Qualcomm Technologies International, Ltd.

QCC711 Bluetooth Low Energy Manufacturing Flow Overview

Application Note

80-61698-1 Rev. AF

March 19, 2025

Qualcomm Technologies International, Ltd. is a company registered in England and Wales
with a registered office at: Churchill House, Cambridge Business Park, Cowley Road,
Cambridge, CB4 0WZ, United Kingdom.
Registered Number: 3665875 | VAT number: GB787433096

Revision history

Revision	Date	Change reason
AA	March 2023	Initial release.
AB	March 2023	Updated graphic in section on creating the OEM Manifest and code line in step 4 of section on APPS image update with NVM programming.
AC	June 2023	Updated release for QCC711.
AD	August 2023	Section on debug locking/ unlocking and preparation for RMA included and minor editorial updates throughout.
AE	November 2023	Minor editorial updates.
AF	March 2025	Editorial updates.

Contents

Tables

Figures

1 Manufacturing flow overview for QCC711

A typical manufacturing flow for QCC711 is illustrated in the figure Manufacturing flow.

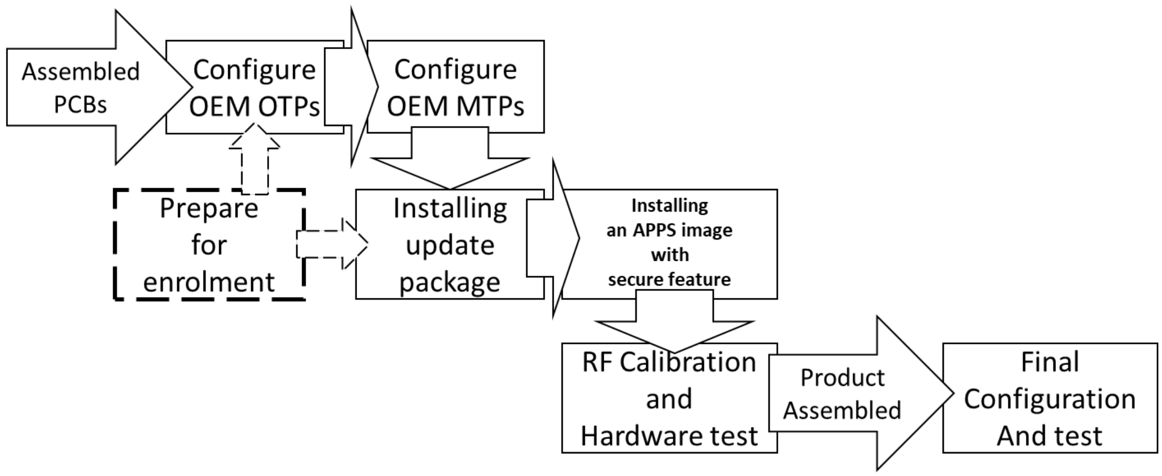


Figure 1-1 Manufacturing flow for QCC711

In general, the flow includes the following phases:

1. Assembled PCBs:
2. Refer to *Typical Solder Reflow Profile for Lead-Free Devices Application Note* (80-CT462-1) for guidance on solder reflow.
3. The QCC711 requires access to the SWD pins (GPIO9 to GPIO12) to program the application code, as well as to the necessary OTP and MTP configurations. To run the QCLI demo for the manufacturing test, it is recommended to reserve UART test points.
4. To measure the crystal frequency, the QCC711 can present these on two GPIOs. The 32 kHz crystal is presented on GPIO13 (TEST_LF) and the 32 MHz is on GPIO11 (TEST_HF). A VSS pin is required to reference these digital signals.
5. The board configuration and testing is conducted before a power source is mounted, thus, a test point for VDD is required. Refer to *QCC711 Bluetooth Low Energy Data Sheet* (80-WL711-1) for further details.
6. Preparations for enrollment
7. To have the material required to create a device image and to be provisioned in OTP to enable its execution at enrollment, the OEM must generate the OEM_MRC_HASH (hash of the OEM root certificate) and OEM_BATCH_SECRET_HASH (hash key of the OEM product batch). QTIL can then provide the OEM with an OEM_ID. Once the enrollment process is completed, the OEM is able to generate an update package.

