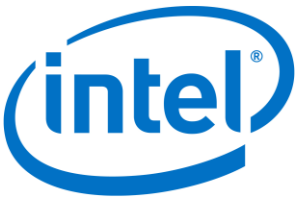


Intel® Curie™ Module

Software User Guide

December 2015

Revision 003

Intel Confidential

Document Number: 333240-003

Contents

Figures

Tables

Revision History

Revision	Description	Date
001	Initial public release.	October 8, 2015
002	Added information on flashing and debugging the board.	October 23, 2015
003	Intel Software for Curie Platform v2 Release	December 2015

§

1 Introduction

This document explains the software aspects of getting started using an Intel® Curie™ module. It addresses the host development environment, source code build and flashing procedure, debug methodologies, and some aspects of driver and service API customization.

1.1 Terminology

Table 1 Terminology

Term	Definition
API	Application program interface
ARC	Argonaut RISC core
BAS	Battery Service
BLE	Bluetooth* Low Energy
BT	Bluetooth*
CPU	Central processing unit
CRB	Customer reference board
DIS	Device Information Service
GNU	GNU's not Unix
GPIO	General purpose input output
GPS	Global positioning system
IO	Input/output
JTAG	Joint Test Action Group
LTS	Long-term support
NFC	Near Field Communications
OHRM	Optical Heart Rate Monitor
SDK	Software development kit
SoC	System on Chip
SPI	Serial peripheral interface
UART	Universal asynchronous receiver/transmitter
USB	Universal serial bus
UUID	Universally unique identifier

§

2 One-time Setup

A Linux* OS-based host PC is required to build the Intel® Curie™ module's source code. Linux* Ubuntu 14.04 LTS is currently the only distribution that has been tested and is supported. Contact your Intel business representative for access to the secure FTP server from which to download the source code and the required tools to build and flash.

Linux* PC setup

Install the required packages:

```
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib build-
essential chrpath libsdl1.2-dev xterm libqtgui4:i386 libtool libc6:i386

$ sudo apt-get install gdebi-core gksu libgksu2-0 fping gdebi p7zip-full
```

Tool setup

Install Intel® Phone Flash Tool Lite:

```
$ sudo dpkg -i phoneflashtoollite_5.3.4.1_linux_x86_64.deb
```

Install the Intel maker/wearable device tool

```
$ sudo dpkg -i Intel_Maker_and_Wearable_Device_Tools_1.0.1.0_linux_x86_64.deb
```

Build prerequisites

If dash is configured as the default shell, remove it with the following command and answer *No* at the confirmation question:

```
$ dpkg-reconfigure dash
```

§

3 Build Intel® Quark™ SE Code

Unpack the source tarball on your Linux* machine:

```
$ tar -xzf intel_iq_sdk-1.0.0-weekly-20.tgz
$ cd intel_iq_sdk
```

Assuming all of the host prerequisites have been fulfilled as described in the previous section, the process of creating a firmware image includes the following steps:

1. Setup
2. Image creation
3. Custom configuration (optional)

Note: All the commands below to configure, create, and flash an Intel® Curie™ module-based project must be issued from the project directory using GNU make. All Intel® Curie™ module-based projects can be found in the ***<intel_iq_sdk>/ wearable_device_sw/projects/*** folder.

3.1 Setup

The SDK build system allows a single project to produce firmware images differentiated along three dimensions:

- The PROJECT identifies the features.
- The BOARD identifies the hardware.
- The BUILDVARIANT is either debug or release.

```
$ cd /path/to/project
# For example;
$ cd wearable_device_sw/projects/curie_reference/
$ sudo make one_time_setup  /*This command checks for all the prerequisites  of
                           *the host and needs to be run only once */

        /* make setup BOARD=<xxx> BUILDVARIANT=<yyy> */
$ make setup
```

Where the environment variables take the values as explained below, if nothing is entered after 'make setup' command, a set of default values for "BOARD=crb" and "BUILDVARIANT=release" are assigned.

1. xxx i.e., BOARD – crb (Intel® Curie™ Reference Board)
2. yyy i.e., BUILDVARIANT – release or debug

For more information, type:

```
$ make help
```

Or optionally view the project *Makefile* which can provide many useful details.

The setup command involves SDK setup and project setup. The SDK setup will create the build output directory and prepare the environment:

- Compile host tools.
- Save setup parameters.

Several build environments can coexist under a project tree, all located under *<intel_iq_sdk>*. The naming convention for a specific build output directory is:

```
./out/PROJECT_BOARD_BUILDVARIANT
```

The current environment (the last one you did a setup for) is identified using a symbolic link.

Example:

```
├── curie_reference_crb_release
├── current ->
/home/user/intel_iq_sdk/wearable_device_sw/../out/curie_reference_crb_release
└── host_tools
```

3.2 Image creation

Use the *make image* command to create firmware images using the image target:

```
$ make image
```

The resulting binaries are created in *<intel_iq_sdk>/out/current/firmware*. This build target is currently entirely project-specific, requiring in particular the project *Makefile* to directly call the *Makefile.build* and *Makefile* commands to generate the SDK library.

§

4 Flash the Board

To flash an Intel® Curie™ board, do the following:

1. Connect the Intel® Curie™ module and debug board with the two 10-pin flat ribbon cables (Figure 1).
2. Set the main board and the serial debug board side by side so that the USB connectors align and point in the same direction as shown.
3. Connect the LI-polymer battery.
4. Connect the 10-pin ribbon cable between the JTAG ports.
5. Connect the 10-pin ribbon cable between the UART ports.
6. Connect a micro USB cable to the SoC USB port on the module's main board and the host Linux* PC which is used as power supply to the CRB.

   Note: A USB/DFU micro USB cable connected to the SoC's USB port also serves the purpose of flashing from a host Linux* PC.
7. Connect a second micro USB cable for serial debug output to the USB port on the debug board and the host Linux PC.

Note: Start *minicom* or *putty* or any other terminal emulation application with the settings as 115200-8-N-1.

