Intel® Quark™ Microcontroller Software Interface Bootloader

User Guide

April 2017
6.3 FM Functionality Usage.................................................................28
6.3.1 Creating and Flashing a QFU Image ...........................................28

7.0 DFU over USB Guide ......................................................................31
7.1 Hardware Setup...............................................................................31
7.2 Software Setup................................................................................32
  7.2.1 Enabling Firmware Management and Firmware Update with Authentication
        Functionality in the Bootloader.......................................................32
  7.2.2 Writing a New Key to Device.......................................................33
7.3 FM Functionality Usage....................................................................34
  7.3.1 Creating and Flashing a QFU Image .............................................34
7.4 Additional Notes and Known Bugs..................................................35
  7.4.1 Bootloader - DFU-UTIL Flashing Error ........................................35

8.0 Firmware Update with No Authentication ........................................36
  8.1.1 Disabling Authentication Feature in Bootloader.........................36
  8.1.2 Simple Way (Using ‘make flash’ Command) .................................36

9.0 Application Erase and System Information Retrieval ......................38
  9.1 Erase Applications..........................................................................38
  9.2 System Information..........................................................................38

10.0 Performing Mass Erase ..................................................................40

11.0 Installing DFU-UTIL-QDA in Windows* .......................................41
  11.1 Windows* Native with MSYS ........................................................41
  11.2 Windows Cross-Compile from Ubuntu 16.04.................................42

Figures
Figure 1. Entering Firmware Management Mode using GPIO and QDA Protocol ........................................19
Figure 2. Firmware Management Protocol Stack .....................................................................................20
Figure 3. Grounding FM GPIO Pin on Intel® Quark™ SE Microcontroller C1000 Developer Board ...............24
Figure 4. Grounding FM GPIO Pin on Intel® Quark™ Developer Kit D2000 .............................................25
Figure 5. Intel® Quark™ Developer Kit D2000 Board Set Up .................................................................26
Figure 6. Intel® Quark™ SE Microcontroller C1000 Developer Board Required USB Ports and Pins ..........31

Tables
Table 1. Supported Transports for Firmware Update.................................................7
Table 2. Terminology ..............................................................................7
Table 3. Boot Functionality Main Features......................................................9
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Bootloader Management Operation</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>Supported Transports for Firmware Management</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>Overview of Bootloader Resources</td>
<td>13</td>
</tr>
<tr>
<td>7</td>
<td>JTAG Probe Pin</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>FM Pin</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>FM UART</td>
<td>14</td>
</tr>
<tr>
<td>10</td>
<td>FM USB</td>
<td>15</td>
</tr>
<tr>
<td>11</td>
<td>BL-Data Flash Section</td>
<td>15</td>
</tr>
<tr>
<td>12</td>
<td>2nd-Stage Flash Partition</td>
<td>15</td>
</tr>
<tr>
<td>13</td>
<td>Flash Protection Regions</td>
<td>16</td>
</tr>
<tr>
<td>14</td>
<td>Memory Protection Regions</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>Firmware Update with No Authentication</td>
<td>36</td>
</tr>
</tbody>
</table>
## Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>April 2017</td>
<td>002</td>
<td>• Added DFU over USB guide, Firmware Update with/without Authentication</td>
</tr>
<tr>
<td></td>
<td></td>
<td>over UART and USB guide.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Removed instruction on accessing Firmware Management mode through</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FM SW1/SW2 and use FM GPIO pins instead.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated the flashing instruction using UART.</td>
</tr>
<tr>
<td>August 2016</td>
<td>001</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>

§
1.0 Introduction

This document contains high-level bootloader information, including features available on bootloaders, booting sequence, and an overview of Firmware Management functionality. The document provides step-by-step instructions on enabling Firmware Management. It also describes procedures to flash firmware onto the target device using UART and USB (particularly on the Intel® Quark™ Microcontroller D2000 and Intel® Quark™ SE Microcontroller C1000).

Table 1. Supported Transports for Firmware Update

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Supported Firmware Update Transport</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Quark™ Microcontroller D2000</td>
<td>UART</td>
</tr>
<tr>
<td>Intel® Quark™ SE Microcontroller C1000</td>
<td>UART, USB</td>
</tr>
</tbody>
</table>

1.1 Terminology

Table 2. Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARC</td>
<td>Argonaut RISC Core</td>
</tr>
<tr>
<td>BSP</td>
<td>Board Support Package</td>
</tr>
<tr>
<td>DFU</td>
<td>Device Firmware Upgrade</td>
</tr>
<tr>
<td>FM</td>
<td>Firmware Management</td>
</tr>
<tr>
<td>FTDI</td>
<td>Future Technology Device International</td>
</tr>
<tr>
<td>GPIO</td>
<td>General-purpose input/output</td>
</tr>
<tr>
<td>LMT</td>
<td>Lakemont Processor Architecture</td>
</tr>
<tr>
<td>MCU</td>
<td>Microcontroller</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>OSC</td>
<td>Oscillator</td>
</tr>
<tr>
<td>OTP</td>
<td>One-Time Programmable</td>
</tr>
<tr>
<td>QMSI</td>
<td>Intel® Quark™ Microcontroller Software Interface</td>
</tr>
<tr>
<td>RISC</td>
<td>Reduced Instruction Set Computing</td>
</tr>
<tr>
<td>ROM</td>
<td>Read-only Memory</td>
</tr>
<tr>
<td>RTC</td>
<td>Real-Time Clock</td>
</tr>
<tr>
<td>SRAM</td>
<td>Static Random Access Memory</td>
</tr>
<tr>
<td>UART</td>
<td>Universal asynchronous receiver/transmitter</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>BIST</td>
<td>Build-in-Self-Test</td>
</tr>
<tr>
<td>EBP</td>
<td>Extended Base Pointer</td>
</tr>
<tr>
<td>GDT</td>
<td>Global Descriptor Table</td>
</tr>
<tr>
<td>QDA</td>
<td>DFU Protocol Adaptation for Intel® Quark™ Microcontrollers</td>
</tr>
<tr>
<td>QFU</td>
<td>Firmware Update Format for Intel® Quark™ Microcontrollers</td>
</tr>
<tr>
<td>QFM</td>
<td>Firmware Management Protocol for Intel® Quark™ Microcontrollers</td>
</tr>
<tr>
<td>IDT</td>
<td>Interrupt Descriptor Table</td>
</tr>
<tr>
<td>FPR</td>
<td>Flash Protection Region</td>
</tr>
<tr>
<td>MPR</td>
<td>Memory Protection Region</td>
</tr>
<tr>
<td>BL</td>
<td>Bootloader</td>
</tr>
</tbody>
</table>
2.0 Features

This section summarizes the main features of the bootloader. They are divided into two main categories:

- Features related to system boot functionality
- Features related to Firmware Management functionality

2.1 System Boot Functionality

The primary function of the bootloader is to initialize the system, then pass control to the application firmware (if present). If there is no application firmware, the bootloader puts the MCU into sleep mode. However, the bootloader still allows for a wake-up in the case of an external request to switch into Firmware Management mode. This allows you to install application firmware.

