# Quick Start for Firmware developers

This section serves the purpose to help future developers to setup their development environment on their PC, get familiar with the application code structure. And finally being able to modify the code and debug on the CC2640 Bluetooth module developed for the company.

For convenience reason, CCS (TI’s eclipse based IDE) is used only for editing source code.

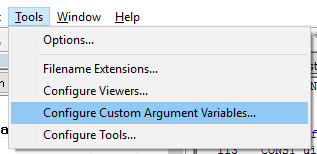
And for the high compiling speed and quality, the IAR 7.50 is used for compiling and generating binary files.

## Setup the IDE

For initial setup follow through 2.5 Setting up the Integrated Development Environment in **swru393b.pdf**

Two things to keep in mind

1. Install IAR 7.50 or above to a path with **no space, no Chinese characters**
2. Download the [BLE-STACK-2-1-1](http://www.ti.com/tool/ble-stack) and install it to the default location which is C:\ti\simplelink\ble\_cc26xx\_2\_01\_01\_44627
3. Verify the Custom Argument Variables for the Workspace are the same as shown below





## Install the Flash Programmer 2

This windows application can be used to flash firmware to the CC2640 for the first time when connected to a debugger like XDS100V3

<http://www.ti.com/tool/flash-programmer>

## Setup the Sensor controller studio

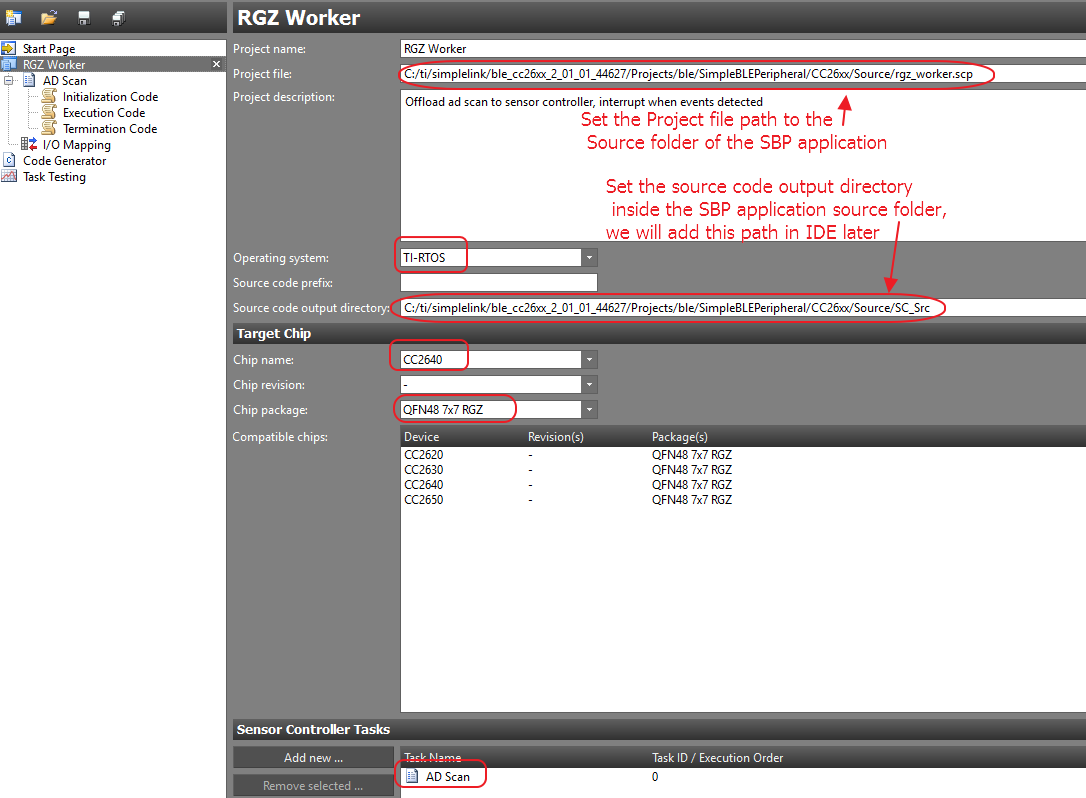
The sensor controller module is used for AD sampling in the background in the application independent of the main processor to save processing time of the main CPU.

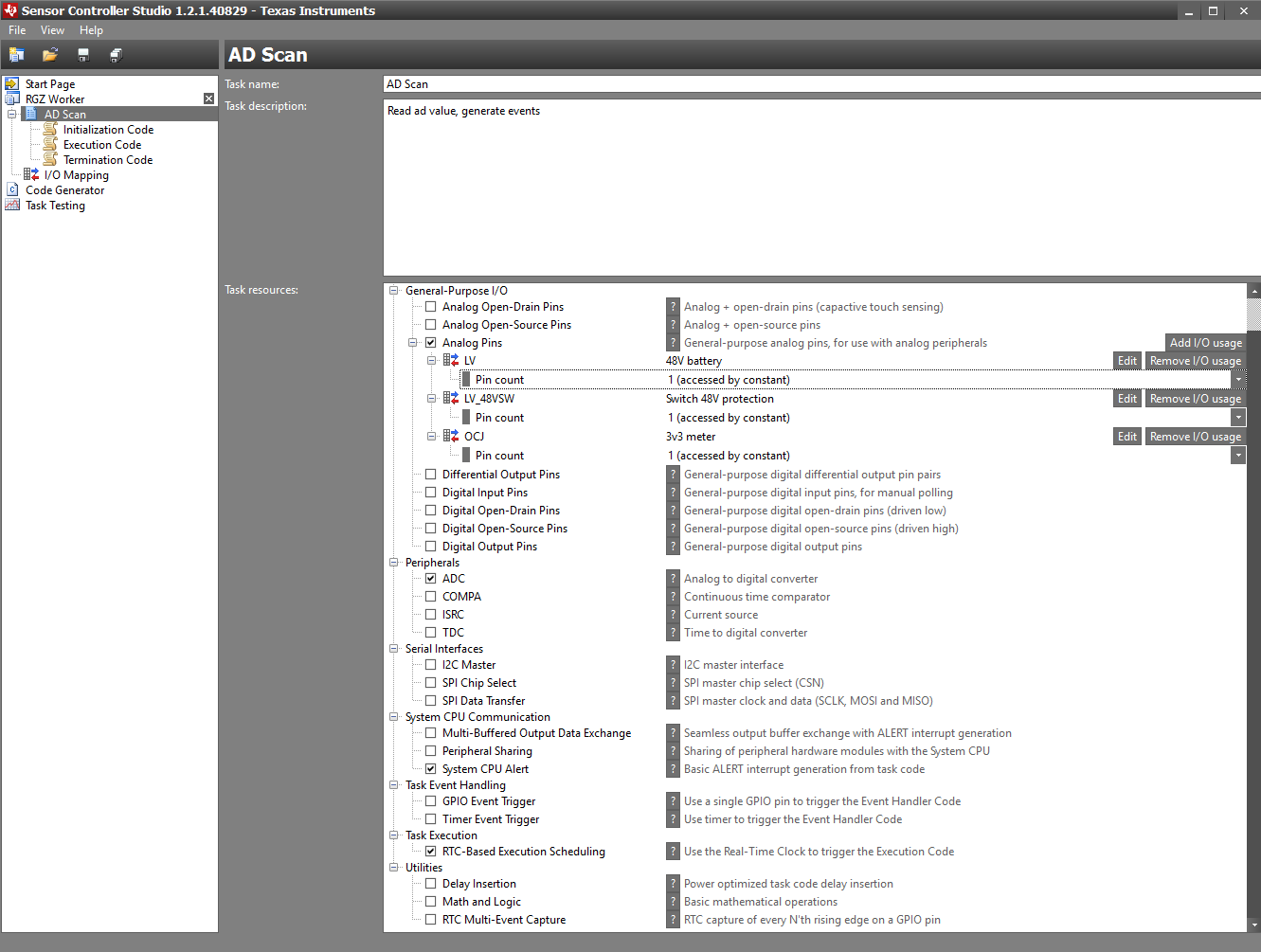
Get Sensor controller studio from TI and install it

<http://www.ti.com/tool/sensor-controller-studio>

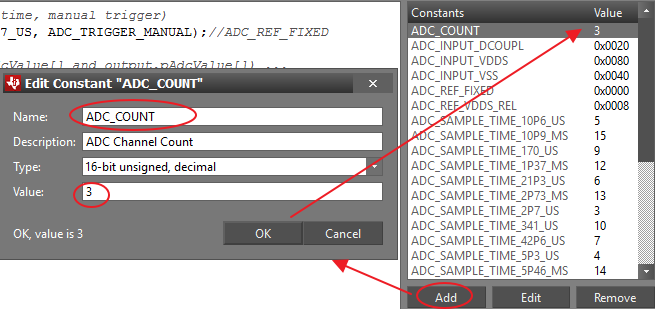
To create a Sensor Controller Studio (SCS) project from scratch

1. After that create a Sensor Controller project named **RGZ Worker**, and a task named **AD Scan**
2. Create 3 ADC pins
3. Enable RTC-Based Execution Scheduling

