



Intel[®] Quark[™] SoC X1000

Board Support Package (BSP)

Build and Software User Guide

Release: 1.2.1

February 2016



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 © 2016, Intel Corporation. All rights reserved.



Contents

1	About This Document	6
Part 1 of 2 - Building the BSP Software		7
2	Before You Begin	8
3	Downloading Software	9
4	Building the EDKII Firmware.....	10
4.1	Dependencies	10
4.2	Pre-build Setup.....	10
4.2.1	Performing pre-build Steps in a Linux/gcc Build Environment	11
4.2.2	Performing pre-build Steps in a Windows Build Environment	11
4.3	Building all the EDKII Firmware Validated Build Configurations [Linux build environment only]	12
4.4	Building a Single EDKII Firmware Build Configuration	13
4.5	EDKII Firmware Build Standalone Output Files.....	14
5	Building the GRUB OS Loader [Linux* Build Environment Only].....	16
6	Creating a File System and Building the Kernel Using Yocto Project*	18
6.1	Build a Full-featured Linux for SD card, USB stick or eMMC.....	19
6.2	Build a Small Linux for SPI Flash	20
6.3	Build a fast boot Linux for SD card or eMMC	22
6.4	Applying a Custom Patch to the Linux kernel Using Yocto Project (optional) .	24
7	Building the Linux* Cross Compile Toolchain Using Yocto Project* [Linux Build Environment Only]	25
8	Creating a Flash Image for the Board [Linux* build environment only].....	28
8.1	Using the SPI Flash Tools	28
9	Platform Data Tool	30
10	Programming Flash on the Board Using Serial Interface	33
10.1	Programming Flash to Release 1.2.1	33
10.1.1	Requirements to Update to Release 1.2.1	33
10.1.2	Steps to Update to Release 1.2.1.....	34
10.2	Programming flash using UEFI shell	34
10.3	Programming Flash Using Linux* Run-time System	37
11	Programming Flash on the Board Using DediProg.....	39
12	Booting the Board From SD Card.....	40
Part 2 of 2 - Using the BSP Software		42
13	Capsule Update	43
14	Capsule Recovery	44
15	Signing Files (Secure SKU only) [Linux* build environment only]	45
15.1	To Program Secure SPI Flash	45
15.2	To Create a Secure Boot SD/eMMC	46



16	Enabling the OpenOCD Debugger	48
Appendix A	Related Documents	49
Appendix B	SPI Flash Tools	50

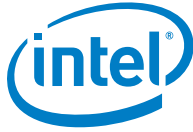
Figures

Figure 1.	Updating Process from Release 1.0.x to Release 1.2.1	34
-----------	--	----



Revision History

Date	Revision	Description
February 2016	010	Updated for Fast boot. Updated Chapters 6,9 , and 15
October 2015	009	Updated Sections 4, 6, 7, 10, 13, and 14 .
January 2015	008	Updated Sections 4 and 9 Added Appendix B
16 June 2014	007	General updates to coincide with the EDKII Update 1.0.2 release including: Updated Section 4, Building the EDKII Firmware (added Windows* build environment).
22 May 2014	006	General updates for software release 1.0.1 including: Updated Section 4, Building the EDKII Firmware (added TPM). Added Section 6.1 Build a Full-featured Linux for SD card or USB Stick Updated Section 9 Platform Data Tool (corrected platform-data.ini filename). Updated Section 14, Capsule Recovery (added DediProg information). Updated with trademarked term: Intel® Quark™ SoC.
04 March 2014	005	General updates for software release 1.0.0 including: Added Section 13 Capsule Update . Added Section 14, Capsule Recovery .
20 January 2014	004	General updates for software release 0.9.0 including: Added Section 4, Building the EDKII Firmware . Added Section 10.3, Programming Flash Using Linux* Run-time System . Updated Section 15, Signing Files (Secure SKU only) . Removed OpenOCD details because patch is now open source. Added Appendix A Related Documents .
15 November 2013	003	Added CapsuleApp.efi to Section 3, Downloading Software .
07 November 2013	002	General updates for software release 0.8.0 including: Added supported boards to list of hardware. Section 8 : Changed SPI Flash tools path from clanton_peak_EDK2 to Quark_EDKII Moved Signing Files (Secure SKU only) section to later in the document.
15 October 2013	001	First release with software version 0.7.5.



1 *About This Document*

This document, the Intel® Quark™ SoC X1000 Board Support Package (BSP) Build and Software User Guide, is divided into two major sections:

- **Part 1 of 2 - Building the BSP Software** contains instructions for installing and configuring the Intel® Quark™ SoC X1000 Board Support Package sources.
- **Part 2 of 2 - Using the BSP Software** provides information on BSP software features and functionality.

Use this document to create an image to boot on your Quark-based board, and to learn more about BSP software features.

The intended audience for this document are hardware/software engineers with experience in developing embedded applications.

This software release supports the following software and hardware:

- Board Support Package Sources for Intel® Quark™ SoC X1000 v1.2.1
- Intel® Galileo Customer Reference Board (CRB) (Fab D with blue PCB)
- Intel® Galileo Gen 2 Customer Reference Board (CRB)
- Intel® Quark™ SoC X1000 Industrial/Energy Reference Design (Cross Hill)
- Intel® Quark™ SoC X1000 Transportation Reference Design (Clanton Hill)
- Intel® Quark™ SoC X1000 Air Quality Management Solution Reference Design (Reliance Creek)



Part 1 of 2 - Building the BSP Software

This section contains the following chapters:

2. [Before You Begin](#)
3. [Downloading Software](#)
4. [Building the EDKII Firmware](#)
5. [Building the GRUB OS Loader \[Linux* Build Environment Only\]](#)
6. [Creating a File System and Building the Kernel Using Yocto Project*](#)
7. [Building the Linux* Cross Compile Toolchain Using Yocto Project* \[Linux Build Environment Only\]](#)
8. [Creating a Flash Image for the Board \[Linux* build environment only\]](#)
9. [Platform Data Tool](#)
10. [Programming Flash on the Board Using Serial Interface](#)
11. [Programming Flash on the Board Using DediProg](#)
12. [Booting the Board From SD Card](#)



2 Before You Begin

Before you begin:

- You need a host PC running either:
 - Linux*; Intel recommends a 64-bit Linux system
 - Microsoft* Windows* 7, x64
- You need an internet connection to download third party sources.
- The build process may require as much as 30 GB of free disk space.
- To program the board you can use:
 - A serial interface using the UEFI shell or Linux* run-time (see [Section 10](#))
 - A DediProg* SF100 SPI Flash Programmer (or equivalent) and the associated flashing software (see [Section 11](#))
 - An Intel® Galileo IDE (Intel® Galileo Gen 2 board only; refer to [Appendix A Related Documents](#) for User Guide details)

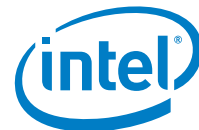
Note: Remove all previous versions of the software before installing the current version.

Individual components require very different environments (compiler options and others). **To avoid cross-pollution, the commands in each section that follows must be run in a new command line window every time.**

Note: If the commands fail or timeout, it may be due to your proxy settings. Contact your network administrator. You may find answers here:
https://wiki.yoctoproject.org/wiki/Working_Behind_a_Network_Proxy

This release has been tested with Windows* 7, 64-bit and Debian* Linux* 7.0 (Wheezy), but will work with most other Linux distributions.

Linux builds have been validated on 64-bit Linux systems and may need additional steps for operation on 32-bit systems.



3 Downloading Software

Download the BSP sources zip file from the following location:

<https://downloadcenter.intel.com/download/23197/Intel-Quark-BSP>

Note: If you are using an Intel® Quark™ Reference Design board, see your Intel representative for the appropriate software download URL.

This release is comprised of:

- Board Support Package (BSP) sources:
 - Board_Support_Package_Sources_for_Intel_Quark_v1.2.1.7z

For customers using the Clanton Hill FFRD, additional CAN software must be downloaded from Intel Business Link (IBL). See your Intel representative for the URL. The CAN package comprises:

- Fujitsu CAN Firmware:
CAN_Firmware_for_Intel_Quark_v1.2.1.zip (36 kB)

If building on a Debian host PC, use the Debian-provided meta package called `build-essential` that installs a number of compiler tools and libraries. Install the meta package and the other packages listed in the command below before continuing:

```
# sudo apt-get install build-essential gcc-multilib vim-common
```



4 Building the EDKII Firmware

You need to build the open source EDKII firmware for the Intel® Quark™ SoC. Additional details may be found here:

- www.tianocore.sourceforge.net
- <https://github.com/tianocore/tianocore.github.io/wiki/Getting-Started-with-EDK-II>

4.1 Dependencies

Linux* build environment dependencies:

- Python 2.6 or 2.7 (Python 3.x not supported)
- GCC and G++ (tested with GCC 4.3 and GCC 4.6)
- subversion client
- uuid-dev
- nasm
- iasl (<https://www.acpica.org/downloads/linux>)
Note: An ACPI5.0 compatible version is required.

