Intel® Quark SoC X1000
Board Support Package (BSP) Build Guide

Release: 0.8.0

07 November 2013

Order Number: 329687-002US

Contents

1 Before you begin 3

2 Downloading software 4

3 Building the GRUB OS loader 4

4 Creating a file system and building the kernel using Yocto 5

5 Building the cross compiler toolchain 7

5.1 Linux* cross compiler 7

5.2 Windows* cross compiler 8

5.3 MAC OS* cross compiler 8

6 Creating a flash image for the board 9

7 Defining the platform data file 10

8 Programming flash on the board using serial interface 11

8.1 Secure SKU Boards only 15

9 Programming flash on the board using DediProg 17

10 Booting the board from SD card 18

11 Signing files (optional) 18

12 Enabling the OpenOCD debugger 19

1 Before you begin

This guide contains instructions for installing and configuring the Intel® Quark SoC X1000 Board Support Package Sources.

This software release supports the following software and hardware:

- Board Support Package Sources for Intel® Quark SoC X1000 v0.8.0
- 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)

Before you begin:

- You need a host PC running Linux*.
- 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 `capsule.efi` (see Section 8)
  - DediProg* SF100 SPI Flash Programmer (or equivalent) and the associated flashing software (see Section 9)
  - 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, it may be due to your proxy settings. 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 other Linux distributions.

2 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 contains multiple zip files, including:

- Board Support Package (BSP) sources:
  - `Board_Support_Package_Sources_for_Intel_Quark_v0.8.0.7z` (5.6 MB)

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
```

3 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 4M Flash image and not in using Yocto, then proceed through this section.

Dependencies:

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

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 Clanton patch.

Run the following commands:

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

For the configure step, if you are not using Debian and had to manually install gnu-efi in a non-system location, then you need to point GNUEFI_LIBDIR at the location where gnu-efi was compiled or installed.

Note: If these commands fail, it may be due to your proxy settings. 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 efi/grub.efi file.

4 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. 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 5 to build development tools such as gcc for different host operating systems.

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:

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

Note: Use the `./setup.sh –s` command for the default BSP or use `./setup.sh –e layer` for the Intel® Galileo board.

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.

Two build methods are supported; the output is slightly different for each one.

Note: You cannot perform the following build methods sequentially, they are mutually exclusive.

1. If you want to build a very small Linux to fit into SPI flash, then run `bitbake image-spi` as shown below:

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

   The output of the build process is found in `./tmp/deploy/images/` and includes the following files:
   - `image-spi-clanton.cpio.gz`
   - `image-spi-clanton.cpio.lzma`
   - `bzImage`
   - `grub.efi`

2. If you want to build a more fully-featured Linux to run out of the SD card, then run `bitbake image-full` as shown below:

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

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

The output of the build process is found in `./tmp/deploy/images/` and includes the following files:
- `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.

5 Building the cross compiler toolchain

This section describes how to build an image that includes development tools for the following operating systems:

- Linux* cross compiler
- Windows* cross compiler
- MAC OS* cross compiler

5.1 Linux* cross compiler

The steps to build the cross compiler toolchain are the same as the steps for the Yocto root file system and kernel build as described in Section 4, with the exception of the `bitbake` command.

To build the tool chain, **open a new terminal session** and follow the steps in Section 4 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-full-eglibc-x86_64-i586-toolchain-1.4.1.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). To mitigate, Intel recommends that you open one 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/poky/1.4.2/environment-setup-x86_32-poky-linux
# ${CC} myfile.c -o myfile
```