Figure 1 Board connections (main board and debug board)

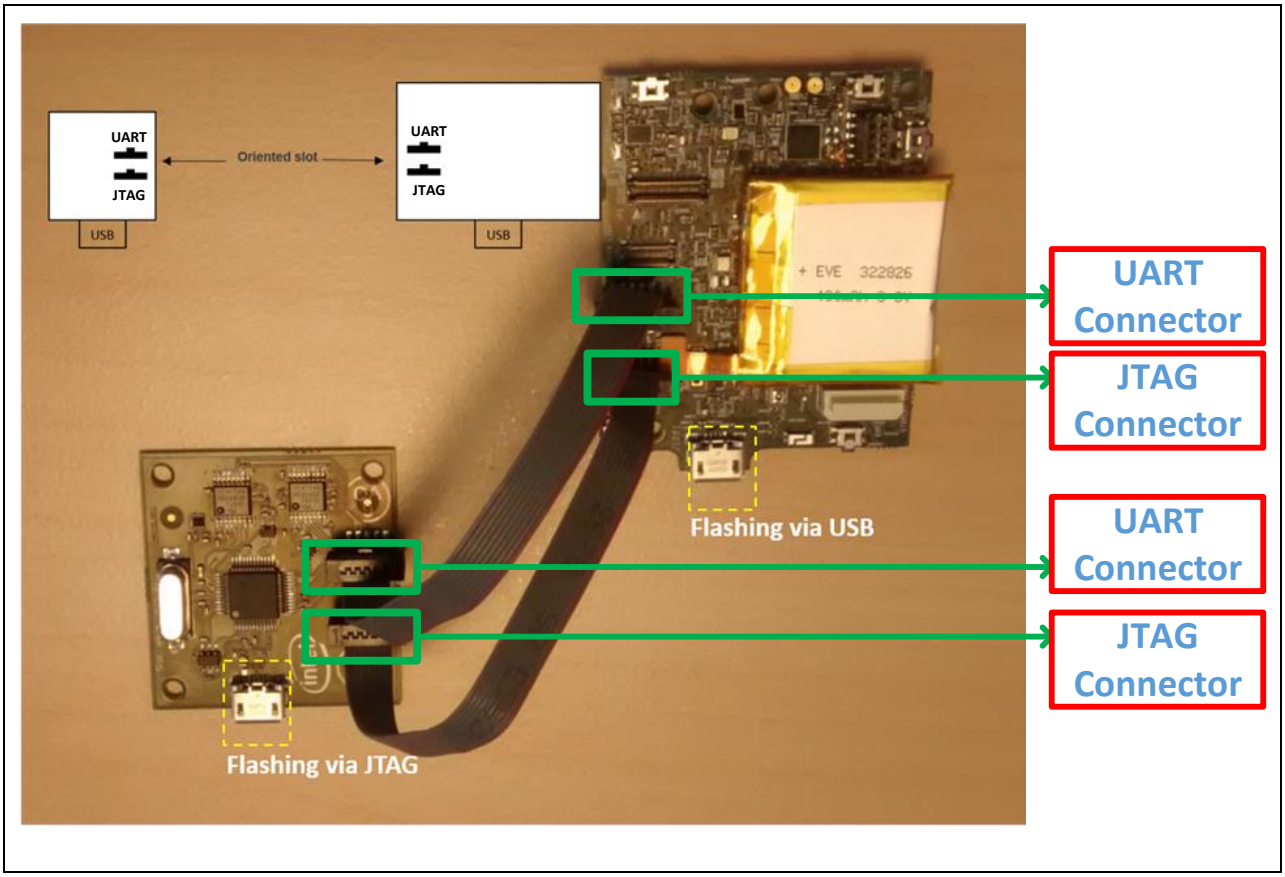
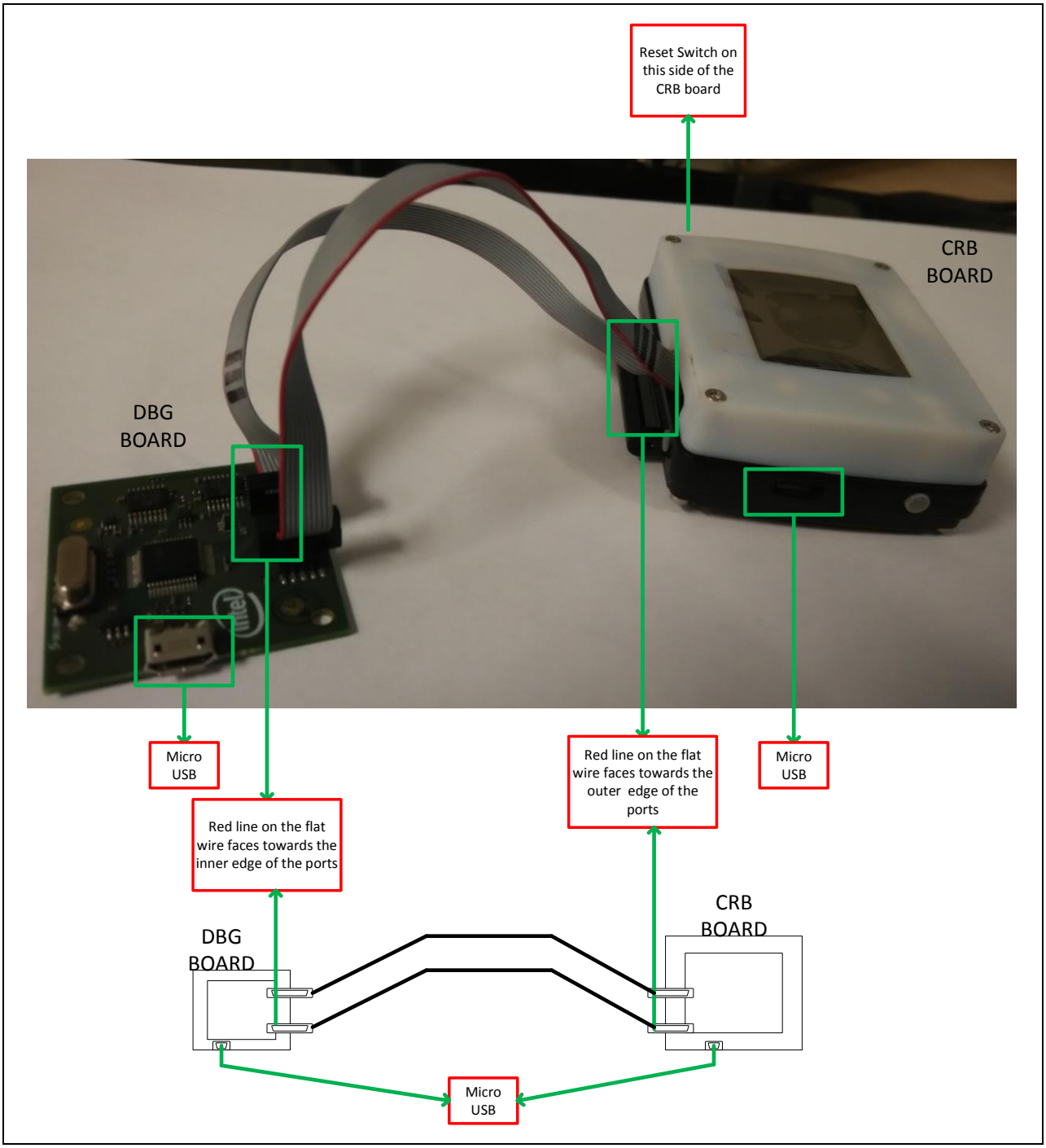


Figure 2 Board connections (main board with display unit and debug board)



4.1 Flash procedure from command prompt

Start the flash procedure from the command line with the following command:

```
$ make flash
```

