Hands-on Intro Lab

Yocto Project Dev Day

Barcelona, 2012

Scott Garman <scott.a.garman@intel.com>

Introduction

Welcome to the **Yocto Project Hands-on Intro Lab**! The lab instructor will give you the login credentials to log into your lab computer when the hands-on portion of the session will begin.

Your system has been setup with the 7.0.1 (“Denzil”) release of the Poky source repository under **`~/poky-denzil-7.0.1/`**. The **`downloads`** and **`sstate-cache`** directories have been pre-populated to reduce the time spent on image assembly.

Yocto Project BSP layers for the Intel Fish River Island 2 and Texas Instruments BeagleBone have also been downloaded to the **~/meta-intel/** and **~/meta-ti/** directories, respectively.

This lab assumes you are comfortable navigating a GNOME desktop-based Linux environment and issuing shell commands. If you need help at any time, please raise your hand, and one of the lab assistants will be happy to provide additional guidance.

It is natural that during hands-on labs, different people will have different speeds for completing the exercises. Because this is an introductory lab that requires no prior experience with the Yocto Project, please don't be shy about asking questions or asking for help – the instructor and our lab assistants are here to help you.

If you find yourself completing the lab exercises much faster than the rest of the class, please be respectful and do not jump ahead to the next exercise unless specifically instructed to. In some cases, additional material has been included for you to investigate if you complete exercises early.

Exercise 1: Poky Directory Tree Layout

In this exercise, you will follow along with the presenter in examining where different kinds of metadata are kept in the Poky directory tree. You can either use a graphical file browser (**`Places → Home Folder`**), or navigate the tree using a terminal window from the command line, whichever you are more comfortable with.

After completing this exercise, you should be able to answer the following questions:

Where in the Poky tree can you find BitBake classes?:

Where is the BitBake recipe for openssh stored?:

Where is the machine definition for qemumips stored?:

Tip: When hunting for individual recipes, the **find** command can be quite useful, e.g:

find ~/poky-denzil-7.0.1/meta/ -name "*ssl*"

Exercise 2: Examining Recipes

In this exercise, you will follow along with the presenter in examining a few Poky recipes. You can either use a graphical file browser (**`Places → Home Folder`**) and right-click on files to open them with the Gedit text editor, or navigate the filesystem using a terminal window and text editor. Gedit, vim, and emacs are popular text editors already installed on the lab machines.

Paths to the recipes we'll be looking at are:

- **~/poky-denzil-7.0.1/meta/recipes-extended/bc/bc_1.06.bb**
  - Uses **`LIC_FILES_CHKSUM`** and **`SRC_URI`** checksums
  - Note the **`DEPENDS`** build dependency declaration
- **~/poky-denzil-7.0.1/meta/recipes-multimedia/flac/flac_1.2.1.bb**
  - Includes custom patches to apply to the sources
  - Customizes autoconf configure options (**EXTRA_OECONF**)
  - Overrides the **`do_configure()`** build step
  - Breaks up output into multiple binary packages (**FILES_packagename**)

- **~/poky-denzil-7.0.1/meta/recipes-connectivity/ofono/ofono_1.5.bb**
    - Splits recipe into common .inc file to share metadata between release version recipe (**ofono_1.5.bb**) and git recipe (**ofono_git.bb**).
    - Sets a conditional build **DEPENDS** based on a distro feature (bluetooth)
    - Sets up an init service (**INITSCRIPT_NAME, INITSCRIPT_PARAMS**)

Exercise 3: Building a Linux Image

Now you'll build your first Linux OS image using the Yocto Project. OpenEmbedded-core includes a few reference images:

- **core-image-minimal** is a basic filesystem which boots up to a console login, and does not include any network services or graphical desktop environments.
- **core-image-sato** is a demonstration image which boots up to the Sato desktop environment, which has been used for PDA-type devices. There are Sato applications for browsing the web, playing music, a calendar and task list, and more.
- **core-image-lsb** is a reference image which includes all of the system software necessary to implement the Linux Standard Base (LSB) specification.

You're going to build **core-image-minimal**.