Windows* build environment dependencies:

- Python 2.6 or 2.7 (Python 3.x not supported)
- Microsoft* Visual Studio* 2008 Professional.
- The Intel® Quark™ SoC EDKII build is validated with the **Win7 x64 / VS2008x86** option shown in:
<https://github.com/tianocore/tianocore.github.io/wiki/Windows-systems-ToolChain-Matrix>

In addition, the `quarkbuild.bat` below enforces the x86 postfix onto the Visual Studio* option for building under x64 Windows.

- TortoiseSVN (1.4.2.8580 or later) installed with optional SVN command line tools.
- iASL Windows binaries (<https://www.acpica.org/downloads/binary-tools>)
Note: An ACPI5.0 compatible version is required and they should be extracted to a C:\ASL directory.

4.2 Pre-build Setup

The following steps are performed one time to prepare the EDKII workspace directory with the required source code before commencing the actual firmware build.

1. Create the EDKII workspace directory and extract the contents of the Intel® Quark™ SoC EDKII BSP into this directory.



The file will have the name `Quark_EDKII_<version>.tar.gz`. After the contents have been extracted the files `quarkbuild.sh` and `quarkbuild.bat` should be in the root of the created workspace directory.

2. Fetch the upstream core EDKII packages using `svn_setup.py` and the SVN command line tool.
3. Optionally, if OpenSSL is required by the build configuration in [Section 4.3](#) or [Section 4.4](#) following, then perform the steps in the `CryptoPkg/Library/OpensslLib/Patch-HOWTO.txt` file.

4.2.1 Performing pre-build Steps in a Linux/gcc Build Environment

Open a new terminal session and enter the following commands:

```
# sudo apt-get install build-essential uuid-dev iasl subversion nasm
# tar -xvf Quark_EDKII_*.tar.gz
# cd Quark_EDKII*
# ./svn_setup.py
# svn update
```

4.2.2 Performing pre-build Steps in a Windows Build Environment

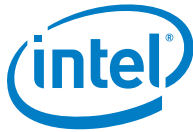
Use a preferred tool to extract the `Quark_EDKII_*.tar.gz` to a user created EDKII Workspace directory and run the `cmd.exe` Windows command. Then, issue the following commands:

```
>cd %USER_SELECTED_EDKII_WORKSPACE_DIR%
>.\svn_setup.py
>svn update
```

Note: The `svn update` command can take a few minutes to complete depending on the speed of your internet connection.

Note: If these commands fail, it may be due to your proxy settings. Contact your network administrator. You may find answers about proxy settings here: https://wiki.yoctoproject.org/wiki/Working_Behind_a_Network_Proxy

Note: The examples above do not show the optional OpenSSL pre build step described in the `CryptoPkg/Library/OpensslLib/Patch-HOWTO.txt` file.



4.3 Building all the EDKII Firmware Validated Build Configurations [Linux build environment only]

This section is only supported in Linux Build environments. The `buildallconfigs.sh` file is used to build all the validated EDKII build configurations. Open a terminal window and `cd` to the `Quark_EDKII*` directory created in [Section 4.2.1 Performing pre-build Steps in a Linux/gcc Build Environment](#).

The script has the following options:

```
buildallconfigs.sh [GCC44 | GCC45 | GCC46 | GCC47 | GCC48 | GCC49] [GCC Path]
[PlatformName]
```

GCC4x	GCC flags used for this build. Set to the version of GCC you have installed. Note: Tested on GCC46. GCC43 is not supported from release 1.2 onwards.
GCC Path	Location where the GCC is installed. GCC Path is not needed for release 1.1 and earlier.
[PlatformName]	Name of the platform package you want to build.

Example usage:

Create a build for an Intel® Quark™ SoC platform based on GCC version 4.6:

For release 1.2 onwards:

```
./buildallconfigs.sh GCC46 /usr/bin/ QuarkPlatform
```

For release 1.1 and earlier:

```
./buildallconfigs.sh GCC46 QuarkPlatform
```

Note: Ensure the selected version of GCC matches the one installed on the system by running the `gcc --version` command.

The build output can be found in the following directories:

- `Build/QuarkPlatform/<Config>/<Target>_<Tools>/FV/FlashModules/`
Contains EDKII binary modules
- `Build/QuarkPlatform/<Config>/<Target>_<Tools>/FV/Applications/`
Contains UEFI shell applications, including `CapsuleApp.efi`

Where:

- `<Config>` = PLAIN | SECURE
- `<Target>` = DEBUG | RELEASE
- `<Tools>` = GCC44 | GCC45 | GCC46 | GCC47 | GCC48 | GCC49

In [Section 8](#), you will run a script that creates a symbolic link to the directory where the EDK binaries are placed.



4.4 Building a Single EDKII Firmware Build Configuration

This section is supported in Linux and Windows build environments. Use `quarkbuild.sh` in a Linux terminal window or `quarkbuild.bat` in a Windows command prompt (created by running `cmd.exe`) and the `cd` command to change directory to the root of the EDKII workspace directory created in [Section 4.2](#).

Build usage:

```
quarkbuild [-r32 | -d32 | -clean]
[GCC44 | GCC45 | GCC46 | GCC47 | GCC48 | GCC49 | subst drive letter] [GCC Path]
[PlatformName] [-DSECURE_LD (optional)] [-DTPM_SUPPORT (optional)]
[-DSECURE_BOOT_ENABLE=TRUE (optional)]
```

The following is the list of options for the `quarkbuild.sh` and `quarkbuild.bat` build commands:

<code>-clean</code>	Delete the build files/folders
<code>-d32</code>	Create a DEBUG build
<code>-r32</code>	Create a RELEASE build
<code>GCC4x</code>	LINUX ONLY: GCC flags used for this build. Set to the version of GCC you have installed.

Note: Validated and tested on GCC46. GCC43 is not supported from Release 1.2 onwards.

`subst drive letter` **WINDOWS ONLY:** `quarkbuild.bat` uses the letter specified here with the Windows `subst` command to associate a drive letter with the EDKII workspace directory path. Associating a drive letter with the EDKII workspace directory reduces flash space requirements for debug executables.

`[PlatformName]` Name of the Platform package you want to build

`[-DSECURE_LD]` Create a Secure Lockdown build (optional).

Note: The policy decisions taken during EDKII boot for this boot option are automatically taken on secure SKU Quark SoC hardware even if `SECURE_LD` build option is not specified.

`[-DTPM_SUPPORT]` Create an EDKII build with TPM support (optional)

`[-DSECURE_BOOT_ENABLE=TRUE]` Create an EDKII build with UEFI Secure Boot support (optional)

Note: The TPM and UEFI Secure Boot build options require the one-time prerequisite described in the `CryptoPkg\Library\OpensslLib\Patch-HOWTO.txt` file. For more details on TPM (Trusted Platform Module) and UEFI Secure Boot, refer to the *Intel® Quark™ SoC X1000 UEFI Firmware Writer's Guide*.



Linux example usage:

Create a RELEASE build with UEFI Secure Boot and TPM support for an Intel® Quark™ SoC platform based on GCC version 4.6:

For release 1.2 onwards:

```
./quarkbuild.sh -r32 GCC46 /usr/bin/ QuarkPlatform -  
DSECURE_BOOT_ENABLE=TRUE -DTPM_SUPPORT
```

For release 1.1 and earlier:

```
./quarkbuild.sh -r32 GCC46 QuarkPlatform -DSECURE_BOOT_ENABLE=TRUE -  
DTPM_SUPPORT
```

Windows example usage:

Create a DEBUG build for a Quark platform. After executing quarkbuild.bat, a virtual drive S: is created which is rooted to EDKII workspace directory:

```
>.\quarkbuild.bat -d32 S QuarkPlatform
```

The built binaries are used in conjunction with [Section 8](#) to create Spi Flash binaries for development / manufacture, capsules for firmware update and recovery files for firmware recovery.

Note: A CAPSULE_FLAGS option can be used with the make utility in [Section 8](#), this option is UEFI specific and will override the default UEFI capsule flags used by the make utility, example > ../../spi-flash-tools*/Makefile CAPSULE_FLAGS=0x00050000. See Create an Update Capsule in Intel® Quark™ SoC X1000 UEFI Firmware Writer's Guide for information on UEFI capsule flags.

4.5 EDKII Firmware Build Standalone Output Files

[Section 8 Creating a Flash Image for the Board \[Linux* build environment only\]](#) is not required if the user only requires EDKII firmware in the SPI Flash on Intel® Quark™ SoC Open SKU silicon. [Sections 4.3](#) and [4.4](#) create the following EDKII standalone output files:

- .\Build\...\FV\FlashModules\Flash-EDKII-missingPDAT.bin

Full 8 MB image for manufacture with just the EDKII SPI flash images. The user is still expected to use [Section 9](#) following to create the final image for the board. The platform data python script referenced in the [Section 9](#) can be used in Linux* or Windows* build environments.

