Intel® Quark™ SoC X1000
Board Support Package (BSP)
Build and Software User Guide

Release: 1.0.1

22 May 2014

Order Number: 329687-006US

Contents

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 Building the BSP Software** contains instructions for installing and configuring the Intel® Quark™ SoC X1000 Board Support Package sources.
- **Part 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 v1.0.0
- Intel® Galileo Customer Reference Board (CRB) (Fab D with blue PCB)
- Kips Bay Customer Reference Board (CRB) (Fab C with green PCB)
- Intel® Quark™ SoC X1000 Industrial/Energy Reference Design (Cross Hill)
- Intel® Quark™ SoC X1000 Transportation Reference Design (Clanton Hill)

Part 1 Building the BSP Software

This section contains the following subsections:

2 Before you begin

Before you begin:

- You need a host PC running Linux*. Intel recommends a 64-bit Linux system.
- 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:
  - serial interface using the UEFI shell or Linux* run-time (see Section 10)
  - DediProg* SF100 SPI Flash Programmer (or equivalent) and the associated flashing software (see Section 11)
  - Intel® Galileo IDE (Galileo board only; see the *Intel® Galileo Board Getting Started Guide* for 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 below must be run in a new terminal session every time**.

Note: If these 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 Debian* Linux* 7.0 (Wheezy) but will work with most other Linux distributions.

This release is 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 here:

https://downloadcenter.intel.com/Detail_Desc.aspx?DwnldID=23197

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_v1.0.1.7z` (2.6 MB)

For customers using the Clanton Hill FFRD, additional CAN software must be downloaded from IBL/CDI. See your Intel representative for the URL.
The CAN package is comprised of:

- Fujitsu CAN Firmware:
  - `CAN_Firmware_for_Intel_Quark_v1.0.1.zip` (36 kB)

Debian provides a 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 Quark. Additional details may be found here:

- www.tianocore.sourceforge.net
- http://sourceforge.net/apps/mediawiki/tianocore/index.php?title=Getting_Started_with_EDK_II

Dependencies:

- Python 2.6 or higher
- GCC and G++ (tested with GCC 4.3 and GCC 4.6)
- `subversion` client
- `uuid-dev`
- `iasl`

The Quark EDKII BSP is named `Quark_EDKII_<version>.tar.gz`. Once it has been extracted, run the `svn_setup.py` script. The script fetches the upstream code required to build the firmware modules.

Open a new terminal session and enter the following commands:

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

Once `svn update` has completed, use the `buildallconfigs.sh` script to build the modules.

Some of the configurations built by the script have a dependency on OpenSSL, therefore, before you run the script you **must** follow the instructions outlined in the `CryptoPkg/Library/OpensslLib/Patch-HOWTO.txt` file.

The script has the following options:

```
buildallconfigs.sh [GCC43 | GCC44 | GCC45 | GCC46 | GCC47] [PlatformName]

    GCC4x       GCC flags used for this build. Set to the version of GCC
                you have installed.
                NOTE: Validated with GCC43; tested on GCC46
[PlatformName]  Name of the Platform package you want to build
```

Example usage:

```
./buildallconfigs.sh GCC46 QuarkPlatform # Create a build for Quark
                  Platform based on GCC version 4.6
```

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> = GCC43 | GCC44 | GCC45 | GCC46 | GCC47
```

In Section 8 you will run a script that creates a symbolic link to the directory where the EDK binaries are placed.

For experienced users only, save time and build a single configuration as follows:

```
quarkbuild.sh [-r32 | -d32 | -clean] [GCC43 | GCC44 | GCC45 | GCC46 | GCC47]
    [PlatformName] [-DSECURE_LD (optional)] [-DTPM_SUPPORT (optional)]

    -clean      Delete the build files/folders
    -d32        Create a DEBUG build
    -r32        Create a RELEASE build
    GCC4x       GCC flags used for this build. Set to the version of GCC
                you have installed.
                NOTE: Validated with GCC43; tested on GCC46.
[PlatformName]  Name of the Platform package you want to build
[-DSECURE_LD]   Create a Secure Lockdown build (optional)
[-DTPM_SUPPORT] Create EDKII build with TPM support (optional)
                Note: This option has a one-time prerequisite described
                in CryptoPkg\Library\OpensslLib\Patch-HOWTO.txt
```

For more details on TPM (Trusted Platform Module), see the Intel® Quark™ SoC X1000 UEFI Firmware Writer's Guide.

Example usage:

```
./quarkbuild.sh -r32 GCC43 QuarkPlatform –DSECURE_LD # Create a Secure
                  Lockdown RELEASE build for Quark platform based on GCC
                  version 4.3
```

5 Building the GRUB OS loader

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

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

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

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-clanton.cpio.gz`.

Dependencies:

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

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

Dependencies:

- `git`
- `diffstat`
- `texinfo`
- `gawk`
- `chrpath`
- `file`

Note: `git` requires proxy configuration. 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

Use Yocto to create a root file system and kernel that boots the system from an SD card or USB key. 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-clanton` tarball into a directory with a short path, for example `/tmp`.

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

```
# sudo apt-get install diffstat gawk chrpath
# tar -xvf meta-clanton*.tar.gz
# cd meta-clanton*
# ./setup.sh
```

Note: The `setup.sh` script takes no parameters. To build the root file system and kernel for the Intel® Galileo board, see the commands below.

Next, source the `oe-init-build-env` command to initialize the Yocto build environment, and run `bitbake <target>` to build the root file system and kernel. You will use SoC-specific `<target>` commands described below.

Note: If you need to patch the Linux kernel (optional), skip to Section 6.1 and apply the patch before running the `bitbake` command.

Two build methods are supported; the output is slightly different for each one. The commands are different for the Intel® Galileo board.

Note: You cannot perform the following build methods sequentially, they are mutually exclusive. If you want both builds, you must perform them on two completely different and isolated directories.

Build a small Linux for SPI Flash

For the Intel® Galileo board, run:

```
# source poky/oe-init-build-env yocto_build
# bitbake image-spi-galileo
```

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

- `image-spi-galileo-clanton.cpio.gz`
- `image-spi-galileo-clanton.cpio.lzma`
- `bzImage`
- `grub.efi`

For other supported boards (not Intel® Galileo), run:

```
# source poky/oe-init-build-env yocto_build
# bitbake image-spi
```

Output files are found in `./tmp/deploy/images/` and include:

- `image-spi-clanton.cpio.gz`
- `image-spi-clanton.cpio.lzma`
- `bzImage`
- `grub.efi`

Build a full-featured Linux for SD card or USB stick

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

For the Intel® Galileo board, run:

```
# source poky/oe-init-build-env yocto_build
# bitbake image-full-galileo
```

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

- `image-full-galileo-clanton.ext3`
- `core-image-minimal-initramfs-clanton.cpio.gz`
- `bzImage`
- `grub.efi`
- `boot (directory)`

For other supported boards (not Intel® Galileo), run `bitbake image-full` as shown below:

```
# source poky/oe-init-build-env yocto_build
# bitbake image-full
```

Output files are found in `./tmp/deploy/images/` and include:

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

The kernel and root file system (`bzImage` and `image-nnnn.gz`, respectively) can be copied onto a USB stick or SD card and booted from grub. Also, the file `grub.conf` must be located in the `/boot/grub/` directory of the USB stick or SD card.

6.1 Applying a custom patch to the Linux kernel using Yocto (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 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 below:
   ```
   $ cp mypatch.patch /PATH/TO/MY_BSP/meta-clanton/meta-clanton-bsp/recipes-kernel/linux/files/
   ```
3. Locate the bitbake recipe file:
   `/PATH/TO/MY_BSP/meta-clanton-bsp/recipes-kernel/linux/linux-yocto-clanton_3.8.bb`
4. Append the following line:
   `SRC_URI += "file://mypatch.patch"`

   For example:
   ```
   echo "SRC_URI += \"file://mypatch.patch\"" >> linux-yocto-clanton_3.8.bb
   ```
5. Return to Section 6 and run the bitbake command to get new images.

More info 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

The steps to build the cross compile toolchain are the same as the steps for the Yocto root file system and kernel build as described in Section 6, with the exception of the `bitbake` command arguments.

To build the tool chain, **open a new terminal session** and follow the steps in Section 6 but modify the `bitbake` command as follows:

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

The same files can be used for both builds, however, you **must** source the `poky oe-init-build-env yocto_build` every time you use a new terminal.

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