or

```
# source  /opt/poky/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.

5.2 Windows* cross compiler

1. Follow all steps from Section 4.
2. Enter the commands below:
   ```
   cd ..
   git clone git://git.yoctoproject.org/meta-mingw
   cd yocto_build
   ```
3. Edit conf/bblayers.conf and add the full path to cloned meta-mingw layer.
4. Make sure the conf/local.conf file contains: SDKMACHINE = “i686-mingw32”
5. Enter the command: bitbake meta-toolchain
6. Toolchain will be created under tmp/deploy/sdk

5.3 MAC OS* cross compiler

1. Follow all steps from Section 4.
2. Enter the commands below:
   ```
   cd ..
   git clone git://git.yoctoproject.org/meta-darwin
   cd yocto_build
   ```
3. Edit conf/bblayers.conf and add the full path to cloned meta-darwin layer.
4. Make sure the conf/local.conf file contains: SDKMACHINE = “i386-darwin”
5. Enter the command: bitbake meta-toolchain
6. Toolchain will be created under tmp/deploy/sdk

6 Creating a flash image for the board

Dependencies:

- GCC
- GNU Make
- EDKII Firmware Volume Tools (base tools)
- OpenSSL 0.9.81 or newer
- `libssl-dev`

The SPI Flash Tools, along with the metadata 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 and Quark EDKII files:

```
# tar -xvf spi-flash-tools*.tar.gz
# tar -xvf Quark-EDKII-volumes*.tar.gz
```

The `sysimage*` directory contains a preconfigured `layout.conf` file. This 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 symlinks to the valid files.

An example for the GRUB component is:

```
# ln -s grub-legacy_*/work/ grub-legacy
```

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 to create one of the following files.

To build a capsule file (recommended), run the commands:

```
# ln -s ../../Quark_EDKII_* ../../Quark_EDKII
# ../../spi-flash-tools*/Makefile capsule
```

The output of this build is a `*.cap` file and other required files. Proceed with *Programming the Flash* instructions in either Section 8 or Section 9.

To build a `Flash.bin` file:

```
# ln -s ../../Quark_EDKII_* ../../Quark_EDKII
# ../../spi-flash-tools*/Makefile
```

The output of this build is a `Flash.bin` file. Continue with Section 7, then Section 9.

7 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 edit the `platform-data/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.ini` file, even on boards (such as Galileo) that have only one Ethernet port.
For Galileo, MAC 0 is the only MAC wired out, however, you must also configure MAC 1 with a dummy valid UNICAST MAC address in the `platform.ini` file.
The default MAC address values in the file are invalid. If you do **not** set valid MAC addresses, 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

# WARNING: the MRC parameters MUST match the platformID 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   #unique MAC address of your device

[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 9.

8 Programming flash on the board using serial interface

These steps assume you have a microSD card inserted into the slot on the board, which contains the release-specific `sysimage_nnnn.cap` file and the `CapsuleApp.efi` file.

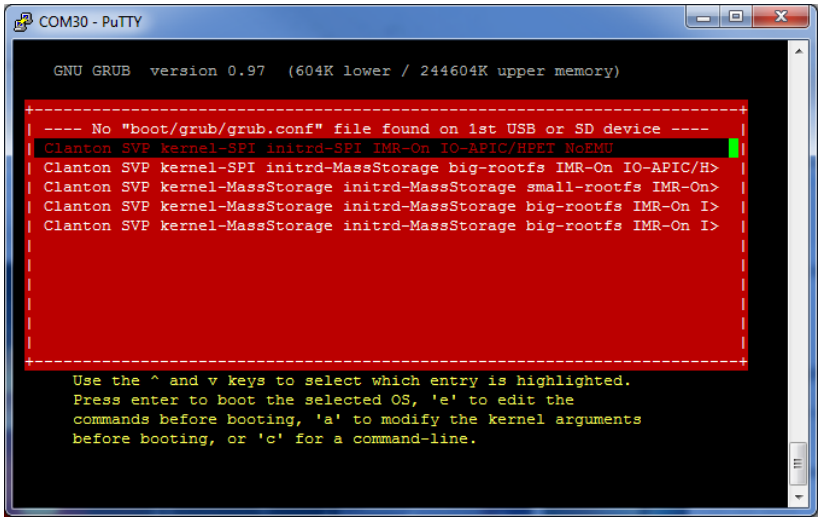
If you have a secure SKU board (Industrial/Energy or Transportation Reference Design), the steps are slightly different. See Section 8.1.