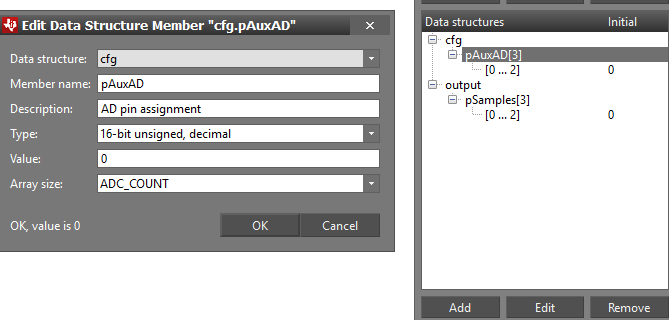


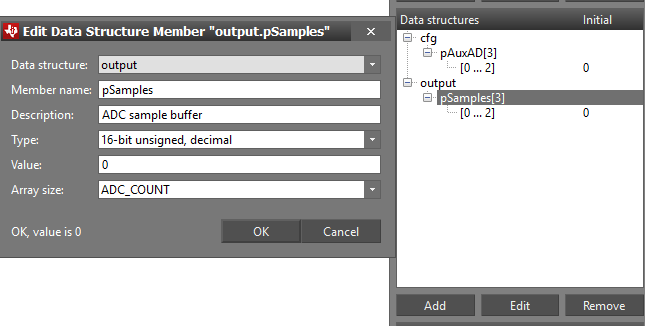


Add one constant

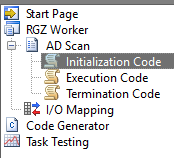


Add two structures





Edit the source code starting from the initialization code



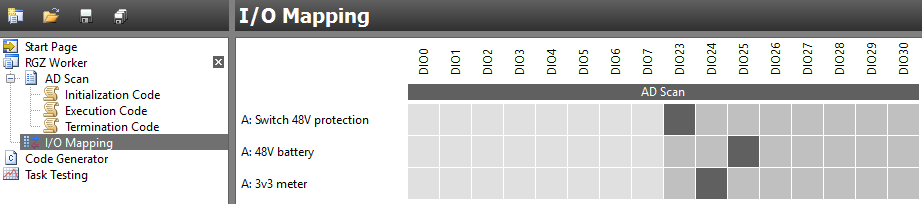
|  |
| --- |
| // Schedule the first execution  fwScheduleTask**(**1**);** |

Execution Code

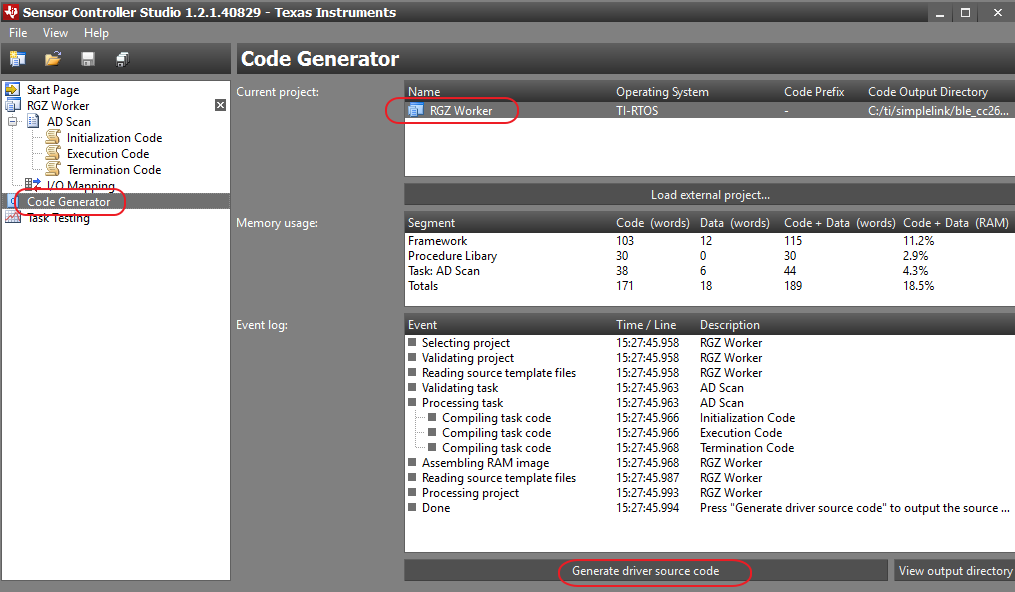
|  |
| --- |
| // Enable the ADC (3.3V reference, 2.7 us sample time, manual trigger)  adcEnableSync**(**ADC\_REF\_VDDS\_REL**,** ADC\_SAMPLE\_TIME\_2P7\_US**,** ADC\_TRIGGER\_MANUAL**);**//ADC\_REF\_FIXED  // For each pin (with one entry per pin in cfg.pAdcValue[] and output.pAdcValue[]) ...  **for** **(**U16 n **=** 0**;** n **<** ADC\_COUNT**;** n**++)** **{**    // Select ADC input  adcSelectGpioInput**(**cfg**.**pAuxAD**[**n**]);**    // Sample the pin and store the ADC value  adcGenManualTrigger**();**  adcReadFifo**(**output**.**pSamples**[**n**]);**  **}**  // Disable the ADC  adcDisable**();**  // Schedule the next execution  fwScheduleTask**(**1**);** |

Leave the Termination Code empty

I/O Mapping, this need to match with the actual hardware configuration



Finally generate the source code for the application



Alternatively we can just open up the Sensor Controller project file **rgz\_worker.scp** provided

## Modified source code

After setting up the IDE, some modification need to be made to the source code

### Board file

This will define pins and peripherals for our customer board

[Read section](#_Customer_Board_files)

### Off chip OAD flash related files

This will define the SPI flash IC page layout, device ID, manufacturer ID etc.

[Read section](#_Adjusting_Off-chip_Flash)

### OAD source code changes

Improved fail safe for off chip OAD

[Read section](#_About_interrupting_OAD)

### Add TI RTOS GPTimer Driver

The GPTimer Driver is not included in the driver folder by default, we will use the pre-released driver

[Read section](#_TI_RTOS_GPTimer)

### Notes on the software

#### Difference compared with legacy code

##### How the BLE AT commands are received

simpleBLEPeripheral.c

Line 1353

|  |
| --- |
| ArrInfo response\_ArrInfo = ATControl(pWrite->data, getUpdateCmdVariableLength()); |

The ATControl take the input string and compare with known AT command stored inside an array of structure, and execute functions if any match were found.

##### How the BLE commands are send as AT command

An **immediate** response message can be send by returning an “ArrInfo” struct from the ATControl function call.

Or we can send arbitrary data by calling SendDataBLE function which take a “ArrInfo” struct as parameter. The message will be send **asynchronously** by the Simple BLE Peripheral (SBP) task.

##### About pin assignment

All pin assignment are defined in the custom Board.h file

##### About loops

The legacy loops relaying on the system tick are moved to the PeriodicTimer.c since the system tick of CC2640 is running at 1ms while the legacy system tick period is much lower.

##### About receiving One Wire message

Since receiving one wire message require much higher IO scanning rate, a hardware **Timer0A** is used for this purpose. In the Timer **ISR**, it queues detected level change and its duration inside a buffer. The **PeriodicTimer task** will read out the queue asynchronously every 100ms.

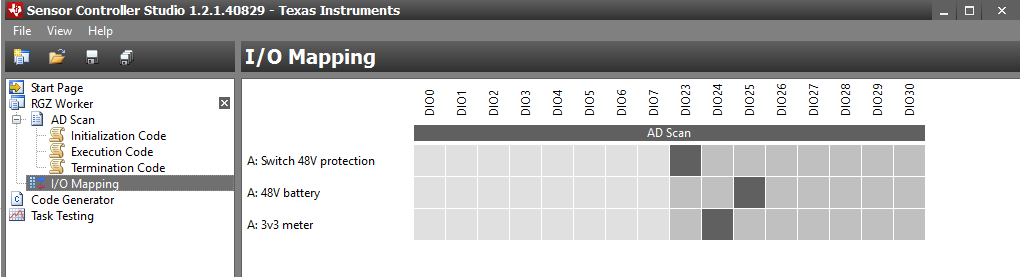
##### About ADC

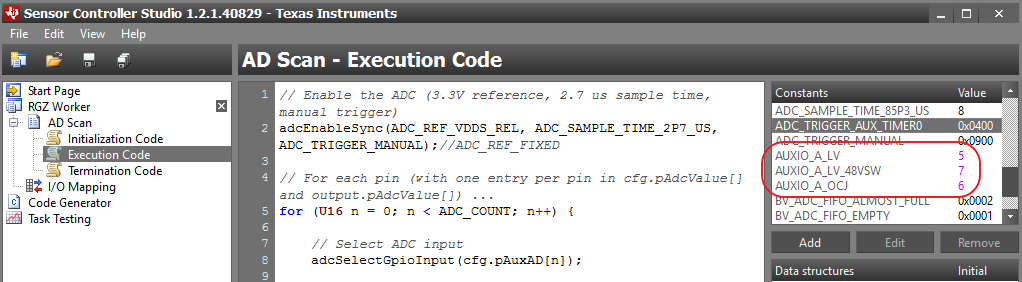
We use sensor control to handle the ADC scan to offload the processing resource from main CPU to sensor controller.

The AD pins configuration are defined in \_Parameter.h

|  |
| --- |
| **#if** BOARD\_VERSION == CC2640\_RGZ\_A  **#define** AUXIO\_A\_LV 3  **#define** AUXIO\_A\_LV\_48VSW 7  **#define** AUXIO\_A\_OCJ 4  **#elif**(BOARD\_VERSION==CC2640\_RGZ\_B)  **#define** AUXIO\_A\_LV 5  **#define** AUXIO\_A\_LV\_48VSW 7  **#define** AUXIO\_A\_OCJ 6  **#endif** |

The number need to match with ones defined in the Sensor Controller Studio once IO is mapped





##### About storing persistent data to on board Flash

Persistent data are stored in the SNV section of the on board Flash

Since the “osal\_snv\_write” and “osal\_snv\_read” functions can only be called within a task which registered ICall like the SBP task. So in the implementation of the FlashSNV.c we use callback to notify the SBP task to do those operations for us by calling “NotifyFlashAction” function.

