Yocto Project Eclipse plug-in and Developer Tools Hands-on Lab

Yocto Project Developer Day

San Francisco, 2013

Jessica Zhang<jessica.zhang@intel.com>

Introduction

Welcome to the Yocto Project Eclipse plug-in and Developers Tools Hands-on Lab! During this session you will learn how to use Yocto Project ADT, interact with Yocto Project developer tools, i.e. hob and yocto-bsp, within the Yocto Project Eclipse plug-in. That allows you to effectively achieve your embedded software development needs for applications or build customization.

Development System Basic Setup

This hands-on lab was designed to be completed on a computer running the Ubuntu 12.04, “Precise Pangolin” operating system. While this specific release of Ubuntu is recommended to avoid unforeseen incompatibilities, you can generally use a recent release of Ubuntu, Fedora, Or OpenSUSE to complete this hands-on lab.

This lab assumes you have a basic understanding of the Yocto Project and bitbake, and are comfortable navigating a UNIX filesystem from the shell and issuing shell commands.

> **Tip:** Throughout the lab you will need to edit various files. Sometimes the pathnames to these files are long. It is critical that you enter them exactly. Remember you can use the **`Tab`** key to help autocomplete path names from the shell. You may also copy and paste the paths from the PDF version of this lab located on the desktop.

To launch a terminal:

`Applications ‣ Accessories ‣ Terminal`

Throughout the lab you may find it useful to work with more than one tab in your terminal. To create additional tabs:

`File ‣ Open Tab`

Before starting these exercises, please ensure your system has the necessary software prerequisites installed as described in the **Yocto Project Quick Start Guide** (see the subsection titled, “**The Packages**”): http://www.yoctoproject.org/docs/current/yocto-project-qs/yocto-project-qs.html

Note: Throughout the labs, we're using a user “appdevlab”. In your setup, please change it to your user login id.

Since Yocto Project eclipse plug-in can be used for both system and application developers, labs are grouped to cover these 2 usage models.

Setup Yocto Project Eclipse IDE

For basic Eclipse IDE download and setup, please follow "**Yocto Project Development Manual**" (http://www.yoctoproject.org/docs/current/dev-manual/dev-manual.html#workflow-using-the-adt-and-eclipse), section 4.2.2.1.1 "**Installing Eclipse IDE**".

Note: In the document, it used Juno 4.2. But the latest Juno at the download site is 4.2.1. So your downloaded tarball should be:

```
eclipse-SDK-4.2.1-linux-gtk-x86_64.tar.gz
```

After you've installed Eclipse IDE, then follow section 4.2.2.1.4.1 "Installing the Pre-built Plug-in from the Yocto Project Eclipse Update Site" to install Yocto Project Eclipse plug-ins. For Eclipse plug-in and required ADT setups, we're going to use Yocto Project 1.3 release.

Applications developers:

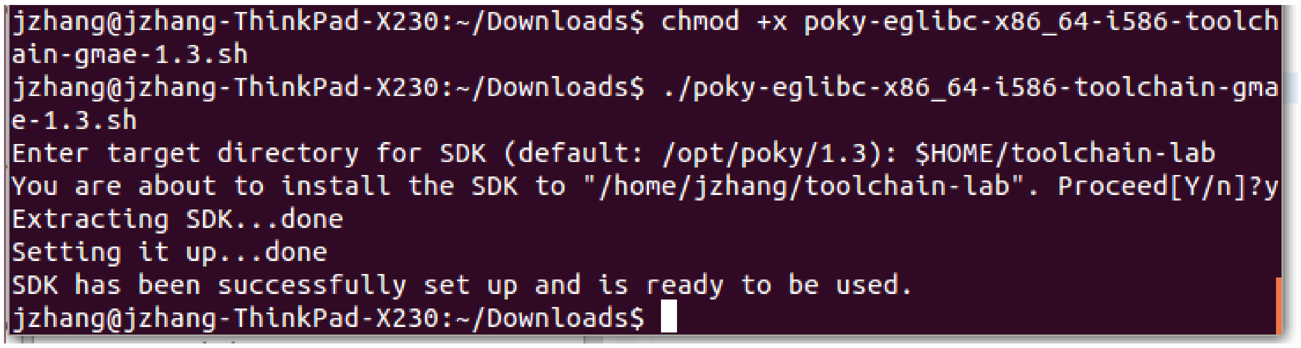
Application developers mainly will need Yocto Project ADT cross development environment. Please follow the following steps for setup:

step 1: setup toolchain

- go to http://downloads.yoctoproject.org/releases/yocto/yocto-1.3/toolchain/, select your development host architecture, e.g. x86_64. Then click on "poky-eglibc-x86_64-i586-toolchain-gmae-1.3.sh". In the pop up window, choose "Save File", which by default should save to your $HOME/Downloads directory
- After the toolchain is downloaded, do the following commands in a terminal:

```
$ cd $HOME/Downloads
$ chmod +x poky-eglibc-x86_64-i586-toolchain-gmae-13.sh
$ ./poky-eglibc-x86_64-i586-toolchain-gmae-1.3.sh
```

when it promps for directory for SDK, enter: $HOME/toolchain-lab. Then let it finish. And your terminal should look like:



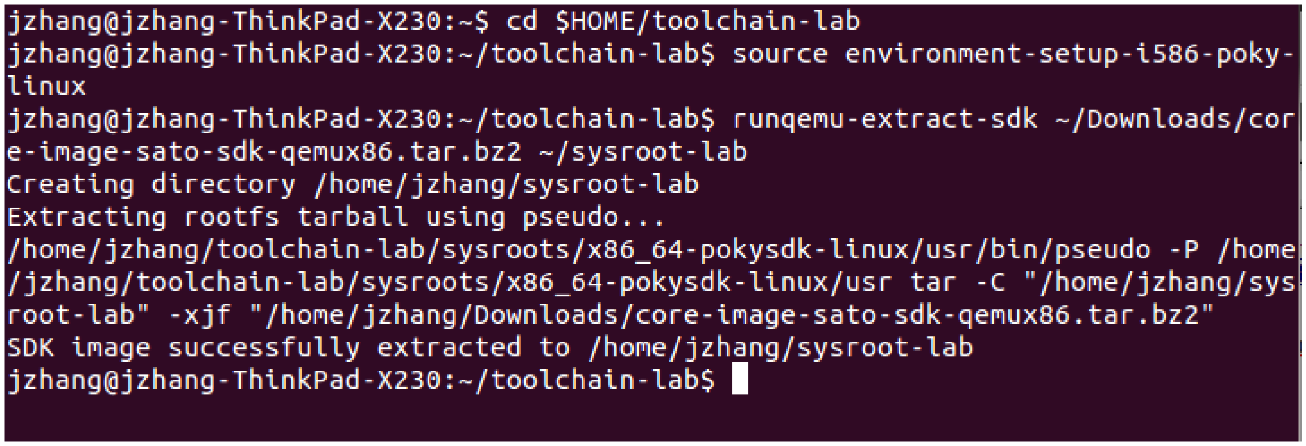
```
jzhang@jzhang-ThinkPad-X230:~/Downloads$ chmod +x poky-eglibc-x86_64-i586-toolch
ain-gmae-1.3.sh
jzhang@jzhang-ThinkPad-X230:~/Downloads$ ./poky-eglibc-x86_64-i586-toolchain-gma
e-1.3.sh
Enter target directory for SDK (default: /opt/poky/1.3): $HOME/toolchain-lab
You are about to install the SDK to "/home/jzhang/toolchain-lab". Proceed[Y/n]?y
Extracting SDK...done
Setting it up...done
SDK has been successfully set up and is ready to be used.
jzhang@jzhang-ThinkPad-X230:~/Downloads$
```

step 2: setup sysroot

- go to http://downloads.yoctoproject.org/releases/yocto/yocto1.3/machines/qemu/qemux86/, and download bzImage-qemux86.bin and core-image-sato-sdk-qemux86.tar.bz2. Save both to default Downloads directory
- do the following commands to extract your sysroot base on your target rootfs:

```
$cd $HOME/toolchain-lab
$source environment-setup-i586-poky-linux
$runqemu-extract-sdk ~/Downloads/core-image-sato-sdk-qemux86.tar.bz2
~/sysroot-lab
```

And your terminal should look like:



```
jzhang@jzhang-ThinkPad-X230:~$ cd $HOME/toolchain-lab
jzhang@jzhang-ThinkPad-X230:~/toolchain-lab$ source environment-setup-i586-poky-
linux
jzhang@jzhang-ThinkPad-X230:~/toolchain-lab$ runqemu-extract-sdk ~/Downloads/cor
e-image-sato-sdk-qemux86.tar.bz2 ~/sysroot-lab
Creating directory /home/jzhang/sysroot-lab
Extracting rootfs tarball using pseudo...
/home/jzhang/toolchain-lab/sysroots/x86_64-pokysdk-linux/usr/bin/pseudo -P /home
/jzhang/toolchain-lab/sysroots/x86_64-pokysdk-linux/usr tar -C "/home/jzhang/sys
root-lab" -xjf "/home/jzhang/Downloads/core-image-sato-sdk-qemux86.tar.bz2"
SDK image successfully extracted to /home/jzhang/sysroot-lab
jzhang@jzhang-ThinkPad-X230:~/toolchain-lab$
```

Now, we're all set to do our hands-on labs as an application developer.

Lab 1: Use Yocto Project eclipse plug-in for cross development

In this lab, you will be configuring Yocto Project eclipse plug-in for cross development based on the Yocto Project ADT settings you've done in setup stage. After that, you'll be cross compiling a Hello World application using the cross toolchain and sysroot setup.

Configure Yocto Project ADT plug-in for cross development

In eclipse IDE, do the following steps:

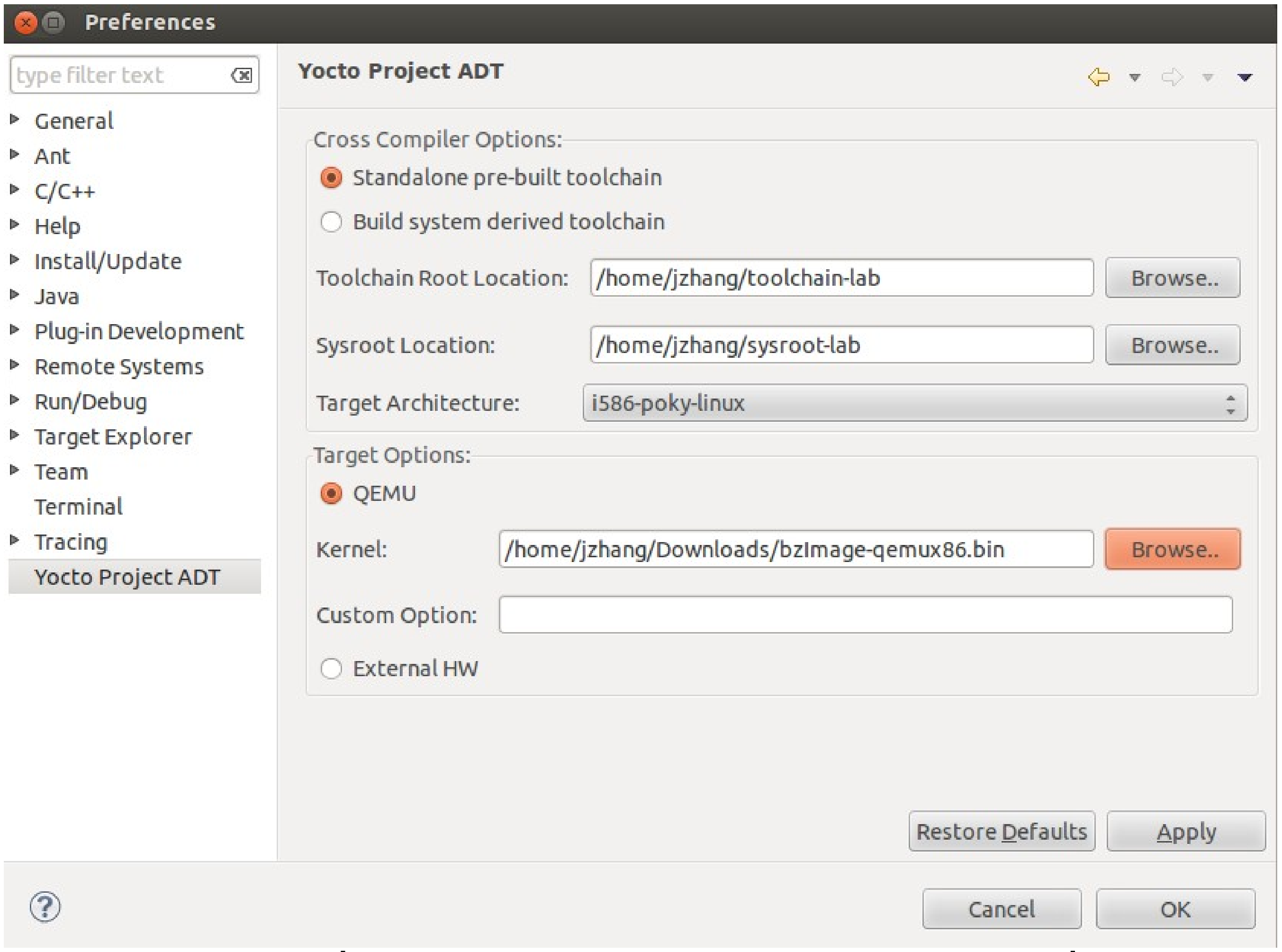
Step 1: Windows ‣ Preferences ‣ Yocto Project ADT

Step 2: In "Yocto Project ADT" preference window, select "Standalone pre-built toolchain". For "Toolchain Root Location" specify the fullpath to your toolchain-lab directory. For "Sysroot Location:" specify the fullpath to your sysroot-lab directory.

Step 3: After Step 2 is done, your "Target Architecture" drop down list should be automatically populated. For our setup, there's only one entry "i586-poky-linux". Select it.

Step 4: For "Target Options:" select QEMU. And for "kernel:" give the full path to your downloaded bzImage-qemux86.bin file. Then click "OK".

So your preference window should be looking like this except the userid should be your login id.