Caution: The `CapsuleApp.efi` and the `*.cap` file are a matched pair and must be used together. When a new `*.cap` file is released, you must use the corresponding `CapsuleApp.efi` to install it.

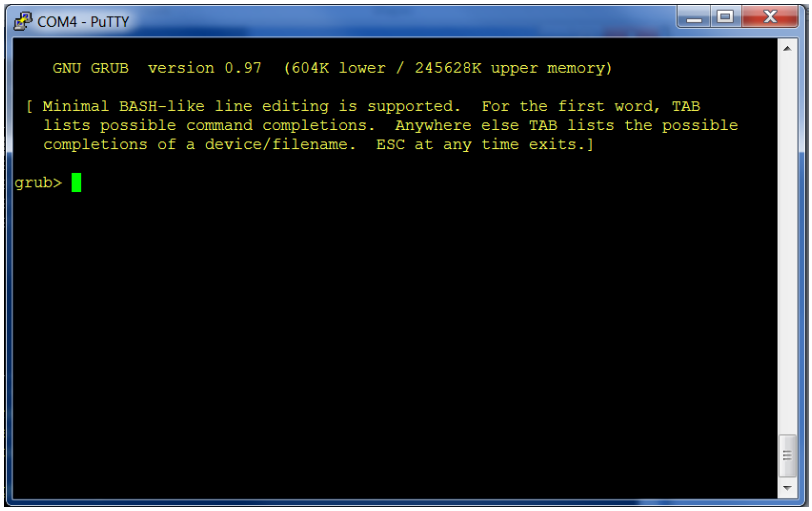
Perform the steps below:

1. Use the files created in Section 6 or download the SPI flash image capsule files as described in Section 2.
2. Copy `CapsuleApp.efi` and `sysimage_nnnnn.cap` to a microSD card 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`.

5. Power on the board. A GNU GRUB boot loader menu is displayed (below). Exit GRUB by entering c and `quit` as shown below.

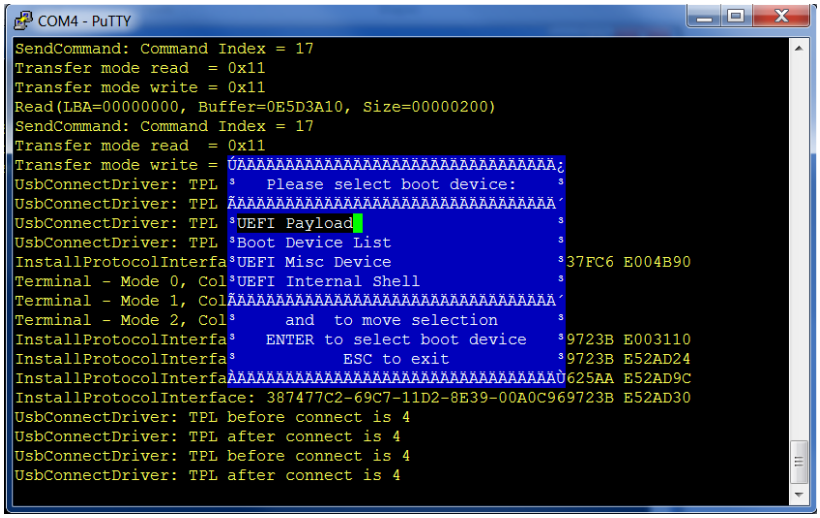


Press c to get a command line.

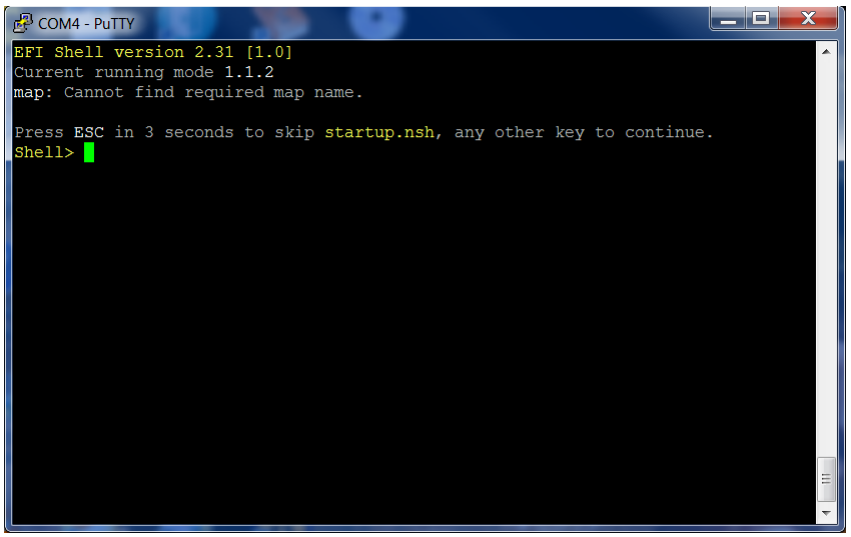


At the GRUB command line, type `quit`.

6. The serial console displays a boot device selection box (below). Select `UEFI Internal Shell`.



You will see a display similar to this:



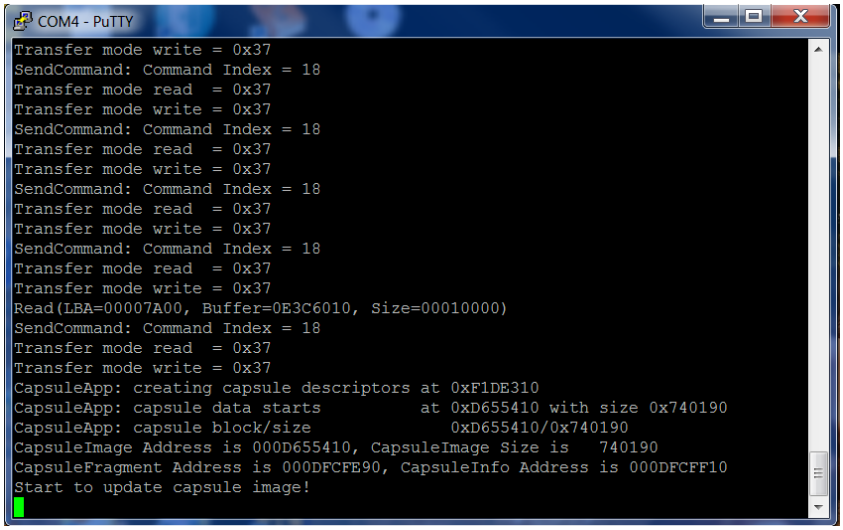
7. You will see a print out, the top line of which looks like this:
   `fs0 :HardDisk - Alias hd7b blk0`

   This is your SD card. To mount it, type: `fs0:`

8. Verify you are using the correct version of `CapsuleApp.efi` by using the `-?` or `–h` option. You **must** use version 1.1 or later.
