Intel® Edison Board Support Package

User Guide

September 2014

Revision 001

Document Number: 331188-001

Contents

Figures

Revision History

Revision	Description	Date
ww26	Initial release.	July 7, 2014
ww32	Improved section about adding external recipes.	August 4, 2014
ww36	Corrected code example in chapter 4.	September 5, 2014
001	First public release.	September 9, 2014

§

1 Introduction

This document is for software and system engineers who are building and customizing images, kernels, and native SDKs for the Intel® Edison Development Platform. Precompiled versions of the BSP are available on the Intel website. Users who don't want to modify the default images don't need to read this document.

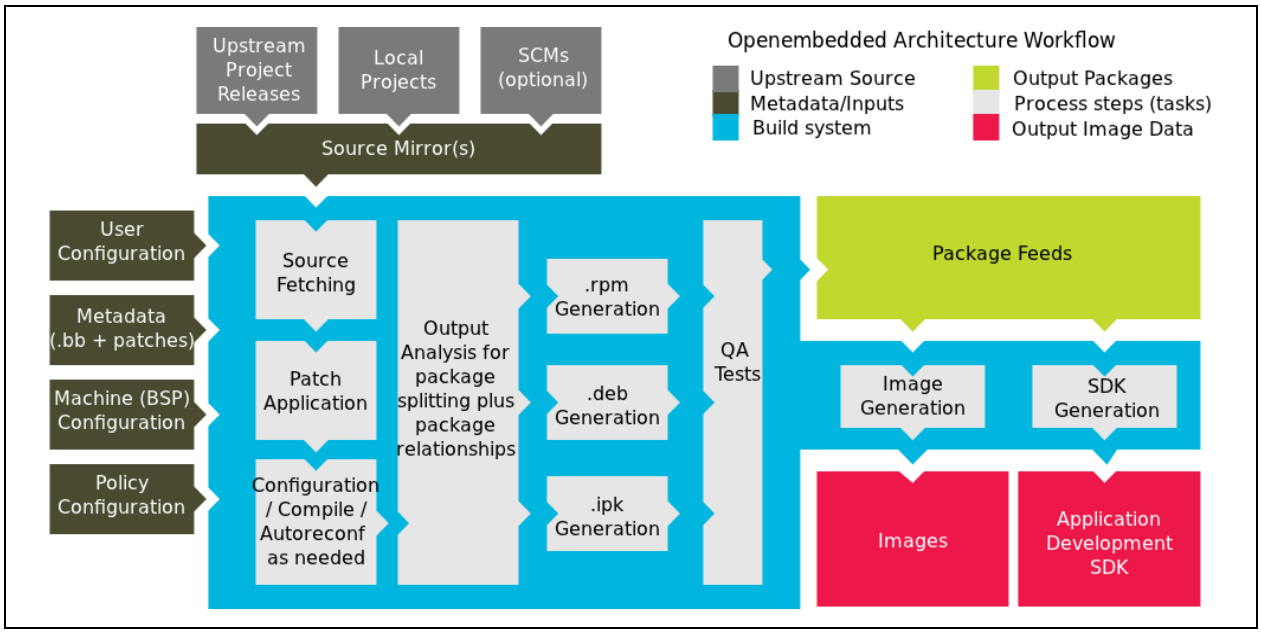
The Intel® Edison Board Support Package offers these features:

- Kernel image based on Linux kernel 3.10.17
- U-boot second stage bootloader
- Bluetooth and Wi-Fi connectivity
- Intel cloud connectivity middleware
- Many base Linux packages provided by the Yocto project

1.1 The Yocto Project

The standard Linux OS shipped on the Edison device is based on Yocto. The Yocto Project is an open source collaboration project that provides templates, tools, and methods to help you create custom Linux-based systems for embedded products.

Figure 1. Building an image



The Edison BSP source package is the set of Yocto source files necessary to generate a Linux image ready to run on the Edison board. It contains:

- The set of Yocto recipes describing the process for building a Linux kernel, a bootloader, and a *rootfs*, which together form the bootable images ready to flash on a device.
- The set of Yocto recipes necessary for creating a Software Developer Kit (SDK) and a cross-compiling tool chain that developers can use to create native applications for Edison.

Note: For details on the Yocto project, consult the documentation on the Yocto website. (See section 1.2.)

