Application Note

Practical Software Debugging Methods and Software Design FAQs for CC27xx and CC23xx

Shuyang Zhong, Island Wei
Simon Huo, Conner Li

ABSTRACT

This document was translated from a simplified Chinese source. (ZHCAFV0)

With the release and evolution of CCS 20.0, some debugging methods for the CC2340R5 and CC2745R10-Q1 differ from before. Therefore, this article will introduce some practical debugging methods using the new CCS 20.0 version as the development platform. It requires the reader to have certain software debugging capabilities and a basic understanding of IDE, SDK, and hardware.

Table of Contents

1 Links to Hardware and Software Design References

When we come into contact with the TI SimpleLink family, setting up the environment is the first step, and getting familiar with the hardware and software design is the second step. For your reference during development, some links to useful technical documents are shared below.

1.1 Data Sheet and User Guide

On product pages for chips: TI's data sheets and user guides are published on corresponding product pages on the website, where customers only need to locate the specific product page to download all publicly available design materials and application notes.

- https://www.ti.com/product/CC2340R5
- https://www.ti.com/product/CC2340R5-Q1
- https://www.ti.com/product/CC2745R10-Q1
- https://www.ti.com/product/CC2745P10-Q1

On product pages for development boards: BOMs and design drawings for development boards are available on their respective product pages.

- LP-CC2340R5
- LP-CC2745R10

On product pages for emulators

- LP-XDS110

1.2 Required and Recommended Software for Environment Setup

- Required software:

  Integrated development environment: https://www.ti.com/tool/download/CCSTUDIO

  Software development kit: https://www.ti.com/tool/download/SIMPLELINK-LOWPOWER-F3-SDK

  Compiler kit: https://www.ti.com/tool/download/ARM-CGT-CLANG
- Recommended software:

  Standalone flashing tool: https://www.ti.com/tool/download/UNIFLASH

  Standalone graphical configuration tool: https://www.ti.com/tool/download/SYSCONFIG

1.3 SDK Examples

- BLE-stack API documentation: API Reference
- Peripheral driver API documentation: TI Driver API Reference

1.4 Hardware Design References

Hardware design files for the CC2745 development board

- RHA package: LAUNCHXL-CC2745R10-Q1 Design Files (Rev. B)

Hardware design files for the CC2340 development board

- RGE package: LP-EM-CC2340r5-RGE-4x4-IS24 Design Files (Rev. A)
- RKP package: LP-EM-CC2340R5 Design Files

Hardware migration

- Migrating from CC26xx to CC23xx and CC27xx: Hardware Migration to CC2340R5

Antenna design

- CC234x and CC27xx Hardware Configuration and PCB Design Considerations

1.5 Power Calculator Table

https://www.ti.com/tool/BT-POWER-CALC

1.6 Links to Product QDID Certificates

CC2340R5: https://www.ti.com/product/CC2340R5

CC2340R5-Q1: https://www.ti.com/product/CC2340R5-Q1

CC2745-Q1: https://qualification.bluetooth.com/ListingDetails/234268

1.7 HSM (CC2745) Certification

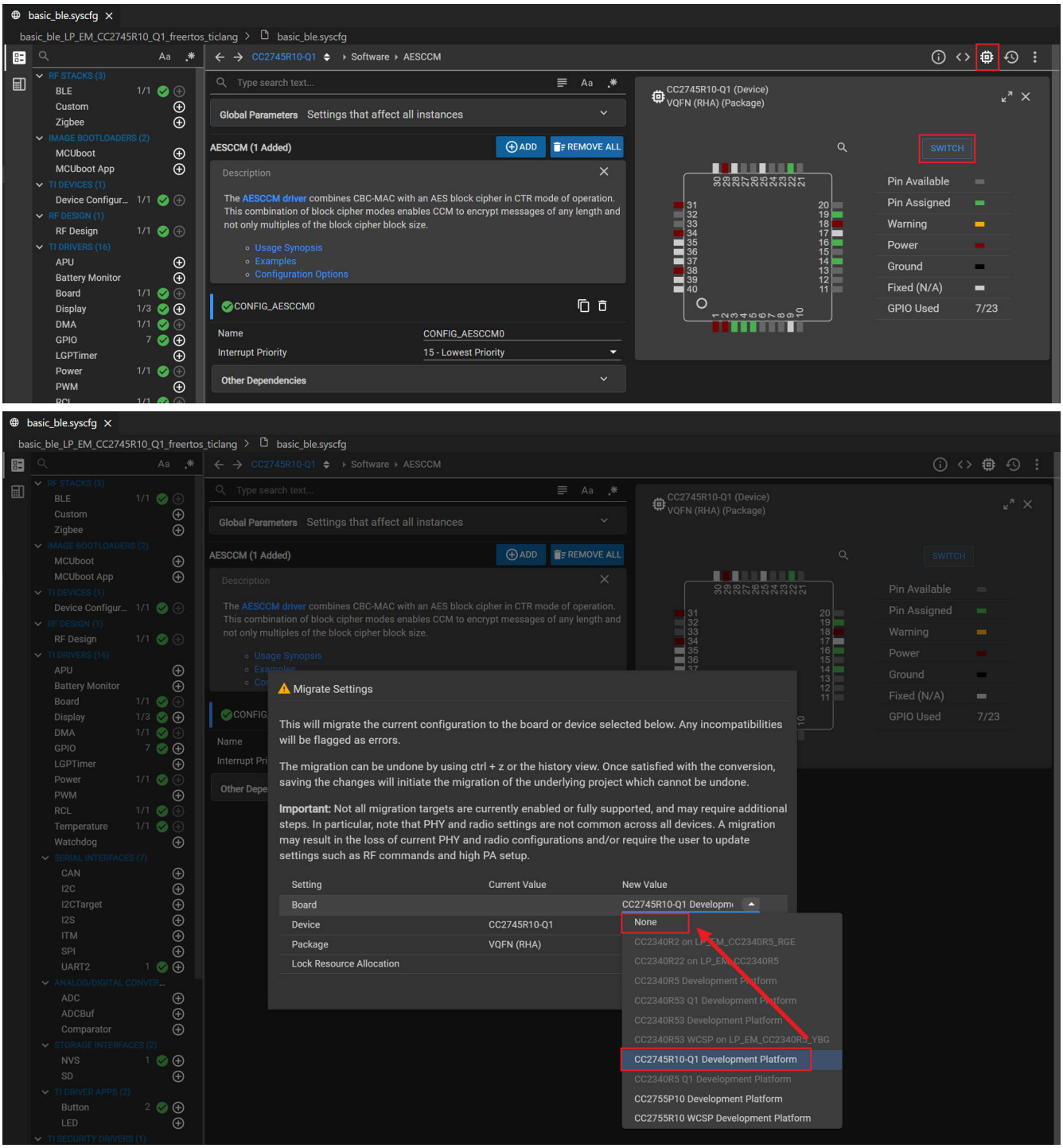
The HSM is a separate hardware security module within the CC2745, primarily responsible for key encryption/decryption and key read/write protection.

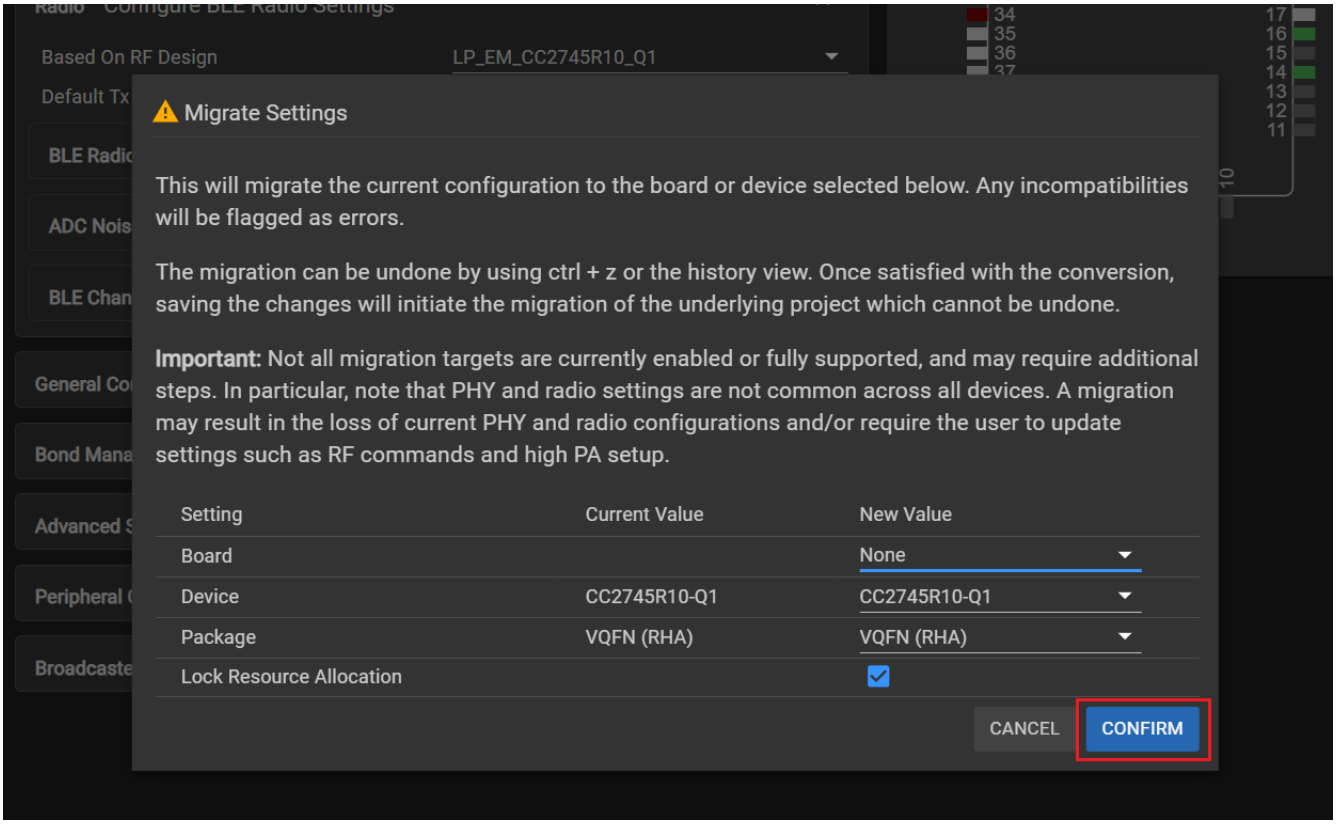
Users often strongly demand guidance on the security level of the HSM. Here are three examples: SESIP, CRNG, and NIST CAVP compliance.

2 CCS 20.0-Based Debugging Methods

2.1 How to Map a Project File to the Customer's Own Board

During development, it is normal that the desired PIN port fails to be mapped to the relevant function, or even that IO port conflicts occur, if no modifications are made. This is because the demos provided by the TI SDK map related interfaces based on the hardware design of the development board. To resolve this issue, it is necessary to configure the board to NONE through SysConfig embedded in CCS, set the corresponding chip, and then click CONFIRM to complete the switch.

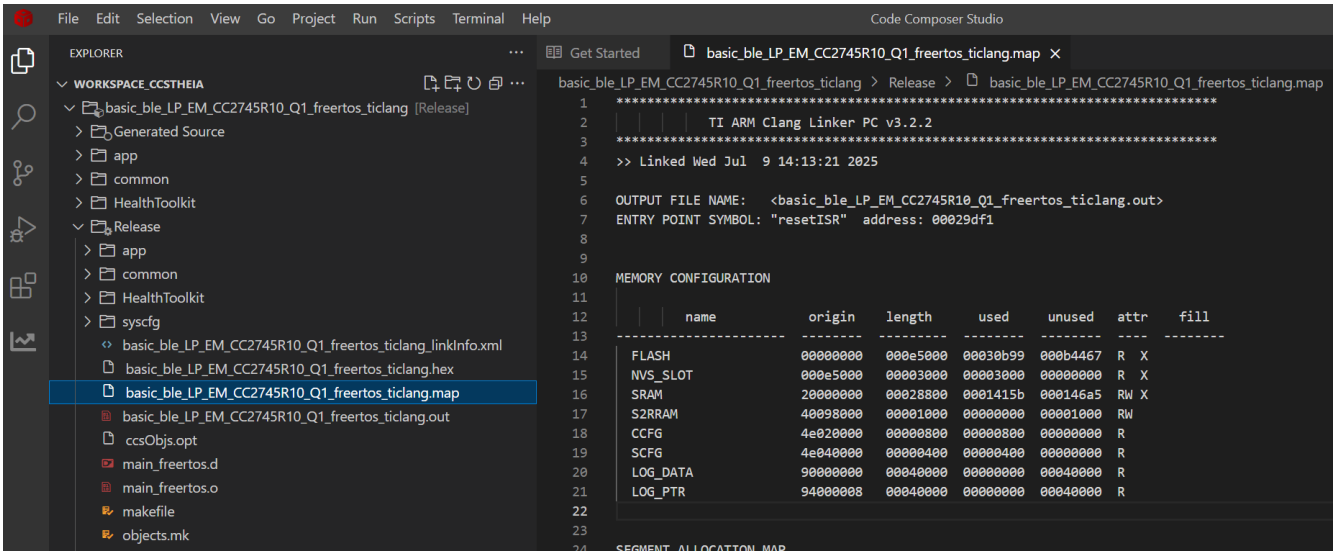




2.2 Evaluating RAM Consumption and Checking for Stack Overflow With Map File and ROV

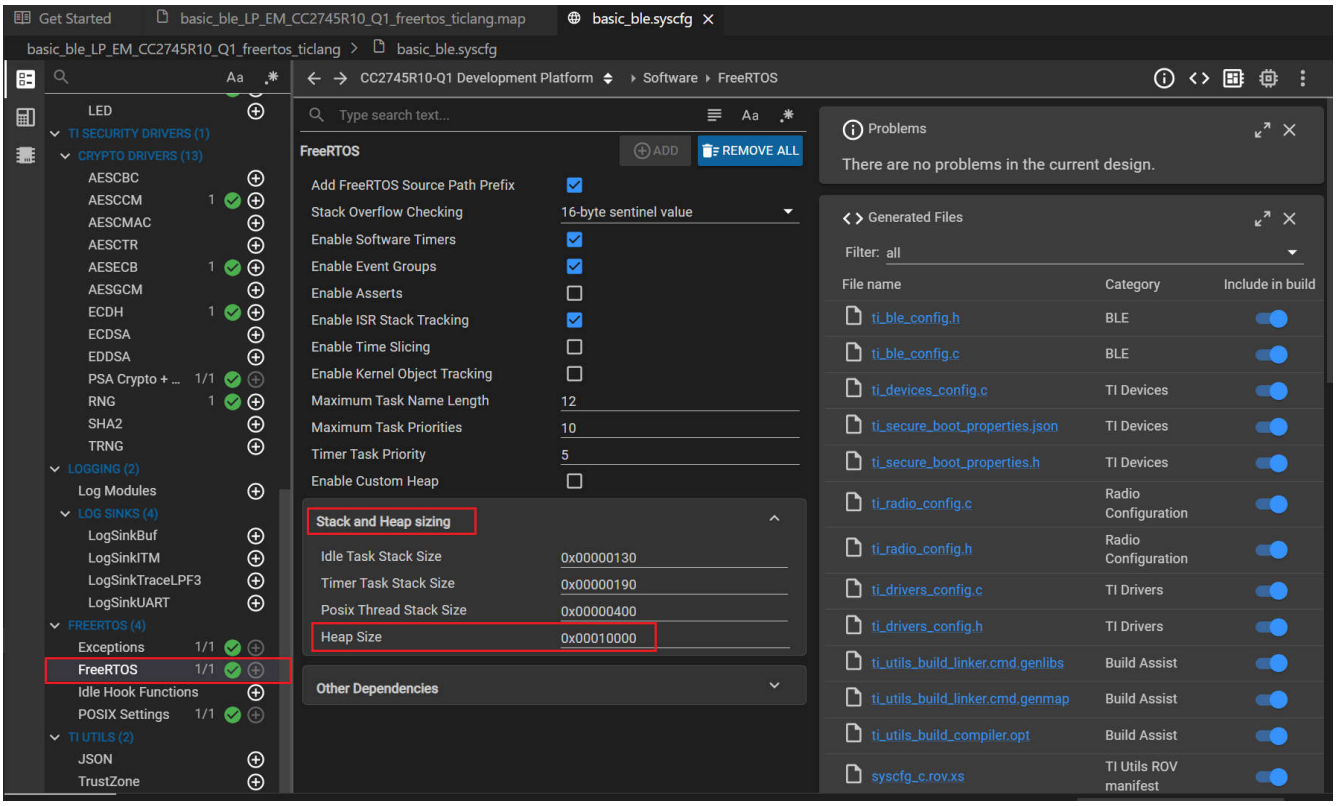
In BLE applications, the question of how much Heap space to allocate to the protocol stack often arises. Developers may address this question by using the .map file and ROV to observe the Heap usage.

Once the project is imported and built, a .map file with the corresponding project name is visible in the Release/ Debug folder under the project directory. This file shows the detailed usage of SRAM, Flash, and CCFG/SCFG on the chip.



This figure shows the used and unused space of the project under the current configuration. The generation of this file corresponds to the .cmd file located in the project directory. Free-RTOS is extensively used in the SimpleLink F3 SDK, and there is an allocation of Free-RTOS Heap space via SysConfig, which allows users

to increase or decrease the allocated Heap space depending on their specific requirements. When searching for .bss in the .map file, you may find that the Heap managed by Free-RTOS is allocated in the .bss section.

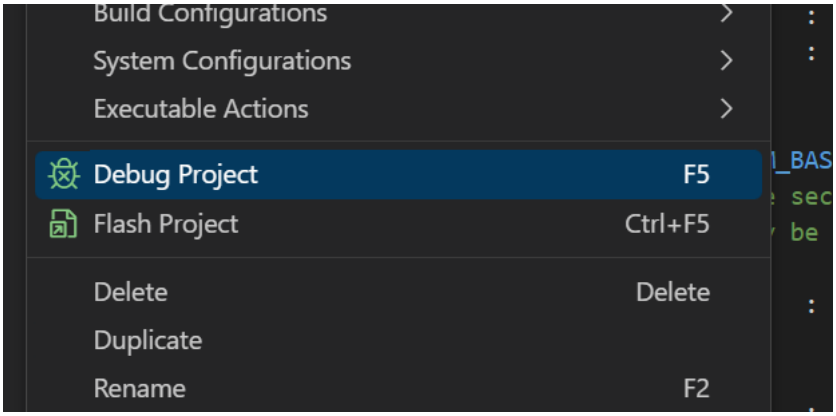


The .bss section points to the RAM space in the .cmd file.