8. This step should be performed before the development process starts. Additionally, it should be performed in a safe and trustworthy environment, not on the production line. Concerning the use of any of the tools, refer to *QCC711 Bluetooth Low Energy Software Programming Guide* (80-61032-1).
9. OEM OTP configuration:
10. Once the enrollment has been completed, the OEM must commit the items generated during the enrollment to each QCC711 device on the production line. The test environment can be separated from the RF calibration and hardware test as it requires a Windows PC containing python 2.7 (preferably version 2.7.17), python 3, GNU Arm embedded tools (version `9-2020-q2` update is recommended), J-Link software (version 6.98d is recommended), and a Segger J-Link debugger (J-Link base 8.08.00 from Segger).
11. The QCC711 SWD interface is used at this stage.
12. Configure MTP fields:
13. In general, this stage involves the configuration of the necessary MTP fields (such as Bluetooth device address (BD_ADDR), the status of GPIOs, and so on) to be used later on while the device is running in mission mode. The OEM needs to program these MTP fields to each QCC711 device on the production line. For this stage, the test environment from stage 3 can be leveraged. The SWD interface of the QCC711 is also used.
14. Installing an Update Package:
15. To update patches, an update package needs to be applied. Once the update package has been generated, the OEM can install it onto each QCC711 device on the production line or through an OTA update. This document focuses on the process of how the OEM can install the update package on the production line and how to leverage the test environment from stage 3 for this. As before, the SWD interface of the QCC711 is also used.
16. Installing an APPS image with secure feature
17. For daily development purposes, the OEM can install a new APPS image on the QCC711 using IAR without having to sign the image every time a minor change is performed. To finalize the device image for testing or production and if confidential device image programming is not an issue, the NVM programmer can be used to program the image onto the QCC711 using a plain text image. Furthermore, APPS image authentication can be enabled when production ready. If device image confidentiality is a concern, the secure NVM programmer tool can be used instead to program encrypted application images.
18. RF calibration and hardware test:
19. Usually, this stage requires RF measurement equipment or a reference unit as well as other test hardware. Hence, it is recommended that this stage is separated from stages 2 to 4.
20. For RF calibration, we recommend that the OEM runs a per-unit crystal trim to achieve the optimized QCC711 power consumption performance. The system requires both crystals (32.768 kHz and 32 MHz) to be trimmed to a target of +/- 4 ppm, followed by RF tests and other hardware tests such as PIOs and LEDs. To avoid costly rework, QTIL recommends testing thoroughly before assembling the product in plastics. The SWD and UART interfaces of the QCC711 are used at this stage.
21. Final configuration and test
22. The product is now in plastics and can be tested as a product to ensure that all the functions work as expected. This stage depends on the different applications that the OEM is running on the QCC711. QTIL leaves this stage to the OEM to carry on according to their own designs.

2 Preparations for QCC711 enrollment

This section introduces the necessary enrollment process. The OEM must generate the `OEM_MRC_HASH` and `OEM_BATCH_SECRET_HASH` files. The creation process of these files is covered in sections on Creation of OEM_MRC_HASH for QCC711 and Creation of OEM_BATCH_SECRET_HASH for QCC711.

Once the OEM has generated the `OEM_MRC_HASH` and `OEM_BATCH_SECRET_HASH` files for enrollment, QTIL shares the OEM-specific OEM_ID. The OEM is ready to create an update package. Instructions for these processes are in sections on creating an `OEM_Manifest` and creating an update package.

NOTE If OEM_ID is not written and the chipset is locked in production, the chipset cannot be debugged afterwards.

2.1 Creation of OEM_MRC_HASH for QCC711

This section provides the steps for creating image singing keys and for producing the `OEM_MRC_HASH`. This process should be executed once in a safe and trustworthy environment. Later, once the `OEM_MRC_HASH` has been created, the OEM needs to commit it into the QCC711 OTP field for every unit on the production line.

Sigining keys generation

The signing key pair must be defined on the NIST P-521 ECC curve and encoded in PKCS#8 format. The following reference example uses OpenSSL to create the key pair:

- Generating a private key:

  ```
  $ openssl ecparam -name secp521r1 -genkey -noout -out privatekey.pem
  ```

- Extracting a public key:

  ```
  $ openssl ec -in privatekey.pem -pubout -out publickey.spki
  ```

- Converting the private key to PKCS#8 format:

  ```
  $ openssl pkcs8 -topk8 -in privatekey.pem -nocrypt -outform pem -out
  privatekey-pkcs8.p8
  ```

Creating the OEM Root Entitlement Certificate

First, download the respective SDK from ChipCode. For the creation of the OEM Root Entitlement Certificate, navigate to SDK folder`\\rot\tools\QCC710-Signing`.

Copy `privatekey-pkcs8.p8` to`\\rot\tools\QCC710-Signing\p521-private-key.p8` and copy `publickey.spki` to`\\rot\tools\QCC710-Signing\p521-public-key.spki`.

Execute the following command using the `cert_qcc710.py` tool:

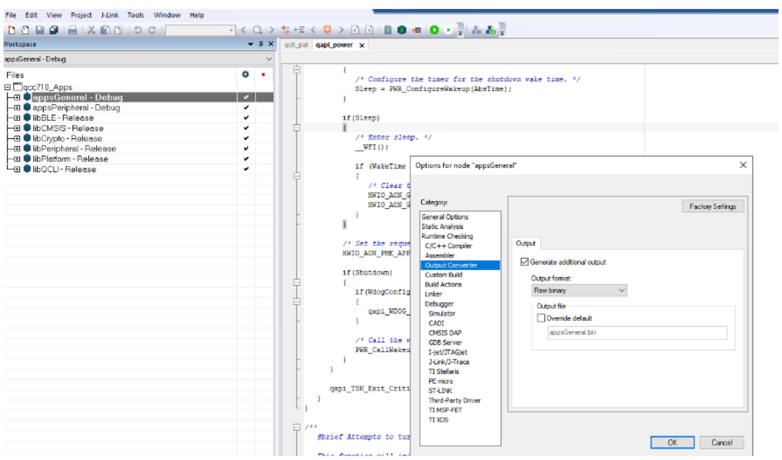
```
>python3 ./cert_qcc710.py --privkey p521-private-key.p8 --pubkey p521-public-key.spki -c ./config/OEM-REC-QCC710.json -o OEM-REC-QCC710
```

As a result, files `OEM-REC-QCC710.bin`, `OEM-REC-QCC710.hex`, and `OEM-REC-QCC710.json` are generated. Place them in the SDK folder`\\rot\tools\QCC710-Signing\input`.