Figure 3 Reference board front side, showing connectors

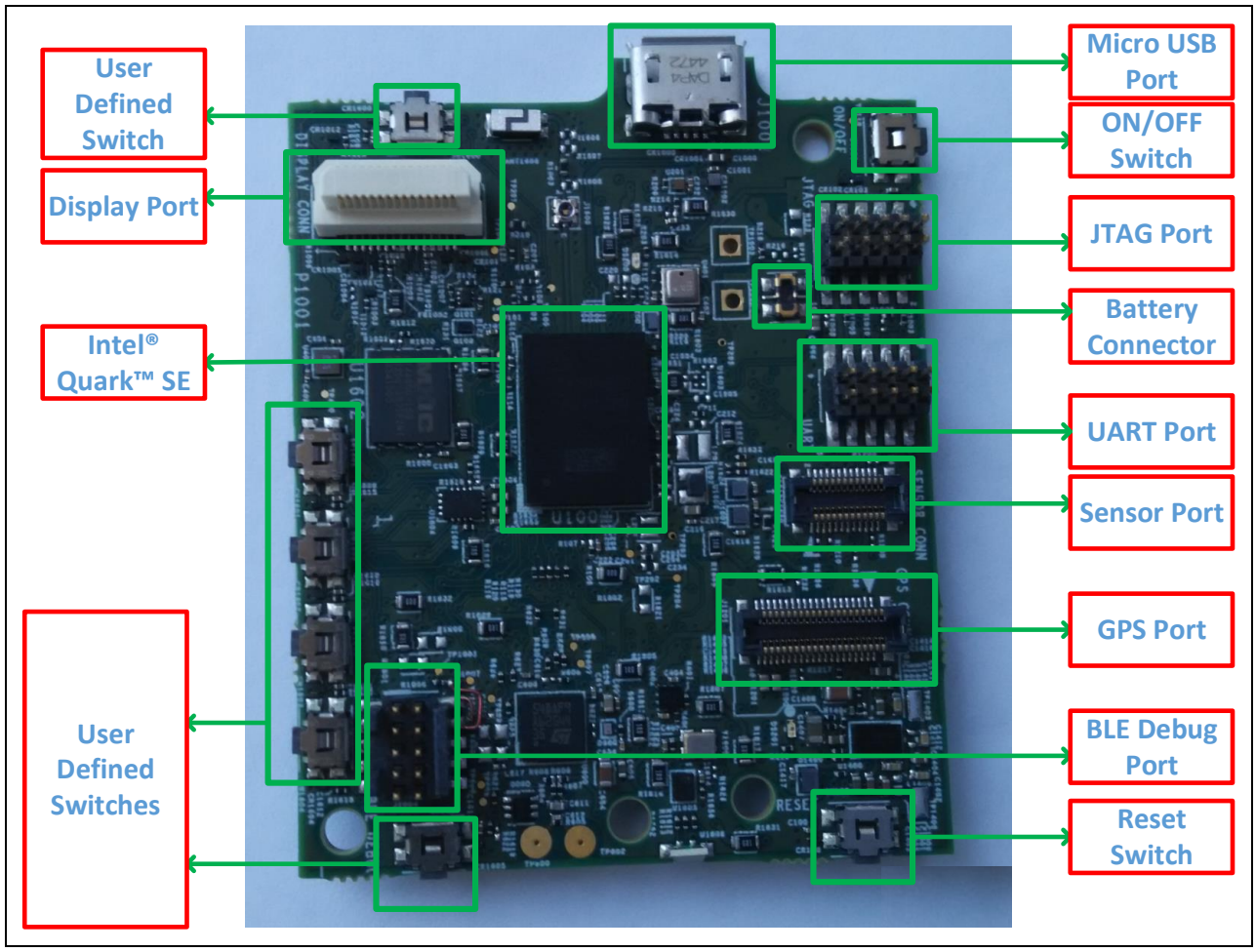


Table 2 CRB front items - description

Item	Description
Micro USB port	Micro USB connection is used for firmware flashing and charging.
On/off switch	Shuts down the board.
Reset switch	Implements reset functionality.
Battery connector	Port to connect external battery.

4.2 Flash the images with Intel® Phone Flash Tool Lite

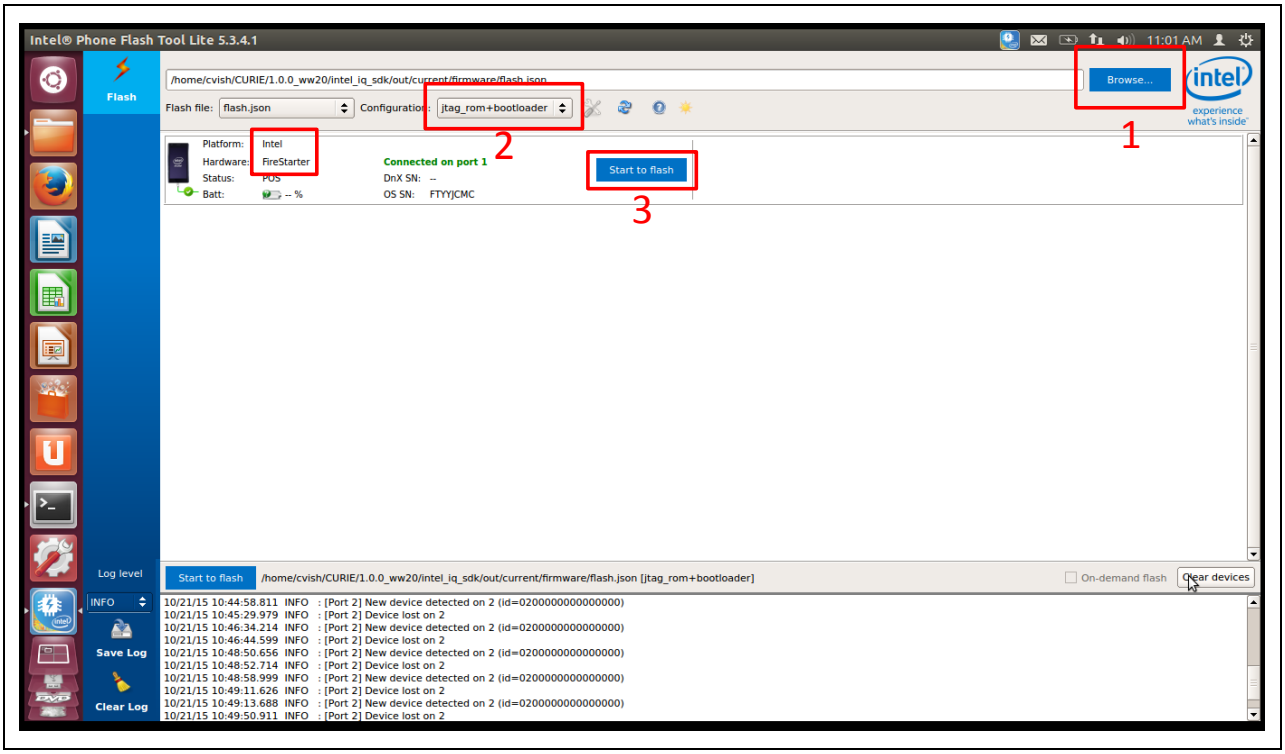
You may also flash the board using Intel® Phone Flash Tool Lite.