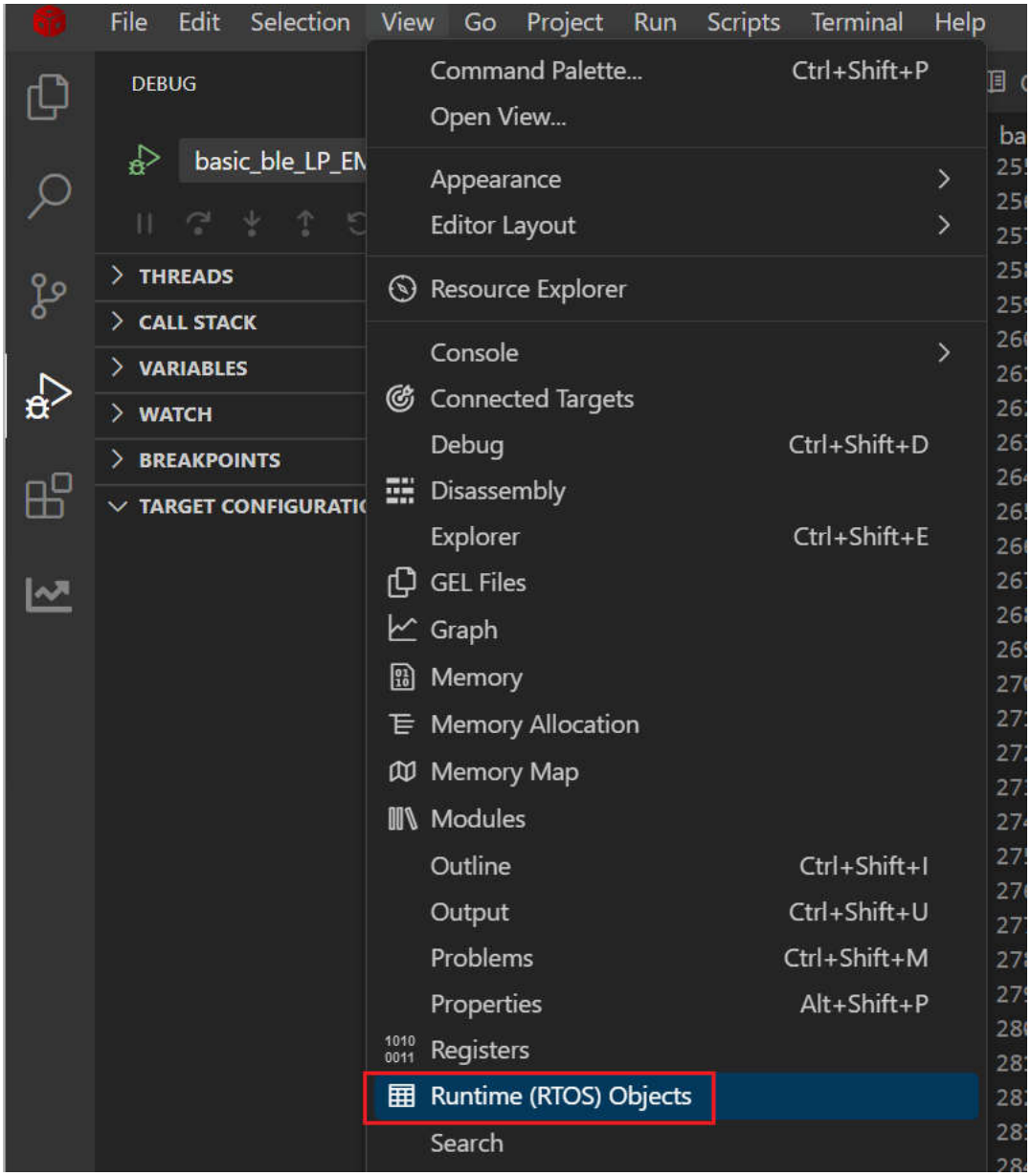
```
273   #if defined(RAM_BASE) && defined(RAM_SIZE)
274       .ramVecs        :   > SRAM, type = NOLOAD, ALIGN(256)
275       .data           :   > SRAM
276       .bss            :   > SRAM
277       .sysmem         :   > SRAM
278       .stack          :   > SRAM (HIGH)
279       .nonretenvar    :   > SRAM
280   #endif
```

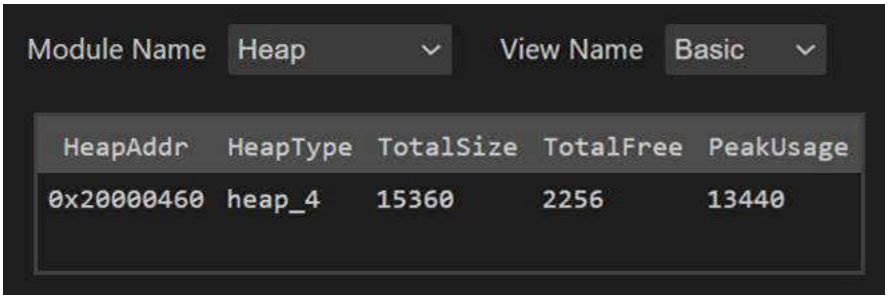
```
2642
2643  .bss        0    20000460    00012bd0    UNINITIALIZED
2644                     20000460    00010000      ti_freertos_config.o (.bss.ucHeap)
2645                     20010460    00000580      hsmddk_cc27xx_its.a : adapter_psa_preprovisioned
2646                     200109e0    00000210      (.common:gl_PSA_Key)
2647                     20010bf0    0000020c      secure_drivers_cc27xx.a : HSMLPF3.c.obj (.bss.op
2648                     20010dfc    00000200      hsmddk_cc27xx_its.a : tfm_internal_trusted_stora
```

After reviewing the .map file, to determine how much Heap space to allocate to the protocol stack for our applications, we can monitor the peak Heap consumption using ROV while maintaining four simultaneous connections, taking BLE in the peripheral role as an example. Open CCS 20.x, select the project you want to test, right-click it, and click Debug Project.



After properly connecting and starting the debugging process, click View -> Run Time Object on the toolbar to launch the ROV feature. Select Heap in the Module Name list and select either Basic or Detailed on the right. Run the code normally, and connect four peer devices to the emulation device in sequence. In the example code environment, the peak Heap consumption under four simultaneous connections is observed to be 13440 bytes (13kB). In this case, to ensure the program runs normally, at least 13440 bytes must be allocated for the Heap.

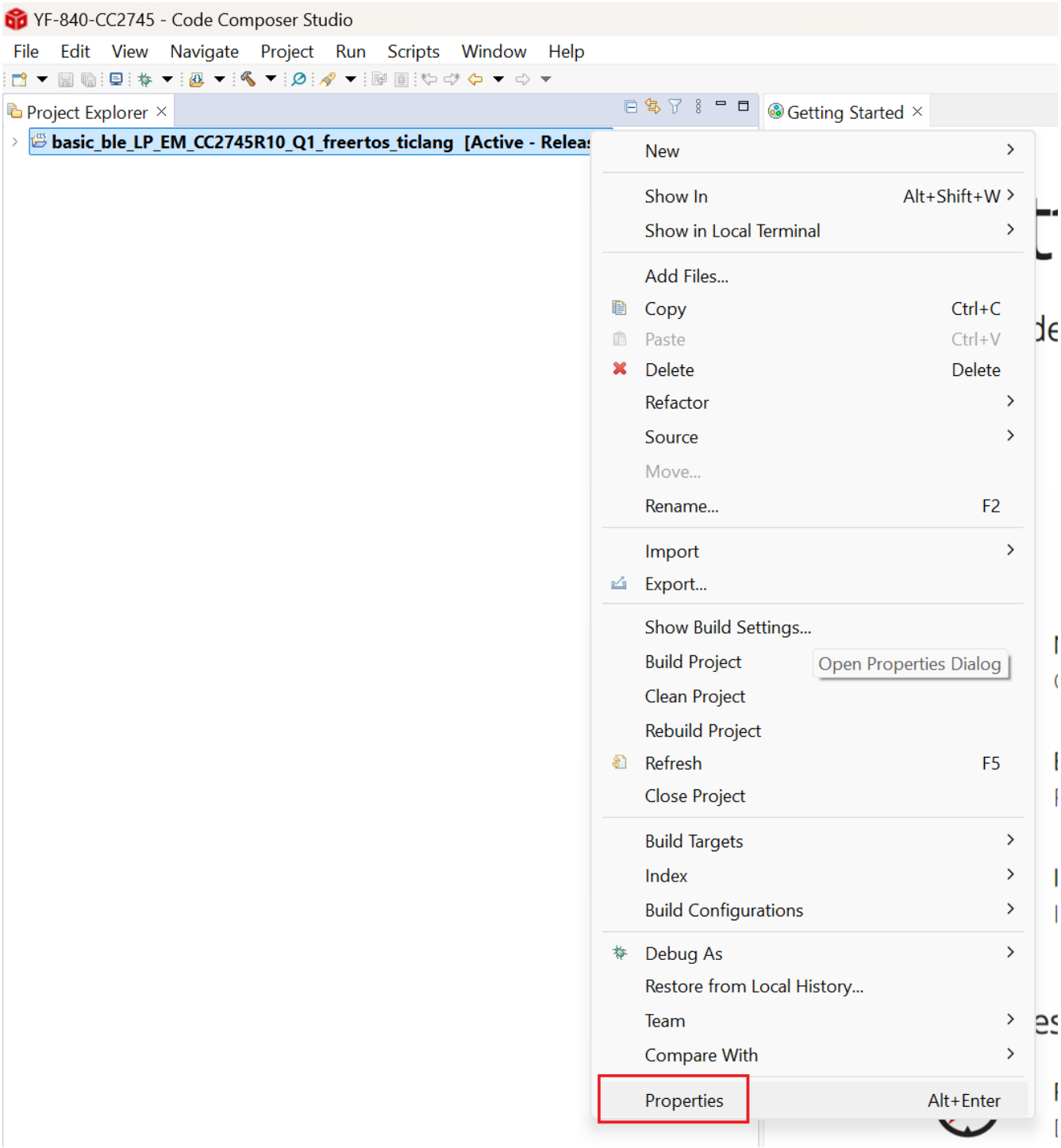


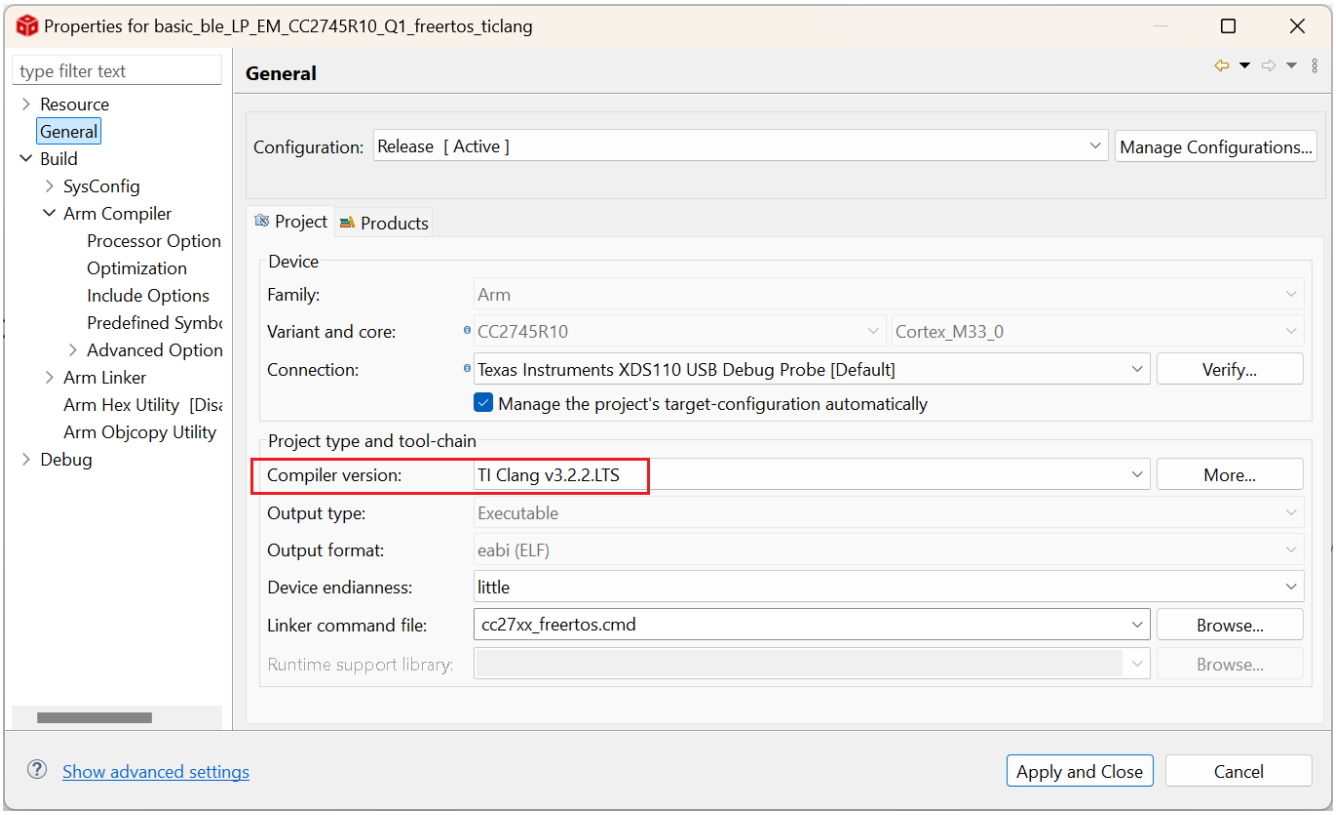


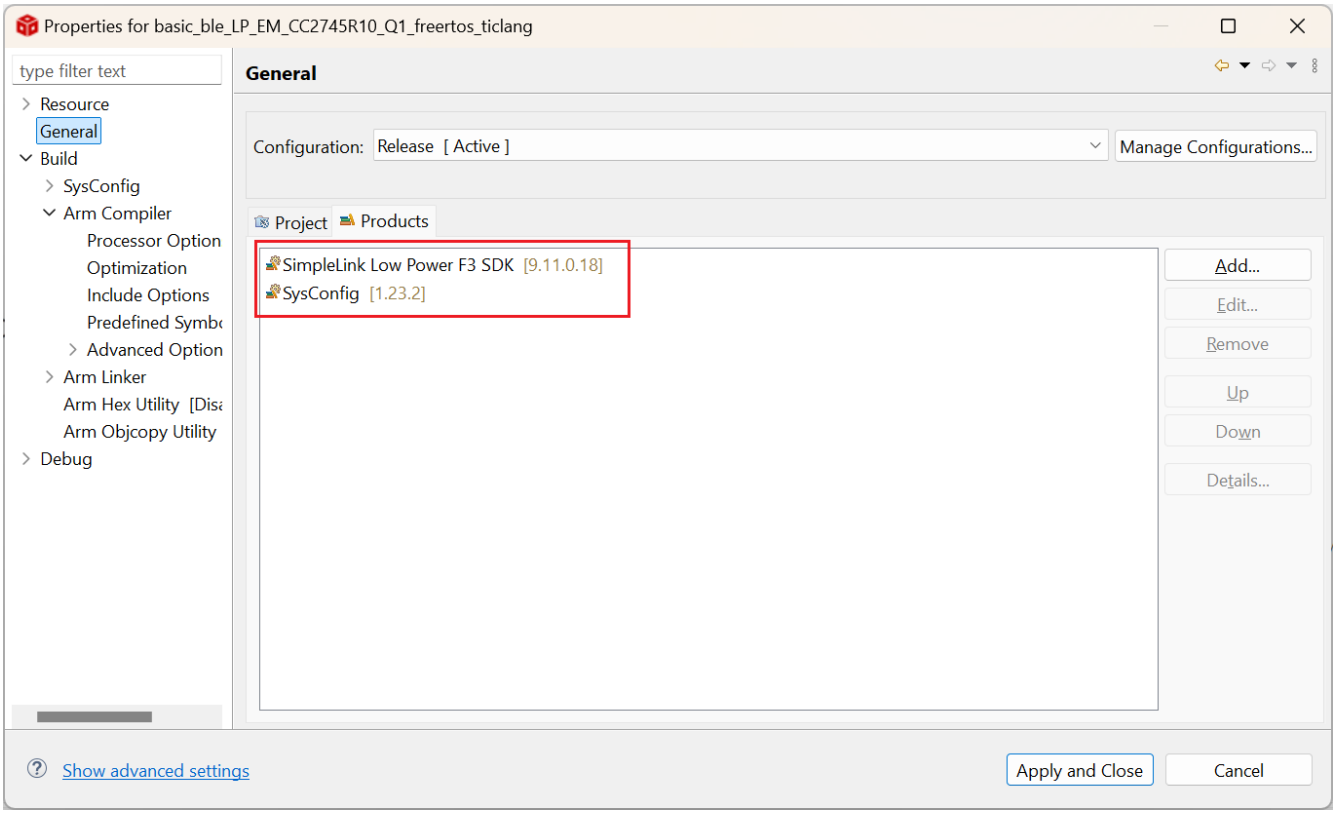
For issues that may occur in the code, it's also recommended to use ROV to check for stack overflow first, which is helpful in resolving the issue. The procedure is similar, but select System Stack from the ROV Module Name list and check the execution of Tasks at the same time. If the System Stack indicates a Stack Overflow, proceed to examine the Tasks.

2.3 How to Port Your Project in CCS 12.x to CCS 20.x for Debugging

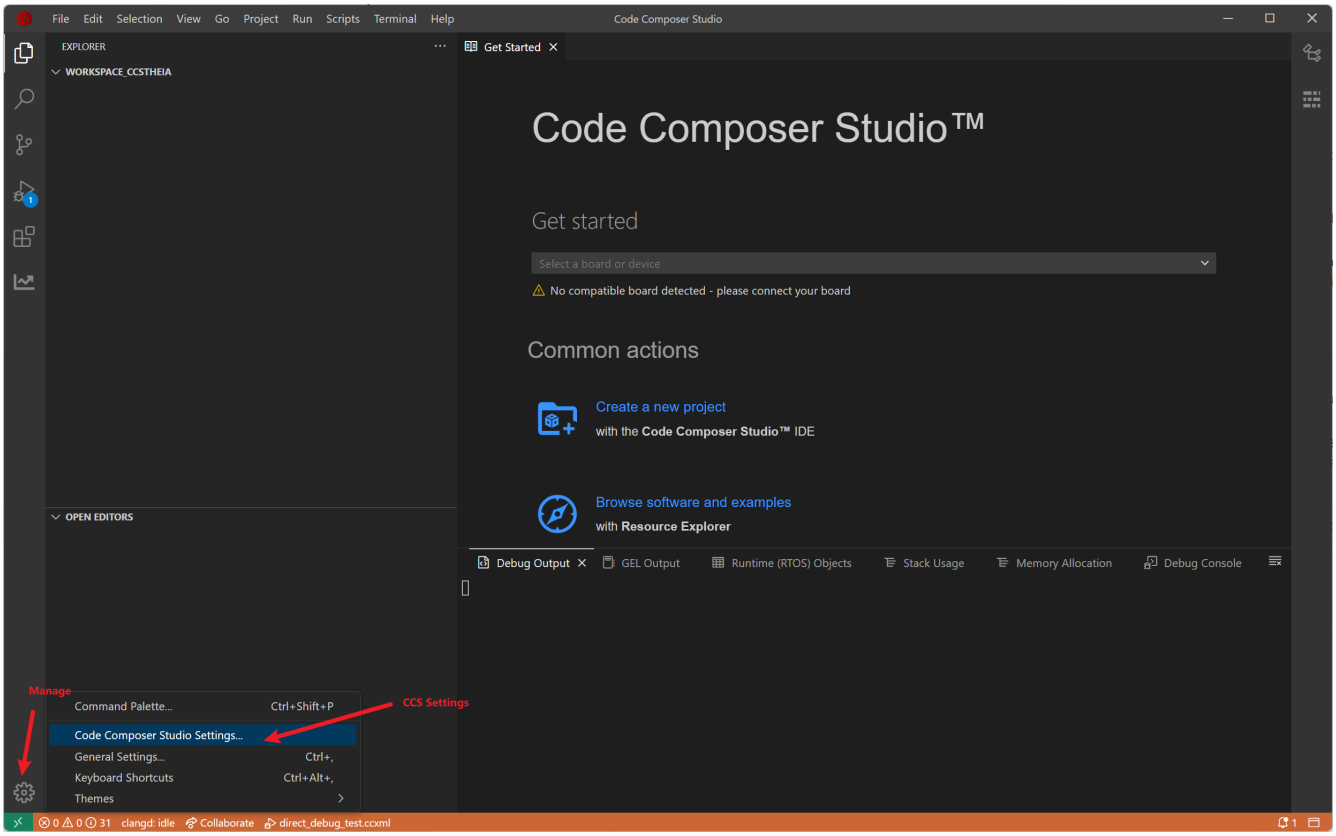
To port a project, always check if the project's dependencies are available in both versions of CCS. Therefore, the first step is to open the project.

