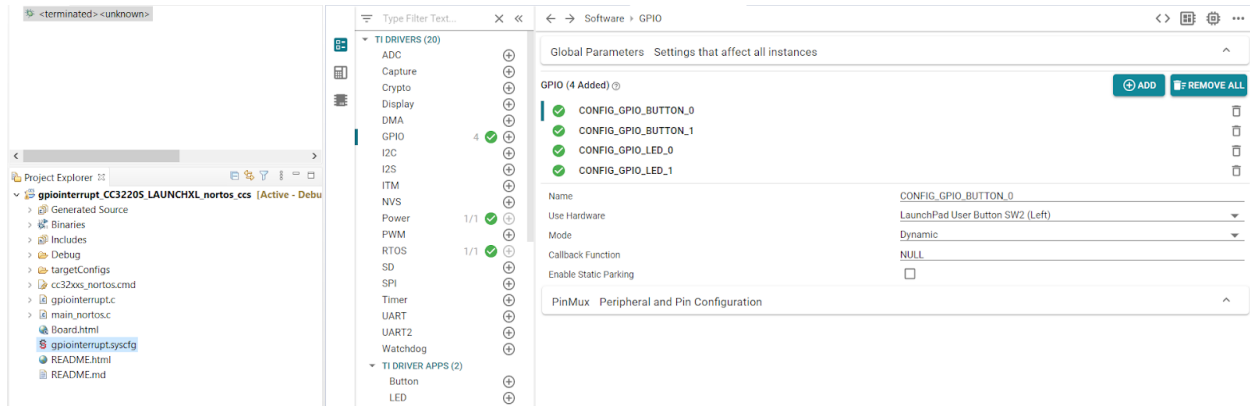
```

CDT Build Console [gpiointerrupt_C
gpiointerrupt_C_CCS32205_LAUNCHXL
-lti_utils_build_linker.cn
<Linking>
Finished building target:
C:/ti/ccs1011/ccs/tools/cc
--only-section .resetVecs

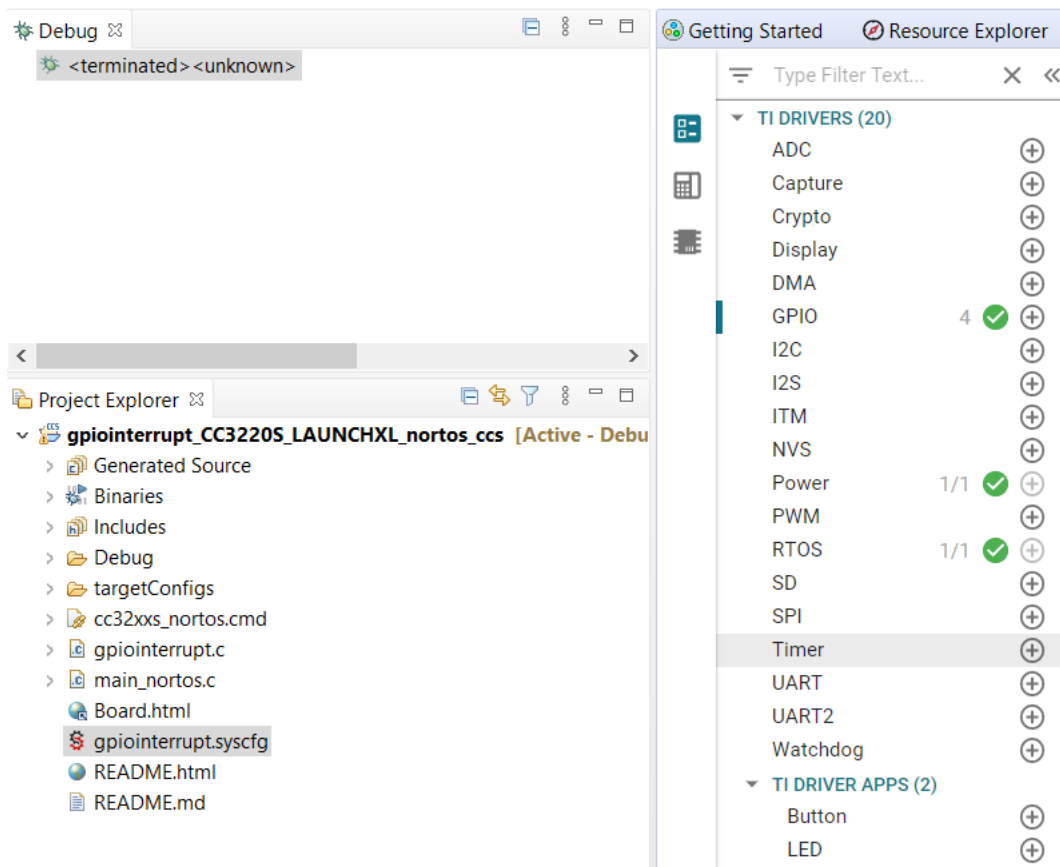
**** Build Finished ****

```

10. Add a timer driver to the project 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 gpinterrupt.syscfg file in the project.



11. To add the timer driver to the project, click the plus sign next to Timer in the TI Drivers list.





12. 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

^

13. Rebuild the code following the same process outlined in step 9. Then integrate the following code into gpinterrupt.c.

