You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting: http://www.intel.com/design/literature.htm

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at http://www.intel.com/ or from the OEM or retailer.

No computer system can be absolutely secure.

Intel, Quark, and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.
## Contents

1.0 **Introduction** .................................................................................................................. 6
  1.1 Document Overview ......................................................................................................... 6
  1.2 Product Overview ............................................................................................................. 6
  1.3 Terminology ..................................................................................................................... 6
  1.4 Reference Documents ..................................................................................................... 7

2.0 **Security Overview** ......................................................................................................... 8
  2.1 Assets ............................................................................................................................. 8
      2.1.1 OTP Memory ............................................................................................................ 8
      2.1.2 JTAG Interface ....................................................................................................... 9
      2.1.3 Flash Memory ......................................................................................................... 9
      2.1.4 SRAM .................................................................................................................... 9
      2.1.5 VR and Oscillator Registers .................................................................................. 10
  2.2 Trusted Computing Base ............................................................................................... 10

3.0 **Trusted Boot** ................................................................................................................ 12
  3.1 Boot Flow ...................................................................................................................... 12

4.0 **Secure Firmware Update** ............................................................................................ 15
  4.1 Overview ....................................................................................................................... 15
      4.1.1 Image Authentication Mechanism ....................................................................... 15
      4.1.2 Image Metadata .................................................................................................... 15
      4.1.3 Authentication Keys ............................................................................................. 16
      4.1.4 First-Time Provisioning ....................................................................................... 16
  4.2 Key Management ............................................................................................................ 16
      4.2.1 Key Assignment Scheme ....................................................................................... 16
      4.2.2 Key Generation Mechanism ................................................................................ 17
      4.2.3 Key Storage .......................................................................................................... 18
      4.2.4 Key Update Security ............................................................................................ 18
  4.3 Upgrade Images .............................................................................................................. 18

5.0 **Security Hardening** ...................................................................................................... 20
  5.1 Flash Write Protection .................................................................................................... 20
  5.2 Read-Protected Flash Regions ....................................................................................... 20
  5.3 Sensor Subsystem Code Protection Region ................................................................... 20
  5.4 Memory Violation Policy ............................................................................................... 21
  5.5 OTP Read Protection ..................................................................................................... 21
  5.6 OTP and JTAG Lock ....................................................................................................... 21
  5.7 Enforce Core Segregation ............................................................................................. 22
  5.8 Limit DMA Access .......................................................................................................... 28
  5.9 Stack Protection .............................................................................................................. 28
Figures

Figure 1. Intel® Quark™ SE Microcontroller C1000 TCB ................................................................. 11
Figure 2. Boot Flow ......................................................................................................................... 14
Figure 3. Default Configuration of Memory Core Segregation ....................................................... 22
Figure 4. Memory Segregation with DMA Access to the x86 Memory Space ................................. 24
Figure 5. Memory Segregation with DMA Access to the Sensor Subsystem Memory Space .......... 25
Figure 6. Default Configuration of Flash Core Segregation ............................................................ 26
Figure 7. Flash Segregation with DMA Access to the Flash x86 Space ........................................ 27
Figure 8. Flash Segregation with DMA Access to the Sensor Subsystem Flash Space ................. 28

Tables

Table 1. Terminology ..................................................................................................................... 6
Table 2. Reference Documents .................................................................................................... 7
## Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>April 2017</td>
<td>001</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>

§
1.0 Introduction

This document describes the security features provided by the bootloader of the Intel® Quark™ Microcontroller Software Interface (Intel® QMSI). Those features can be grouped into two main functionalities: authenticated firmware upgrades and SoC security hardening.

1.1 Document Overview

Chapter 2.0 of this document discusses the security assets and trusted computing base of an Intel® Quark™ microcontroller. Chapter 3.0 deals with the trusted boot process. Chapter 4.0 then deals with the secure firmware update, including an explanation of the authentication mechanism, key management, and the format of upgrade images. Finally, Chapter 5.0 deals with security hardening.

1.2 Product Overview

Intel® Quark™ microcontrollers are secure, low power, x86-based MCUs that are designed for deeply embedded applications. They are differentiated via a set of security capabilities that enable Intel's customers to build solutions with a security solution that is best in class for the targeted market.

