Intel® Curie™ Module

Software User Guide

February 2016

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	Added information on downloading code from GITHUB.	January 28, 2016

§

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	Curie™ reference board
DFU	Device Firmware Update
DIS	Device Information Service
GDB	GNU Debugger
GNU	GNU's not Unix
GPIO	General purpose input output
GPS	Global positioning system
IO	Input/output
ISPC	Intel Software Platform for Curie™
JTAG	Joint Test Action Group
LTS	Long-term support
NFC	Near Field Communications
OHRM	Optical Heart Rate Monitor
OTA	Over-the-air
SDK	Software development kit
SoC	System on Chip
SPI	Serial peripheral interface
UART	Universal asynchronous receiver/transmitter
USB	Universal serial bus
USB/DFU	USB Device Firmware Update
UUID	Universally unique identifier

§

2 One-time Setup

A Linux* OS-based host PC is required to build the Intel® Curie™ Module source code. Linux* Ubuntu 14.04 LTS is currently the only distribution that has been tested and is supported.

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
```

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
```

2.1 Access to Intel Github and Intel® IQ software web link.

Only Registered users can download the Intel® ISPC source code, tools, sample SDK examples and relevant documents from the Intel® Curie™ developer portal (*https://sandbox.iqsoftwarekit.intel.com)*.

Please note this is a two-step process. First register to the Intel® Curie™ developer portal. Then register your "github username" with Intel to be able to access the sources, tools and documentation.

Step-1

In order to access the Intel® Software Platform for Curie™ Beta Site (*https://sandbox.iqsoftwarekit.intel.com*), send e-mail to *ndg-dev-ops@intel.com* requesting to create an account in the IQ Software Kit sandbox environment. You will receive an e-mail once your access is approved. Please follow the below steps to activate your account.

a) Go to the portal *https://iqsoftwarekit.intel.com*.

b) Click on the "Sign in" Button.

c) Click on "Forgot Your Password?" Link.

d) Please enter the registered email Id.
You will receive an auto-generated email.

e) In the auto-generated email you receive, click on "Reset password."

f) Enter the new password and click on "Submit."

g) Log in to the Developer Portal.

Step-2

Please send an e-mail mentioning your "***github user ID"*** to *iqsoftwarekit-support@intel.com* requesting access to the iqsoftware github server from Intel®

(NOTE: If you don't already have a github account, you can sign up for GitHub in *https://github.com*)

3 Build Intel® Quark™ SE Code

3.1 Download the sources

You can download the source code using the Linux* command line, or using the Curie Beta Site landing page.

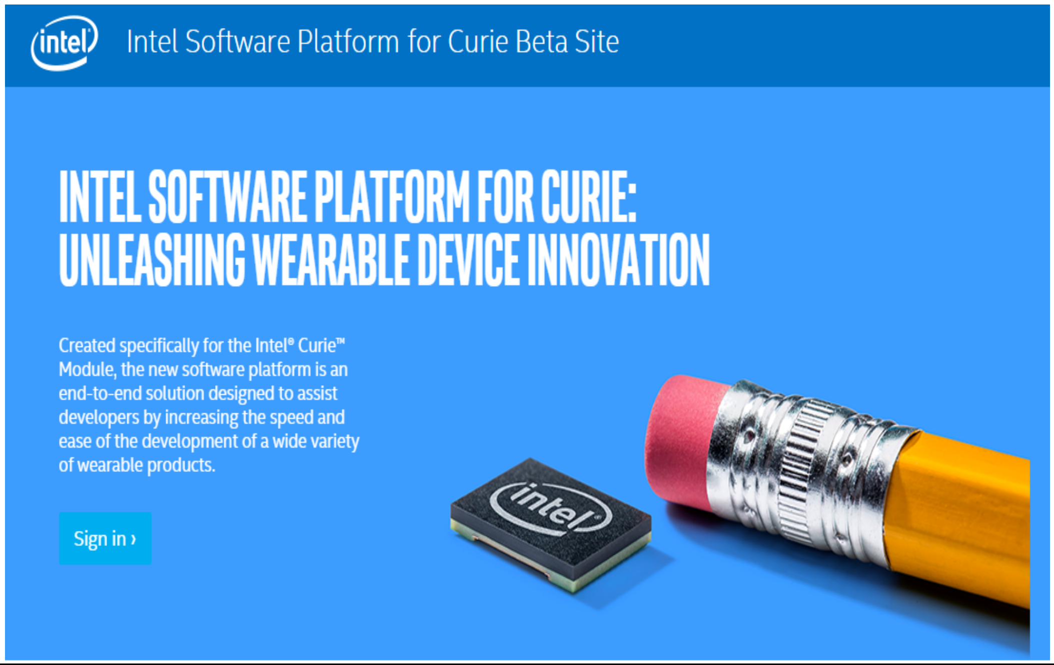
3.1.1 To download the source code at the command line

You will need to provide your GitHub user name (which should already be registered with Intel as described in Step-2 of section 2.1) and password.

```
$ git clone https://github.com/intel-ndg/iqsoftwarekit.git
```

3.1.2 To download the source code from the Curie™ Beta Site

1. Log in to Intel Software Platform for Curie™ Beta Site (*https://sandbox.iqsoftwarekit.intel.com*). The Curie™ Beta Site landing page will look like this:

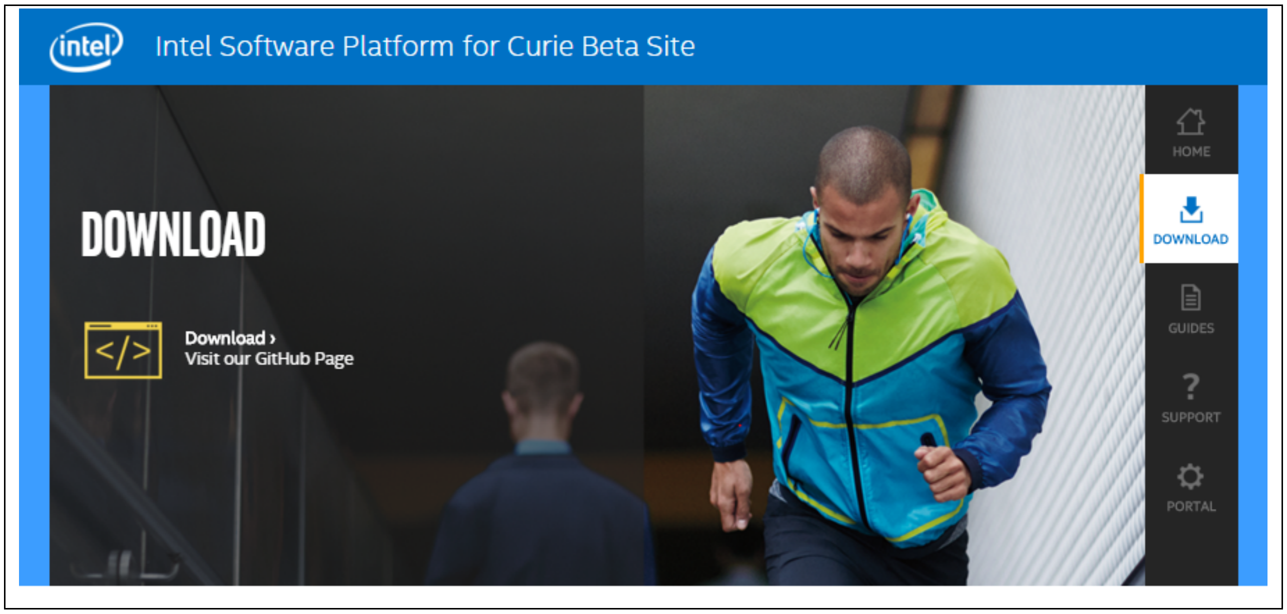


INTEL SOFTWARE PLATFORM FOR CURIE: UNLEASHING WEARABLE DEVICE INNOVATION

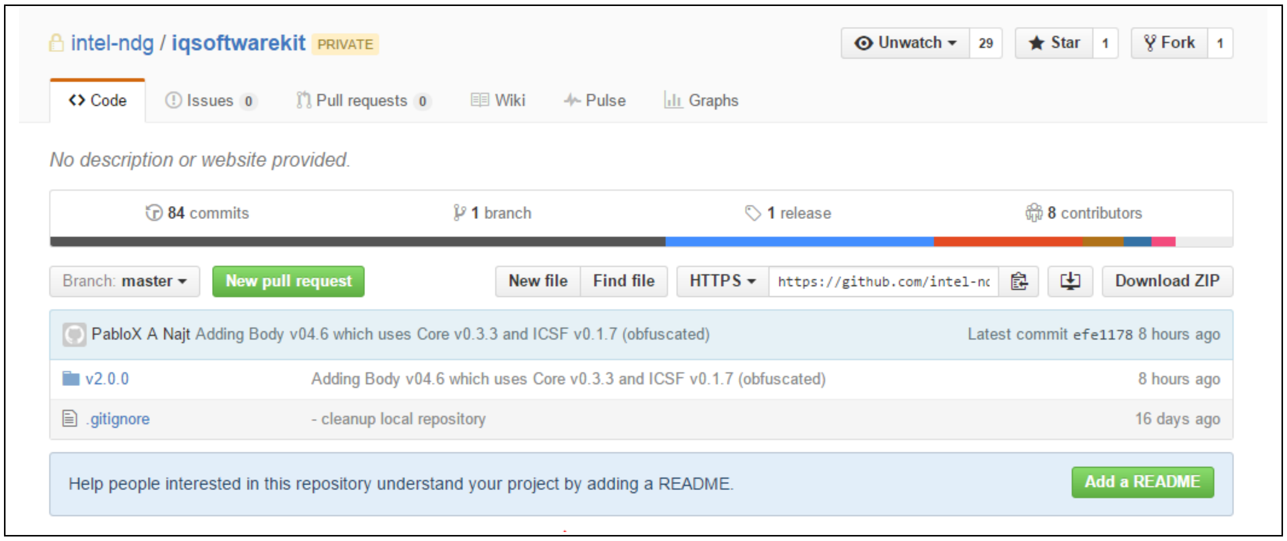
Created specifically for the Intel® Curie™ Module, the new software platform is an end-to-end solution designed to assist developers by increasing the speed and ease of the development of a wide variety of wearable products.

2. Click on the "Sign In" icon and enter your registered e-mail and password to log in. Once you log in, you can see sidebars on the home page.
3. Click on the "**Download**" icon which will redirect you to the Intel® GitHub server.

   Note: Please make sure you are logged in to GitHub prior to clicking on the "Visit our GitHub Page" icon.



4. Click on the "*Download Zip*" icon to download the sources.



intel-ndg / iqsoftwarekit PRIVATE

Unwatch 29 | Star 1 | Fork 1

Code | Issues 0 | Pull requests 0 | Wiki | Pulse | Graphs

No description or website provided.

84 commits | 1 branch | 1 release | 8 contributors

Branch: master | New pull request | New file | Find file | HTTPS | https://github.com/intel-nc | Download ZIP

PabloX A Najt Adding Body v04.6 which uses Core v0.3.3 and ICSF v0.1.7 (obfuscated)		Latest commit efe1178 8 hours ago
v2.0.0	Adding Body v04.6 which uses Core v0.3.3 and ICSF v0.1.7 (obfuscated)	8 hours ago
.gitignore	- cleanup local repository	16 days ago

Help people interested in this repository understand your project by adding a README. Add a README

5. Unzip the downloaded source files on your Linux* machine:

```
$ unzip iqsoftwarekit-master.zip
```

3.2 Tools Installation (One-time setup)

Install Intel® Phone Flash Tool Lite:

```
$ cd v2.0.0/device_software/tools
$ sudo dpkg -i phoneflashtoollite_5.3.4.1_linux_x86_64.deb
```

Install the Intel maker/wearable device tool

```
$ cd v2.0.0/device_software/tools
$ sudo dpkg -i Intel_Maker_and_Wearable_Device_Tools_1.0.2.0_linux_x86_64.deb
```

Assuming all of the host prerequisites and tools setup 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. (Optional) Custom configuration

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.