First you must specify what machine target you're building in your **`~/poky-denzil-7.0.1/build/conf/local.conf`** file. Open this file in your text editor of choice and confirm that the **`MACHINE`** variable is set to **`qemux86`** (this should be the default, so in this case you won't need to change it). Notice what other variables also get set within local.conf.

Now open a terminal window and run the following commands:

```
cd ~/poky-denzil-7.0.1
source oe-init-build-env
bitbake core-image-minimal
```

Because your sstate-cache directory has already been populated with precompiled sstate cache packages of everything that core-image-minimal needs, this process should only take a few minutes while the root filesystem is generated.

When done, you should find the generated image files under **`build/tmp/deploy/images/`**.

Note that multiple files were generated, including a **.ext3** image file and a compressed tarball (**.tar.bz2**) of the rootfs files.

Fun Fact: The tarball version of the root filesystem is handy when you want to extract the rootfs someplace, and then point QEMU to mount its rootfs over NFS. The Yocto Project includes a userspace NFS server and can automate this with the **runqemu** script.

If you finish early:

In addition to bitbake'ing image recipe targets, you can also bitbake individual recipes. Running these examples will show you what bitbake output looks like when it actually has to run most of the buildsteps in building a recipe (the **do_fetch** tasks were done already, since network access at the conference was not assumed to be reliable).

You can run a specific task by passing the **-c** option to bitbake. The **cleansstate** task deletes the **sstate-cache** and **WORKDIR** for the given target, allowing you to rebuild a package from scratch. Let's rebuild the bzip2 file compression utility:

```
bitbake bzip2 -c cleansstate
bitbake bzip2
```

Exercise 4: Booting Your Linux Image with QEMU

Since we built an image using a **MACHINE** type that QEMU supports, we can boot it using the **runqemu** script.

This **runqemu** script was designed to make it as easy as possible to boot our reference images – all you have to do is tell the script which QEMU architecture you wish to boot, and it will try to locate the appropriate kernel and root filesystem from your **images** directory. Give it a try:

```
runqemu qemux86
```

Once your Linux image has booted, you can log into it with the username **root**, and just hit **Enter** to provide an empty password. Feel free to explore the filesystem – the **df -h** command will show you how big the filesystem is (about 8 MB).

Fun Fact: As of the 1.2 release of the Yocto Project (April 2012), we now include a reference distribution policy and layer called **`poky-tiny`** that fits the kernel and rootfs within 4 MB.

Once you're done testing your image, shut down QEMU.

From within the QEMU login session:

```
shutdown -h now
```

If you finish early:

Take a look at the contents of the the following recipes:

- **`~/poky-denzil-7.0.1/meta/recipes-sato/leafpad/`**
- **`~/poky-denzil-7.0.1/meta/recipes-devtools/e2fsprogs/`**
- **`~/poky-denzil-7.0.1/meta/recipes-support/curl/`**

Are there variables used in these recipes that you don't recognize? Look them up in the Yocto Project Reference Manual. If you open up the Firefox web browser on your lab computer, it should automatically load a local copy of the Yocto Project Reference Manual. **`Appendix F`** of this manual defines commonly used variables in recipes. You can also search the entire document by pressing **`Ctrl-F`** and typing your search query.

Exercise 5: Creating a Custom Layer

The most maintainable way to make customizations with Poky is not to change the recipe files directly, but to store your changes in a custom layer. Let's create a layer to house a custom image recipe we'll be developing later on. The name of this layer will be **`meta-ypdd`**.

Start by creating the directory structure of a basic layer. To do this, open a new terminal window, and run:

```
mkdir -p meta-ypdd/conf
mkdir -p meta-ypdd/recipes-ypdd/images
```

Now navigate into your **meta-ypdd/conf** directory and create a new file named **`layer.conf`** with the following in it:

```
BBPATH := “${BBPATH}:${LAYERDIR}”

BBFILES := “${BBFILES} ${LAYERDIR}/recipes-*/*/*.bb \
            ${LAYERDIR}/recipes-*/*/*.bbappend”

BBFILE_COLLECTIONS += “ypdd”
BBFILE_PATTERN_ypdd := “^${LAYERDIR}/”
BBFILE_PRIORITY_ypdd = “6”
```

The above variables are required to tell BitBake where it can find recipe files in your custom layer.