1.3 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>BL-Data</td>
<td>Bootloader data</td>
</tr>
<tr>
<td>DoS</td>
<td>Denial of service</td>
</tr>
<tr>
<td>FPR</td>
<td>Flash Protection Region</td>
</tr>
<tr>
<td>GDT</td>
<td>Global Descriptor Table</td>
</tr>
<tr>
<td>IDT</td>
<td>Interrupt Descriptor Table</td>
</tr>
<tr>
<td>Intel® QMSI</td>
<td>Intel® Quark™ Microcontroller Software Interface</td>
</tr>
<tr>
<td>ISV</td>
<td>Independent software vendor</td>
</tr>
</tbody>
</table>
## Introduction

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IVT</td>
<td>Interrupt vector table</td>
</tr>
<tr>
<td>JTAG</td>
<td>Joint Test Action Group</td>
</tr>
<tr>
<td>MCU</td>
<td>Microcontroller unit</td>
</tr>
<tr>
<td>MPR</td>
<td>Memory Protection Regions</td>
</tr>
<tr>
<td>OEM</td>
<td>Original equipment manufacturer</td>
</tr>
<tr>
<td>OTP</td>
<td>One Time Programmable</td>
</tr>
<tr>
<td>SoC</td>
<td>System on chip</td>
</tr>
<tr>
<td>SRAM</td>
<td>Static random-access memory</td>
</tr>
<tr>
<td>SVN</td>
<td>Security version number</td>
</tr>
<tr>
<td>TCB</td>
<td>Trusted computing base</td>
</tr>
<tr>
<td>UART</td>
<td>Universal asynchronous receiver/transmitter</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>VR</td>
<td>Voltage regulator</td>
</tr>
</tbody>
</table>

### 1.4 Reference Documents

Table 2. Reference Documents

<table>
<thead>
<tr>
<th>Document</th>
<th>Document No./Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® Quark™ Microcontroller Software Interface Bootloader User Guide</td>
<td>334718</td>
</tr>
</tbody>
</table>

§
2.0 Security Overview

This section provides an outline of the various assets that the platform must protect. Reviewing this section will help architects and designers to identify other security requirements that must be added, or requirements that emerge due to future design or use case changes.

*Note:* All security features provided by the Intel® QMSI and its bootloader are just a reference. The ultimate responsibility for security lies with ISV/OEM.

2.1 Assets

Assets on the platform are typically considered in one of three high-level categories:

- **Secrets**
  
  Platform-specific or global secret values (for example, keys) that are used for encryption and integrity protection of sensitive user data, content protection, authentication, or other security objectives.

- **Execution Integrity**

  For various subsystems (host Root of Trust and remaining firmware stack, Sensor Subsystem firmware), assets in this category refer to avoiding the introduction of malware into the platform, or other unexpected external manipulation of the execution environment.

- **Intermediate assets**

  Platform resources that have access restrictions to protect the true assets on the platform, the secrets, and execution integrity. The intermediate assets are considered to define rules for their access, to build the framework for protection of the true assets.

2.1.1 OTP Memory

Intel® Quark™ microcontrollers feature an on-die OTP-capable flash where the x86 processor core reset vector is located. This is the first code execution stage of the system boot flow and effectively represents a hardware root of trust. The bootstrap code is therefore meant to be located in the OTP flash.

*Note:* The OTP-capable flash can be hardware locked to become an OTP. The OTP lock must be enabled before the device is deployed, otherwise the security mechanism described in this document becomes void.
2.1.2 JTAG Interface

The JTAG interface is an entry point to the trusted computing base discussed in Section 2.2, “Trusted Computing Base”. When the OTP lock is enabled, JTAG is disabled, which means that the ROM code cannot be modified.

2.1.3 Flash Memory

Flash memory contains the following information that must be protected:
- BL-Data (which is where authentication keys are stored)
- Application firmware

BL-Data must be protected against both reading and writing, to avoid keys being replaced or leaked by malware. As discussed in Section 5.0, “Security Hardening”, the bootloader sets up an FPR to protect BL-Data from reading and disables flash writing to protect it from writing.

In general, application firmware must be protected against writing to prevent attackers from changing it (for example, by exploiting some application bug that allows for arbitrary code execution). However, a protection against reading may be required as well, since application firmware may contain private information (for example, proprietary intellectual property).

