



## Advanced Lab

Henry Bruce, Brian Avery, David Reyna, Belen Barros  
Pena, Khem Raj, Mark Hatle, Bruce Ashfield

**Yocto Project Developer Day •  
San Diego • 7 April 2016**

# Agenda – The Advanced Class

8:30- 9:00	Opening Session
9:00- 9:45	Henry - New workflows: Toaster, CROPS and devtool
9:45-10:15	Brian - Deep dive presentation and demo on CROPS
<b>10:15-10:30</b>	<b><i>Morning Break</i></b>
10:30-10:45	Pass out class accounts
10:45-11:30	David - Deep dive and class exercises on devtool
11:30-12:00	David - Mirrors and SState Pitfalls and Practices
<b>12:00- 1:00</b>	<b><i>Lunch</i></b>
1:00- 1:45	Belen - Deep dive on Toaster and related topics
1:45- 2:15	Khem - poky-tiny
<b>2:15- 2:30</b>	<b><i>Afternoon Break</i></b>
2:30- 4:00	Mark - Deep dive on PRServer, User/Group creation
4:00- 4:30	Bruce - Kernel
4:30- 5:00	Q and A

## Notes for the Advanced Class:

- The class will be given with YP-2.0 (Jethro) , on qemuarm (Toaster will use a special latest branch)
- Slides: <http://devday.yocto.link/>
- **Wifi Access:**
  - SSID: Marriott\_CONFERENCE
  - Password: linux1991
- **Your IP access addresses**
  - SSH:  
`ssh ilab01@devday-a.yocto.io -p 10000 (+your session number)`
  - HTTP:  
`http://devday-a.yocto.io:30000 (+your session number)`



## Activity One

# New workflows with Toaster, CROPS and devtool

Henry Bruce



## Activity Two

# CROPS Deep Dive and Demo

**Brian Avery**



## Activity Three

# Class Account and Project Setup

# Yocto Project Dev Day Lab Setup

- You will be given information on how to log into your personal virtual host.
- The virtual host's resources can be found here:
  - Your Projects: `"/scratch/working"`
  - Toaster Install: `"/scratch/master-toaster/poky"`
  - Extensible-SDK Install: `"/scratch/sdk"`
  - Sources: `"/scratch/src"`
  - Yocto Project: `"/scratch/yocto/poky"`
  - Downloads: `"/scratch/downloads"`
  - Sstate-cache: `"/scratch/sstate-cache"`
- You will be using **SSH** to communicate with your virtual server.
- You may want to change the default password after you log on, in case someone accidentally uses the same account address as yours.

# Build your class QEMUARM project

```
$ cd /scratch/working
$ source ../yocto/poky/oe-init-build-env build-qemuarm

# append to local.conf:
$ vi conf/local.conf

MACHINE = "qemuarm"
SSTATE_DIR = "/scratch/sstate-cache"
DL_DIR = "/scratch/downloads"
IMAGE_INSTALL_append = " gdbserver openssh libstdc++"
EXTRA_IMAGEDEPENDS_append = " gdb-cross-arm"
INHIBIT_PACKAGE_STRIP_pn-linux-yocto = "1"
PRSERV_HOST = "localhost:0 "
IMAGE_GEN_DEBUGFS = "1"

$ bitbake core-image-base

# since we have provided the sstate-cache, this should
# only take a few minutes.
```





# Activity Four

## Devtool and Extensible SDKs

David Reyna

# Devtool - A tool for the application developer

Before devtool, developer teams writing new applications had the following options:

1) Use build engineer's setup & write new recipe & use bitbake to rebuild image

## *Drawbacks:*

- push changes to repo each iteration
- Long build times for image rebuilds

2) Use build engineer provided sdk/toolchain

## *Drawbacks:*

- Difficult to update the sdk if app is library or depends on lib in development - may require multiple sdk updates per day
- Doesn't work for testing distro changes (like systemd-related work)  
Can't easily create/test the updated package as built by build engineer