- .\Build\...\FV\RemediationModules\Flash-EDKII.cap

Capsule with just EDKII flash images that can be used instead of the capsule file referenced in [Section 10](#). Applying this file only updates the EDKII components of the SPI flash (as provided in the capsule). All other SPI flash assets remain intact.

Note: If programming Flash-EDKII.cap on a board, it is required that the SPI Flash version of the target board is at production level V1.0.0 or later. (Provided by the Flash.cap or Flash+PlatformData.bin files referenced in the sections that follow.)



- `.\Build\...\FV\RemediationModules\CapsuleApp.efi`

UEFI application referenced in [Section 10.2](#).

- `.\Build\...\FV\RemediationModules\FVMAIN.fv`

Recovery file that can be used is included in [Section 14](#). This version of the file only has the EDKII SPI flash images.

Note: To change the default UEFI Capsule flags for the EDKII standalone builds the **quarkbuild.bat** or **quarkbuild.sh** files must be changed.



5 *Building the GRUB OS Loader* *[Linux* Build Environment Only]*

If you will run Yocto, skip this section and use the file output by Yocto in this directory: `yocto_build/tmp/deploy/images/quark/grub.efi`

If you are only interested in building a Flash image without Linux* and not in using Yocto, then proceed through this section.

Note: GRUB is provided in two places: inside the meta-quark Yocto BSP or independently.

Tip: If you want to build a Flash image without a Yocto Linux system (for example, because you plan to boot a larger Yocto Linux system from an SD card or USB stick), you should modify the appropriate `layout.conf` file and delete the sections for `bzImage` and `core-image-minimal-initramfs-quark.cpio.gz`.

Dependencies:

- GCC (tested with version 4.3.4 and 4.6.3, and `libc6-dev-i386`)
- `gnu-efi-3.0u` library (tested with version `>= 3.0`)
- GNU Make
- Autotools (`autoconf`, `automake`, and `libtool`)
- Python 2.6 or higher
- `git`
- `gcc-multilib`
- `texinfo`

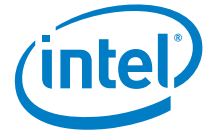
This GRUB build requires the 32 bit `gnu-efi` library which is included with many Linux distributions. Alternatively, you can download the latest version from:

<http://sourceforge.net/projects/gnu-efi/files>

Unpack and compile the `gnu-efi` library using the commands:

```
# tar -xvf gnu-efi*
# cd gnu-efi*/gnuefi
# make ARCH="ia32"
# cd -
```

To build GRUB, **first open a new terminal session**, extract the `grub` package, and run the `gitsetup.py` script. The script downloads all the upstream code required for `grub` and applies the patch.



Note: If you are not using Debian and had to manually install `gnu-efi` in a non-system location, then you must point `GNUEFI_LIBDIR` at the location where `gnu-efi` was compiled or installed.

Run the following commands:

```
# sudo apt-get install git autoconf
# tar -xvf grub-legacy_*.tar.gz
# cd grub-legacy_*
# ./gitsetup.py
# cd work
# autoreconf --install
# export CC4GRUB='gcc -m32 -march=i586 -fno-stack-protector'
# export GNUEFI_LIBDIR=/full/path/to/gnu-efi-3.0/gnuefi/
# CC="{CC4GRUB}" ./configure-quark.sh
# make
# cd -
```

Note: If these commands fail, it may be due to your proxy settings. Contact your network administrator. You may find answers about proxy settings here:

https://wiki.yoctoproject.org/wiki/Working_Behind_a_Network_Proxy

The required output from this build process is the `work/efi/grub.efi` file.



6 *Creating a File System and Building the Kernel Using Yocto Project**

Dependencies:

- git
- diffstat
- texinfo
- gawk
- chrpath
- lzop
- file

Note: Before running `git`, install `patchutils` so that the system can patch all external sources that are required when running `setup.sh`.

`git` requires proxy configuration. If these commands fail, it may be due to your proxy settings. Contact your network administrator. You may also find answers and tips about proxy settings here:

https://wiki.yoctoproject.org/wiki/Working_Behind_a_Network_Proxy

Yocto Project* can be used to create a root file system and kernel which boots from an SD card, USB key or SPI flash. Do not run any of the commands in this section as root.

Note: See [Section 7](#) to build development tools (`gcc`) for the Linux* operating system.

To avoid a known issue unzipping packages with long file paths, extract the `meta-quark` tarball into a directory with a short path, for example `/tmp`.

First, **open a new terminal session**, extract the Yocto Project layer, and run the `setup.sh` script to download the external sources required for the Yocto Project build:

```
# tar -xvf meta-quark*.tar.gz
# cd meta-quark*
# ./setup.sh
```

Note: The `setup.sh` script takes no parameters.



When this scripts runs successfully, it will add additional folders such as bitbake, meta-yocto, repo-ext, etc.

Next, source the `oe-init-build-env` command to initialize the Yocto Project build environment. This command takes the build directory name as its parameter:

```
# source ./oe-init-build-env yocto_build
```

After this command runs, the current directory will now be the directory specified in the parameter (`yocto_build` in this case). From this directory, run `bitbake <target>` to build the root file system and kernel.

The SoC-specific `<target>` commands described below will determine whether the resulting components are built for SPI flash or for SD/USB/eMMC (memory media). The output is different for each target.

Note: Starting from Intel® Quark™ v1.1 release, it is possible to perform both (memory media and SPI) build methods from the same directory. However, the user should take note of the DISTRO setting in `conf/local.conf` file. For the memory media image, please make sure `DISTRO ?="iot-devkit-multilibc"` is uncommented only. For SPI image, please make sure `DISTRO ?="iot-devkit-spi"` is uncommented only. The output of the memory media image is under `yocto_build/tmp` and the output of SPI image is under `yocto_build/tmp-spi`.

6.1 Build a Full-featured Linux for SD card, USB stick or eMMC

Note: A Yocto Project build can take several hours to complete, depending on your internet connection speed and your machine's specifications.

To build an image suitable for running on memory media, set the `bitbake` target to `image-full`:

```
# bitbake image-full
```

When the image build completes, the following output files are found in the `./tmp/deploy/images/quark/` directory. Copy these files to the media device. Rename the files according to the following instructions (resulting paths are relative to the media's root path):

- `image-full-quark-YYYYMMDDhhmmss.rootfs.ext3`
(Rename this file to `image-full-quark.ext3` because the `initramfs` will look for the file in that name, unless the configuration is updated).
- `core-image-minimal-initramfs-quark-YYYYMMDDhhmmss.rootfs.cpio.gz`
(Optional: rename this file to `core-image-minimal-initramfs-quark.cpio.gz`)
- `bzImage--3.14-r0-quark-YYYYMMDDhhmmss.bin`
(Optional: rename this file to `bzImage.bin`)
- `grub.efi`
- `boot` (directory)



The `grub.conf` file must be located in the `/boot/grub/` directory of the media device. Update the filenames in the `grub.conf` file after renaming.

If build error happens, install ACPICA manually from <https://www.acpica.org/downloads>.

6.2 Build a Small Linux for SPI Flash

An image capable of running in SPI flash must not exceed 8Mb in size and is therefore, a smaller, less featured image than the media image.

Before building out a SPI image we first need to edit a configuration file. From the build directory (`yocto_build` in our case), edit the `conf/local.conf` file and change the `DISTRO` variable. This can be done easily by commenting out (using the hash symbol `#`) the default value at line 115 and uncommenting the value used for SPI flash images at line 114:

```
DISTRO ?= "iot-devkit-spi"
#DISTRO ?= "iot-devkit-multilibc"
```

Before

```
102 # Default policy config
103 #
104 # The distribution setting controls which policy settings are used as
defaults.
105 # The default value is fine for general Yocto project use, at least initially.
106 # Ultimately when creating custom policy, people will likely end up
subclassing
107 # these defaults.
108 #
109 # Notes:
110 # (1) If you wish to build an image for the Galileo's on-board SPI flash you
111 #     need to set DISTRO to one of the iot-devkit-spi value listed below.
112 # (2) If you wish to build an image (Linux kernel, grub and root file-system)
that
113 #     is stored on SD card, you need to set DISTRO to iot-devkit-multilibc
114 #DISTRO ?= "iot-devkit-spi"
115 DISTRO ?= "iot-devkit-multilibc"
```



After

```
102 # Default policy config
103 #
104 # The distribution setting controls which policy settings are used as
defaults.
105 # The default value is fine for general Yocto project use, at least initially.
106 # Ultimately when creating custom policy, people will likely end up
subclassing
107 # these defaults.
108 #
109 # Notes:
110 # (1) If you wish to build an image for the Galileo's on-board SPI flash you
111 #     need to set DISTRO to one of the iot-devkit-spi value listed below.
112 # (2) If you wish to build an image (Linux kernel, grub and root file-system)
that
113 #     is stored on SD card, you need to set DISTRO to iot-devkit-multilibc
114 DISTRO ?= "iot-devkit-spi"
115 #DISTRO ?= "iot-devkit-multilibc"
```

The bitbake target for a SPI image is image-spi:

```
# bitbake image-spi
```

Note: Because the SPI image is smaller than the full image, it completes quicker but may still take over an hour depending on network speed and machine specifications.