As discussed later in this document, the bootloader disables flash writing on the entire flash by default, thus protecting application firmware from modification. However, the precise protection against read access is left to the user.

2.1.4 SRAM

Some assets are also stored in SRAM. Specifically, the x86 portion of RAM contains the x86 stack, the x86 Global Descriptor Table (GDT), and the x86 Interrupt Descriptor Table (IDT). The sensor subsystem RAM contains the ARC stack, and the ARC interrupt vector table (IVT).

The IVT on an ARC architecture includes the main reset vector location. The Sensor Subsystem has a default IVT location that it fetches from reset. However, the IVT base address can be relocated by the ARC processor in kernel mode.

Any change to the reset vector or the location of that IVT would enable an attacker to effectively control the Sensor Subsystem boot sequence, leading to the opportunity to bypass security measures put in place during the later boot stages.

In the reference boot flow, the Sensor Subsystem is meant to be started by the x86 application, which sets the ARC reset vector in the IVT, protects the IVT with an MPR, and then activates the ARC core. However, as discussed in Section 5.7, “Enforce Core
Segregation*, a more secure (but less flexible) behavior is to have OTP code (that is, the bootloader) start the ARC.

Some application scenarios may also allow for a complete segregation between the ARC and the x86 core (each core should not be able to access RAM and flash memory from the other code). These scenarios are discussed in Section 5.7, “Enforce Core Segregation”.

2.1.5 VR and Oscillator Registers

The SoC provides a set of memory-mapped registers for configuring the integrated voltage regulators (VRs) and oscillators. These determine voltage supply to the rest of the SoC and the speed at which the SoC operates. The configuration can be locked through a specific register.

Tampering with the configuration of either the VRs or the oscillators may lead to permanent denial of service. However, the reference bootloader does not lock the configuration as different application scenarios require different configurations. Intel recommends that application developers change the bootloader to make it set up and lock the configuration they need. Specifically, Intel recommends that this change is added to the routine that sets up the application security context (for more details, see the boot flow in Chapter 3.0, “Trusted Boot”).

2.2 Trusted Computing Base

The Trusted Computing Base (TCB) denotes the set of components that must be trusted for the overall platform and application to be secure.

For both the Intel® Quark™ Microcontroller D2000 and the Intel® Quark™ SE Microcontroller C1000, the TCB includes the Lakemont (LMT) core, the on-die SRAM, and the on-die flash. For the Intel® Quark™ SE Microcontroller C1000, the TCB also includes the Sensor Subsystem (including the pattern matching engine).
Figure 1. Intel® Quark™ SE Microcontroller C1000 TCB
3.0 Trusted Boot

This chapter describes the ROM/OTP boot stage, which is part of the TCB and would be immutable in production (once the OTP/JTAG is locked).

The current boot flow supports the secure firmware upgrade feature described in Chapter 4.0 and sets up the security hardening described in Chapter 5.0.

The ISV/OEM can extend the boot flow to improve or customize the security hardening or to build other security schemes, such as secure boot, on top of it.

3.1 Boot Flow

The bootloader flow begins with the initialization of the x86 core, which includes loading the Global Descriptor Table (GDT) and entering 32-bit protected mode.

Then, on the Intel® Quark™ SE Microcontroller C1000, the bootstrap code checks if the SoC is returning from sleep. If so, the application security context is set up and the x86 application execution resumes. The application security context consists of the following:

- Enabling flash write protection
- Setting up an FPR to read-protect BL-Data (so that no agent can modify it)
- Setting up an MPR to read/write-protect the GDT and IDT of the x86 core (which becomes the only agent allowed to modify it)

If, instead, the SoC is not resuming from sleep, the bootloader continues the normal boot process by initializing the RAM (clearing .bss and loading .data) and setting up primary peripherals (power and clock configuration).

Next, the bootloader checks the status of the JTAG_PROBE_PIN. If the pin is grounded, the bootloader simply waits until it is ungrounded. This procedure is used to unbrick a device with firmware that prevents JTAG from working correctly. This step becomes useless in production mode, when JTAG access is disabled by locking the OTP.