1. Launch Intel® Phone Flash Tool Lite (installed in section 2).
2. Click *Browse* in the upper right and select the *flash.json* file to upload (#1 in Figure 4).
3. Select *jtag_rom+bootloader* under the *Configuration* dropdown menu (#2 in Figure 4).

4. Click *Start to Flash* next to *Firestarter* (#3 in Figure 4).

   Note: The *flash.json* file from the current build is located in the *<intel_iq_sdk>/out/current/firmware* folder.

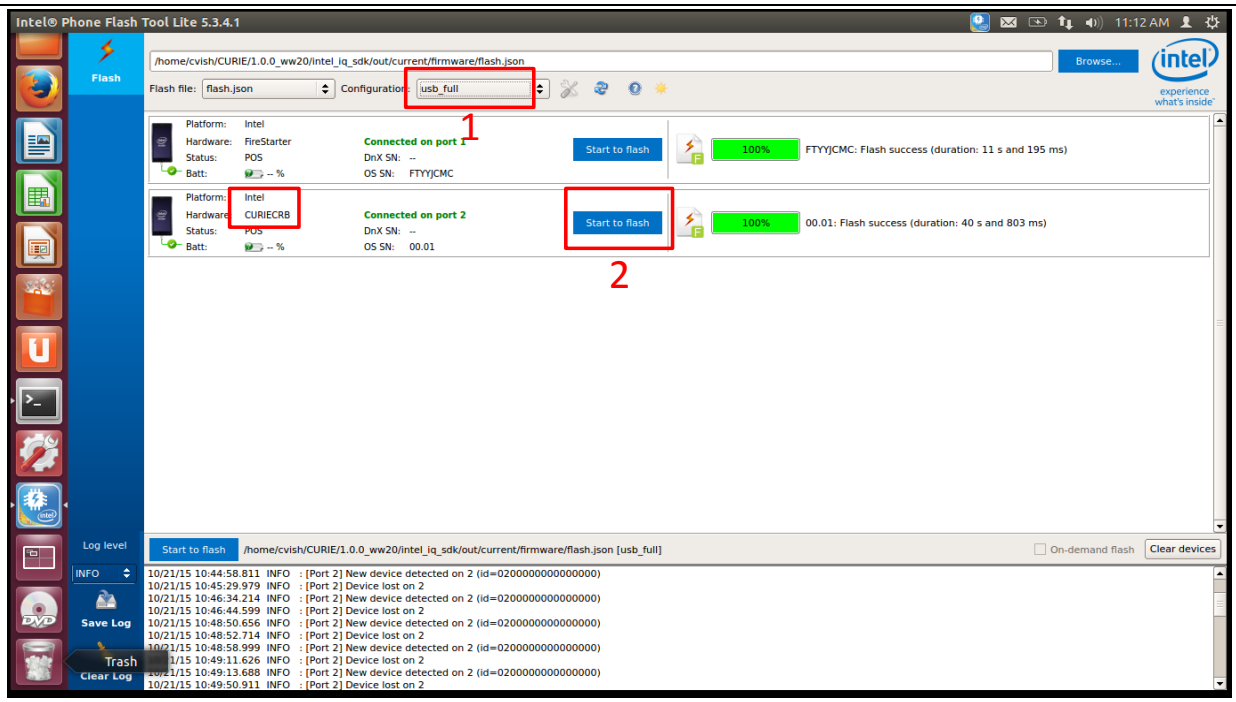
Figure 4 Intel® Phone Flash Tool > Firestarter



5. Once *Hardware: Firestarter* is flashed, *Hardware: CURIECRB* will appear in Intel® Phone Flash Tool Lite (Figure 5).
6. After flashing on *FireStarter* is successful, select *usb_full* under the *Configuration* dropdown menu.
7. Click *Start to Flash* next to *CURIECRB* (#1 and #2 in Figure 5).

   Caution: You must flash with *usb_full* in less than 10 seconds after the first flash. If you're not sure about timing, you can push the reset button and flash with *usb_full* (within 10 seconds after the reset).

Figure 5 Intel® Phone Flash Tool > CRB



§

5 Customize the Source Code

5.1 Create a new Intel® Curie module-based project

The Intel® Curie™ module's SDK already implements most generic build targets required to create Intel® Curie™ module-based products. As a consequence, the process of creating a project is greatly simplified: The only requirement to create a minimal Makefile for an Intel® Curie™ module-based project is to include the Intel® Curie™ module's common targets sub-makefile and to define a few variables.

The *projects/curie_sdk/curie_common_targets.mk* file contains all minimal targets required to build a Intel® Curie™ module-based product:

- bootloader
- bootupdater
- charging OS
- main core image
- sensor core image
- Bluetooth core image.

It shall be included in the project Makefile, after having defined the following variables:

- PROJECT, BOARD, BUILDVARIANT, as defined in the generic build process.
- QUARK_DEFCONFIG, ARC_DEFCONFIG, BLE_CORE_DEFCONFIG, which should point to valid build configurations for the main, sensor, and Bluetooth cores.

```
PROJECT      = curie_reference
BOARD        ?= crb
BUILDVARIANT ?= debug

QUARK_DEFCONFIG ?= $(PROJECT_PATH)/quark/defconfig
ARC_DEFCONFIG ?= $(PROJECT_PATH)/arc/defconfig
BLE_CORE_DEFCONFIG ?= $(PROJECT_PATH)/ble_core/defconfig

include ../curie_sdk/build/curie_common_targets.mk
```

The project-specific sources are not listed in the project Makefile, but should instead be organized according to a generic project source tree; the root folder *must* be the directory that contains the project Makefile. The Intel® Curie™ module's SDK already includes several generic boards, drivers, and services, but your project can add custom items using optional Kconfig extension files. Example of project source tree:

```
├── Kbuild.mk
├── project.Kconfig
├── services.Kconfig
├── drivers.Kconfig
├── arc
│   ├── Kbuild.mk
│   ├── arc_defconfig_ctb
│   ├── arc_defconfig_crb
│   └── main.c
├── quark
│   ├── Kbuild.mk
│   ├── quark_defconfig_ctb
│   ├── quark_defconfig_crb
│   ├── ble_app.c
│   ├── ble_app.h
│   └── main.c
├── ble_core
│   └── ble_core_defconfig
```

```
└── Makefile
```

5.1.1 Project source tree

The Intel® Curie™ module's SDK uses a recursive build system inspired by the Linux* kernel"s Kbuild. The SDK expects to find the project source tree at a location pointed at by the PROJECT_PATH environment variable (defined in the Makefile).

This directory contains:

- The optional project root Kbuild.mk.
- The optional project specific configuration files (see section 5.2).