For the Intel® Galileo board, output files are found in `./tmp-spi/deploy/images/quark` and include the following components:

- image-spi-quark.cpio.gz
- image-spi-quark.cpio.lzma
- bzImage
- grub.efi

Intel® Quark™ Linux on the SPI image uses `uclibc`, which is a C library optimized for embedded systems. This enables a very small Linux footprint that can fit into 8Mb SPI flash together with the UEFI bootloader and Grub OS loader.



These components now need to be loaded into SPI flash memory. The processes for doing this are covered in [Sections 8, 9, 10, and 11](#).

6.3 Build a fast boot Linux for SD card or eMMC

The fast boot image is capable of boot up from off state to Linux prompt within a few seconds, it has the same feature configuration as the SPI flash image, and boots directly from SD card or eMMC.

Similarly as building out a SPI image, we first need to edit the configuration files to enable fast boot. From the build directory (yocto_build in this case), edit the `conf/local.conf` file and change the DISTRO variable to "quark-fast".

```
#DISTRO ?= "iot-devkit-spi"

#DISTRO ?= "iot-devkit-multilibc"

DISTRO ?= "quark-fast"
```

Before

```
109 # Notes:

110 # (1) If you wish to build an image for the Galileo's on-board SPI flash you
111 #     need to set DISTRO to one of the iot-devkit-spi value listed below.

112 # (2) If you wish to build an image (Linux kernel, grub and root file-system)
113 #     that
114 #     is stored on SD card, you need to set DISTRO to iot-devkit-multilibc
115 #DISTRO ?= "iot-devkit-spi"
116 DISTRO ?= "iot-devkit-multilibc"
```

After

```
109 # Notes:

110 # (1) If you wish to build an image for the Galileo's on-board SPI flash you
111 #     need to set DISTRO to one of the iot-devkit-spi value listed below.

112 # (2) If you wish to build an image (Linux kernel, grub and root file-system)
113 #     that
114 #     is stored on SD card, you need to set DISTRO to iot-devkit-multilibc
115 #DISTRO ?= "iot-devkit-spi"
116 #DISTRO ?= "iot-devkit-multilibc"
```



```
116 DISTRO ?= "quark-fast"
```

Next we need to include "meta-intel-quark-fast" to the BBLAYERS setting in `conf/bblayer.conf` as the example shown in "meta-intel-quark-fast/conf/bblayer.conf.sample". e.g.

```
9  BBLAYERS ?= " \
    . . . .
12  ${BBPATH}/../meta-intel-iot-devkit \
13  ${BBPATH}/../meta-intel-quark-fast \
    . . . .
```

The bitbake target for image-fast is:

```
# bitbake image-fast
```

The output files are found in `./tmp-fast/deploy/images/quark` and include the following components:

- image-fast-quark.ext3
- image-fast-quark.hddimg
- image-fast-quark.manifest
- modules-quark.tgz

The fast boot image has the following limitations:

- The fast boot is targeted only for non-secure X1000 platforms
- The fast boot does not provide support for all Quark iot-devkit/makers specific services
- The fast boot uses glibc instead of uclibc
- The fast boot image enables all the auto-load driver modules during Linux boot time as configured in the flash image-spi

The fast boot solution uses the OpenEmbedded Image Creator tool `wic` (available in the "scripts" directory) to create the partitioned boot disk is capable of direct boot from SD card or eMMC (USB is not supported).

To create a bootable partition image using `wic`:

```
# wic create ../meta-intel-quark-fast/mkefidisk.wks \
    -e image-fast -o ./fast_boot_image
```



To create the fast boot image on SD card or eMMC using dd command (assume we are still in yocto_build directory):

WARNING. The following step will erase all the existing contents on the SD card or eMMC.

```
# cd ./fast_boot_image/build
# sudo dd if=mkefidisk-<date-time>-mmcblk0.direct of=/dev/mmcblk0; sync
```

TIP. Always run "sync" after "dd" command to make sure the operating systems file cache is flushed to SD card/eMMC.

Now the fast boot partition is successfully created on the SD card or eMMC, it should be able to boot directly on your Intel® Quark™ X1000 platform.

NOTE. The default size of fast boot partition is 64Mb, it can be configured through the "--size" parameter defined in meta-intel-quark-fast/mkefidisk.wks file.

6.4 Applying a Custom Patch to the Linux kernel Using Yocto Project (optional)

If you need any customization of your kernel (such as additional debug statements or custom driver behavior), then you may need to patch the Linux kernel. This optional step must be done **before** you run the bitbake command.

1. For customization of Yocto Project source code, extract the updates to a patch from git using the git diff or git format-patch commands.
2. Copy the patch to the location as follows:

```
$ cp mypatch.patch /PATH/TO/MY_BSP/meta-quark-bsp/recipes-
kernel/linux/files/
```

3. Locate the bitbake recipe file:

```
/PATH/TO/MY_BSP/meta-intel-quark/recipes-kernel/linux/linux-yocto-
quark_3.14.bb
```

4. Append the following line:

```
SRC_URI += "file://mypatch.patch"
```

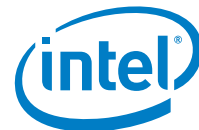
For example:

```
printf '%s\n' 'SRC_URI += "file://mypatch.patch"' >> linux-yocto-
quark_3.14.bb
```

5. Return to [Section 6](#) and run the bitbake command to get new images.

More information can be found here:

- http://www.yoctoproject.org/docs/current/ref-manual/ref-manual.html#var-SRC_URI
- <http://www.yoctoproject.org/docs/current/dev-manual/dev-manual.html#platdev-appdev-devshell>



7 *Building the Linux* Cross Compile Toolchain Using Yocto Project* [Linux Build Environment Only]*

To develop applications for the target architecture, a toolchain that allows cross compilation is required. There are a number of different methods for building out a cross toolchain. Section 3 of the Yocto Project* Application Developer's Guide at the link below provides a detailed description of each method.

<http://www.yoctoproject.org/docs/1.7/adt-manual/adt-manual.html#adt-prepare>

There are options for creating a toolchain in the current build directory and for creating toolchain installer scripts which will allow the toolchain to be installed on other (SDK) machines different to the build machine.

When building out a cross toolchain, a matching target sysroot is also needed because it contains metadata, libs and headers used to build the appropriate cross toolchain.

Creating a cross toolchain in the current build directory

The easiest method for creating a toolchain, assuming the file system and kernel have already been built in [Section 6](#), is to use the following bitbake command:

```
bitbake meta-ide-support
```

This command will create an environment setup file suitable for the target machine defined by the MACHINE variable in the local.conf file:

```
(excerpt)
# Machine Selection
MACHINE ??= "quark"
```

The resulting environment setup file will be located in /tmp and will contain "environment-setup" string in the name. For example, "environment-setup-i586-poky-linux". This method assumes that the build machine and the SDK machine are the same and that the sysroot is in the current build directory.

Creating a cross toolchain installer

In order to build a cross toolchain capable of being installed on a different machine or in a different area to the current build directory, a cross toolchain installer is required.

If building for a different SDK machine, ensure that the SDKMACHINE variable in local.conf describes this architecture:

```
(excerpt)
```



```
# This variable specifies the architecture to build SDK/ADT items for and
means
# you can build the SDK packages for architectures other than the machine
you are
# running the build on (i.e. building i686 packages on an x86_64 host).
# Supported values are i686 and x86_64
#SDKMACHINE ?= "i686"
```

You **must** source the `oe-init-build-env yocto_build` every time you use a new terminal. To build the toolchain installer to match the target root filesystem already built, **open a new terminal session** and use the `bitbake` command as follows:

```
# bitbake image-full -c populate_sdk
```

The output of the build process is a script that installs the toolchain on another system:

```
iot-devkit-glibc-x86_64-image-full-i586-toolchain-1.7.2.sh
```

The script is located in `./tmp/deploy/sdk`. Run the script and extract the cross toolchain to a target directory (default: `/opt/iot-devkit/1.7.2`). This will extract the cross toolchain along with the sysroot to the target directory.

Note: The environment setup script may change your environment significantly, thus breaking other, non-Yocto Project tools you might be using (including anything which uses Python). **You must open a new terminal session** to source the Yocto Project environment and run `make`, and run all your other commands in other terminal sessions.