The next step is to check, and optionally sanitize, BL-data:

- If this is the first boot (and therefore BL-Data is blank), the bootloader initializes its persistent metadata by creating two identical copies of BL-Data in flash.
- If this is a subsequent boot, the bootloader verifies the integrity of the two copies of BL-Data:
  - If one of the two copies is corrupted, the copy is restored using the content of the uncorrupted copy.
– If both copies are corrupted, the bootloader enters a faulty state consisting of an infinite loop (since this situation can happen only if there is a hardware fault or a security attack).

Then, the bootloader initializes the Interrupt Description Table (IDT) and enables interrupts.

Next the bootloader sets up the default memory violation policy, which consists of triggering a warm reset.

Then, the bootloader checks if Firmware Management (FM) mode is requested—that is, the FM pin is grounded, or the FM sticky bit of General Purpose Sticky register 0 (GPS0) is set. If so, the bootloader sets up FM security context and enters FM mode. The FM security context setup consists of setting up an MPR and an FPR to restrict RAM and Flash access to the x86 core only.

If FM mode is not requested, the bootloader checks if the x86 application is present by checking if the first double word is different from 0xFFFFFFFF. If the application is present, the bootloader sets up the application security context and jumps to the application. If the x86 application returns, the x86 core enters an infinite loop, while the state of the sensor subsystem is not modified.

If no application is present, the bootloader enters FM mode (after setting up the FM security context described earlier).
Figure 2. Boot Flow

- **Reset begin**: Initialize x86 core
  - **Quark SE C2000 only**: Restore from sleep?
    - Yes: Set-up application security context → Restore app
    - No: Enter infinite loop
  - **Valid x86 Image?**
    - Yes: Boot App
    - No: Enter infinite loop

- **Setup SRAM and primary peripherals**
  - **Enter recovery?**
    - Yes: JTAG wait
    - No: Check and optionally sanitize BL-Data
      - Initialize IDT and enable interrupts
        - Set memory violation policy to warm reset
          - **Enter FM?**
            - Yes: Clear FM_GPS_BIT
            - No: **Valid x86 Image?**
              - Yes: Set-up application security context → Boot App
              - No: Set-up FM security context

- **FM_GPIO or FM_GPS_BIT asserted**
  - Enter infinite loop

- **JTAG_PROBE_PIN asserted**
  - Enter infinite loop

- **QM_GPS0_BIT_X86_WAKEUP**
  - Enter infinite loop
4.0 Secure Firmware Update

The bootloader provides a Firmware Management (FM) feature that allows application firmware to be updated via UART or USB (see the firmware manager user guide for more information).

The FM feature can be compiled with authentication support (enabled by default). When authentication is enabled, firmware upgrades can be done only using signed images: the Firmware Manager rejects any image that is unsigned or that is signed with the wrong firmware authentication key. Also, when authentication is enabled, the bootloader also provides a mechanism for setting and updating authentication keys.

This section describes both the authenticated firmware upgrade feature and the key update functionality, highlighting those aspects that have security implications.

4.1 Overview

4.1.1 Image Authentication Mechanism

Images are signed using a 256-bit hash-based message authentication code (HMAC-SHA-256). This is a symmetric-key algorithm that generates an SHA-256 keyed hash by combining the image with an authentication key that is shared between the device and the host.

Since on-DIE flash is part of the SoC TCB and usually write protected, the image is authenticated during firmware upgrade but not at every boot. Note also that images are not encrypted.

The ISV/OEM must generate authentication keys and set them to the devices before deployment. The ISV/OEM must also store the keys somewhere, since the keys are required for signing upgrade images. The ISV/OEM is responsible for the security of the key storage system. For more information about authentication keys, see Section 4.1.3, “Authentication Keys”.

4.1.2 Image Metadata

Each image has an associated security version number (SVN) that the image creator must specify. A device can be updated only with an image that has an SVN greater than the SVN of the currently installed image, or, if the application firmware has been deleted, the previously installed image.
4.1.3 Authentication Keys

The firmware manager uses two kinds of key: the firmware key and the revocation key. The firmware key authenticates both firmware images and key updates. The revocation key authenticates key updates in combination with a firmware key (in other words, key updates are double signed, using both the firmware key and the revocation key).

Both keys must be 32 bytes long.

Both keys can be updated. During either a firmware key update or a revocation key update, the new key is signed using both current keys (that is, the keys currently installed in the device). The purpose of the revocation key is to provide a recovery option using a key that can be stored offline, as it is not required during usual operation. The reason for authenticating key updates with both keys is to reduce the risk of an attacker compromising either the revocation or firmware key (single point of failure).