```
clanton-tiny-uclibc-x86_64-i586-toolchain-1.4.2.sh
```

The script is located in `./tmp/deploy/sdk`

Note: The script may change your environment significantly, thus breaking other, non-Yocto tools you might be using (including anything which uses Python). **You must open a new terminal session** to source the Yocto 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/clanton-tiny/1.4.2/environment-setup-x86_32-poky-linux
# ${CC} myfile.c -o myfile
```

or

```
# source  /opt/clanton-tiny/1.4.2/environment-setup-x86_64-poky-linux
# ${CC} myfile.c -o myfile
```

For general details, see the Yocto 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 may be found here: http://www.yoctoproject.org/docs/current/dev-manual/dev-manual.html#usingpoky-extend-customimage-localconf

Quark Linux uses `uclibc`, which is a C library optimized for embedded systems. This enables a very small Linux that can fit into SPI flash with the UEFI bootloader and Grub OS loader.

If you do not have any size constraints, you can change the C library to a more fully featured C library. Detailed instructions are here:
http://www.yoctoproject.org/docs/current/mega-manual/mega-manual.html
specifically how to change the `TCLIBC` variable selecting the C library to be used.

To build an `eglibc` image, overwrite the default value of the `DISTRO` variable as follows:

```
DISTRO="clanton-full" bitbake <image-name>
```

8 Creating a flash image for the board

Dependencies:

- GCC
- GNU Make
- EDKII Firmware Volume Tools (base tools)
- OpenSSL 0.9.8w
- `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 `Flash.bin` 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 `Flash.bin` 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.

A script is provided that creates symbolic links. Run the script with the command:

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

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

Note: If you are using the Intel® Galileo board, you may need to modify the `layout.conf` file in the `[Ramdisk]` section from `image-spi-clanton.cpio.lzma` to `image-spi-galileo-clanton.cpio.lzma` to successfully generate your `.cap` 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-missingPDAT.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 Dediprog, as described in Section 9, 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 10.2).

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 Defining the platform data file

Note: If you created a `*.cap` file in the previous section, a platform data file is not required and you can skip this section.

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

The platform data patching script is stored in the SPI Flash Tools archive. Before running the script, **open a new terminal session** and copy and edit the file `spi-flash-tools/platform-data/sample-platform-data.ini` 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 Galileo) that have only one Ethernet port.
For Galileo, 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, the following error message is returned:
`HALT: Multicast Mac Address configured for Ioh MAC`

Galileo / Kips Bay Fab D example is below, recommended values are shown in **bold** text:

```
[Platform Type]
id=1
desc=PlatformID
data.type=hex.uint16
# ClantonPeak 2, KipsBay 3, CrossHill 4, ClantonHill 5, KipsBay-fabD 6
data.value=6
```