When you are ready to compile your application, first run the `source` command below to define default values for `CC`, `CONFIGURE_FLAGS`, and other environment variables, then you can compile:

```
# source /opt/iot-devkit/1.7.2/environment-setup-i586-poky-linux
# ${CC} myfile.c -o myfile
```

For general details, see the Yocto Project Application Development Toolkit (ADT) information: <https://www.yoctoproject.org/tools-resources/projects/application-development-toolkit-adt>

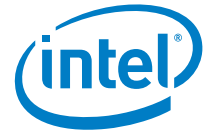
Instructions about adding a package to the Linux* build are found in Section 5.2 of the Yocto Project Development Manual:

<http://www.yoctoproject.org/docs/1.7/dev-manual/dev-manual.html#usingpoky-extend-customimage>

If you do not have any size constraints, you can change the C library (using the `TCLIBC` variable) to a more fully featured C library. Detailed instructions are found in Section 5.10 of the Yocto Project Quick Start:

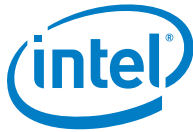
<http://www.yoctoproject.org/docs/1.7/mega-manual/mega-manual.html>

It is also possible to write a Yocto Project custom recipe which can be built using the `bitbake` command. One advantage of this is that the application can be automatically added to the target root file system.



See Section 5.3 of the Yocto Project Development Manual for more detail on writing a new recipe:

<http://www.yoctoproject.org/docs/1.7/dev-manual/dev-manual.html#new-recipe-writing-a-new-recipe>



8 *Creating a Flash Image for the Board [Linux* build environment only]*

Dependencies:

- GCC
- GNU Make
- EDKII Firmware Volume Tools (base tools)
- OpenSSL 0.9.8zb
- libssl-dev

8.1 Using the SPI Flash Tools

The SPI Flash Tools, along with the metadata and flash image configuration in the sysimage archive, are used to create a binary file that can be installed on the board and booted.

Open a new terminal session and extract the contents of the sysimage archive:

```
# tar -xvf sysimage_*.tar.gz
```

Extract and install SPI Flash Tools:

```
# tar -xvf spi-flash-tools*.tar.gz
```

Note: Extract all files to a directory that does not include the original tar files.

The sysimage* directory contains the following preconfigured layout.conf files:

- release build base SKU (non-secure)
- debug build base SKU (non-secure)
- release build secure SKU
- debug build secure SKU

Depending on what kind of image you want to build, you must be in either the sysimage.CP-8M-debug or the sysimage.CP-8M-release directory.

The layout.conf file defines how the various components will be inserted into the final binary file to be flashed onto the board. The layout.conf consists of a number of [sections] with associated address offsets, file names, and parameters. Each section must reference a valid file, so it is necessary to update the paths or create symbolic links to the valid files.

Note: For more information, refer to [Appendix B SPI Flash Tools](#).

A script is provided that creates symbolic links. Run the script with the command from /PATH/TO/BSP/ directory:



```
# ./sysimage/create-symlinks.sh
```

Ensure there is no whitespace around the values defined in the `layout.conf` file.

Once a valid `layout.conf` has been created, run the SPI Flash Tools makefile with the command:

```
# ../../spi-flash-tools*/Makefile
```

The output of this build is located in either the `sysimage.CP-8M-debug` or the `sysimage.CP-8M-release` directory (depending on what kind of image was selected).

The output of this build includes:

- `Flash.cap` - standard capsule file.
Use this file to program your board using the serial interface by following the *Programming the Flash* instructions in [Section 10](#).
- `Flash-missingPDAT.bin` - flash file with no platform data.
Use this file to program your board with the platform data tool and a `Dediprogram`, as described in [Section 9](#) and then [Section 11](#).
- `FVMAIN.fv` – board-specific recovery file.
See [Section 14](#) for an overview of capsule recovery. If you are using the Intel® Galileo board, refer to the *Intel® Galileo Board User Guide* for details. For other boards, contact your Intel representative for details.

The capsule file contains a BIOS, bootloader, and compressed Linux* run-time system to allow a Quark-based board to boot. Use the capsule update mechanism described in [Section 10](#) to program the SPI flash on your board.

Note: The same build process and same image files are used for both secure and non-secure board SKUs, however, secure SKUs have certain restrictions on where a capsule update can be performed. If you have a secure SKU board (Industrial/Energy or Transportation Reference Design), you **must** update your board using the Linux run-time system ([Section 10.3](#)).

For experienced users, you can build all sysimages configuration in just one command by running the following command at the top-level directory of the sysimage package:

```
../spi-flash-tools/Makefile [ -j ] sysimages
```

Note: Be aware of the plural `sysimages` in the command.
The `-j` option builds concurrently, which completes in a shorter time, however the output may be harder to read.



9 Platform Data Tool

The platform data file provides platform personality values such as platform type and board personality values such as Ethernet MAC addresses that must be patched into the previously mentioned Flash-missingPDAT.bin and Flash-EDKII-missingPDAT.bin files. This section is required for users who wish to update flash contents using flash programmers, for example, during a manufacturing process. This section can also be used just to generate a .pdat file that can be used as one of the capsule images placed in the firmware update .cap files or recovery FVMAIN.fv files (see [Section 4](#) and [Section 8](#)).

Platform data is part-specific, unique data placed in SPI flash. Every binary image flashed to the board must be patched individually to use platform data. A data patching script is provided in this release.

Note: The *Intel® Quark™ SoC X1000 UEFI Firmware Writer's Guide* contains information on common platform data items.

The platform data patching script (`platform-data-patch.py`) is stored in the `platform-data` directory within the `spi-flash-tools` tarball in the Intel® Quark™ SoC BSP. The following text is written as if the user is executing the script on a Linux* build machine, but the script may also be run on a Windows* build machine which has Python 2.6 or Python 2.7 installed.

Before running the `platform-data-patch.py` script, open a new terminal session and copy and edit the `spi-flash-tools/platform-data/sample-platform-data.ini` file to include platform-specific data such as MAC address, platform type, and MRC parameters.

On reference platforms, the MAC address to be programmed is printed on the product label.

Note: The Intel® Quark™ SoC X1000 contains two MACs and each must be configured with one address in the `platform-data.ini` file, even on boards (such as Intel® Galileo board) that have only one Ethernet port. For the Intel® Galileo board, MAC 0 is the only MAC wired out. The default MAC 0 address value in the `platform-data.ini` file is invalid and must be set to the value allocated to your system, typically this is identified on a sticker. MAC 1 must also have a valid UNICAST MAC address and the `platform-data.ini` file contains a dummy but valid address for MAC 1. If you do **not** set a valid MAC address, system will halt during boot.



In the following UEFI Secure boot Crosshill example, recommended values are shown in **bold** text:

```
[Platform Type]
id=1
desc=PlatformID
data.type=hex.uint16data.value=4
# Examples: KipsBay 3, CrossHill 4, ClantonHill 5, Galileo 6,
# GalileoGen2 8, RelianceCreek 9, RelianceCreekSPU A
```

Note: In the [Mrc Params] section below, the MRC data.value MUST correspond to the platform data.value used above.

```
[Mrc Params]
id=6
ver=1
desc=MrcParams
data.type=file
# Examples:
# KipsBay
# data.value=MRC/kipsbay.v1.bin
# CrossHill
data.value=MRC/crosshill.v1.bin
# ClantonHill
# data.value=MRC/clantonhill.v1.bin
# Galileo
# data.value=MRC/kipsbay-fabD.v1.bin
# GalileoGen2
# data.value=MRC/GalileoGen2.bin
# RelianceCreek
# data.value=MRC/RelianceCreek.bin
# RelianceCreekSPU
# data.value=MRC/RelianceCreekSPU.bin

[MAC address 0]
id=3
desc=1st MAC
data.type=hex.string
data.value=001320FDF4F2 #replace with MAC address from sticker on board

[MAC address 1]
id=4
desc=2nd MAC
data.type=hex.string
data.value=02FFFFFFFF01 #replace with MAC address from sticker on board
```

Next, run the script as follows:

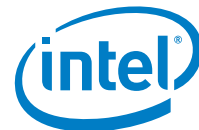
```
# cd spi-flash-tools/platform-data/
# ./platform-data-patch.py -p sample-platform-data.ini \
  -i ../../sysimage_*/sysimage.CP-8M-release/Flash-missingPDAT.bin
# cd -
```

This creates a Flash+PlatformData.bin file to be programmed on the board, as well as a sample-platform-data.pdat file containing the same data that was inserted in the Flash image.

To program your board using Dediprog, skip to [Section 11](#).



Note: If creating Flash+PlatformData.bin and the sample-platform-data.ini placed UEFI Secure boot certificates in the patched binary file then the generated sample-platform-data.pdat must be used as one of the capsule images placed in the recovery FVMAIN.fv file (see [Section 8](#)), to allow the UEFI Secure boot system to be recovered. By default EDKII firmware will not update the MAC address in the platform data area if programming an EDKII capsule (refer to the *Intel® Quark™ SoC X1000 UEFI Firmware Writer's Guide*).