3.3 Build Environment 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
<iqsoftwarekit>/v2.0.0/device_software/intel_iq_sdk/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 following values:

1. xxx = `crb`
   (for Intel® Curie™ Reference Board)
2. yyy = `release` or `debug`

If nothing is entered after the ‘make setup’ command, the default values “BOARD=crb” and “BUILDVARIANT=release” are assigned.

For more information, type:

```
$ make help
```

Or optionally view the project *Makefile* which can provide many useful details. The setup command creates the build output directory, and prepares the project environment by compiling host tools and saving 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 most recent build environment you set up, the current environment, is identified using a symbolic link.

Example:

```
├── curie_reference_crb_release
├── current -> /home/user/iqsoftwarekit/v2.0.0/device_software/
intel_iq_sdk/wearable_device_sw/../out/curie_reference_crb_release
└── host_tools
```

3.4 Image creation

- If you want to flash your device using a USB cable, then create a flash firmware image. See "To create a flash firmware image."
- If you want to flash your firmware image using BLE, then create an OTA firmware image. See "To create an OTA firmware image."

3.4.1 To create a flash firmware image

Use the *make package* command to a create firmware image. Then jump to "Section 4 Flash the Board."

```
$ make package
```

The resulting binaries are created in the folder *<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.

3.4.2 To create an OTA firmware image

Use "make otapackage" to create the firmware image. See the *Over-the-Air Firmware Update Guide* for detailed instructions.

```
$ make otapackage
```

The resulting binaries are created in the folder *<intel_iq_sdk>/out/current/ota/pub/* .

The OTA firmware image is ***package.ota.bin***.

§

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 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)

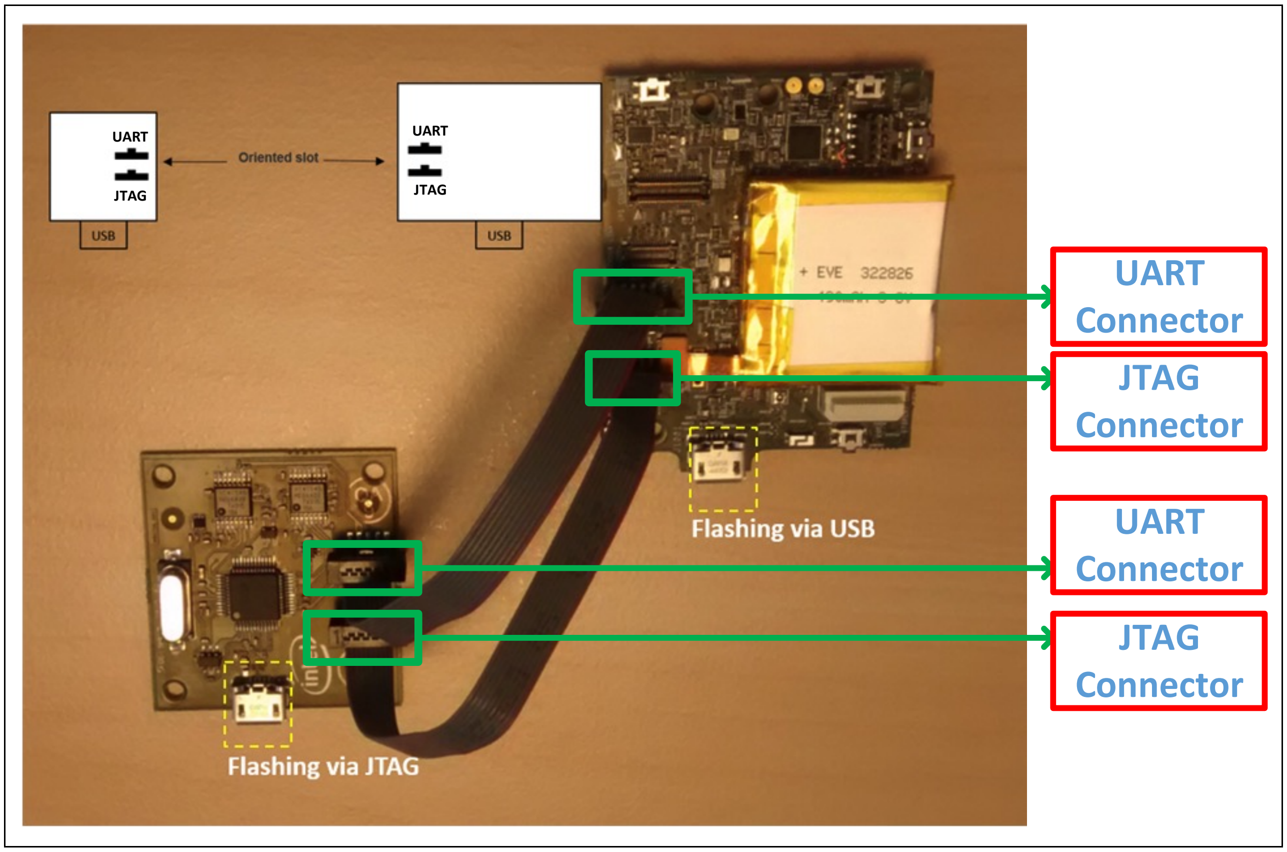
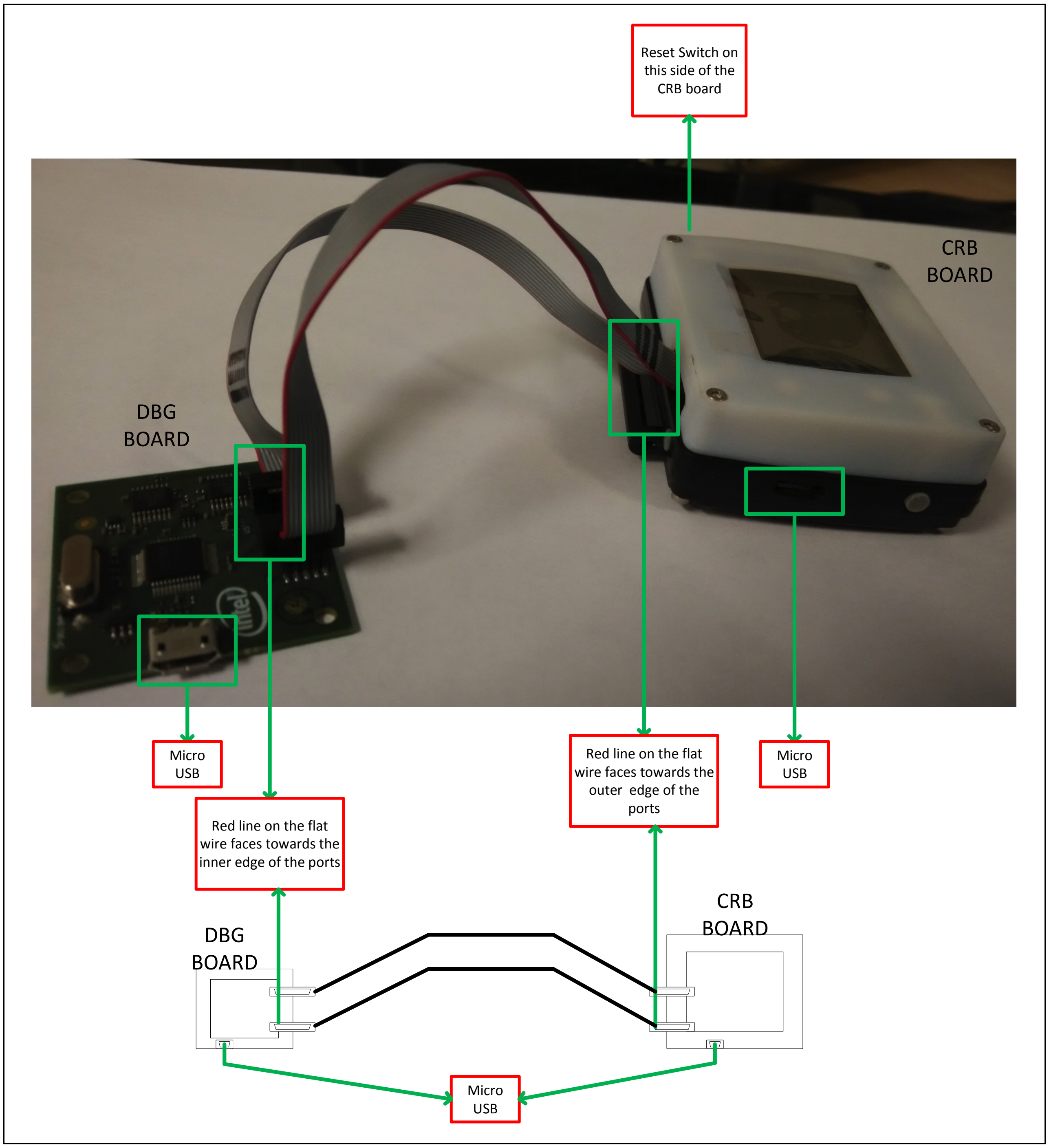