Creating the OEM Manifest

Refer to steps 1, 2, and 3 for material preparation. All materials must be placed into the SDK folder`\\rot\tools\QCC710-signing` for creating the OEM Root Entitlement Certificate.

1. Copy keys `p521-public-key.spki` and `p521-private-key.p8` to SDK folder`\\rot\tools\QCC710-Signing\key`.
2. Run`\\qcc711_sdk\iar\qcc710_Apps.eww` from the installed SDK.
3. Use the following steps to generate the appsGeneral.bin file and to place it in the SDK folder`\\rot\tools\QCC710-Signing\input`:
   a. As illustrated in **Figure**, right-click on ‘appsGeneral’ in **IAR** → **Debug**, and select **Options**.
   b. Navigate to **Output Converter**, select ‘Generate additional output’, and set the Output format to ‘Raw binary’. Then, click **OK**.
   c. The generated 'appsGeneral.bin' file is in`\\qcc711_sdk\iar\apps\Debug\Exe\`.



4. Copy the file `btcfg_app_mode.bin` from the SDK folder`\\qcc711_sdk\bin` to the SDK folder `\\rot\tools\QCC710-Signing\input`.

Once the previous preparations have been completed, navigate to the SDK folder`\\rot\tools\QCC710-Signing` and then execute the following command using `sign_qcc710.py` tool:

```
>python3 ./sign_qcc710.py -t manifest -c .\input\OEM-REC-QCC710.json -b .\config\OEM-IAR-BMHT.json --privkey .\key\p521-private-key.p8 --pubkey .\key\p521-public-key.spki -C .\input\btcfg_app_mode.bin -A .\input\appsGeneral.bin -o .\output\APSS_OEM_Manifest_unlicense.bin
```

After these steps, the `OEM_MRC_HASH` has been generated inside the `APPS_OEM_Manifest_unlicense.hash` file. Users can see this on the row marked ‘sign:’, then followed by the `OEM_MRC_HASH`, or on the row marked ‘otp:’, followed by the `OEM_MRC_HASH` in 2 bytes reverse order.

The OEM shares the `OEM_MRC_HASH` from the ‘sign’ row with QTIL for the enrollment. Also, the OEM commits the 2 bytes reverse `OEM_MRC_HASH` from the ‘otp’ row into the ‘OTP’ field for each QCC711 device on the production line. Instructions for this are covered in later sections in this set.

NOTE `APSS_OEM_Manifest_unlicense.bin` is not needed in any further steps, so it can be deleted.

2.2 Creation of OEM_BATCH_SECRET_HASH for QCC711

The OEM picks a 128-bit random sequence for the `OEM_BATCH_SECRET` (note that this is NOT to be shared with QTIL). Keep this 128-bit random number safe as it is provisioned in the OTP later. Compute the `OEM_BATCH_SECRET_HASH` with sha512 encryption using the following command:

```
$ echo -n 8ba1a85b6e0b66d88d77d7a2bb5f8774 |xxd-r -p| openssl dgst –sha512
```

where '`0x8ba1a85b6e0b66d88d77d7a2bb5f8774`' is an example of a randomly selected 128-bit sequence.

QTIL encourages the OEM to use a secure wrap tool for programming the `OEM_BATCH_SECRET` in the factory as it raises the security level for the protection of the OEM IP. If the OEM is not planning to use the secure wrap tool, then the following steps can be skipped to proceed directly to Section 3.2.

To wrap the secure sources in the binary file, QTIL provides the factory_provisioning_key for the OEM at enrollment. The wrapping process is then done according to the following steps:

1. The OEM picks a random 256-bit sequence as the batch_secret_wrapping_key and an `OEM_PRODUCT_SEED` as a diversifier.
2. Using the `factory_provisioning_key`, `OEM_PRODUCT_SEED`, and `batch_secret_wrapping_key`, update the `config.json` found in the SDK folder`\\rot\tools\QCC710-SecureWrap\securewrap`. The following is an example of the contents of a `config.json` file:

```
 Config.json(
{
   "entity_dev_fact_prov_key" : "256 random bits from QTIL",
   "oem_batch_secret" : "128 random bits chosen by OEM",
   "batch_secret_wrapping_key" : "256 random bits chosen by OEM",
   "NISTKDFContext" : {
   "oem_product_seed" : "128 random bits chosen by OEM",
   "is_oem_lcs_activated" : false,
   "oem_lcs" : 0
 }
})
```

3. Refer to`\\rot\tools\QCC710-SecureWrap\ README.md` for installation instructions. Only install the 3.4.7 version of the cryptography module.
4. Generate the `secure_otp.bin` with the following command:

```
>python3 ./secure_wrap.py -j config.json -o secure_otp.bin
```

Once the `secure_otp.bin` has been generated, the OEM needs to use the `OTP_Programmer.py` to program the `OEM_BATCH_SECRET` onto the board using the generated `secure_otp.bin`. Refer to section on **OEM OTP configuration with secure wrap tool** for details.