If you finish this exercise early, you may continue on to Exercise 6 now.

Exercise 6: Creating a Custom Image Recipe

Our new layer will be used to store a custom image recipe. Let's name our image **`ypdd-image`**. Therefore, we'll want to create an image recipe file **`ypdd-image.bb`** within the **`meta-ypdd/recipes-ypdd/images/`** directory.

Since we want this image recipe to be based on core-image-minimal, let's copy and rename the core-image-minimal image recipe from the Poky tree into our layer.

```
cp ~/poky-denzil-7.0.1/meta/recipes-core/images/core-image-minimal.bb
~/meta-ypdd/recipes-ypdd/images/ypdd-image.bb
```

Note: the above command text is intended to be one continuous line – do not type it out on two separate lines.

Open your newly-created **ypdd-image.bb** recipe, and you should see this:

```
#
# Copyright (C) 2007 OpenedHand Ltd.
#
DESCRIPTION = "A small image just capable of allowing a device to boot."

IMAGE_INSTALL = "task-core-boot ${ROOTFS_PKGMANAGE_BOOTSTRAP} \
${CORE_IMAGE_EXTRA_INSTALL}"

IMAGE_LINGUAS = " "

LICENSE = "MIT"

inherit core-image

IMAGE_ROOTFS_SIZE = "8192"

# remove not needed ipkg informations
ROOTFS_POSTPROCESS_COMMAND += "remove_packaging_data_files ; "
```

Let's add the psplash and dropbear packages to this image by appending them to the **IMAGE_INSTALL** variable, resulting in the file looking like this:

```
#
# Copyright (C) 2007 OpenedHand Ltd.
#
DESCRIPTION = "A small image just capable of allowing a device to boot."

IMAGE_INSTALL = "task-core-boot ${ROOTFS_PKGMANAGE_BOOTSTRAP} \
                 ${CORE_IMAGE_EXTRA_INSTALL} psplash dropbear"

IMAGE_LINGUAS = " "

LICENSE = "MIT"

inherit core-image

IMAGE_ROOTFS_SIZE = "8192"

# remove not needed ipkg informations
ROOTFS_POSTPROCESS_COMMAND += "remove_packaging_data_files ; "
```

This completes Exercise 6. You may now continue on to Exercise 7.

Exercise 7: Build and Boot Your Custom Image

Now we're ready to build our custom image, right?

```
cd ~/poky-denzil-7.0.1
source oe-init-build-env
bitbake ypdd-image
```

You should see:

```
ERROR: Nothing PROVIDES 'ypdd-image'
```

This error means that BitBake was unable to find a recipe named **`ypdd-image`** to build. This is because we need to tell our build configuration to include the custom layer we created when looking for recipes.

So open **`~/poky-denzil-7.0.1/build/conf/bblayers.conf`** in a text editor and add the path to our **meta-ypdd** layer, so the file looks like:

```
# LAYER_CONF_VERSION is increased each time build/conf/bblayers.conf
# changes incompatibly
LCONF_VERSION = “4”

BBFILES ?= “”
BBLAYERS = “ \
  /home/introlab/poky-denzil-7.0.1/meta \
  /home/introlab/poky-denzil-7.0.1/meta-yocto \
  /home/introlab/meta-ypdd \
  “
```

And now you should be able to run

```
bitbake ypdd-image
```

You should see dropbear and psplash get built, and the ypdd-image root filesystem get generated. Verify that the rootfs image files were created in **`~/poky-denzil-7.0.1/build/tmp/deploy/images/`**

Now you're ready to boot your custom Linux image using the **runqemu** script. But since the name of our root filesystem files do not match one of the reference images Yocto supports, we'll need to provide a bit more information to the script – namely, by giving it a path to the rootfs filename we want it to use.

```
runqemu qemux86 tmp/deploy/images/ypdd-image-qemux86.ext3
```

As this image boots up, notice that a graphical Yocto Project logo appears once init starts (psplash works as an init service, so the kernel boot messages will still appear on the screen first. There are other ways of completely silencing this output, but that's outside the scope of this hands-on lab).