Figure 2 Board connections (main board with display unit and debug board)


Reset Switch on this side of the CRB board
CRB BOARD
DBG BOARD
Micro USB
Red line on the flat wire faces towards the inner edge of the ports
Red line on the flat wire faces towards the outer edge of the ports
Micro USB
CRB BOARD
DBG BOARD
Micro USB

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

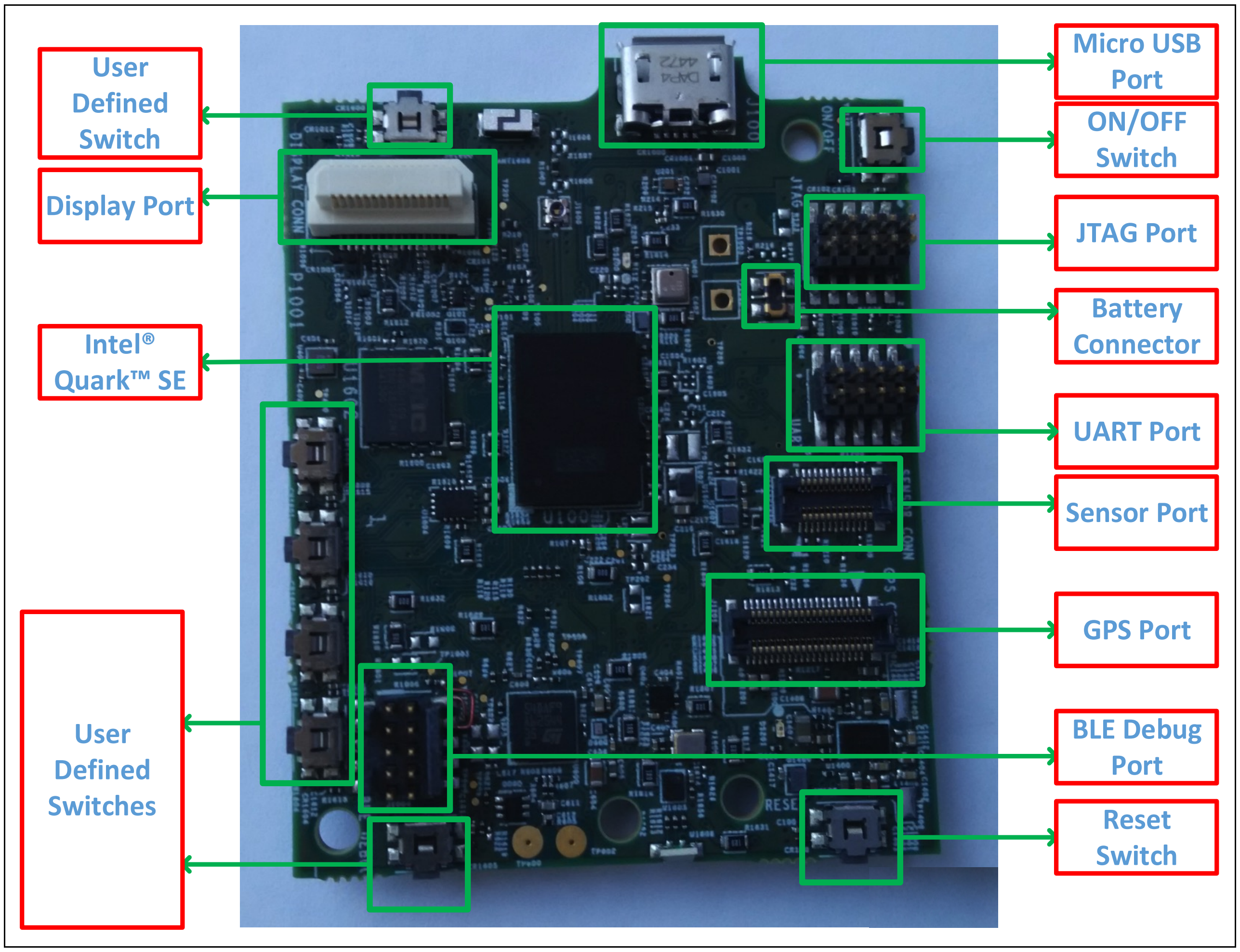


Table 2 CRB front items – description

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

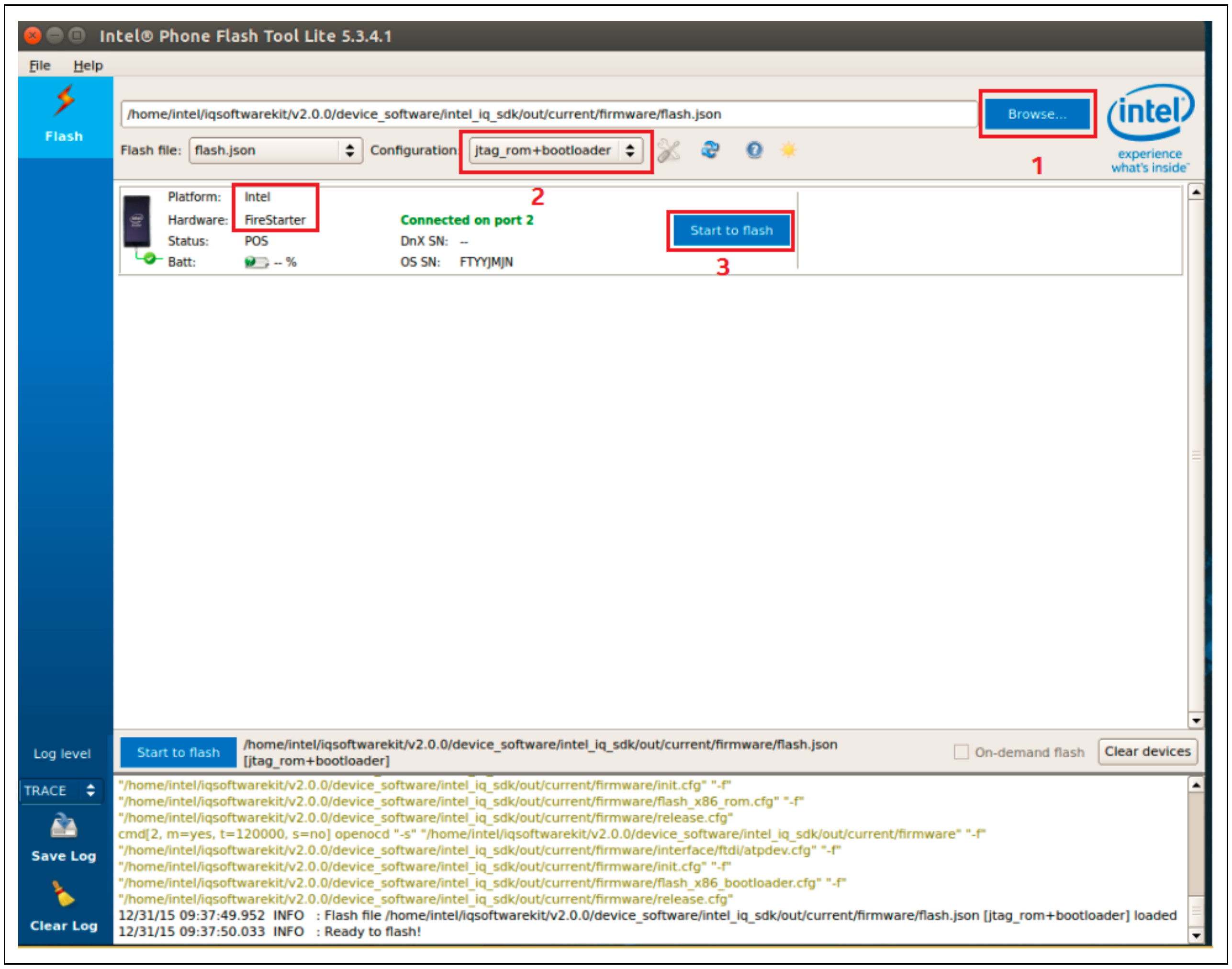
You may also flash the board using Intel® Phone Flash Tool Lite.