2.3 Creation of OEM_Manifest for QCC711

The OEM is then ready to create an `OEM_Manifest` (refer to this section) and a signed update package (refer to the section on the creation of an update package).

Create the `OEM_Manifest` with a license with the following steps:

1. Copy the file `qti_manifest-app_mode-unlicensed.bin` to the SDK folder`\\rot\tools\QCC710-Signing\input`.
2. Move the files `btcfg_app_mode.bin` and `btss_patch_app_mode.bin` from the SDK folder`\\qcc711_sdk\bin` to the SDK folder`\\rot\tools\QCC710-Signing\input`.
3. Move `patch.bin` from the SDK folder`\\rot\images` to the SDK folder`\\rot\tools\QCC710-Signing\input`.

```
>python3 ./sign_qcc710.py -t manifest -c .\input\OEM-REC-QCC710.json -b
.\config\OEM-IAR-BMHT.json --privkey .\key\p521-private-key.p8 --pubkey
.\key\p521-public-key.spki -C .\input\btcfg_app_mode.bin -A .\input
\appsGeneral.bin -o
.\output\APPS_OEM_Manifest_license.bin
```

As a result, `APPS_OEM_Manifest_license.bin` is generated. Place it in the SDK folder`\\rot\tools\QCC710-Signing\input`.

2.4 Creation of an update package

Create a Signed Update Package with a license for BTSS and BTCFG with the following command:

```
>python3 ./sign_qcc710.py -t update -c .\input\OEM-REC-QCC710.json -b .\config
\OEM-IAR-BMHT.json --privkey .\key\p521-private-key.p8 --pubkey .\key\p521-
public-key.spki -Q .\input\qti_manifest-app_mode-unlicensed.bin -B .\input
\btss_patch_app_mode.bin -O .\input\APPS_OEM_Manifest_license.bin -C .\input
\btcfg_app_mode.bin -T .\input\patch.bin -o .\output\update_btss_btcfg.bin
```

An update package `update_btss_btcfg.bin` is generated. Install it to each QCC711 device on the production line.

3 OEM OTP configuration for QCC711

Once the preparations for the OEM onboarding enrollment have been finished, the OEM commits their relevant OTPs for each device on the production line. In the following sections, this process is done using a secure wrap tool and without a secure wrapping tool.

With secure wrap tool

The OEM must commit their `OEM_ID`, `OEM_MRC_HASH`, and `OEM_BATCH_SECRET` files to each QCC711 device on the production line. If a secure wrap tool is used, then `OEM_PRODUCT_SEED` is required.

Navigate to the SDK folder`\\qcc711_sdk\tools\scripts` and execute the following commands to commit the necessary fields:

1. `>python2 .\otp_programmer.py --write OEM_PRODUCT_SEED`

   This is the random 128-bit sequence chosen by the OEM in section on the Creation of OEM_BATCH_SECRET_HASH for QCC711.

   NOTE When using the OTP programmer tool to program the `OEM_PRODUCT_SEED`, assign it in 2-byte reverse order against the `OEM_PRODUCT_SEED` in the `config.json` file.

   For example:

   ```
   OEM_PRODUCT_SEED =
   112233445566778899OOAABBCCDDEEFF>python2 .\otp_programmer.py --write
   OEM_PRODUCT_SEED=0xFFEEDDCCBBAA00998877665544332211
   ```

2. `>python2 .\otp_programmer.py --write OEM_ID`

   This is a unique ID for OEM, assigned by Qualcomm®.

3. `>python2 .\otp_programmer.py --write OEM_MRC_HASH`

   This is derived from the OEM entitlement key pair according to the instructions in the section on the Creation of OEM_MRC_HASH for QCC711.

   NOTE Use the `OEM_MRC_HASH` in 2-byte reverse order from the 'otp' row in the `APPS_OEM_Manifest_unlicense.hash` file.

4. `>python2 .\otp_programmer.py --write -f .\secure_otp.bin`

   This was generated by the OEM according to the instructions in the section on the Creation of OEM_BATCH_SECRET_HASH for QCC711.

Without secure wrap tool

The OEM can program the `OEM_BATCH_SECRET` in plain text without using a secure wrap tool and, instead, using the following commands. However, this is not recommended as it lowers the security protection in the factory.

Navigate to the SDK folder `\\qcc711_sdk\tools\scripts` and execute the following commands in python 2.7 to commit the files:

1. `>python2 .\otp_programmer.py --write OEM_ID`

   This is a unique ID for OEM, assigned by Qualcomm.