Table 3. Boot Functionality Main Features

<table>
<thead>
<tr>
<th>Features</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firmware Image Residency</td>
<td>The bootloader firmware image resides in the ROM OTP Flash.</td>
</tr>
<tr>
<td>32 MHz Si Trim Setup</td>
<td>A library to read manufacturing trim data and program the 32 MHz Si OSC trim functions is provided.</td>
</tr>
<tr>
<td>Unique Device Identifier (from OTP Flash)</td>
<td>A library to read the device unique ID is provided.</td>
</tr>
<tr>
<td>Boot Time – MCU (bare metal)</td>
<td>Time from reset to exit from bootloader (without system update) is less than 10 ms.</td>
</tr>
</tbody>
</table>

2.2 Firmware Management (FM) Functionality

The bootloader provides a Firmware Management (FM) mode to allow you to perform management operations on the device (for example, the installation of a new or updated application). Such management functionality is accessible to you by a specific management tool running on a host machine. The host machine (and the tool) uses a UART and USB to connect to the device.

Even if the Firmware Management mode is enabled in the bootloader, you can continue to use JTAG for firmware updates until the OTP bit is set. Once set, the JTAG is inaccessible.
2.2.1 Management Operations

Table 4. Bootloader Management Operation

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieve Firmware/Device Information</td>
<td>Allows you to retrieve information regarding the currently installed firmware (for example, the version) and the device (for example, hardware type).</td>
</tr>
<tr>
<td>Update the Application Firmware</td>
<td>The bootloader supports updating System Flash applications.</td>
</tr>
<tr>
<td>Erase the Application</td>
<td>Allows you to erase all the application code in flash.</td>
</tr>
<tr>
<td>Cryptographic key provisioning</td>
<td>Enable firmware update with authentication</td>
</tr>
</tbody>
</table>

A “firmware extraction” function is not provided, as such functionality is rarely considered useful for the user while presenting significant security risks.

2.2.2 Supported Transports

The Firmware Management functionality supports the UART communication mechanism and USB DFU, which is configured at compile time.

Table 5. Supported Transports for Firmware Management

<table>
<thead>
<tr>
<th>Transport</th>
<th>Description/Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART</td>
<td>Serial Communication</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
</tbody>
</table>

2.2.3 Security

The Firmware Management functionality prevents an attacker from installing unauthorized or malevolent firmware by implementing an update mechanism based on signed firmware using HMAC256 and SHA256 for firmware authentication and verification. Such a mechanism, once enabled, allows only the installation of authenticated firmware.

For a full description of the security features, see the Intel® Quark™ Microcontroller Software Interface – Bootloader Security Features Programmer’s Guide.
3.0 Bootloader

3.1 Bootloader Flow

This section describes the flow of the bootloader, from hardware boot to the start of the application.

The bootloader has some dedicated resources, which are described in Section 3.2, Bootloader Resources.

The bootloader flow is as follows:

1. Initialize x86 core.
   - Invalidate cache (Intel® Quark™ SE Microcontroller C1000 only).
   - Store Built-in-Self-Test (BIST) value in EBP.
   - Load the GDT table into the descriptor.
   - Move to 32-bit protected mode.
   - Enable cache (Intel® Quark™ SE Microcontroller C1000 only).

2. Check resume from sleep condition:
   Available only to Intel® Quark™ SE Microcontroller C1000; compile option: “ENABLE_RESTORE_CONTEXT=1”
   - Resume application execution if the device was put into sleep mode. A soft reboot is performed when a device comes out of sleep mode. The bootloader checks if the 'restore bit' of “GPS0” sticky register is set, then proceeds as follows:

<table>
<thead>
<tr>
<th>Restore bit set</th>
<th>Restore bit not set</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Set up security hardening:</td>
<td>• Continue normal boot.</td>
</tr>
<tr>
<td>— Enable flash write protection (Compile option: “ENABLE_FLASH_WRITE_PROTECTION=1”)</td>
<td></td>
</tr>
<tr>
<td>— Set-up BL-Data FPR (“FPR 0”).</td>
<td></td>
</tr>
<tr>
<td>— Set-up IDT/GDT MPR (“MPR 0”).</td>
<td></td>
</tr>
<tr>
<td>• Jump to the restored address stored in SRAM.</td>
<td></td>
</tr>
</tbody>
</table>

3. Set up primary peripherals and registers:
   - Stack pointer set-up
   - RAM set-up
   - Power set-up
   - Clock set-up

4. Check for JTAG probe:
   - The bootloader checks if the JTAG_PROBE_PIN is asserted (grounded). If so, the bootloader waits until the pin is deasserted (ungrounded). This unbricks a device with firmware that prevents JTAG from working correctly.
5. Initialize/sanitize Bootloader Data (BL-Data):
   – Check if BL-Data is blank or corrupted:
     | Condition                  | Description                                           |
     |---------------------------|-------------------------------------------------------|
     | BL-Data blank             | Initialize BL-Data (including trim-code computation). |
     | One copy of BL-Data corrupted | Recover BL-Data using valid copy.                     |
     | Both copies corrupted     | Enter infinite loop (unrecoverable error).             |
   – Sanitize partitions
     The bootloader checks for partitions marked as 'inconsistent' and erases them.

6. Set up secondary peripherals:
   – IRQ set-up
   – IDT set-up
   – Enable interrupts

7. Set memory violation policy:
   – Configure memory violation policy (for both RAM and flash) to trigger a warm reset.

8. Check if Firmware Management (FM) is requested:
   Compile option: “ENABLE_FIRMWARE_MANAGER=[uart|2nd-stage]”
   – Check if the FM pin is asserted (grounded) or the FM bit of sticky register “GPS0” is set. If so, the bootloader enters FM mode.