3) Use **externalsrc** to work in own sandbox, building with recipes

## *Drawbacks:*

- Difficult to get all the details right **UNTIL devtool**

# devtool - Baking in a sandbox

**Class will cover these usecases for devtool**

- **Setup using an extensible SDK**
- **Development cycle with a new recipe**
  - Create a recipe from a source tree, then we will build, deploy, edit, build, and deploy
- **Development cycle with existing recipe**
  - Extract recipe and source, the edit, build, and deploy
  - Update the sdk with changes

# devtool - subcommands

## Beginning work on a recipe:

add  
modify  
upgrade

Add a new recipe  
Modify the source for an existing recipe  
Upgrade an existing recipe

## Getting information:

status  
search

Show workspace status  
Search available recipes

## Working on a recipe in the workspace:

build  
edit-recipe  
configure-help  
update-recipe  
reset

Build a recipe  
Edit a recipe file in your workspace  
Get help on configure script options  
Apply changes from external source tree to recipe  
Remove a recipe from your workspace

## Testing changes on target:

deploy-target  
undeploy-target  
build-image

Deploy recipe output files to live target machine  
Undeploy recipe output files in live target  
Build image including workspace recipe packages

## Advanced:

create-workspace  
extract  
sync

Set up workspace in an alternative location  
Extract the source for an existing recipe  
Synchronize the source tree for an existing recipe

## Devtool – Creating an Extensible SDK

- *Include an SSH client on the target to enable the devtool deploy function, with either*

```
EXTRA_IMAGE_FEATURES = "ssh-server-dropbear"
```

*... or ...*

```
IMAGE_INSTALL_append = " openssh"
```

- *Built the Extensible SDK:*

```
$ bitbake core-image-minimal -c populate_sdk_ext
```

```
$ cp tmp/deploy/sdk/poky-glibc-x86_64-core-image-base-armv5e-toolchain-ext-2.0.1.sh /scratch/sdk
```

- **The above steps have been done for you. We just need to extract the SDK:**

```
$ cd /scratch/sdk
```

```
$ ./poky-glibc-x86_64-core-image-base-armv5e-toolchain-ext-2.0.1.sh -d `pwd`
```

# Devtool – Preparing the Extensible SDK

- In clean shell, source both the class environment and the devtool build environment

```
$ cd /scratch/sdk  
$ source environment-setup-armv5e-poky-linux-gnueabi
```

- Note that we have a kernel, but not a rootfs (we will need both for QEMU)

```
$ ls tmp/deploy/images/qemuarm/*qemuarm.ext4  
$
```

- Use devtool to build one

```
$ devtool build-image core-image-base
```

- Now the rootfs is there:

```
$ ls tmp/deploy/images/qemuarm/*qemuarm.ext4  
tmp/deploy/images/qemuarm/core-image-base-qemuarm.ext4  
$
```

# devtool Development Cycle

- **1. Add application to workspace:**  
`devtool add [--version xxx] myapp /path/to/source`
- **2. Build it:**  
`devtool build myapp`
- **3. Write to target device (w/network access):**  
`devtool deploy-target myapp root@ipaddr`
- **4. Edit source code & repeat steps 2-3 as necessary**

# Devtool - hooking your application into the build

- Run the devtool 'add recipe-name /path/to/source'

```
$ devtool add --version 1.0 bballs /scratch/src/bballs
```

- Generates a minimal recipe in the arm-sdk/workspace layer
- Adds EXTERNALSRC in an arm-sdk/workspace/appends bbappend file that points to the source
- ***Note: this does not add your image to the original build engineer's image, which requires changing the platform project's conf/local.conf***

```
IMAGE_INSTALL_append = " bballs"
```



# After the add

## Build files in the sdk directory

```
sdk> ls -FC
buildtools/
cache/
conf/
downloads/
environment-setup-armv5e-poky-
linux-gnueabi
layers/
preparing_build_system.log
site-config-armv5e-poky-linux-
gnueabi
sstate-cache/
sysroots/
tmp/
version-armv5e-poky-linux-gnueabi
workspace/
```