2. `>python2 .\otp_programmer.py --write OEM_MRC_HASH`

   This is derived from the OEM entitlement key pair, as instructed in the section on the Creation of OEM_MRC_HASH for QCC711.

   NOTE Use the `OEM_MRC_HASH` in 2-byte reverse order from the 'otp' row in the `APPS_OEM_Manifest_unlicense.hash` file.

3. `>python2 .\otp_programmer.py --write OEM_BATCH_SECRET`

   This is the random 128-bit sequence chosen by the OEM.

   NOTE Assign the `OEM_BATCH_SECRET` in 2-byte reverse order against the `OEM_BATCH_SECRET` chosen in the section on the Creation of OEM_BATCH_SECRET_HASH for QCC711.

   For example:

   ```
   OEM_BATCH_SECRET =
   112233445566778899OOAABBCCDDEEFF>python2 .\otp_programmer.py --write
   OEM_BATCH_SECRET=0xFFEEDDCCBBAA00998877665544332211
   ```

NOTE Before committing the previously mentioned OTPs to the QCC711, ensure that the values are correct or it may render the boards unusable.

4 OEM MTP configuration for QCC711

The following OTP fields are listed in the current `otp_field_list.json` file. Users can reference this file directly for details about the supported fields.

Table 4-1 MTP fields

Field	Max size (bits)	Read	Write
BD_ADDR	96	Yes	Yes
XTAL_32K_TRIM_CL	9	Yes	Yes
XTAL_32K_GAIN	6	Yes	Yes
LDO_VMA_SEL_PT	1	Yes	Yes
AUX_OEM_CTRL3	15	Yes	Yes
AUX_OEM_DRV_T1	15	Yes	Yes
AUX_OEM_CTRL4_SET0	12	Yes	Yes
AUX_OEM_CTRL4_SET1	12	Yes	Yes
AUX_OEM_DRV_T3	15	Yes	Yes
AUX_OEM_DRV_T4	15	Yes	Yes
GPIO_CFG0	12	Yes	Yes
GPIO_CFG1	12	Yes	Yes
GPIO_CFG2	12	Yes	Yes
GPIO_CFG3	12	Yes	Yes
GPIO_CFG4	12	Yes	Yes

Table 4-1 MTP fields (cont.)

Field	Max size (bits)	Read	Write
`GPIO_CFG5`	12	Yes	Yes
`GPIO_CFG6`	12	Yes	Yes
`GPIO_CFG7`	12	Yes	Yes
`GPIO_CFG8`	12	Yes	Yes
`GPIO_CFG9`	12	Yes	Yes
`GPIO_CFG10`	12	Yes	Yes
`GPIO_CFG11`	12	Yes	Yes
`GPIO_CFG12`	12	Yes	Yes
`GPIO_CFG13`	12	Yes	Yes
`GPIO_CFG14`	12	Yes	Yes
`GPIO_CFG15`	12	Yes	Yes
`GPIO_CFG16`	12	Yes	Yes
`GPIO_CFG17`	12	Yes	Yes
`GPIO_CFG18`	12	Yes	Yes
`GPIO_CFG19`	12	Yes	Yes
`GPIO_CFG20`	12	Yes	Yes
`GPIO_CFG21`	12	Yes	Yes
`GPIO_CFG22`	12	Yes	Yes
`GPIO_CFG23`	12	Yes	Yes
`GPIO_CFG24`	12	Yes	Yes
`GPIO_CFG25`	12	Yes	Yes

Table 4-1 MTP fields (cont.)

Field	Max size (bits)	Read	Write
`GPIO_OE`	26	Yes	Yes
`GPIO_OUT`	26	Yes	Yes
`XTAL_32K_DELAY`	5	Yes	Yes

GPIO contents configuration

As part of the warm boot process, the GPIO contents (inputs, outputs, logic level, and so on) are restored based on specific MTP sets. The following Table 4-2 and Table 4-3 list the relevant MTP fields that APPS can write to and configure default GPIO contents during a warm boot.

Table 4-2 GPIO configuration

GPIO_CFG [25:0]	Description
Bits	
11	0 for CMOS input 1 for Schmitt input
[8:6]	0 = 2 mA fast 2 = 4 mA fast 1 = 8 mA fast 3 = 12 mA fast 4 = 2 mA slow 6 = 4 mA slow 5 = 8 mA slow 9 = 12 mA slow
[5:2]	This is the alternate function selection register for each PAD. Refer to the PIO multiplexing functions table in the *QCC711 Bluetooth Low Energy Data Sheet* (80-WL711-1) for multiplexed functions on each PAD.
[1:0]	Bit0 = Pull enables: 1=enabled, 0=disabled. Bit1 = Pull selects: 1=pull up, 0=pull down.