Yocto Project ADT

Cross Compiler Options:

(●) Standalone pre-built toolchain
( ) Build system derived toolchain

Toolchain Root Location: /home/jzhang/toolchain-lab

Sysroot Location: /home/jzhang/sysroot-lab

Target Architecture: i586-poky-linux

Target Options:

(●) QEMU

Kernel: /home/jzhang/Downloads/bzImage-qemux86.bin

Custom Option:

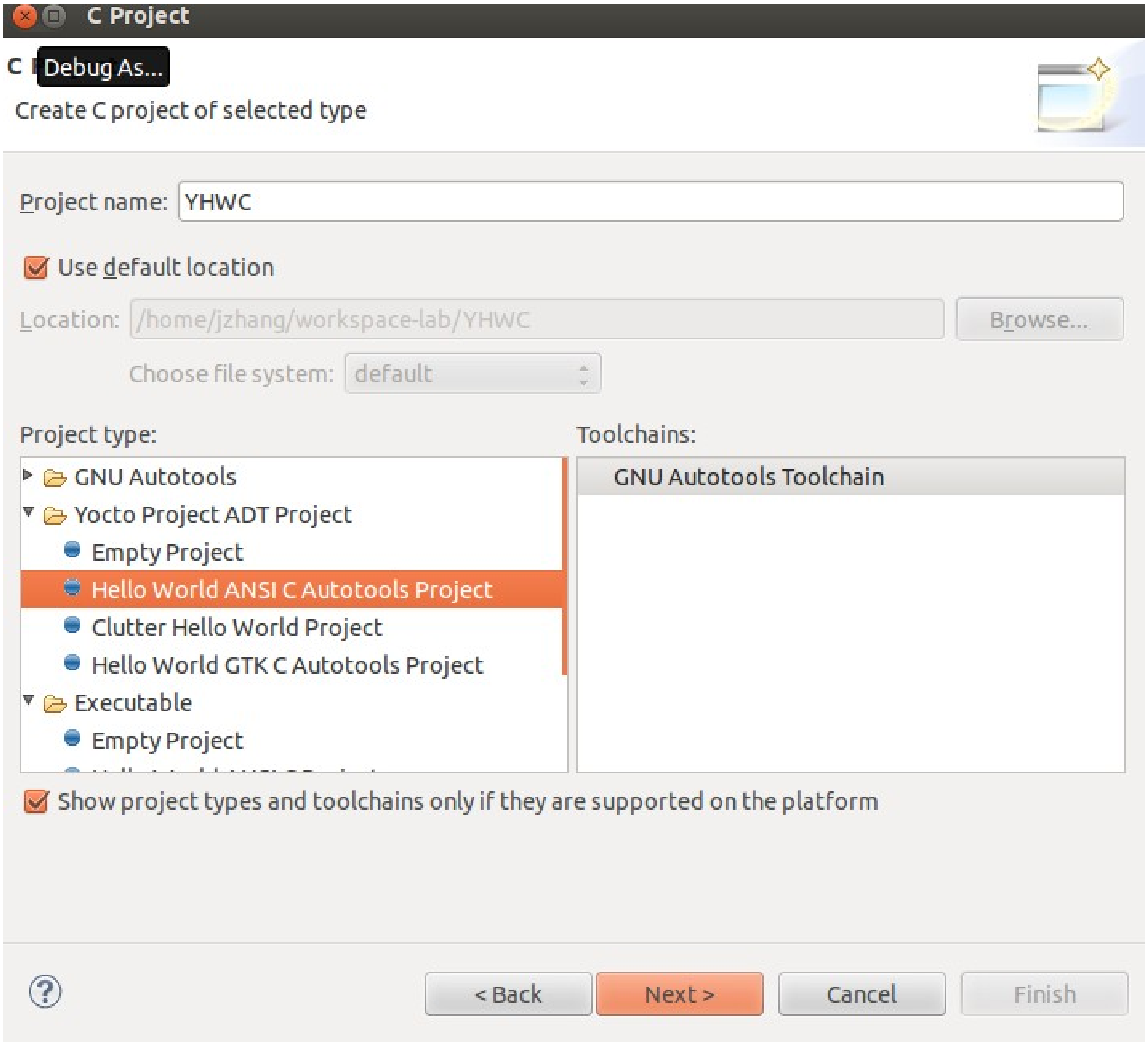
( ) External HW

Create a Yocto Project ADT base Hello World c Autotool Project

In eclipse IDE, do the following steps:

Step 1: File ‣ New Project ‣ C/C++ ‣ C Project, click on “Next” button

Step 2: In “C Project” wizard window, for “Project name: “ enter “YHWC”. In Project Type: extend “Yocto Project ADT Project” and select “Hello World ANSI C Autotools Project”.


C Project

Debug As...

Create C project of selected type

Project name: YHWC

☑ Use default location

Location: /home/jzhang/workspace-lab/YHWC

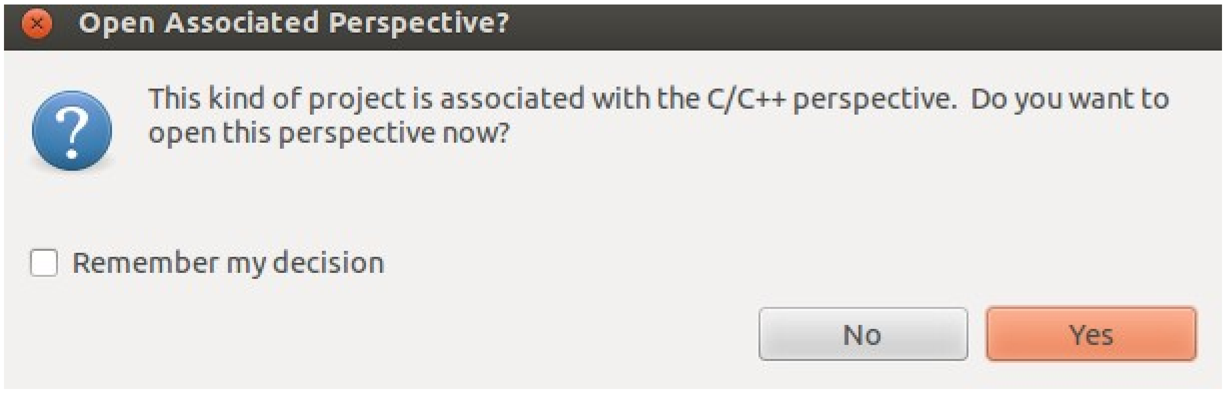
Choose file system: default

Project type:
- GNU Autotools
- Yocto Project ADT Project
  - Empty Project
  - Hello World ANSI C Autotools Project
  - Clutter Hello World Project
  - Hello World GTK C Autotools Project
- Executable
  - Empty Project

Toolchains: GNU Autotools Toolchain

☑ Show project types and toolchains only if they are supported on the platform

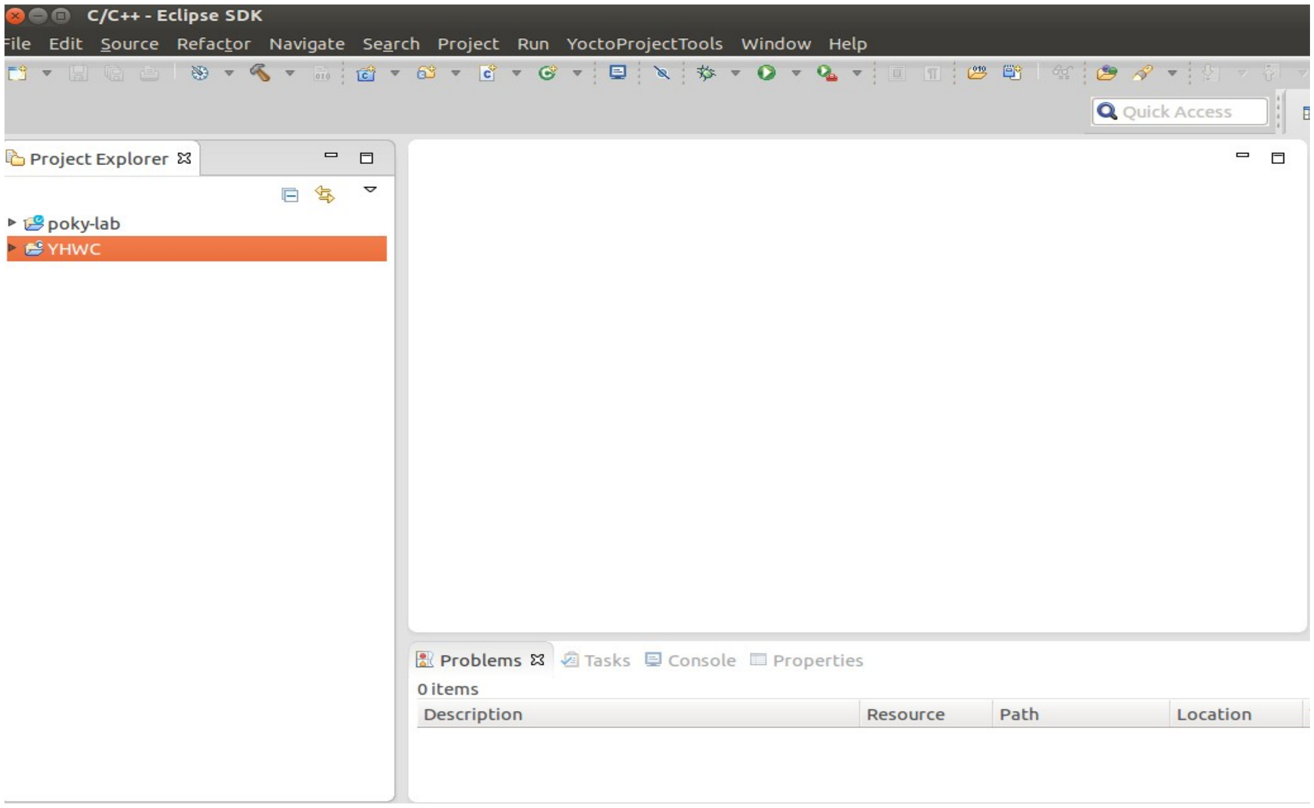
Step 3: Click on "Next", enter Author field and make necessary changes to other project properties if you like, e.g. License. Then click "Finish", you may see an "Open Associated Perspective?" pop out window, just click on "Yes".


Open Associated Perspective?

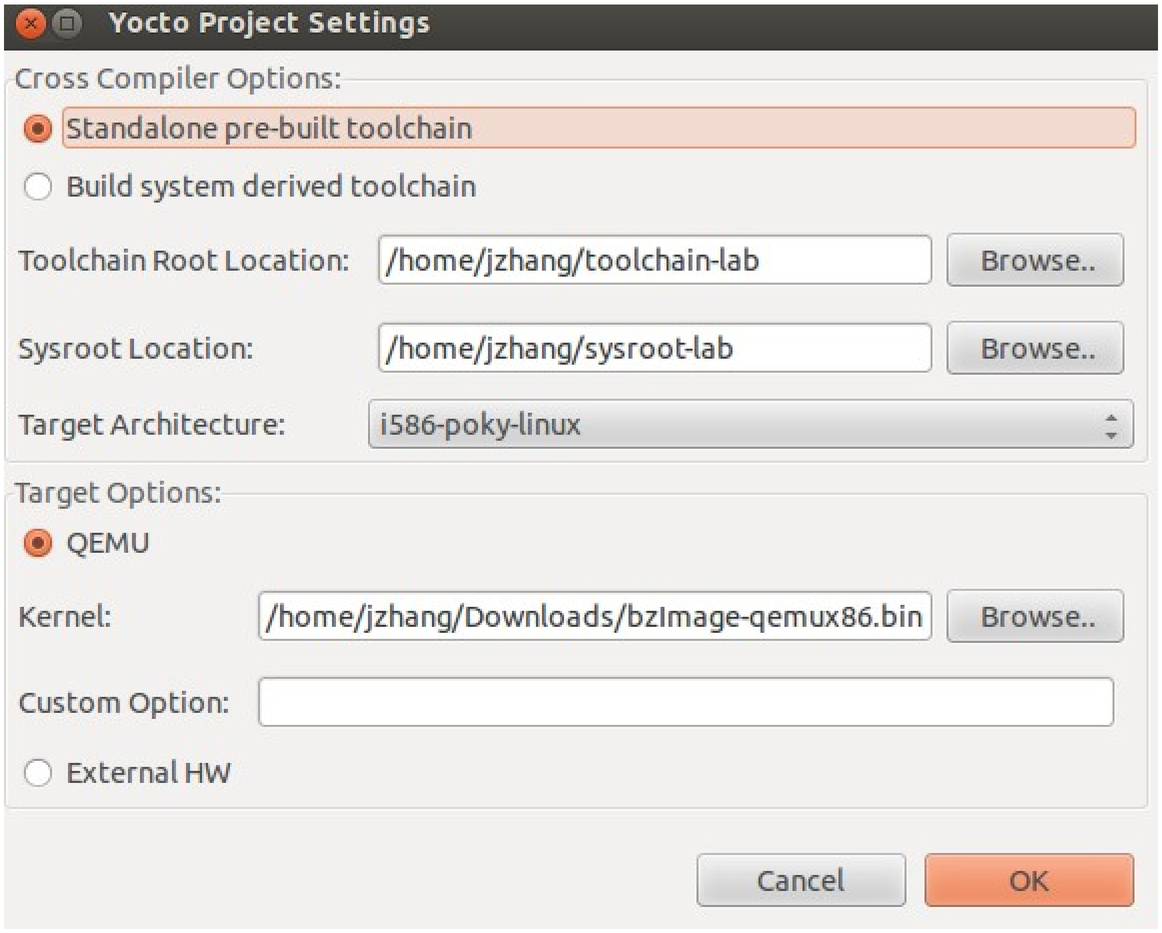
This kind of project is associated with the C/C++ perspective. Do you want to open this perspective now?

☐ Remember my decision

Now, your newly created project will be show in Project Explorer list.



Step 4: Click “Project” Menu, and select “Change Yocto Project Settings”. In the “Yocto Project Settings” window, you'll notice that your Yocto Project ADT preference settings are by default inherited here. If you have any project specific changes, you can make changes in this window. For our lab, we'll be using default. So we just click on “Cancel” to dismiss the window.