10 Programming Flash on the Board Using Serial Interface

Dependencies: CapsuleApp.efi (built in [Section 4](#), located in Build/QuarkPlatform/<Config>/<Target>_<Tools>/FV/Applications/)

The BSP provides a mechanism to update SPI flash contents based on EDKII capsules. These capsules contain a BIOS, bootloader, and compressed Linux* run-time system sufficient to boot a Quark-based board, such as the Intel® Galileo board.

The capsule update mechanism can be triggered from an EDKII shell ([Section 10.2](#)) or from a Linux run-time system ([Section 10.3](#)). In both situations, you must have root privileges on the system.

If you have a secure SKU board (Industrial/Energy or Transportation Reference Design), you **must** update your board using the Linux run-time system ([Section 10.3](#)).

10.1 Programming Flash to Release 1.2.1

The layout and size of Stage 1 EDKII firmware volume have changed since release 1.1. The SPI flash layout is incompatible with previous releases before the release 1.2.0, therefore, the user needs to do the following:

If you are updating from release 1.2.0, please refer section 10.2 and 10.3 for direct update to release 1.2.1

If you are updating from an earlier release than release 1.2.0:

- i) Update firmware to 1.1 (if applicable)
- ii) Update the 1.1 firmware to 1.1 interim
- iii) Update from 1.1 interim to release 1.2.1

The interim version firmware is release 1.1 version firmware with FlashMap.h and SMIFlashDxe.c files replaced with those of release 1.2.1. Updating to the interim version is required to update the flash mapping of EDKII and capsule update code.

NOTE: Update release 1.0.x and older to release 1.1 so that the files are suitable for update.

10.1.1 Requirements to Update to Release 1.2.1

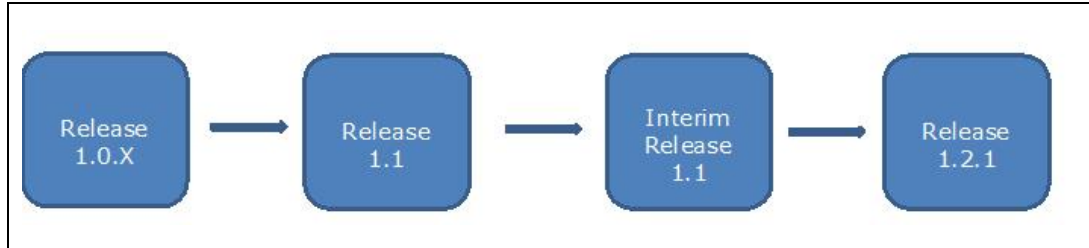
To update to release 1.2.1, download the following EDKII source codes from <https://downloadcenter.intel.com/search?keyword=Intel%20ae+Quark%e2%84%a2+SoC>:

1. Quark EDKII of Quark BSP Release 1.1
2. Quark EDKII of Quark BSP Release 1.2.1



Figure 1 shows the updating process from release 1.0.x to release 1.2.1.

Figure 1. Updating Process from Release 1.0.x to Release 1.2.1



10.1.2 Steps to Update to Release 1.2.1

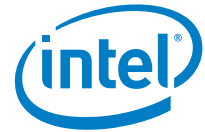
The following are the steps to update the system firmware to release 1.2.1:

1. Download and build the Quark EDKII release 1.1 as described in [Section 4](#).
 - i. If you have an older release, e.g. release 1.0.x, update to release 1.1 firmware.
Note: The GCC path is not needed as a parameter in build commands for release 1.1 and earlier versions.
Refer to [Section 4.3](#) for changes.
2. Download and extract Quark EDKII release 1.2.1.
3. Create and update to the interim firmware of release 1.1:
 - i. To create the interim firmware, replace the *Flashmap.h* and *SMIFlashDxe.c* files of release 1.1 with those of release 1.2.1. The files can be found at following locations, respectively:
 - *QuarkPlatformPkg/Include/FlashMap.h*
 - *QuarkPlatformPkg/Platform/DxeSmm/SMIFlashDxe/SMIFlashDxe.c*
 - ii. Rebuild the EDKII with changes in step 3i. The release 1.1 commands to build the EDKII can be found in <https://downloadcenter.intel.com/download/23823/Intel-Quark-BSP-Release-Archive>.
 - iii. Generate the interim release 1.1 capsule files, *Flash.cap* with steps described in [Section 8](#).
Note: The *layout.conf* and tools (*spi-flash-tools*, *sysimage*) used are based on Release 1.1.
 - iv. Update the system firmware to this interim firmware with the *Flash.cap* capsule file generated in step 3ii.
 - v. The system is now updated with the interim firmware of release 1.1.
4. The user can now update to release 1.2.1 EDKII firmware using the release 1.2.1 capsule file, *Flash.cap*.

Note: Firmware downgrade from release 1.2.1 to an older release does not require any special steps, only the upgrade process requires additional attention.

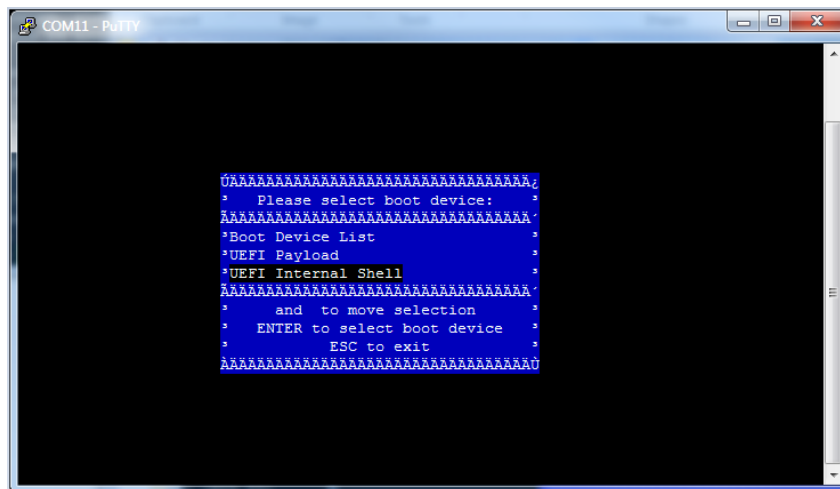
10.2 Programming flash using UEFI shell

This procedure cannot be used for a secure SKU board (Industrial/Energy or Transportation Reference Design) because the UEFI shell is not available on secure SKU boards. Follow the [Section 10.3](#) procedure instead.

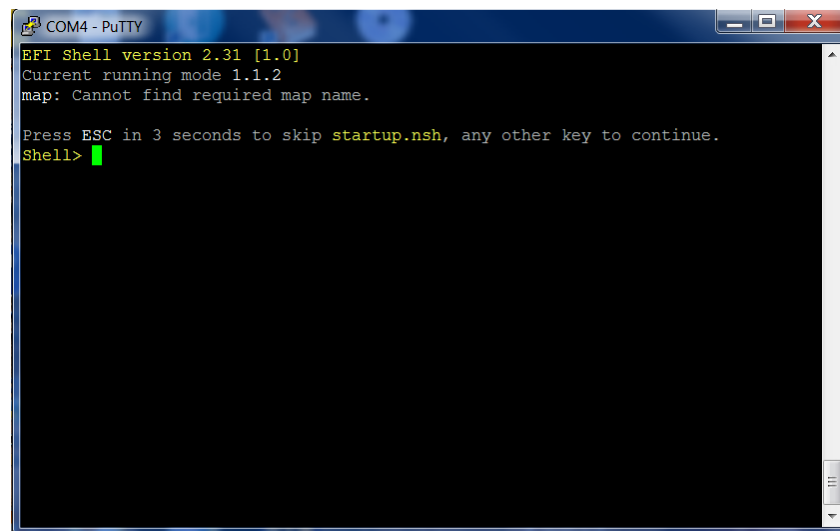


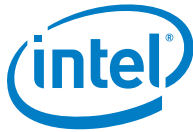
Perform the steps that follow:

1. Use the files created in [Section 8](#).
2. Copy `CapsuleApp.efi` and `Flash.cap` to a microSD card (or USB stick) and insert it into the slot on the board.
3. Connect the serial cable between the computer and the board. Set up a serial console session (for example, PuTTY) and connect to the board's COM port at 115200 baud rate.
4. Configure the serial console session to recognize special characters. For example, if you are using PuTTY, you must explicitly enable special characters. In the PuTTY Configuration options, go to the `Terminal > Keyboard` category and set the `Function keys` and `Keypad` option to `SCO`. You may also set `Backspace` to the `Control-H` key.
5. Power on the board. Enter the EFI shell before grub starts by pressing F7.
6. The serial console displays a boot device selection box (below). Select `UEFI Internal Shell`.



You will see a display similar to this:





6. You will see a print out, the top line of which looks like this:
fs0 :HardDisk - Alias hd7b blk0

This is your SD card. To mount it, type: fs0:
7. Verify you are using the correct version of CapsuleApp.efi by using the -v option. You **must** use version 1.01 or later.
8. Enter the following command:
CapsuleApp.efi Flash.cap

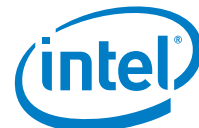
Note: You must enter the full filename of the Flash.cap file.