Table 4-3 GPIO output enable

GPIO_OE [25:0]	Description
Bits	
[25:0]	Each bit controls the output enable of a single PAD. For example, bit0 → PIO_0's OE, bit1 → PIO_1's OE and so on. 1=PIO output, 0=PIO input, 1=High, 0=Low

The following commands are examples of reading the GPIO23 status and configuring the GPIO23 for alternate function 2, enabling GPIO output, enabling GPIO pull up, and GPIO driving level 2 mA fast.

```
>python2  .\otp_programmer.py --read GPIO_CFG23 GPIO_OE GPIO_OUT
>python2  .\otp_programmer.py --write GPIO_CFG23=0x00B
>python2  .\otp_programmer.py --write GPIO_OE=0x0800000
>python2  .\otp_programmer.py --write GPIO_OUT=0x0800000
```

BD address

Bluetooth devices are identified with a Bluetooth device address (`BD_ADDR`). The following commands can be used for reading the `BD_ADDR` from the MTP and writing it to the MTP:

`>python2 .\otp_programmer.py --read BD_ADDR` (for reading the BD address), and

`>python2 .\otp_programmer.py --write BD_ADDR=0x0202DEAD0001` (for writing the BD address).

5 Installing an update package for QCC711

This section presents the instructions for installing update packages. The OEM must install their update package to each QCC711 device on the production line.

Once the update package has been generated (refer to section on preparing for enrollment), install the package according to the following instructions:

1. Copy the generated `update_btss_btcfg.bin` file to the SDK folder`\\qcc711_sdk\tools\scripts`.
2. Install the update package with the following command:

   ```
   >python2 nvm_programmer.py -b 0x10248000 -U update_btss_btcfg.bin
   ```

Figure 5-1 Successful installation of an update package

If the image can be seen successfully programmed like in the example of Figure 5-1, then the signed image has been installed and is working accordingly.

6 Programming an APPS image update for QCC711

This section covers the two ways for programming the APPS image update by using the NVM programmer tool with or without its security feature enabled. If the OEM has a concern about the confidentiality of their device image programming, using the NVM programming tool with the security feature enabled is then recommended.

The following command illustrates how to generate an update APPS image:

```
>python3 ./sign_qcc710.py -t update -c .\input\OEM-REC-QCC710.json -b .\config
\OEM-IAR-BMHT.json --privkey .\key\p521-private-key.p8 --pubkey .\key\p521-
public-key.spki -Q .\input\qti_manifest-app_mode-unlicensed.bin -A .\input
\appsGeneral.bin -O .\input\APPS_OEM_Manifest_license.bin -o .\output
\update_apps.bin
```

APPS image update with NVM programming

Use the NVM Programmer tool to program an APPS image with the following steps:

1. Copy `apps.bin` into the SDK folder`\\qcc711_sdk\tools\scripts`. Verify the binary as an update package, and verify all images as if an APPS debug is about to be locked:

   ```
   >python2 .\nvm_programmer.py -b 0x10248000 -V update_apps.bin
   ```

2. Execute the following command to program the update APPS image:

   ```
   >python2 .\nvm_programmer.py -b 0x10248000 -U update_apps.bin
   ```

3. Navigate to SDK folder`\\qcc711_sdk\tools\scripts` and execute the following command to enable OEM authentication:

   ```
   >python2 .\otp_programmer.py --write OEM_SECURITY_POLICY=0x80
   ```

4. Execute the following command to lockout APPS debugging. This means that the OEM will not be able to use the NVM Programmer for updates, other tools, and debugging in general, so exercise caution with this step.

   ```
   >python2 .\nvm_programmer.py -b 0x10248000 -U bin/update_apps.bin
   ```

APPS image update with security feature enabled NVM programming

Program encrypted APPS images with the following steps if confidentiality is a concern.

1. Refer to the *QCC711 Bluetooth Low Energy Software Programming Guide* (80-61032-1) on how to generate a `key_file.json` file. The following example displays the contents of an example `key_file.json` file:

```
{
 "KeyType": "Derived",
 "Nonce": "256 random bits chosen by the OEM",
 "OemBatchSecret": "product from section Creation of
OEM_BATCH_SECRET_HASH",
 "Label": "128 random-bit sequence chosen by the OEM",
 "DebugState": "ENABLE",
 "SecureBootState": "False",
 "Algorithm": "AES128_GCM",
 "OemId": "provided by QTIL; refer to the section on Preparations for QCC711
enrollment",
 "OemRcHash": "picks up OEM_MRC_HASH from 'sign' row; refer to section
Creation of OEM_MRC_HASH",
 "OemLcs": "OEM_LCS_PRODUCTION",
 "IsOemLcsActivated": "False"
}
```

2. Navigate to the SDK folder`\\qcc711_sdk\tools\scripts` and run the following command:

```
>python3 .\encrypt_file.py update_apps.bin encrypted_update_apps.bin
key_file.json
```

3. Copy the `secure_loader.bin` file from SDK folder`\\qcc711_sdk\tools\bin` to SDK folder `\\rot\tools\QCC710-Signing\input` and execute the following command:

```
>python3 ./sign_qcc710.py -t manifest -c .\input\OEM-REC-QCC710.json -b
.\config\OEM-IAR-BMHT.json --privkey .\key\p521-private-key.p8 --pubkey
.\key\p521-public-key.spki -A .\input\secure_loader.bin -C .\input
\btcfg_app_mode.bin -o
.\ouput\OEM_Manifest_Secure_Loader.bin
```