9. Check if x86 application is present:
   – Check if the first 4 bytes of the x86 partition are different from 0xffffffff (indicating that an application is present):
     | Application present | Application not present |
     |---------------------|-------------------------|
     | • Set up security hardening: | • Start FM mode.                  |
     |   — Enable flash write protection. |   (Compile option: |
     |     (Compile option: “ENABLE_FLASH_WRITE_PROTECTION=1”) |     “ENABLE_FIRMWARE_MANAGER=[uart|2nd-stage]”) |
     |   — Set up BL-Data FPR (“FPR 0”). |   — Clean up RAM to prevent leaking private data to application: |
     |   — Set up IDT/GDT MPR (“MPR 0”). |     — Clear x86 portion of RAM. |
     | • Clean up RAM to prevent leaking private data to application: |     — Reset stack pointer. |
     |   — Clear x86 portion of RAM. |   • Start x86 application: |
     |   — Reset stack pointer. |     — Jump to the application entry point. |
     | • Start x86 application: | |
     |   — Jump to the application entry point. | |

10. Enter infinite loop:
    – If no x86 application is present and FM is not enabled, or the application returns, the bootloader enters an infinite loop.
3.2 Bootloader Resources

The bootloader uses dedicated resources to guarantee its flow. To avoid unexpected behavior, hardware designers and application developers should pay attention how they use the resources. The bootloader does not use resources associated with a specific compile option if such an option is not set. Some used resources are freed before starting the application, while others remain reserved.

3.2.1 Overview

Table 6. Overview of Bootloader Resources

<table>
<thead>
<tr>
<th>Resources</th>
<th>Intel® Quark™ Microcontroller D2000</th>
<th>Intel® Quark™ SE Microcontroller C1000</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash (for BL-Data)</td>
<td>0x00200000 0x00201000</td>
<td>0x4002F000 0x40030000</td>
<td>Reserved</td>
</tr>
<tr>
<td>Sleep storage (in RAM)</td>
<td>N/A</td>
<td>0xA8013FDC 0xA8013FE0</td>
<td>Reserved</td>
</tr>
<tr>
<td>FM register</td>
<td>GPS0 bit 0</td>
<td>GPS0 bit 0</td>
<td>Reserved</td>
</tr>
<tr>
<td>Sleep register</td>
<td>N/A</td>
<td>GPS0 bit 1 GPS0 bit 2</td>
<td>x86 restore bit ARC restore bit</td>
</tr>
<tr>
<td>JTAG probe</td>
<td>GPIO_13</td>
<td>GPIO_15</td>
<td>Do not ground</td>
</tr>
<tr>
<td>FM GPIO pin</td>
<td>GPIO_2</td>
<td>AON_GPIO_4</td>
<td>Do not ground</td>
</tr>
<tr>
<td>UART port</td>
<td>UART 0</td>
<td>UART 1</td>
<td>Available to app</td>
</tr>
<tr>
<td>USB controller</td>
<td>N/A</td>
<td>USB 0</td>
<td>Available to app</td>
</tr>
<tr>
<td>Flash protection (for BL-data)</td>
<td>FPR 0</td>
<td>FPR 0</td>
<td>Locked</td>
</tr>
<tr>
<td>SRAM protection (for IDT and GDT)</td>
<td>FPR 0</td>
<td>FPR 0</td>
<td>Locked</td>
</tr>
</tbody>
</table>

3.2.2 GPIO Pins

The bootloader uses the following pins:
- JTAG probe pin
- FM pin

Table 7. JTAG Probe Pin

<table>
<thead>
<tr>
<th>SoC</th>
<th>Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Quark™ SE Microcontroller C1000</td>
<td>GPIO_15</td>
</tr>
<tr>
<td>Intel® Quark™ Microcontroller D2000</td>
<td>GPIO_13</td>
</tr>
</tbody>
</table>
The JTAG probe pin puts the bootloader into recovery mode. Recovery mode can be used when the JTAG is not able to connect to the device during runtime.

Table 8. FM Pin

<table>
<thead>
<tr>
<th>SoC</th>
<th>Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Quark™ SE Microcontroller C1000</td>
<td>AON_GPIO_4</td>
</tr>
<tr>
<td>Intel® Quark™ Microcontroller D2000</td>
<td>GPIO_2</td>
</tr>
</tbody>
</table>

The FM pin starts the bootloader in FM mode.

*Note:* During the boot process, do not ground GPIO pins that are used by the bootloader.

### 3.2.3 Sticky Registers

The bootloader uses the following sticky registers:

- Resume from sleep:
  - Compile option: “ENABLE_RESTORE_CONTEXT=1”
  - Intel® Quark™ SE Microcontroller C1000: “GPS0 bit1 and the 4 bytes in esram_shared”

  The bootloader uses this sticky register to handle resuming the application from a sleep power state. The 4 bytes of the common RAM that are defined as section esram_restore_info save the restore trap address.

- FM sticky bit:
  - Compile option: “ENABLE_FIRMWARE_MANAGER=[uart|2nd-stage]”
  - All SoCs: “GPS0 bit 0”

  This register starts the bootloader in FM mode.

*Note:* The application must not use the sticky registers used by the bootloader.

### 3.2.4 Peripherals

The bootloader uses the following peripherals:

Table 9. FM UART

<table>
<thead>
<tr>
<th>SoC</th>
<th>Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Quark™ SE Microcontroller C1000</td>
<td>UART1</td>
</tr>
<tr>
<td>Intel® Quark™ Microcontroller D2000</td>
<td>UART0</td>
</tr>
</tbody>
</table>
### Table 10. FM USB

Compile option: "ENABLE_FIRMWARE_MANAGER=2nd-stage" (and 2nd-stage bootloader programmed)

<table>
<thead>
<tr>
<th>SoC</th>
<th>Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Quark™ SE Microcontroller C1000</td>
<td>USB0</td>
</tr>
<tr>
<td>Intel® Quark™ Microcontroller D2000</td>
<td>N/A</td>
</tr>
</tbody>
</table>

The application can use both the FM UART and USB freely.

#### 3.2.5 Time Constraints

The bootloader guarantees a boot-time smaller than 10ms during normal boot. The boot time could increase during the first boot or after an FM session.

#### 3.2.6 Flash Resources

The bootloader is located in the OTP flash, but it also uses other portions of the flash (for example, to store metadata).

##### 3.2.6.1 BL-Data Flash Section

A part of the flash of the device is reserved for the bootloader data (BL-data), as follows.

<table>
<thead>
<tr>
<th>SoC</th>
<th>BL-Data Address Range</th>
<th>Size</th>
<th>Flash Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Quark™ Microcontroller D2000</td>
<td>0x00200000 - 0x00201000</td>
<td>4kB</td>
<td>Data</td>
</tr>
<tr>
<td>Intel® Quark™ SE Microcontroller C1000</td>
<td>0x4002F000 - 0x40030000</td>
<td>4kB</td>
<td>System flash 0</td>
</tr>
</tbody>
</table>

##### 3.2.6.2 2nd-Stage Flash Partition

When the 2nd-stage bootloader is enabled, a portion of flash is reserved for the 2nd-stage and therefore is not available to the application.

<table>
<thead>
<tr>
<th>SoC</th>
<th>BL-Data Address Range</th>
<th>Size</th>
<th>Flash Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Quark™ Microcontroller D2000</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
3.2.7 Memory Protection

3.2.7.1 Flash Protection

Part of the previously mentioned BL-Data flash section is available for the firmware to be read, while the other part is private for the bootloader itself.