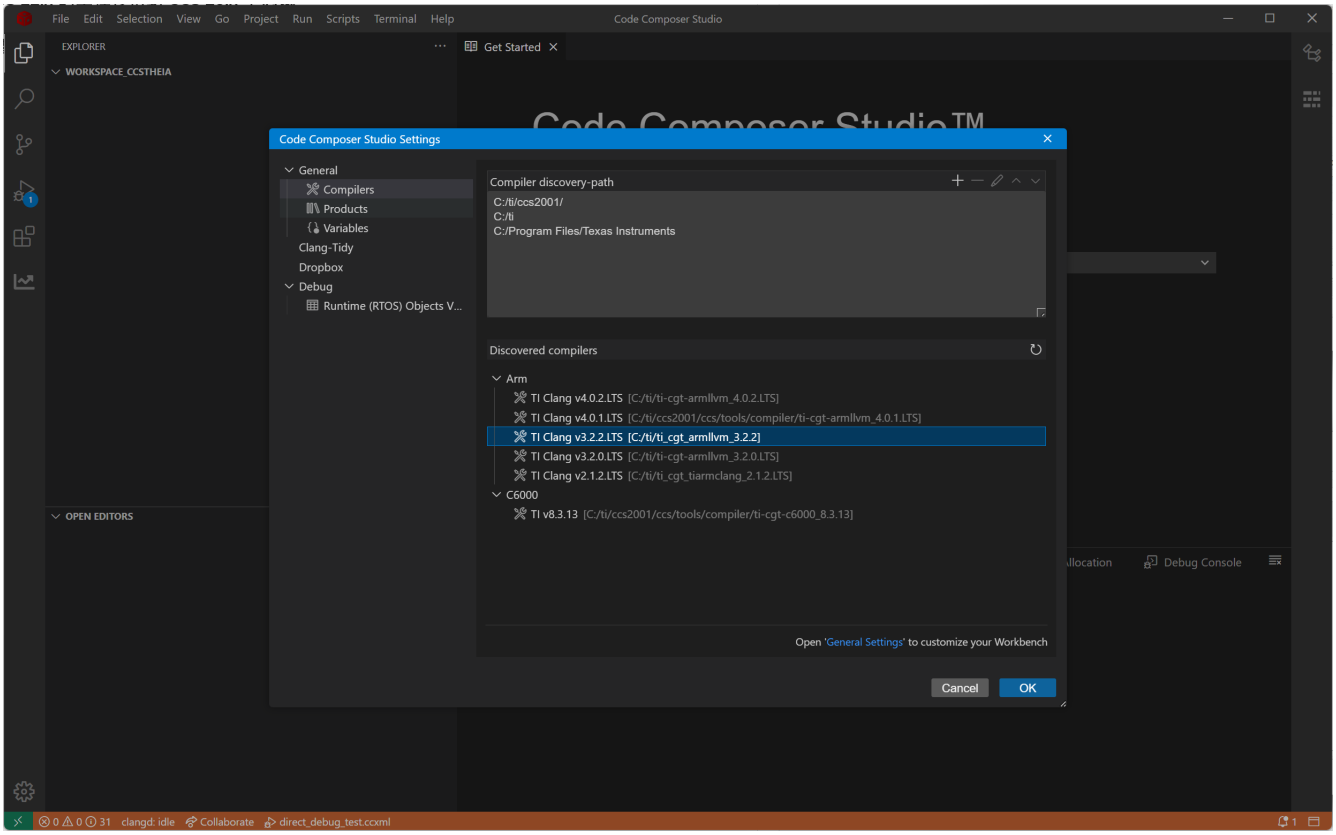


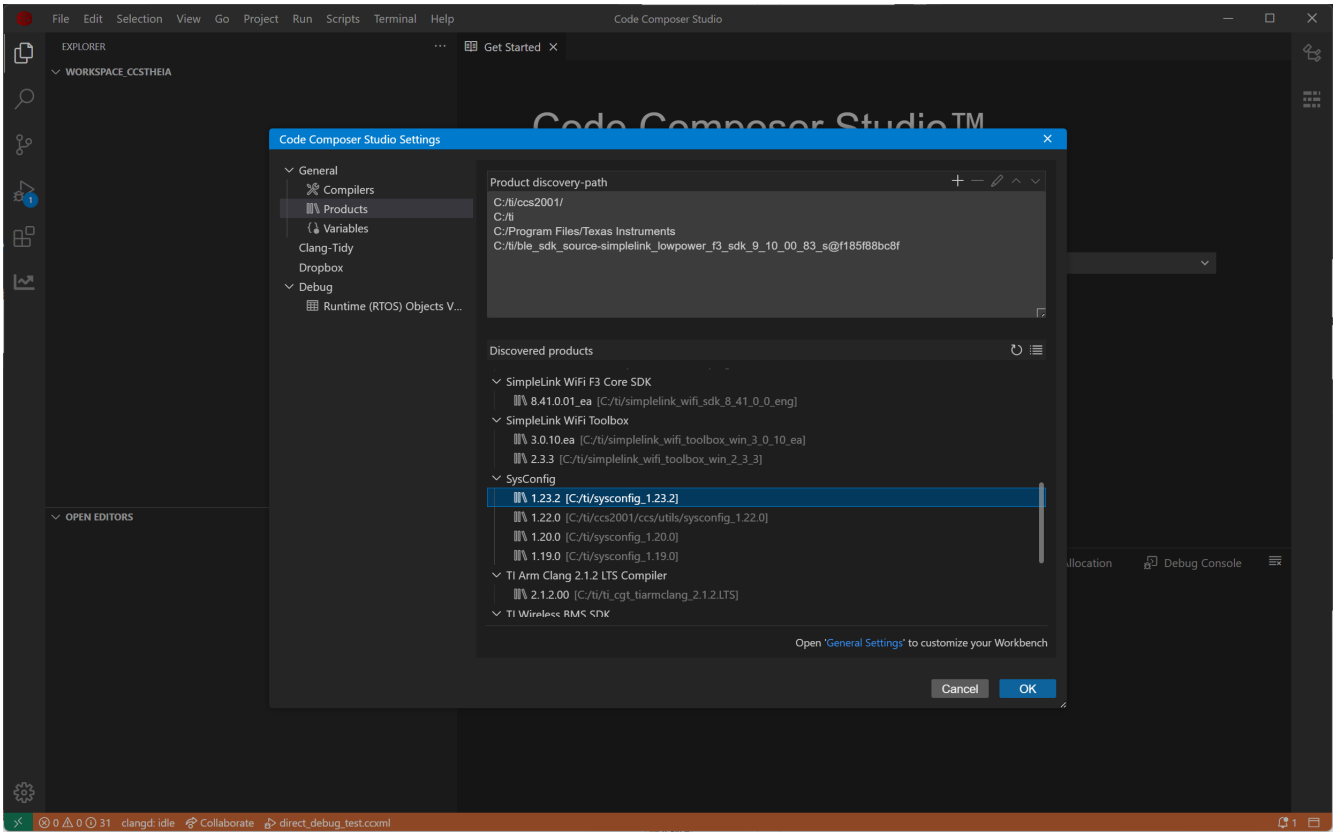




In CCS 20.0, go to Manage and Code Composer Studio Settings to check if the same dependencies are present.

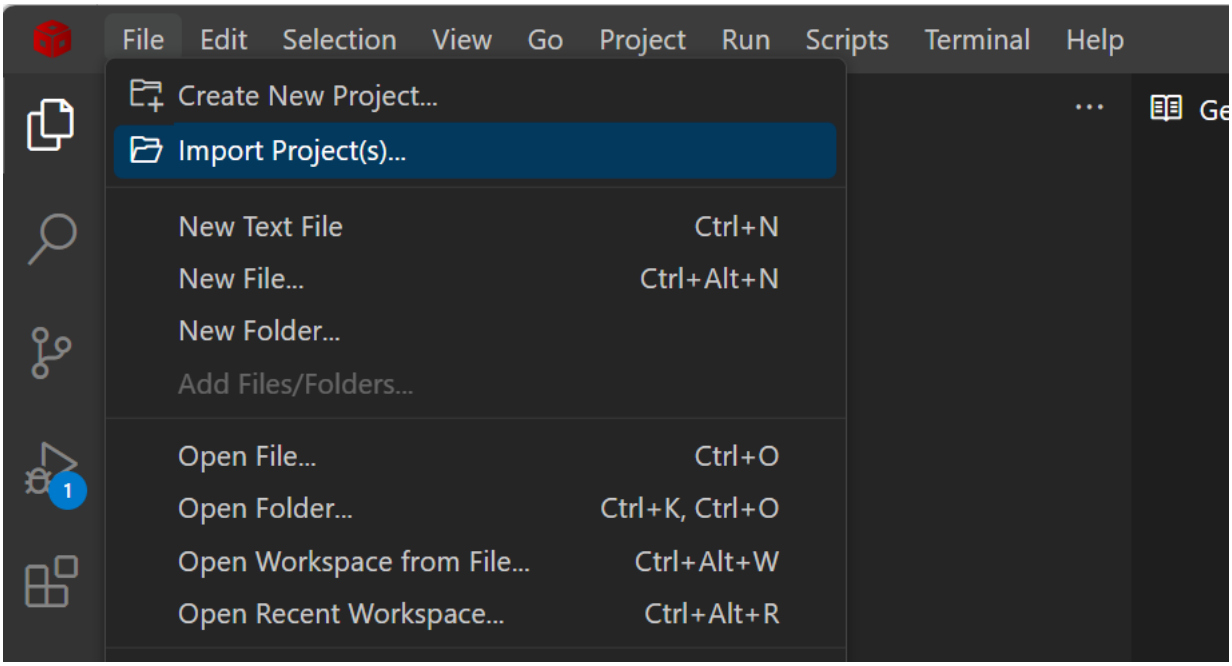




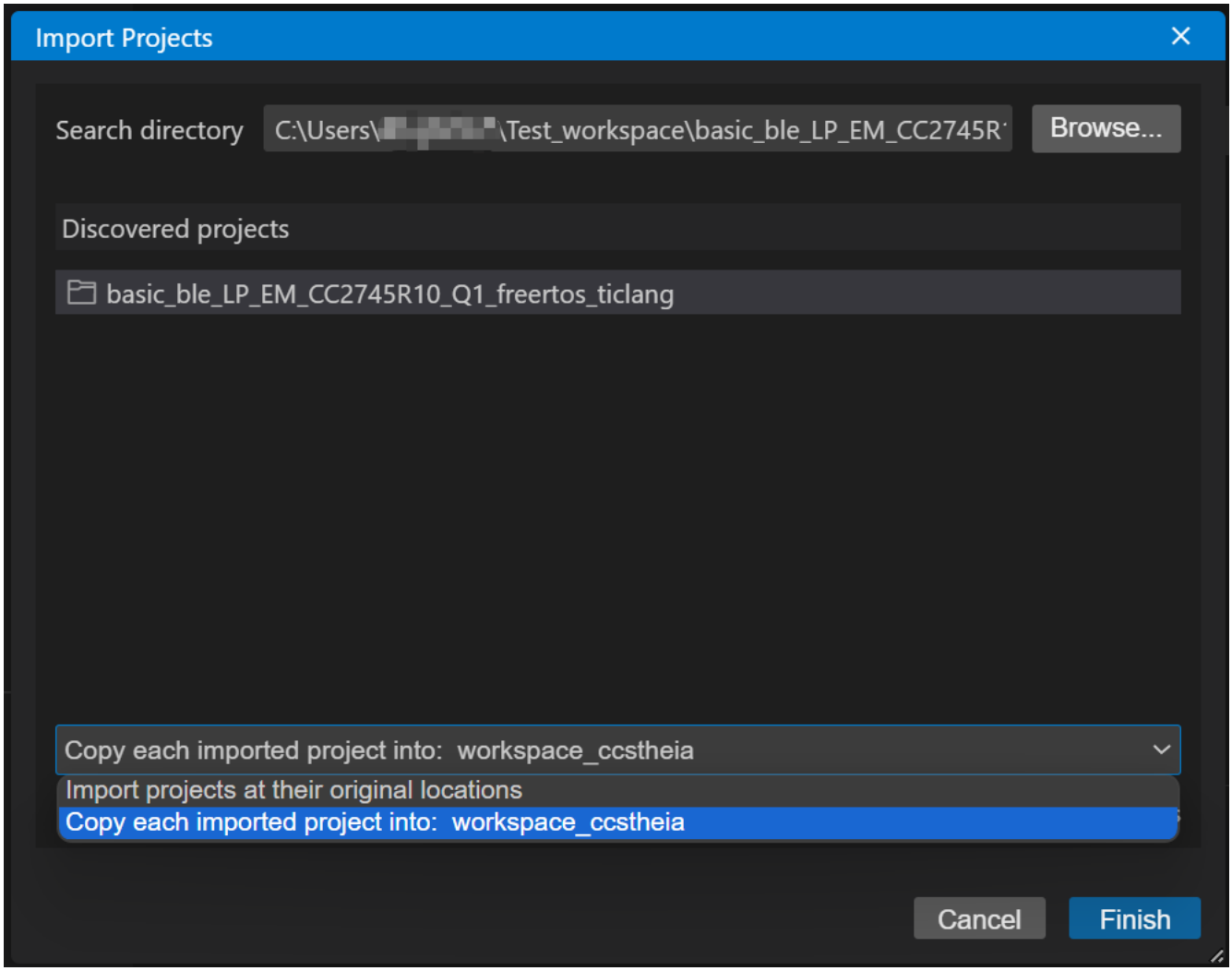


If they are missing, visit the official website to download and install them. After a successful installation, click the refresh icon to the right of Discovered Products.

If the dependencies are already in place, manually copy the project into the CCS 20.x workspace.



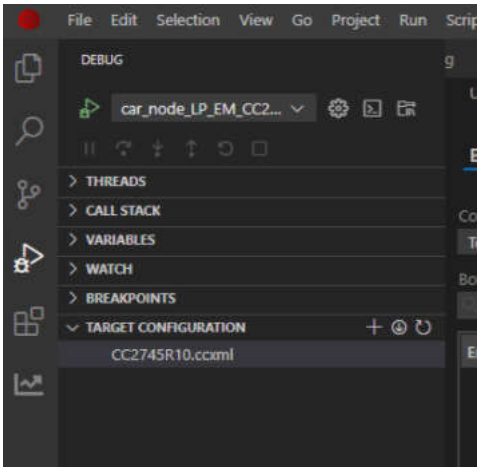
Select Copy each imported project into: xxx_workspace.



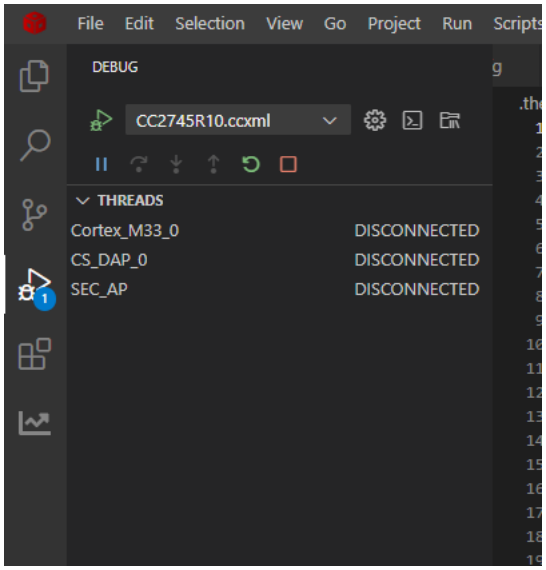
2.4 Performing Chip Erase on the CC2745

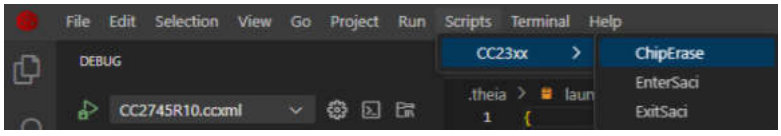
UniFlash has removed the chip erase functionality for the CC2745, which often causes confusion for engineers who need to verify the residual flash status or delete application code, including pairing records. We can perform a chip erase in CCS.

In CCS20.x, go to Debug → Target Configurations, click the second icon on the right to import the target configuration, select the targetConfigs folder of any CC2745 project in the workspace, right-click CC2745R10.ccxml, and select Start Project-less Debug.



In the THREADS window, right-click Cortex_M33_0 and select Show All Cores, then select CS_DAP_0 (**do not click to connect**).

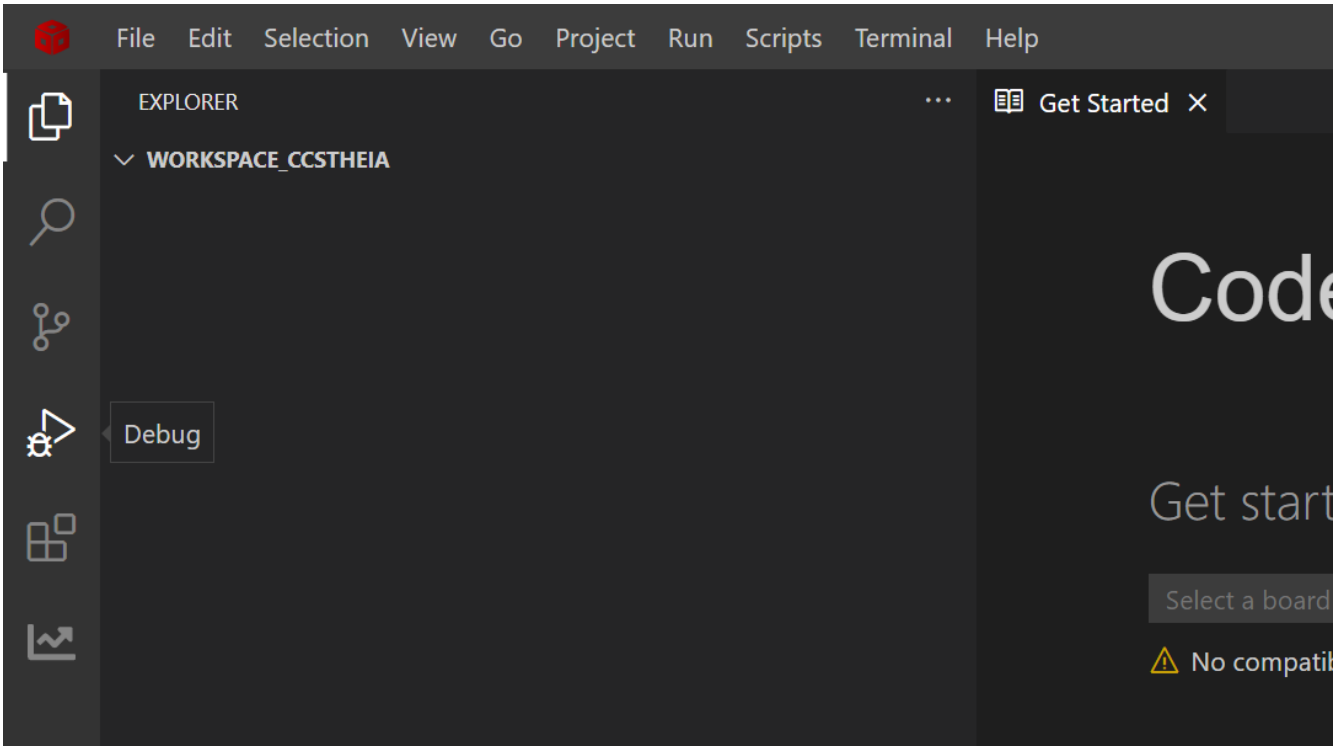




Click Scripts → CC23xx → Chip Erase. There is no need to distinguish the specific model as CC23xx is TI's unified family of automotive wireless connectivity chips. Click Scripts → CC23xx → Chip Erase. There is no need to distinguish the specific model as CC23xx is TI's unified family of automotive wireless connectivity chips.