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
#data.value=MRC/clantonpeak.v1.bin
#data.value=MRC/kipsbay.v1.bin
#data.value=MRC/crosshill.v1.bin
#data.value=MRC/clantonhill.v1.bin
data.value=MRC/kipsbay-fabD.v1.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
```

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.

To program your board using Dediprog, skip to Section 11.

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 sufficent to boot a Quark-based board, such as the Intel® Galileo board.

The capsule update mechanism can be triggered from an EDKII shell (Section 10.1) or from a Linux* run-time system (Section 10.2). In both situations, you must have root privileges on the system.

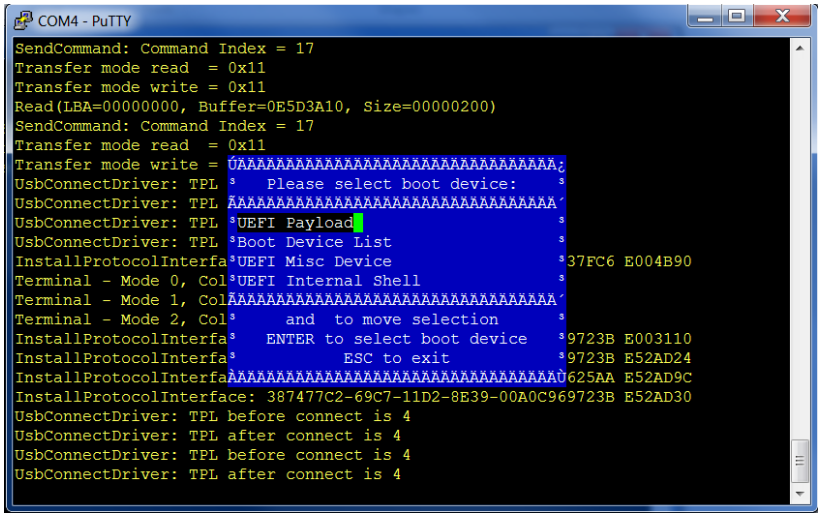
If you have a secure SKU board (Industrial/Energy or Transportation Reference Design), you **must** update your board using the Linux* run-time system (Section 10.2).

10.1 Programming flash using UEFI shell

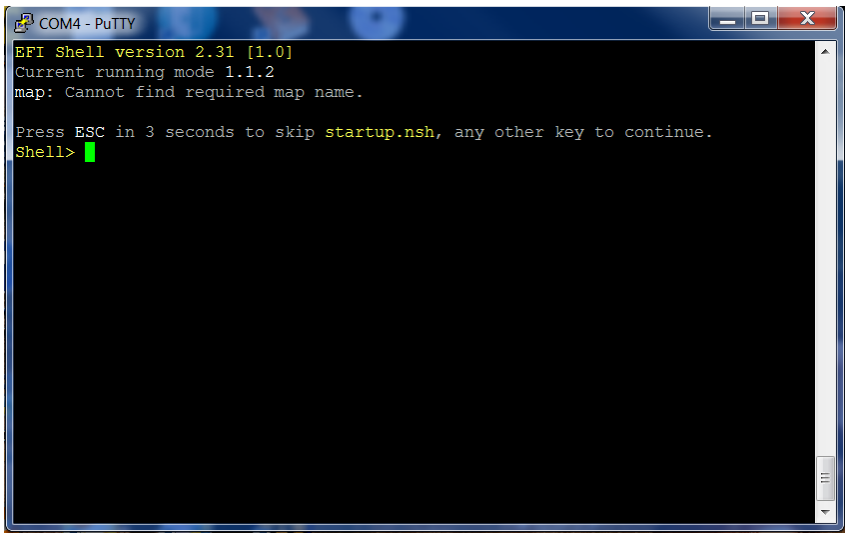
This procedure cannot be used for a secure SKU board (Industrial/Energy or Transportation Reference Design) because the UEFI shell is not available on secure SKU boards. Follow the Section 10.2 procedure instead.

Perform the steps below:

1. Use the files created in Section 8.
2. Copy `CapsuleApp.efi` and `Flash-missingPDAT.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:

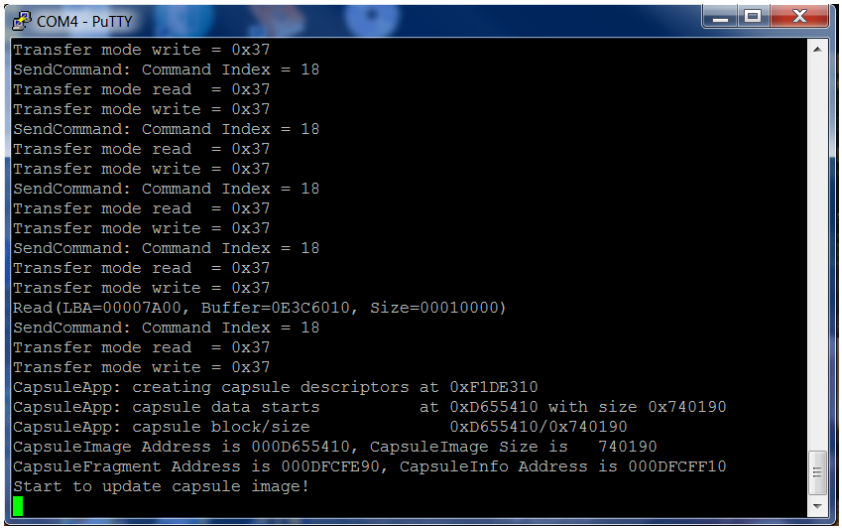


7. You will see a print out, the top line of which looks like this:
   `fs0  :HardDisk - Alias hd7b blk0`

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

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

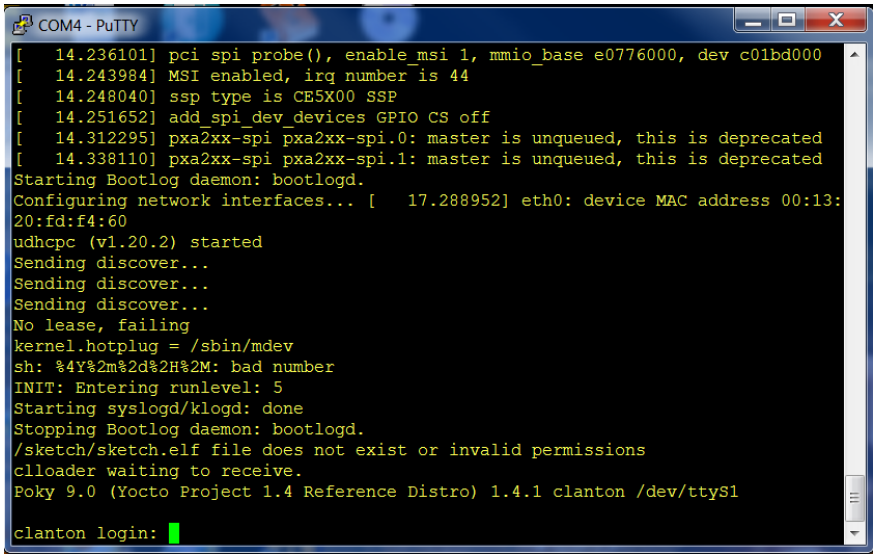
You will see a display similar to this:



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.

10. When the update completes, the board will automatically reboot. You will see a display similar to this:



10.2 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-missingPDAT.cap` from the `sysimage` directory onto an SD card (or USB stick) and insert it into the board.
3. **Release 0.7.5 and Release 0.8.0 only:**
   Run the command:
   ```
   # insmod /tmp/<release>/efi_capsule_update.ko
   ```
   where: `<release> = 0.7.5` or `0.8.0`
4. **Release 0.9.0, Release 1.0.0, and later:**
   Run the command:
   ```
   # modprobe efi_capsule_update
   ```
5. **All releases:**
   Run the following commands:
   ```
   # modprobe sdhci-pci
   # modprobe mmc-block
   # mkdir /lib/firmware
   # cd /media/mmcblk0p1/
   # cp Flash-missingPDAT.cap /lib/firmware/Flash-missingPDAT.cap
   # echo -n Flash-missingPDAT.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.