To enforce this, before jumping to the application, the bootloader sets up a Flash Protection Region (FPR) to read-protect the private portion of BL-Data. This FPR is locked so it cannot be disabled or reused by the firmware. Trying to read a protected flash address triggers a warm reset.

Table 13. Flash Protection Regions

<table>
<thead>
<tr>
<th>SoC</th>
<th>FPR</th>
<th>Flash Controller</th>
<th>Protected Range</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Quark™ SE Microcontroller C1000</td>
<td>FPR_0</td>
<td>Controller 0</td>
<td>0x00200400 - 0x00201000</td>
<td>3kB</td>
</tr>
<tr>
<td>Intel® Quark™ SE Microcontroller C1000</td>
<td>FPR_0</td>
<td>Controller 0</td>
<td>0x4002F400 - 0x40030000</td>
<td>3kB</td>
</tr>
</tbody>
</table>

3.2.7.2 SRAM Protection

The bootloader also sets up a Memory Protection Region (MPR) to protect the portion of SRAM containing system data that is critical to the proper functioning of the SoC. Specifically, the MPR protects the x86 GDT and IDT against read/write access by agents different from the x86 core. This is a security hardening feature that limits the damage an attacker can do by hijacking some other agents (such as ARC or DMA). The MPR is locked and cannot be reused by the application.

**Note:** Since MPR granularity is 1kB and the combined size of GDT and IDT is smaller than 1kB, the MPR also protects part of the stack (which in the default RAM layout is located before the IDT). The main consequence for application developers is that they cannot use DMA to access local variables on the stack.

Table 14. Memory Protection Regions

<table>
<thead>
<tr>
<th>SoC</th>
<th>MPR</th>
<th>Protected Range</th>
<th>Size</th>
<th>Protected Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Quark™ Microcontroller D2000</td>
<td>MPR_0</td>
<td>0x00281C00 - 0x00282000</td>
<td>1kB</td>
<td>IDT + GDT + part of stack</td>
</tr>
<tr>
<td>SoC</td>
<td>MPR</td>
<td>Protected Range</td>
<td>Size</td>
<td>Protected Data</td>
</tr>
<tr>
<td>------------------------</td>
<td>------</td>
<td>-----------------------</td>
<td>------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>Intel® Quark™ SE</td>
<td>MPR_0</td>
<td>0xA8013C00 - 0xA8014000</td>
<td>1kB</td>
<td>IDT + GDT + part of stack + x86 restore info</td>
</tr>
</tbody>
</table>
4.0 Firmware Management

4.1 Overview

Firmware Management (FM) mode is an alternative to the normal operation mode. In FM mode, the device is ready to serve system management requests (for example, a system update request) from an external host. While FM mode is active, the MCU does not run its application, and serves management requests from the external host only.

4.1.1 Functionality

The Firmware Management mode provides the following services:

- System information retrieval (MCU type, system firmware version, and so on)
- Application firmware update
- Application firmware erase (erase all application code in flash) for non-authenticated mode
- Cryptographic key provisioning

4.1.2 Entering Firmware Management Mode

The MCU enters the Firmware Management mode only after an explicit request from the host.

Such a request can be received at any state during run-time. Moreover, the Firmware Management request can also be received when the bootloader does not find a valid firmware application image and puts the system into a sleep state. Therefore, the request mechanism can wake up the MCU.

If the application is running, it must be shut down before switching to Firmware Management mode. Indeed, the management operations may involve an application update or erase.

To summarize, entering the Firmware Management mode consists of the steps depicted in the following figure.
The specific wake-up event that the host is expected to generate depends on the communication medium chosen for Firmware Management: a GPIO pin is being used for UART and USB.

### 4.1.3 Exiting Firmware Management Mode

To exit the Firmware Management mode, reboot the system while the FM GPIO pin is not grounded. Such a reboot is performed after an explicit request from the host.

### 4.1.4 Firmware Management Protocol Stack

Even if FM functionality is available through different kinds of transports, a single common high-level protocol is defined to perform FM operations. This protocol is based on the USB DFU specification, a standard for performing firmware upgrades for USB devices. Even if the USB DFU specification focuses on firmware updates, the proposed FM protocol is generic. The protocol can also be used to enable the other functions envisioned for FM (that is, system information retrieval, key provisioning and application erase). Moreover, the DFU protocol can be adapted to transports other than USB.
As shown in Figure 2, the first step to adapt DFU to non-USB media is to provide a reliable and message-based transport layer. For UART, Xmodem-CRC is used to this end.

A DFU Protocol Adaptation for Intel® Quark™ microcontrollers, called QDA, is defined on top of the transport layer. The goal of this adaptation is to replicate some USB functionality and all the DFU commands on top of non-USB transport media.

Finally, a common DFU-based Firmware Management protocol (QFM), and a common firmware image format (QFU) is defined. The QFM Protocol also enables the use of DFU to perform Firmware Management operations that are different from firmware updates. (For example, system information retrieval, application erase, and key provisioning.) The QFU Format maintains firmware image compatibility with plain DFU (that is, the device can download firmware using any standard DFU tool).

Visit the GitHub* project for more information on the Intel® Quark™ microcontroller software stack and firmware management.
5.0 Initial Software Setup

5.1 Installing DFU-UTIL-QDA (DFU over UART)

DFU-UTIL-QDA is the tool required to flash QFU images over UART to the target or manage the device. You can download the software from GitHub* by following this [link]. Intel recommends that you perform this setup in root/admin mode so that you can run commands with root/admin privilege.

5.1.1 Linux

For a Linux host machine, before building the host tool, configure the build with the following command:

```
cd <QM_DFU_UTIL_DIR>
sudo apt-get install dh-autoreconf
./autogen.sh
./configure
```

Then compile it with the following:

```
make
```

The generated binary is `src/dfu-util-qda`. Install it into the GNU/Linux host with the following command:

```
sudo make install
```

The binary is put into `/usr/local/bin/`.

5.1.2 Windows

Intel provides a pre-built Windows binary that is available at this [link]. You can also build the binary yourself. For more information, see Chapter 11.0, “Installing DFU-UTIL-QDA in Windows*”.

5.1.3 Installing qmfmlib Library

The qmfmlib library supports the host side features of the Intel® Quark™ microcontroller sysupdate. The library is used by qm_make_dfu.py to convert compiled binaries to an Intel® Quark™ microcontroller-compatible DFU image before downloading to the device. The library is also used by qm_manage.py for firmware erase and information retrieval.