Yocto Project Settings

Cross Compiler Options:

(●) Standalone pre-built toolchain

( ) Build system derived toolchain

Toolchain Root Location: /home/jzhang/toolchain-lab

Sysroot Location: /home/jzhang/sysroot-lab

Target Architecture: i586-poky-linux

Target Options:

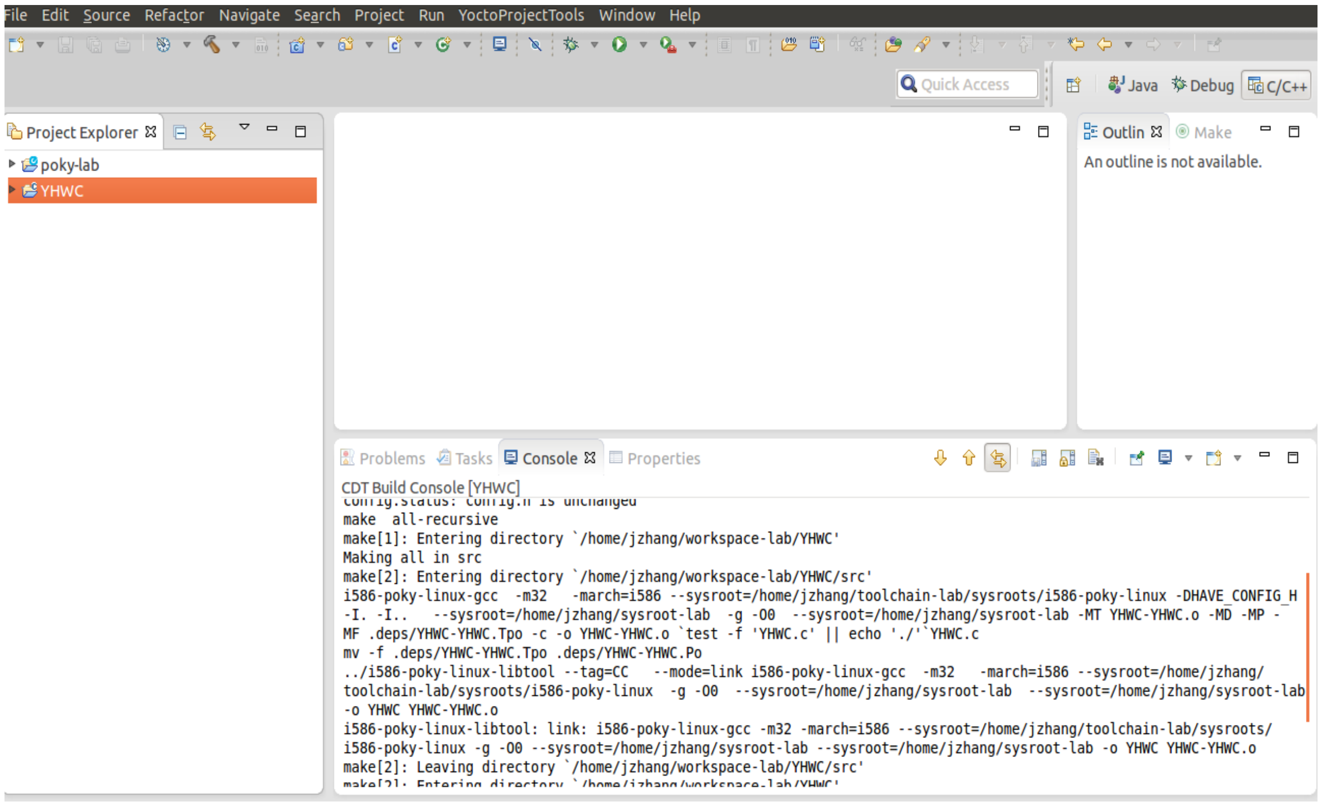
(●) QEMU

Kernel: /home/jzhang/Downloads/bzImage-qemux86.bin

Custom Option:

( ) External HW

Step 5: Highlight "YHWC" project, in "Project" menu, select "Build Project". Then click on Console and ensure it's using "CDT build console". In the console window, you'll notice the cross tools, e.g. cross compiler and your sysroot are taken into effect when building the Hello World autotool based project.


File Edit Source Refactor Navigate Search Project Run YoctoProjectTools Window Help
Quick Access
Java
Debug
C/C++
Project Explorer
poky-lab
YHWC
Outlin
Make
An outline is not available.
Problems
Tasks
Console
Properties

```
CDT Build Console [YHWC]
config.status: config.h is unchanged
make  all-recursive
make[1]: Entering directory `/home/jzhang/workspace-lab/YHWC'
Making all in src
make[2]: Entering directory `/home/jzhang/workspace-lab/YHWC/src'
i586-poky-linux-gcc  -m32   -march=i586 --sysroot=/home/jzhang/toolchain-lab/sysroots/i586-poky-linux -DHAVE_CONFIG_H
-I. -I..   --sysroot=/home/jzhang/sysroot-lab  -g -O0  --sysroot=/home/jzhang/sysroot-lab -MT YHWC-YHWC.o -MD -MP -
MF .deps/YHWC-YHWC.Tpo -c -o YHWC-YHWC.o `test -f 'YHWC.c' || echo './'`YHWC.c
mv -f .deps/YHWC-YHWC.Tpo .deps/YHWC-YHWC.Po
../i586-poky-linux-libtool --tag=CC   --mode=link i586-poky-linux-gcc  -m32   -march=i586 --sysroot=/home/jzhang/
toolchain-lab/sysroots/i586-poky-linux  -g -O0  --sysroot=/home/jzhang/sysroot-lab  --sysroot=/home/jzhang/sysroot-lab
-o YHWC YHWC-YHWC.o
i586-poky-linux-libtool: link: i586-poky-linux-gcc -m32 -march=i586 --sysroot=/home/jzhang/toolchain-lab/sysroots/
i586-poky-linux -g -O0 --sysroot=/home/jzhang/sysroot-lab --sysroot=/home/jzhang/sysroot-lab -o YHWC YHWC-YHWC.o
make[2]: Leaving directory `/home/jzhang/workspace-lab/YHWC/src'
make[2]: Entering directory `/home/jzhang/workspace-lab/YHWC'
```

Lab 1 Conclusion

Congratulations! You have created a Yocto Project ADT autotools Hello World project and successfully cross compiled it with your Yocto Project ADT cross development setup. This concludes Lab 1.

Lab 2: Use Yocto Project eclipse plug-in for cross debugging

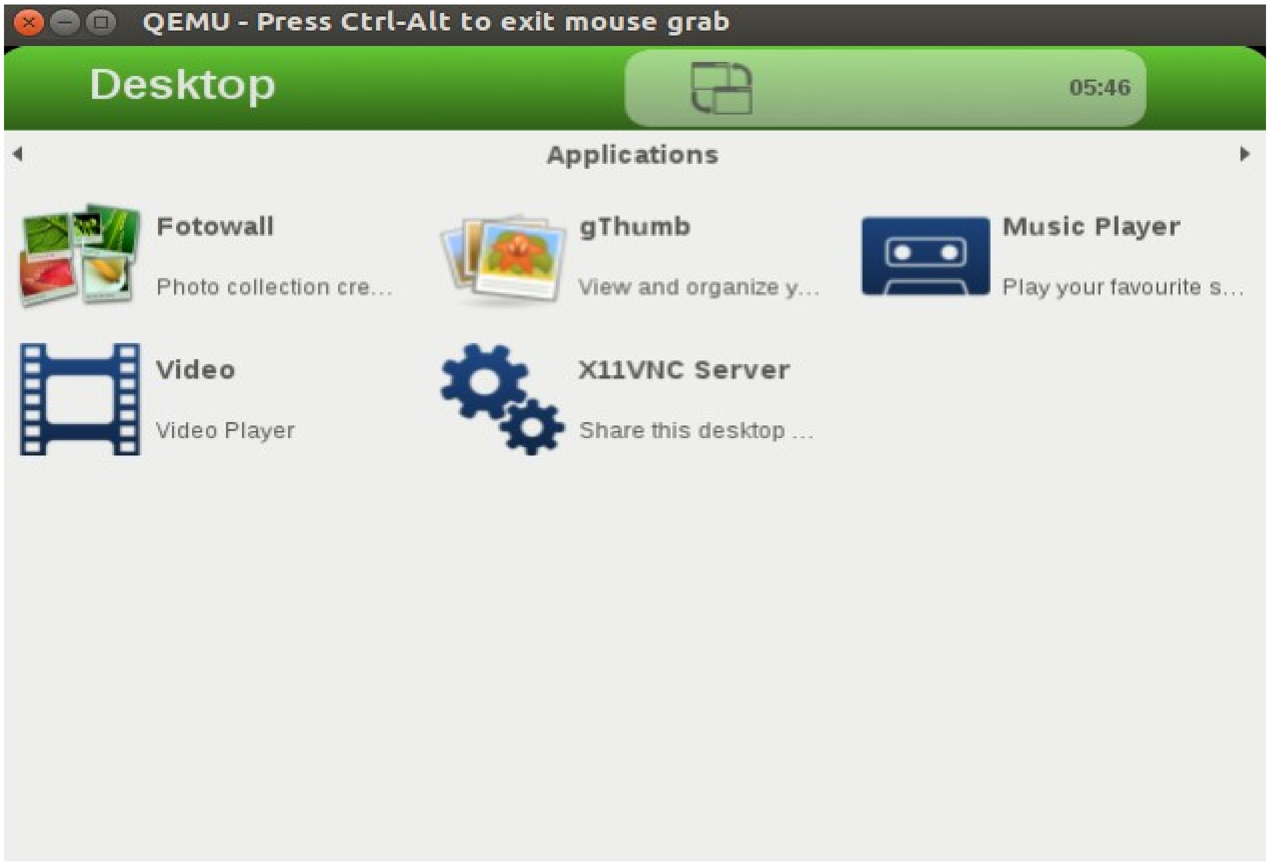
In this lab, you'll be doing cross deploy and debugging of lab 1 YHWC project against qemu within eclipse IDE.

Start a qemu instance from eclipse IDE

`Run ‣ External Tools`, You should see a "qemu_i586-poky-linux" launcher entry. This is automatically created based on your project's sysroot setting. Click on it, it'll boot up an qemu with unfs using your extract target rootfs which is your sysroot.

Note: Please watch out for an xterm window during qemu boot up, it may ask you to enter user password or prompt you for missing setup, e.g. rpcbind, for qemu to boot up using unfs.

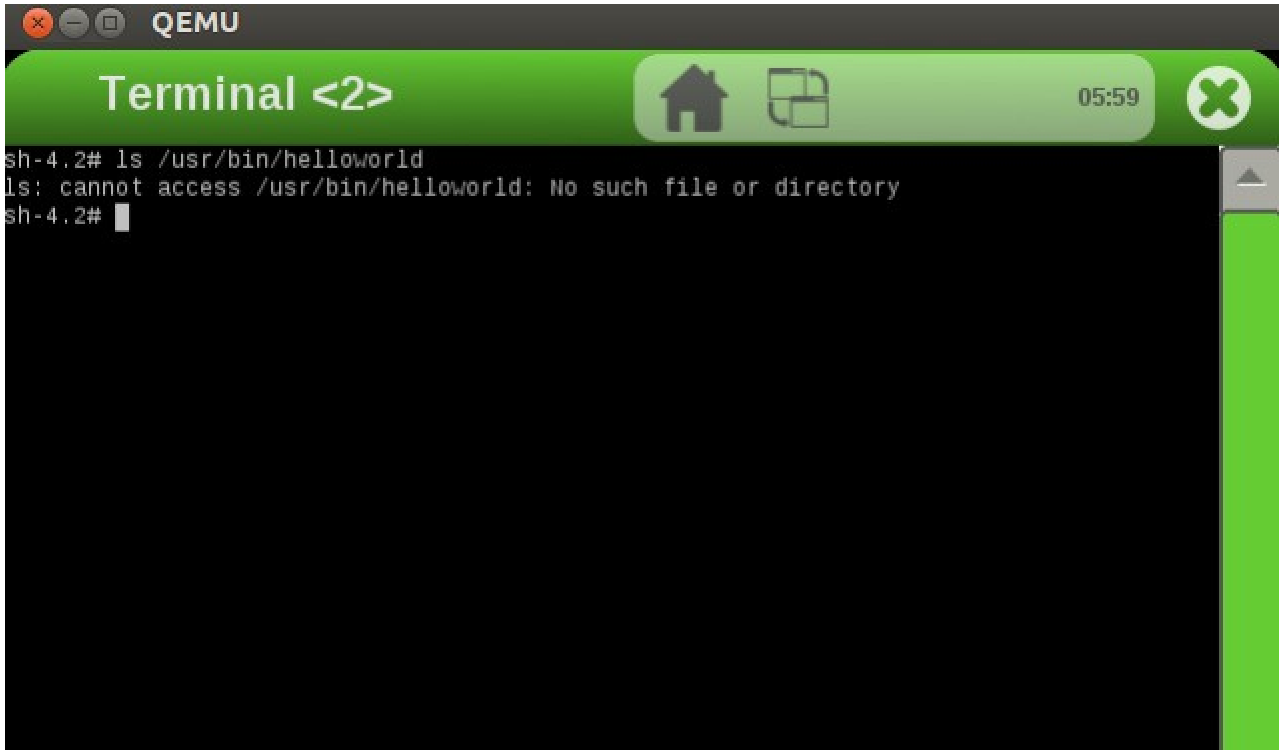
Since our target rootfs is core-image-sato-sdk. Qemu will boot into sato desktop.