Note: Key updates are authenticated, but not encrypted. This means that a trusted agent must perform updates using a secure channel, as discussed in Section 4.2.4, “Key Update Security”.

First-time provisioning is a special case of the key update process, and is discussed in the following section.

4.1.4 First-Time Provisioning

Authentication keys must be set before a device is deployed. The provisioning mechanism consists of first setting the revocation key, then setting the firmware key. The firmware key cannot be set until the revocation key is set.

Since the device is unprovisioned, a default key is used to sign the key updates, in the absence of revocation and firmware keys.

Note: The default key is publicly known and used purely for convenience of initial provisioning and to minimize implementation footprint on the device side. Never deploy unprovisioned devices, since anybody may use the default key to set the devices’ authentication keys and thus take control of them.

To enforce key-provisioning, the firmware upgrade functionality is disabled until both keys are set.

4.2 Key Management

4.2.1 Key Assignment Scheme

There are multiple possibilities for key assignment schemes.
One possibility is to assign each device its own firmware key and revocation key. This is the best solution in terms of security. However, it may be impractical as an upgrade to a class of devices would require a different upgrade image for each device (that is, the same image signed with each device firmware key). In this scheme, the vendor must have a way to match each device to its firmware key and revocation key.

Another possibility is to assign one firmware key to a class of devices with different revocation keys. This would allow a class of devices to be upgraded using a single upgrade image. However, this also means that if the common firmware key is leaked, the firmware on all devices that use this key can be replaced with malware. Key update requests must still be signed with the device-specific revocation key, which prevents attackers from taking full control of the devices by changing their keys. In this scheme, the vendor must have a way to match each device to its revocation key only.

Yet another possibility is to assign one firmware key and one revocation key to a class of devices. This is similar to the above mentioned scheme. It allows a class of devices to be upgraded using a single upgrade image, and also allows keys to be updated across a class of devices using a single key update request. However, as in the previous case, this also means that if the common firmware key is leaked, the firmware on all devices that use this key can be replaced with malware. Also, unlike the previous case, if the common revocation key is leaked along with the common firmware key, the keys on all devices in the class can be changed and an attacker can take complete control of all devices in this class.

Other schemes (for example, the same firmware key across different classes of devices) are not recommended, as they increase security risks without adding any real value.

In the rest of this document, the second scheme (single firmware key per device class and different revocation key for each device) is taken as a reference, since it provides a reasonable compromise between flexibility (upgrade images that work on all the devices of a specific class) and security (reducing the risk of having authentication keys of the entire class compromised).

ISV/OEMs are responsible for choosing the scheme that best suits their application scenarios and requirements.

4.2.2 Key Generation Mechanism

As discussed in the previous section, the reference key assignment scheme requires the ISV/OEM to generate a revocation key for each device and maintain the mapping between each device and its revocation keys.

A typical approach to achieve this is to use a master-slave keying scheme, in which each group or class of devices is associated with a master revocation key (RVm). The master revocation key is never disclosed, but used only to generate individual slave keys that are distributed to the devices. Slave revocation keys (RV) can be generated as follows:

\[ RV = \text{HMAC}(RV_m, \text{device-id}) \]
4.2.3 Key Storage

Choose the revocation key and the firmware key independently from a true random source. Store the keys separately to reduce the likelihood of both keys being leaked. The leak of both keys would result in the system being completely compromised (that is, an attacker would be able to change both keys and take full control of the device.)

**Note:** When only one key is leaked, the leaking of the firmware key is more critical than the leaking of the revocation key. An attacker can use the leaked firmware key to update devices with malicious firmware, while the revocation key alone does not allow the attacker to update the keys.

4.2.4 Key Update Security

Key updates are not encrypted, and therefore keys can be exposed during key updates.

Updates over USB and UART are safe if the communication link is secure. For example, updates over long serial cables can pose a security risk because eavesdropping may be possible.

Remote updates (for example, via gateways wired to the devices) also pose a security risk because of the possibility of eavesdropping. If ISVs/OEMs want to implement secure remote upgrade functionality, they are responsible for fully assessing the security implication of their extension to the current solution.