You must download the QM-Bootloader package from this link, and enter the following in CMD/terminal:

```
cd <QM_BOOTLOADER_DIR>/tools/sysupdate/
sudo python setup.py install
```

**Note:** Ensure that Python 2.7.x is already installed.

```
sudo apt-get install python
```

5.2 Installing DFU-UTIL (DFU over USB)

DFU-UTIL is the tool required to flash the QFU images over USB to the target or manage the device. This tool is supported on the Ubuntu*, Fedora*, and Windows* operating systems. Intel recommends that you perform this setup in root/admin mode so that you can run commands with root/admin privilege.

On Debian and Ubuntu systems you can install DFU-UTIL by typing the following:

```
sudo apt-get install dfu-util
```

On Fedora the command line command to install DFU-UTIL is:

```
sudo dnf install dfu-util
```

DFU-UTIL can also be installed manually on Windows. To install it, follow the instructions at this link.

5.3 Setting up the Software Environment

To use firmware management and firmware update with authentication functionality over UART and USB on the target platform, set up the environment first. Then flash the ROM that has enabled this functionality. To do that, perform the following steps:

1. Download and extract the latest toolchain (this is packaged with Intel® System Studio for Microcontroller), TinyCrypt, QM-Bootloader, and the QMSI package. If
Initial Software Setup

Intel® System Studio for Microcontrollers is not installed, you can get the toolchain from [this standalone tarball](#).

2. Remove the "-Master" extension on every downloaded file, for example: qmsi-master to qmsi, or qm-bootloader-master to qm-bootloader.

3. Export/set the following environment variable in CMD/terminal:

   ```
   export QMSI_SRC_DIR=<PATH TO SOURCE>/qmsi
   export QM_BOOTLOADER_DIR=<PATH TO SOURCE>/qm-bootloader
   export TINYCRIPT_SRC_DIR=<PATH TO SOURCE>/tinycrypt
   export IAMCU_TOOLCHAIN_DIR=<PATH TO SOURCE>/ISSM_2016.2.xxx/tools/compiler/gcc-ia/5.2.x/bin
   export ARCMCU_TOOLCHAIN_DIR=<PATH TO SOURCE>/ISSM_2016.2.xxx/tools/compiler/gcc-arc/4.8.x/bin
   ```

   **Note:** For Windows system, use `SET` instead of `export` and use `\` instead of `/` while pointing directories.

   **Note:** If using ISSM toolchain in Linux, the `<PATH TO SOURCE>` to IAMCU and ARCMCU toolchain could be in either these two locations:

   i. `/opt/intel/issm_2016.2.xxx/tools/compiler/bin/
   ii. `~/intel/issm_2016.2.xxx/tools/compiler/`
6.0  DFU over UART Guide

This chapter explains how to access FM mode and to upload a secured application image onto the target device through UART. Only signed images can be programmed when this feature is active.

These features are available on both the Intel® Quark™ Microcontroller D2000 and Intel® Quark™ SE Microcontroller C1000. FM is supported over both the Linux and Windows environments.

6.1  Hardware Setup

The hardware setup consists of three main steps:

- Connecting the board to the computer using the FM UART
- Connecting the FM GPIO pin to ground to enable Firmware Management mode.
- Connecting power supply to the board

*Note:* Power and FM UART share a USB port (C2) for Intel® Quark™ SE Microcontroller C1000.

**Figure 3.** Grounding FM GPIO Pin on Intel® Quark™ SE Microcontroller C1000 Developer Board
The FM UART is the UART that the bootloader uses for FM. It can be configured at bootloader compile time. The defaults are as follows:

- UART 1 for Intel® Quark™ SE Microcontroller C1000 Developer Board. UART 1 located in Primary USB Port (Serial Interface FTDI) as in Figure 6.

The FM GPIO pin is the pin used to put the device into FM mode. To enter FM mode, the device must be reset manually by pressing the reset button while the FM GPIO pin is grounded. To exit FM mode, unground the FM GPIO pin and press the reset button. The device cannot exit FM mode while the FM GPIO pin is grounded.

Intel advises you not to use the FM GPIO pin in a FM-enabled ROM. Otherwise, the board is likely to reboot and enter FM mode unexpectedly.

Section 6.1.1 describes additional hardware setup for the Intel® Quark™ Developer Kit D2000. For the Intel® Quark™ SE Microcontroller C1000 Developer Board, no extra setup is required.

6.1.1 Additional Hardware Setup for Intel® Quark™ Developer Kit D2000

Follow the Prepare Your Hardware instructions available at this link, and connect an FTDI cable to the UART pins as shown in the following figure.
6.2 Software Setup

6.2.1 Enabling Firmware Management and Firmware Update with Authentication Functionality in the Bootloader

1. While in QM-Bootloader directory, setup the software environment as explained in Section 5.3, “Setting up the Software Environment.” Intel recommends that you perform this setup in root/admin mode so that you can run commands with root/admin privilege.

2. Perform mass_erase, as explained in Section 10.0, “Performing Mass Erase”.

3. Enable the Firmware Manager for UART in the bootloader. The firmware update with authentication feature is enabled by default:
make clean

make SOC=<TARGET_SOC> ENABLE_FIRMWARE_MANAGER=uart

**Note:** `<TARGET_SOC>` values can be `quark_se` or `quark_d2000`

4. Flash the new ROM image to the target:

```
python $QMSI_SRC_DIR/tools/jflash/jflash.py -r quarkse_dev ./build/release/quark_se/rom/quark_se_rom_fm_hmac.bin
```

**Note:** For Intel® Quark™ Microcontroller D2000, change `quarkse_dev` to `d2000_dev`.

**Note:** To flash the created ROM in Windows system, use '\' to point directories.

**Note:** For Windows*, replace `$QMSI_SRC_DIR` with `%QMSI_SRC_DIR%`.

### 6.2.2 Writing a New Key to Device

With both Firmware Management and Authentication enabled on the bootloader, you must set a new authentication key to the target device.

The firmware manager uses two keys, a firmware key and a revocation key. Both keys can be written using `qm_manage.py`. The revocation key must be written to the device before the firmware key. As a security precaution, firmware update cannot be performed until both keys have been correctly provided to the device. Once a key is set, it can be updated only if the old key is known.

To generate the `<RV/FW_KEY_FILE>` key-file, `openssl` can be used as an alternative to other cryptographic tool. However, you must make sure both keys are 32 bytes long and passed to `qm-manage` as a binary file. For example:

```
sudo openssl rand 32 > <KEY_FILE>
```

**Note:** It is good practice never to store RV and FW key file in the same place.

Before assigning key to the device, the device must be in FM mode. To access FM mode, you must ground the FM-GPIO pin and power cycle the board.