Kconfig files are required only if you want to add custom configuration rules. *Kbuild.mk* is required only if you want to have some of your projects files built with the SDK build system. All project-specific sources that need to be compiled must be available from the root *Kbuild.mk* file, possibly conditioned by the configuration flags you have defined.

5.2 Defining a custom build configuration (basic step-by-step)

The Intel® Curie™ module's SDK exposes a set of features to be selected based on the actual hardware capabilities of each platform. The consistency of build configurations is guaranteed by rules written in the *Kconfig* language.

Note: For details about *Kconfig*, visit: *https://www.kernel.org/doc/Documentation/kbuild/kconfig-language.txt*.

5.2.1 Steps to custom build configuration

Step 1:

```
$ make setup
```

Step 2: Execute either of the below menuconfig commands depending upon the usage. The below commands will launch a menu-driven text-based configurator. You can traverse into different menus of your choice and modify the configuration options.

```
$ make quark_menuconfig      // For Intel® Quark™
```

...or:

```
$ make arc_menuconfig        // For ARC
```

...or:

```
$ make ble_core_menuconfig   // For BLE
```

Custom configurations can be saved and reused by running the following commands. The same commands can be used for ARC (DSP sensor hub) and ble_core by replacing 'quark' with 'arc' and 'ble_core' respectively.

```
$ make quark_savedefconfig DEFCONFIG=/path/to/defconfig
```

...and:

```
$ make quark_defconfig DEFCONFIG=/path/to/defconfig
```

Step 3:

```
$ make image
```

5.3 Configuration patterns

- Features
- Services
- Drivers
- Boards

5.3.1 Features

Features represent hardware capabilities of the target, like USB support, the ability to control SPI peripherals, an embedded flash, etc. Features are described in build configurations using two flags:

- HAS_XX flag to indicate that a feature is available.
- XX flag to indicate that a feature is selected.

Example:

```
HAS_USB, USB
HAS_FLASH, FLASH
```

Generic features can be split into more specific features that implement them to offer more flexibility.

Example:

```
config HAS_SPI_FLASH
    select HAS_FLASH
```

5.3.2 Services

Services represent high-level functional capabilities of the target, like battery status or sensors. Services can typically be distributed on multiple cores. Services are usually described in build configurations using two flags:

- SERVICES_XX selects the service API
- SERVICES_XX_IMPL selects the service implementation.

Services implementations typically depend on the availability of specific hardware features, and will select them automatically if they are available.

Enable the required service under Services, from quark menu config window - *UIServicesMenuConfig*

Example:

```
config SERVICES_STORAGE_IMPL
    depends on HAS_FLASH
    select SPI_FLASH if HAS_SPI_FLASH
    select SOC_FLASH if HAS_SOC_FLASH
```

5.3.3 Drivers

Drivers represent the piece of software giving access to a specific hardware feature. Drivers are described by flags that correspond to the component for which they provide access. Drivers are typically dependent on the availability of the feature they give access to.

Example:

```
config SPI_FLASH_MX25U12835F
    bool "SPI NOR Flash Macronix MX25U12835F"
    depends on HAS_SPI_FLASH
```

5.3.4 Boards

Boards are meta configuration flags used to simplify build configuration for the same hardware platform. Boards will typically declare hardware features availability and select the corresponding driver when the feature is selected.

Example:

```
config BOARD_CURIE_CRB
    ...
    select HAS_SPI_FLASH if QUARK
    select SPI_FLASH_MX25U12835F if SPI_FLASH
```

According to the set of rules defined in the previous paragraphs as examples, the following base configuration:

```
CONFIG_BOARD_CURIE_CRB=y
CONFIG_SERVICES_STORAGE_IMPL=y
```

...will produce the following expanded configuration:

```
CONFIG_BOARD_CURIE_CRB=y
CONFIG_SERVICES_STORAGE_IMPL=y
CONFIG_HAS_FLASH=y
CONFIG_HAS_SPI_FLASH=y
CONFIG_SPI_FLASH=y
CONFIG_SPI_FLASH_MX25U12835F=y
```

Because:

- the board will advertise the availability of a SPI flash,
- that provides the flash feature,
- allowing the storage service implementation to be selected,
- in turn selecting the SPI flash feature,
- Telling the board to select its SPI flash driver.

5.4 Add a new driver or service

1. All device drivers are located under "*<intel_iq_sdk>/wearable_device_sw/bsp/src/drivers/*" To add your own driver, create a new directory under "*<intel_iq_sdk>/wearable_device_sw/bsp/src/drivers/*" with a name of your choice.
2. Create a Kbuild.mk and Kconfig file and add the corresponding entries.
3. Add the corresponding entries of the newly created directory in "*<intel_iq_sdk>/wearable_device_sw/bsp/src/drivers/Kconfig*" and "*<intel_iq_sdk>/wearable_device_sw/bsp/src/drivers/Kbuild.mk*".
4. *Kconfig* is the file that drives the configuration process and lists the features present in the subdirectory.
5. Configure the source code to add your newly created driver in the image.
6. You can see your driver listed in the menuconfig window. Select your driver and rebuild the source code.

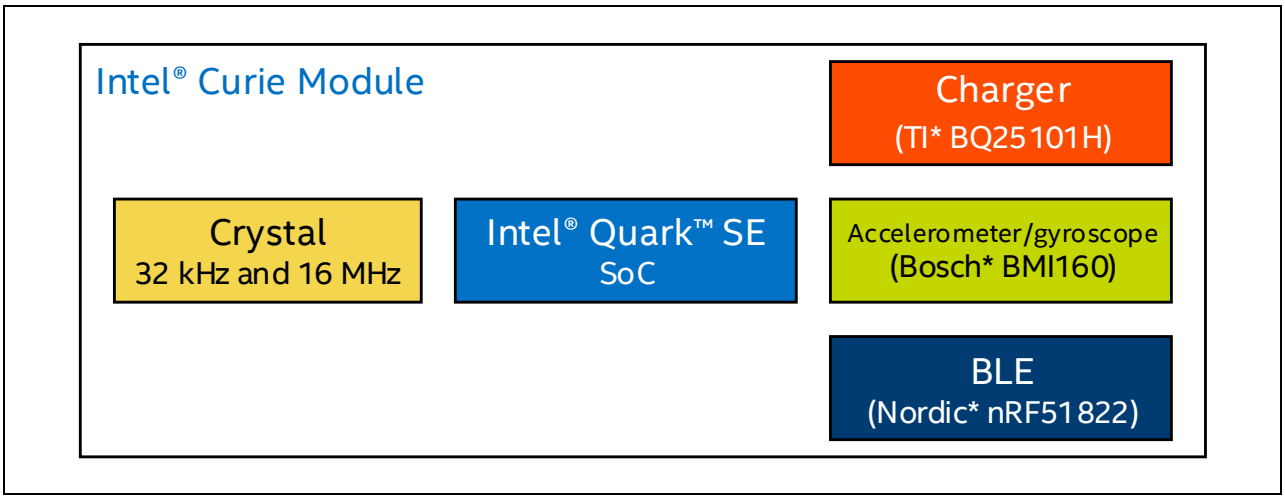
Table 3 Options of interest in Kconfig file

Option	Description
config	This defines the configuration option.
bool	The variable can be set only to y or n.
select <Option>	If you select this variable, the option present in the argument to select will be automatically enabled.
depends on <condition>	The option can only be enabled if the required conditions are met.