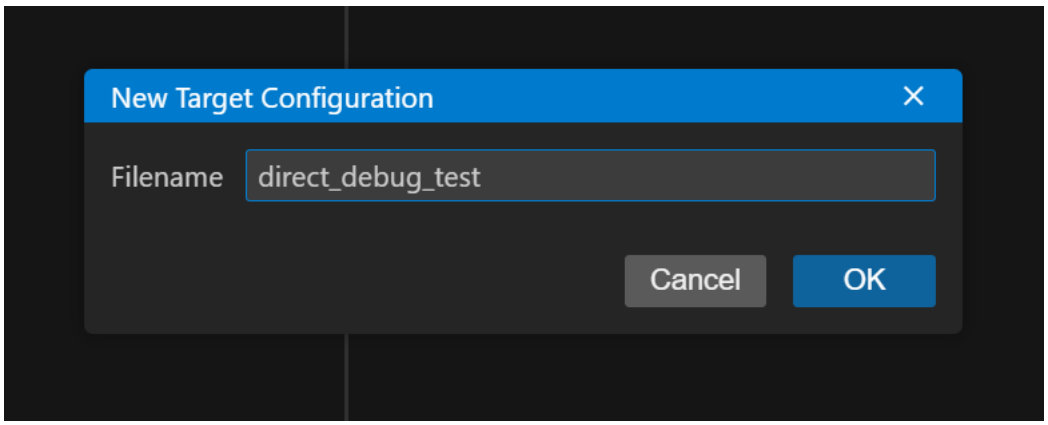
2.5 How to Debug a Project (.out) Directly in CCS 20.x Without Importing Project Code

Open CCS 20.x, and click the Debug icon in the left bar.

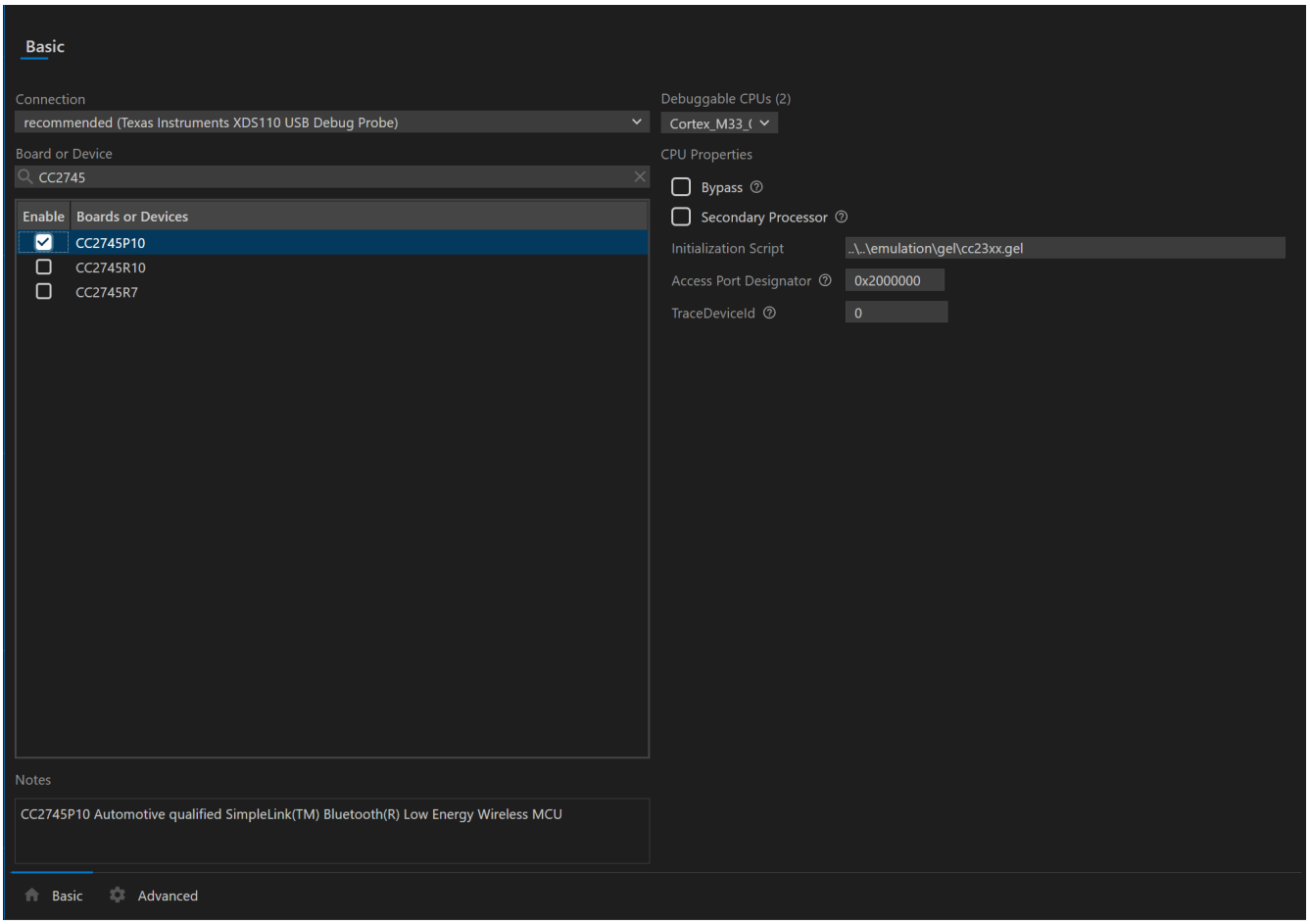


The Debug view appears, listing Thread, Call Stack, Variables, Watch, Breakpoints, and Target Configuration from top to bottom. Click the + (New Target Configuration) to the right of Target Configuration.

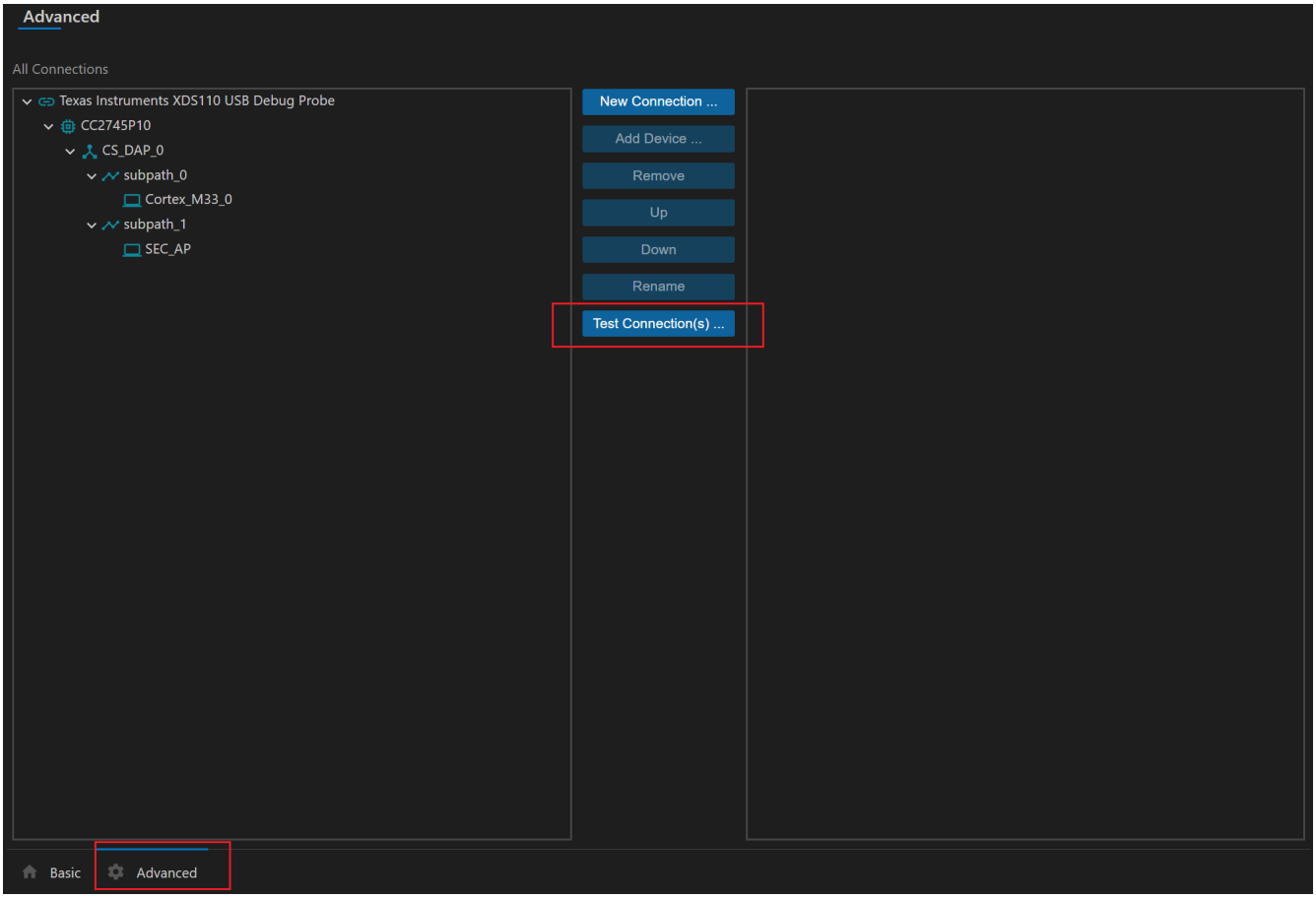
Name the configuration profile (.ccxml) and click OK.



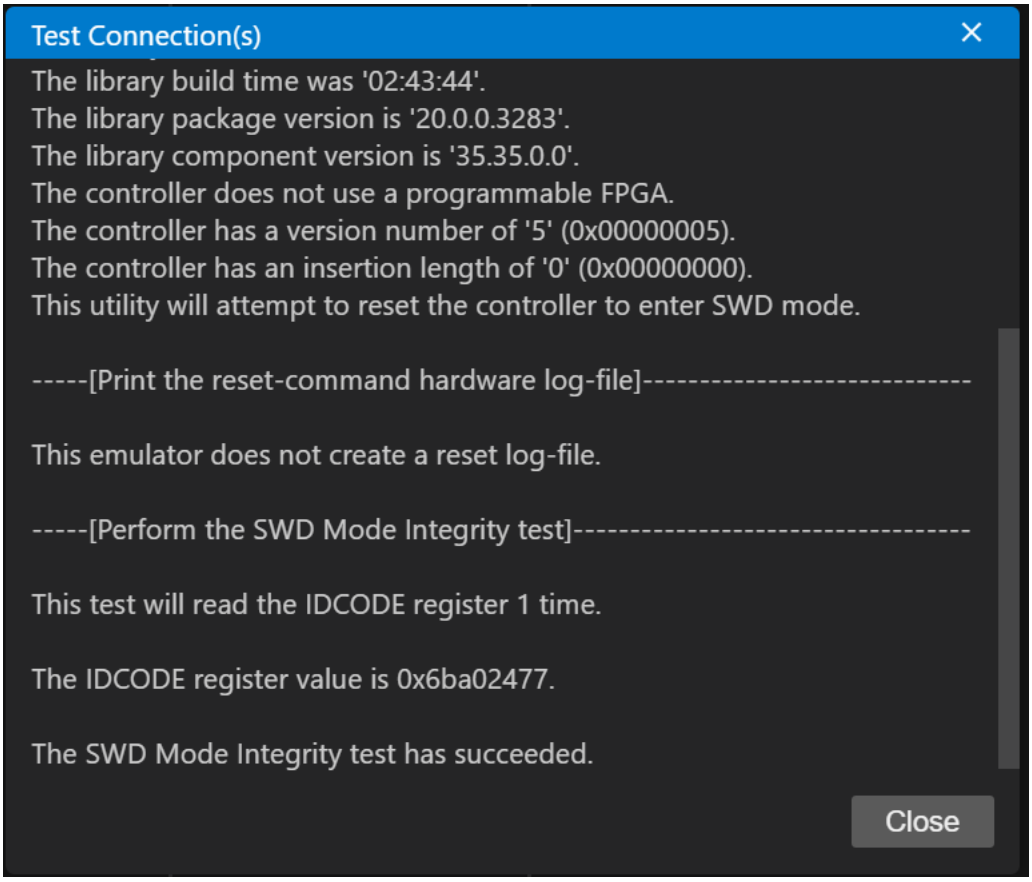
Then a Basic file appears. The emulator used in the example is TI's XDS110. For Connection, select Texas Instruments XDS110 USB Debug Probe. Depending on the chip or board family you are using, search for CC2340 or CC2745 in Board or Device. In this example, CC2745R10 is selected.



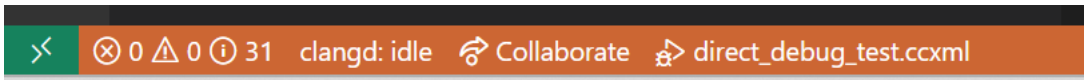
Now connect your board to the computer using the XDS110. To test your configuration, click Advanced at the bottom, and then Test Connection to test the communication with the board.

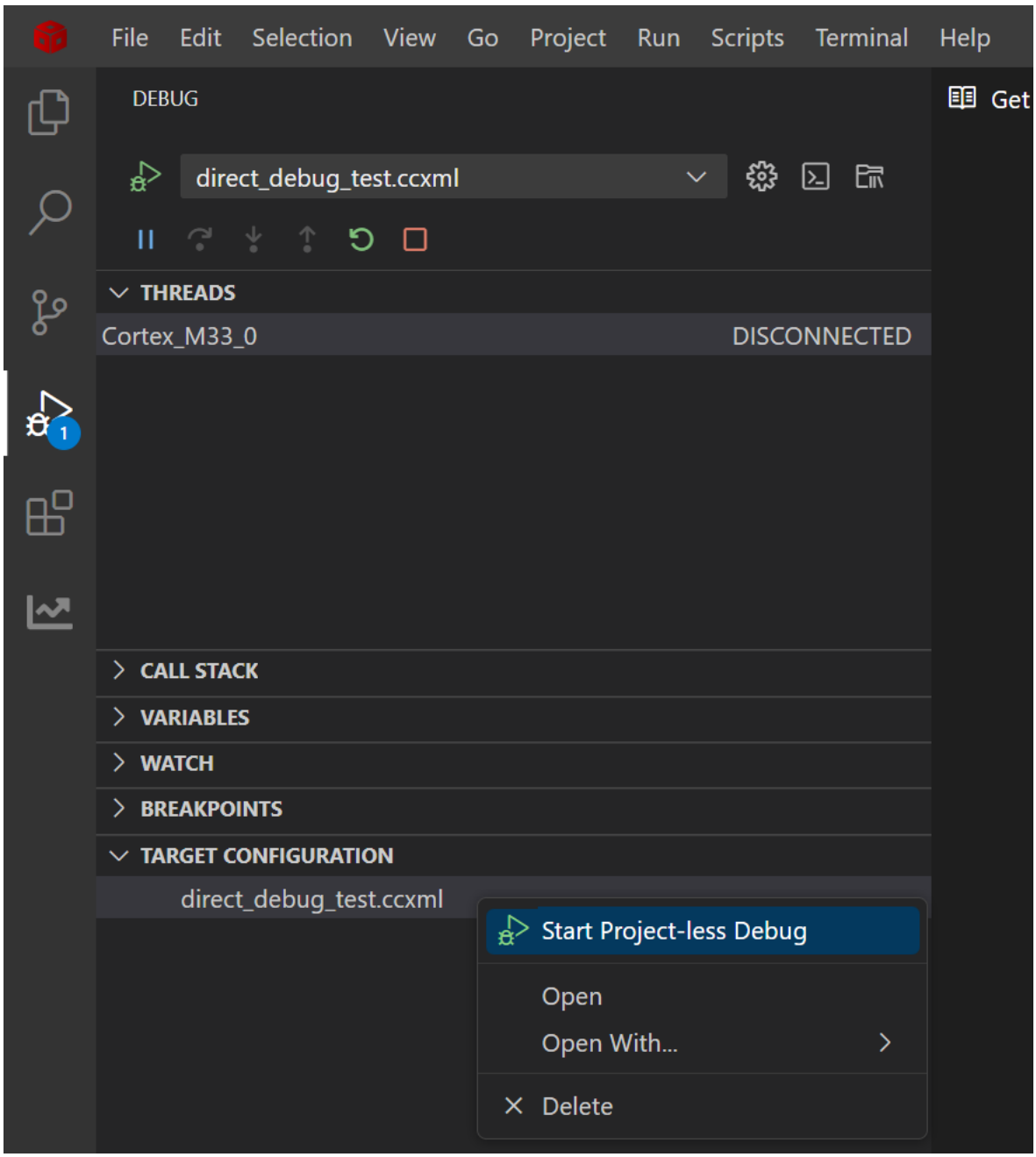


If the following interface with logs is displayed, it is verified that the connection to the board is successful.

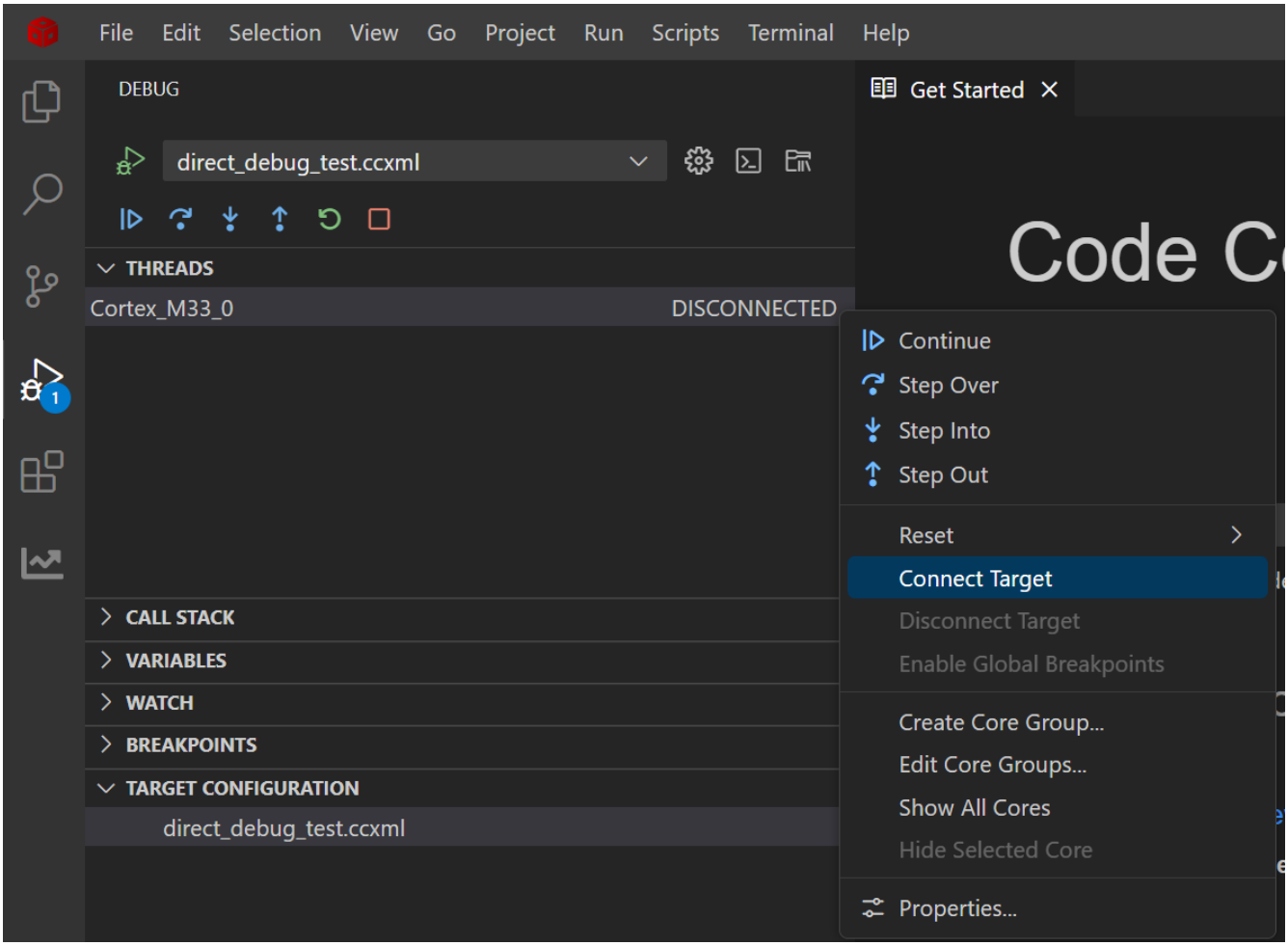


Next, right-click the newly created .ccxml file and click Start Project-less Debug. You will find that the corresponding configuration has been launched at the bottom of the CCS view, and the status is idle.