Click on ▸ to go "Utilities" and click on "Terminal" icon to bring up a Terminal. And type:

```
ls /usr/bin/helloworld
```

Since there's no helloworld binary under /usr/bin in the image, you should get:



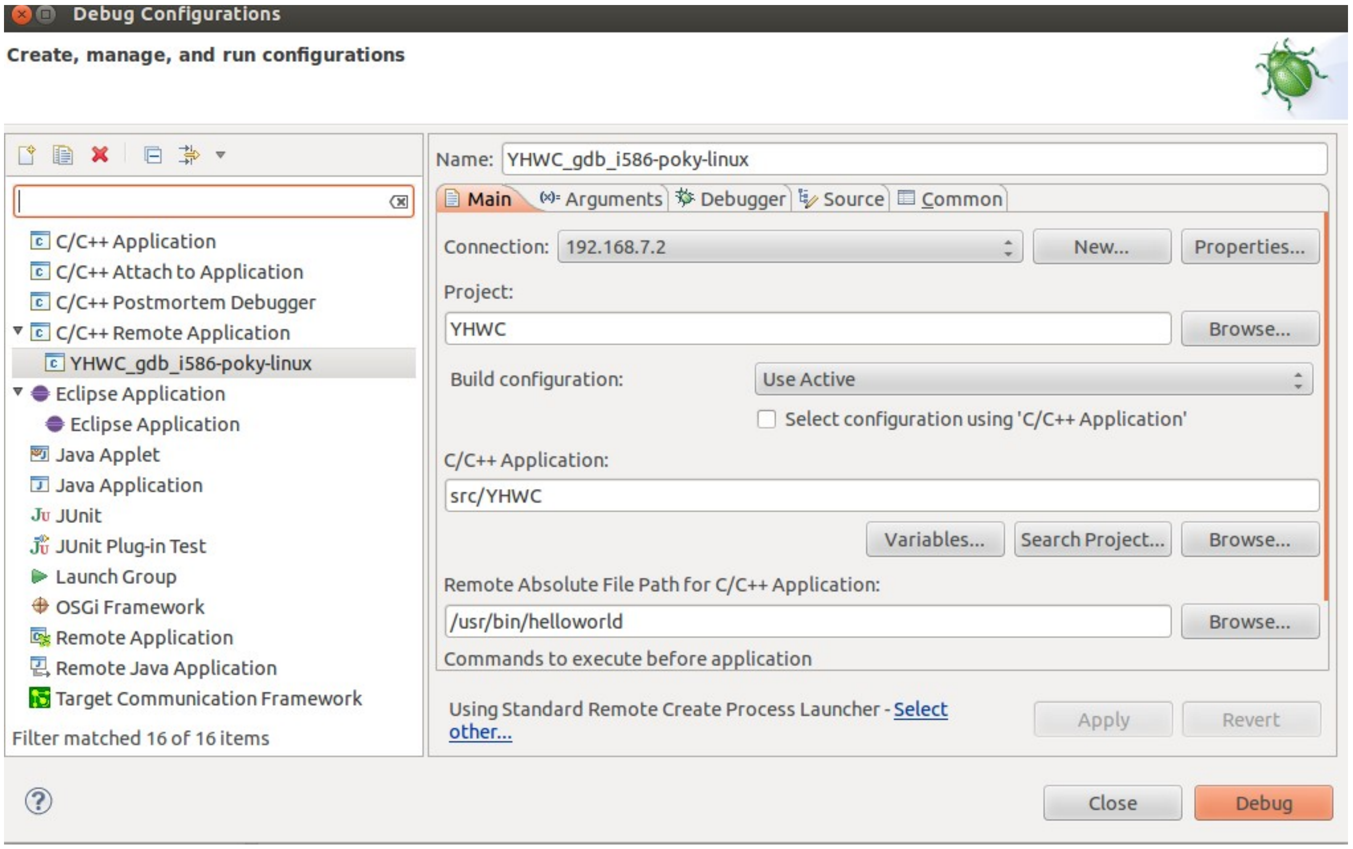
```
sh-4.2# ls /usr/bin/helloworld
ls: cannot access /usr/bin/helloworld: No such file or directory
sh-4.2# 
```

cross deploy and debug against qemu

Go back to eclipse IDE, highlight the YHWC project. Then go to “Run” menu, select “Debug Configurations...”. Extend “C/C++ Remote Application”, you should see “YHWC_gdb_i586-poky-linux” remote debug launcher is created for you. On the right hand side, you need to further refine your remote debug configuration settings.

Step 1: Create a new connection to remote target. For this lab, it is our just booted qemu instance. Click on “New” button besides connection. In the new connection wizard, select “SSH only” then click on “next” button. For “Host Name” replace l local host with 192.168.7.2, which is our qemu ip address. Then click “Finish”. You'll see the connection is populated in the remote debug configuration.

Step 2: For “Remote Absolute File Path for C/C++ Application: “ type “/usr/bin/helloworld”. Since all the required fields are set, the “Debug” button is enabled. Click on it.


Debug Configurations

Create, manage, and run configurations

C/C++ Application
C/C++ Attach to Application
C/C++ Postmortem Debugger
C/C++ Remote Application
YHWC_gdb_i586-poky-linux
Eclipse Application
Eclipse Application
Java Applet
Java Application
JUnit
JUnit Plug-in Test
Launch Group
OSGi Framework
Remote Application
Remote Java Application
Target Communication Framework

Filter matched 16 of 16 items

Name: YHWC_gdb_i586-poky-linux

Main | Arguments | Debugger | Source | Common

Connection: 192.168.7.2 [New...] [Properties...]

Project: YHWC [Browse...]

Build configuration: Use Active

Select configuration using 'C/C++ Application'

C/C++ Application: src/YHWC [Variables...] [Search Project...] [Browse...]

Remote Absolute File Path for C/C++ Application: /usr/bin/helloworld [Browse...]

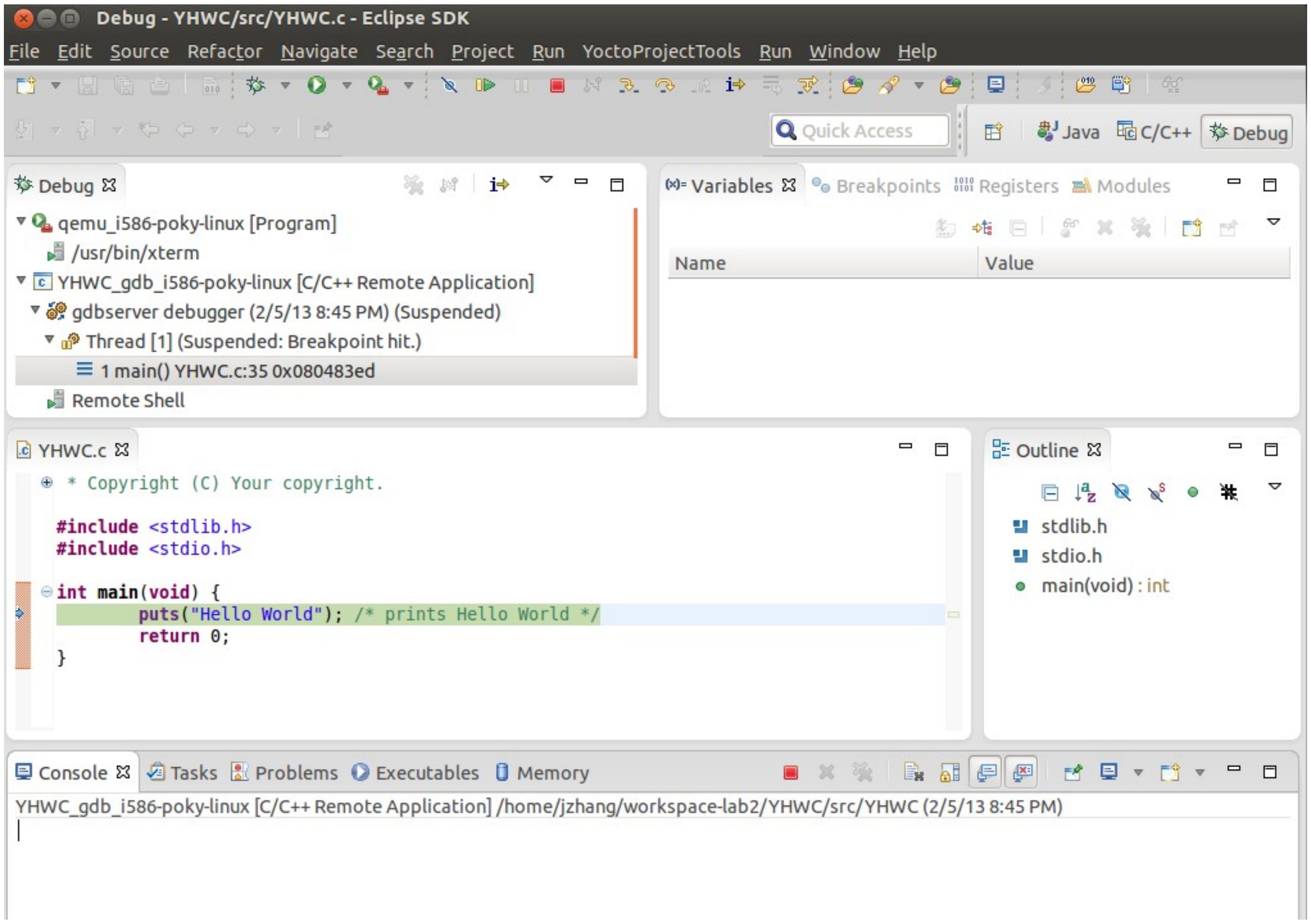
Commands to execute before application

Using Standard Remote Create Process Launcher - Select other...

[Apply] [Revert]

[Close] [Debug]

An “Enter Password” window will pop out asking for usrid and password to log into remote system using SSH. For our sato-sdk image, the user id is “root” and there's no password. Then, there will be a prompt to confirm to switch to debug perspective, click on “yes”. Now you should enter the debug view as captured here.


Debug - YHWC/src/YHWC.c - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run YoctoProjectTools Run Window Help
Quick Access
Java C/C++ Debug
Debug
Variables Breakpoints Registers Modules
Name Value
YHWC.c
Outline
stdlib.h
stdio.h
main(void) : int
Console Tasks Problems Executables Memory

```
qemu_i586-poky-linux [Program]
  /usr/bin/xterm
YHWC_gdb_i586-poky-linux [C/C++ Remote Application]
  gdbserver debugger (2/5/13 8:45 PM) (Suspended)
    Thread [1] (Suspended: Breakpoint hit.)
      1 main() YHWC.c:35 0x080483ed
  Remote Shell
```

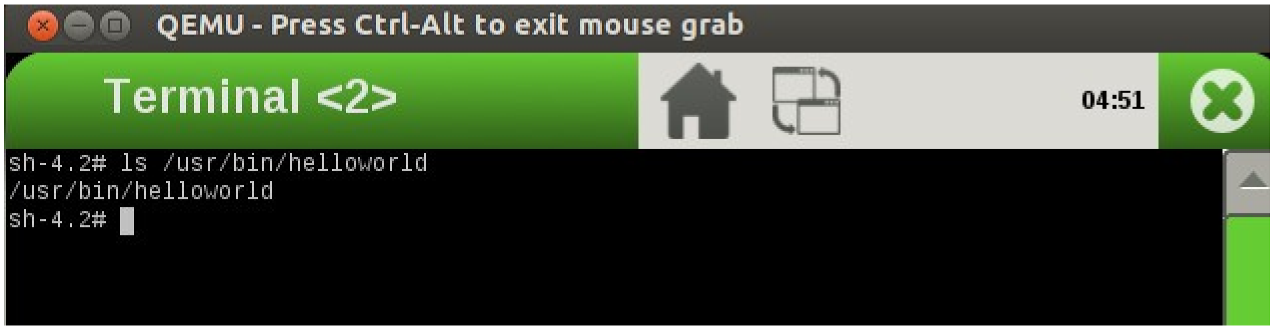
```c
 * Copyright (C) Your copyright.

#include <stdlib.h>
#include <stdio.h>

int main(void) {
	puts("Hello World"); /* prints Hello World */
	return 0;
}
```

```
YHWC_gdb_i586-poky-linux [C/C++ Remote Application] /home/jzhang/workspace-lab2/YHWC/src/YHWC (2/5/13 8:45 PM)
```

Switch to qemu terminal and do:

```
# ls /usr/bin/helloworld
```

you should see



```
sh-4.2# ls /usr/bin/helloworld
/usr/bin/helloworld
sh-4.2#
```

The cross debugging by default also deploy the application to your target. This conclude our lab2.

Lab2 conclusion:

In this lab, we exercised how to do remote cross debugging using Eclipse IDE.

System Developers:

System developers mainly interact with Yocto Project meta data and layers and using bitbake build system. We're going to use Yocto Project upcoming 1.4 release milestone one release for metadata. Please follow the following commands:

```
$ cd $HOME
$ git clone git://git.yoctoproject.org/poky poky-lab
$ cd ~/poky-lab
$ git checkout 1.4_M1
```

To speed up your build, please change bitbake parallel configuration by doing the following commands:

```
$ source oe-init-build-env
$ vi conf/local.conf
```

uncomment **BB_NUMBER_THREADS** and **PARALLEL_MAKE. I**f you have a fast machine, you can change the number from “**4” to “8”** for both values.

Lab 1: Yocto Bitbake Commander Project and HOB

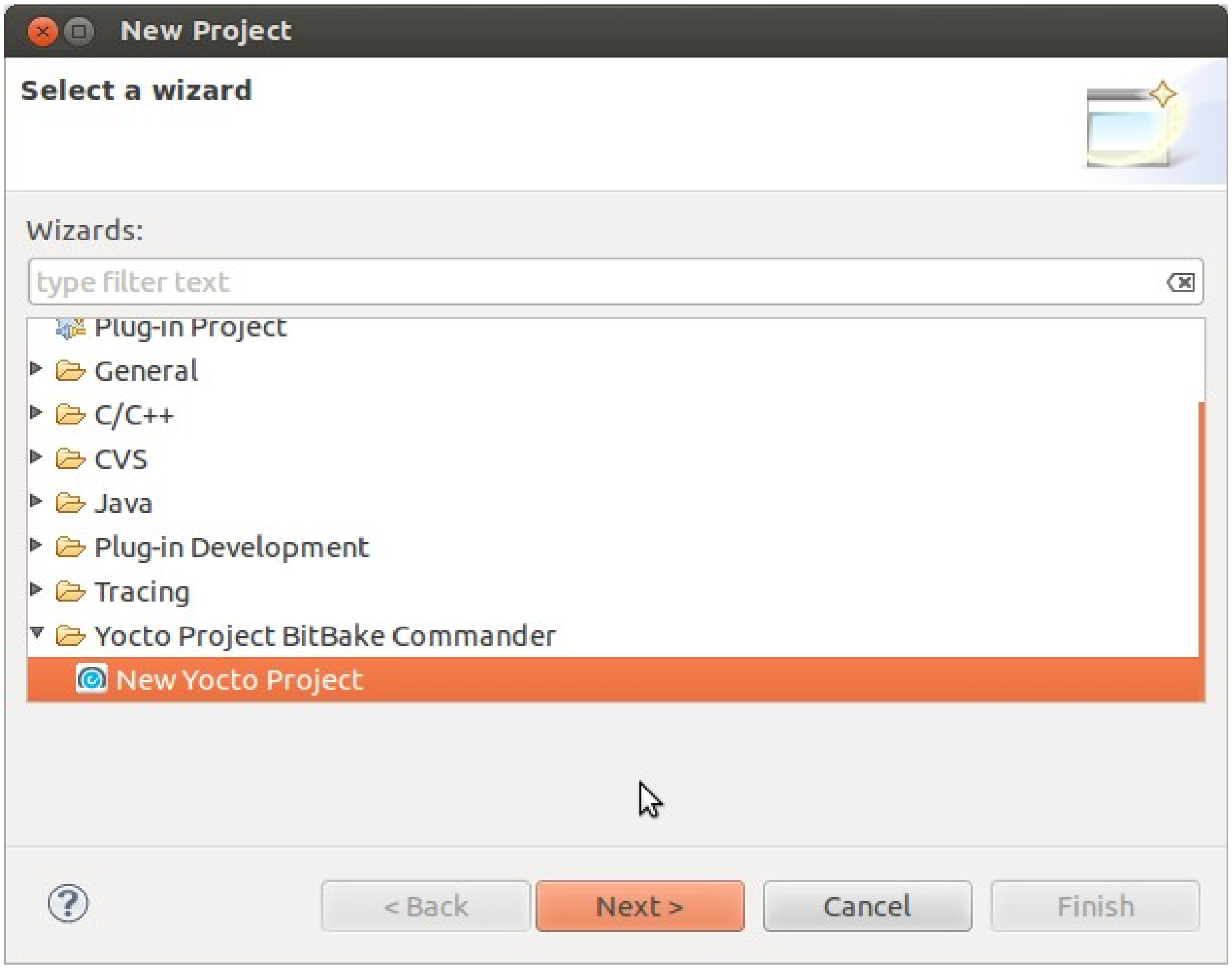
In this lab you will create a bitbake commander project within Eclipse IDE against the pre-populated poky tree on your local machine. Then you'll bring up hob against the bitbake commander project's meta data. You will then build a predefined qemu image using hob. After the build, boot up the qemu image. This will familiarize you with the basic workflow as a system developer for using Yocto Project Eclipse plug-in and hob to run a build.