# SW Debug Note

## OAD

### Off-Chip OAD

[Forum Q&A](http://www.deyisupport.com/question_answer/wireless_connectivity/bluetooth/f/103/t/105330.aspx)

#### Adjusting Off-chip Flash Memory Layout

C:\ti\simplelink\ble\_cc26xx\_2\_01\_01\_44627\Projects\ble\Profiles\OAD\CC26xx\ext\_flash\_layout.h

Change the value to match with our 2M-bit Flash W25X20CLUXIG (256KB)

4M-bit (512KB) W25X40CL 🡪 0x78000

2M-bit (256KB) W25X20CL 🡪 0x3F000

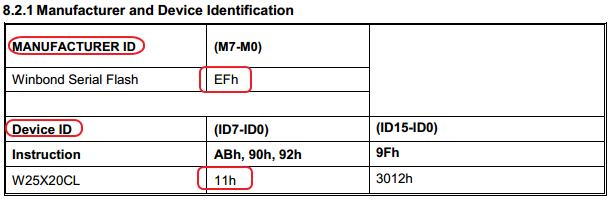
1M-bit (128kB) W25X10CL 🡪 0x1F000

|  |
| --- |
| // Image information (meta-data)  **#define** EFL\_ADDR\_META 0x3F000 //0x78000 |

Change **Flash Manufacturer and Device Identification**

**Refer to Flash IC Datasheet**

e.g. in section **8.2.1 Manufacturer and Device Identification** of the **W25X20CLUXIG**’s datasheet we have



**C:\ti\simplelink\ble\_cc26xx\_2\_01\_01\_44627\Projects\ble\util\BIM\_extflash\CC26xx\Source\CC26XXST\_0120\** **bsp.h**

**And**

|  |
| --- |
| /\* External flash manufacturer and device ID \*/  **#define** EXT\_FLASH\_MAN\_ID 0xEF// Winbond Serial Flash  **#define** EXT\_FLASH\_DEV\_ID 0x11//0x12 //for w25x20cl it is 0x11, for w25x40cl it is 0x12 |

#### Build BIM\_extflash (BIM\_ext.hex)

Open up the IAR workspace

**BIM\_extflash.eww**

**Note**: use the FlashOnly configuration by default, not the sensortag configuration

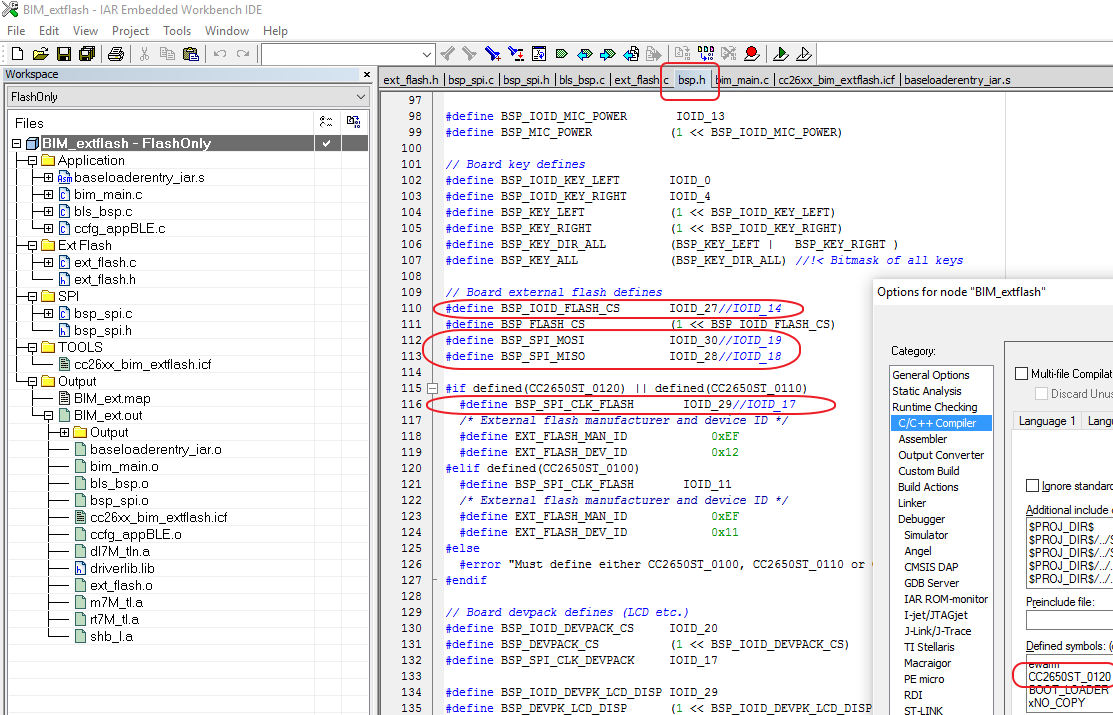
File location

C:\ti\simplelink\ble\_cc26xx\_2\_01\_01\_44627\Projects\ble\util\BIM\_extflash\CC26xx\IAR\BIM\_extflash.eww

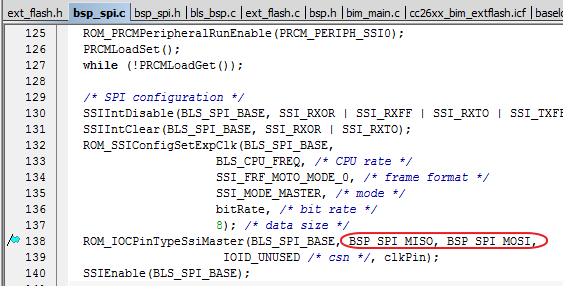
To change **SPI pins**, open up **bsp.h** and change the 4 pins’ assignment accordingly

The file location

C:\ti\simplelink\ble\_cc26xx\_2\_01\_01\_44627\Projects\ble\util\BIM\_extflash\CC26xx\Source\CC26XXST\_0120



Those pins are referenced in file **bsp\_spi.c**



Application-Stack combined image generated by IAR

FlashOnly\_OAD\_ST\_ExtFlash\Exe\OAD\_FULL\_IMAGE.hex

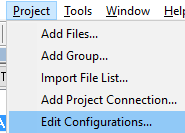
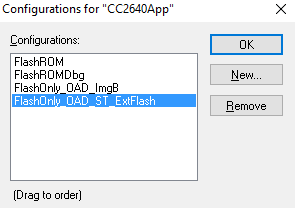
#### ExtFlash OAD image(OAD\_FULL\_IMAGE.hex)

Open up SBP project

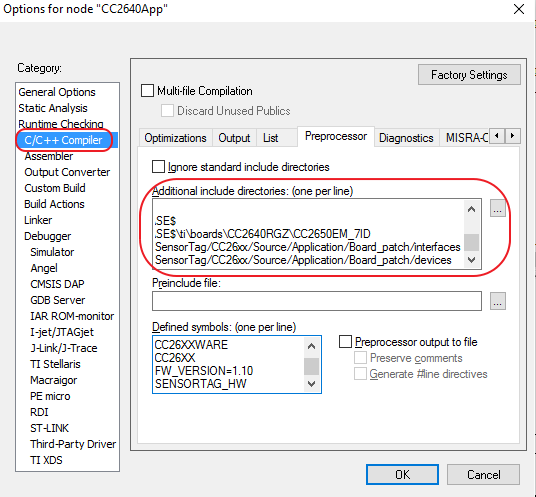
Edit the **SPI pins** in **Board.h** file, add other Marcos as shown bellow

|  |
| --- |
| **#define** Board\_SPI0\_MISO IOID\_28  **#define** Board\_SPI0\_MOSI IOID\_30  **#define** Board\_SPI0\_CLK IOID\_29  **#define** Board\_SPI0\_CSN IOID\_27  /\* External flash manufacturer and device ID \*/  **#define** EXT\_FLASH\_MAN\_ID 0xEF// Winbond Serial Flash  **#define** EXT\_FLASH\_DEV\_ID 0x11//0x12 //for w25x20cl it is 0x11, for w25x40cl it is 0x12  /\* SPI \*/  **#define** Board\_SPI\_FLASH\_CS Board\_SPI0\_CSN  **#define** Board\_FLASH\_CS\_ON 0  **#define** Board\_FLASH\_CS\_OFF 1  **#define** Board\_DEVPK\_CS\_ON 1  **#define** Board\_DEVPK\_CS\_OFF 0 |

Switch to **FlashOnly\_OAD\_ST\_ExtFlash** configuration

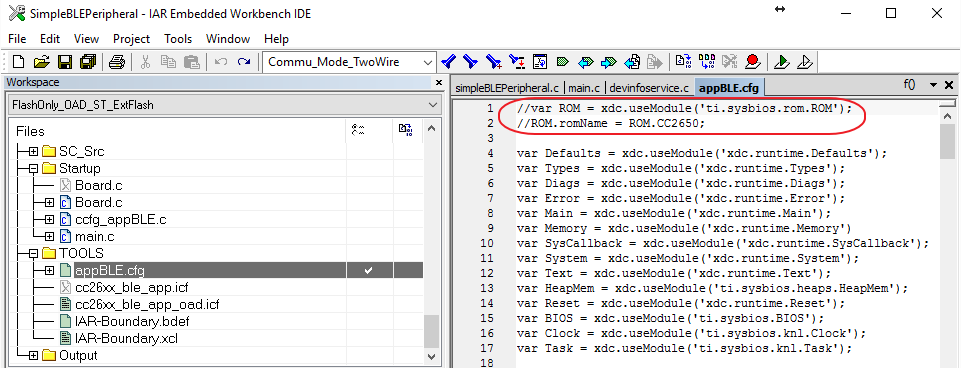
 

We have modified the **Additional include directories**. We added sensor controller generated source code directory and modified the Board files directory to our Board.c/h files.



|  |
| --- |
| $PROJ\_DIR$  $PROJ\_DIR$/../../../Source/Application  $PROJ\_DIR$/../../../Source/SC\_Src  $PROJ\_DIR$/../../../../../../../Projects/ble/Include  $PROJ\_DIR$/../../../../../../../Projects/ble/ICall/Include  $PROJ\_DIR$/../../../../../../../Projects/ble/Profiles/Roles/CC26xx  $PROJ\_DIR$/../../../../../../../Projects/ble/Profiles/Roles  $PROJ\_DIR$/../../../../../../../Projects/ble/Profiles/DevInfo  $PROJ\_DIR$/../../../../../../../Projects/ble/Profiles/SimpleProfile/CC26xx  $PROJ\_DIR$/../../../../../../../Projects/ble/Profiles/SimpleProfile  $PROJ\_DIR$/../../../../../../../Projects/ble/Profiles/OAD/CC26xx  $PROJ\_DIR$/../../../../../../../Projects/ble/common/cc26xx  $PROJ\_DIR$/../../../../../../../Components/applib/heap  $PROJ\_DIR$/../../../../../../../Components/ble/hci  $PROJ\_DIR$/../../../../../../../Components/ble/controller/CC26xx/include  $PROJ\_DIR$/../../../../../../../Components/ble/host  $PROJ\_DIR$/../../../../../../../Components/hal/target/CC2650/rom  $PROJ\_DIR$/../../../../../../../Components/hal/target/CC2650TIRTOS  $PROJ\_DIR$/../../../../../../../Components/hal/target/\_common/cc26xx  $PROJ\_DIR$/../../../../../../../Components/hal/include  $PROJ\_DIR$/../../../../../../../Components/osal/include  $PROJ\_DIR$/../../../../../../../Components/services/sdata  $PROJ\_DIR$/../../../../../../../Components/services/saddr  $PROJ\_DIR$/../../../../../../../Components/icall/include  $PROJ\_DIR$/../../../../../../../Components/ble/include  $CC26XXWARE$  $TI\_RTOS\_DRIVERS\_BASE$  $TI\_RTOS\_DRIVERS\_BASE$\ti\boards\CC2640RGZ\CC2650EM\_7ID  $PROJ\_DIR$/../../../../../SensorTag/CC26xx/Source/Application/Board\_patch/interfaces  $PROJ\_DIR$/../../../../../SensorTag/CC26xx/Source/Application/Board\_patch/devices |