1.2 References

Reference	Name	Number/location
331188	Intel® Edison Board Support Package User Guide	(This document)
331189	Intel® Edison Module Hardware Guide	
331190	Intel® Edison Breakout Board Hardware Guide	
331191	Intel® Edison Kit for Arduino* Hardware Guide	
331192	Intel® Edison Native Application Guide	
331193	Intel® Edison Quick Start Guide	
[RN]	Intel® Edison Board Support Package Release Notes	
[GSG]	Intel® Edison Getting Started Guide	
[YPQSG]	Yocto Project Quick Start Guide	*http://www.yoctoproject.org/docs/current/yocto-project-qs/yocto-project-qs.html*
[YDM]	Yocto Developer Manual	*http://www.yoctoproject.org/docs/current/dev-manual/dev-manual.html*
[YKDM]	Yocto Kernel Developer Manual	*http://www.yoctoproject.org/docs/latest/kernel-dev/kernel-dev.html*

1.3 Terminology

Term	Definition
SSH	Secure shell
FTP	File Transfer Protocol
GDB	GNU debugger

§

2 Building a Standard Edison Image

Building a standard Edison image requires downloading and installing several prerequisite packages. These instructions are valid for a recent Ubuntu Linux distribution and should be valid for other distributions with minor changes.

Note: Make sure your working directory is not part of an encrypted filesystem, such as **eCryptFS**. Because encrypted filesystems restrict file length, the build will fail.

To build a standard Edison image, do the following:

1. Install the prequisite packages with the following command:

```
sudo apt-get install build-essential git diffstat gawk chrpath texinfo libtool
gcc-multilib
```

2. Download the BSP source package *edison-src.tgz*. The package includes the full Yocto environment, and Edison-specific Yocto recipes to build the image (including the Linux kernel), a bootloader, and all necessary packages. Download the BSP source package to your working directory and decompress it.

```
tar xvf edison-src.tgz
cd edison-src/
```

3. Use the *setup.sh* script to initialize the build environment for Edison. Optionally, you can move your download and build cache (also called *sstate*) directories from the default location under the build directory, using the *--dl_dir* and *--sstate_dir* options. Doing this will make it easier to share this data between build environments, and allow much faster build and download time when rebuilding the full image, even after a full manual cleanup (by means of deleting everything under your build directory).

```
./device-software/setup.sh --dl_dir=/path/bitbake_download_dir  --
sstate_dir=/path/bitbake_sstate_dir
```

4. Configure the shell environment with the *source* command below. After the command executes, the directory changes to the *edison-src/build* folder.

```
source poky/oe-init-build-env
```

5. Now you are ready to build the full Edison image with the *bitbake* command:

```
bitbake edison-image
```

Building all the packages from scratch can take up to 5 or 6 hours, depending on your host. After the first build (provided you have not done any major cleanups), you can expect much faster rebuilds, depending on your host and the amount of changes. When the bitbake process completes, images to flash are created in the *edison-src/build/tmp/deploy/images* directory. To simplify the flash procedure, run the script below to copy the necessary files to the *build/toFlash* directory.

```
./edison-src/device-software/utils/flash/postBuild.sh
```

The images are ready to flash on the Intel® Edison Development Board. Refer to *Intel® Edison Quick Start Guide* for details on the flashing procedure.

2.1 Build the Edison native SDK

To cross-compile native applications for your image, you must generate an SDK containing a cross-compiler toolchain and sysroot. You can generate a full SDK for the Edison Development Board with the following command:

```
bitbake edison-image -c populate_sdk
```

When the bitbake process completes, the SDK installer script is created:

```
ls ../edison-src/build/tmp/deploy/sdk

poky-edison-eglibc-i686-edison-image-core2-32-toolchain-1.6.sh
```

Refer to the [NAG] for SDK installation and Eclipse integration instructions, and to start native application development.

§

3 Creating Custom Edison Images

In this section, we describe how to customize standard Linux images for Intel® Edison.

3.1 Adding standard Yocto packages in the image

Yocto comes with a large set of recipes allowing you to simply add packages to our image. The available packages are on *http://packages.yoctoproject.org*. In order to add a package to our image, you simply need to add it to the *IMAGE_INSTALL* variable. For example, if you want to add the lib PNG to the image, add the following line to the *edison-src/device-software/meta-edison-distro/recipes-core/images/edison-image.bb* file:

```
IMAGE_INSTALL += "libpng"
```

Now rebuild the image to have *libpng* included in it.

Note: If you need to add patches to existing upstream sources, consult the Yocto documentation [YDM].