Upon logging in (username **root**, no password), confirm that the Dropbear SSH server is running

From within your QEMU session:

```
ps | grep dropbear
```

When you're done exploring your custom image's filesystem, shutdown your QEMU session with the `shutdown -h now` command.

If you finish early:

- Try ssh'ing into the running QEMU session. The default IP address of the running QEMU session should be 192.168.7.2.
- What happens if you increase the `IMAGE_ROOTFS_SIZE` in your `ypdd-image.bb` recipe and rebuild it?
- What does the `remove_packaging_data_files` function in your `ypdd-image.bb` file do? Use `grep` to search through the Bitbake classes directory (`~/poky-denzil-7.0.1/meta/classes`) to find out.

Exercise 8: Boot core-image-minimal on an Intel Fish River Island 2 embedded system

The Intel Fish River Island 2 (FRI2) board is encased in a black box. The Texas Instruments Beaglebone is an exposed embedded board. If you have a BeagleBone at your computer, skip ahead to **Exercise 9**.

To build an image for an embedded board, you need a Yocto Project Board Support Package (BSP) which corresponds that board and the version of Poky that you are working with. The Yocto Project “Denzil” 7.0.1 BSP for the FRI2 has already been downloaded to your computer and has been extracted to **~/meta-intel/.**

To enable this BSP, you need to point your **bblayers.conf** file to the location where the BSP was extracted to, and then set the **MACHINE** variable in your **local.conf** file to the name of the machine you wish to build for, which in this case is **fri2-noemgd**.

Edit your **~/poky-denzil-7.0.1/build/conf/bblayers.conf** file to include the FRI2 BSP as follows:

```
# LAYER_CONF_VERSION is increased each time build/conf/bblayers.conf
# changes incompatibly
LCONF_VERSION = “4”

BBFILES ?= “”
BBLAYERS = “ \
  /home/introlab/poky-denzil-7.0.1/meta \
  /home/introlab/poky-denzil-7.0.1/meta-yocto \
  /home/introlab/meta-intel \
  /home/introlab/meta-intel/meta-fri2 \
  “
```

Fun Fact: The reason that we need to include two layers from meta-intel within our bblayers.conf file is explained in the README file you can find under **meta-intel/meta-fri2/**. This file also explains what the MACHINE name is for the FRI2.

Now, edit your **~/poky-denzil-7.0.1/build/conf/local.conf** file and change the **MACHINE** variable from

```
MACHINE ??= “qemux86”
```

to

```
MACHINE ??= “fri2-noemgd”
```

Now we are ready to build core-image-minimal for the FRI2:

```
bitbake core-image-minimal
```

Again, your lab computer includes a populated sstate-cache where packages have already been built for this MACHINE, so it should only take a few minutes for the rootfs to be generated.

Once the image has been created, you should find it in **tmp/deploy/images/core-image-minimal-fri2-noemgd.hddimg**. The **.hddimg** file extension means that this is a bootable image that you can write to a USB thumbdrive and boot the FRI2 using it.

Now plug a USB thumbdrive into your computer, and use the ddimage script to write the .hddimge file to the USB drive. The following command is intended to be on the same line:

```
sudo ddimage tmp/deploy/images/core-image-minimal-fri2-noemgd.hddimg
/dev/sdb
```

You should then see the following output in the Device Details section:

```
Device details
==============
  device: /dev/sdb
  vendor: General
   model: USB Flash Disk
    size: 4008706048 bytes
```

WARNING! If you do not see "USB Flash Disk" on the model line, stop now and raise your hand to ask for help from an instructor. If this command is run incorrectly, it is possible to overwrite your lab computer's hard drive and make the system unusable.

If the model line above does say "USB Flash Disk", then it is safe to confirm the write operation by pressing 'y'. When it completes, remove the flash thumbdrive from your computer and plug it into the FRI2 box (there is only one full-size USB port that the flash thumbdrive will fit into on the FRI2).

A USB cable should already be connected between your computer and the top of the FRI2 – this is the serial connection you will use to see the system boot.

Connect the power cable to the FRI2. The power cable connects just above where you have inserted the USB thumbdrive.