Make sure we have commented out those two lines in TOOLS/appBLE.cfg



Update the **devInfoFirmwareRev** inside file **PROFILES/devinfoservice.c**

|  |
| --- |
| **static** uint8 devInfoFirmwareRev[DEVINFO\_STR\_ATTR\_LEN+1] = "1"; |

In CCS we have added

|  |
| --- |
| "${ORG\_PROJ\_DIR}/../../../Source/SC\_Src" |

In the entry

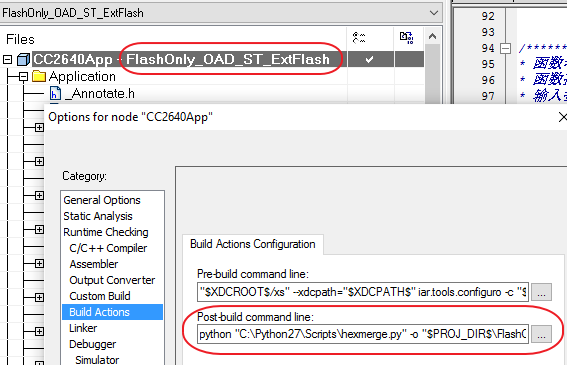
Also replaced

|  |
| --- |
| ~~"${TI\_RTOS\_BOARD\_BASE}/CC26XXST\_0120"~~ |

With

|  |
| --- |
| "${TI\_RTOS\_DRIVERS\_BASE}/ti/boards/CC2640RGZ/CC2650EM\_7ID" |

Generate the Full OAD image **OAD\_FULL\_IMAGE.hex**



Have this post build command in IAR

|  |
| --- |
| python "C:\Python27\Scripts\hexmerge.py" -o "$PROJ\_DIR$\FlashOnly\_OAD\_ST\_ExtFlash\Exe\**OAD\_FULL\_IMAGE.hex**" -r "1000:1CFFF" --overlap=error "$PROJ\_DIR$\FlashOnly\_OAD\_ST\_ExtFlash\Exe\**SimpleBLEPeripheral\_OADExtFlash.hex**":1000:EFFF "$PROJ\_DIR$\..\..\Stack\CC2640\FlashROM\Exe\**SimpleBLEPeripheralStackFlashROM.hex**":F000:1CFFF |

#### Flash Super image

We can generate a super image which combines the two hex file (Stack,App + BIM)

This Post Build script will generate

**OFFCHIP\_APP\_OAD.bin** for Mobile OAD

**OAD\_FULL\_IMAGE.hex** for BLE Device monitor to OAD both Stack and Application

**SUPER\_FULL\_IMAGE.hex** to be flashed to chip first time using Flash Programmer 2

This SUPER\_FULL\_IMAGE.hex file generated bellow need to be flashed to chip first using Flash Programmer 2

|  |
| --- |
| @**ECHO** OFF  **SET** PROJ\_DIR**=%1**  **SET** BINFILE**=**OFFCHIP\_APP\_OAD.bin  **ECHO** project directory is "**%PROJ\_DIR%**"  **ECHO** merge app and stack hex files to one file  python "C:\Python27\Scripts\hexmerge.py" -o "**%PROJ\_DIR%**\FlashOnly\_OAD\_ST\_ExtFlash\Exe\OAD\_FULL\_IMAGE.hex" -r "1000:1CFFF" --overlap**=**error "**%PROJ\_DIR%**\FlashOnly\_OAD\_ST\_ExtFlash\Exe\SimpleBLEPeripheral\_OADExtFlash.hex":1000:EFFF "**%PROJ\_DIR%**\..\..\Stack\CC2640\FlashROM\Exe\SimpleBLEPeripheralStackFlashROM.hex":F000:1CFFF  **ECHO** Generate OAD binary file "**%BINFILE%**"  python "C:\Python27\Scripts\hex2bin.py" -r "1000:BFFF" "**%PROJ\_DIR%**\FlashOnly\_OAD\_ST\_ExtFlash\Exe\SimpleBLEPeripheral\_OADExtFlash.hex" "**%PROJ\_DIR%**\FlashOnly\_OAD\_ST\_ExtFlash\Exe\**%BINFILE%**"  **ECHO** Generate OAD super hex file "SUPER\_FULL\_IMAGE.hex"  python "C:\Python27\Scripts\hexmerge.py" -o "**%PROJ\_DIR%**\FlashOnly\_OAD\_ST\_ExtFlash\Exe\SUPER\_FULL\_IMAGE.hex" -r "0000:1FFFF" --overlap**=**error "**%PROJ\_DIR%**\FlashOnly\_OAD\_ST\_ExtFlash\Exe\SimpleBLEPeripheral\_OADExtFlash.hex":1000:1EFFF "**%PROJ\_DIR%**\..\..\Stack\CC2640\FlashROM\Exe\SimpleBLEPeripheralStackFlashROM.hex":1000:1EFFF "**%PROJ\_DIR%**\..\..\..\..\..\..\ble\util\BIM\_extflash\CC26xx\IAR\FlashOnly\Exe\BIM\_ext.hex":0000:1FFFF |

Those generated hex files reside in directory

C:\ti\simplelink\ble\_cc26xx\_2\_01\_01\_44627\Projects\ble\SimpleBLEPeripheral\CC26xx\IAR\Application\CC2640\FlashOnly\_OAD\_ST\_ExtFlash\Exe

#### Do the OAD

We have change out application code

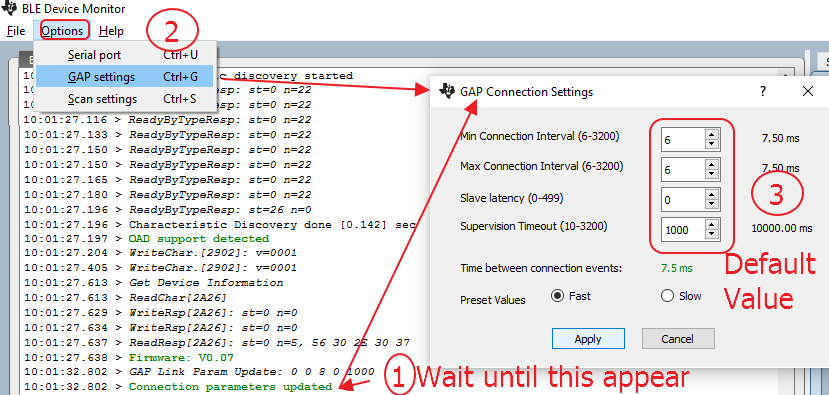
Updated the **devInfoFirmwareRev** inside file **PROFILES/devinfoservice.c**

|  |
| --- |
| **static** uint8 devInfoFirmwareRev[DEVINFO\_STR\_ATTR\_LEN+1] = "2"; |

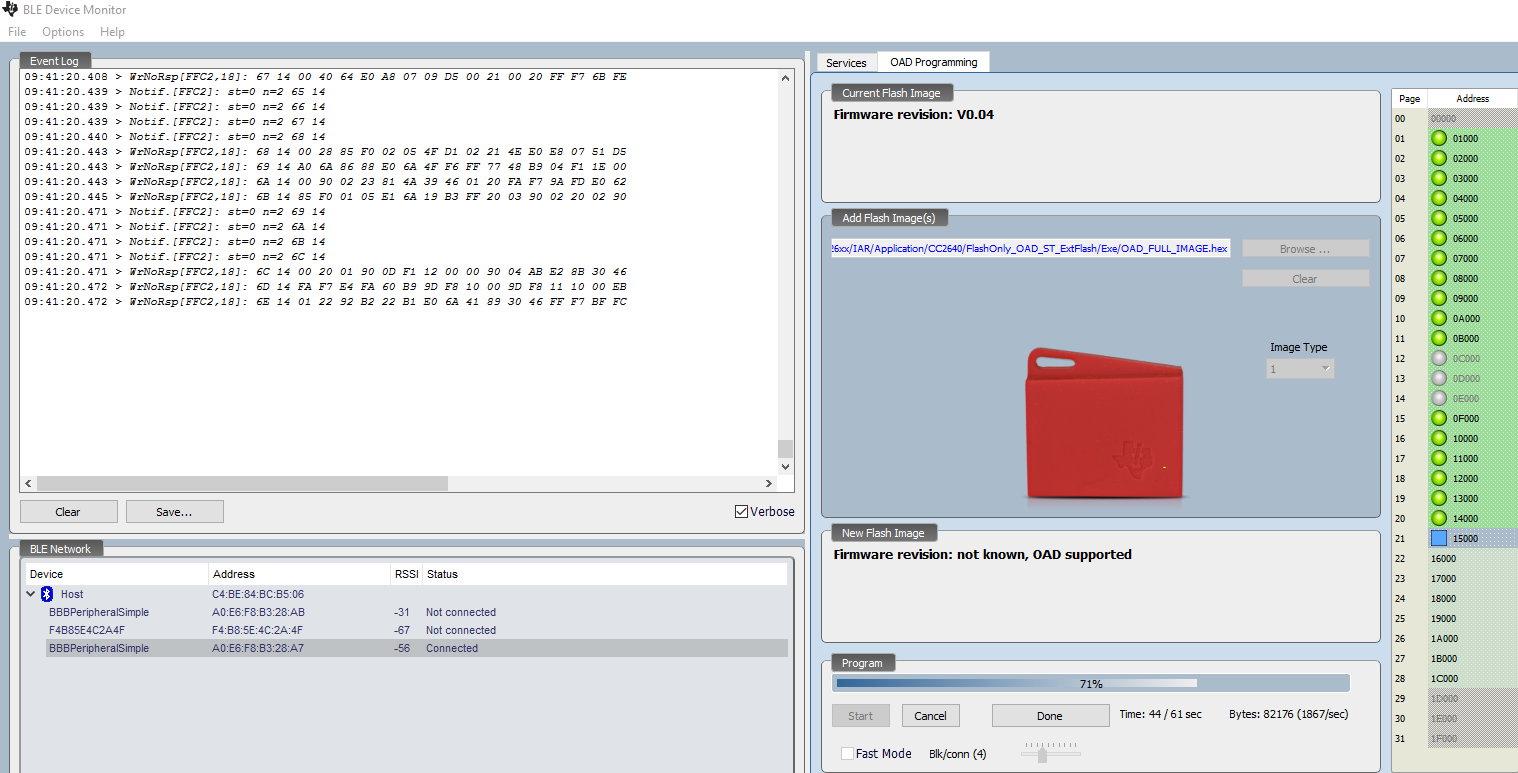
And compiled successful, which generated new OAD\_FULL\_IMAGE.hex file