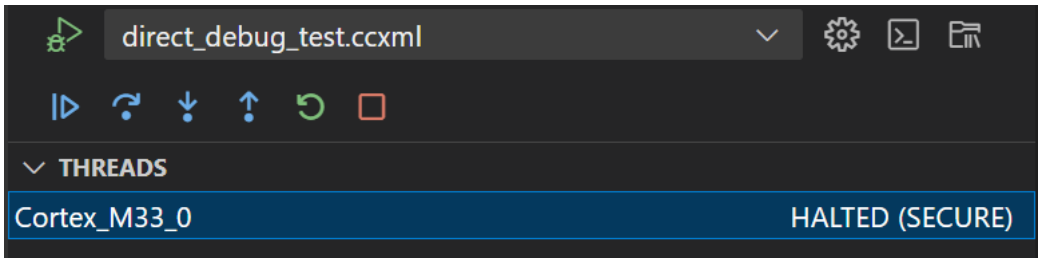




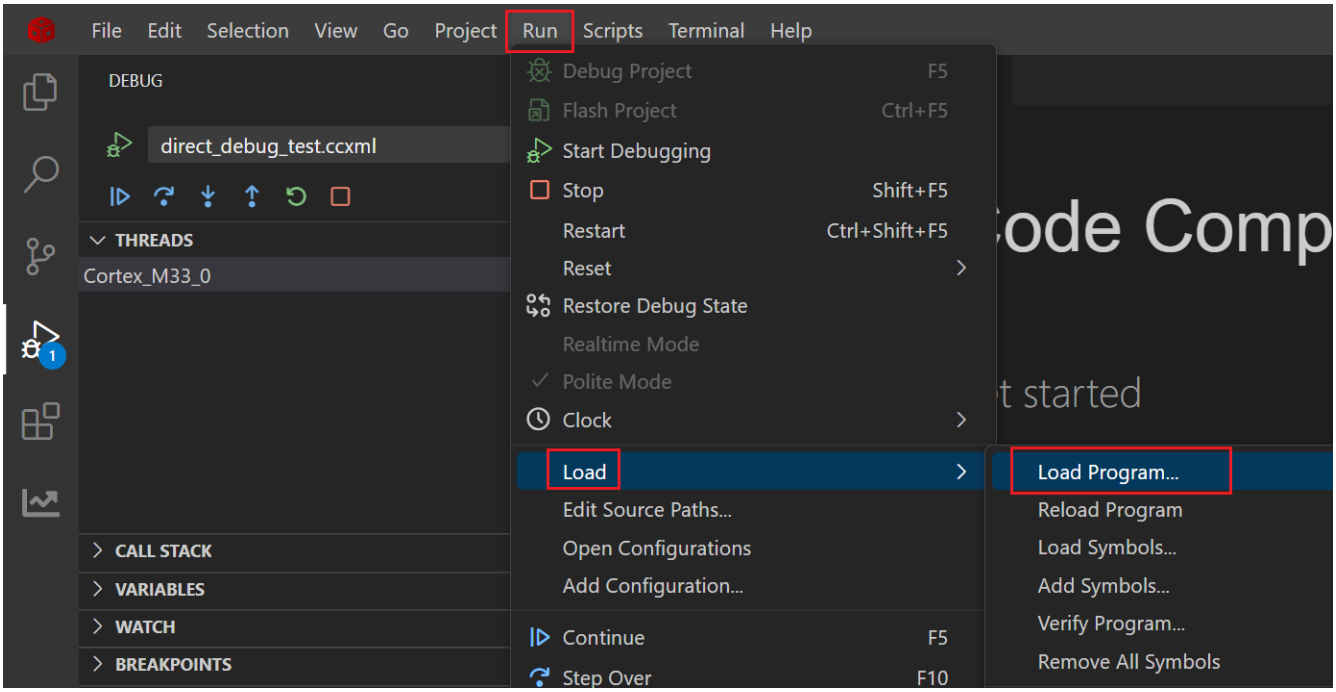
Now Cortex_M33_0 is disconnected. Right-click on Cortex_M33_0 and click Connect Target.



The connection status changes to HALTED (SECURE).

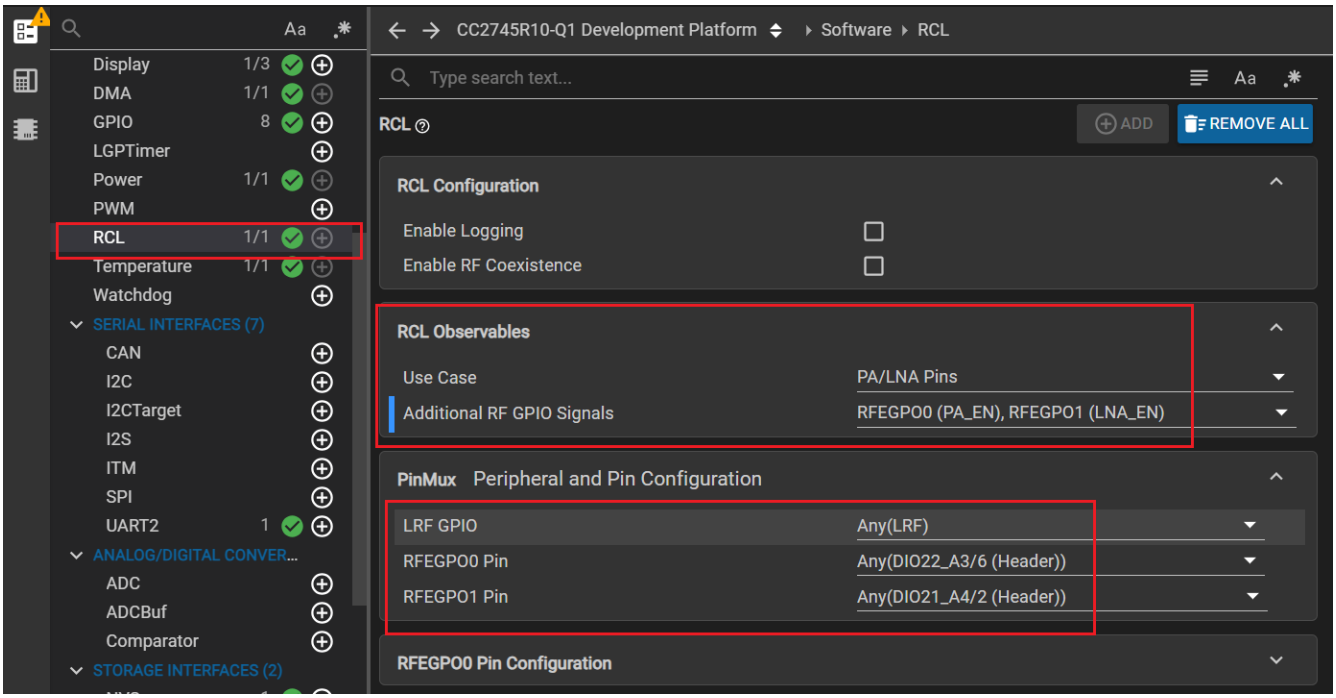


Then click Run → Load → Load Program… Select the .out file you want to debug and load it.

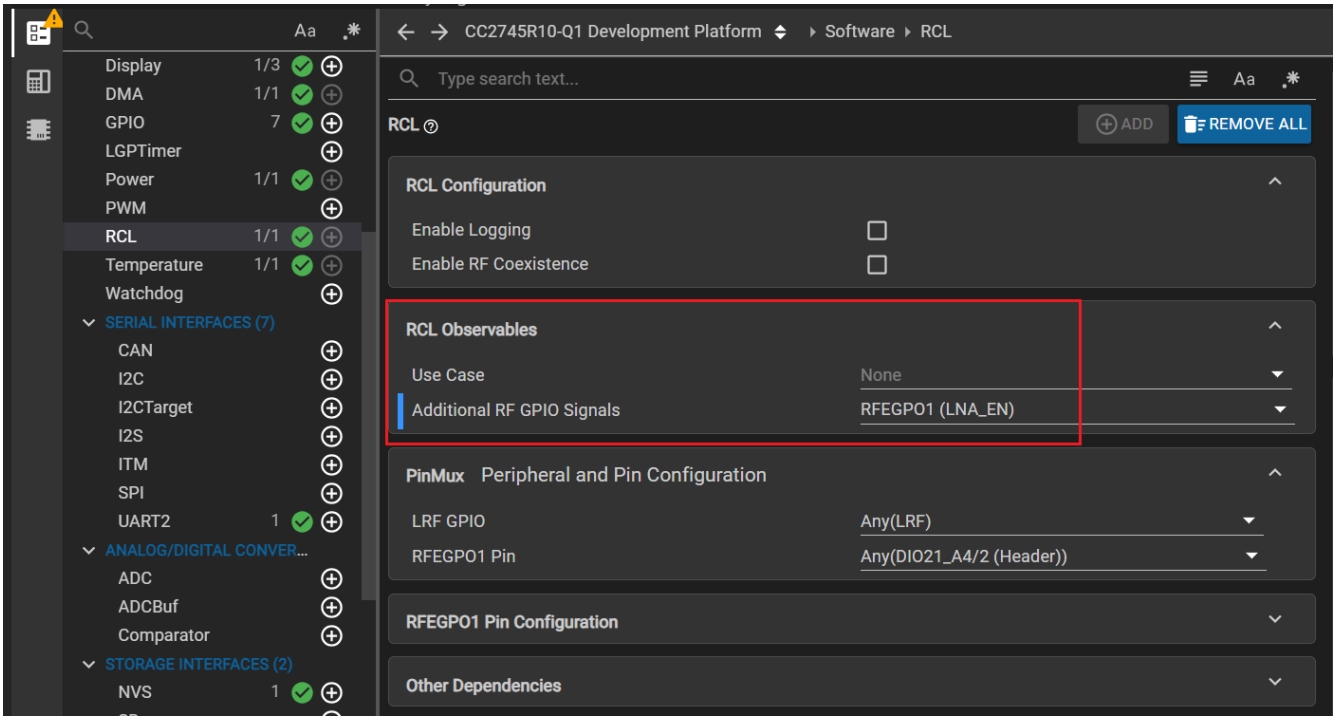


2.6 How to Verify Whether the Chip's RF Is Working Properly

The CC23xx and CC27xx families of chips both support mapping internal RF signals to pins to observe RF operation. There are two observable signals, which are also the two most commonly used: LNA and PA, where LNA corresponds to the receiver (Rx) and PA corresponds to the transmitter (Tx). They are enabled as follows:



It is shown that the PA_EN and LNA_EN signals have been mapped to DIO22 and DIO21. Additionally, if no extra IOs are available on your board, or if one signal is enough to identify the issue, deselect PA/LNA Pins in the Use Case, then directly select the desired signal and the corresponding pin for mapping to connect a logic analyzer for observation.



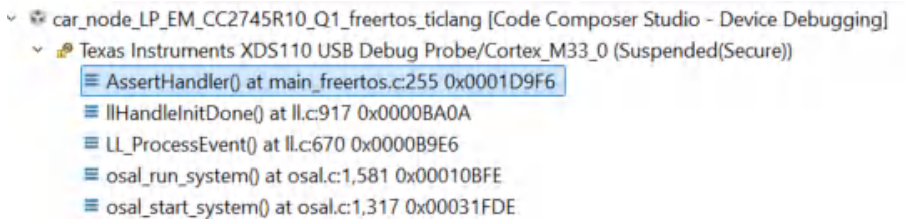
It is important to note that only the LRF Digital Output pins are available for use. Therefore, when designing custom boards, consider RF signal debugging requirements and reserve or reuse these pins accordingly.

LRF Digital Output	LRFD0	19	I/O	O	LRF digital output 0
	LRFD1	20			LRF digital output 1
	LRFD2	21			LRF digital output 2
	LRFD3	22			LRF digital output 3
	LRFD4	23			LRF digital output 4
	LRFD5	24			LRF digital output 5
	LRFD6	32			LRF digital output 6
	LRFD7	33			LRF digital output 7

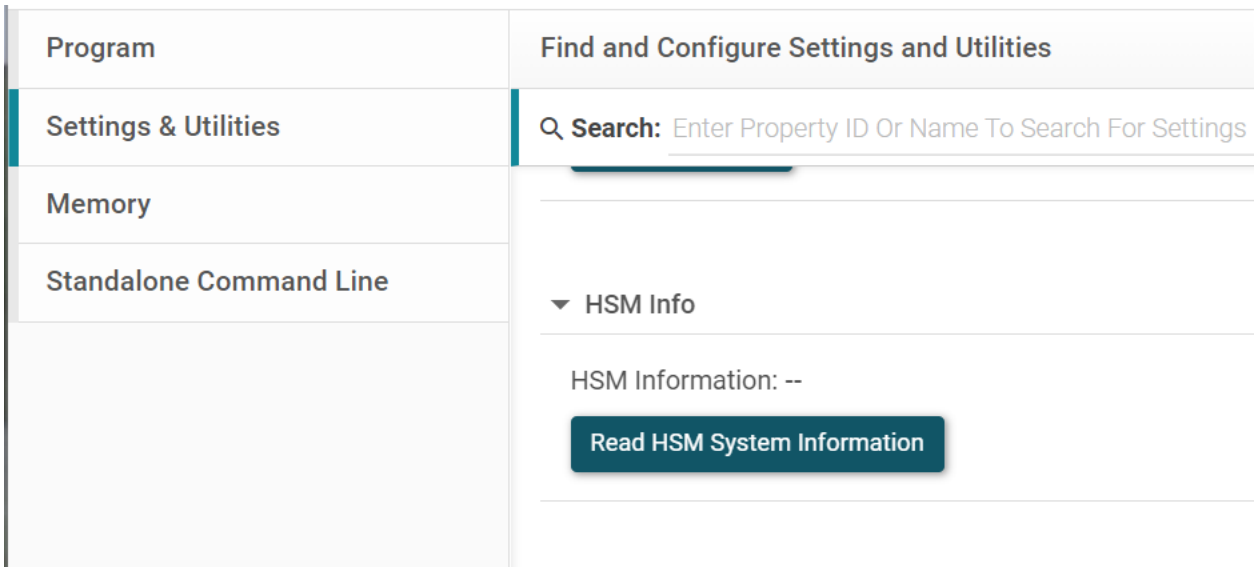
3 FAQs for Development Using CC2340 and CC2745

3.1 Why Does the Program Fail to Run Correctly Although It Has Been Successfully Flashed via UniFlash or CCS

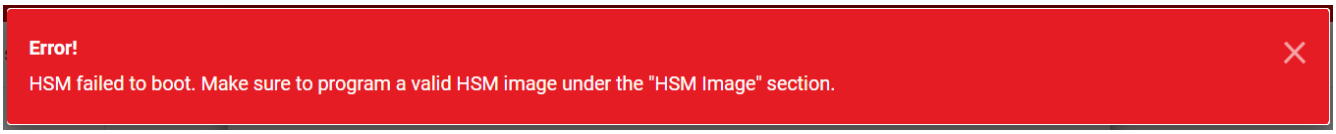
A common issue encountered when using the CC2745 for the first time is that the program fails to run correctly on the chip, even when debugged using the official routines included with the TI SDK or when a pre-compiled hex file is directly flashed. For example, when CCS 12.8.1 is used to build the basic_ble_LP_EM_CC2745R10_Q1_freeertos_ticlang project, the assert below is reported:



The reason is that the CC2745 blank board requires the HSM firmware to be flashed first for proper initialization; otherwise, the chip fails to start properly. It is recommended that engineers perform an HSM firmware check before flashing the chip. Go to UniFlash → Settings & Utilities → Read HSM System Information.



An error message appears when the HSM firmware is missing.



The solution is to locate the HSM firmware in the SDK at *<SDK path>\bin\hsm\cc27xxx10_hsm_fw_v3.0.0.bin, enter this path into UniFlash -> Settings & Utilities -> HSM Image*, and click Program HSM Image to flash the HSM firmware.

3.2 How to Change the Tx Power for Bluetooth Devices

There are two ways to change the Tx Power:

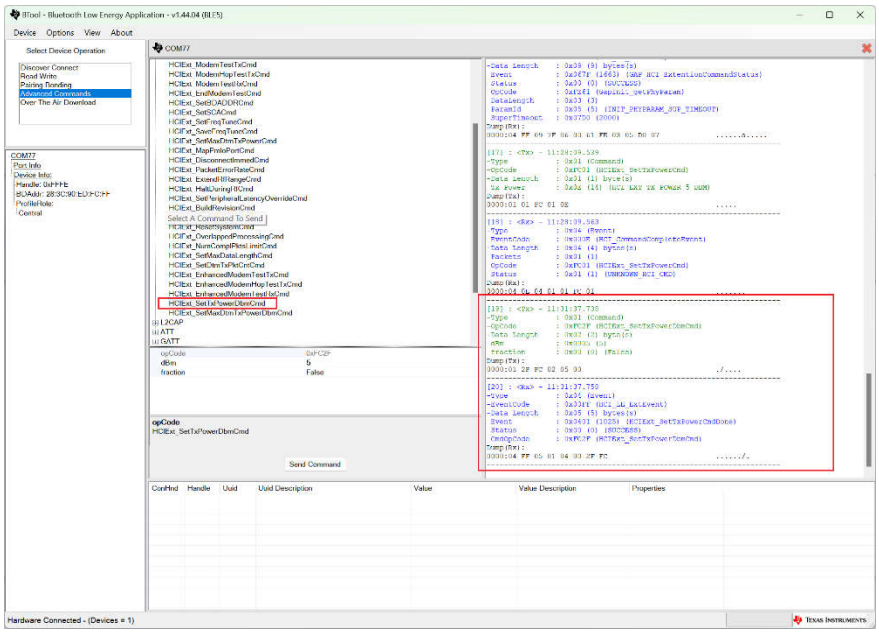
1. Use HCI_EXT_SetMaxDtmTxPowerDbmCmd() to change the power while the program is running
2. Use HCI_EXT_SetTxPowerDbmCmd() for BLE operations such as ADV/CONN

For the second case, BTool is a tool provided by TI for customers to conduct product validation tests. For hardware connections, see also the document Configuration Methods for BQB RF-PHY Testing and SRRC Fixed Frequency Testing of TI Bluetooth Products (Rev. A). Click here to access the BTool User Manual.

BTool path: <SDK_PATH>\tools\ble\btool

Host_test path: <SDK_PATH>\examples\rtos\LP_EM_CC2745R10_Q1\ble\host_test

It is important to note that the HCIExt_SetTxPowerCmd command in BTool is no longer used. Instead, use Advanced Commands → HCI → HCIExt_SetTxPowerDbmCmd to change the dBm at the bottom to your desired value.



3.3 Why Does My Program Enter iCall_abort

A common error when calling the Bluetooth API is that the program enters iCall_abort. There are two most common causes for iCall_abort.

- The Bluetooth API is directly called from a custom task
- The Bluetooth API is directly called from an interrupt handler or callback function

To avoid entering iCall_abort, use the BLEAppUtil_invokeFunction function for context switching. The BLEAppUtil_invokeFunction function passes the function pointer of the API to be called into the message queue as an argument. The BLEAppUtil task then reads the message queue and executes the corresponding API.