### 6.2.2.1 Setting Revocation Key for the First Time

For first-time provision over UART, you must specify the new key and the port number, as follows:

```
make clean

make SOC=<TARGET_SOC> ENABLE_FIRMWARE_MANAGER=uart

python $QMSI_SRC_DIR/tools/jflash/jflash.py -r quarkse_dev ./build/release/quark_se/rom/quark_se_rom_fm_hmac.bin
```
cd <QM_BOOTLOADER_DIR>
sudo python ./tools/sysupdate/qm_manage.py set-rv-key <RV_KEY_FILE> -p <PORT>

6.2.2.2 Setting Revocation Key for a Subsequent Time

For subsequent revocation key updates over UART, you must specify the current revocation key. You must also specify the current firmware key after the firmware key has been changed:

sudo python ./tools/sysupdate/qm_manage.py set-rv-key <RV_KEY_FILE> --curr-rv-key <CURRENT_RV_KEY_FILE> --curr-fw-key <CURRENT_FW_KEY_FILE> -p <PORT>

6.2.2.3 Setting Firmware Key for the First Time

For first-time firmware key provisioning over UART, you must specify the port number and the current revocation key.

sudo python ./tools/sysupdate/qm_manage.py set-fw-key <FW_KEY_FILE> --curr-rv-key <CURRENT_RV_KEY_FILE> -p <PORT>

6.2.2.4 Setting Firmware Key for a Subsequent Time

For subsequent key updates over UART, you must also specify the current firmware key.

sudo python ./tools/sysupdate/qm_manage.py set-fw-key <FW_KEY_FILE> --curr-fw-key <CURRENT_FW_KEY_FILE> --curr-rv-key <CURRENT_RV_KEY_FILE> -p <PORT>

6.3 FM Functionality Usage

6.3.1 Creating and Flashing a QFU Image

This example shows how to use the Bootloader Firmware Management functionality to build and flash the LED blink example application to the Intel® Quark™ Developer Kit D2000 and Intel® Quark™ SE Microcontroller C1000 Developer Board. Intel recommends that you perform these actions in root/admin mode so that you can run commands with root/admin privilege.

Note: To give permission to access serial port, use the following command:
To create and flash the QFU image, follow these steps:

1. Ground the FM GPIO pin and reset the board.
2. While in QMSI directory, setup the software environment (as explained in Section 5.3, “Setting up the Software Environment”).
3. Build the project:

   ```
   make -C examples/blinky SOC=${TARGET_SOC} TARGET=${CORE_TYPE}
   ```

   **Note:** ${TARGET_SOC} values can be *quark_se* or *quark_d2000*.

   **Note:** ${CORE_TYPE} values can be *x86* for LMT core, or *sensor* for ARC core. The Intel® Quark™ Microcontroller D2000 supports LMT core only.

4. Create a secure DFU image:

   ```
   python $QM_BOOTLOADER_DIR/tools/sysupdate/qm_make_dfu.py --soc=${TARGET_SOC} -v examples/blinky/release/quark_se/x86/bin/blinky.bin --key <FW_KEY_FILE> -p 1
   ```

   **Note:** ${TARGET_SOC} values can be *quark_se* or *quark_d2000*. If not declared, *quark_se* is the default value.

   **Note:** The `-p` option is used to choose the flash partition. Partition 1 is used by the x86 core and partition 2 is used by the Sensor Subsystem. The Intel® Quark™ Microcontroller D2000 supports partition 1 only.

   **Note:** The `-v` option makes the tool output some information about the generated image.

   **Note:** Make sure *qmfmlib* library is installed.

   **Note:** For Windows*, replace `$QM_BOOTLOADER_DIR` with `%QM_BOOTLOADER_DIR%`.

5. If `-v` was added as a parameter, you get the following output:

   ```
   qm_make_dfu.py: QFU-Header and DFU-Suffix content:
   
   Partition:  1
   Vendor ID:  0
   ```
DFU over UART Guide

**Note:** To get a description of the QFU Image Creator parameters, run the following command:

```python
python qm_make_dfu.py --help
```

6. Download the DFU-image.

```bash
dfu-util-qda -D examples/blinky
/release/quark_se/x86/bin/blinky.bin.dfu -p <PORT> -R -a 1
```

**Note:** The `-a` option is used to choose the flash partition. Partition 1 is used by the x86 core and partition 2 is used by the Sensor Subsystem. The Intel® Quark™ Microcontroller D2000 supports partition 1 only.

**Note:** Ensure that no serial terminal is using the port while flashing the device. To check for a connected serial port on a Windows* system, open Device Manager. On a Linux* system, run `dmesg` in terminal to see the connected serial port. Once the serial port is identified, replace `<PORT>` with `/dev/ttyUSBx` or `COMXX`.

**Note:** For a description of DFU-UTIL–QDA parameters, run `dfu-util-qda --help`. For more information on `qm_make_dfu`, visit this GitHub* page.

**Note:** If DFU-download returns an error, redo the flashing step.

7. Unground the FM GPIO pin and press the reset button to run the application.
7.0  DFU over USB Guide

This chapter explains how to access FM mode and to upload a secured application image onto the target device through USB. Only signed images can be programmed when this feature is active.

This feature is available on the Intel® Quark™ SE Microcontroller C1000 only. FM is supported in both the Linux* and Windows* environments.

7.1  Hardware Setup

The hardware used in this guide is an Intel® Quark™ SE Microcontroller C1000 Developer Board. The setup consists of three main steps:

• Connect the board using the Secondary USB Port (USB DFU) to the computer.

• Connect the FM GPIO pin (AON_GPIO_4) as shown in the following figure. On the Intel® Quark™ SE Microcontroller C1000 Developer Board, this connection is J14-P43.

• Connect the power supply to the board via Primary USB port.

Figure 6.  Intel® Quark™ SE Microcontroller C1000 Developer Board Required USB Ports and Pins

The FM GPIO pin puts the device into FM mode. To enter FM mode, reset the device manually by pressing the reset button while the FM GPIO pin is grounded. To exit FM
mode, unground the FM GPIO pin and press the reset button. The device cannot exit FM mode while the FM GPIO pin is grounded.

Intel advises you not to use the FM GPIO pin in a FM-enabled ROM. Otherwise, the board is likely to reboot and enter FM mode unexpectedly.