To do the off chip OAD

* Open up BLE Device Monitor, [click here for info to make a HostTest USB dongle](#_Flash_the_HostTestApplication)
* C:\ti\simplelink\ble\_cc26xx\_2\_01\_01\_44627\Accessories\HexFiles\ CC2640\_SmartRF-7ID\_HostTestRelease\_All.hex or C:\ti\simplelink\ble\_sdk\_2\_02\_00\_31\examples\hex\cc2650lp\_host\_test\_rel.hex
* Update the GAP Settings

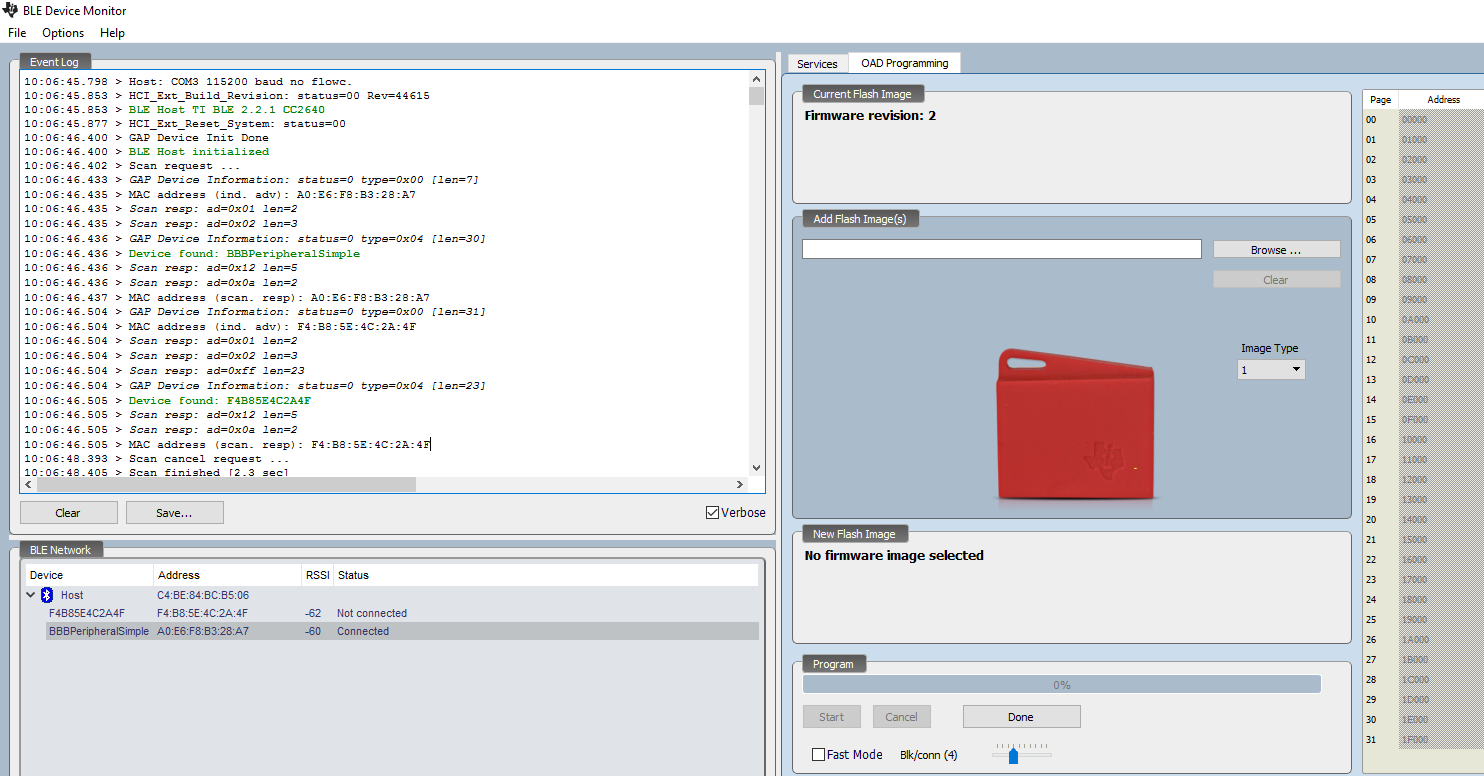


* Select the new OAD\_FULL\_IMAGE.hex or just the application hex SimpleBLEPeripheral\_OADExtFlash.hex if the stack is not modified.
* Click start download the code

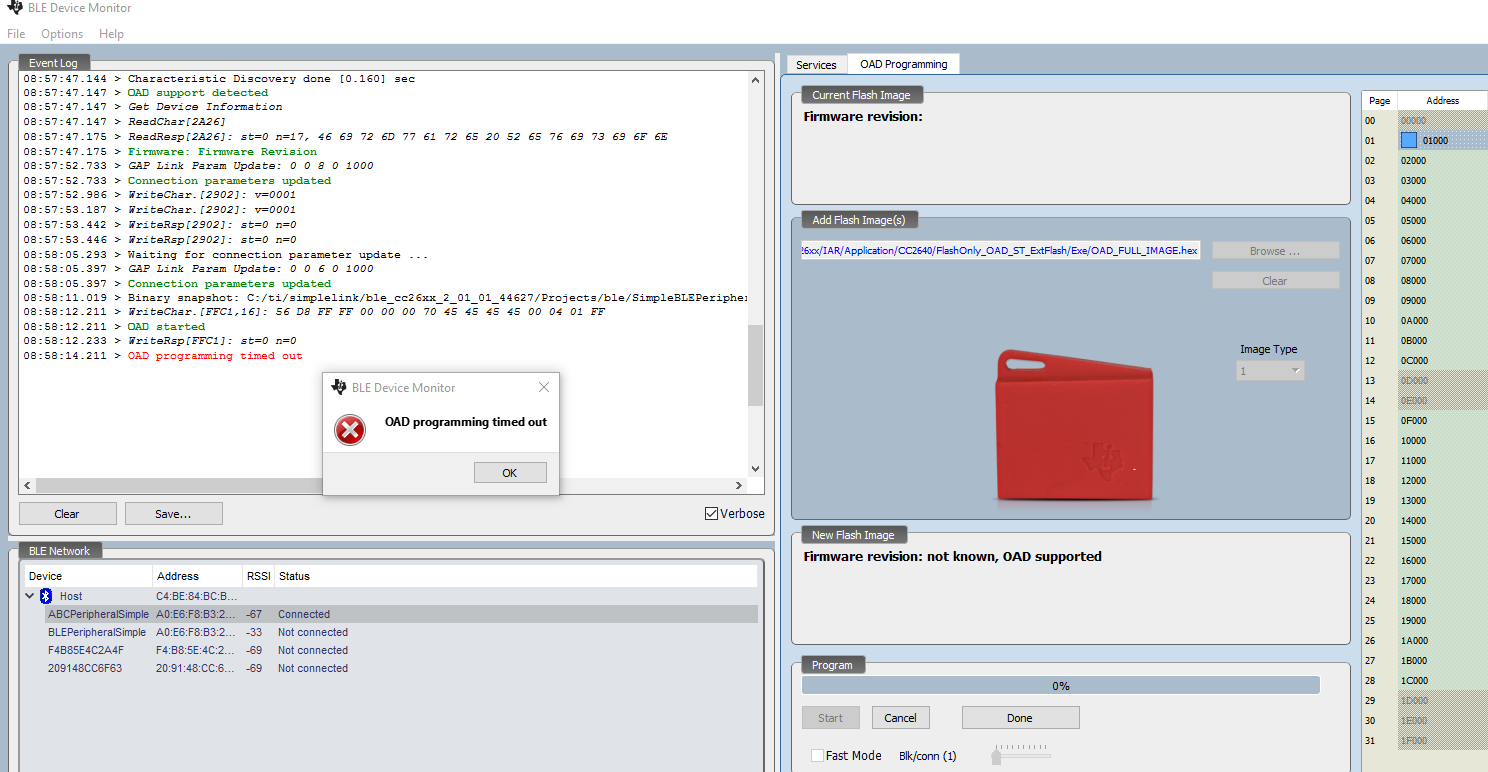


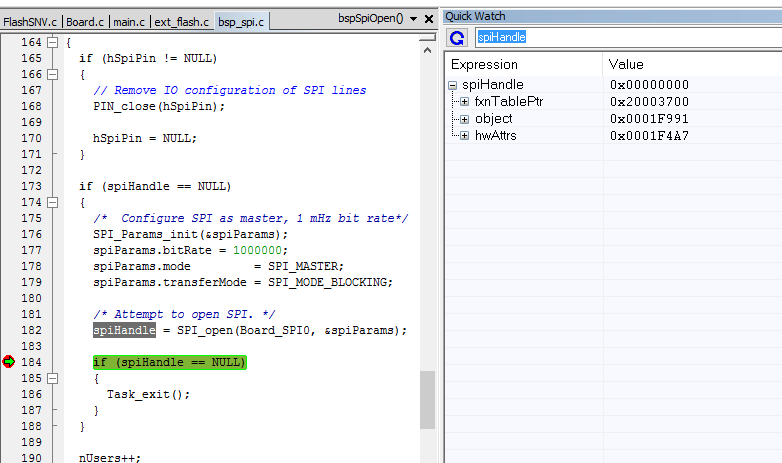
After the OAD success info, board will reset itself. If the board is not resetting itself, it might be a sign of problem.