You will see a display similar to this:

```
COM4 - PuTTY
Transfer mode write = 0x37
SendCommand: Command Index = 18
Transfer mode read = 0x37
Transfer mode write = 0x37
SendCommand: Command Index = 18
Transfer mode read = 0x37
Transfer mode write = 0x37
SendCommand: Command Index = 18
Transfer mode read = 0x37
Transfer mode write = 0x37
SendCommand: Command Index = 18
Transfer mode read = 0x37
Transfer mode write = 0x37
Read(LBA=00007A00, Buffer=0E3C6010, Size=00010000)
SendCommand: Command Index = 18
Transfer mode read = 0x37
Transfer mode write = 0x37
CapsuleApp: creating capsule descriptors at 0xF1DE310
CapsuleApp: capsule data starts      at 0xD655410 with size 0x740190
CapsuleApp: capsule block/size      0xD655410/0x740190
CapsuleImage Address is 000D655410, CapsuleImage Size is 740190
CapsuleFragment Address is 000DFCFE90, CapsuleInfo Address is 000DFCFF10
Start to update capsule image!
```

The CapsuleApp will update your SPI flash image. This process takes about 5 minutes.

- Warning:** DO NOT remove power or try to exit during this process. Wait for the prompt to return, otherwise your board will become non-functional.
9. When the update completes, the board will automatically reboot. You will see a display similar to this:



```
COM4 - PuTTY
[ 14.236101] pci spi probe(), enable_msi 1, mmio_base e0776000, dev c01bd000
[ 14.243984] MSI enabled, irq number is 44
[ 14.248040] ssp type is CE5X00 SSP
[ 14.251652] add_spi_dev devices GPIO CS off
[ 14.312295] pxa2xx-spi pxa2xx-spi.0: master is unqueued, this is deprecated
[ 14.338110] pxa2xx-spi pxa2xx-spi.1: master is unqueued, this is deprecated
Starting Bootlog daemon: bootlogd.
Configuring network interfaces... [ 17.288952] eth0: device MAC address 00:13:
20:fd:f4:60
udhcpd (v1.20.2) started
Sending discover...
Sending discover...
Sending discover...
No lease, failing
kernel.hotplug = /sbin/mdev
sh: %4Y%2m%2d%2H%2M: bad number
INIT: Entering runlevel: 5
Starting syslogd/klogd: done
Stopping Bootlog daemon: bootlogd.
/sketch/sketch.elf file does not exist or invalid permissions
clloader waiting to receive.
Poky 9.0 (Yocto Project 1.4 Reference Distro) 1.4.1 clanton /dev/ttyS1
clanton login: █
```

10. Updating to release 1.2.1 requires additional steps in [Section 10.1](#).

10.3 Programming Flash Using Linux* Run-time System

If you are updating from an earlier release of the BSP software (0.7.5 and 0.8.0), you need a release-specific kernel module. Note that a 0.7.5 kernel module cannot be loaded on a 0.8.0 BSP and vice-versa.

Open a new terminal session and perform the following steps:

1. Use the files created in [Section 8](#).
2. Copy `Flash.cap` from the `sysimage` directory onto an SD card (or USB stick) and insert it into the board.
3. **Release 0.9.0, Release 1.0.0, and later:**
Run the command:
`# modprobe efi_capsule_update`
4. **Release 1.1.0 to Release 1.2.1:**
Require additional steps in [Section 10.1](#).
Continue with step 5 to update the flash.
5. **All releases:**
Run the following commands:
`# modprobe sdhci-pci`
`# modprobe mmc-block`
`# modprobe efi_capsule_update`
`# mkdir /lib/firmware`
`# cd /media/mmcblk0p1/`
`# cp Flash.cap /lib/firmware/Flash.cap`
`# echo -n Flash.cap >`
`/sys/firmware/efi_capsule/capsule_path`
`# echo 1 > /sys/firmware/efi_capsule/capsule_update`
`# reboot`



Note: Make sure you use the `reboot` command; removing/reinserting the power cable will **not** work.

Warning: It is critical to ensure that the older `sysfs` entries used by release 0.7.5 and release 0.8.0 are **not** used due to known issues:
 `/sys/firmware/efi/capsule_update`
 `/sys/firmware/efi/capsule_path`

The capsule update method for release 0.9.0 and later uses the following corrected entries:

`/sys/firmware/efi_capsule/capsule_update`
 `/sys/firmware/efi_capsule/capsule_path`



11 Programming Flash on the Board Using DediProg

You can use a DediProg* SF100 SPI Flash Programmer and the associated flashing software to program your board.

Note: These steps require the `Flash+PlatformData.bin` file that was created in [Section 9](#).

Once the software has been installed and the programmer is connected to the board, **open a new terminal session**, and run the DediProg Engineering application.

Use the following steps to flash the board:

1. Select the memory type if prompted when the application starts.
2. Select the File icon and choose the `*.bin` file you wish to flash.
3. Optionally select the Erase button to erase the contents of the SPI flash.
4. Select `raw` file format.
5. Select the Prog icon to flash the image onto the board.
6. Optionally select the Verify icon to verify that the image flashed correctly.

Note: Intel recommends that you disconnect the programmer before booting the system.



12 Booting the Board From SD Card

To boot your board from an SD card and enable persistent `rootfs`, follow these steps. You can also use this procedure to boot your board from a USB stick or eMMC.

If you are using an Intel® Galileo board, this setup allows you to save your Arduino* sketch to the board, so it will be able to repeat sketches after board power-down. This also enables a persistent `/sketch` folder and `rootfs`.

Dependencies:

- You ran the command `bitbake image-full` in [Section 6](#)
- Your SD card must meet the following requirements:
 - SD card must be formatted as FAT or FAT32.
 - SD card size must be 32GB (or smaller) and SDHC format. SDXC format is **not** supported.

1. The output of the build process in [Section 6](#) is found in `./tmp/deploy/images/quark/`

Copy the following kernel and root file system files to an SD card:

- `boot` (directory)
- `bzImage`
- `core-image-minimal-initramfs-quark.cpio.gz`
- `image-full-quark.ext3`

Be sure to set up your SD card with the files and structure shown as follows.

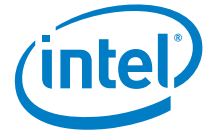
Name	Date modified	Type	Size
boot	6/11/2015 2:57 PM	File folder	
bzImage	6/11/2015 4:45 PM	File	2,274 KB
core-image-minimal-initramfs-quark.cpio.gz	6/11/2015 4:45 PM	WinZip File	2,771 KB
image-full-quark.ext3	6/11/2015 4:47 PM	EXT3 File	307,200 KB

2. Insert the SD card, then power on the board.

Note: The first time you boot the board may take several minutes. This is expected behavior due to the SSH component creating cryptographic keys on the first boot.

Troubleshooting tips:

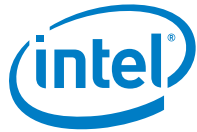
To boot from SD/USB/eMMC, the grub instance embedded in the SPI flash is hardcoded to search for a `boot/grub/grub.conf` file in partition 1 on the media



device. This is compatible with the factory formatting of most SD/USB devices. By default, the UEFI firmware does not try to boot from SD or USB, it is handled by grub.

If you use an SD or USB device that has been reformatted after manufacturing, you might experience problems booting from it. First, try to boot with a different memory device and see if the problem goes away. If you isolate the problem to a specific SD card, you can restore the factory formatting using this tool from the SD association: https://www.sdcard.org/downloads/formatter_4/

It is not recommended to use normal operating system tools to format flash memory devices.



Part 2 of 2 - Using the BSP Software

This section contains the following subsections:

- 13. [Capsule Update](#)
- 14. [Capsule Recovery](#)
- 15. [Signing Files \(Secure SKU only\) \[Linux* build environment only\]](#)
- 16. [Enabling the OpenOCD Debugger](#)

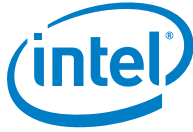


13 Capsule Update

The BSP software provides a mechanism to update SPI flash contents based on EDKII capsules. These capsules contain a BIOS, bootloader, and compressed Linux* run-time system sufficient to boot a Quark-based board, such as the Intel® Galileo board. Capsule update is comprised of the following high-level steps:

- Building a `Flash.cap` capsule file
- Connecting a USB key or SD card that contains this file to the board
- Running the capsule update mechanism as described in [Section 10.2](#) or [Section 10.3](#).
- If updating to release 1.2.1, additional steps in [Section 10.1](#) must be executed.

Note: If you have a secure SKU board (Industrial/Energy or Transportation Reference Design), you **must** update your board using the Linux run-time system ([Section 10.3](#)).