6 Testing the Intel® Curie™ module

The Intel® Curie™ module is a hardware platform, designed with an Intel® Quark™ SE SoC, a 6-axis sensor using a Bosch* BMI160, Bluetooth communication using an on-module Nordic* nRF51822, a battery charger built using a Texas Instruments* BQ25101H, and other external devices (sensors, GPS, display, NFC, etc.).

Figure 6 Intel® Curie™ module



6.1 BLE test

The BLE communication on the Intel® Curie™ module is supported by a Nordic* nRF51822. Currently the following BLE services are implemented and supported:

- Simplified security handling
- **BLE DIS service** (UUID: 0x180a): Different device info's (SW version, etc.) are displayed as per BT spec.
- **BLE BAS service** (UUID: 0x180f): BLE Battery level service as per BT spec. It uses the battery service to update automatically the battery level characteristic

How to test BLE services:

1. Install the Curie test app (*agm_test_app-debug.apk*) in your Android device.

```
$ adb install -r agm_test_app-debug.apk
```

Android phone pre-requisites and troubleshooting:

- Android version on the phone should be at least 4.3 (prior versions do not support BLE)
- You must have enabled developer mode on your phone, and activated USB debugging. In most of the phones it can be done using going into: "Settings" ->"Developer Options" ->"USB Debugging"
- The first time you connect your host to your phone, do not forget to tick the box in the confirmation dialog.
- Host prerequisites: You must have adb installed, and if needed enable 32 bits app on 64 bits hosts.
- If you encounter any authorization issues, please make sure that your adb server runs as root:

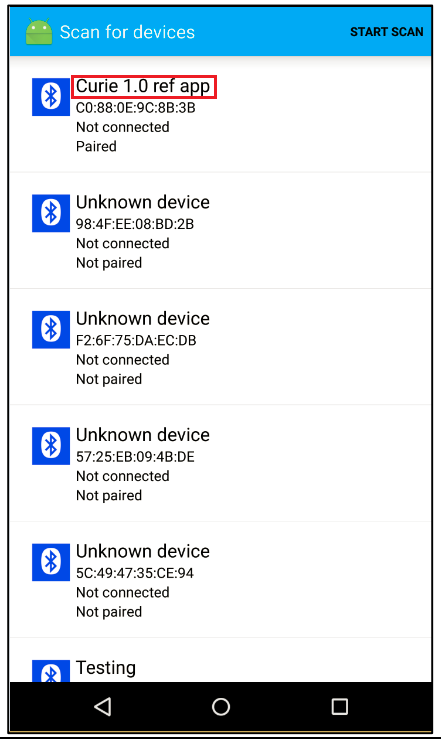
```
$ adb kill-server
$ sudo adb start-server
```

2. Launch the "Curie Test" app and press on "***Find devices"*** to start scanning.

Figure 7 Intel® Curie™ Android* app: Launch

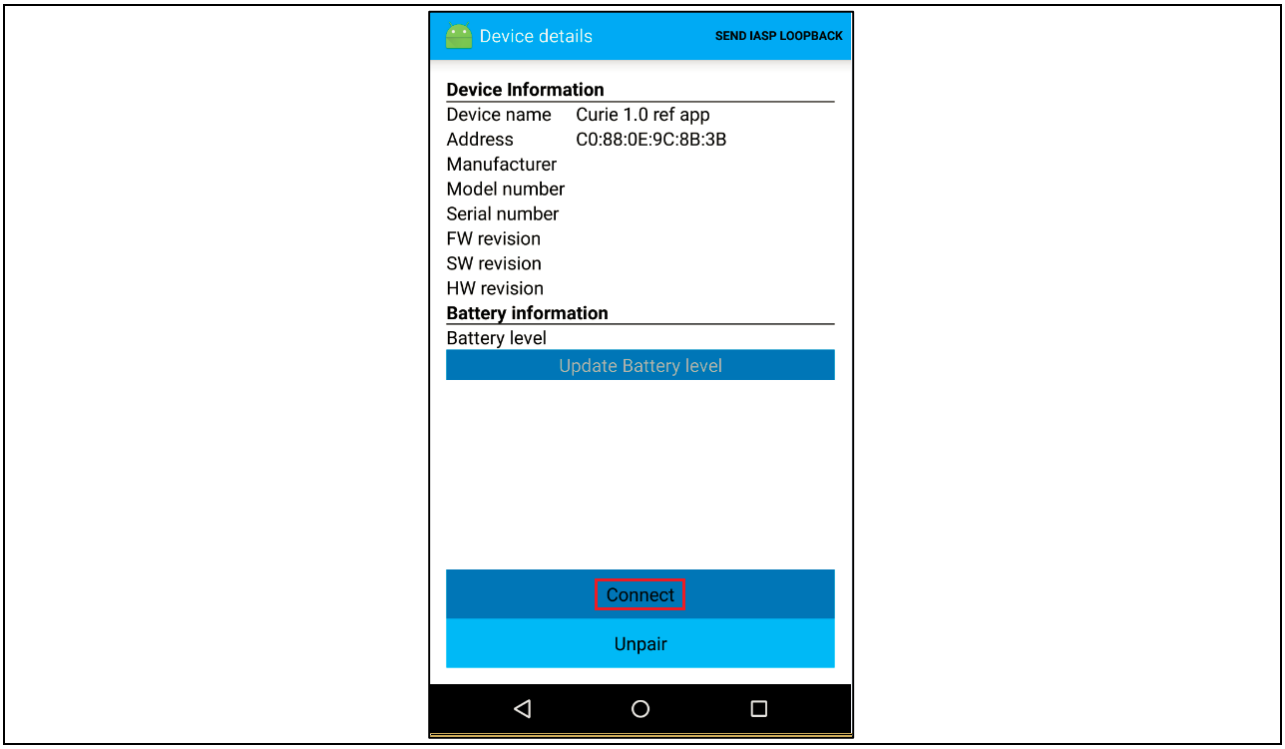
3. Once the scan is complete, you can see your device listed as below