Do another scan and connect to our board, verify the board FW was updated, as the Firmware revision shown 2 below

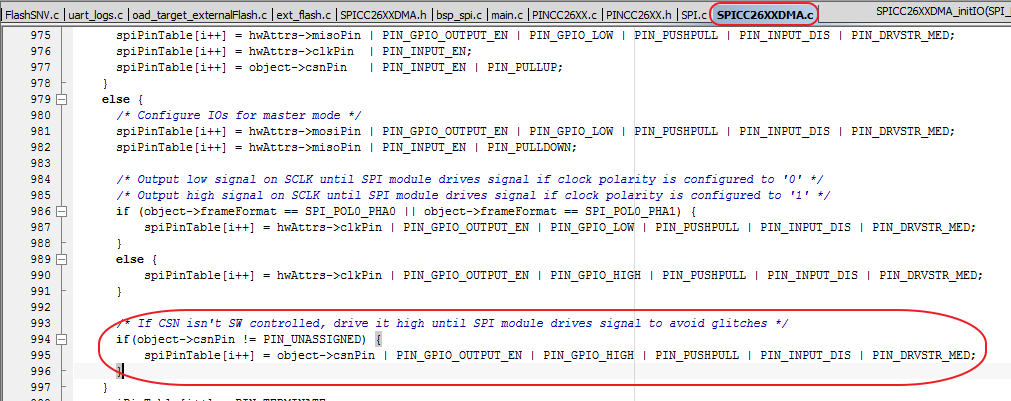


#### Debug the off chip OAD

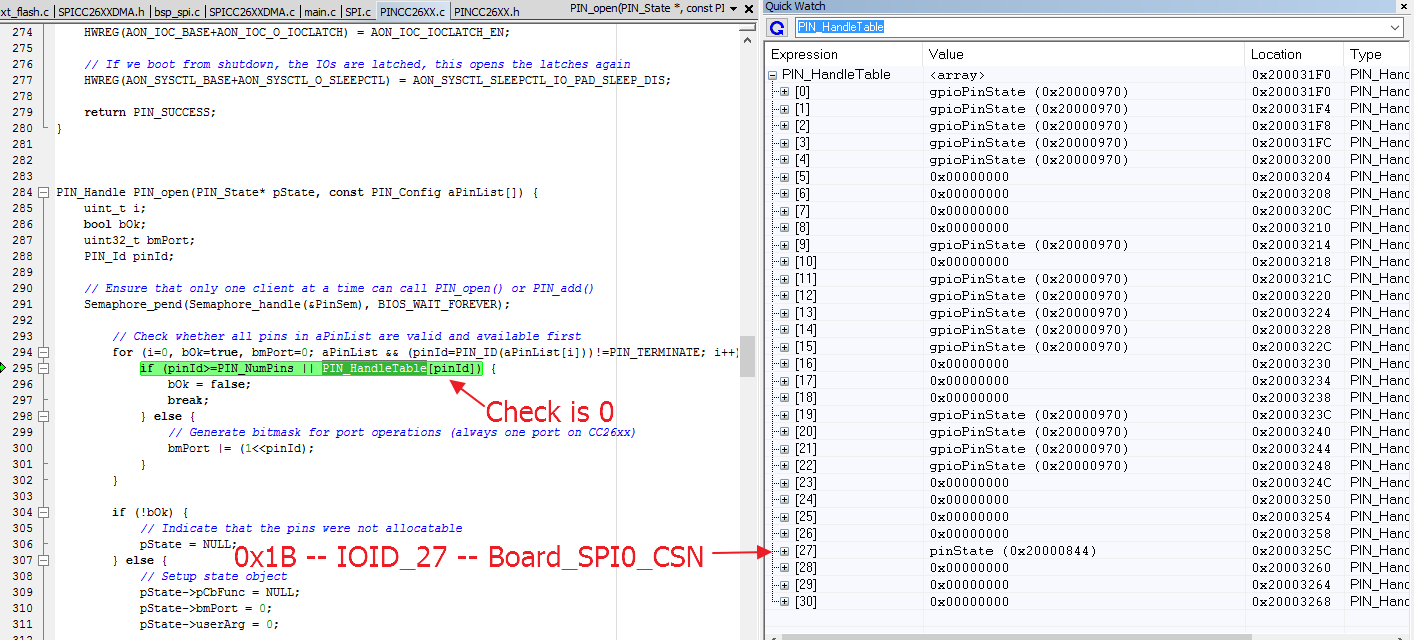




We add the cs pin to open



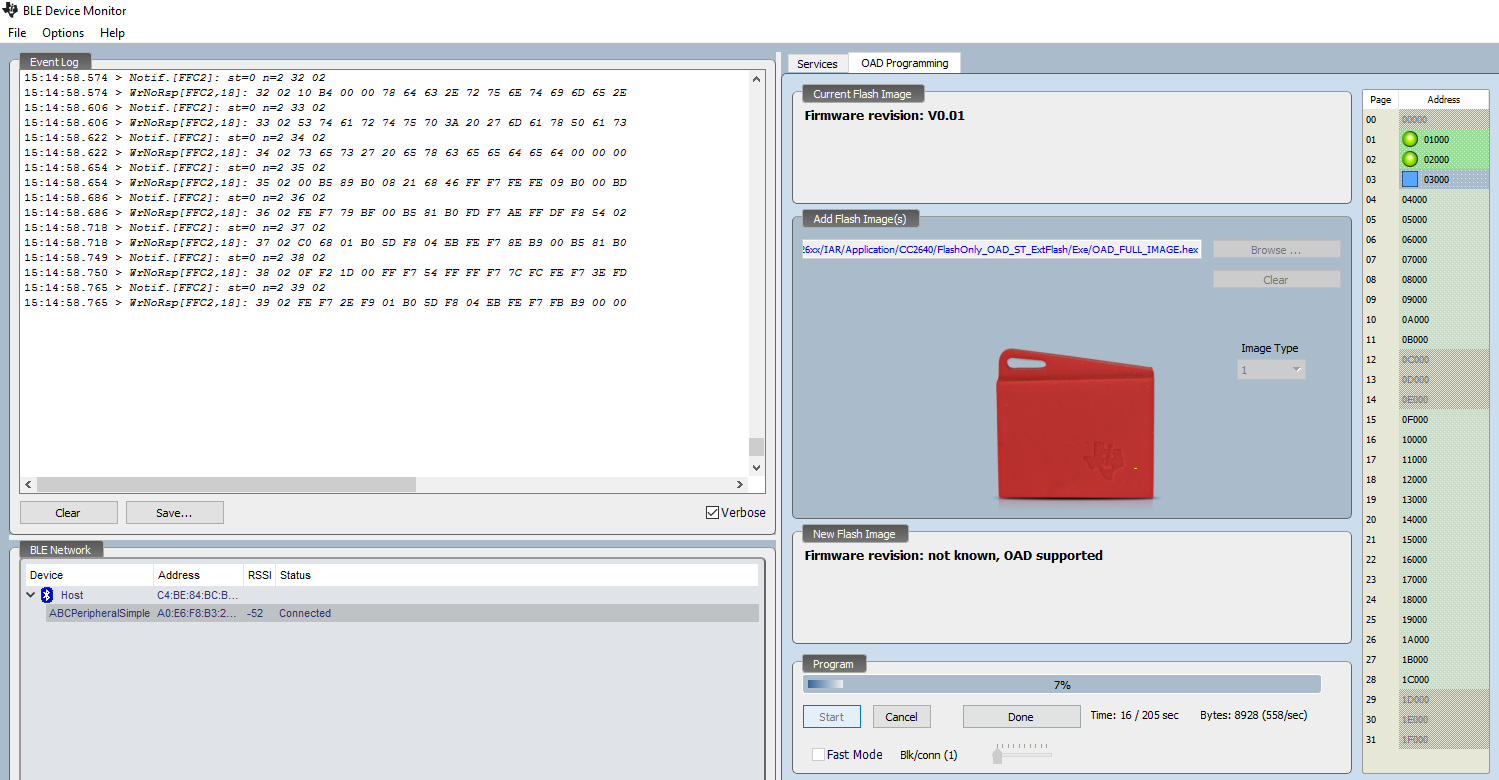
Open will fail here since cs pin is already opened



Solution

In Board.h file

|  |
| --- |
| **#define** Board\_SPI0\_CSN PIN\_UNASSIGNED |



|  |
| --- |
| Actually there are two slightly different off-chip OAD models. One is used in SensorTag OAD sample application and the other is used in SimpleBLEPeripheral OAD sample application runnng on SensorTag hardware. The OAD User's Guide describes only the latter one which is more fail-safe than the former one.  The differences between SensorTag OAD sample app and SBP OAD sample app are as follows:  <SensorTag OAD sample app> - Works with the BIM of the configuration 'BIM\_extflash - FlashOnly\_SensorTag' - The application image starts at 0x00000 - When reset, the application starts. - When OAD completes, the application calls the BIM so that it copies the downloaded image from the off-chip flash to the on-chip flash and reset the system.  - Works with Android/iOS SensorTag app for OAD  <SBP OAD sample app running on SensorTag HW> - Works with the BIM of the configuration 'BIM\_extflash - FlashOnly' - The application image starts at 0x01000 - When reset, the BIM starts and copies the image from the off-chip flash to the on-chip flash if any. Then, jumps to the application image. - When OAD completes, the application reset the system. - Works with Windows BLE Device Monitor application for OAD  Hence, which model is more proper for your product should be considered first. Once the model is chosen, please start with the sample application corresponding to the chosen OAD model. |

<https://e2e.ti.com/support/wireless_connectivity/bluetooth_low_energy/f/538/t/448400>

Try use the BIM\_extflash – FlashOnly\_SensorTag configuration

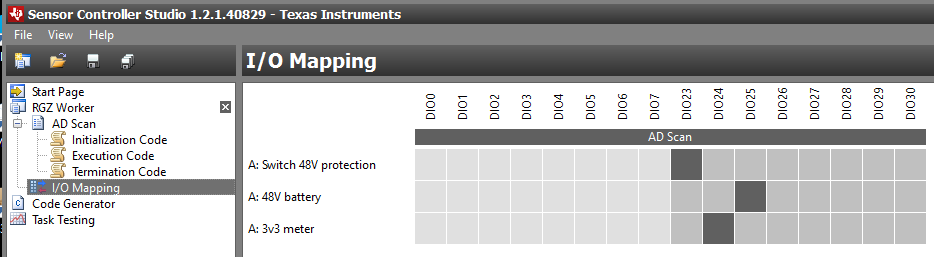
Try OAD the super image

Try disconnect all debug pins while doing OAD

|  |
| --- |
| (1) Config BIM project to FlashOnly\_SensorTag (2) DON'T follow the official guide to change .icf file ! You MUST use the original one ! (3) DON'T change APP\_IMAGE\_START ! (4) Modify something, change FW\_VERSION, and re-build all three projects, BIM, App, and Stack. (5) Combine ALL THREE .hex files into one OAD .bin image by using JFlash. (6) Use Android app to do OAD with this .bin image  Indeed, I did nothing except for (1) and (5). Yes, that's it. |

Finally check if the Sensor controller is using any pins conflict with the Flash SPI pins, otherwise the OAD will fail.