Create a bitbake commander project

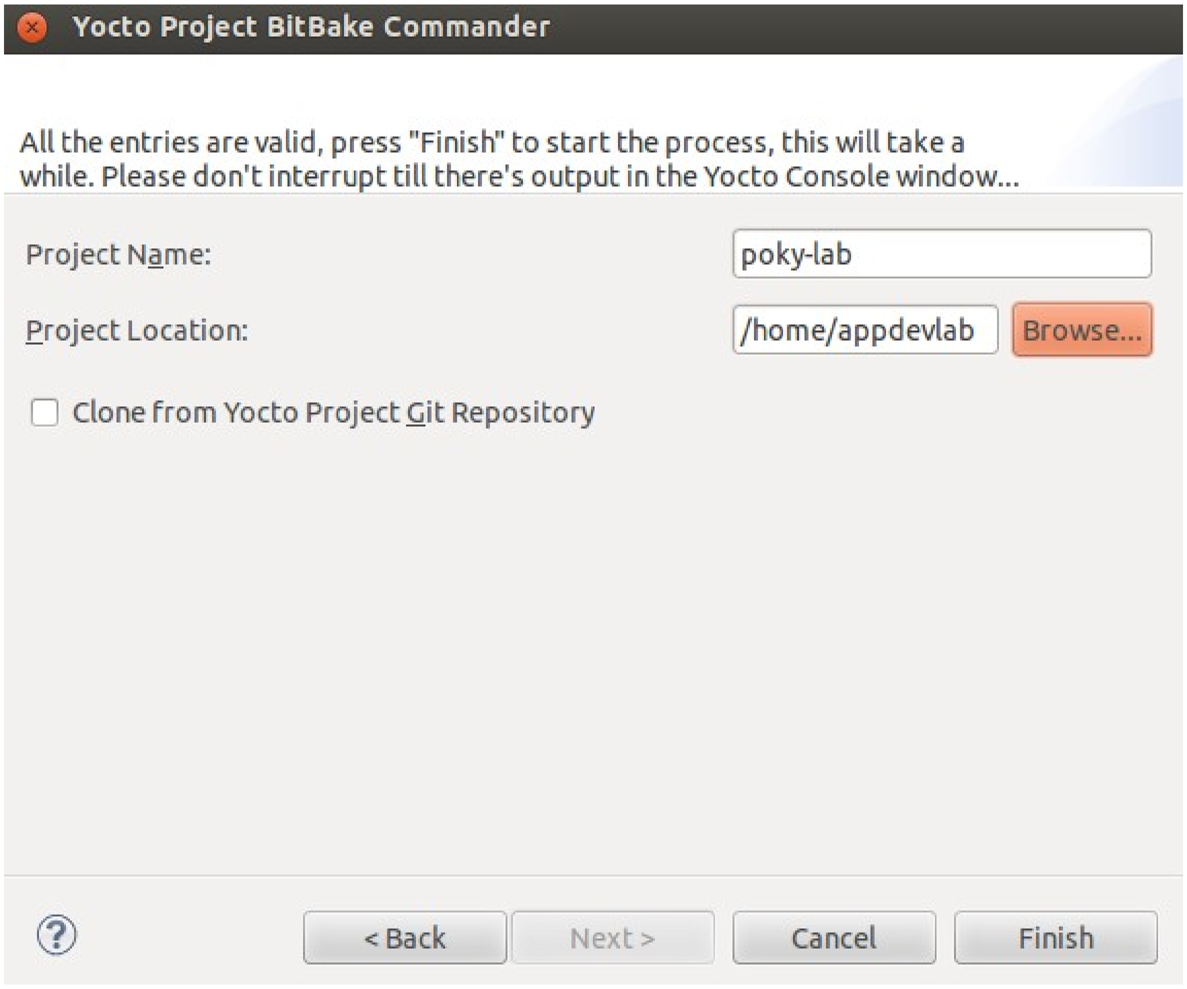
In eclipse IDE, do the following steps:

Step 1: File ‣ New ‣ Project

Step 2: In “New Project” window, click on ‣ besides “Yocto Project BitBake Commander”, and select “New Yocto Project”. Then press “Next” button.



Step 3: In "Yocto Project BitBake Commander" window, enter "poky-lab" for "Project Name" field, and browse for "Project Location" field to select "/home/appdevlab". Then click "Finish".


Yocto Project BitBake Commander

All the entries are valid, press "Finish" to start the process, this will take a while. Please don't interrupt till there's output in the Yocto Console window...

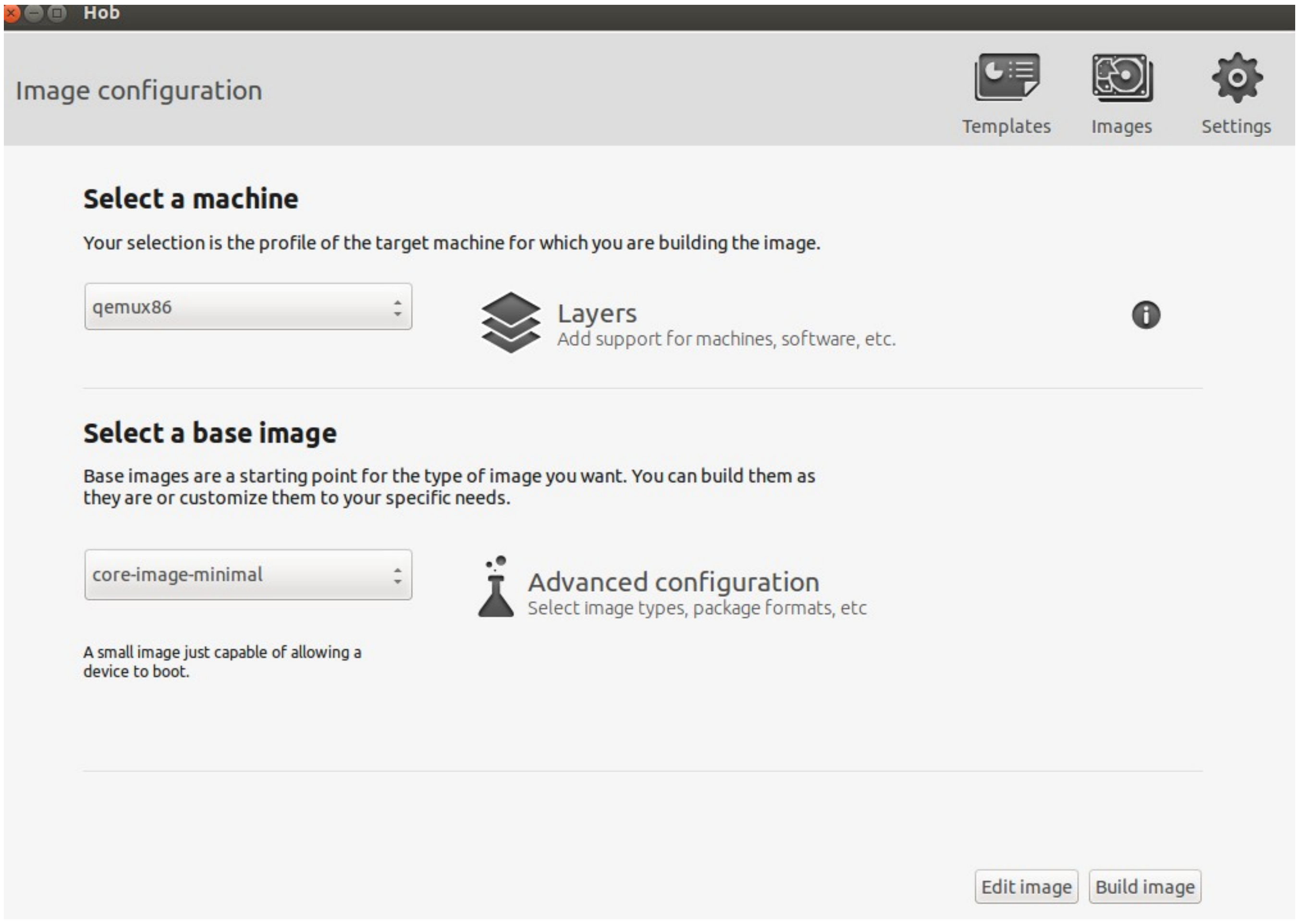
Project Name: poky-lab

Project Location: /home/appdevlab

Browse...
Clone from Yocto Project Git Repository
< Back
Next >
Cancel
Finish

Step 4: Wait till the poky-lab project is created, highlight it and go to " Project ▸ Launch HOB". In "Bitbake build directory:" browse to ensure the selection is made as: /home/appdevlab/poky-lab/build (if you don't have build directory under poky-lab, just type it in the text box). Then click "OK" to bring up hob. If this is the first time you've done build, it'll build pseudo first before hob will come up, so give it couple minutes.

Step 5: In hob, for "Select a machine" choose "qemux86". After parsing is done, for "Select a base image" choose "core-image-minimal".



Select a machine

Your selection is the profile of the target machine for which you are building the image.

qemux86

Select a base image

Base images are a starting point for the type of image you want. You can build them as they are or customize them to your specific needs.

core-image-minimal

A small image just capable of allowing a device to boot.

Then press “Build Image”. It'll bring you to hob build view. Wait till the build is done. Click on “Run image”, this will boot up the qemu image you just built using hob.

Note: In the qemu boot terminal, it may ask you for the user password in the xterm, please pay attention.

Step 6: At login prompt, type “root” and hit “enter”. Now, you're logged into qemu. Type the following commands and note down the information, this will prepare us for the next two labs:

```
# powertop
```

and you should get:

```
-sh: powertop: not found
```

then type:

```
# uname -r
```

and you should get:

```
3.4.18-yocto-standard
```

Now quit qemu, this concludes lab 1.

Lab 1 Conclusion

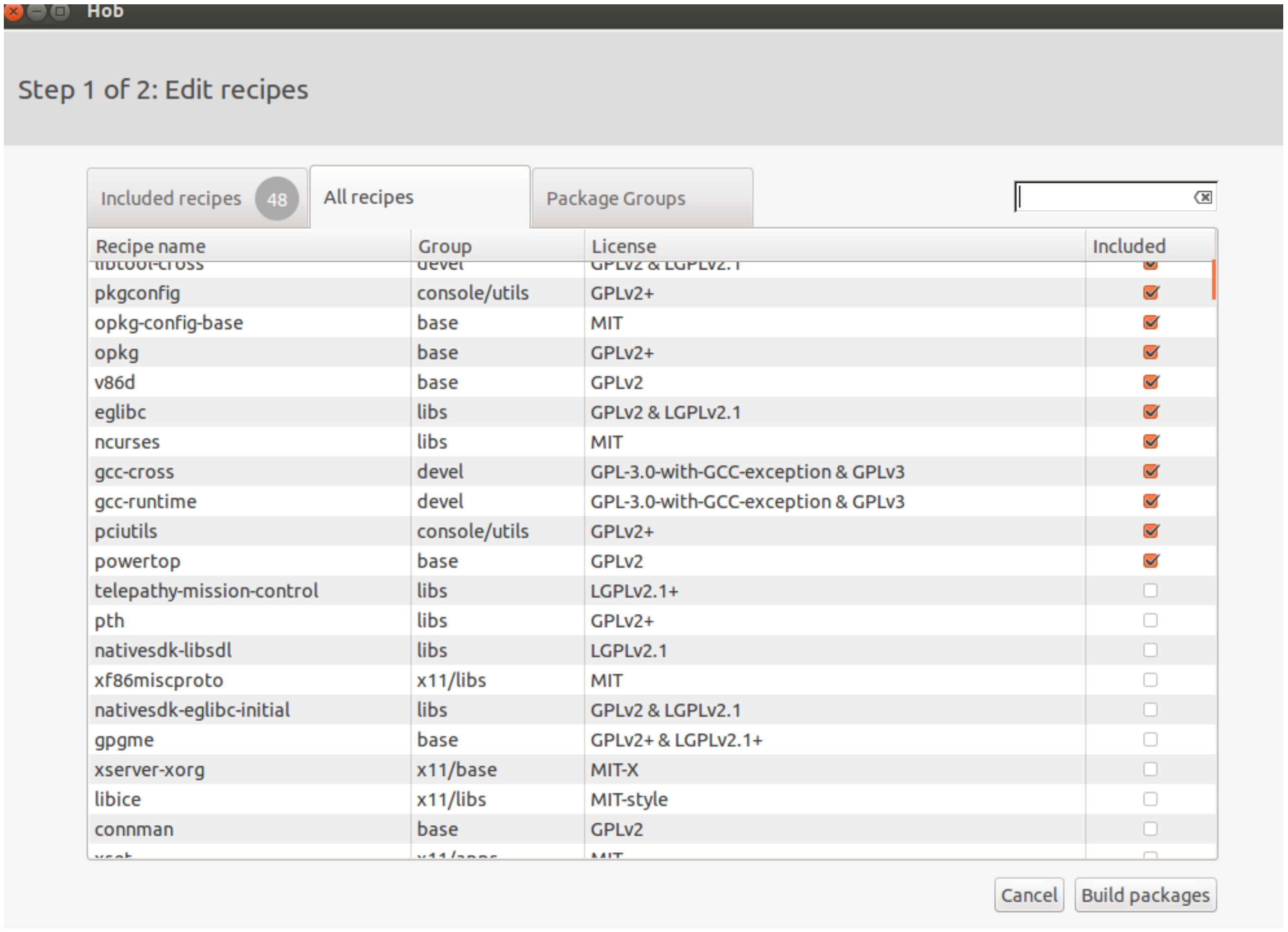
Congratulations! You have created a bitbake commander project in Eclipse IDE and built a qemu image using hob against the bitbake commander project meta data. You've logged into your built qemu image and ran couple commands to set the stage for the next two labs. This concludes Lab 1.