3.2 Excluding packages from the image

To exclude unnecessary packages from the image, either remove the matching entry from the *IMAGE_INSTALL* variable (see previous section), or add the package name to the *PACKAGE_EXCLUDE* variable in the *build/conf/local.conf* file.

```
PACKAGE_EXCLUDE = "package1 package2"
```

3.3 Add third-party packages to the image

If Yocto does not provide a package you need by default, chances are good that someone else has created a Yocto recipe for it. In this section, we will add a set of Yocto recipes (from a third-party Yocto layer named *meta-oe*) to the Edison source. The recipes contained in this layer allow you to add many packages in a custom Edison image. The *meta-oe* layer can be found at this OpenEmbedded GitHub location: *https://github.com/openembedded/meta-oe*.

As an example, the *opencv* library will be added to the image. The example assumes a standard image has been created by running the *setup.sh* script and *bitbake edison-image* as described in the previous sections.

1. Get the OpenEmbedded Yocto layer collection from GitHub. We use the "daisy" branch matching the version of Yocto that is used by edison.

   ```
   cd edison-src/device-software
   git clone https://github.com/openembedded/meta-openembedded.git
   cd meta-openembedded
   git checkout daisy
   ```

2. Tell `bitbake` to look for recipes contained in the new `meta-oe/` layer. Edit the *build/conf/bblayer.conf* file and append the path to the new layer into the `BBLAYERS` variable:

   ```
   BBLAYERS ?= " \
     [..]
     Full/path/to/edison-src/device-software/meta-openembedded/meta-oe \ "
   ```

3. You now can add any recipe provided by the new *meta-oe* layer to your image. As in section 3.1, to add *opencv* to the image, simply add it to the *IMAGE_INSTALL* variable. You can do this in the *build/conf/local.conf* file, for example. In the particular case of *opencv*, to avoid bringing too many dependencies, you should also redefine a specific variable so that the library is built without *gtk* support:

   ```
   IMAGE_INSTALL += "opencv"
   PACKAGECONFIG_pn-opencv="eigen jpeg libav png tiff v4l"
   ```

4. Save the file and rebuild the image as follows:

```
cd edison-src
source poky/oe-init-build-env
bitbake edison-image
```

3.4 Write a Yocto recipe from scratch

It is also possible to create your own Yocto recipes from scratch and add them to the image. This section describes the required steps to add a *hello_world* C program to our image. The GNU *hello_world* is a real project that you can download from *http://ftp.gnu.org/gnu/hello/hello-2.7.tar.gz*.

1. The first step is to tell *bitbake* where to download the code, and how to build the package. This is done by adding a new recipe (`.bb`) file in the right directory. To do this, create the recipe file *hello_2.7.bb* in the *device-software/meta-edison-distro/recipes-support/hello* directory, with the following content:

```
DESCRIPTION = "GNU Helloworld application"
LICENSE = "GPLv3+"
LIC_FILES_CHKSUM = "file://COPYING;md5=d32239bcb673463ab874e80d47fae504"
SRC_URI = "${GNU_MIRROR}/hello/hello-${PV}.tar.gz"
SRC_URI[md5sum] = "fc01b05c7f943d3c42124942a2a9bb3a"

inherit autotools gettext
```

Note: As the *hello_world* project makes use of the autotools, it is enough to inherit the autotool yocto class to tell bitbake how to configure and build the project. Refer to the Yocto documentation for details on the *.bb* syntax.

The *hello world* recipe is ready, but you still need to add it to your image. To do so, add the following line to the *edison-src/device-software/meta-edison-distro/recipes-core/images/edison-image.bb* file:

```
IMAGE_INSTALL += "hello"
```

Then rebuild the image:

```
bitbake edison-image
```

3.5 Add a recipe for a systemd service

Developers may choose to add their own application as a service to Edison. On Edison, services are special applications that run in the background. They are managed by *systemd*, a system and service manager for Linux.

A *systemd* service is described by a .service file that needs to be deployed on the Edison board usually in */lib/systemd/system*. This service file contains information on how and when to start the service, which are its dependencies. etc.

Note: Refer to *systemd* documentation *http://www.freedesktop.org/wiki/Software/systemd* for an overview of the base *systemd* concepts, and a description of the associated tools.

The Edison BSP Source includes a sample recipe for creating a *systemd* service application using Yocto. The sample is located in the *meta-edison-distro/recipes-support/watchdog-sample* folder.

A system service is described by a *.service* file. Refer to the sample file *watchdog-sample.service* at: *http://www.freedesktop.org/software/systemd/man/systemd.service.html*.