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

void timerCallback(Timer_Handle myHandle, int_fast16_t status)
{
}

void initTimer(void)
{
    Timer_Handle timer0;
    Timer_Params params;

    Timer_init();
    Timer_Params_init(&params);
    params.period = 1000000;
    params.periodUnits = Timer_PERIOD_US;
    params.timerMode = Timer_CONTINUOUS_CALLBACK;
    params.timerCallback = timerCallback;

    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) {}
    }
}
```



14. After working through this process, you will need to address the following questions:

- What is the purpose of the `timerCallback()` function?
- What does `period` mean in this context?
- How does the `Timer_CONTINUOUS_CALLBACK` parameter impact the driver?
- What is `gpioButtonFxn0()` used for?
- What is the purpose of `GPIO_CFG_IN_INT_FALLING`?

15. Now it is time to modify `gpinterrupt.c` and the code provided in step 13. Your code will need to accomplish the following:

- Call your state machine every 500000 us.
- Continuously blink SOS in Morse code on the green and red LEDs.
- If a button is pushed, toggle the message between SOS and OK. Pushing the button in the middle of a message should NOT change the message until it is complete. For example, if you push the button while the 'O' in SOS is being blinked out, the message will not change to OK until after the SOS message is completed. Remember in a previous module your work with UART included an example of how to turn an LED on or off (as opposed to toggle).

16. You may wish to perform your own research on how Morse code works to help better inform your work. The following guidance will help provide a base for the functionality you are creating:

- Dot = red LED on for 500ms
- Dash = green LED on for 1500ms
- 3\*500ms between characters (both LEDs off)
- 7\*500ms between words (both LEDs off), for example SOS 3500ms SOS

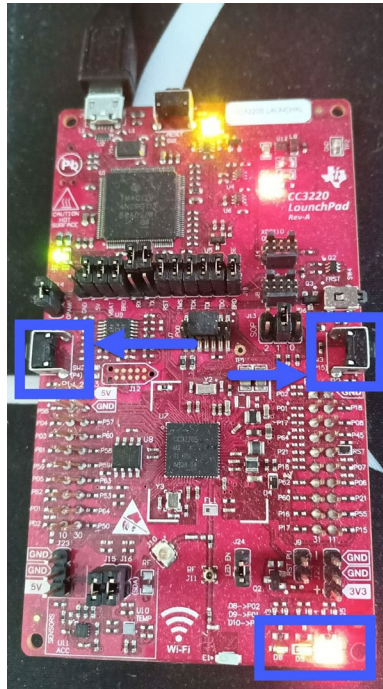
17. Remember, if a button is pushed at any time during a message, the new message will not start until the end of the 7\*500ms gap between messages. The following example assumes the button is pushed during the first SOS and during the last 3500ms.

SOS 3500ms OK 3500ms OK 3500ms OK 3500ms SOS








500 ms	500 ms	500 ms	500 ms	500 ms	500 ms	500 ms	500 ms	500 ms	500 ms	500 ms	500 ms	500 ms	500 ms	500 ms	500 ms	
led off	led off	led off	led off	led off	led off	led off	led off									break between words
led on	led off	led on	led off	led on	led off	led off	led off									S
led on	led on	led on	led off	led on	led on	led on	led off	led on	led on	led on	led off	led off	led off			O
led on	led off	led on	led off	led on												S
led on	led on	led on	led off	led on	led off	led on	led on	led on								K



18. Make sure you pay attention to the button you select 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. The LEDs are located in the corner diagonally opposite the USB cable.



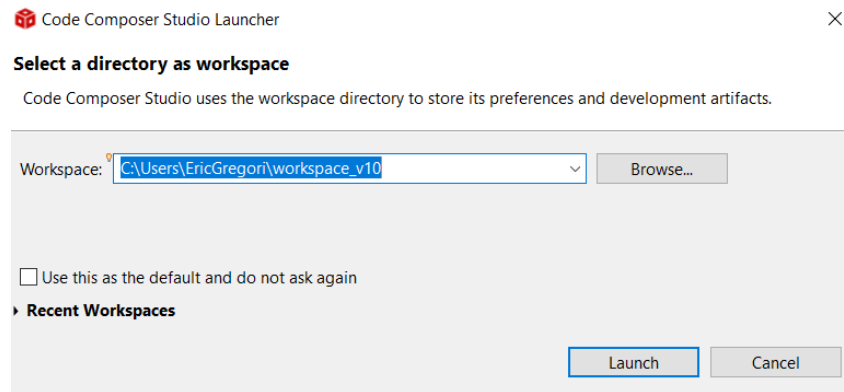
19. As you work, remember: the easiest method for executing your code is to use the debug icon (green bug). Clicking the debug icon will load the code into the board and put the tool into debug mode. Once in debug mode, six additional icons become live.

-  Debug your code
-  Execute your code
-  Exit debug mode
-  Step into
-  Step over
-  Step out
-  Halt the code

20. You will need to create a state machine in order to successfully complete this work. Remember, 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.



21. 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.



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

Dot Dot Dot = Red.

DASH DASH DASH = Green.