### 7.2 Software Setup

#### 7.2.1 Enabling Firmware Management and Firmware Update with Authentication Functionality in the Bootloader

1. While in the QM-Bootloader directory, set up the software environment as explained in Section 5.3, “Setting up the Software Environment”.
2. Perform `mass erase` as explained in Chapter 10.0, “Performing Mass Erase”.
3. A second stage bootloader is required to use the USB DFU-UTIL. To enable the second stage bootloader, compile the ROM code:

```make
make clean
make SOC=quark_se ENABLE_FIRMWARE_MANAGER=2nd-stage
```

4. Flash the new ROM image to the target:

```python
python $QMSI_SRC_DIR/tools/jflash/jflash.py -r quarkse_dev ./build/release/quark_se/rom/quark_se_rom_fm_2nd_stage_hmac.bin
```

5. Compile the second stage bootloader. Firmware authentication feature will be enabled by default:

```make
make -C 2nd-stage
```

6. Flash the second stage bootloader to address 0x4005b000:

```python
python $QMSI_SRC_DIR/tools/jflash/jflash.py -u quarkse_dev ./2nd-stage/release/quark_se/x86/bin/2nd_stage_usb_hmac.bin
```

**Note:** Do not flash the board in a different order than the one explained in this guide. The "quark_se_rom_fm_2nd_stage_hmac.bin" must be flashed prior flashing the "2nd_stage_usb_hmac.bin" in step 4. If you flash in the wrong order, ground PIN J14-P28, reset the board and go back to step 4.

**Note:** To flash the created ROM in Windows system, use '/' to point directories.

**Note:** For Windows*, replace `$QMSI_SRC_DIR` with `%QMSI_SRC_DIR%`. 
7.2.2 Writing a New Key to Device

With both Firmware Management and Authentication enabled on the bootloader, you must set a new authentication key to the target device.

The firmware manager uses two keys, a firmware key and a revocation key. Both keys can be written using qm_manage.py. The revocation key must be written to the device before the firmware key. As a security precaution, firmware update cannot be performed until both keys have been correctly provided to the device. Once a key is set, it can be updated only if the old key is known.

To generate the `<RV/FW_KEY_FILE>` key-file, openssl can be used as an alternative to another cryptographic tool. However, you must make sure both keys are 32 bytes long and passed to qm-manage as a binary file. For example:

```
sudo openssl rand 32 > <KEY_FILE>
```

*Note:* It is good practice never to store RV and FW key file in the same place.

Before assigning a key to the device, the device has to be in the FM mode. To access FM mode, you must ground the FM-GPIO pin and power cycle the board.

7.2.2.1 Setting Revocation Key for the First Time

For first-time provision over USB, you must specify the new key and the Device ID (Product ID and Vendor ID) number:

```
cd <QM_BOOTLOADER_DIR>
sudo python ./tools/sysupdate/qm_manage.py set-rv-key <RV_KEY_FILE> -d <VID:PID>
```

7.2.2.2 Setting Revocation Key for a Subsequent Time

For subsequent revocation key updates over USB, you must specify the current revocation key. You must also specify the current firmware key after the firmware key has been changed:

```
sudo python ./tools/sysupdate/qm_manage.py set-rv-key <RV_KEY_FILE> --curr-rv-key <CURRENT_RV_KEY_FILE> --curr-fw-key <CURRENT_FW_KEY_FILE> -d <VID:PID>
```

7.2.2.3 Setting Firmware Key for the First Time

For first-time firmware key provisioning over USB, you must specify the Device ID (Product ID and Vendor ID) and the current revocation key:
7.2.2.4 Setting Firmware Key for a Subsequent Time

For subsequent key updates over USB, you must also specify the current firmware key.

```bash
sudo python ./tools/sysupdate/qm_manage.py set-fw-key
<FW_KEY_FILE> --curr-fw-key <CURRENT_FW_KEY_FILE> --curr-rv-key
<CURRENT_RV_KEY_FILE> -d <VID:PID>
```

7.3 FM Functionality Usage

7.3.1 Creating and Flashing a QFU Image

This example shows how to build and flash the `blinky` example to the Intel® Quark™ SE Development Platform using the Secondary USB Port, USB DFU. This example is illustrated in Figure 6. Intel recommends that you perform these actions in root/admin mode so that you can run commands with root/admin privilege.

To create and flash QFU image, follow these steps:

1. Reset the device while grounding the FM GPIO (J14-P43).
2. While in the QMSI directory, set up the software environment as explained in Section 5.3, “Setting up the Software Environment”.
3. Build the project as follows:

   ```bash
   make -C examples/blinky SOC=quark_se TARGET=x86
   ```

   **Note:** The default build application is for the LMT core. A complete Make command can specify the target core by adding `TARGET=x86` for the LMT core, or `TARGET=sensor` for the ARC core.

4. Create a secure DFU image:

   ```bash
   python $QM_BOOTLOADER_DIR/tools/sysupdate/qm_make_dfu.py --soc=<TARGET_SOC> -v
   examples/blinky/release/quark_se/x86/bin/blinky.bin --key
   <FW_KEY_FILE> -p 1
   ```

   **Note:** `<TARGET_SOC>` values can be `quark_se` or `quark_d2000`. `quark_se` is the default value if not declared.
Note: The -p option is used to choose the flash partition. Partition 1 is used by the x86 core and partition 2 is used by the Sensor Subsystem.

Note: The -v option makes the tool output some information about the generated image.

5. If you use the -v option, you get the following output:

```
qm_make_dfu.py: QFU-Header and DFU-Suffix content:
  Partition:   1
  Vendor ID:   0
  Product ID:  0
  Version:     0
  Block Size:  2048
  Blocks:      2
  DFU CRC:     0x8741e6e7
qm_make_dfu.py: blinky.dfu written
```

6. Flash the secure image to the target:

```
dfu-util -D
examples/blinky/release/quark_se/x86/bin/blinky.bin.dfu -d <VID:PID> -R -a 1
```

7. Once completed, unground the FM GPIO pin and press the reset button to run the application.

Note: Run dfu-util --help for more information on the command usage.

Note: The -a option is used to choose the flash partition. Partition 1 is used by the x86 core and partition 2 is used by the Sensor Subsystem.

7.4 Additional Notes and Known Bugs

7.4.1 Bootloader - DFU-UTIL Flashing Error

The error might occur occasionally when flashing an image with DFU-UTIL to an Intel® Quark™ SE Microcontroller C1000 Developer Board using USB with second stage bootloader. This error has no side effects, other than the need to re-execute the flash command.
8.0 Firmware Update with No Authentication

Firmware update with no authentication feature is available as follows:

Table 15. Firmware Update with No Authentication

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Supported Firmware Update Transport</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Quark™ Microcontroller D2000</td>
<td>UART</td>
</tr>
<tr>
<td>Intel® Quark™ SE Microcontroller C1000</td>
<td>UART, USB</td>
</tr>
</tbody>
</table>

8.1.1 Disabling Authentication Feature in Bootloader

To disable the secure boot option, the bootloader must be built with the ENABLE_FIRMWARE_MANAGER_AUTH=0 option. Export or set the environment as explained in Section 5.3, "Setting up the Software Environment", then enter the following commands.