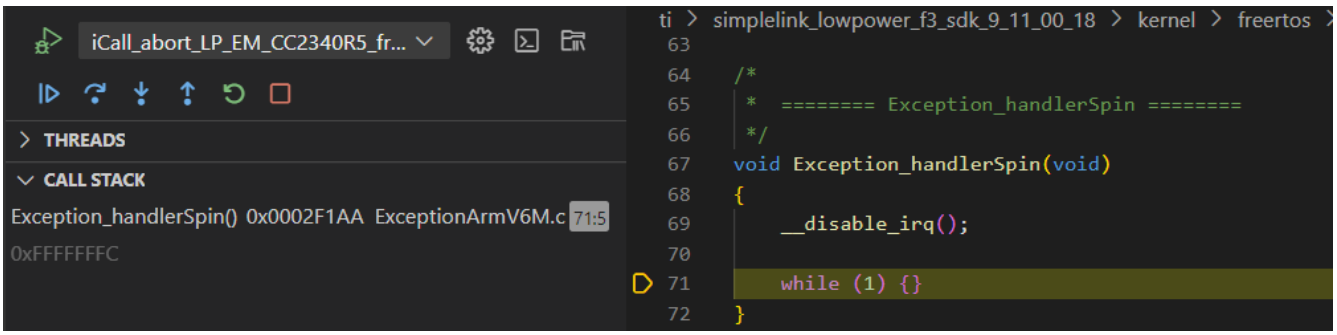
The following is a typical case where the user creates a new task and calls SimpleGatt_notifyChar4 from the main function (myTaskMain) of the task to send Bluetooth data. In this scenario, the user should use BLEAppUtil_invokeFunction to switch contexts, instead of directly calling SimpleGatt_notifyChar4:

```
void *myTaskMain(void *arg)
{
    while(1)
    {
        if(sendNotifyTrigger == 1)
        {
            // Call BLE API directly from a non-iCall task will cause iCall_abort
            //SimpleGatt_notifyChar4();
            // Using BLEAppUtil_invokeFunctionNoData to avoid iCall_abort
            BLEAppUtil_invokeFunctionNoData(SimpleGatt_notifyChar4);

            sendNotifyTrigger = 0;
        }
    }
}
```

Another point to note when using the BLEAppUtil_invokeFunction function is that its second argument, pData, must point to a memory region allocated by BLEAppUtil_malloc. Do not pass the address of a global variable or a local variable directly. In the example below:

```
if(sendParamUpdateReqTrigger == 1)
{
    bStatus_t status;
    gapUpdateLinkParamReq_t pParamUpdateReq =
    {
        .connectionHandle = 0,
        .intervalMin = 400,
        .intervalMax = 800,
        .connLatency = 0,
        .connTimeout = 600
    };

    // Send a connection param update request
    BLEAppUtil_invokeFunction((InvokeFromBLEAppUtilContext_t)BLEAppUtil_paramUpdateReq,
                              (char *)&pParamUpdateReq);
}
```

Since the address of pParamUpdateReq is passed directly into BLEAppUtil_invokeFunction, it is highly prone to memory errors upon release (the underlying cause is the specific implementation of malloc when the Memory Advanced feature is enabled), which leads to an entry into Exception_handlerSpin:



The correct calling procedure is: Create a pointer variable, allocate a memory region for it using BLEAppUtil_malloc, and pass the pointer as an argument into BLEAppUtil_invokeFunction. By doing so, it can be released safely.

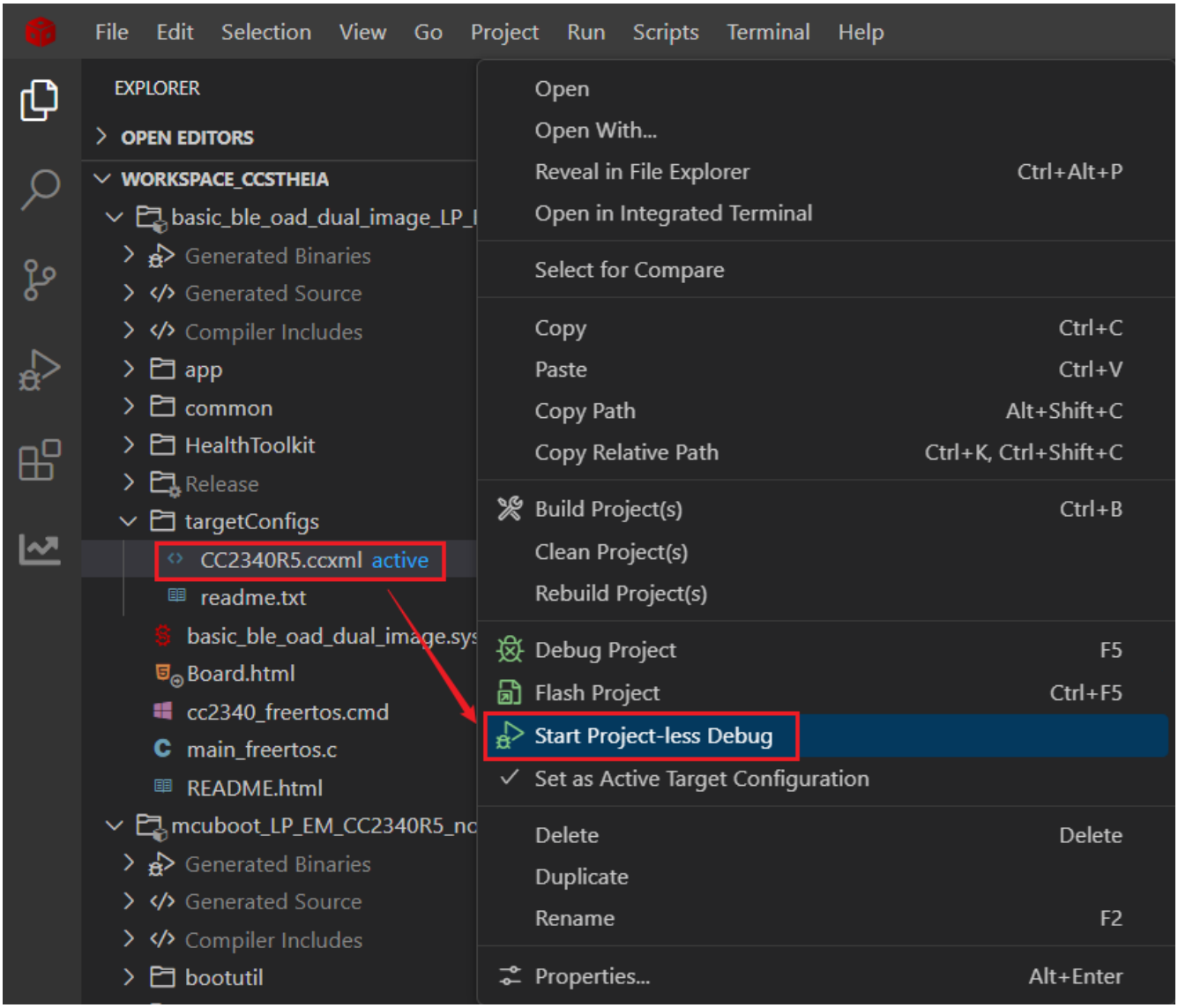
```
if(sendParamUpdateReqTrigger == 1)
{
    bStatus_t status;
    gapUpdateLinkParamReq_t *pParamUpdateReq = BLEAppUtil_malloc(sizeof(gapUpdateLinkParamReq_t));
    pParamUpdateReq->connectionHandle = 0;
    pParamUpdateReq->intervalMin = 400;
    pParamUpdateReq->intervalMax = 800;
    pParamUpdateReq->connLatency = 0;
    pParamUpdateReq->connTimeout = 600;

    // Send a connection param update request
    BLEAppUtil_invokeFunction((InvokeFromBLEAppUtilContext_t)BLEAppUtil_paramUpdateReq,
                              (char *)pParamUpdateReq);
}
```

3.4 How to Debug a Project with MCUboot

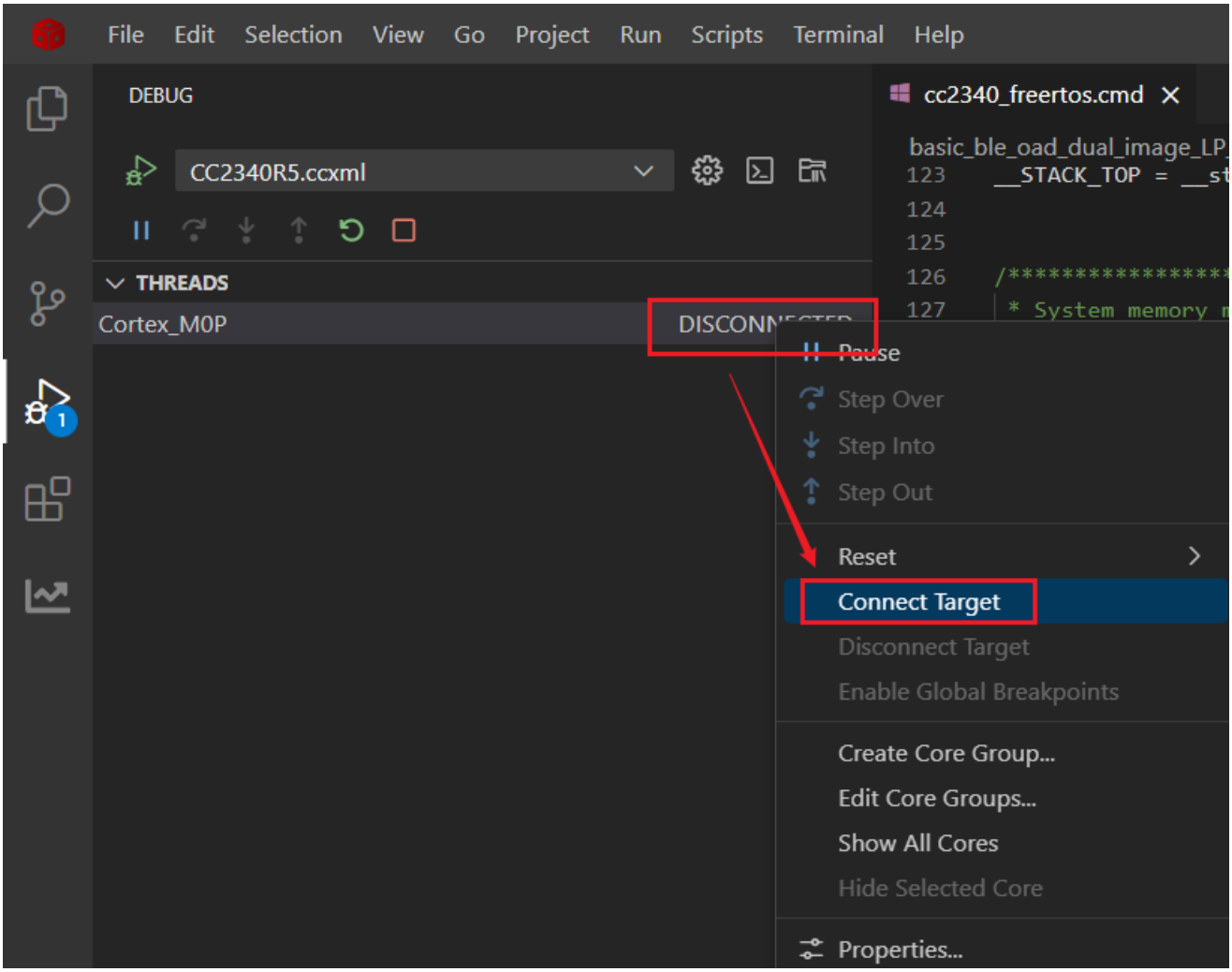
Some programs require multiple images to be flashed simultaneously, such as programs with the over-the-air (OTA) upgrade functionality, which must be flashed together with MCUboot to function properly. Such projects can be debugged in CCS by following the steps below.

Take the MCUboot and basic_ble_oad_dual_image projects in CC2340R5 as an example:

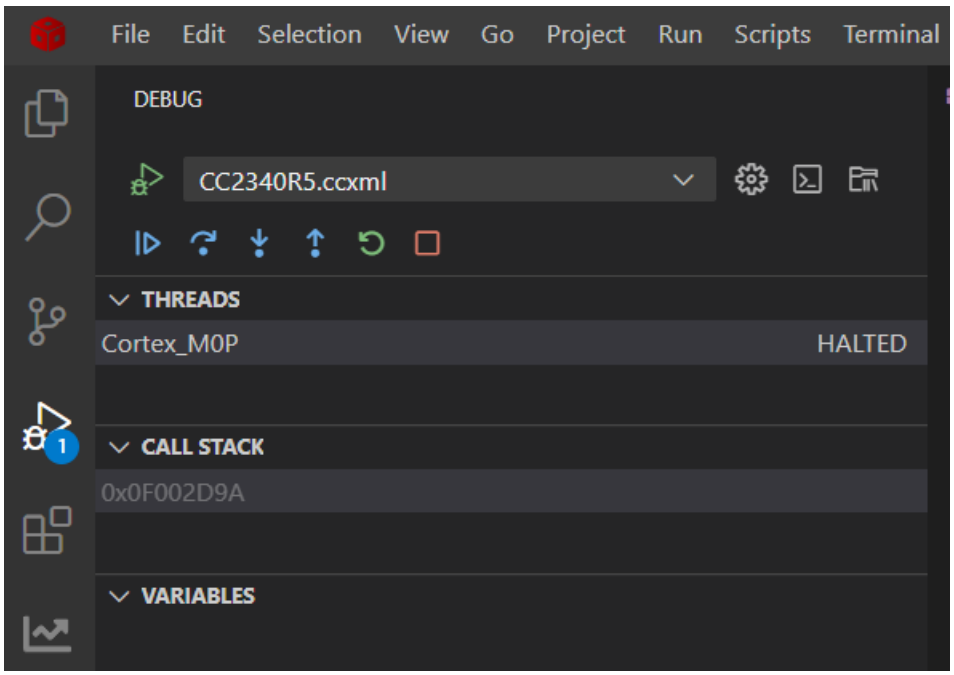
1. First, use UniFlash to flash both the MCUboot and basic_ble_oad_dual_image projects into the CC2340 simultaneously. For the detailed procedure, see also OAD Target Setup.
2. Right-click the targetConfigs/CC2340R5.ccxml file in the MCUboot project or the basic_ble_oad_dual_image project, and select Start Project-less Debug from the menu:



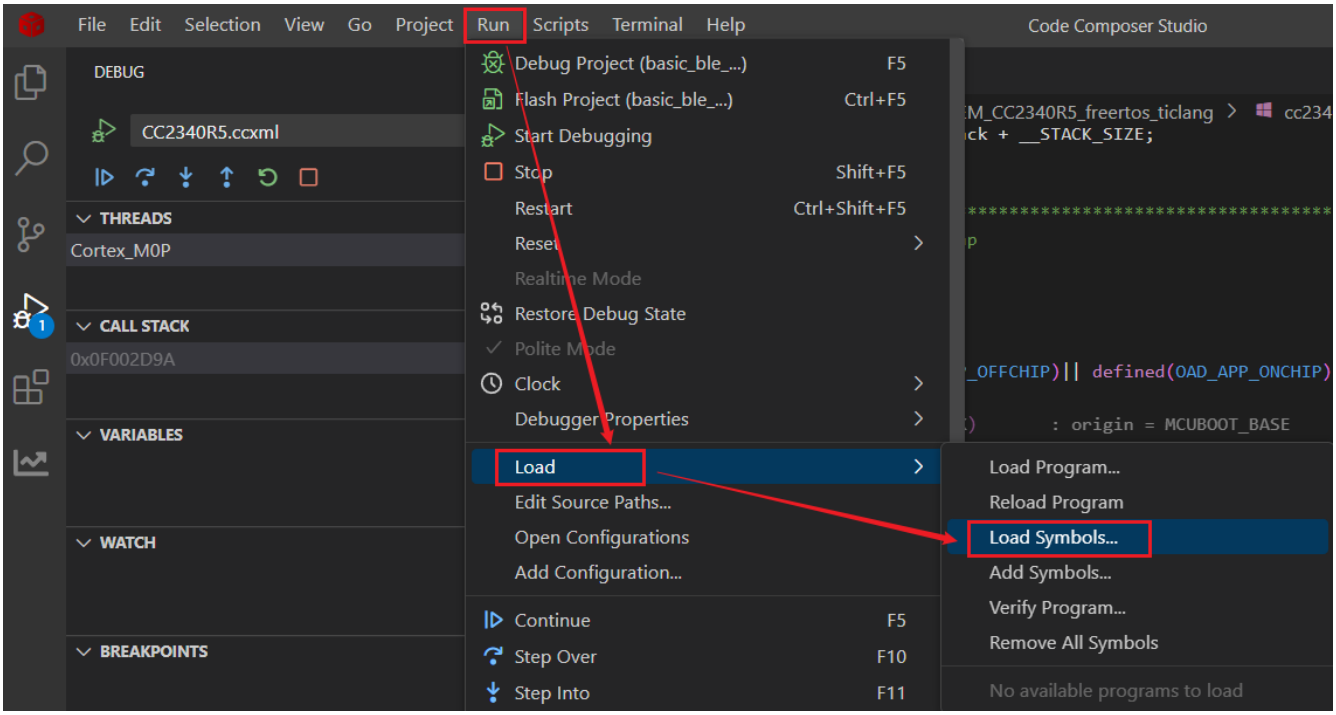
3. CCS opens the Debug view, expands the THREADS menu, and shows that the target chip is currently in the DISCONNECTED state. Right-click on DISCONNECTED and click Connect Target from the menu to connect to the target chip:

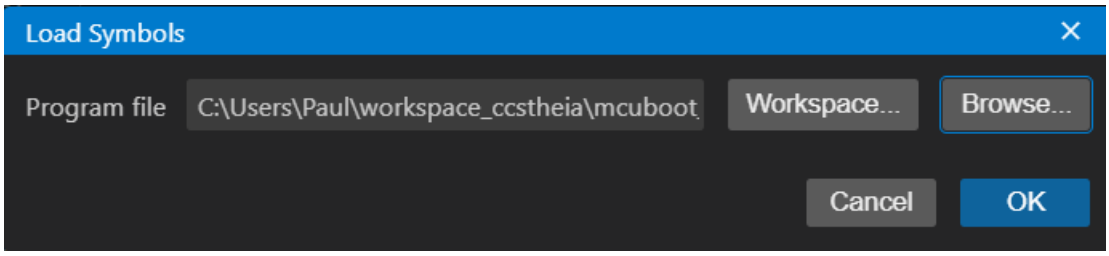


4. Once the connection is successful, the status changes to HALTED, and the top row of debug buttons turn on, allowing you to perform debug operations. However, the CALL STACK field only displays the address of the current program pointer and cannot display the corresponding code because we have not loaded the symbol file for the program images:



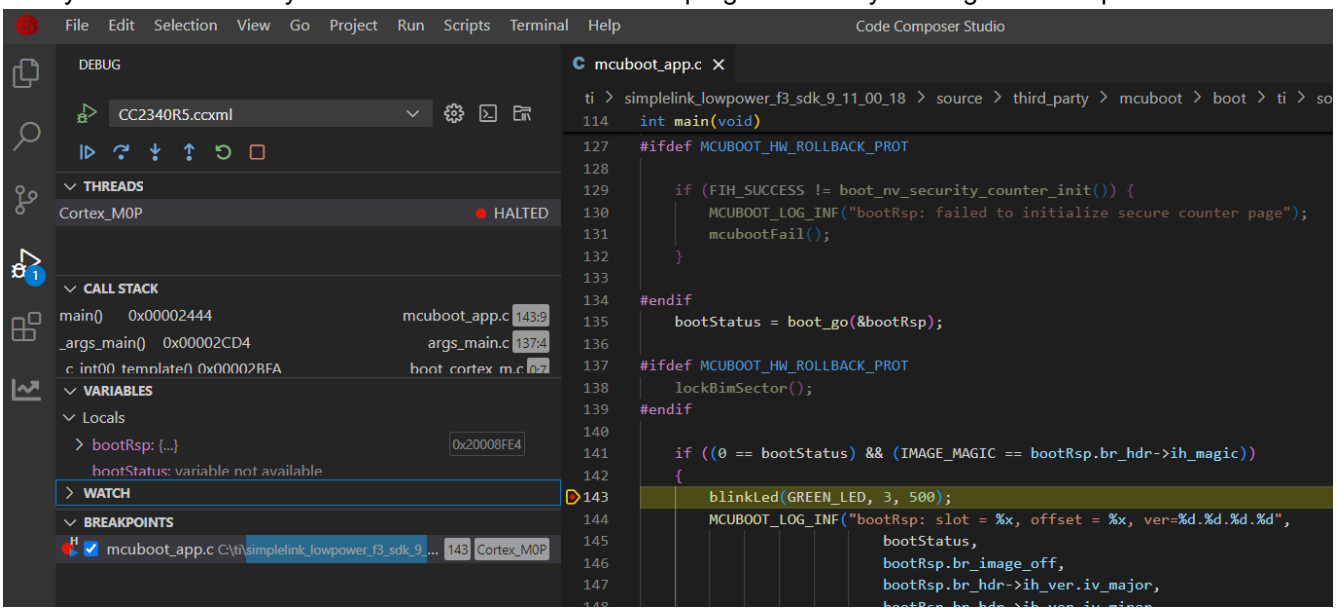
5. Click Run → Load → Load Symbols and select Browse in the pop-up dialog. Locate the .out file for the project (for example, mcuboot_LP_EM_CC2340R5_nortos_ticlang.out), and click OK to load it. Note that the symbol file here must be in .out format, and other formats (hex or bin) do not contain symbol information:



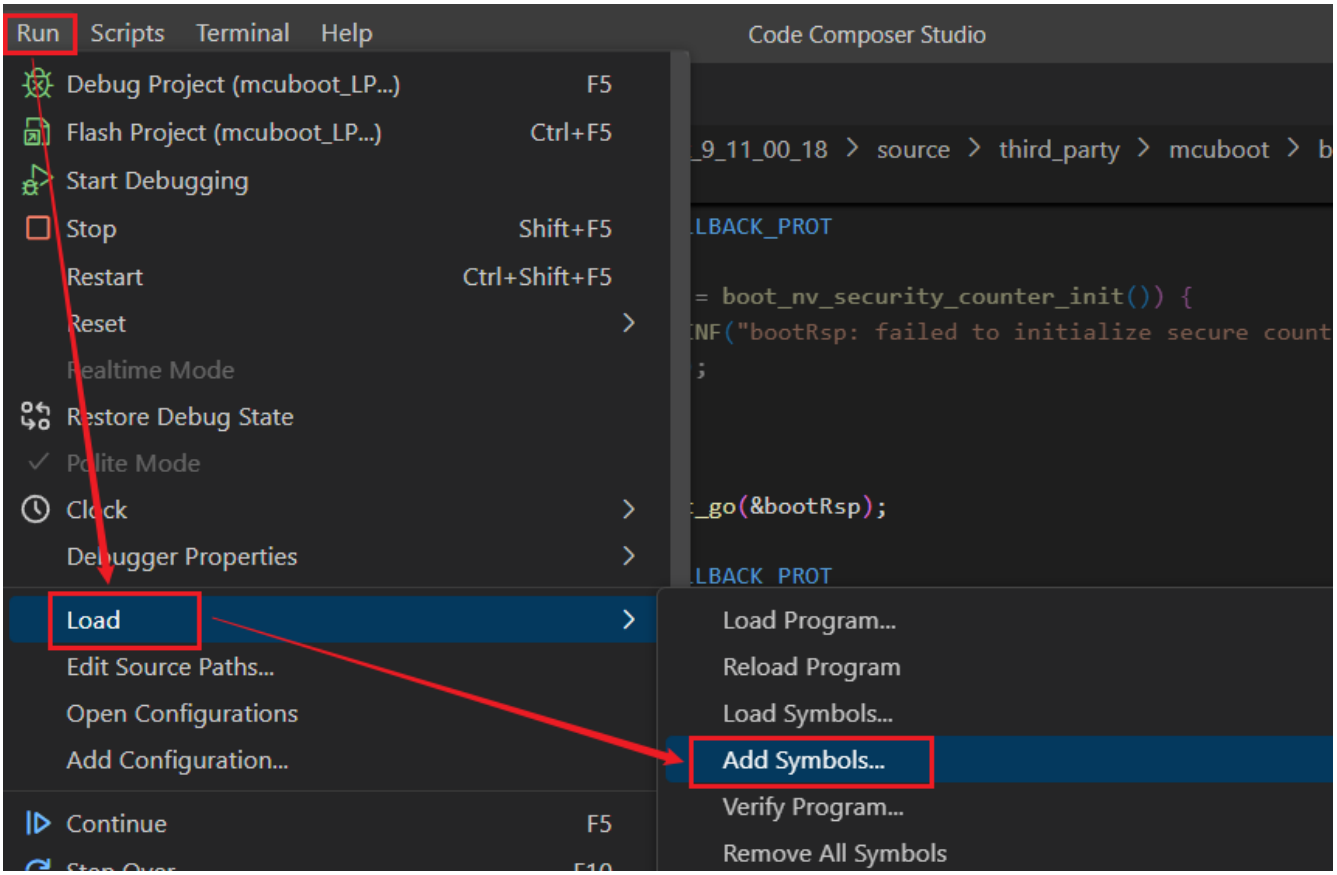


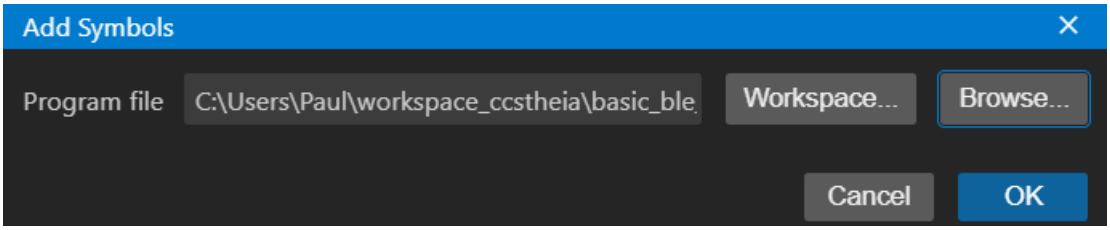
6. After loading the symbols, we can debug normally in CCS, such as adding breakpoints in the program, single-stepping, viewing variables and registers, and so forth. Note that loading symbols simply loads

the mapping of variables, functions, and addresses into the project and does not re-flash the program. Therefore, the program that was previously flashed by UniFlash is still running on the chip. If the program has been modified, re-flash it using UniFlash and load symbols by following the steps above; otherwise, the symbols in CCS may not be the same as those in the program actually running on the chip:



7. Now we have loaded the symbols for MCUboot only. Once the program jumps from MCUboot to basic_ble_oad_dual_image, it is also necessary to add the symbol file for the basic_ble_oad_dual_image project. Click Run → Load → Add Symbols (note to click Add Symbols, not Load Symbols) to load basic_ble_oad_dual_image_LP_EM_CC2340R5_freertos_ticlang.out into CCS. You can now debug both projects simultaneously:



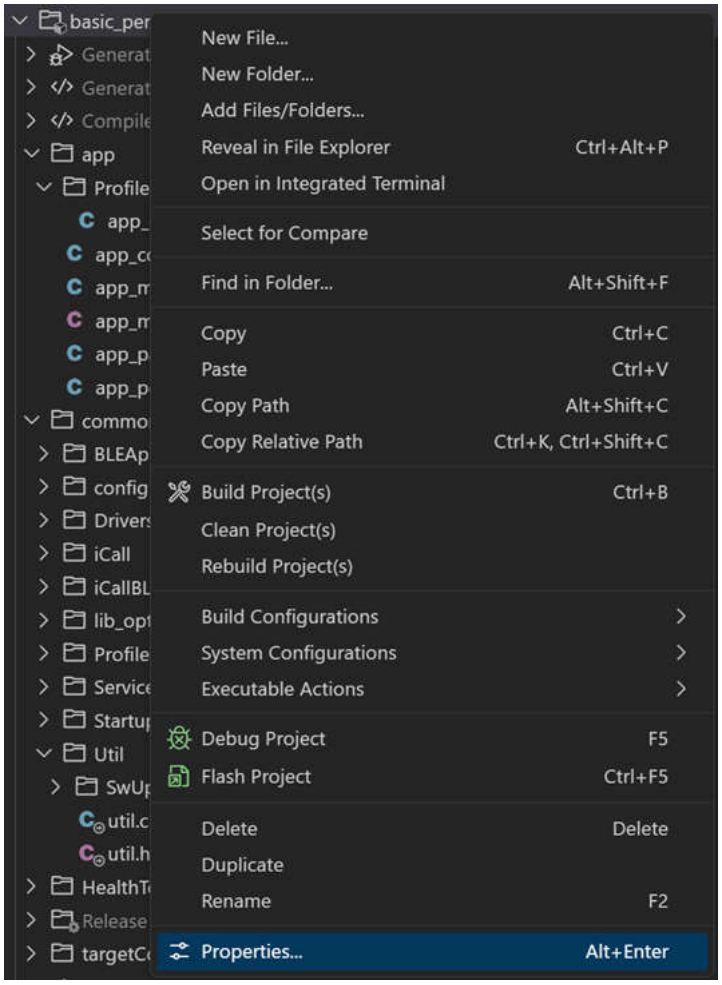


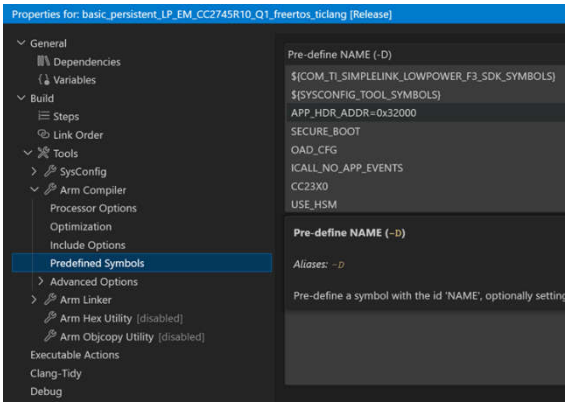
Tips: To check the ROV information for the basic_ble_oad_dual_image project, select basic_ble_oad_dual_image_LP_EM_CC2340R5_freertos_ticlang.out in the Load Symbols step, and then add mcuboot_LP_EM_CC2340R5_nortos_ticlang.out. Otherwise, the ROV information may not be displayed properly because the two projects are configured differently.

3.5 How to Adjust the Flash Location and Size of the App in an OAD Application

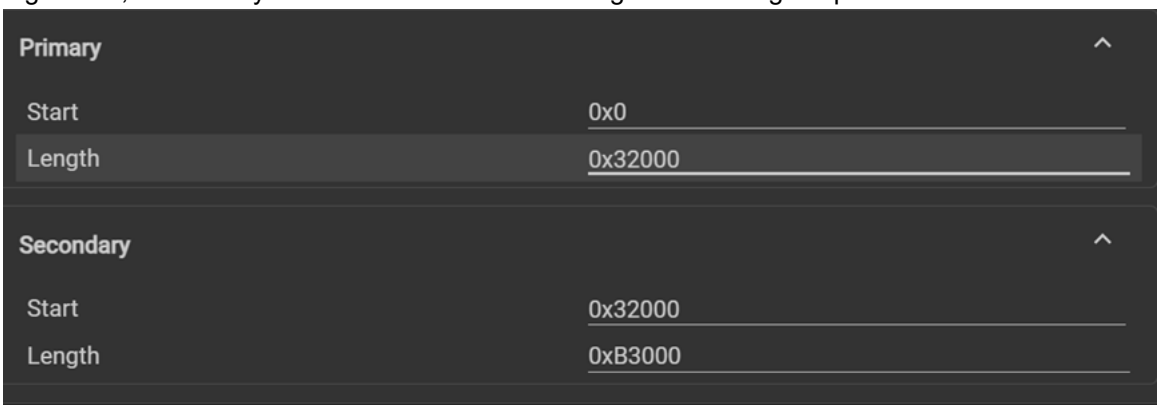
In the basic_ble_oad_onchip routine, two components, basic_persistent and oad_onchip, are configured, which are flashed to addresses 0x6000 and 0x32000, respectively. However, engineers may have a further need to change the location occupied by the App. To do this, follow the steps below:

1. Open the basic_persistent project, go to Properties → Predefined Symbols, configure APP_HDR_ADDR to a new persistent project end address (typically, this is also the start address of the basic oad project).

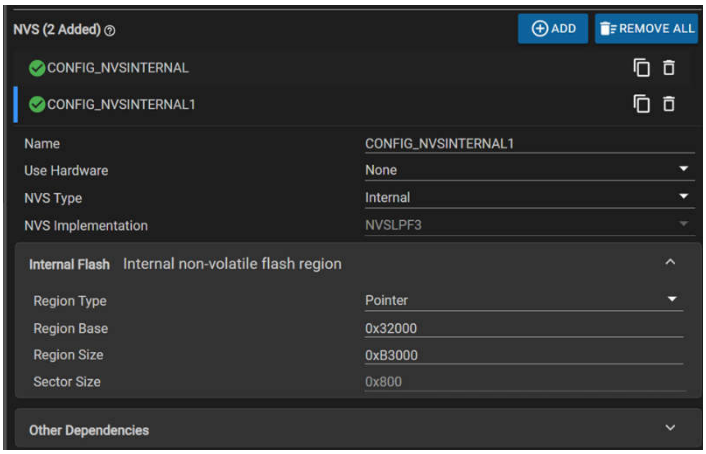




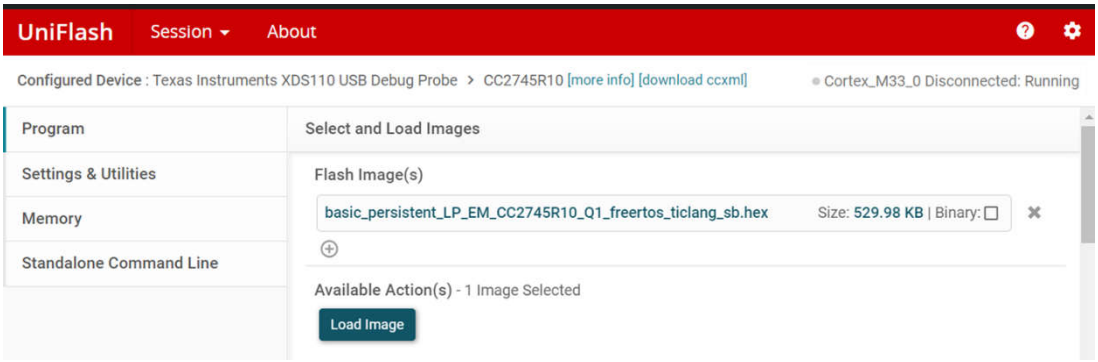
2. Locate the primary and secondary images under basic_persistent.syscfg → Device Configuration → Security Configuration, and modify their start addresses and lengths according to specific needs.



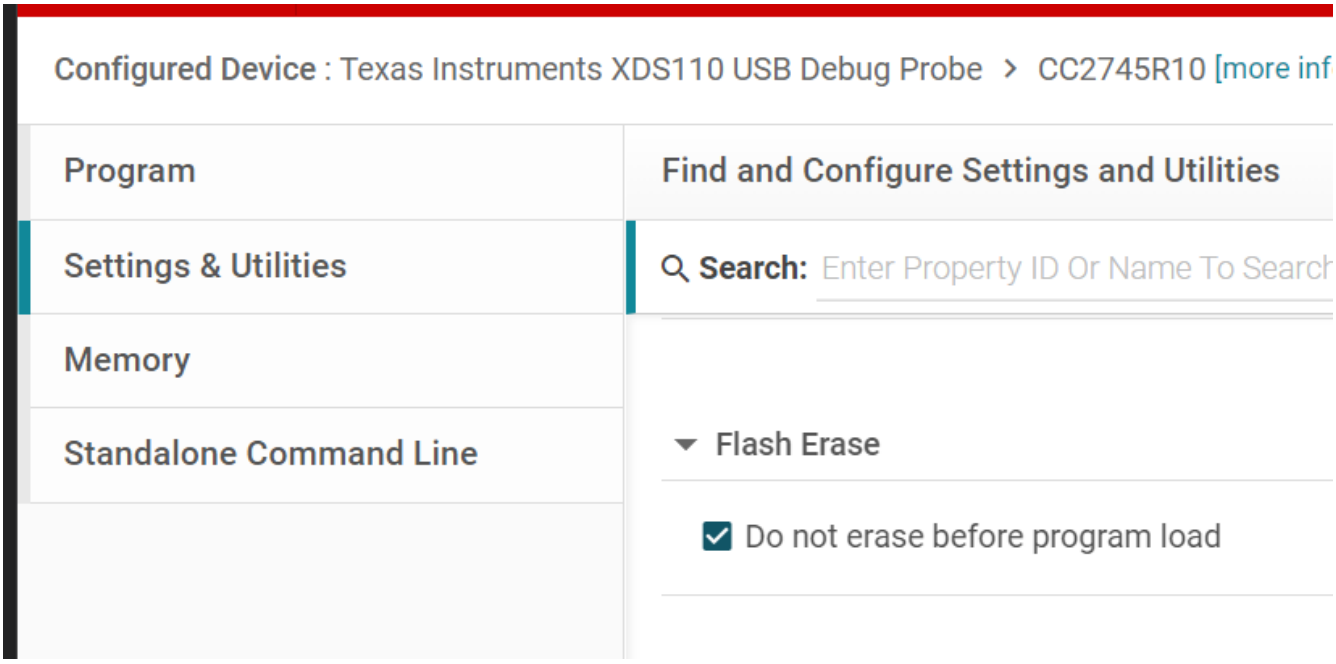
3. Modify CONFIG_NVSINTERNAL1 under basic_persistent.syscfg → NVS to match the secondary image configurations defined in the previous step. Repeat steps 1-3 for the basic_ble_oad_onchip project. Build both projects, and generate the basic_ble_persistent_LP_EM_CC2745R10_Q1_freertos_ticlang_sb.hex and basic_ble_oad_onchip_LP_EM_CC2745R10_Q1_freertos_ticlang_sb.bin files.



Flash basic_ble_persistent_LP_EM_CC2745R10_Q1_freertos_ticlang_sb.hex using UniFlash.



After flashing is complete, modify the flashing settings by going to Settings & Utilities → Flash Erase and checking “Do not erase before program load.”



Flash basic_ble_oad_onchip_LP_EM_CC2745R10_Q1_freertos_ticlang_sb.bin from the secondary image start address just configured.

3.6 How to Customize a Section in .cmd and Use it in Code

In some project implementations, engineers want to allocate a section of their own data space to support the application, and this section is defined by a specific start address and end address. To do this, declare a new section space in the SECTIONS section of the .cmd file.

A .cmd file is a linker command file that allocates specified sections to Flash or SRAM. For information on how to use this file, see also the reference document Linker Command. Regarding Linker Command files, the primary focus is on understanding and using the Memory and Section modules. For detailed guidance, see also the Memory User Manual and the Section User Manual .

```
SECTIONS
{
    .mysection :
{
    __my_section_start__ = ;
*(.my_section)
__my_section_end__ = .;
} > RAM
}
```

In the code, use extern uint8_t __my_section_start__; and extern uint8_t __my_section_end__; to reference it.