4. Copy the `OEM_Manifest_Secure_Loader.bin` file from SDK folder`\\rot\tools\QCC710-Signing\output` to SDK folder`\\rot\tools\QCC710-Signing\input` and execute the following command:

```
>python3 ./sign_qcc710.py -t update -c .\input\OEM-REC-QCC710.json -b
.\config\OEM-IAR-BMHT.json --privkey .\key\p521-private-key.p8 --pubkey
.\key\p521-public-key.spki -A .\input\secure_loader.bin -O
.\input\OEM_Manifest_Secure_Loader.bin -o update_secure_loader.bin
```

5. The encrypted update package can now be programmed using the `Secure_Programmer.py`. Copy the `update_secure_loader.bin` and `encrypted_update_apps.bin` files into the SDK folder`\\qcc711_sdk\tools\scripts` and execute the following command:

   ```
   >python2 .\secure_programmer.py update_secure_loader.bin
   encrypted_update_apps.bin com58
   ```

   NOTE Find the corresponding UART communications port from the device manager. In the example before, this port is 'com58'.

   NOTE Execute the following commands to install the required pyserial and pywin32 packages:

   ```
   python -m pip install pyserial
   python -m pip install pywin32
   ```

6. If locking out APPS debugging, go back to step 1 and change the contents of the `key_file.json` as follows:

   ```
   "DebugState": "DISABLE"
   "SecureBootState": "True"
   ```

7. Then repeat steps 2 to 4 and execute the following command:

   ```
   >python2 .\secure_programmer.py update_secure_loader.bin
   encrypted_update_apps.bin com58 -L
   ```

7 RF calibration and hardware test for QCC711

This stage of the flow process focuses on QCC711 RF calibration and validation as well as hardware test at the factory. This stage requires specific equipment, for example, among others, a spectrum analyzer, a frequency counter, and a shield box. Additionally, a customized test jig may be needed to test the OEM peripheral interfaces such as SPI, I²C, UART, and LEDs.

QTIL provides the MFG demo as well as the peripheral demo, which allows an OEM to excuse RF calibration, RF validation and peripheral interface validation. Refer to the software package for more details. This document covers the RF calibration only to the extent of ensuring that specific important parameters have been written correctly.

RF calibration

The RF calibration is for crystal trimming only. The QCC711 has two crystal oscillators (32.768 kHz and 32 MHz). The QCC711 32.768 kHz crystal oscillator is used for accurate wake-up as well as the Bluetooth® Low Energy sleep clock. The 32 MHz crystal oscillator is used for RF activity, the UART baud clock, and for running applications when Bluetooth Low Energy is not awake.

The QCC711 contains an array of internal capacitors that can be attached to the XTAL_IN and XTAL_OUT nodes. These can be switched to pull the crystal to frequency and compensate for initial frequency errors by using a simple per-device trim on the production line. Table 7-1 presents the recommended values for relevant MTP fields for 32 MHz crystal oscillators.

Table 7-1 Relevant MTP fields for 32 MHz

Relevant fields for the 32 MHz crystal in MTP	Recommended value	Description
`AUX_OEM_DRV_T1`	0x0B54	The time for ignoring CLK_DET at the beginning of a crystal start-up.
`AUX_OEM_DRV_T3`	0x0B54	A settling period after the SEL_CL has been enabled.
`AUX_OEM_DRV_T4`	0x0B54	A settling period after the crystal oscillator current is reduced from the maximum value to a predefined value.
`AUX_OEM_CTRL3`	Unique per board, contains the coarse and fine trim values	The load capacitance (CL) uniquely set per board to bring the crystal to frequency.

For the trim, the 32.768 kHz clock output is routed from GPIO13. Only coarse trim is available for 32.768 kHz, its fine trim value being ‘0’.

The 32 MHz crystal can be trimmed in two ways: the 32 MHz clock output can be routed from GPIO11, or the 32 MHz crystal can be trimmed by monitoring the carrier (CW) frequency offset from the RF port. Refer to the *QCC71x Crystal Trimming User Guide* (80-18078-1) for details.

8 Debug locking/unlocking for QCC711 and preparation for RMA

For security reasons, the QCC711 is designed with the ability for an OEM to prevent debugger attachment. This section demonstrates how an OEM can lock or unlock the debug attachment, and therefore proceed with an RMA for themselves, or for Qualcomm.

Debug locking

To lock the debug, set bit 3, 4, 6, 9 in the `DEBUG_DISABLE_VECTOR OTP` field using:

```
>python2 otp_programmer.py -w DEBUG_DISABLE_VECTOR=0x0258
```

After the debug is locked and the SoC reset, a debugger cannot be used for APSS execution debugging, or NVM inspection and updating.

Debug unlocking for OEM RMA

For OEM RMA investigations that require a debug unlock, create a debug option in a console, or use an OTA command to open a communication session with the RoT. A RoT session must be started before the Debug Unlock Tool can successfully run on a board. Use the Debug Unlock Tool to communicate over an SWD with the RoT, and provide a debug unlock certificate. For the details, refer to the *QCC711 Bluetooth Low Energy Software Programming Guide* (80-61032-1).

