



Intel[®] Quark SoC X1000

Board Support Package (BSP) Build Guide

Release: 0.7.5

15 October 2013



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	Signing files (optional)	7
6	Building the cross compiler toolchain	7
	6.1 Linux* cross compiler	7
	6.2 Windows* cross compiler	8
	6.3 MAC OS* cross compiler	9
7	Creating a flash image for the board	9
8	Defining the platform data file	10
9	Programming flash on the board using serial interface	11
10	Programming flash on the board using DediProg	15
11	Booting the board from SD card	16
12	Enabling the OpenOCD debugger	17



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 hardware and software:

- Intel® Galileo Customer Reference Board (CRB) (Fab D with blue PCB)
- Board Support Package Sources for Intel® Quark SoC X1000 v0.7.5

Before you begin:

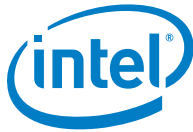
- You need a host PC running Linux*.
- You need an internet connection to download third party sources.
- The build process may require as much as 30 GB of free disk space.
- To program the board you can use:
 - serial interface using `capsule.efi` (see [Section 9](#))
 - DediProg* SF100 SPI Flash Programmer (or equivalent) and the associated flashing software (see [Section 10](#))
 - Intel® Galileo IDE (see the *Intel® Galileo Board Getting Started Guide*)

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

Go here to download the software:

<https://communities.intel.com/community/makers/software/drivers>

This release contains multiple zip files, including:

- Board Support Package (BSP) sources:
 - Board_Support_Package_Sources_for_Intel_Quark_v0.7.5.7z (3.2 MB)
- (Optional) Files for updating SPI flash using serial interface ([Section 9](#)).
 - LITTLE_LINUX_IMAGE_FirmwareUpdate_Intel_Galileo_v0.7.5.7z (5.5 MB)
 - CapsuleApp.efi
 - sysimage_Intel_Galileo_v0.7.5.cap

You will create a binary capsule file as part of the build process in this guide.

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 $\geq 4.6.3$ and `libc6-dev-i386`)
- `gnu-efi` library (tested with version ≥ 3.0)
- GNU Make
- Autotools (`autoconf` and `libtool`)
- Python ≥ 2.6
- `git`
- `xxd`



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'
# [ GNUMEFI_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 `GNUMEFI_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 6](#) to create a Linux image with development tools such as gcc.

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
```

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 Galileo-specific `<target>` commands described below.

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

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`

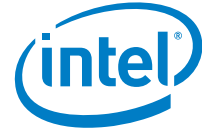
2. If you want to build a more fully-featured Linux to run out of the SD card, then run `bitbake image-full` as shown below:

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

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 Signing files (optional)

This step 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_4M*/sysimage.CP-4M-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.

6 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](#)

6.1 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-eglbc-x86_64-i586-toolchain-1.4.1.sh
```

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

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

Note: The script above 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.

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.

6.2 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`



6.3 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`

7 *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_4M*.tar.gz
```

Extract and install SPI Flash Tools:

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

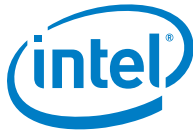
The directory contains a preconfigured `layout.conf` file. This file defines how the various components will be inserted into the final `Flash.bin` file to be flashed onto the board. The `layout.conf` consists of a number of `[sections]` with associated address offsets, file names, and parameters. Each 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 ../../clanton_peak_EDK2_* ../../clanton_peak_EDK2
# ../../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 9](#) or [Section 10](#).

To build a Flash.bin file:

```
# ln -s ../../clanton_peak_EDK2_* ../../clanton_peak_EDK2
# ../../spi-flash-tools*/Makefile
```

The output of this build is a Flash.bin file. Continue with [Section 8](#).

8 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.

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_4M*/sysimage.CP-4M-debug/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 10](#).

9 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.

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 7](#) 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 Galileo board. Set up a serial console session (for example, PuTTY) and connect to the Galileo 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.

```
COM4 - PuTTY
GNU GRUB version 0.97 (604K lower / 245628K upper memory)

+-----+
| Clanton SVP kernel-SPI initrd-SPI IMR-On IO-APIC/HPET NoEMU |
| Clanton SVP kernel-SPI initrd-MassStorage big-rootfs IMR-On IO-APIC/H> |
| Clanton SVP kernel-MassStorage initrd-MassStorage small-rootfs IMR-On> |
| Clanton SVP kernel-MassStorage initrd-MassStorage big-rootfs IMR-On I> |
| Clanton SVP kernel-MassStorage initrd-MassStorage big-rootfs IMR-On I> |
+-----+

Use the ^ and v keys to select which entry is highlighted.
Press enter to boot the selected OS, 'e' to edit the
commands before booting, 'a' to modify the kernel arguments
before booting, or 'c' for a command-line.