9. Enter the following command:
   `CapsuleApp.efi sysimage_nnnnn.cap`

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

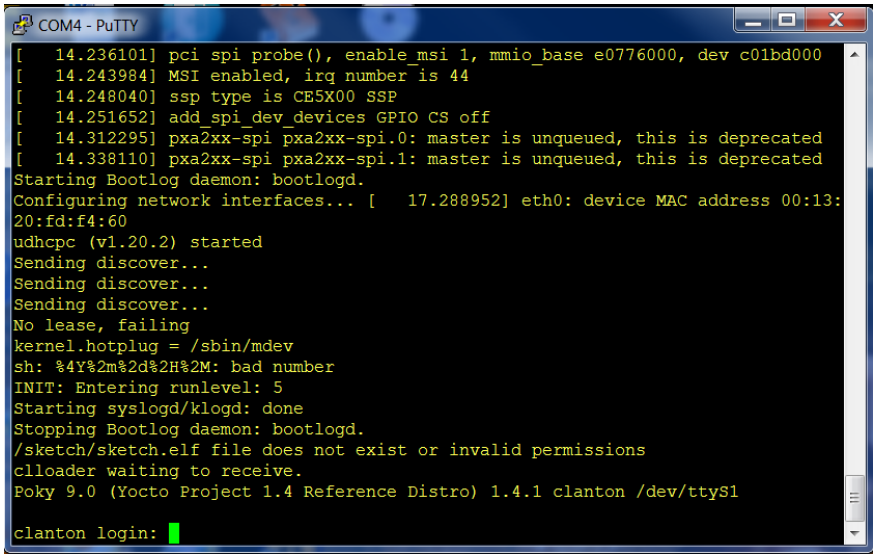
You will see a display similar to this:



The CapsuleApp will update your SPI flash image. This process takes about 2 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.

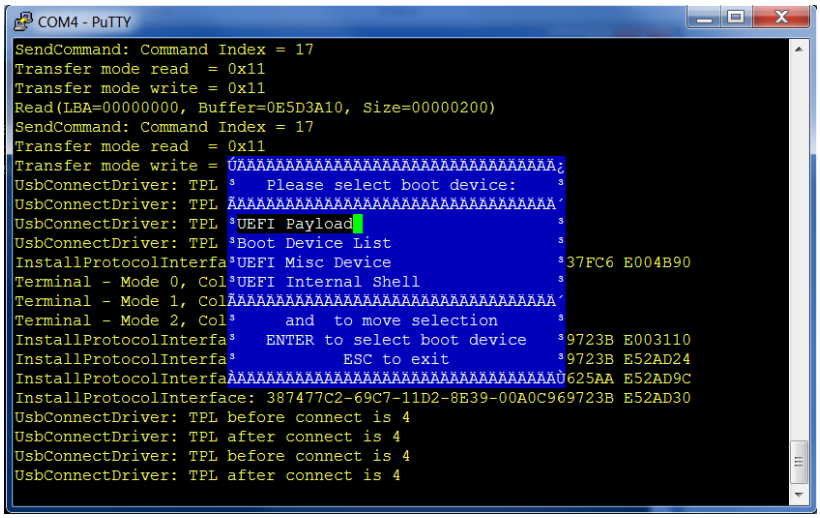
10. When the update completes, the board will automatically reboot. You will see a display similar to this:



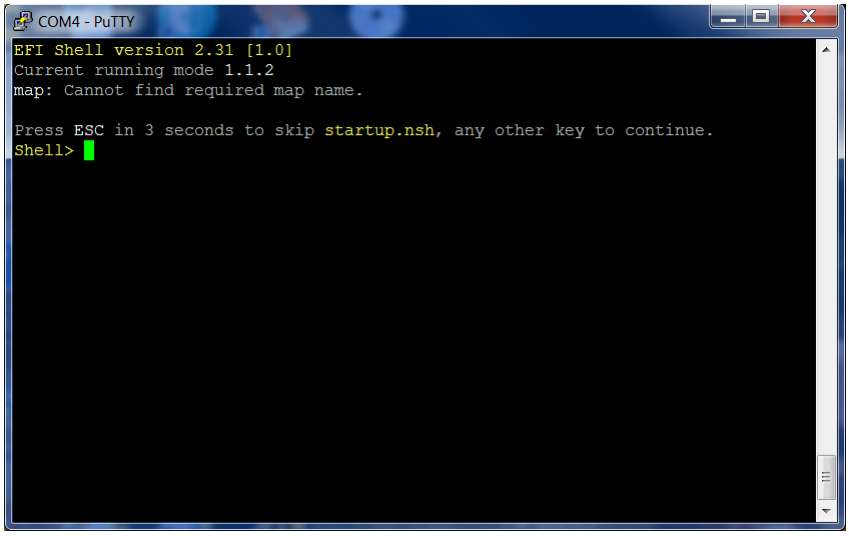
8.1 Secure SKU Boards only

Perform the steps below:

1. Use the files created in Section 6 or download the SPI flash image capsule files as described in Section 2.
2. Copy `CapsuleApp.efi` and `sysimage_nnnnn.cap` to a microSD card 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`.
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:

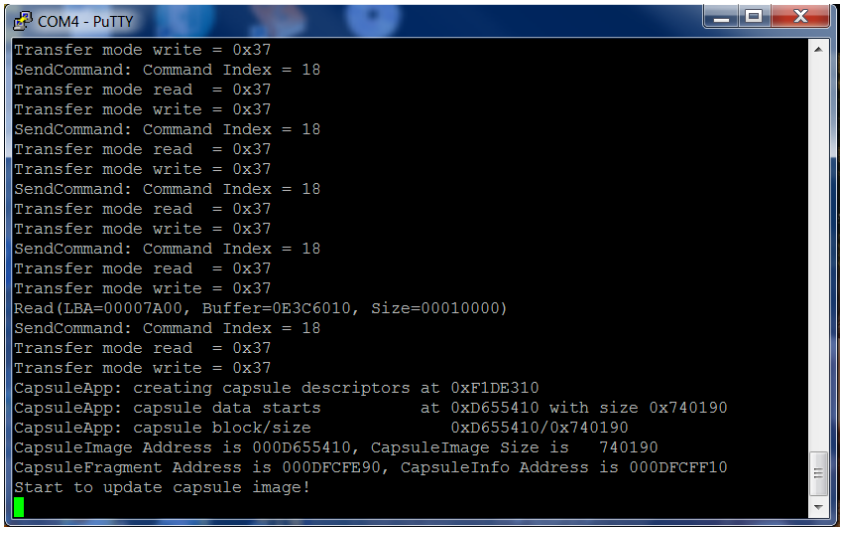


7. You will see a print out, the top line of which looks like this:
   `fs0  :HardDisk - Alias hd7b blk0`

   This is your SD card. To mount it, type: `fs0:`
8. Verify you are using the correct version of `CapsuleApp.efi` by using the `-?` or `–h` option. You **must** use version 1.1 or later.
9. Enter the following command:
   `CapsuleApp.efi sysimage_nnnnn.cap`

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

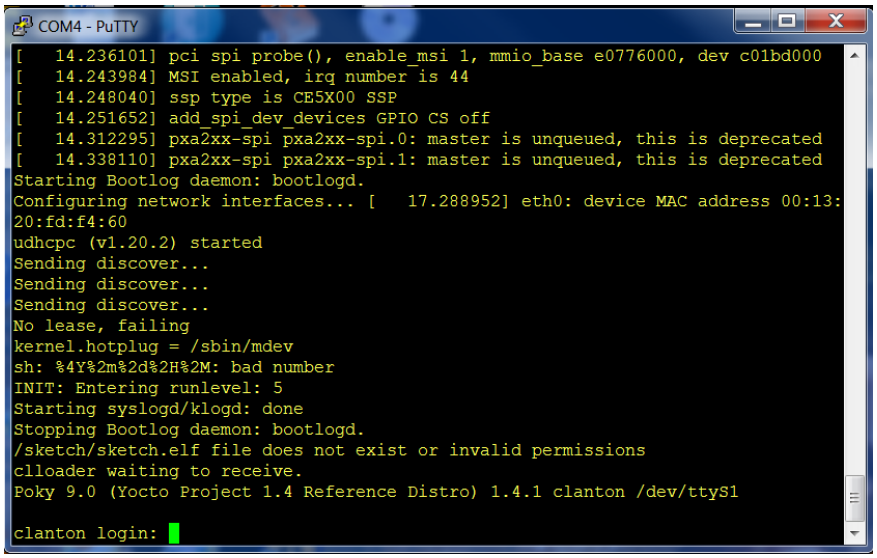
You will see a display similar to this:



The CapsuleApp will update your SPI flash image. This process takes about 2 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:



9 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 7.

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.

10 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.

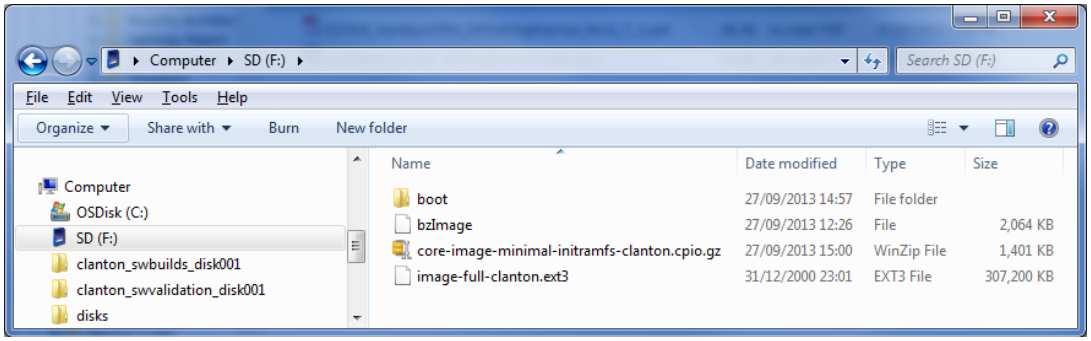
If you are using a Galileo board, this setup allows you to save your sketch to the board, so it will be able to repeat sketches after board power-down. This also enables a persistent `/sketch` folder.

Dependencies: You ran the command `bitbake image-full` in Section 4.

1. The output of the build process in Section 4 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`

   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.

11 Signing files (optional)

This step is optional for most users; it is only needed for secure boot, otherwise it can be skipped.

Kernel and root file system (`bzImage` and `image-full-clanton.cpio.gz`) require signature files for verification.

Open a new terminal session and use the following commands:

```
# cd sysimage_*/sysimage.CP-8M-release
# export LINUX_PATH=../../meta-clanton*/yocto_build/tmp/deploy/images

# ../../spi-flash-tools*/Makefile SRCS_SIGN=$LINUX_PATH/bzImage \
      $LINUX_PATH/bzImage.SVNINDEX=6 $LINUX_PATH/bzImage.signed
# ../../spi-flash-tools*/Makefile \
      SRCS_SIGN=$LINUX_PATH/image-full-clanton.cpio.gz \
      $LINUX_PATH/image-full-clanton.cpio.gz.SVNINDEX=7 \
      $LINUX_PATH/image-full-clanton.cpio.gz.signed
# cd $LINUX_PATH
```

Signature files for kernel and root file system (`bzImage.csbh` and `image-full-clanton.cpio.gz.csbh`, respectively) can be copied onto a USB stick or SD card.

12 Enabling the OpenOCD debugger

Dependencies:

- GCC (tested with version 4.5)
- GNU Make
- `libtool`

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

Use the following commands:

```
# tar -xvf openocd_*.tar.gz
# cd openocd_openocd_*
# ./gitsetup.py
# cd work
# ./configure4clanton.sh
# make
```

Basic usage:

```
# cp work/src/openocd work/tcl
# cd work/tcl
```

Note: Errors in this point may be caused by proxy settings for http URLs.
You may find answers about proxy settings here:
https://wiki.yoctoproject.org/wiki/Working_Behind_a_Network_Proxy

First connect a JTAG debugger to the CRB. Next, run OpenOCD with the correct interface configuration file for your JTAG debugger.

The example that follows is for an "olimex-arm-usb-ocd-h" JTAG debugger:

```
# sudo ./openocd -f interface/olimex-arm-usb-ocd-h.cfg -f \
      board/clanton_board.cfg
```

You can connect to this OpenOCD session using telnet, and issue OpenOCD commands as follows:

```
# telnet localhost 4444
# halt
# resume
```

For more information, see the *Using OpenOCD and Source Level Debug on Intel® Quark SoC X1000 Application Note*.

Revision History

Date	Revision	Description
07 November 2013	002	General updates for software release 0.8.0 including: Added supported boards to list of hardware. Section 6: Changed SPI Flash tools path from `clanton_peak_EDK2` to `Quark_EDKII` Moved Signing files (optional) section to later in the document.
15 October 2013	001	First release with software version 0.7.5.

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 and the Intel logo 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.

Copyright © 2013, Intel Corporation. All rights reserved.