Kips Bay Fab C
Internet of Things (IoT) Kit
BSP Build Guide

Release: 0.6.1

9 October 2013

Intel Confidential

Document Number: 537508, Revision: 0.6.1a

Contents

1 Environment

This guide contains instructions for installing and configuring the Build Support Package (BSP) for the Kips Bay Fab C IoT Kit.

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 30GB of free disk space.

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.

This guide has been tested with Debian* Linux* 7.0 (Wheezy) but may work with other Linux distributions. Debian provides a meta package called `build-essential` that installs a number of compiler tools and libraries.

Please note that it is important to configure your SVN client to use the appropriate HTTP proxy that your company requires. This is usually done in `~/.subversion/servers`. See the documentation for your SVN client.

Note: Intel users may find answers about proxy settings here:
http://wiki.ith.intel.com/display/clanton/Internet+access+-+proxies
or https://opensource.intel.com/linux-wiki/Poky/UsingWithinIntel

2 Software Package

Download the software package here:

https://downloadcenter.intel.com/Detail_Desc.aspx?agr=Y&DwnldID=23262

There is one zip file containing two packages, release 0.6.0 and release 0.6.1.

The password for the zip file is `quarkbsp`

Note: You **must** install release 0.6.0 first, then install release 0.6.1.
Release 0.6.1 is an incremental release, meaning that it contains updates to only two packages (`clanton_linux` and `meta-clanton`).

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 EDKII Firmware

Dependencies:

- Python 2.x
- GCC and G++ (tested with GCC 4.3 and GCC 4.6)
- Subversion client
- uuid-dev
- iasl

The Clanton EDKII BSP is named `clanton_peak_EDK2_<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:

```
tar -xvf clanton_peak_EDK2_*.tar.gz
cd clanton_peak_EDK2*
./svn_setup.py
svn update
```

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

Please note that it is important to configure your SVN client to use the appropriate HTTP proxy that your company requires. This is usually done in `~/.subversion/servers`. See the documentation for your SVN client.

Once `svn update` has completed, use the `clnbuild.sh` script to build the modules. `clnbuild.sh` has the following options:

```
clnbuild.sh [-r32 | -d32 | -clean] [GCC43 | GCC44 | GCC45 | GCC46] [CP]
```

```
-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: Only validated with GCC43.
CP          Build for the Clanton Peak Platform
```

Example usage:

```
./clnbuild.sh -clean # Clean the source code before performing a build
./clnbuild.sh -d32 GCC46 CP # Create a DEBUG build for Clanton Peak based
                 on GCC # version 4.6 (also compatible with GCC 4.7)
```

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 directory:
`Build/ClantonPeakCRBPlatform/<Target>_<Tools>/FV/FlashModules/`

```
<Target> = DEBUG | RELEASE
<Tools> = GCC43 | GCC44 | GCC45 | GCC46
```

Create a symbolic link to the directory where the EDK binaries are placed:

```
cd ../clanton_peak_EDK2/Build/ClantonPeakCRBPlatform/
ln -s DEBUG_GCC* DEBUG_GCC
cd -
```

4 GRUB Bootloader

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

If you intend on running Yocto, then you can skip this section and use the file output by Yocto in this directory:

```
yocto_build/tmp/deploy/images/grub.efi
```

On the other hand, if you are only interested in building a 4M Flash image and not in using Yocto then you have to 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
- xxd

This GRUB build requires the 32 bit gnu-efi library which is included with many Linux distributions. Alternatively, it can be downloaded and compiled as follows:

```
wget http://sourceforge.net/projects/gnu-efi/files/latest/download/\
        gnu-efi_3.0u.orig.tar
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 gnuefi in a non-system location then you need to point GNUEFI_LIBDIR at the location where gnu-efi has been compiled or installed.

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

5 SPI Flash Tools / Sysimage

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
```

On some Linux systems the file `/usr/bin/gmake` is only available as `/usr/bin/make`. If this applies in your case, you need to create a link OR change the first line of the file `spi-flash-tools*/Makefile`.

```
cd sysimage_4M*/sysimage.CP-4M-debug
```

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 a `Flash.bin` file:

```
ln -s ../../clanton_peak_EDK2_* ../../clanton_peak_EDK2
        ../../spi-flash-tools*/Makefile \
      BASETOOLS=../../clanton_peak_EDK2/BaseTools/Bin/CYGWIN_NT-5.1-i686/
```

6 Platform Data

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.

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.

Clanton contains two MACs and each must be configured with one address in the platform ini file, even on boards which have a single ethernet port. For instance, for KipsBay this means that one of the two MAC entries in the ini file should contain a dummy address.

7 Flash Programming

To install the `Flash.bin` on the board, you must use a DediProg* SF100 SPI Flash Programmer and the associated flashing software.

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 `Flash.bin` file you wish to flash.
3. Optionally select the Erase button to erase the contents of the SPI flash.
4. Select the Prog icon to flash the image onto the board.
5. 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.

8 Yocto

Dependencies:

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

Use Yocto to create a root file system and kernel that boots the system from an SD card or USB key.

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 to build the root file system and kernel:

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

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

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

The kernel and root file system (`bzImage` and `image-full-clanton.cpio.gz`, respectively) can be copied onto a USB stick or SD card and booted from grub.

9 Signing kernel and root file system

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.

10 Cross compiler toolchain

The steps to build the cross compiler toolchain are the same as the steps for the Yocto root file system and kernel build as outlined above, with the exception of the bitbake command.

To build the tool chain, open a new terminal session and follow the steps in Section 8 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` 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`

11 OpenOCD

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 Clanton OpenOCD patch.

Use the following commands:

```
tar -xvf openocd_*.tar.gz
cd openocd_openocd_*
./gitsetup.py
```

```
./configure4clanton.sh
make
```

Basic usage:

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

First connect a JTAG debugger to Clanton. 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
9 October 2013	0.6.1a	No technical changes, updates include: • Changed subtitle to “BSP Build Guide” to avoid confusion with other documents. • Updated URL for downloading BSP sources in Section 2.
17 September 2013	0.6.1	Combined release 0.6.0 and 0.6.1 installation instructions.