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



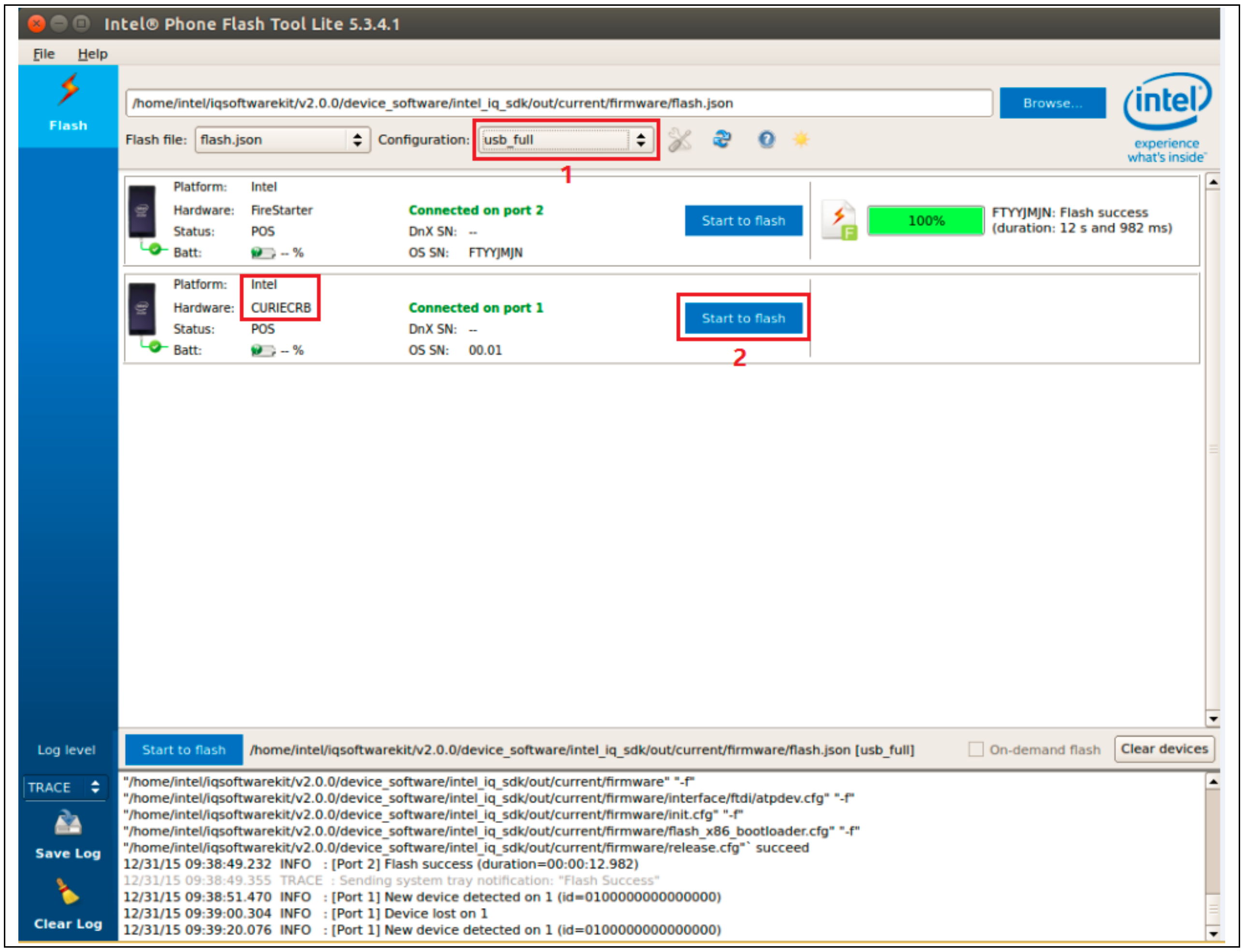
```
/home/intel/iqsoftwarekit/v2.0.0/device_software/intel_iq_sdk/out/current/firmware/flash.json
```

```
/home/intel/iqsoftwarekit/v2.0.0/device_software/intel_iq_sdk/out/current/firmware/flash.json
[jtag_rom+bootloader]
"/home/intel/iqsoftwarekit/v2.0.0/device_software/intel_iq_sdk/out/current/firmware/init.cfg" "-f"
"/home/intel/iqsoftwarekit/v2.0.0/device_software/intel_iq_sdk/out/current/firmware/flash_x86_rom.cfg" "-f"
"/home/intel/iqsoftwarekit/v2.0.0/device_software/intel_iq_sdk/out/current/firmware/release.cfg"
cmd[2, m=yes, t=120000, s=no] openocd "-s" "/home/intel/iqsoftwarekit/v2.0.0/device_software/intel_iq_sdk/out/current/firmware" "-f"
"/home/intel/iqsoftwarekit/v2.0.0/device_software/intel_iq_sdk/out/current/firmware/interface/ftdi/atpdev.cfg" "-f"
"/home/intel/iqsoftwarekit/v2.0.0/device_software/intel_iq_sdk/out/current/firmware/init.cfg" "-f"
"/home/intel/iqsoftwarekit/v2.0.0/device_software/intel_iq_sdk/out/current/firmware/flash_x86_bootloader.cfg" "-f"
"/home/intel/iqsoftwarekit/v2.0.0/device_software/intel_iq_sdk/out/current/firmware/release.cfg"
12/31/15 09:37:49.952  INFO   : Flash file /home/intel/iqsoftwarekit/v2.0.0/device_software/intel_iq_sdk/out/current/firmware/flash.json [jtag_rom+bootloader] loaded
12/31/15 09:37:50.033  INFO   : Ready to flash!
```

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



/home/intel/iqsoftwarekit/v2.0.0/device_software/intel_iq_sdk/out/current/firmware/flash.json

Flash file: flash.json Configuration: usb_full

Platform:	Intel		
Hardware:	FireStarter	Connected on port 2	
Status:	POS	DnX SN:	--
Batt:	-- %	OS SN:	FTYYJMJN

FTYYJMJN: Flash success (duration: 12 s and 982 ms)

Platform:	Intel		
Hardware:	CURIECRB	Connected on port 1	
Status:	POS	DnX SN:	--
Batt:	-- %	OS SN:	00.01

/home/intel/iqsoftwarekit/v2.0.0/device_software/intel_iq_sdk/out/current/firmware/flash.json [usb_full]

Log level: TRACE

```
"/home/intel/iqsoftwarekit/v2.0.0/device_software/intel_iq_sdk/out/current/firmware" "-f"
"/home/intel/iqsoftwarekit/v2.0.0/device_software/intel_iq_sdk/out/current/firmware/interface/ftdi/atpdev.cfg" "-f"
"/home/intel/iqsoftwarekit/v2.0.0/device_software/intel_iq_sdk/out/current/firmware/init.cfg" "-f"
"/home/intel/iqsoftwarekit/v2.0.0/device_software/intel_iq_sdk/out/current/firmware/flash_x86_bootloader.cfg" "-f"
"/home/intel/iqsoftwarekit/v2.0.0/device_software/intel_iq_sdk/out/current/firmware/release.cfg"` succeed
12/31/15 09:38:49.232  INFO   : [Port 2] Flash success (duration=00:00:12.982)
12/31/15 09:38:49.355  TRACE  : Sending system tray notification: "Flash Success"
12/31/15 09:38:51.470  INFO   : [Port 1] New device detected on 1 (id=0100000000000000)
12/31/15 09:39:00.304  INFO   : [Port 1] Device lost on 1
12/31/15 09:39:20.076  INFO   : [Port 1] New device detected on 1 (id=0100000000000000)
```

§

5 Customize the Source Code

5.1 Create a new Intel® Curie™ Module-based project

The Intel® Curie™ Module 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 common targets sub-makefile, and to define a few variables.