Lab 2: Customize image using hob

In this lab you will further explore one of hob main features that allows you to efficiently customize your image. In lab 1, we noticed that "powertop" is not included in the core-image-minimal. In lab 2, we'll be using hob to add it to our customized image. We'll finish this lab with following steps:

Step 1: Switch back to hob. And click on "Build new image", this will bring us back to the hob first window.

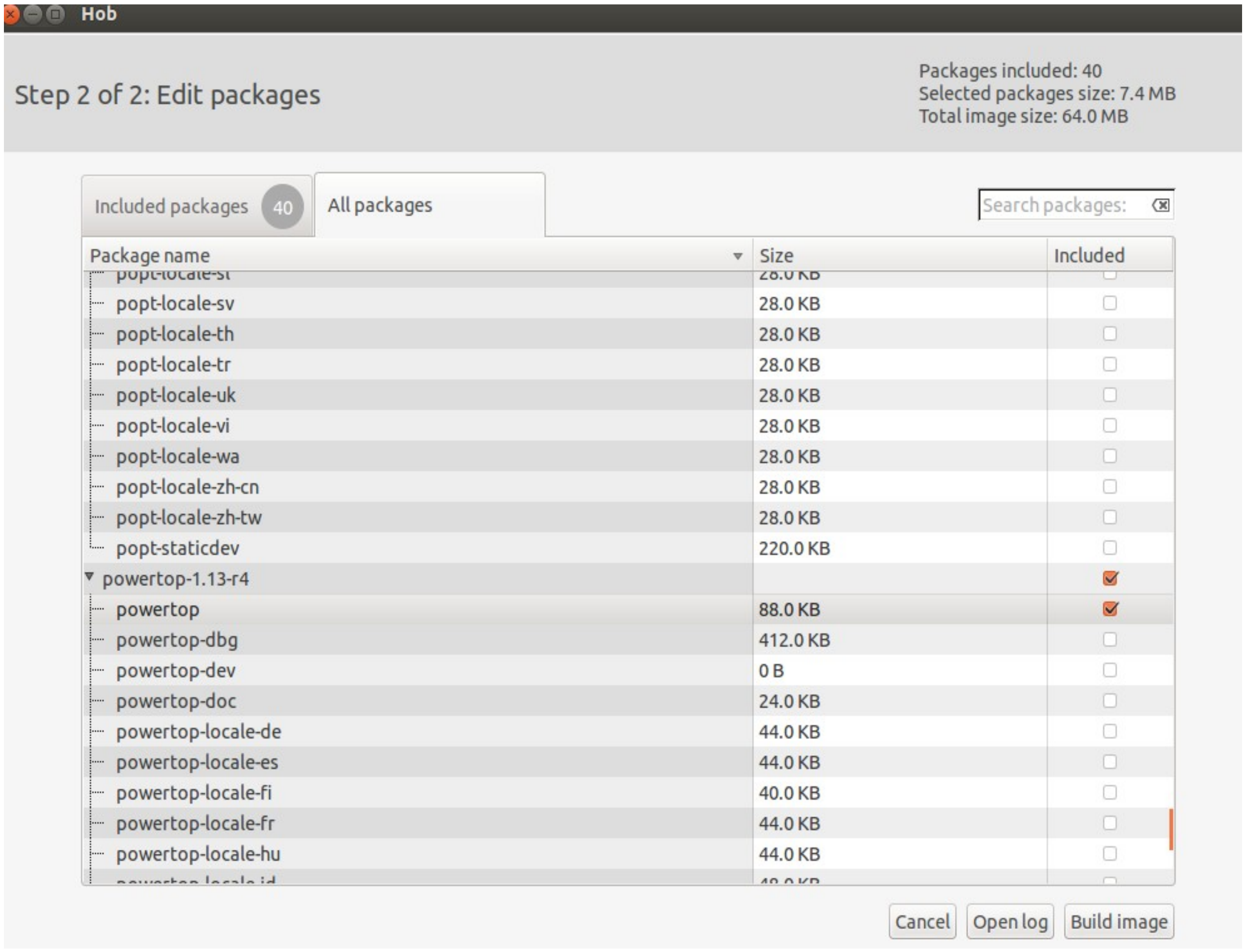
Step 2: Click on "Edit Image" to bring up the recipe view. Then, click on "All recipes" tab. In top right hand "Search recipes:" box, type "powertop" and hit enter. This will jump to powertop recipe entry. You'll notice the "Included" box is not ticked. Click on that box to ensure it's ticked. Then click on "Build packages".



Recipe name	Group	License	Included
libtool-cross	devel	GPLv2 & LGPLv2.1	✓
pkgconfig	console/utils	GPLv2+	✓
opkg-config-base	base	MIT	✓
opkg	base	GPLv2+	✓
v86d	base	GPLv2	✓
eglibc	libs	GPLv2 & LGPLv2.1	✓
ncurses	libs	MIT	✓
gcc-cross	devel	GPL-3.0-with-GCC-exception & GPLv3	✓
gcc-runtime	devel	GPL-3.0-with-GCC-exception & GPLv3	✓
pciutils	console/utils	GPLv2+	✓
powertop	base	GPLv2	✓
telepathy-mission-control	libs	LGPLv2.1+	☐
pth	libs	GPLv2+	☐
nativesdk-libsdl	libs	LGPLv2.1	☐
xf86miscproto	x11/libs	MIT	☐
nativesdk-eglibc-initial	libs	GPLv2 & LGPLv2.1	☐
gpgme	base	GPLv2+ & LGPLv2.1+	☐
xserver-xorg	x11/base	MIT-X	☐
libice	x11/libs	MIT-style	☐
connman	base	GPLv2	☐

This will bring up the package build window, which you can see all the related packages for powertop will be built. Wait till the build is done

Step 3: After the package build is done. The "Edit Packages" window will be showing. Here, click on "All packages" tab. In top right hand "Search packages:" box, type "powertop" and hit enter. This will bring us to the powertop packages list. There, only click on the included box for powertop.



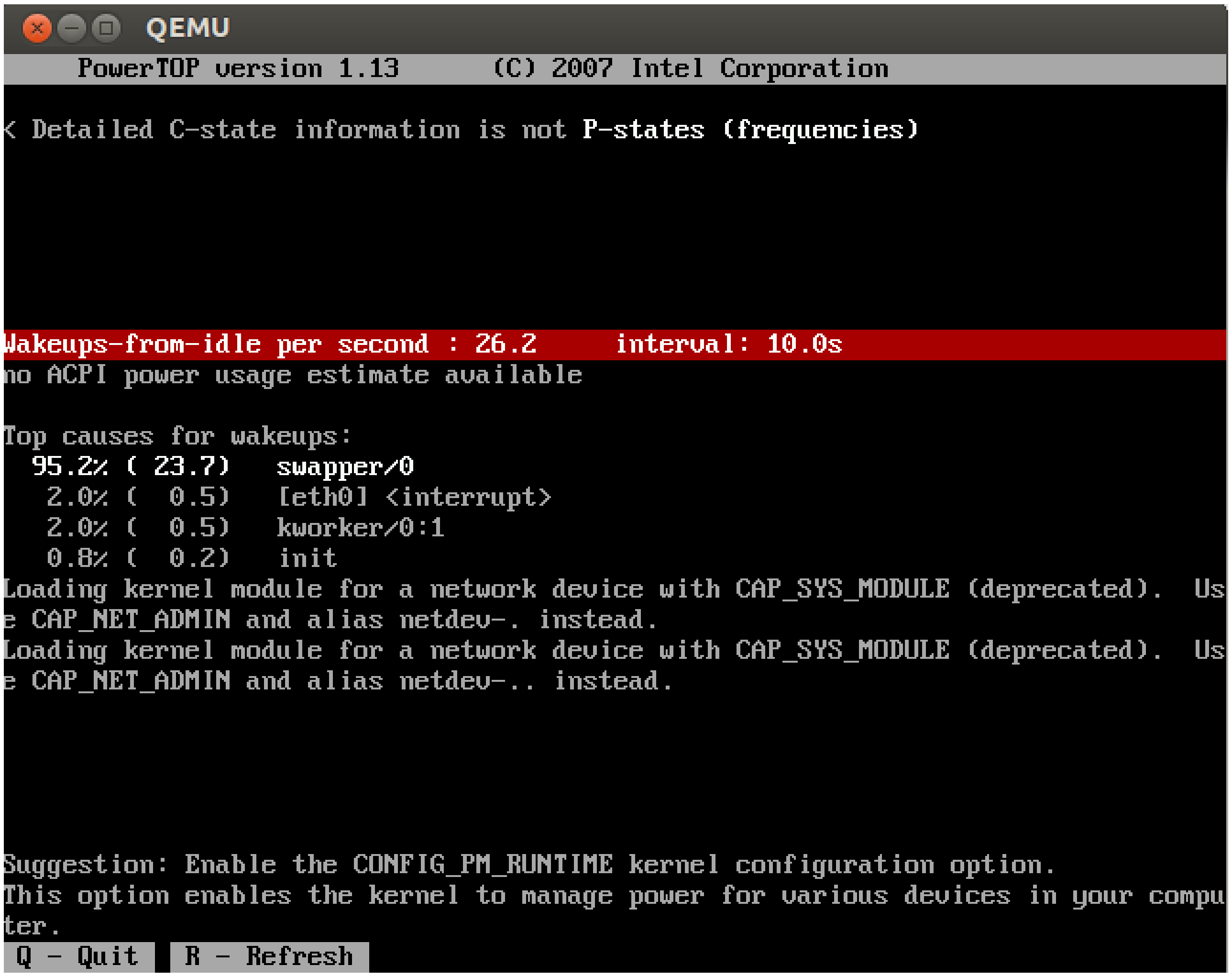
Package name	Size	Included
popt-locale-sv	28.0 KB	☐
popt-locale-th	28.0 KB	☐
popt-locale-tr	28.0 KB	☐
popt-locale-uk	28.0 KB	☐
popt-locale-vi	28.0 KB	☐
popt-locale-wa	28.0 KB	☐
popt-locale-zh-cn	28.0 KB	☐
popt-locale-zh-tw	28.0 KB	☐
popt-staticdev	220.0 KB	☐
powertop-1.13-r4		✓
powertop	88.0 KB	✓
powertop-dbg	412.0 KB	☐
powertop-dev	0 B	☐
powertop-doc	24.0 KB	☐
powertop-locale-de	44.0 KB	☐
powertop-locale-es	44.0 KB	☐
powertop-locale-fi	40.0 KB	☐
powertop-locale-fr	44.0 KB	☐
powertop-locale-hu	44.0 KB	☐

Step 4: Click on "Build image", this will build a new customized image base on core-image-minimal that include powertop. It'll take a while for the build to be done, so please be patient and don't interrupt the build. After the build is successfully done, click on "Run image" and follow the same step as in lab 1 to login qemu as root.

Step 5: After login qemu as root, type

```
# powertop
```

since the new image has powertop, it'll be running and you'll see



```
     PowerTOP version 1.13       (C) 2007 Intel Corporation

< Detailed C-state information is not P-states (frequencies)






Wakeups-from-idle per second : 26.2      interval: 10.0s
no ACPI power usage estimate available

Top causes for wakeups:
  95.2% ( 23.7)    swapper/0
   2.0% (  0.5)    [eth0] <interrupt>
   2.0% (  0.5)    kworker/0:1
   0.8% (  0.2)    init
Loading kernel module for a network device with CAP_SYS_MODULE (deprecated).  Us
e CAP_NET_ADMIN and alias netdev-. instead.
Loading kernel module for a network device with CAP_SYS_MODULE (deprecated).  Us
e CAP_NET_ADMIN and alias netdev-.. instead.






Suggestion: Enable the CONFIG_PM_RUNTIME kernel configuration option.
This option enables the kernel to manage power for various devices in your compu
ter.
 Q - Quit   R - Refresh
```

click on Q to quit powertop. Then close qemu. Quit hob as well. This concludes our lab 2.

Lab 2 Conclusion

In this lab you've used hob and success did an image customization by adding powertop to your customized image that based on core-image-minimal. The steps involved in image customization is: first, you need to select powertop recipe to build the related packages; next, you need to select the wanted packages to be include in the image an build the image.

Lab 3: Create a BSP using yocto-bsp plug-in and hob

Up until now, you have build a predefined image and did a simple image customization using hob. But when you need to do more complicated image and build customization, that is typically a layer will need to be defined. For example, all of yocto BSPs are created using this layer approach.

The yocto-bsp tool is a nice tool of the Yocto Project. It consists of a small set of scripts which create a standardized Yocto BSP layer, including machine configuration, supporting recipes, and **README** files. It allows the user to add (and remove) patches and kernel config fragments to a linux-yocto kernel without having to edit or learn the sordid details of the linux-yocto meta-data.