Figure 8 Intel® Curie™ Android* app: Scan completed



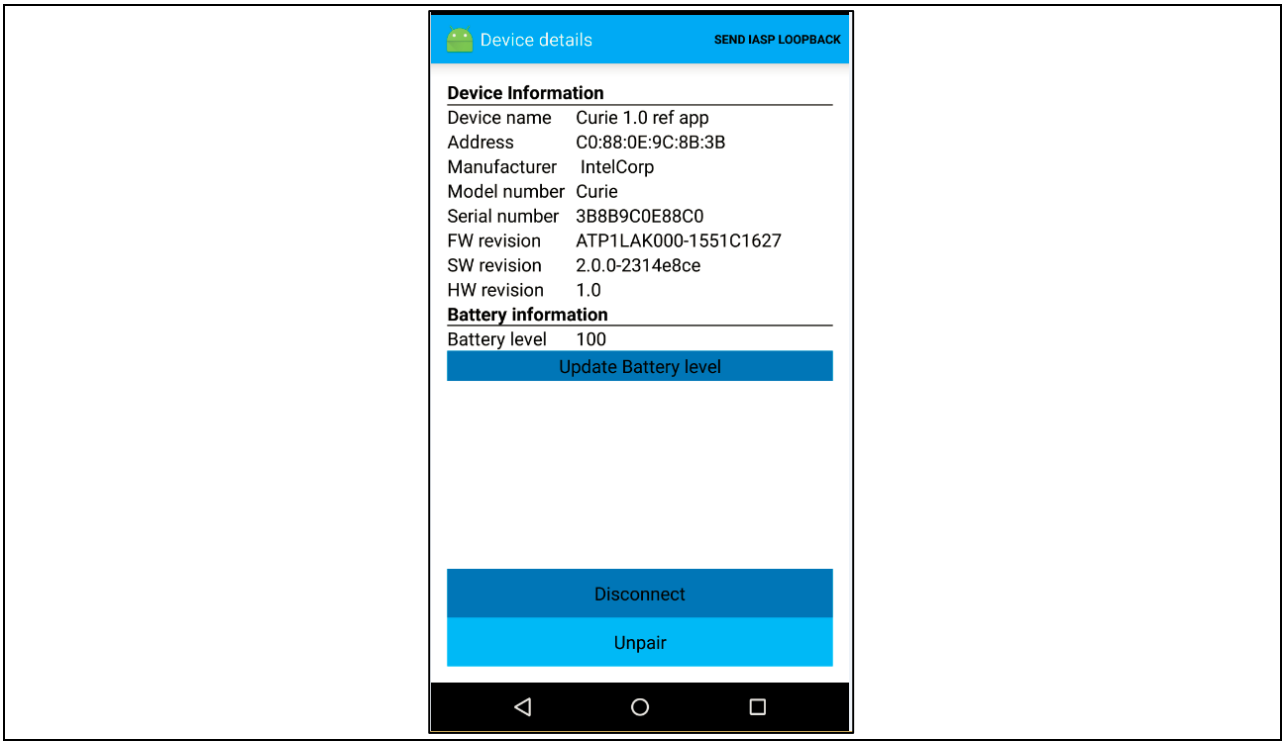
4. Click on your device and then click on "***Connect***"

Figure 12 Intel® Curie™ Android* app: Connect



5. Once the device is connected, you can see the registered services (DIS and BAS) and you can read Device Information and Battery Information.

Figure 13 Intel® Curie™ Android* app: Connect



6.2 Sensor test

The Intel® Curie™ module has an embedded Bosch* BMI160 6-axis accelerometer/gyroscope sensor.

There is already a reference application to test sensors for the *curie_reference* project. The reference application can be found in the *sensing.c (<intel_iq_sdk>/wearable_device_sw/projects/curie_reference/quark/sensing.c)* file. The application subscribes data for gesture, tapping, and step-counter. You can write your own algorithm and process the required data.

The v1 release provides a sample reference application code, which includes the open source sensor code and reports accelerometer's raw data from the embedded BMI160. The file is present in *<intel_iq_sdk>/wearable_device_sw/framework/src/sensors/sensor_core/open_core/algo_support_src/opencore_demo.c* and can be used a starting point to integrate and develop a new algorithm.

How to test sensor services:

1. Build the source code for the *curie_reference* project. Refer to chapter 3 for details on building the sources.
2. Flash the generated images on the CRB. Refer to chapter § for details on flashing.
3. Once the device boots up, the sensors data are subscribed by default.
4. Open a serial log console to print the debug messages:

```
screen -L /dev/ttyUSB1 115200
```

Note: *ttyUSB1* is just an example; it is important to determine the correct port of the device used for serial connection by the host PC.

Wave or move the board to simulate walking/running or perform any gestures. You can record those activities on the serial logs. Figure 9 shows a typical log message from the sample demo app.

Figure 9 Serial communication log messages

```
20499|QRK|    IQ| INFO| steps:6  activity:1
20919|QRK|    IQ| INFO| steps:7  activity:1
21328|QRK|    IQ| INFO| steps:8  activity:1
21749|QRK|    IQ| INFO| steps:9  activity:1
22168|QRK|    IQ| INFO| steps:10  activity:1
22578|QRK|    IQ| INFO| steps:11  activity:1
22999|QRK|    IQ| INFO| steps:12  activity:1
23409|QRK|    IQ| INFO| steps:12  activity:2
23410|QRK|    IQ| INFO| steps:13  activity:2
23829|QRK|    IQ| INFO| steps:14  activity:2
24249|QRK|    IQ| INFO| steps:15  activity:2
24658|QRK|    IQ| INFO| steps:16  activity:2
25909|QRK|    IQ| INFO| steps:17  activity:2
26329|QRK|    IQ| INFO| steps:18  activity:2
27159|QRK|    IQ| INFO| steps:19  activity:2
27159|QRK|    IQ| INFO| steps:19  activity:1
36728|QRK|    IQ| INFO| steps:19  activity:0
48467|QRK|    IQ| INFO| TAPPING=2
52627|QRK|    IQ| INFO| TAPPING=3
53877|QRK|    IQ| INFO| TAPPING=2
55127|QRK|    IQ| INFO| TAPPING=2
```

§

7 Android Firmware Update OTA

Intel® Curie™ software image can be updated using OTA. Intel® Curie™ Firmware update Over the Air (FOTA) is done via BLE.

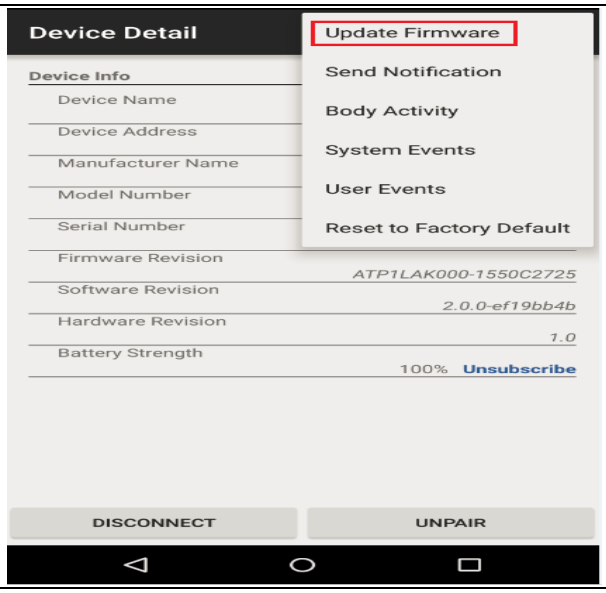
Procedure to update Firmware Image:

1. Install Core-SDK-app.apk on your Android device having the BLE capability.
2. Start the Core-SDK app and click on "***Start Scan***" to scan for BLE devices.