## Workspace layer layout

```
sdk> tree workspace
.
├── appends
│   └── bballs_1.0.bbappend
├── conf
│   └── layer.conf
├── README
└── recipes
    └── bballs
        └── bballs_1.0.bb
```

**4 directories, 4 files**

## Devtool - edit recipe

# Edit the new workspace recipe

```
$ vi workspace/recipes/bballs/bballs_1.0.bb
```

```
do_install () {  
    # NOTE: unable to determine what to put here  
    # - there is a Makefile but no target named  
    # "install", so you will need to define this  
    # yourself  
-   :  
+   install -d ${D}${bindir}  
+   install -m 0755 bballs ${D}${bindir}  
}
```

# Devtool -build/deploy/run app

- **Build the app**

```
$ devtool build bballs
```

- **Deploy the output (*the target's ip address may change*)**

```
$ devtool deploy-target -s bballs root@192.168.7.2
```

*NOTE: the '-s' option will note any ssh keygen issues, allowing you to (for example) add this IP address to the known hosts table*

- **Run app on target**

```
# /usr/bin/bballs
```

# Devtool - iterate

- **Iterate ...**

- edit source file to instantiate more balls

```
$ vi /scratch/src/bballs/b_main.cpp
-int num_hard = 2;
-int num_soft = 2;
-int num_spin = 2;
+int num_hard = 5;
+int num_soft = 5;
+int num_spin = 5;
```

- ...rebuild, redploy, retest, more stuff bouncing

```
$ devtool build bballs
$ devtool deploy-target bballs root@192.168.7.2
```

## devtool - sandbox-to-repo

- **You can use ‘modify’ and ‘update-recipe’ to work with source in your sandbox, and update the sdk/git-repo recipe as a patch/srcrev**
  - `devtool modify -x ...` : extract source from for a recipe in a layer, into your sandbox
  - `iterate`: modify source in sandbox, build , deploy, test
  - `devtool update-recipe...` create a patch to the sdk or commit to the source git repo

# devtool modify: Extract source and modify

- **NOTE: If you do not have git configured for your host, preset some values now**

```
$ git config --global user.email JoeSmith@nowhere.com
$ git config --global user.name "Joe Smith"
```

- **Run the devtool command to extract src and setup sandbox**

```
$ devtool modify -x which /scratch/src/which
```

- **Modify the package in your sandbox**

```
$ vi /scratch/src/which/which.c
- print_usage(stdout);
+ printf("hello class\n");
```

- **On host, rebuild the package and deploy it**

```
$ devtool build which
$ devtool deploy-target which root@192.168.7.2
```

- **On target, demonstrate the change**

```
# which --help
Hello class
```

# devtool update-recipe : Push changes back to sdk/repo layers

- **In your sandbox, commit the change for the tip of the rev**

```
$ cd /scratch/src/which
$ git add which.c
$ git commit -m "changes for class"
```

- **Update-recipe to modify the recipe in the sdk, keeping existing patches**

```
$ devtool update-recipe -n which
```

- **verify that the extensible sdk has been changed**

```
$ grep changes-for-class $(find /scratch/sdk -name "which*bb")
which_2.2.1.bb file:///0001-changes-for-class.patch
$ find /scratch/sdk -name 0001-changes-for-class.patch
... recipes-extended/which/which/0001-changes-for...
```

- **If you reset the recipe, extract 'which' again, you will see the change (but extract to new location)**

## devtool - A few more commands

- We've shown usage for sub-commands add, modify, build, deploy-target (implicitly undeploy-target), runqemu

- **devtool status: shows status of workspace**

```
$ devtool status
bballs: /scratch/src/bballs
which: /scratch/src/which
```

- **devtool package <recipe>: creates installable packages for a recipe**

```
$ devtool package bballs
```

```
...
```

```
NOTE: Your packages are in /scratch/sdk/tmp/deploy/rpm
```

```
$ cd tmp/deploy/rpm/armv5e
```

```
$ ls *bball*
```

```
bballs-1.0-r0.armv5e.rpm
```

```
bballs-dev-1.0-r0.armv5e.rpm
```

```
bballs-debug-1.0-r0.armv5e.rpm
```

```
$
```