Debug unlocking for Qualcomm RMA

If the QCC711 is to be dispatched to Qualcomm (QTIL) for further failure analysis, it is recommended to use a debug option, or an OTA command that can erase any sensitive data on the device. Start a RoT communication session after each APSS restart without closing it.

Document references

Document	Reference, date
QCC711 Bluetooth Low Energy Data Sheet	80-WL711-1
QCC711 Bluetooth Low Energy Software Programming Guide	80-61032-1
QCC71x Crystal Trimming User Guide	80-18078-1
Typical Solder Reflow Profile for Lead-Free Devices Application Note	80-CT462-1

LEGAL INFORMATION

Your access to and use of this material, along with any documents, software, specifications, reference board files, drawings, diagnostics and other information contained herein (collectively this “Material”), is subject to your (including the corporation or other legal entity you represent, collectively “You” or “Your”) acceptance of the terms and conditions (“Terms of Use”) set forth below. If You do not agree to these Terms of Use, you may not use this Material and shall immediately destroy any copy thereof.

1) Legal Notice.

This Material is being made available to You solely for Your internal use with those products and service offerings of Qualcomm Technologies, Inc. (“**Qualcomm Technologies**”), its affiliates and/or licensors described in this Material, and shall not be used for any other purposes. If this Material is marked as “**Qualcomm Internal Use Only**”, no license is granted to You herein, and You must immediately (a) destroy or return this Material to Qualcomm Technologies, and (b) report Your receipt of this Material to qualcomm.support@qti.qualcomm.com. This Material may not be altered, edited, or modified in any way without Qualcomm Technologies’ prior written approval, nor may it be used for any machine learning or artificial intelligence development purpose which results, whether directly or indirectly, in the creation or development of an automated device, program, tool, algorithm, process, methodology, product and/or other output. Unauthorized use or disclosure of this Material or the information contained herein is strictly prohibited, and You agree to indemnify Qualcomm Technologies, its affiliates and licensors for any damages or losses suffered by Qualcomm Technologies, its affiliates and/or licensors for any such unauthorized uses or disclosures of this Material, in whole or part.

Qualcomm Technologies, its affiliates and/or licensors retain all rights and ownership in and to this Material. No license to any trademark, patent, copyright, mask work protection right or any other intellectual property right is either granted or implied by this Material or any information disclosed herein, including, but not limited to, any license to make, use, import or sell any product, service or technology offering embodying any of the information in this Material.

THIS MATERIAL IS BEING PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESSED, IMPLIED, STATUTORY OR OTHERWISE. TO THE MAXIMUM EXTENT PERMITTED BY LAW, QUALCOMM TECHNOLOGIES, ITS AFFILIATES AND/OR LICENSORS SPECIFICALLY DISCLAIM ALL WARRANTIES OF TITLE, MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, COMPLETENESS OR ACCURACY, AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MOREOVER, NEITHER QUALCOMM TECHNOLOGIES, NOR ANY OF ITS AFFILIATES AND/OR LICENSORS, SHALL BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY EXPENSES, LOSSES, USE, OR ACTIONS HOWSOEVER INCURRED OR UNDERTAKEN BY YOU IN RELIANCE ON THIS MATERIAL.

Certain product kits, tools and other items referenced in this Material may require You to accept additional terms and conditions before accessing or using those items.

Technical data specified in this Material may be subject to U.S. and other applicable export control laws. Transmission contrary to U.S. and any other applicable law is strictly prohibited.

Nothing in this Material is an offer to sell any of the components or devices referenced herein.

This Material is subject to change without further notification.

In the event of a conflict between these Terms of Use and the *Website Terms of Use* on www.qualcomm.com, the *Qualcomm Privacy Policy* referenced on www.qualcomm.com, or other legal statements or notices found on prior pages of the Material, these Terms of Use will control. In the event of a conflict between these Terms of Use and any other agreement (written or click-through, including, without limitation any non-disclosure agreement) executed by You and Qualcomm Technologies or a Qualcomm Technologies affiliate and/or licensor with respect to Your access to and use of this Material, the other agreement will control.

These Terms of Use shall be governed by and construed and enforced in accordance with the laws of the State of California, excluding the U.N. Convention on International Sale of Goods, without regard to conflict of laws principles. Any dispute, claim or controversy arising out of or relating to these Terms of Use, or the breach or validity hereof, shall be adjudicated only by a court of competent jurisdiction in the county of San Diego, State of California, and You hereby consent to the personal jurisdiction of such courts for that purpose.

2) Trademark and Product Attribution Statements.

Qualcomm is a trademark or registered trademark of Qualcomm Incorporated. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the U.S. and/or elsewhere. The Bluetooth® word mark is a registered trademark owned by Bluetooth SIG, Inc. Other product and brand names referenced in this Material may be trademarks or registered trademarks of their respective owners.

Snapdragon and Qualcomm branded products referenced in this Material are products of Qualcomm Technologies, Inc. and/or its subsidiaries. Qualcomm patented technologies are licensed by Qualcomm Incorporated.