14 Capsule Recovery

The BSP software provides a mechanism for the SPI flash contents to be recovered if the board will not boot. For example, if power was lost during a normal SPI flash update, the board would be unbootable.

Capsule recovery is comprised of the following high-level steps:

- building a `FVMAIN.fv` recovery file
- connecting a USB key with this file to the board
- booting the board in recovery mode

Note: If you are using the Intel® Galileo board, refer to the *Intel® Galileo Board User Guide* for details. For other boards, contact your Intel representative for details on how to boot in recovery mode.

- waiting for the recovery firmware to update the SPI flash and reboot the board
- Capsule recovery is expected to recover from the same version of recovery file in the system.

Booting in recovery mode is board specific. Please refer to the board user guide for details.

Alternatively, the SPI flash contents can be recovered using a DediProg* SF100 SPI Flash Programmer and the associated flashing software to program your board as described in [Section 11](#).



15 Signing Files (Secure SKU only) [Linux* build environment only]

This step is optional for most users; it is only needed for booting on a secure SKU.

The SPI Flash Tools package includes the Asset Signing Toolset, an application used for signing assets for secure boot. Follow the steps below to compile the signing tool, then sign assets.

For complete details on the Asset Signing Toolset, including all of the command line options, refer to the *Intel® Quark™ SoC X1000 Secure Boot Programmer's Reference Manual* (see [Appendix A](#)).

Note: For convenience during development, the software release includes a default Private Key `key.pem` file. During development, all assets are signed with the default key that is stored in the `config` directory. The default key **cannot** be used in a production system; it is not secure due to its inclusion in the release package. Contact your Intel representative for details.

Note: Please contact your intel representative for a production Signed Key Module if necessary

Dependencies: `libssl-dev`

15.1 To Program Secure SPI Flash

Copy `SignedKeyModule.bin` and `private_key.pem` to:

`<Path to BSP>/sysimage*/sysimage.CP-8M-release-secure/config/`

In `<Path to BSP>/sysimage*/sysimage.CP-8M-release-secure/layout.conf` edit the file to point to this signed key module just copied.

ie.

```
[signed-key-module]
```

```
.....
```

```
Item_file=config/<name of signed key module>.bin
```

From here create a spi flash image as per chapter 8 and program the board using a `dediprog` as per chapter 11.



15.2 To Create a Secure Boot SD/eMMC

Use files in <Path to BSP/meta-clanton/yocto_build/tmp/deploy/images/quark/> files located by grub require signature files for verification. This includes kernel, grub.conf, bzImage, and core-image-minimal-initramfs-quark.cpio.gz.

Open a new terminal session and use the following commands:

```
# cd spi-flash-tools
# make asset-signing-tool/sign
```

After compiling the signing tool, you can sign assets as shown in the following example:

```
# path/to/spi-flash-tools/asset-signing-tool/sign -i <input file>
-s <svn> -x <svn index> -k <key file> -c
```

The output for this example is a signed binary file called <input file>.signed in the same directory as the <input file> but adding the -c command creates a signature file, .csbh file.

<Key file> here can point to customer private_key.pem

<svn> and <svn index> can be set to "0" in this instance.

Pass the -c command line option which creates <input file>.csbh as output in the same directory as the <input file>.

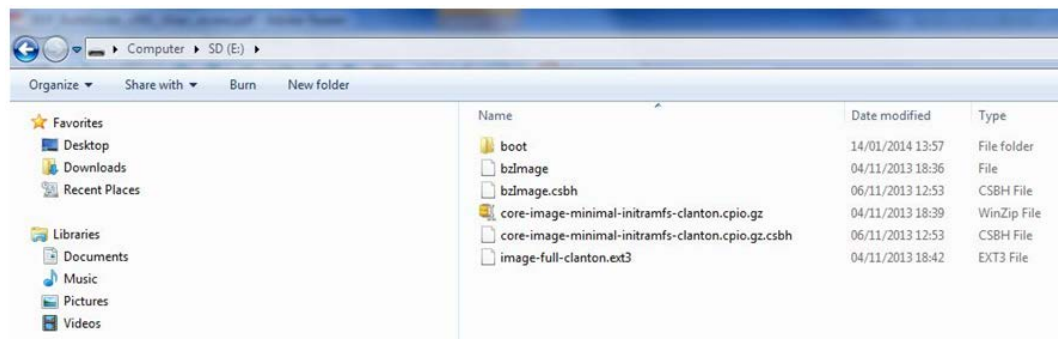
To get a full list of command line options, run the signing tool with no option.

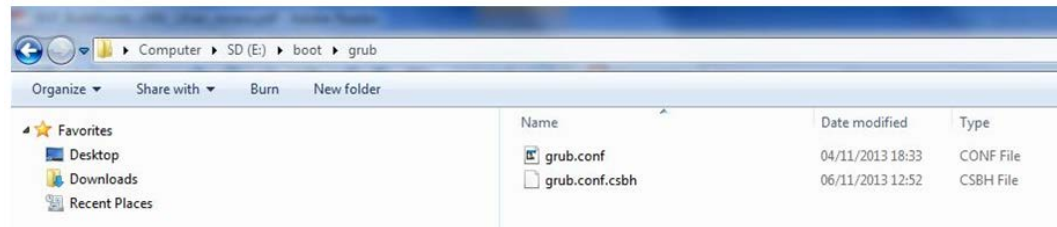
The signature files can be copied onto the boot media and must comply with the following requirements:

- Each .csbh file must be in the same directory as the corresponding non-signed file.
- grub.conf must be located in the /boot/grub/ directory.
- Other files can be placed anywhere as long as grub.conf is configured with their location.

The screenshots below show an example SD card with signature files:

- Copy signature files core-image-minimal-initramfs-quark.cpio.gz.csbh and bzImage.csbh to the root directory.
- Copy grub.csbh to the /boot/grub/ directory.







16 *Enabling the OpenOCD Debugger*

Complete instructions for using the OpenOCD debugger can be found in the *Source Level Debug using OpenOCD/GDB/Eclipse on Intel® Quark™ SoC X1000 Application Note*, refer to [Appendix A](#).



Appendix A *Related Documents*

The documents in the following table provide more information about the software in this release.

Document Name	Number
Intel® Quark™ SoC X1000 Board Support Package (BSP) Build and Software User Guide (this document)	329687
Intel® Quark™ SoC X1000 Software Release Notes	330232
Intel® Quark™ SoC X1000 Secure Boot Programmer's Reference Manual	330234
Intel® Quark™ SoC X1000 Linux* Programmer's Reference Manual	330235
Intel® Quark™ SoC X1000 UEFI Firmware Writer's Guide	330236
EDKII Update for Intel® Quark™ SoC X1000 Software Release Notes	330729
Source Level Debug using OpenOCD/GDB/Eclipse on Intel® Quark™ SoC X1000 Application Note https://communities.intel.com/docs/DOC-22203	330015
Intel® Quark™ SoC X1000 Datasheet https://communities.intel.com/docs/DOC-21828	329676
Intel® Quark™ SoC X1000 Core Developer's Manual https://communities.intel.com/docs/DOC-21826	329679
Intel® Quark™ SoC X1000 Core Hardware Reference Manual https://communities.intel.com/docs/DOC-21825	329678
Intel® Galileo Board User Guide https://communities.intel.com/docs/DOC-22475	330237
6LoWPAN Support on Intel® Galileo Gen 2	559445



Appendix B *SPI Flash Tools*

The main functionality of the SPI flash tools is provided by a Make file. By default the Make file should be called from within a directory with a valid layout.conf and will output a flash binary and capsule.

Example of usage is in [Section 6.2 Build a Small Linux for SPI Flash](#).

The Make file provides a number of targets which can be used to change the behavior or output of the tool:

- all (default) - creates the flash binary, capsule and recovery image
- raw - create only the flash binary
- capsule - create only the capsule
- recovery - creates a recovery image
- clean - delete all intermediate files in the current directory
- sysimages - Search all subdirectories for a layout.conf file and if found build the images
- sysimages-clean - as above but calls clean
- sysimages-install - as above but copies all of the output files into an "installs" directory

There are also a number of variables which can be prefixed to the Makefile call on the command line to modify the behavior of the tool:

- KEYFILE - can be used to select an encryption key to use if assets are being signed
- LAYOUTFILE - specify the name of a layout file if it has a non-standard name
- PDAT_IN_CAPSULE - include the platform data in the capsule
 - true ==> include the platform data in the capsule
 - false ==> do not include the platform data in the capsule (default false)
- CAPSULE_FLAGS:
 - 0x00010000 ==> do NOT update MAC addresses
 - 0x00010001 ==> update the MAC addresses (default: 0x00010000)
- CAPSULE_SVN - update CAPSULE_SVN when changing KEYS for UEFI Secure Boot Systems (default = 0)

Example: KEYFILE=/path/to/key.pem ../spi-flash-tools/Makefile

- PDAT_INI ==> name of test system pdata .ini file (defaults to platform-data.ini if not given)