The highlighted entry will be booted automatically in 4 seconds.
```

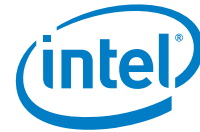
Press `c` to get a command line.

```
COM4 - PuTTY
GNU GRUB version 0.97 (604K lower / 245628K upper memory)

[ Minimal BASH-like line editing is supported. For the first word, TAB
lists possible command completions. Anywhere else TAB lists the possible
completions of a device/filename. ESC at any time exits.]

grub> █
```

At the GRUB command line, type `quit`.



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

```

COM4 - PuTTY
SendCommand: Command Index = 17
Transfer mode read = 0x11
Transfer mode write = 0x11
Read(LBA=00000000, Buffer=0E5D3A10, Size=00000200)
SendCommand: Command Index = 17
Transfer mode read = 0x11
Transfer mode write = 0x11
UsbConnectDriver: TPL Please select boot device:
UsbConnectDriver: TPL
UsbConnectDriver: TPL UEFI Payload
UsbConnectDriver: TPL Boot Device List
InstallProtocolInterface: UEFI Misc Device 37FC6 E004B90
Terminal - Mode 0, Col UEFI Internal Shell
Terminal - Mode 1, Col
Terminal - Mode 2, Col and to move selection
InstallProtocolInterface: ENTER to select boot device 9723B E003110
InstallProtocolInterface: ESC to exit 9723B E52AD24
InstallProtocolInterface: 625AA E52AD9C
InstallProtocolInterface: 387477C2-69C7-11D2-8E39-00A0C969723B E52AD30
UsbConnectDriver: TPL before connect is 4
UsbConnectDriver: TPL after connect is 4
UsbConnectDriver: TPL before connect is 4
UsbConnectDriver: TPL after connect is 4
    
```

You will see a display similar to this:

```

COM4 - PuTTY
EFI Shell version 2.31 [1.0]
Current running mode 1.1.2
map: Cannot find required map name.

Press ESC in 3 seconds to skip startup.nsh, any other key to continue.
Shell>
    
```

- 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:`
- Verify you are using the correct version of `CapsuleApp.efi` by using the `-?` or `-h` option. You **must** use version 1.1 or later.
- 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:

```
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 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, reboot the board by removing and reconnecting the power cable. 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 Programming flash on the board using DediProg

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

Note: These steps require the `Flash+PlatformData.bin` file that was created in [Section 8](#).

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.

When flashing a 4MB image onto an 8MB SPI part it is important that an offset of 4MB is used. In the DediProg application, **Config > Program Configuration > Program from specific address of a chip** should be set to `0x400000`.

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

11 Booting the board from SD card

To boot your board from an SD card, follow these steps. This setup allows you to save your sketch to the Galileo board, so it will be able to repeat sketches after board power-down. This also enables persistent `rootfs` and persistent `/sketch` folder.

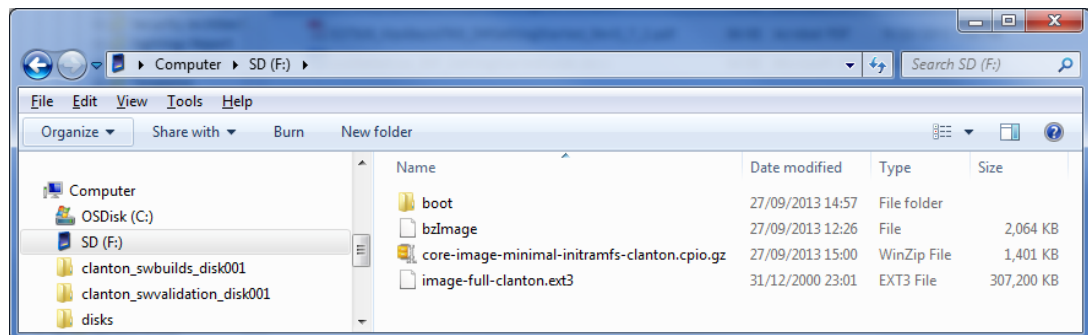
Dependencies: Ran the command `bitbake image-full` in [Section 4](#).

1. The output of the build process in [Section 4](#) is found in `./tmp/dep/dep/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.



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.

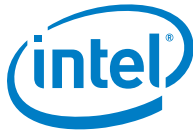
The example that follows is for an "olimex-arm-usb-ocd-h" JTAG debugger:

```
# 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 Clanton Application Note* which is included in this release.



Revision History

Date	Revision	Description
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.