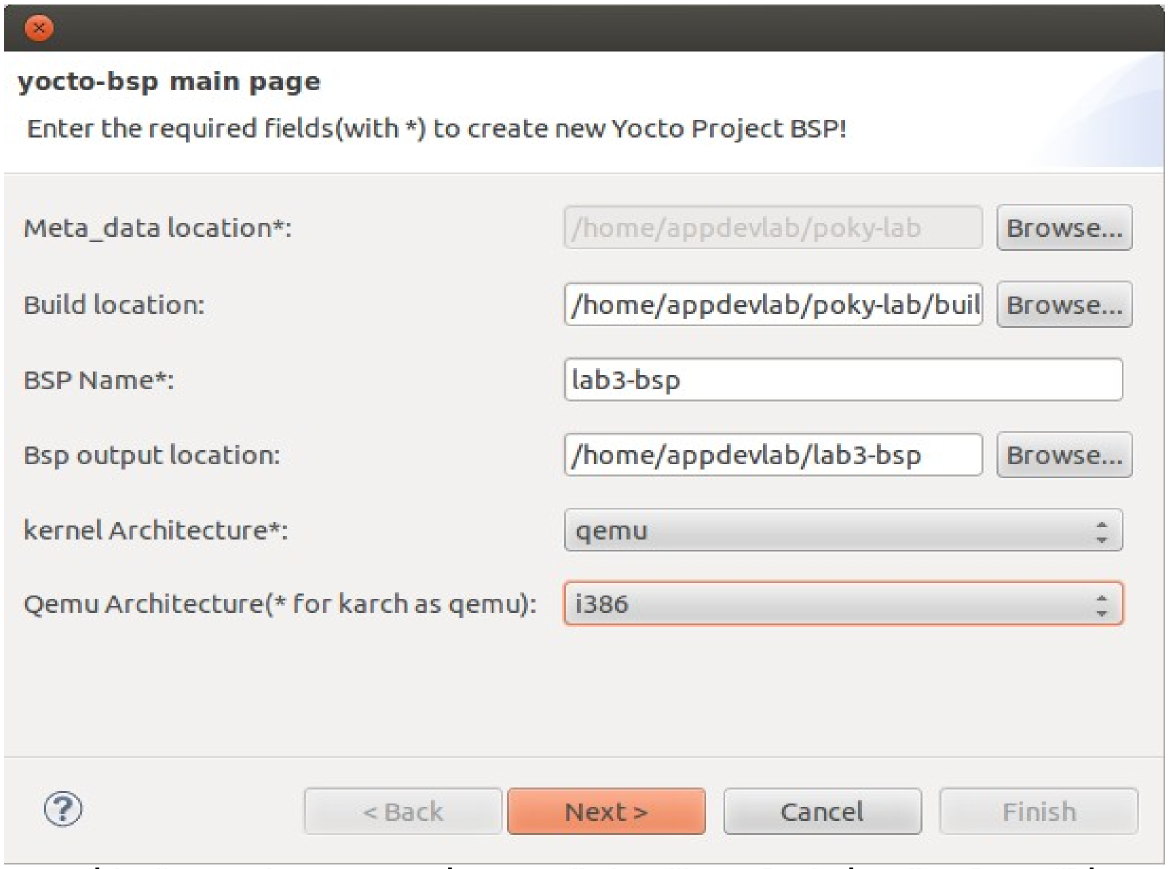
In this lab, we'll be using yocto-bsp eclipse plug-in to create a bsp layer with your customizations. Then modified the bitbake commander project meta data to include the newly defined layer. Finally, we'll use hob to build the customized BSP image and boot it up.

Create the lab3 bsp layer

Inside eclipse IDE, do the following steps:

Step 1: YoctoProjectTools ‣ yocto-bsp to bring up yocto-bsp plug-in interface.

Step 2: Fill in the wizard as following with browse button or type the text, then press "Next>" Note: replacing appdevlab with your true userid



yocto-bsp main page

Enter the required fields(with *) to create new Yocto Project BSP!

Meta_data location*: /home/appdevlab/poky-lab

Build location: /home/appdevlab/poky-lab/buil

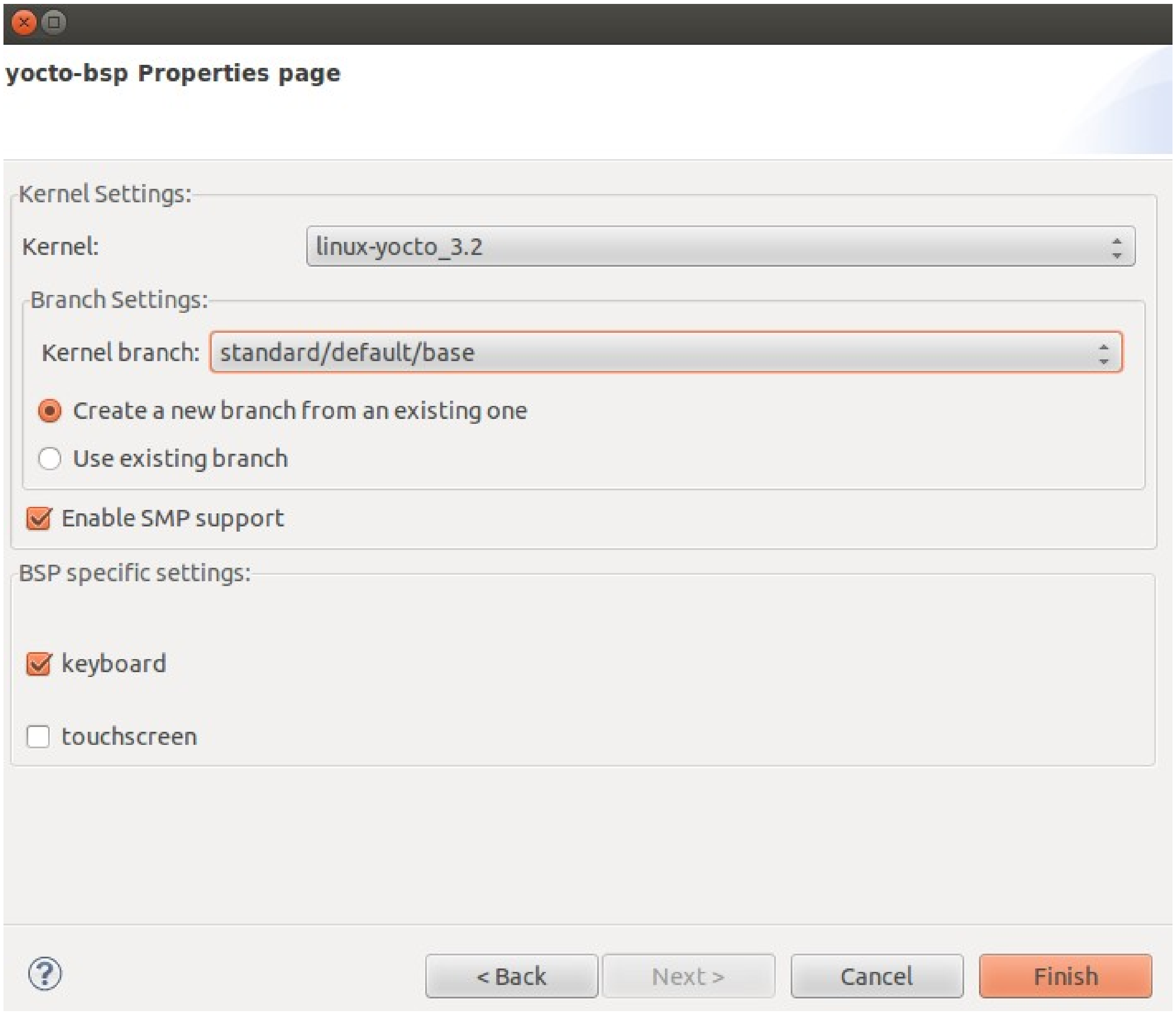
BSP Name*: lab3-bsp

Bsp output location: /home/appdevlab/lab3-bsp

kernel Architecture*: qemu

Qemu Architecture(* for karch as qemu): i386

Step 3: Inside "yocto-bsp properties page" for "kernel choices" select "linux-yocto_3.2", remember during lab1, we noted down the default kernel version is 3.4.6. We'll customize our BSP to the 3.2 kernel version. Then for "Kernel Branch Settings:" we'll select "Existing", which means we'll use an existing kernel tree. In the drop down list, we'll pick "standard/default/base". For everything else, we leave them as it.


yocto-bsp Properties page

Kernel Settings:

Kernel: linux-yocto_3.2

Branch Settings:

Kernel branch: standard/default/base

- [✓] Create a new branch from an existing one
- [ ] Use existing branch

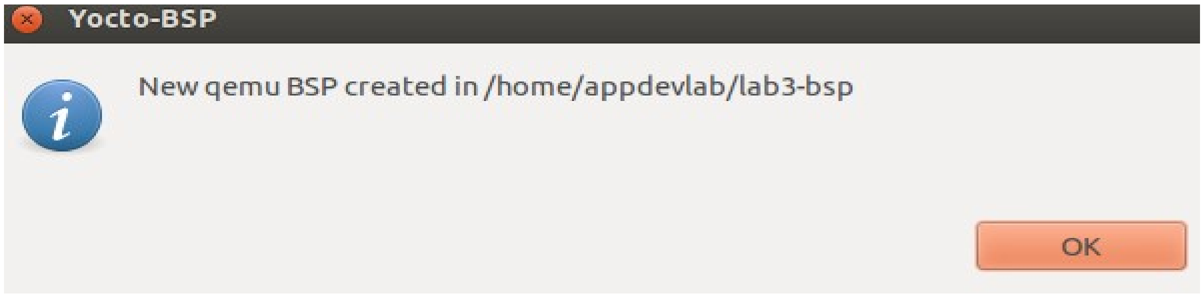
[✓] Enable SMP support

BSP specific settings:

[✓] keyboard

[ ] touchscreen

Step 4: Click on "Finish", and you'll get the following prompt.


Yocto-BSP

New qemu BSP created in /home/appdevlab/lab3-bsp

Congratulations! You've successfully created your bsp layer. Now, dismiss the prompt by click on "OK".

Create the lab3 bsp layer

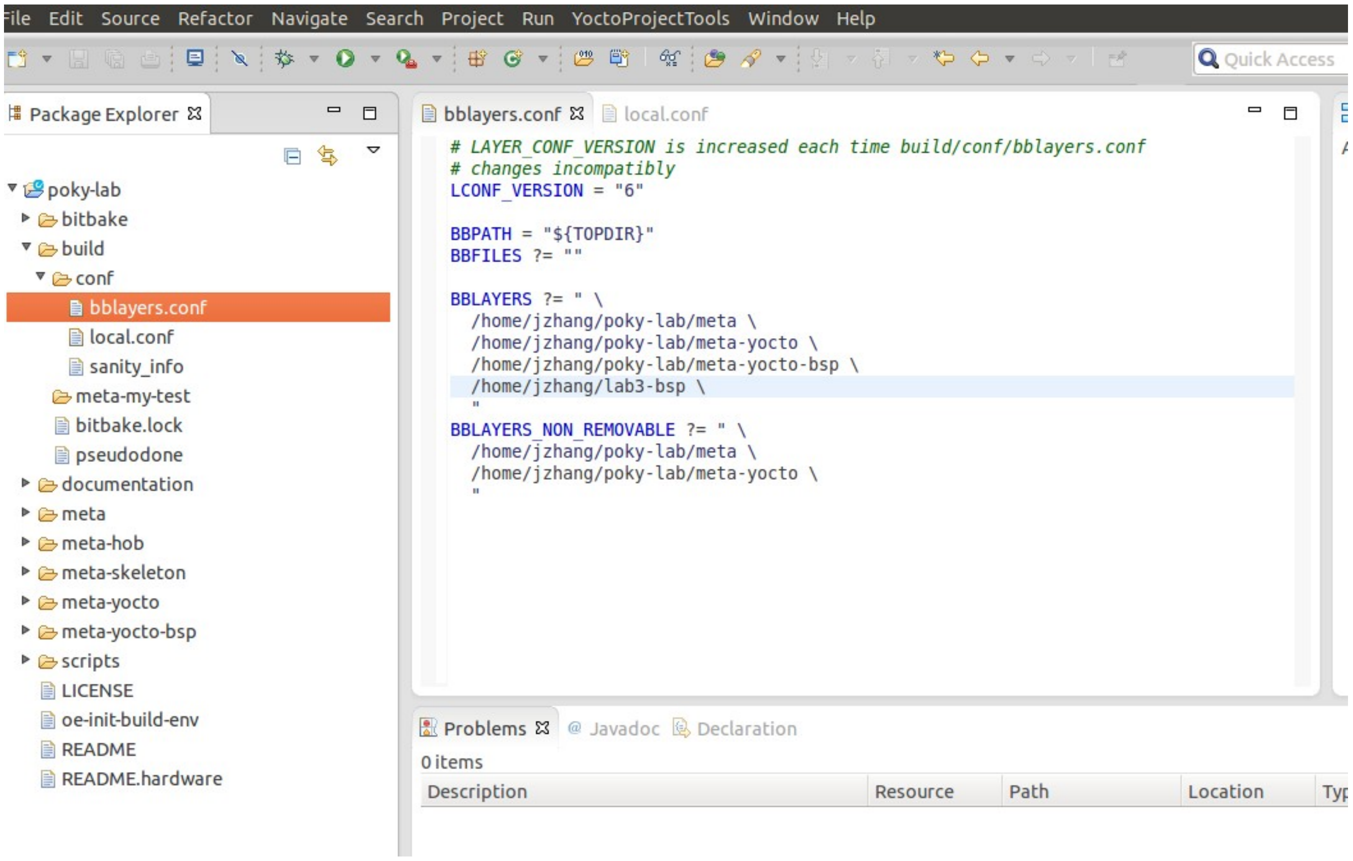
Back to eclipse IDE, we'll be modifying poky-lab meta data by adding the bsp layer information that we've just created. Here's the steps:

Step 1: Extend the poky-lab bitbake commander project in a tree view by clicking on the ▸. Then further extend the build directory under poky-lab and conf directory under build.

Step 2: Double click "bblayers.conf" file to bring its content to the bitbake commander editor. Here append the following line above the ":

```
/home/appdevlab/lab3-bsp \
```

Note: change appdevlab to your user login name. After the change, the contents insides the editor will look as:



```
# LAYER_CONF_VERSION is increased each time build/conf/bblayers.conf
# changes incompatibly
LCONF_VERSION = "6"

BBPATH = "${TOPDIR}"
BBFILES ?= ""

BBLAYERS ?= " \
  /home/jzhang/poky-lab/meta \
  /home/jzhang/poky-lab/meta-yocto \
  /home/jzhang/poky-lab/meta-yocto-bsp \
  /home/jzhang/lab3-bsp \
  "
BBLAYERS_NON_REMOVABLE ?= " \
  /home/jzhang/poky-lab/meta \
  /home/jzhang/poky-lab/meta-yocto \
  "
```

Click on the disk symbol on the tool bar to save the file.