Screenshot of RGZ\_B board sensor controller ADC pins assignment



#### About interrupting OAD process

Interrupting the OAD process by stopping the Android App,

The target need a manual reset before restarting the process again.

This manual reset is needed since the external flash is opened and cannot be reopened again in the official OAD logic. This is a limitation of the official OAD source code, and could be customized to remove the limitation.

The flash open is inside oad\_target\_externalFlash.c

|  |
| --- |
| **void** **OADTarget\_getCurrentImageHeader**(img\_hdr\_t \*pHdr) |

This limitation is resolved by checking for previous incomplete OAD and close the Flash first before proceed to open up the flash again. Highlighted change is shown below in the Profiles\OAD\CC26xx\oad.c file

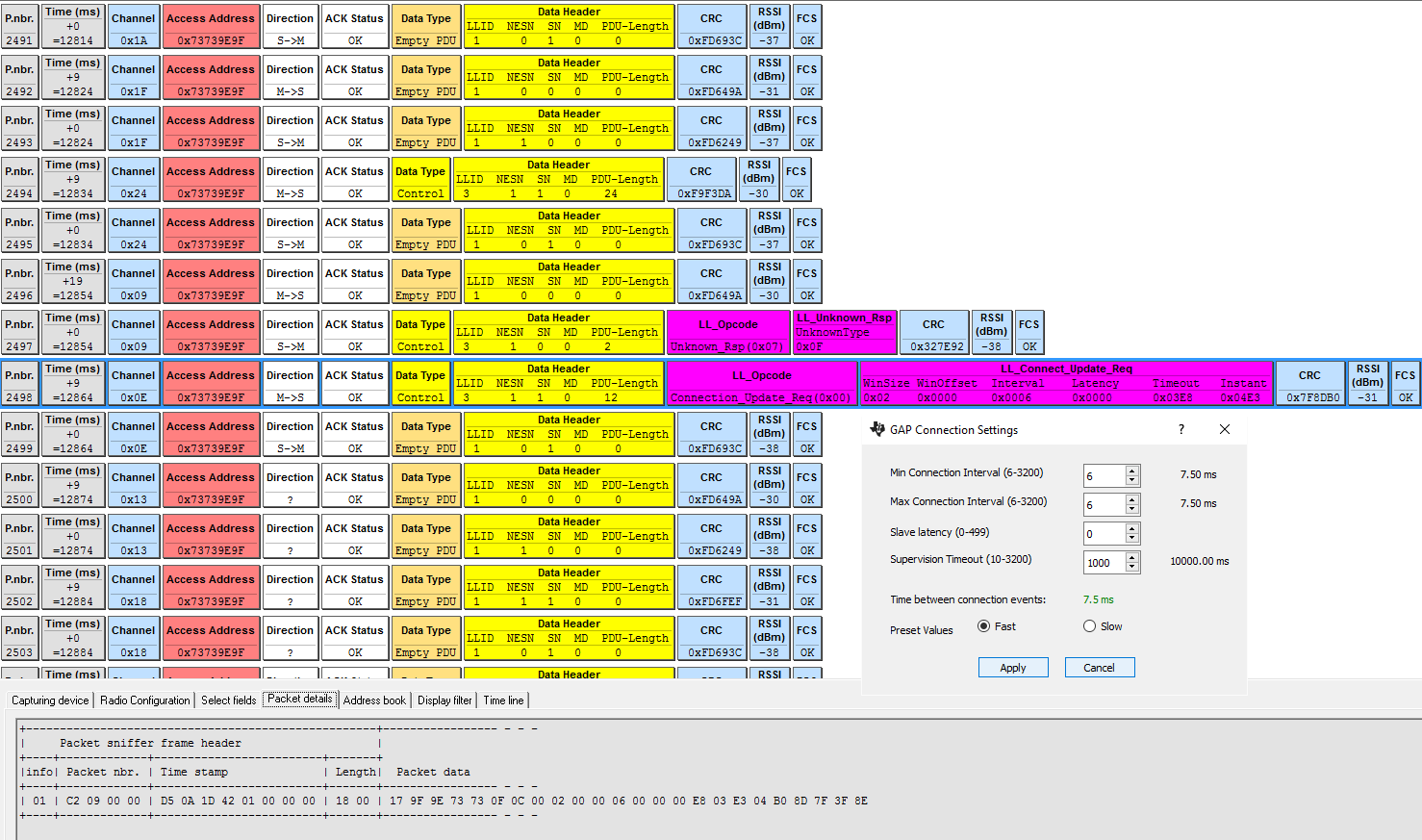
|  |
| --- |
| **static** uint8\_t oadTargetOpened = 0;  **. . .**  **void** **OAD\_imgIdentifyWrite**(uint16\_t connHandle, uint8\_t \*pValue)  {  img\_hdr\_t ImgHdr;  uint8\_t hdrOffset = 0;    **if** (OADTarget\_hasExternalFlash())  {  hdrOffset = 4;  }    // Store the new image's header  OADTarget\_storeImageHeader(pValue);  //check if there is a incomplete OAD, then close the Flash first  **if**(oadTargetOpened > 0){  extFlashClose();  }  // Read out running image's header.  OADTarget\_getCurrentImageHeader(&ImgHdr);    // Calculate block total of the new image.  oadBlkTot = BUILD\_UINT16(pValue[hdrOffset + 2], pValue[hdrOffset + 3]) /  (OAD\_BLOCK\_SIZE / HAL\_FLASH\_WORD\_SIZE);  oadBlkNum = 0;  /\* Requirements to begin OAD:  \* 1) LSB of image version cannot be the same, this would imply a code overlap  \* between currently running image and new image.  \* 2) Total blocks of new image must not exceed maximum blocks supported, else  \* the new image cannot fit.  \* 3) Block total must be greater than 0.  \* 4) Optional: Add additional criteria for initiating OAD here.  \*/  **if** (OADTarget\_validateNewImage(pValue + hdrOffset, &ImgHdr, oadBlkTot))  {  // Determine where image will be stored.  imageAddress = OADTarget\_imageAddress(pValue+hdrOffset);  imagePage = imageAddress / HAL\_FLASH\_PAGE\_SIZE;  // Open the target interface  oadTargetOpened = OADTarget\_open();  // Image accepted, request block 0.  OAD\_getNextBlockReq(connHandle, 0);  } |

### About updating device info

Inside PROFILES/devinfoserivice.c

|  |
| --- |
| // Firmware Revision String characteristic  **static** uint8 devInfoFirmwareRevProps = GATT\_PROP\_READ;  **static** uint8 devInfoFirmwareRev[DEVINFO\_STR\_ATTR\_LEN+1] = "V0.02";  // Hardware Revision String characteristic  **static** uint8 devInfoHardwareRevProps = GATT\_PROP\_READ;  **static** uint8 devInfoHardwareRev[DEVINFO\_STR\_ATTR\_LEN+1] = "RGZ\_\*";  // Software Revision String characteristic  **static** uint8 devInfoSoftwareRevProps = GATT\_PROP\_READ;  **static** uint8 devInfoSoftwareRev[DEVINFO\_STR\_ATTR\_LEN+1] = "Ver0.02";  // Manufacturer Name String characteristic  **static** uint8 devInfoMfrNameProps = GATT\_PROP\_READ;  **static** uint8 devInfoMfrName[DEVINFO\_STR\_ATTR\_LEN+1] = "DongDa"; |

### Change GAP Connection parameters

Write

### About porting Sensortag mobile app

#### Generate Binary file for Android

Create post build script named **OffChipOADPostBuild.bat** to generate the binary file needed for the OAD, the OFFCHIP\_APP\_OAD.bin will be what we needed for mobile OAD