In the case of firmware update over UART:

```
make SOC=<TARGET_SOC> ENABLE_FIRMWARE_MANAGER=uart
ENABLE_FIRMWARE_MANAGER_AUTH=0
```

**Note:** For firmware update over UART, `<TARGET_SOC>` value can be `quark_se` or `quark_d2000`

In the case of firmware update over USB:

```
make SOC=quark_se ENABLE_FIRMWARE_MANAGER=2nd-stage
make -C 2nd-stage ENABLE_FIRMWARE_MANAGER_AUTH=0
```

To flash the created ROM, see Section 6.2.1 for UART, and Section 7.2.1 for USB.

8.1.2 Simple Way (Using ‘make flash’ Command)

By disabling the authentication feature in the bootloader, you can use the Make-flash command to directly compile and flash an application into the target device. The steps are as follows:

1. Connect the board to the host machine. For DFU over USB, see Section 7.1. For DFU over UART, see Section 6.1.
2. Ground the FM GPIO pin and reset the board.
3. While in the QMSI directory, set up the software environment as explained in Section 5.3.
4. Compile and flash the targeted sample code with the Make-flash command, as follows:

   Over UART:

   ```
   make -C examples/blinky flash SOC=<TARGET_SOC>
   TARGET=<CORE_TYPE> SERIAL_PORT=<PORT>
   ```

   Over USB:

   ```
   make -C examples/blinky flash SOC=<TARGET_SOC>
   TARGET=<CORE_TYPE> USB_DEVICE=<VID:PID>
   ```

   **Note:** `<TARGET_SOC>` values can be `quark_se` or `quark_d2000`.

   **Note:** `<CORE_TYPE>` values can be `x86` for LMT core, or `sensor` for ARC core. The Intel® Quark™ Microcontroller D2000 supports LMT core only.

   **Note:** Ensure that no serial terminal is using the port while flashing the device over UART. To check for a connected serial port on a Windows* system, open Device Manager. On a Linux* system, run `dmesg` in terminal to see the connected serial port. Once the serial port is identified, replace `<PORT>` with `/dev/ttyUSBx` or `COMXX`.

   **Note:** To give permission to access the serial port, use the following command:

   ```
   sudo usermod -a -G dialout $USER
   ```

5. Unground the FM GPIO pin and press the reset button to run the application.

8.1.2.1 Step-by-Step Process (Not Using 'make flash')

For a step-by-step firmware update without authentication, see Section 6.3.1, “Creating and Flashing a QFU Image” (for UART), and Section 7.3.1, “Creating and Flashing a QFU Image” (for USB). The `--key <FW_KEY_FILE>` must be removed in the command while creating the DFU image.

§
9.0 Application Erase and System Information Retrieval

A Python script can retrieve system information located in the QM-Bootloader repository's tools/sysupdate directory. This script uses the DFU-UTIL/DFU-UTIL-QDA binary to communicate with the device.

9.1 Erase Applications

1. Enter device DFU mode by resetting the device while the FM GPIO pin is connected to ground.
2. Run the following commands to erase any application on both LMT and ARC (except bootloader):
   - Over UART:
     ```
     cd <QM_BOOTLOADER_DIR>/tools/sysupdate
     sudo python qm_manage.py erase -p <PORT>
     ```
   - Over USB:
     ```
     cd <QM_BOOTLOADER_DIR>/tools/sysupdate
     sudo python qm_manage.py erase -d <VID:PID>
     ```

   **Note:** This script can be used for application erase, but only if the device has a ROM with the authentication feature disabled.

9.2 System Information

1. Enter device DFU mode by resetting the device while the FM GPIO pin is connected to ground.
2. The system information retrieval command is as follows:
   - Over USB:
     ```
     cd <QM_BOOTLOADER_DIR>/tools/sysupdate
     sudo python qm_manage.py info -d <VID:PID>
     ```

   To check for Vendor ID (VID) and Product ID (PID), you can use `lsusb` in the Linux* terminal or check Device Manager in Windows* before executing the previous commands.
Over UART

cd <QM_BOOTLOADER_DIR>/tools/sysupdate

sudo python qm_manage.py info -p <PORT>

To check for a connected serial port on a Windows* system, open Device Manager. On a Linux* system, run dmesg in terminal to see the connected serial port. Once the serial port is identified, replace <PORT> with /dev/ttyUSBx or COMXX.

**Note:** Root/admin privilege is required to access the serial port.

**Note:** To display possible commands, run the python script `python qm_manage.py --help`. For more information on qm_manage, visit this GitHub* page.

**Note:** By specifying --format, you can set the output format to text or JSON.
Performing Mass Erase

10.0 Performing Mass Erase

To avoid any unexpected behavior, perform mass_erase before flashing a new FM-enabled bootloader into the target device. Performing mass_erase wipes the user application and device bootloader.

OpenOCD must be used to perform mass_erase. Optionally, you can use GDB or Telnet as a frontend for OpenOCD, as follows:

```
cd <PATH TO SOURCE>/ISSM_2016.2.xxx/tools/debugger/openocd
```

For the Intel® Quark™ Microcontroller D2000, start OpenOCD with the following command:

```
./bin/openocd -f scripts/board/quark_d2000_onboard.cfg
```

For the Intel® Quark™ SE Microcontroller C1000, start OpenOCD with the following command:

```
./bin/openocd -f scripts/board/quark_se_onboard.cfg
```

Open a new terminal/CMD session at this point, and launch a Telnet session:

```
telnet localhost 4444
```

Next, perform mass_erase to the target device, then exit the Telnet session:

```
> reset halt
> mass_erase
> exit
```
11.0 Installing DFU-UTIL-QDA in Windows*

Intel® provides a pre-built Windows binary that is available at this link. You can also build the binary using any of the following methods.

11.1 Windows* Native with MSYS

Windows binaries can be built in an MSYS2 environment, which provides a Linux*-like environment for Windows*. You can download the MSYS2 from this link.

Install MSYS2, then copy the QM-DFU-UTIL folder to <MSYS2_installation_folder>\home\<your_username>.

Open MSYS2 Shell and install the needed components:

```bash
pacman -S autoconf
pacman -S perl
pacman -S automake
pacman -S make
pacman -S mingw-w64-x86_64-gcc
PATH=$PATH:/mingw64/bin/
```

Before building the host tool, configure the build with the following:

```bash
cd <QM_DFU_UTIL_DIR>
./autogen.sh
./configure
```

Then compile it with the following command:

```
make
```

The generated binary is src/dfu-util-qda.exe.
11.2 Windows Cross-Compile from Ubuntu 16.04

Windows binaries can also be cross-compiled from Ubuntu. First, install MinGW and dh-autoreconf, as follows:

```
sudo apt-get install mingw-w64
dsudo apt-get install dh-autoreconf
```

Then configure the build with the following:

```
./autogen.sh
./configure --host=x86_64-w64-mingw32
```

Finally, compile it with the following command:

```
maker
```

The generated binary is `src/dfu-util-qda.exe`. §