Finally, if ISVs/OEMs do not trust their end users, delegating key updates to them is generally not secure. Users may try to sniff the unencrypted keys during the update.

4.3 Upgrade Images

Upgrade firmware images use the Intel® Quark™ microcontroller image format. The image is divided into equal sized blocks. The first block contains the header and following blocks contain the raw firmware image.

The header contains common information for processing the image, such as the vendor ID, product ID, and the partition number (which identifies where the image is meant to be flashed). The base header can be followed by an extended header, which contains information for image verification and authentication.

The extended header contains a security version number (SVN), an SHA256 hash of each image block and a 256-bit hash-based message authentication code (HMAC-SHA-256). The SVN is discussed in Section 4.1.2, “Image Metadata”. The hashes verify the firmware image. The HMAC is calculated using the firmware key and the hash of the entire header, including the hashes of each image block.

**Note:** Upgrade images are unencrypted. If they are made public, intellectual property (IP) may be exposed.
It is important that the SVN is updated when the image fixes a security bug present in the previous version of the application. Failure to update the SVN number allows attackers to roll back the previous firmware to exploit its security issues.
5.0 Security Hardening

As discussed briefly in Chapter 3.0, “Trusted Boot”, the bootloader sets up some HW security features to harden the security of both the Firmware Management (FM) feature and the application code.

This chapter describes in more detail the protection put in place by the bootloader. It also provides some suggestions on how application developers can improve the protection, based on their application use case.

5.1 Flash Write Protection

Flash writes are disabled for the entire flash.

This disabling prevents bootloader data, which includes authentication keys, from being overwritten. It also ensures that only the firmware manager can modify the flash content. Even if there is a bug in an application that allows an attacker to run arbitrary code, that code cannot be installed permanently.

However, this protection limits the application's ability to use the flash. For example, no data logging is possible. Therefore, a compilation flag is provided to disabled it (specifically, customers must recompile the bootloader with ENABLE_FLASH_WRITE_PROTECTION=0).

However, disabling this protection poses great security risks because it allows malware to change the keys stored in bootloader data. Also, attackers may conceive advanced attack strategies that exploit the write capability to guess the keys, thus leading to a key leakage that may compromise an entire class of devices (depending on the key assignment scheme used).

5.2 Read-Protected Flash Regions

The bootloader disables read access to the bootloader data (BL-Data) region of flash. This is done by using FPRO and granting read access to no agents. Application code should not try to access BL-Data flash region as attempting to do so triggers a memory violation event, resulting in a warm reset.

5.3 Sensor Subsystem Code Protection Region

The Sensor Subsystem exposes a register (SS_CFG.PROT_RANGE) that disables any ARC load or store operation to the specified SRAM region. During the boot process, this register is set and locked to an ineffective state, to prevent malware using it as a DoS vector against ARC.
The locking is done using register CFG_LOCK.PROT_RANGE_LOCK.

5.4 Memory Violation Policy

The bootloader sets the memory violation policy for both flash and RAM to trigger a warm reset. This is done by unmasking halt interrupts for the SRAM and flash controllers and redirecting halt interrupts to trigger a reset.

For complete security, lock the masking configuration for halt interrupts, to prevent malware from masking them to disable the warm reset. Since the lock applies to the halt interrupt configuration of every peripheral, this is not done by the bootloader because doing so prevents applications from unmasking other halt interrupts that they may need.

Therefore, application developers are responsible for enabling the lock. They should change the bootloader code to unmask all the halt interrupts they need, then lock the configuration in there. The configuration can be locked by setting bit LOCK_HOST_HALT_MASK of the LOCK_INT_MASK_REG register of the System Control Subsystem. Alternatively, if changing the bootloader is unpractical, application developers can set the lock during the application initialization.

5.5 OTP Read Protection

The bootloader does not enable read protection of OTP flash since the OTP region of flash does not contain any private information.

If an application developer decides to store private information in OTP, they must read protect the OTP in the bootloader before jumping to the application, or when the x86 application starts. The read protection can be done by setting the ROM_RD_DIS_U and ROM_RD_DIS_L bits of the CTRL register of flash controller 0.

ROM_RD_DIS_L protects the lower 4kB of OTP while ROM_RD_DIS_U protects the upper 4kB. If the protection is enabled in the bootloader, the developer can protect only half of the OTP. They must ensure that all code to be run after the protection setup is located in the unprotected half of the OTP, or a memory violation will be triggered.