Now open a new terminal window and start **minicom**, a terminal emulator which has been preconfigured to connect to the FRI2's serial output (/dev/ttyUSB0, 115200 baud, N81).

```
sudo minicom
```

Within minicom, press **Enter** to start the boot process and get to a login prompt. You should be able to log in as **root** with an empty password and inspect the running OS.

You should now skip Exercise 9 and move ahead to the **Conclusion**.

Exercise 9: Boot core-image-minimal on a Texas Instruments Beaglebone embedded system

The Intel Fish River Island 2 (FRI2) board is encased in a black box. The Texas Instruments BeagleBone is an exposed embedded board. If you have an Intel FRI2 at your computer, go back to **Exercise 8**.

To build an image for an embedded board, you need a Yocto Project Board Support Package (BSP) which corresponds that board and the version of Poky that you are working with. The Yocto Project “Denzil” 7.0.1 BSP for the BeagleBone has already been downloaded to your computer and has been extracted to **~/meta-ti/.**

To enable this BSP, you need to point your **bblayers.conf** file to the location where the BSP was extracted to, and then set the **MACHINE** variable in your **local.conf** file to the name of the machine you wish to build for, which in this case is **beaglebone**.

Edit your **~/poky-denzil-7.0.1/build/conf/bblayers.conf** file to include the BeagleBone BSP as follows:

```
# LAYER_CONF_VERSION is increased each time build/conf/bblayers.conf
# changes incompatibly
LCONF_VERSION = “4”

BBFILES ?= “”
BBLAYERS = “ \
  /home/introlab/poky-denzil-7.0.1/meta \
  /home/introlab/poky-denzil-7.0.1/meta-yocto \
  /home/introlab/meta-ti \
  “
```

Now, edit your **~/poky-denzil-7.0.1/build/conf/local.conf** file and change the **MACHINE** variable from

```
MACHINE ??= “qemux86”
```

to

```
MACHINE ??= “beaglebone”
```

Now we are ready to build core-image-minimal for the BeagleBone:

```
bitbake core-image-minimal
```

Again, your lab computer includes a populated sstate-cache where packages have already been built for this MACHINE, so it should only take a few minutes for the rootfs to be generated.

Once the image has been created, you should find it in **tmp/deploy/images/core-image-minimal-beaglebone.tar.bz2**. This filesystem needs to be extracted onto a microSD card which can then be booted from on the BeagleBone.

To simplify this process, a script has been included on your system named **sdwriter**.

The BeagleBone at your lab computer should have a 4 GB microSD card already in it. The card is located at the opposite end of the board from the ethernet jack. Press down on the card and it should eject from the card holder. Now place the microSD card into the card reader/writer which is attached to your lab computer. Finally, use the **sdwriter** script to extract the image onto the microSD card.

WARNING! Please stop here and confirm with the lab instructor what device name to use as the second argument to the sdwriter script. If this command is run incorrectly, it is possible to overwrite your your lab computer's hard drive and make the system unusable.

```
sudo sdwriter tmp/deploy/images/core-image-minimal-beaglebone.tar.bz2
/dev/sdb
```

The writing process will take a minute or two to complete. When it is done, remove the microSD card from the reader/writer and insert it back into the BeagleBone.

Now connect the USB power cable to the BeagleBone. The USB cable uses a mini-USB connector which plugs into the BeagleBone just to the left of the ethernet port. This connector is mounted to the underside of the circuit board.

Now open a new terminal window and start **minicom**, a terminal emulator which has been preconfigured to connect to the BeagleBone's serial output (/dev/ttyUSB0, 115200 baud, N81).

```
sudo minicom
```

Once minicom is running, you can press the reset button on the BeagleBone to ensure the boot process occurs while you can view it. The reset button is a tiny button near the mini-USB connector, but on the top side of the circuit board.

You should now be able to watch as the BeagleBone boots and brings you to a login prompt. You can log in as **root** with an empty password and inspect the running OS.

Conclusion

Thanks for participating in this hands-on lab session! By now you should have a better understanding of key concepts and workflow processes of using the Yocto Project to create custom Linux images.

For more information, please visit our web site and join our mailing lists – www.yoctoproject.org.

The author of this lab session, Scott Garman, would love any feedback you have to clarify or otherwise improve this intro hands-on lab. Please send such feedback to scott.a.garman@intel.com.