## Devtool - A few more commands

- **devtool reset <recipe>**: removes a recipe from the workspace, but not the source tree
- **devtool extract <recipe> <dest-src-tree>**: extracts a recipe's source files to a source tree, but not into workspace, and not ready for building
- **devtool search**: broad regex search into package data, including recipe DESCRIPTION so less useful for packages with names like 'which'

# Yocto devtool - References

## 1. Yocto devtool documentation

<http://www.yoctoproject.org/docs/latest/dev-manual/dev-manual.html#using-devtool-in-your-workflow>

## 2. Tool Author Paul Eggleton's ELC Presentation:

[http://events.linuxfoundation.org/sites/events/files/slides/yocto\\_project\\_dev\\_workflow\\_elc\\_2015\\_0.pdf](http://events.linuxfoundation.org/sites/events/files/slides/yocto_project_dev_workflow_elc_2015_0.pdf)

## 3. Trevor Woerner's Tutorial

<https://drive.google.com/file/d/0B3KGzY5fW7laQmgxVXVTSDJHeFU/view?usp=sharing>



# Activity Five

## Mirrors and SState

David Reyna

# Mirrors and Pre-Mirrors

- **MIRRORS** specifies additional paths from which the build system gets source code. When the build system searches for source code, it first tries the local download directory. If that location fails, the build system tries locations defined by **PREMIRRORS**, the upstream source, and then locations specified by **MIRRORS** in that order.
- **PREMIRROR** is a set of rules applied to URIs prior to fetching. The rules are composed of an RE followed by a substitution rule.
- **BB\_ALLOWED\_NETWORKS** specifies a space-delimited list of hosts that the fetcher is allowed to use to obtain the required source code, with limited RE support, for example:  
`BB_ALLOWED_NETWORKS = "*.gnu.org"`
- **BB\_NO\_NETWORK** disables network access in the BitBake fetcher modules. With this access disabled, any command that attempts to access the network becomes an error.

# Uses for pre-mirrors

- **No Network Access**
  - When you need to completely lock out external content, either for security reasons or code contamination reasons, you can use `BB_NO_NETWORKS`.
- **Limited Network Access**
  - When you want to allow only certain network paths, internal and/or external, for example to manage development repositories, you can use `BB_ALLOWED_NETWORKS`.
- **Managed and Redirected Network Access**
  - When you want to allow substitutions of networks when and if they are available, for example to access internal repositories when in the factory and then public repositories when in the field, you can use `PREMIRROR`.
- **Protocol Swaps**
  - When you want to substitute potentially blocked repository ports with unblocked ports, for example swap git access for HTTP access, using the RE features of `PREMIRROR`.

# How PREMIRROR RE substitution works

- The URL parser decodes an URL into tokens, specifically “scheme”, “network location”, “path”, “user”, “password”, and “parameters”
- The parser will then match the set of PREMIRROR’s rules with the URL on a per token basis.
- The parser will then repeat PREMIRROR’s rules matching on this new set of URLs. This repeats until no more matches are found.
- The result will be a list of the original URLs together with all of the recursive matches.
- Example of a complex rule:



# Infinite PREMIRROR Loop Example

- **The bad news is this facility can lead to infinite loops that can overrun python and even your disk. The good news is this has been fixed in Yocto Project 2.0. The bad news is that it could still happen to you.**

- **Example Rules:**

```
git://.*/* http://A/A_
```

```
http://.*/* http://B/B_
```

- **Let us begin (without fix in 2.0):**

```
git://a.b.com/foo.git -> http://A/A_foo.git
```

```
http://A/A_foo.git -> http://B/B_A_foo.git
```

```
http://B/B_A_foo.git -> http://B/B/B_B_A_foo.git
```