To deploy the service from a Yocto recipe, you need to inherit the Yocto systemd class. Refer to *http://www.yoctoproject.org/docs/current/ref-manual/ref-manual.html#ref-classes-systemd*.

§

4 Customizing the Linux Kernel

Customizing the kernel is important on embedded systems for making new devices and sensors.

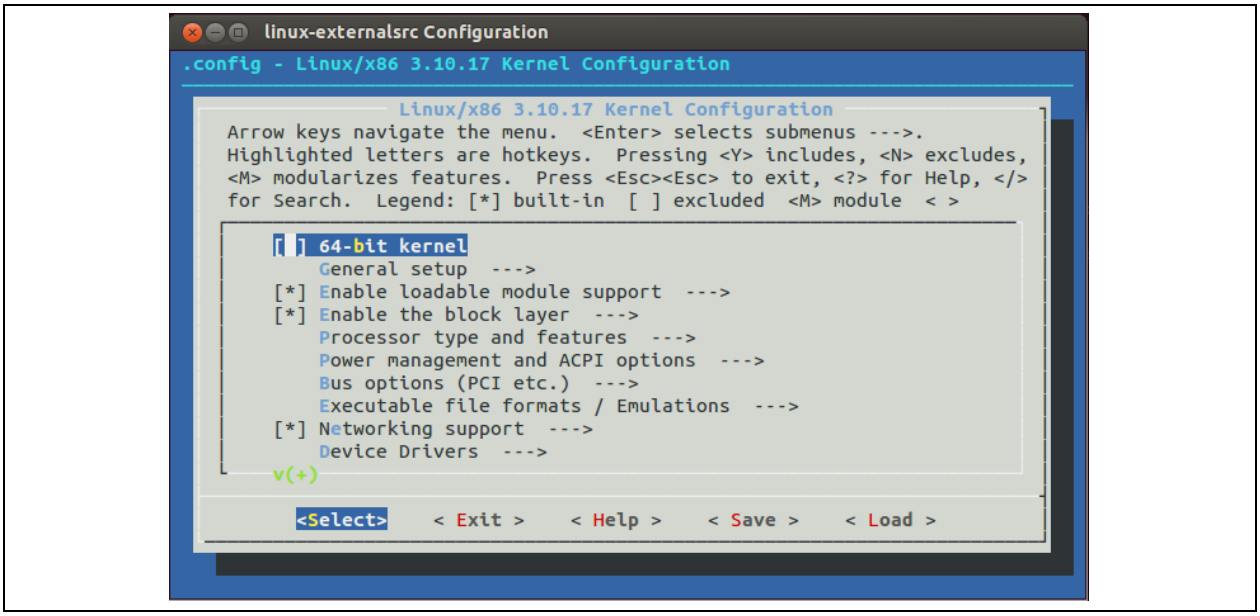
This section contains a brief overview of making kernel modifications. For more detailed information, see the Yocto Kernel Developer Manual [YKDM] at: *http://www.yoctoproject.org/docs/latest/kernel-dev/kernel-dev.html*. Check it out for additional ways of configuring the kernel, for example through using more compact and modular configuration fragments. The approach described below is good for ad-hoc modifications while configuration fragments are shorter than full kernel configuration, and it allows you to create and distribute your own Yocto recipes for modifying specific kernel features.

The base kernel config file is delivered with *edison-src.tar.gz* and is located in the *edison-src/device-software/meta-edison/recipes-kernel/linux/files/defconfig* file.

The *menuconfig* tool provides an easy interactive method with which to define kernel configurations. For general information on *menuconfig*, see *http://en.wikipedia.org/wiki/Menuconfig*. The following command opens the *menuconfig* terminal for configurations:

```
bitbake virtual/kernel -c menuconfig
```

Figure 2 Linux kernel configuration



When the configuration is completed, replace *defconfig* with *.config*, then rename it back to *defconfig*. We also suggest taking a backup of the *defconfig* file. Force bitbake to copy the modified *defconfig* file to the actual build directory. Then the new image with modified kernel is ready to build.

```
cp /build/tmp/work/edison-poky-linux/linux-
yocto/3.10.17+gitAUTOINC+6ad20f049a_c03195ed6e-r0/linux-edison-standard-
build/.config build/tmp/work/edison-poky-linux/linux-
yocto/3.10.17+gitAUTOINC+6ad20f049a_c03195ed6e-
r0/linux/arch/x86/configs/i386_edison_defconfig

bitbake virtual/kernel -c configure -f -v
bitbake edison-image
```

§