This Post Build script will generate

**OFFCHIP\_APP\_OAD.bin** for Mobile OAD

**OAD\_FULL\_IMAGE.hex** for BLE Device monitor to OAD both Stack and Application

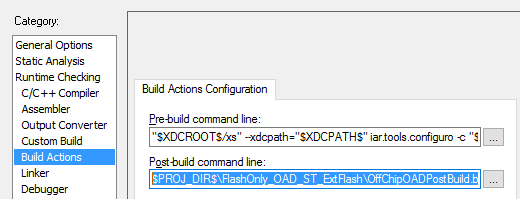
**SUPER\_FULL\_IMAGE.hex** to be flashed to chip first time using Flash Programmer 2

|  |
| --- |
| @**ECHO** OFF  **SET** PROJ\_DIR**=%1**  **SET** BINFILE**=**OFFCHIP\_APP\_OAD.bin  **ECHO** project directory is "**%PROJ\_DIR%**"  **ECHO** merge app and stack hex files to one file  python "C:\Python27\Scripts\hexmerge.py" -o "**%PROJ\_DIR%**\FlashOnly\_OAD\_ST\_ExtFlash\Exe\OAD\_FULL\_IMAGE.hex" -r "1000:1CFFF" --overlap**=**error "**%PROJ\_DIR%**\FlashOnly\_OAD\_ST\_ExtFlash\Exe\SimpleBLEPeripheral\_OADExtFlash.hex":1000:EFFF "**%PROJ\_DIR%**\..\..\Stack\CC2640\FlashROM\Exe\SimpleBLEPeripheralStackFlashROM.hex":F000:1CFFF  **ECHO** Generate OAD binary file "**%BINFILE%**"  python "C:\Python27\Scripts\hex2bin.py" -r "1000:BFFF" "**%PROJ\_DIR%**\FlashOnly\_OAD\_ST\_ExtFlash\Exe\SimpleBLEPeripheral\_OADExtFlash.hex" "**%PROJ\_DIR%**\FlashOnly\_OAD\_ST\_ExtFlash\Exe\**%BINFILE%**"  **ECHO** Generate OAD super hex file "SUPER\_FULL\_IMAGE.hex"  python "C:\Python27\Scripts\hexmerge.py" -o "**%PROJ\_DIR%**\FlashOnly\_OAD\_ST\_ExtFlash\Exe\SUPER\_FULL\_IMAGE.hex" -r "0000:1FFFF" --overlap**=**error "**%PROJ\_DIR%**\FlashOnly\_OAD\_ST\_ExtFlash\Exe\SimpleBLEPeripheral\_OADExtFlash.hex":1000:1EFFF "**%PROJ\_DIR%**\..\..\Stack\CC2640\FlashROM\Exe\SimpleBLEPeripheralStackFlashROM.hex":1000:1EFFF "**%PROJ\_DIR%**\..\..\..\..\..\..\ble\util\BIM\_extflash\CC26xx\IAR\FlashOnly\Exe\BIM\_ext.hex":0000:1FFFF |

Those generated hex files reside in directory

C:\ti\simplelink\ble\_cc26xx\_2\_01\_01\_44627\Projects\ble\SimpleBLEPeripheral\CC26xx\IAR\Application\CC2640\FlashOnly\_OAD\_ST\_ExtFlash\Exe

Then change the setting in IAR to use this post build script



|  |
| --- |
| $PROJ\_DIR$\FlashOnly\_OAD\_ST\_ExtFlash\OffChipOADPostBuild.bat $PROJ\_DIR$ |

## Security related issue

那么为了防止其他APP 恶意连接，光靠设置配对密码是无效的。

我们一般的做法是，连接成功后需要APP主动发送一包特定的数据给slave，如果slave没有收到这包数据，主动发起断开连接request断开连接。

[CC2541之配对与绑定](http://blog.csdn.net/feilusia/article/details/50212945http:/blog.csdn.net/feilusia/article/details/50212945)

**修改特征值char2在属性表中的属性,**

**This change make the read action needs bond/pair**

|  |
| --- |
| GATT\_PERMIT\_AUTHEN\_READ,//GATT\_PERMIT\_READ, |

## TI RTOS GPTimer Driver

[Pre-release CC26XX / CC13XX PWM and GPTimer driver](https://e2e.ti.com/support/wireless_connectivity/bluetooth_low_energy/f/538/t/474000)

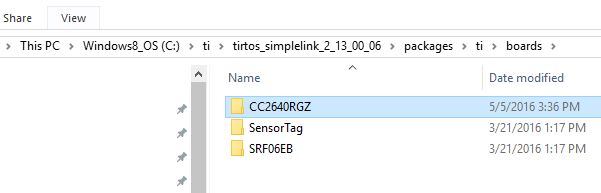
|  |
| --- |
| **Usage**   * Add the necessary hardware descriptors, objects and configuration to your board files as shown in **Board.c/h**. * Add a compiler search path to the drivers base or add the drivers to your TI RTOS installation(**C:\ti\tirtos\_simplelink\_2\_13\_00\_06\packages\ti\drivers\timer**) * Add the driver source to your workspace * Follow the examples included (main\_\*.c), there is also an example to trigger timer periodic interrupt inside file **GPTimerCC26XX.h** |

[Another Timer PWM example](https://e2e.ti.com/support/wireless_connectivity/bluetooth_low_energy/f/538/p/405132/1435673#1435673)

[Local timer library DOC](file:///C:\ti\tirtos_cc13xx_cc26xx_2_16_00_08\products\cc26xxware_2_23_01_16780\doc\driverlib\group__timer__api.html)

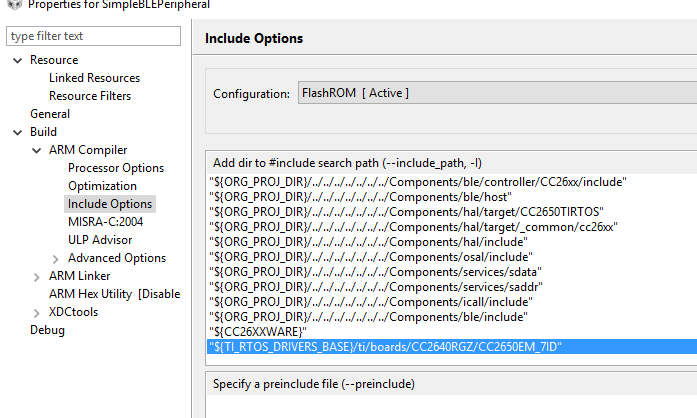
## Customer Board files

First make a copy of SRF06EB folder and rename it to our **customer board CC2640RGZ**,

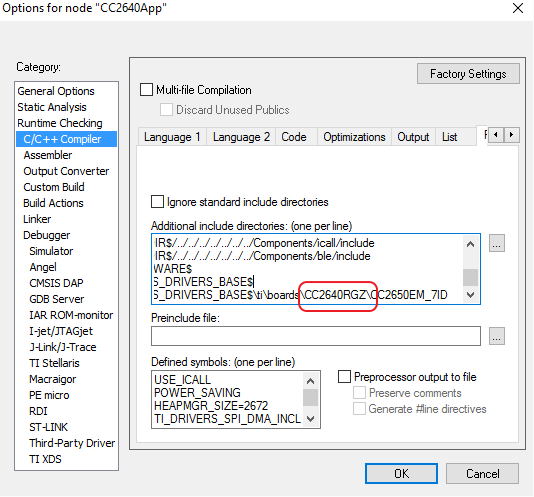


Change the **include path**

CCS

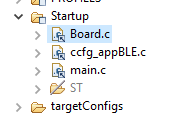


IAR

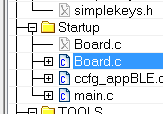


Change the linked **Board.c** file

CCS



IAR



## Flash Simple NV

Open up the stack project, In the predefined symbol, define

**OSAL\_SNV=2**

Inside bcomdef.h

|  |
| --- |
| /\*\* @defgroup BLE\_NV\_IDS BLE Non-volatile IDs  \* @{  \*/  // Device NV Items - Range 0 - 0x1F  **#define** BLE\_NVID\_IRK 0x02 //!< The Device's IRK  **#define** BLE\_NVID\_CSRK 0x03 //!< The Device's CSRK  **#define** BLE\_NVID\_SIGNCOUNTER 0x04 //!< The Device's Sign Counter  // Bonding NV Items - Range 0x20 - 0x5F - This allows for 10 bondings  **#define** BLE\_NVID\_GAP\_BOND\_START 0x20 //!< Start of the GAP Bond Manager's NV IDs  **#define** BLE\_NVID\_GAP\_BOND\_END 0x5f //!< End of the GAP Bond Manager's NV IDs Range  // GATT Configuration NV Items - Range 0x70 - 0x79 - This must match the number of Bonding entries  **#define** BLE\_NVID\_GATT\_CFG\_START 0x70 //!< Start of the GATT Configuration NV IDs  **#define** BLE\_NVID\_GATT\_CFG\_END 0x79 //!< End of the GATT Configuration NV IDs  // Customer NV Items - Range 0x80 - 0x8F - This must match the number of Bonding entries  **#define** BLE\_NVID\_CUST\_START 0x80 //!< Start of the Customer's NV IDs  **#define** BLE\_NVID\_CUST\_END 0x8F //!< End of the Customer's NV IDs |

The **osal\_snv\_read / write** can be executed in the same task as the **SimpleBLEPeripheral\_taskFxn**

Or any other tasks registered with **ICall**.

# Sensor Controller

To execute a program on the sensor controller, the program image must first be uploaded to the AUX  
RAM by the system CPU or DMA.

The Sensor Controller is set up using a PC-based configuration tool, called Sensor Controller Studio, and  
potential use cases may be (but are not limited to):  
• Analog sensors using integrated ADC  
• Digital sensors using GPIOs, bit-banged I2C, and SPI  
• UART communication for sensor reading or debugging  
• Capacitive sensing  
• Waveform generation  
• Pulse counting  
• Keyboard scan  
• Quadrature decoder for polling rotation sensors  
• Oscillator calibration

Pop-ups can also be triggered by moving the cursor to the procedure call and pressing *Ctrl+Space*.

**Data Structure Members**

Data structure members are added or edited in a separate pop-up window:

* **Data structure**: Select one of the following:
* **cfg**: Used to perform run-time configuration of the task, before the task is started
* **input**: Used to pass input data to the task (e.g. dynamic parameters for an external sensor)
* **output**: Used to pass output data to the System CPU application (e.g. accelerometer data)
* **state**: Internal variables used to store the task's state between iterations
* **Member name**: Enter the name of the data structure member
* **Description**: Enter an optional description of the data structure member. This text is included in generated driver source code documentation.
* **Type**: Select one of the following:
* **16-bit unsigned, decimal**
* **16-bit signed, decimal**
* **16-bit unsigned, hexadecimal** (equivalent to **16-bit unsigned, decimal**, but the value is diplayed differently in Sensor Controller Studio)
* **1-bit unsigned, boolean** (equivalent to **16-bit unsigned, decimal**, but the value is diplayed differently in Sensor Controller Studio)
* **Value**: Enter a numeric value or an expression using the same syntax as for constant values. For example:
* 123
* PIN\_COUNT \* 2
* ((A\_PARAM + B\_PARAM) & 0x003E) | (0x8000 | C\_PARAM)

**Array size**: Select "Not array" for single data structure members, or the name of a user-defined constant for array data structure members

To create a pointer to a data structure member, use the # operator.

## Add sensor controller code to existing project

We have set the SCS to generate code inside folder Source/SC\_Src

Then we add the path to included directories in both IDEs(IAR, CCS)