The *projects/curie_common/build/curie_common_targets.mk* file contains targets minimally required to build an Intel® Curie™ Module-based product:

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

The *curie_common_targets.mk* file will be included in the project Makefile after having defined the following variables:

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

```
PROJECT      ?= curie_reference

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

# Version of the reference app. We use Intel IQ SDK version number for the
# reference application
include ../../build/wearable_device_sw_version.mk
VERSION_MAJOR   := $(WEARABLE_DSW_VERSION_MAJOR)
VERSION_MINOR   := $(WEARABLE_DSW_VERSION_MINOR)
VERSION_PATCH   := $(WEARABLE_DSW_VERSION_PATCH)

include ../curie_common/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 SDK already includes several generic boards, drivers, and services. But your project can add custom items using optional Kconfig extension files. Example of a project source tree:

```
├── Kbuild.mk
├── project.Kconfig
├── services.Kconfig
├── drivers.Kconfig
├── arc
│   ├── Kbuild.mk
│   ├── defconfig_ctb
│   ├── defconfig_crb
│   ├── defconfig_orb
│   └── main.c
```

```
├── quark
│   ├── Kbuild.mk
│   ├── defconfig_ctb
│   ├── defconfig_crb
│   ├── defconfig_orb
│   ├── main.c
│   └── sensing.c
├── ble_core
│   └── ble_core_defconfig
└── Makefile
```

5.1.1 Project source tree

The Intel® Curie™ Module SDK uses a recursive build system inspired by the Linux* kernel Kbuild. The SDK expects to find the project source tree at a location specified by the PROJECT_PATH environment variable The PROJECT_PATH environment variable is defined in the Makefile.

This directory contains:

- The optional project root Kbuild.mk.
- The optional project specific configuration files. See "Defining a custom build configuration (basic step-by-step)."

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 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: Execute the make setup command:

```
$ make setup
```

Step 2: Execute one of the following menuconfig commands depending upon your specific usage:

- `$ make quark_menuconfig      // For Intel® Quark™`
- `$ make arc_menuconfig        // For ARC`
- `$ make ble_core_menuconfig   // For BLE`

These commands launch a menu-driven text-based configurator. You can traverse into different menus of your choice and modify the configuration options.

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

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

Step 3: Execute the make package command:

```
$ make package
```

5.3 Configuration patterns

- Features
- Services
- Drivers
- Boards

5.3.1 Features

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

- HAS_XX flag indicates that a feature is available.
- XX flag indicates that a feature is selected.

Example:

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

To offer more flexibility, generic features can be split into more specific features that implement them.

Example:

```
config HAS_SPI_FLASH
    select HAS_FLASH
```

5.3.2 Services

Services represent high-level functional capabilities of the target such as 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 the Quark Menu Configuration window

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 software giving access to a specific hardware feature. Drivers are described by flags corresponding to the component they provide access to. 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
```

In this configuration example:

- The board will display the availability of an SPI flash.
- The SPI flash provides the flash feature.
- The flash feature enables the storage service implementation to be selected.
- In turn, the SPI flash feature is selected.
- The board selects its SPI flash driver.

5.4 Adding a new driver or service

All device drivers are located under "*<intel_iq_sdk>/wearable_device_sw/bsp/src/drivers/*"

To add your own driver:

1. 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.
   Kconfig is the file that drives the configuration process and lists the features present in the subdirectory.
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. Configure the source code to add your newly created driver in the image.
5. 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

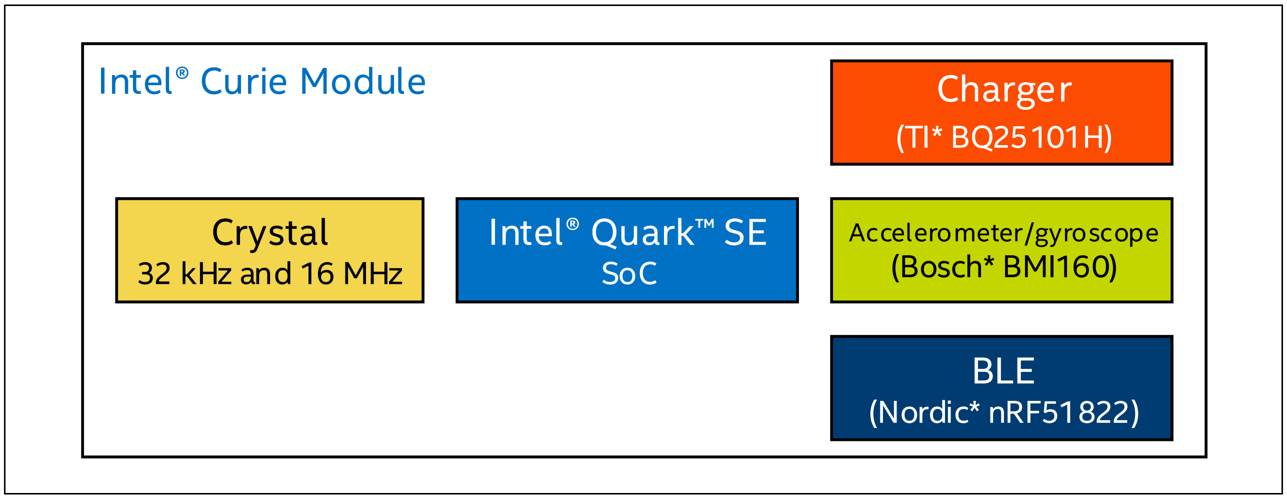
Option	Description
config	This defines the configuration option.
bool	The variable can be set only to y or n.
select <Option>	If you select this variable, the option present in the argument to select will be automatically enabled.
depends on <condition>	The option can only be enabled if the required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

- A Bosch* BMI160
- Bluetooth communication using an on-module Nordic* nRF51822
- A battery charger built using a Texas Instruments* BQ25101H
- Other external devices such as sensors, GPS, display, and NFC

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:

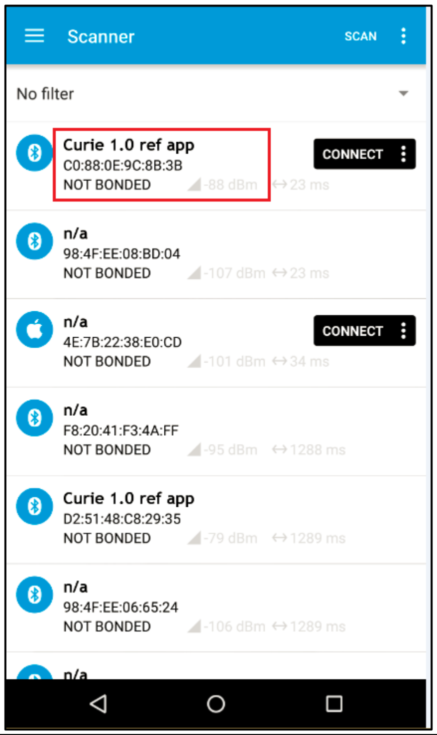
- Simplified security handling
- **BLE DIS service** (UUID: 0x180a): Device data such as software version are displayed as per the Bluetooth specification.
- **BLE BAS service** (UUID: 0x180f): BLE Battery level service as per the Bluetooth specification. It uses the battery service to automatically update the battery level characteristic.

To test BLE services:

1. Build the source code for the *curie_reference* project. See "Build Intel® Quark™ SE Code" for details on building the sources.
2. Flash the generated images on the Intel® Curie™ Module. See "Flash the Board" for details on flashing.

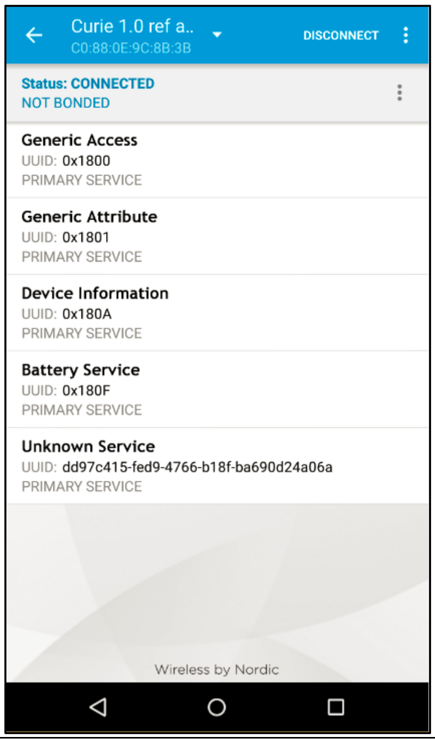
   A BLE demo application has been implemented for the *curie_reference* project. The demo application enables the BLE stack and registers the following BLE services: Device Information Service (DIS) and Battery Service (BAS). Once the BLE services are registered, the initialized services are displayed. The default name of the registered Intel® Curie™ Module BLE device is "Curie 1.0 ref app"
3. Go to *https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp&hl=en* to download and install the Nordic* *nRF Master Control Panel* application onto your BLE-capable Android* device.
4. Open the *nRF Master Control Panel* application in your Android* device. You can see "Curie 1.0 ref app" listed in the found devices.

Figure 7 Nordic* Android* app



5. Connect to "Curie 1.0 ref app". Once the device is connected, you can tap on the registered service tabs to see more details.

Figure 11 Nordic* Android* app: connected


Curie 1.0 ref a..
C0:88:0E:9C:8B:3B
DISCONNECT

Status: CONNECTED
NOT BONDED

Generic Access
UUID: 0x1800
PRIMARY SERVICE

Generic Attribute
UUID: 0x1801
PRIMARY SERVICE

Device Information
UUID: 0x180A
PRIMARY SERVICE

Battery Service
UUID: 0x180F
PRIMARY SERVICE

Unknown Service
UUID: dd97c415-fed9-4766-b18f-ba690d24a06a
PRIMARY SERVICE

Wireless by Nordic

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 file *sensing.c (<intel_iq_sdk>/wearable_device_sw/projects/curie_reference/quark/sensing.c)*. The application subscribes data for gestures, tapping, and step-counter. You can write your own algorithm and process the required data.

The v2 release provides a sample reference application code, which includes the open source sensor code and reports the accelerometer 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 as 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. See "Build Intel® Quark™ SE Code" for details on building the sources.
2. Flash the generated images on the CRB. See "Flash the Board" 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 to perform gestures. You can record those activities on the serial logs. Figure 8 shows a typical log message from the sample demo application.

Figure 8 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 Start Calibration of Raw Sensor Data

To enable sensor related test commands:

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 package
$ make flash
```

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

Steps to Calibrate:

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

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

   - For accelerometer calibration, position the target axle vertical to a horizon line and move the board slowly until the target value reaches the maximum value.
   - For gyroscope calibration, just set the board to No Motion status.
5. After the board is in No Motion status for a while, start to get the calibration result data by inputting < *ss clb get 0 ACCEL* > in the test command interface.
   Raw sensor data output will continue display, until a set of calibration offset values xyz is finally displayed.
6. Make note of any calibration offset values displayed.
   After a while, the raw data output displayed will change to the corrected value, and stored into the sensor_core system. During the “get calibration result” process, make sure the board is in No Motion status the entire time.

   For accelerometer calibration, repeat steps 4 and step 5 to calibrate all the three axles.
7. Set calibration by inputting < *ss clb set 0 ACCEL x y z* > at the test command interface.
   These xyz values were determined in steps 5 and step 6, and these data will be stored into flash.
8. (Optional) Stop calibration by inputting < *ss clb stop 0 ACCELEROMETER* > at the test command interface.
9. Reboot the device.
   After boot up, the stored calibration result data will be set automatically.

8 Debug Procedure

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

8.1 Intel® Quark™ processor debug process

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

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

2. Start the server and the console client from the project folder. (Terminal-2 in the host PC).
   The following example sets up the GNU debug environment and starts the GNU debugger (GDB) client console:

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

8.2 ARC (DSP sensor hub) debug process

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

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

2. Start the server and the console client from the project folder. (Terminal-2 in the host PC).
   The following example sets up the GNU debug environment and starts the GDB client console.:

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

8.3 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)

§