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`.

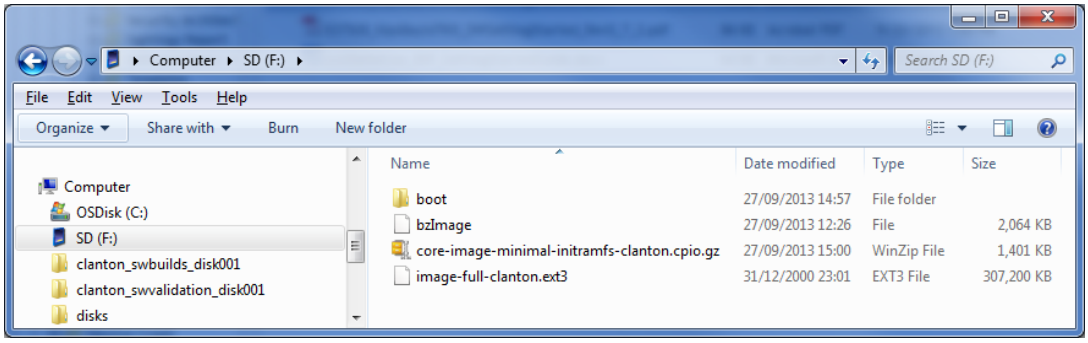
Dependencies:

- You ran the command `bitbake image-full` in Section 6 (or `bitbake image-full-galileo` if using an Intel® Galileo board)
- 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/`

   Copy the following kernel and root file system files to an SD card:
   - `boot (directory)`
   - `bzImage`
   - `core-image-minimal-initramfs-clanton.cpio.gz`
   - `image-full-clanton.ext3` or `image-full-galileo-clanton.ext3` for the Intel® Galileo board

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



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, the grub instance embedded in the SPI flash is hardcoded to search for a `boot/grub/grub.conf` file in partition 1 on the SD/USB card. 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 Using the BSP Software

This section contains the following subsections:

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 sufficent 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-missingPDAT.cap` capsule file
- connecting a USB key with this file to the board
- running the capsule update mechanism as described in Section 10.1 or Section 10.2.

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 10.2).

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

Booting in recovery mode typically requires wires to be soldered to the board. An alternate solution is to use 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)

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

Dependencies: `libssl-dev`

All files located by grub require signature files for verification. This includes kernel, `grub.conf`, `bzImage`, and `core-image-minimal-initramfs-clanton.cpio.gz`.

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.

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>
```

The output for this example is a signed binary file called `<input file>.signed` in the same directory as the `<input file>`.

To create a separate signature file, pass the `–c` command line option which creates `<input file>.csbh` as output in the same directory as the `<input file>`.

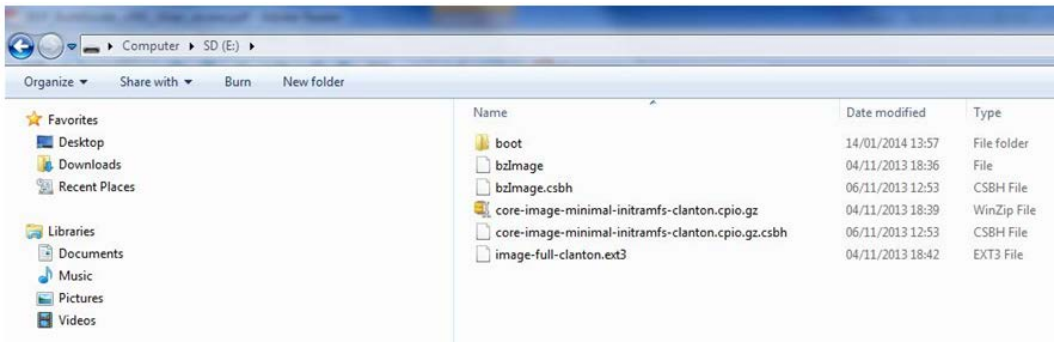
To get a full list of command line options, run the signing tool with no option.

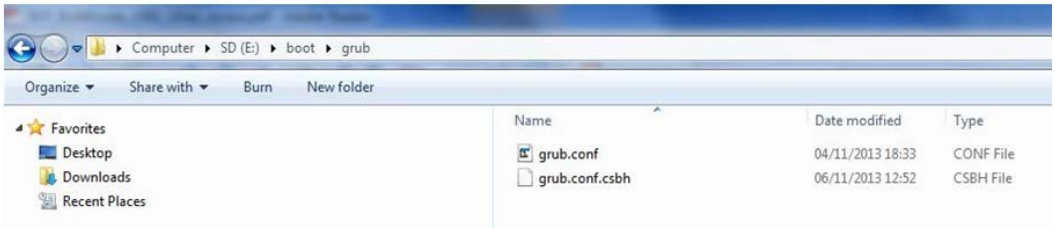
The signature files can be copied onto a USB stick or SD card and must comply with the following requirements:

- 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-clanton.cpio.gz.csbh` and `bzImage.csbh` to the root directory.
- Copy `grub.cbsh` 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*, see Appendix A.

Appendix A Related Documents

The documents below 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
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

Revision History

Date	Revision	Description
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, Applying a custom patch to the Linux kernel using Yocto (optional). Updated Section 9, Defining the platform data file (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 10.2, 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.

Legal Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document 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.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

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

Intel, the Intel logo, and Quark are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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