5.6 OTP and JTAG Lock

The OTP lock must be enabled before the device is distributed or deployed. This ensures that JTAG is disabled and ROM code cannot be modified.
5.7 Enforce Core Segregation

On the Intel® Quark™ SE Microcontroller C1000, the bootloader code and the ARC startup code ensure some basic segregation between the x86 and the ARC applications. The x86 core is the only one that can access or modify the x86 GDT and IDT in SRAM, while the ARC core is the only one that can access the ARC IVT in SRAM. The bootloader sets up the x86 GDT/IDT protection using MPR0, while the ARC activation code sets up the ARC IVT protection using MPR1.

*Figure 3. Default Configuration of Memory Core Segregation*

*Figure 3* illustrates the default configuration of the memory core segregation. Only ARC can access the IVT and only Lakemont can access the GDT and IDT. The rest of the memory is unprotected and all agents can access it.

If the application use case allows (for example, no shared memory is required between the two cores), Intel recommends that application developers improve the default core-segregation policy by completely separating the memory (both flash and SRAM) of the two cores. This ensures that data is not leaked from one core to the other, and that a compromised core cannot be used to attack the other core.
This can be done by setting up two extra MPRs and two extra FPRs, as follows:

- Set up MPR2 to protect the x86 memory space. Only the x86 core and, if needed, DMA/USB should have access to this memory region.

**Note:** If DMA/USB access is not required, MPR0 can be reused by extending it to cover the entire x86 RAM region.

- Set up MPR3 to protect the ARC memory space. Only the ARC core and, if needed, DMA/USB should have access to this memory region.

**Note:** If DMA/USB access is not needed, MPR1 can be reused by extending it to cover the entire ARC RAM region.

- Set up FPR0 on Flash controller 1 to protect the x86 flash space. Only the x86 core and, if needed, DMA/USB should have read access to this memory region.

- Set up FPR1 on Flash controller 0 to protect the ARC flash space. Only the Sensor subsystem core and, if needed, DMA/USB should have read access to this memory region.

Allow DMA on only one of the two cores. Otherwise, it could be abused to transfer memory from one core's memory space to the second core's memory space.
Figure 4. Memory Segregation with DMA Access to the x86 Memory Space
Figure 5. Memory Segregation with DMA Access to the Sensor Subsystem Memory Space

On-die SRAM

- IVT
- Sensor Stack
- Sensor subsystem memory space
- GDT
- IDT
- x86 memory space

MPR1
ARC only (R/W)

MPR2
ARC + DMA (R/W)

MPR0
x86 only (R/W)

x86

SENSE (ARC)

DMA

Intel® Quark™ Microcontroller Software Interface –
Bootloader Security Features
Programmer's Guide

April 2017
Document Number: 335728-001EN
Figure 6. Default Configuration of Flash Core Segregation

- SENSOR (ARC)
- DMA
- FLASH 0 (write-protected) — Sensor subsystem flash space
- BL-DATA
- FLASH 1 (write-protected) — x86 flash space
- FPR0 — No access
- BL-DATA
- x86
Figure 7. Flash Segregation with DMA Access to the Flash x86 Space
5.8 Limit DMA Access

As previously discussed, use FPR and MPR to limit DMA access to those parts of flash and SRAM that are necessary. This limits the amount of damage an attacker can cause by exploiting DMA-related bugs.

5.9 Stack Protection

To improve security, limit the access of each processor stack to the processor itself (that is, DMA, USB, or the other core should not have access to it).

To ensure out-of-the-box compatibility with Zephyr 1.7, this limitation is only partially done by the bootloader. Specifically, MPR0, which protects the x86 GDT/IDT, only protects part of the x86 stack.

Covering the full x86 stack requires you to align the stack to the MPR granularity (that is, 1kB-alignment) to prevent the MPR protecting the stack from also protecting part of the RAM used for statically allocated variables (that is, the .bss and .data section).
If Zephyr 1.7 is not used, Intel recommends that application developers change the linker script and the bootloader code to extend the MPR to cover the full x86 stack.

The ARC stack should also be protected. This can be done by moving the stack close to the ARC IVT and extending MPR1, which protects it.