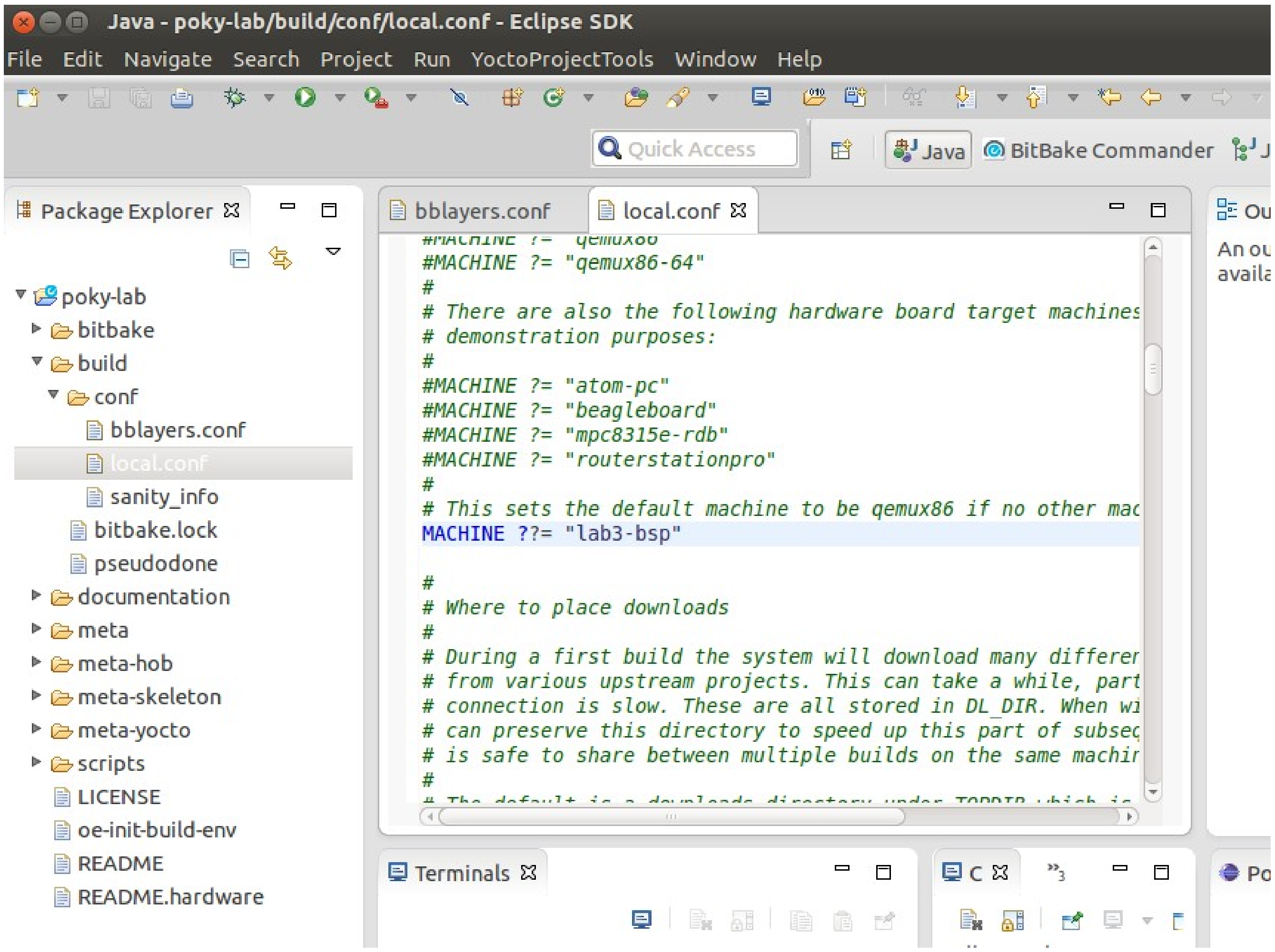
Step 3: Double click "local.conf" file to bring its content to the bitbake commander editor. Scroll down to

```
MACHINE ??= "qemux86"
```

and change it to:

```
MACHINE ??= "lab3-bsp"
```

After the change, the contents insides the editor will look as:



```
#MACHINE ?= "qemux86-64"
#
# There are also the following hardware board target machines
# demonstration purposes:
#
#MACHINE ?= "atom-pc"
#MACHINE ?= "beagleboard"
#MACHINE ?= "mpc8315e-rdb"
#MACHINE ?= "routerstationpro"
#
# This sets the default machine to be qemux86 if no other mac
MACHINE ??= "lab3-bsp"

#
# Where to place downloads
#
# During a first build the system will download many differer
# from various upstream projects. This can take a while, part
# connection is slow. These are all stored in DL_DIR. When wi
# can preserve this directory to speed up this part of subseq
# is safe to share between multiple builds on the same machir
#
```

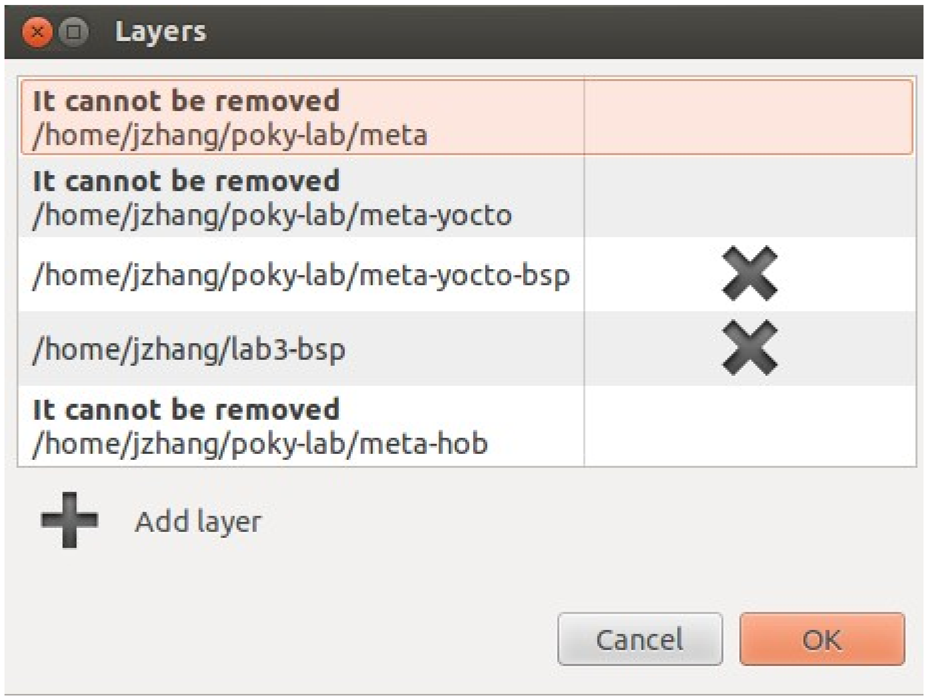
Click on the disk symbol on the tool bar to save the file. After this, the needed modification to include the new bsp layer is done.

Use hob build the bsp image

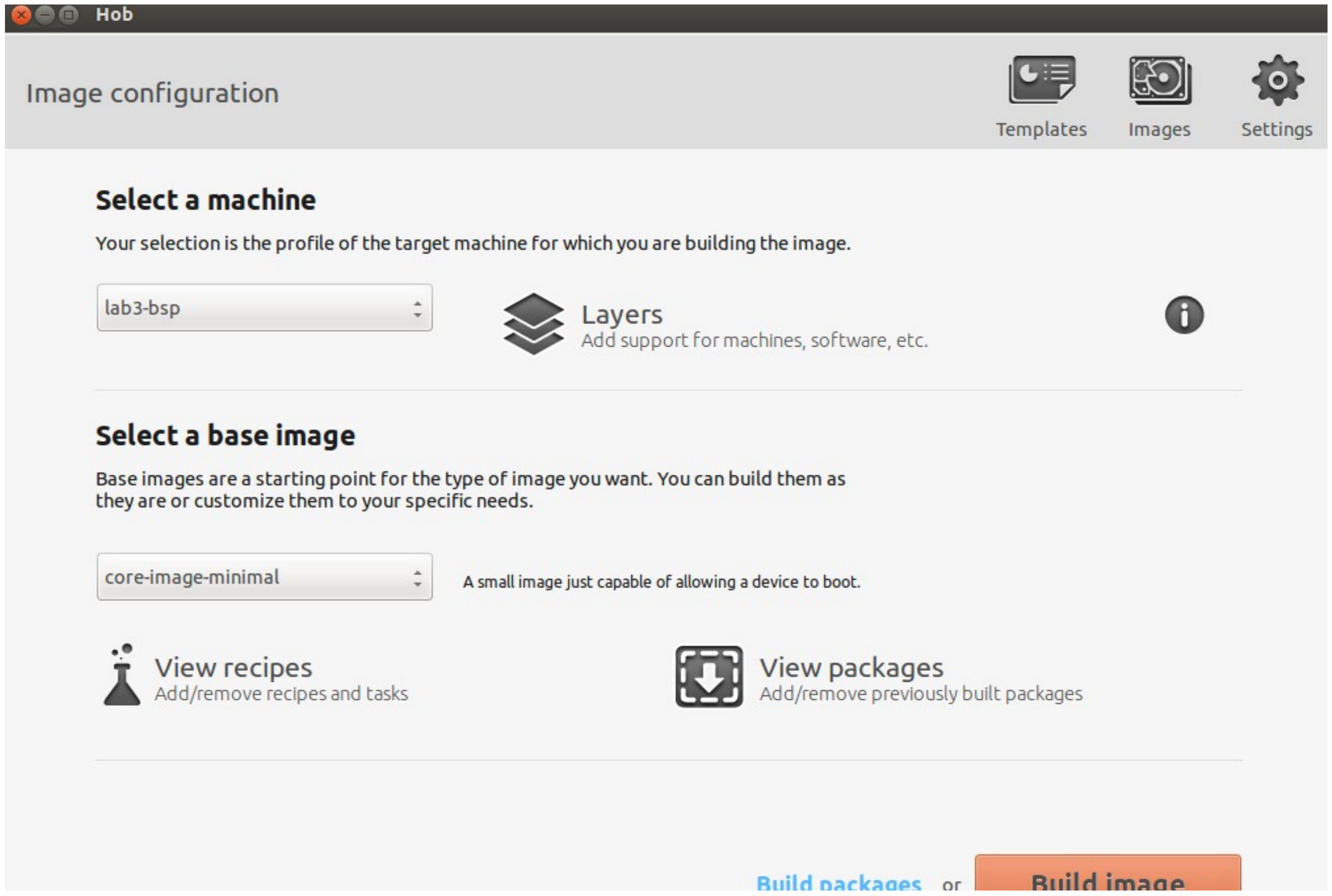
Following these steps to use hob and build out the new bsp image and boot it up to check on our kernel version customization:

Step 1: Highlight "poky-lab" project, then go to " Project ▸ Launch HOB". In "Bitbake build directory:" browse to ensure the selection is made as: /home/appdevlab/poky-lab/build. Then click "OK" to bring up hob.

Step 2: In hob, for "Select a machine" drop down list, you'll notice the "lab3-bsp" is listed. Select it. Click on layer icon. You'll see "/home/appdevlab/lab3-bsp" is listed. All of these reflect the changes we just made to poky-lab meta-data. Dismiss the Layers window by clicking on OK.


Layers
It cannot be removed
/home/jzhang/poky-lab/meta
It cannot be removed
/home/jzhang/poky-lab/meta-yocto
/home/jzhang/poky-lab/meta-yocto-bsp
/home/jzhang/lab3-bsp
It cannot be removed
/home/jzhang/poky-lab/meta-hob
Add layer
Cancel
OK

Step 3: For "Select a base image", still choose "core-image-minimal". Then press "Build Image". It'll bring you to hob build view.



Select a machine

Your selection is the profile of the target machine for which you are building the image.

lab3-bsp

Layers
Add support for machines, software, etc.

Select a base image

Base images are a starting point for the type of image you want. You can build them as they are or customize them to your specific needs.

core-image-minimal

A small image just capable of allowing a device to boot.

View recipes
Add/remove recipes and tasks
View packages
Add/remove previously built packages
Build packages
or
Build image

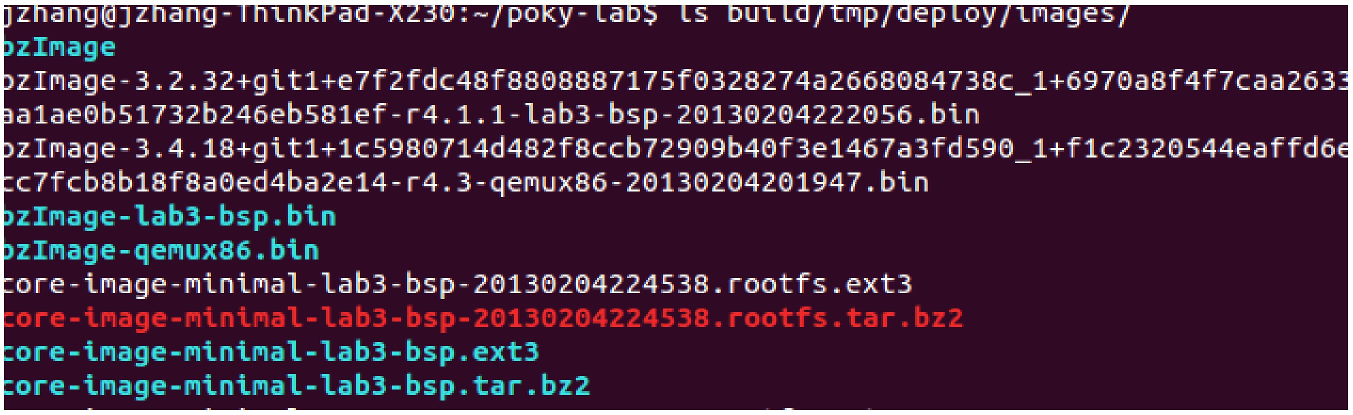
Step 4: Monitoring the build progress in the hob build window. It'll be a while to finish the build, please don't interrupt it. Once the build is successfully

done, exit hob.

Step 5: Bring up a terminal, and type following command:

```
$ cd ~/poky-lab
$ ls build/tmp/deploy/images
```

and you should see the new kernel file and rootfs images file for your new machine matching to your bsp (lab3-bsp) is created:



```
jzhang@jzhang-ThinkPad-X230:~/poky-lab$ ls build/tmp/deploy/images/
bzImage
bzImage-3.2.32+git1+e7f2fdc48f8808887175f0328274a2668084738c_1+6970a8f4f7caa2633
aa1ae0b51732b246eb581ef-r4.1.1-lab3-bsp-20130204222056.bin
bzImage-3.4.18+git1+1c5980714d482f8ccb72909b40f3e1467a3fd590_1+f1c2320544eaffd6e
cc7fcb8b18f8a0ed4ba2e14-r4.3-qemux86-20130204201947.bin
bzImage-lab3-bsp.bin
bzImage-qemux86.bin
core-image-minimal-lab3-bsp-20130204224538.rootfs.ext3
core-image-minimal-lab3-bsp-20130204224538.rootfs.tar.bz2
core-image-minimal-lab3-bsp.ext3
core-image-minimal-lab3-bsp.tar.bz2
```

Lab3 Conclusion

In this lab you used the yocto-bsp tool to generate a complete BSP layer. Modified meta data to include the layer and built the bsp image. This concludes Lab 3.

NOTES

NOTES