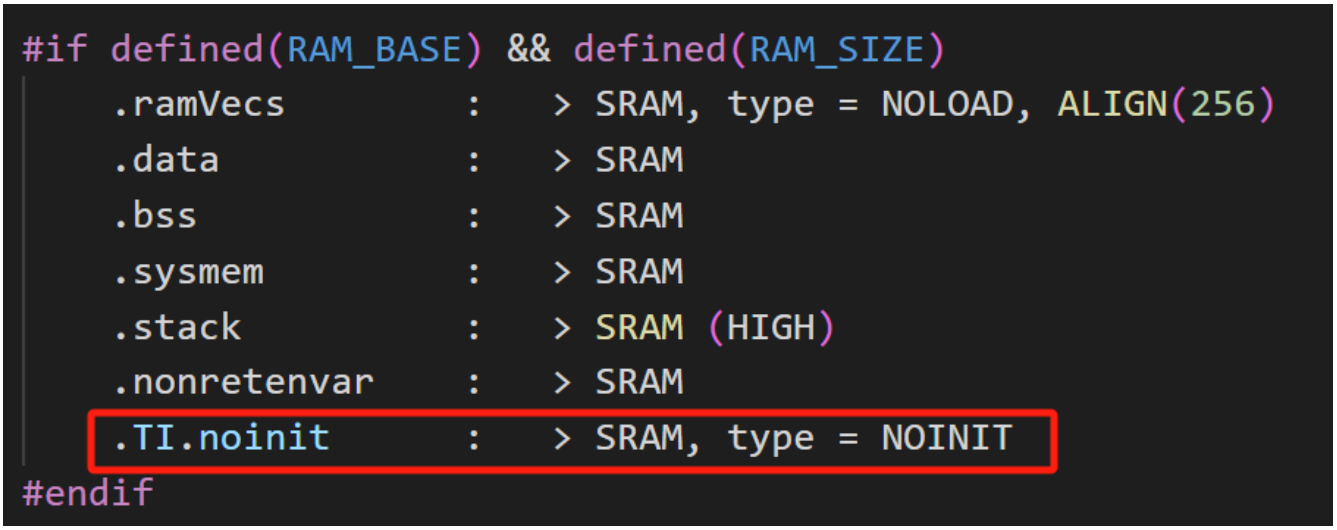
3.7 How to Use the Non-initialized Variable Section

A non-initialized variable section is the memory region that is not cleared or reassigned upon system power-up or soft reset (such as an OTA upgrade), and contains data that must be preserved after a program restart.

To define and use variables in the non-initialized variable section, three steps are required:

1. In the .cmd file mentioned above, declare the space within SECTIONS in RAM, as shown in the figure.

```
.TI.noinit : > SRAM, type = NOINIT
```



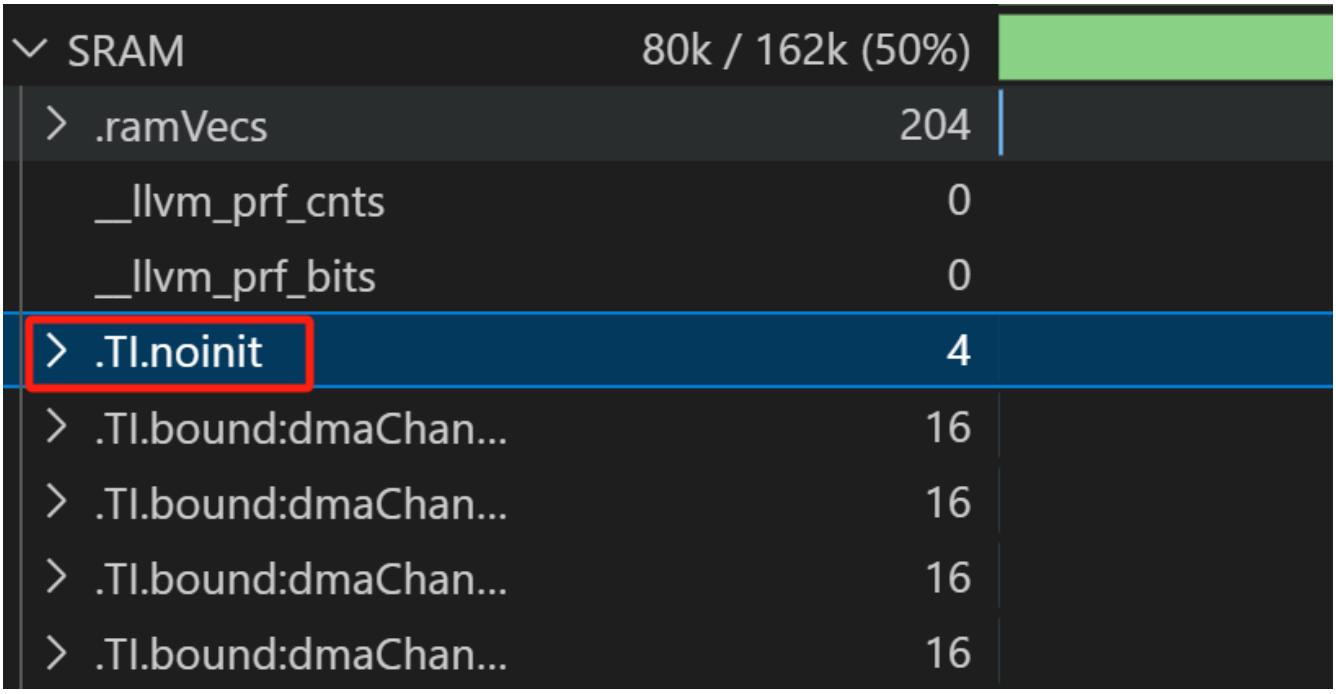
2. To place data in a non-initialized variable section, the application must use compiler attributes to declare it and assign it to the .TI.noinit section.

```
__attribute__((noinit)) int noinit_global;
__attribute__((noinit,location(0x100))) int noinit_location_global;
```

```
87  #endif //#if defined( HOST_CONFIG ) && ( HOST_CC
88
89
90  __attribute__((noinit)) int noinit_global;
91
92  //*****************************************************
93  //! Functions
94  //*****************************************************
95
96  /*****************************************************
97   * @fn      criticalErrorHandler
98   *
99   * @brief   Application task entry point
100  *
101  * @return  none
```

3. Using this variable in your code to build your program will add the corresponding definition to Memory Allocation.

```
186      }
187
188      noinit_global++;
189      MenuModule_printf(line: APP_
190
191      sleep(5);
192      Power_reset();
193
```



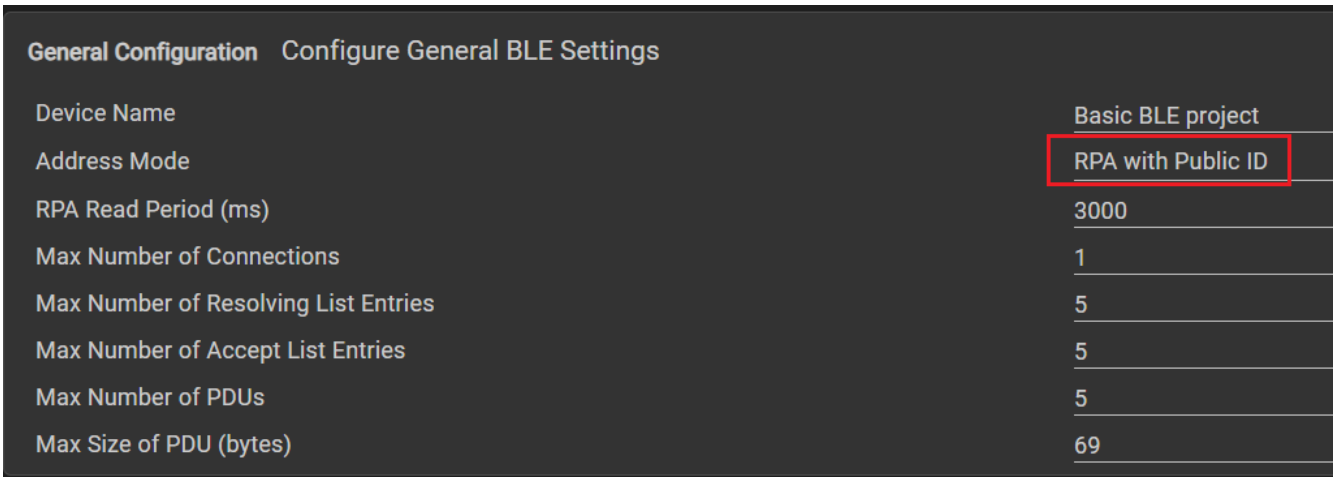
The code in the figure above is an example. A soft reset occurs every 5 seconds. noinit_global will increment continuously and will not be initialized to 0.

Note that a hardware reset (power cycle/reset pin reset) will reset this variable. If power-down hold is required, use non-volatile (NV) memory.

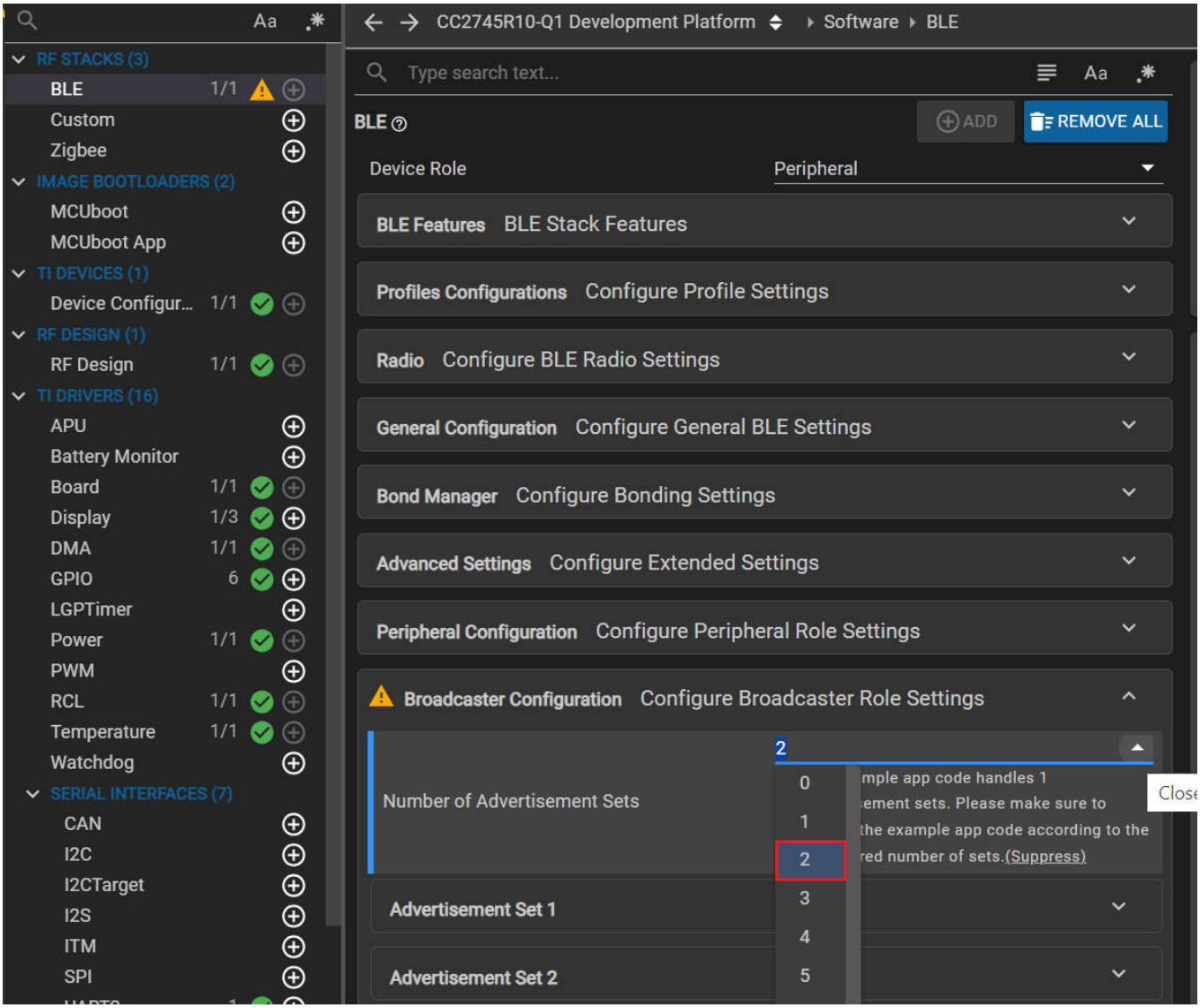
3.8 How to Simultaneously Send Two ADV Sets With Public and RPA Addresses

In the automotive key application field, there is often a need to broadcast multiple advertisement sets with different address modes due to specifications such as CCC, ICCE, and ICCOA. Here, we learn how to add two advertisement sets based on the basic_ble example program.

1. Go to SysConfig → BLE → General Configuration, and change Address Mode to RPA with Public Address.



2. Add one advertisement set under Broadcast Configuration, and ignore the warning prompt from SysConfig because we will later add the handling of the second advertisement set in code.

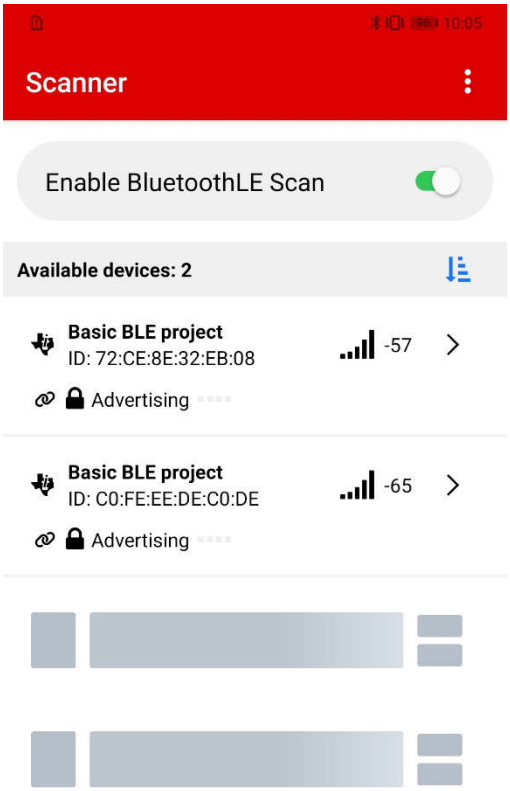


3. Declare the second ADV set in app_peripheral.c and modify the Peripheral_start function as follows.

```
uint8 peripheralAdvHandle_2;

const BLEAppUtil_AdvInit_t advSetInitParamsSet_2 =
{
    .advDataLen         = sizeof(advData2),
    .advData            = advData2,

    .scanRespDataLen    = sizeof(scanResData2),
    .scanRespData       = scanResData2,

    .advParam           = &advParams2
};

const BLEAppUtil_AdvStart_t advSetStartParamsSet_2 =
{
    /* Use the maximum possible value. This is the spec-defined maximum for */
    /* directed advertising and infinite advertising for all other types */
    .enableOptions          = GAP_ADV_ENABLE_OPTIONS_USE_MAX,
    .durationOrMaxEvents    = 0
};
```

```
bStatus_t Peripheral_start()
{
    bStatus_t status = SUCCESS;

    status = BLEAppUtil_registerEventHandler(eventHandler: &peripheralConnHandler);
    if(status != SUCCESS)
    {
        // Return status value
        return(status);
    }

    status = BLEAppUtil_registerEventHandler(eventHandler: &peripheralAdvHandler);
    if(status != SUCCESS)
    {
        return(status);
    }

    status = BLEAppUtil_initAdvSet(advHandle: &peripheralAdvHandle_1, advInitInfo: &advSetInitParamsSet_1);
    if(status != SUCCESS)
    {
        // Return status value
        return(status);
    }

    status = GapAdv_setParam(peripheralAdvHandle_2, GAP_ADV_PARAM_OWN_ADDRESS_TYPE, 0x00);
    if(status != SUCCESS)
    {
        return(status);
    }

    status = BLEAppUtil_advStart(handle: peripheralAdvHandle_1, advStartInfo: &advSetStartParamsSet_1);
    if(status != SUCCESS)
    {
        // Return status value
        return(status);
    }

    status = BLEAppUtil_initAdvSet(advHandle: &peripheralAdvHandle_2, advInitInfo: &advSetInitParamsSet_2);
        if(status != SUCCESS)
    {
        return(status);
    }

       status = BLEAppUtil_advStart(handle: peripheralAdvHandle_2, advStartInfo: &advSetStartParamsSet_2);
    if(status != SUCCESS)
    {
        // Return status value
        return(status);
    }

    // Return status value
    return(status);
}
```

4. After compiling and flashing, open the SimpleLink Connect App on your phone, and you will find two advertisement sets with different addresses.



3.9 What is the Difference Between E0 and E1 on the CC2745R10 Chip

The CC2745R10-Q1 is available in two options in the TI store: CC2745R10E0WRHARQ1 and CC2745R10E1WRHARQ1.

This is because during the development of the CC2745, there were two sample versions: PG1.0 and PG2.0. In the mass-production phase, they are numbered CC2745R10E0WRHARQ1 and CC2745R10E1WRHARQ1. Both chips are production-ready. However, the primary production chip is the CC2745R10E1WRHARQ1.

3.10 How to Enable More DMA Channels to Support My Driver Requirements

Engineers may require more DMA channels in some cases, such as enabling both SPI0/1 and UART0/1 for the purpose of

simultaneous data transmission on four serial ports. The 12 DMA channels defined for the CC2745 are described in section 19.3.1 of the CC27xx Technical Reference Manual (Rev. A).

Table 19-1 lists µDMA channel assignments to peripherals.

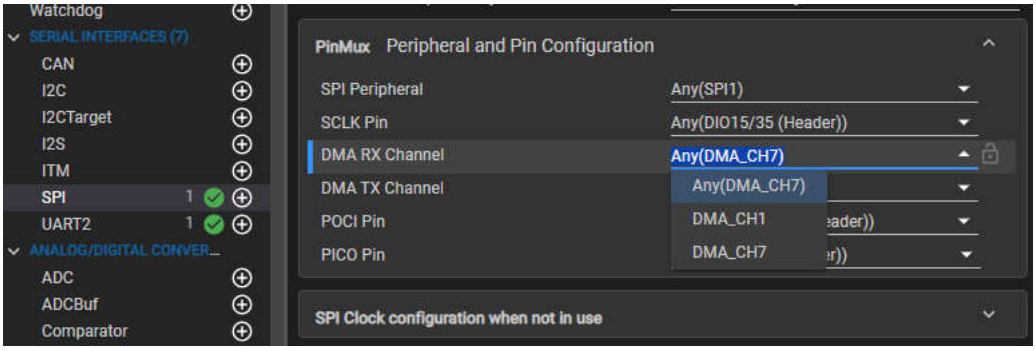
Table 19-1. Channel Assignments

µDMA Channel	Type (1) (2)	DTB (3)	Triggers
0	DCH	YES	SPI0TXTRG, UART1RXTRG
1	DCH	YES	SPI0RXTRG, UART1TXTRG
2	DCH	YES	LRFDTRG, UART0TXTRG
3	DCH	YES	ADC0TRG, UART0RXTRG
4	DCH	YES	LAESTRGA, LRFDTRG
5	DCH	YES	LAESTRGB, ADC0TRG
6	DCH	YES	CANTRGA,SPI1TXTRG
7	DCH	YES	CANTRGB,SPI1RXTRG
8	ECH	YES	ANY
9	ECH	YES	ANY
10	ECH	YES	ANY
11	ECH	YES	ANY

(1) Dedicated µDMA Channel (DCH) type can only be connected to peripherals with dedicated µDMA interface
(2) Event Publisher Channel (ECH) type can only be connected to publishers,

See also Section 4.4.5.1
(3) For DTB output, only SREQ and REQ is considered.

If engineers want to adjust the DMA channel for a driver or add a new channel, they can modify it in the project's SysConfig, using SPI0 as an example:



As shown in the table, SPI can be configured to use DMA_CH0/1 or DMA_CH6/7, which aligns with the actual configuration in SysConfig. Engineers who want to use DMA_CH8-11 should note that CH8-11 are the Event Publisher Channels and can only be used if there is an external publisher. They cannot be arbitrarily configured to be used with drivers for SPI or UART.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Last updated 10/2025