3. Once the scanning is complete, you can see a list of all BLE devices. Select your CRB device and click on it.

4. Click on "***CONNECT***"

5. Once the CRB device is connected, you can see all the device details listed. Click on the top right menu and select "***Update Firmware***"



6. Click on "***CHECK FOR UPDATE***" icon.

7. It will display a message if there is a new version of firmware is available. Click on "***DOWNLOAD***".

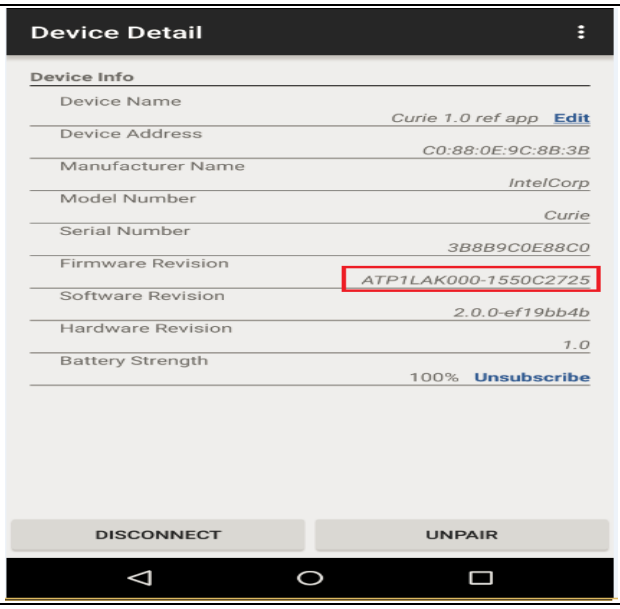
8. After the download is done, click on "***UPLOAD TO WEARABLE***" button

9. After the firmware binary file is completely transferred to the CRB over BLE (takes around 12 minutes) you should see Success message as below:

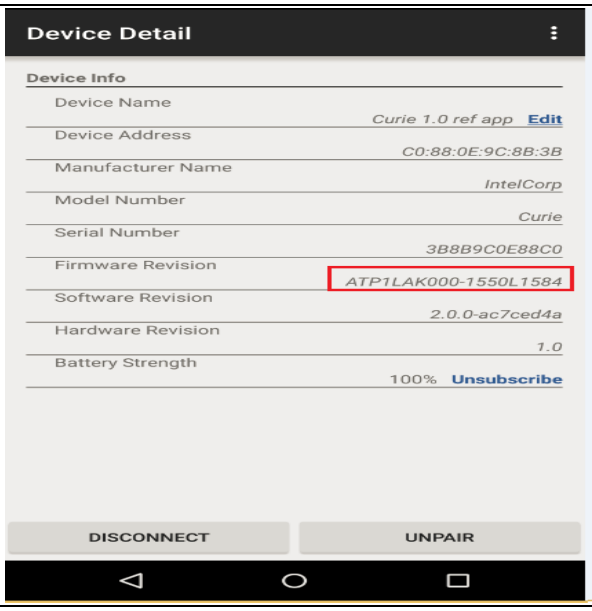
10. Wait one more minute for the CRB to reboot and reconnect the device.

Confirming the Firmware process:

1. Here we have started with CRB at version number: 2725 (the last 4 digits in the Firmware Revision):



2. Once the firmware update is complete and after reconnecting, you will see that Firmware revision is changed with the updated one.



8 Start Calibration of Raw Sensor Data

To enable sensor related test commands, we need to build the source code with BUILDVARIANT=debug. Compile and flash the generated binary by following the below steps:

```
$ cd /path/to/project
# For example;
$ cd wearable_device_sw/projects/curie_reference/

        /* make setup BOARD=<xxx> BUILDVARIANT=<yyy> */
$ make setup BOARD=crb BUILDVARIANT=debug
$ make image
$ make flash
```

Once you flash the device, you can see SS command at test command interface.

Steps to Calibrate:

1. Start sensor scanning for accelerometer sensor by inputting < *ss startsc ACCELEROMETER* > at test command interface.
2. Start sensor subscribe for accelerometer sensor by inputting < *ss sbc ACCEL 1 1000* > at test command interface.
3. Start sensor calibration to be ready for calibration by inputting < *ss clb start 0 ACCEL* > at test command interface. It will begin to output accelerometer raw sensor data from test command interface.
4. According to the raw sensor data, put the board in proper position and fasten it to make sure it should be in no motion status.

   In terms of proper position, handle the board as below:

   - In case of accelerometer calibration, make the target axle vertical to horizon line and move softly until the target value reach the maximum value.
   - In case of gyroscope calibration, just make the board in no motion status.
5. After the board is into no motion status for a while, start to get the calibration result data by inputting < *ss clb get 0 ACCEL* > at test command interface. It will go on outputting raw sensor data and a set of calibration offset xyz. And we need to remember them. After a while, the outputting raw data will be change to the corrected value and stored into sensor_core system. During get calibration result process, make sure the board is in no motion status all the time.

   In case of accelerometer calibration, we need to take turns to process step 4 and step 5 to calibrate all the three axles.
6. Set calibration by inputting < *ss clb set 0 ACCEL x y z* > at test command interface, these x/y/z were decided by step 4 and step 5. And these data will be stored into flash.
7. Optional Step: Stop calibration by inputting < *ss clb stop 0 ACCELEROMETER* > at test command interface.
8. Reboot the device, and after boot up the stored calibration result data will be set automatically.

9 Debug Procedure

To debug an Intel® Curie™ module, connect a micro USB cable to SoC USB and debug USB connectors for power.

9.1 Intel® Quark™ processor debug process

1. Connect to the CRB: (Terminal-1 in the host PC).

```
screen -L /dev/ttyUSB1 115200
```

2. Start server and console client from the project folder. (Terminal-2 in the host PC). This command sets up the gdb debug environment and starts the gdb client console.

```
make debug_console_qrk ELF=../../../out/current/firmware/quark.elf
```

9.2 ARC (DSP sensor hub) debug process

1. Connect to the CRB: (Terminal-1 in the host PC).

```
screen -L /dev/ttyUSB1 115200
```

2. Start server and console client from the project folder. (Terminal-2 in the host PC). This command sets up the gdb debug environment and starts the gdb client console.

```
make debug_console_arc ELF=../../../out/current/firmware/arc.elf
```

9.3 Useful debug commands

Serial number	Useful commands
1	(gdb) hb "file:line"
2	(gdb) print "var_name"
3	(gdb) info breakpoints
4	(gdb) delete "num_breakpoint"
5	(gdb) delete break (delete all breakpoints)

§