**(loop!)**

# PREMIRROR Advice

- Design your rules so that a conversion rule does not match itself. Here is an example of a set of rules that explicitly blocks self-matching after the first substitution:

```
git://.*!.*(.) https://${CORP_MIRRORS}/gadget/oe-core-dl-1.8/${CORP_MIRRORS_SHOW_BRANCH}:downloads^1
```

```
https://^(?!${CORP_MIRRORS}).*!  
https://${CORP_MIRRORS}/gadget/oe-core-dl-1.8/${CORP_MIRRORS_SHOW_BRANCH}:/
```

- However, with that said, there still can be implicit loops, such that one rule will run over the results of a previous rule.
- And of course it's the sum of PREMIRRORS, <regular>, MIRRORS in the download order. The more rules you put in, the bigger the table, the more matching attempts may be processed before a non-match.



# PREMIRROR

- There is a regression test under “bitbake/lib/bb/tests” that is run to see if the distribution as the issue.
- Here is that test’s output. It does show that the recursion is fixed. Surprisingly, it also showed an extra conversion that was not expected!

```
recmirrorvar = "https://.*/[^/]*      http://AAAA/A/A/A/ \n" \  
              "https://.*/[^/]*      https://BBBB/B/B/B/ \n"  
def test_recursive(self):  
    fetcher = bb.fetch.FetchData("https://downloads.yoctoproject.org/releases/bi  
tbake/bitbake-1.0.tar.g$  
    mirrors = bb.fetch2.mirror_from_string(self.recmirrorvar)  
    uris, uds = bb.fetch2.build_mirroruris(fetcher, mirrors, self.d)  
    self.assertEqual(uris, ['http://AAAA/A/A/A/bitbake/bitbake-1.0.tar.gz',  
                            'https://BBBB/B/B/B/bitbake/bitbake-1.0.tar.gz',
```

# Sstate and Downloads

- **SSTATE**
  - Each build has a sstate-cache directory that collects the generated packages. The build will look in this directory first to avoid unnecessary rebuilding of otherwise unchanged packages
  - In the name of each cached file contains a checksum string, known as the signature. The signature is a calculated sum of identifying dependence artifacts that track if the package source has changed. When the calculated sum matches a cached sum, the cached content is reused, else the package is re-built (and ends up with a new checksum)
- **Downloads**
  - Each build also has a download directory, where all source packages that are not local are copied and placed
  - For the download directory, uniqueness is determined by the source package's name, which normally have version information

# Sstate and Downloads: Sharing and Portability

- **The sstate and download directories can be shared between projects, allowing the work of one project save time for the other projects. This is done by setting the SSTATE\_DIR and the DL\_DIR values in each project to common external directories.**
- **The sstate and download directories can be shared across networks, for example across an NFS mount.**
- **The sstate and download directories are portable, meaning that it can be copied from one machine to another.**
- **The sstate-cache is access-rights safe between users. The bitbake required umask of 022 insures that all such files are readable by all other users.**

# Sstate: Mirrors

- **What we learned about Mirrors can apply Sstate**
- **SSTATE\_MIRRORS**
  - Configures the build system to search other mirror locations for prebuilt cache data objects before building out the data. This variable works like fetcher MIRRORS and PREMIRRORS and points to the cache locations to check for the shared objects.
- **SSTATE\_MIRROR\_ALLOW\_NETWORK**
  - If set to "1", allows fetches from mirrors that are specified in SSTATE\_MIRRORS to work even when fetching from the network has been disabled by setting BB\_NO\_NETWORK to "1". Using the SSTATE\_MIRROR\_ALLOW\_NETWORK variable is useful if you have set SSTATE\_MIRRORS to point to an internal server for your shared state cache, but you want to disable any other fetching from the network.

# Sstate: recommendations

- In complex systems it is recommended to separate sstate directories, for example native and non-native sstate directories, and also different BSPs and arches.
- Reusing a single directory will grow very large very quickly. Use atime to delete old files. Note: this requires the filesystem mounted with atime/relatime which we normally recommend to disable for build performance.

```
find ${sstate_dir} -name 'sstate*' -atime +3 -delete; fi
```

- Rebuild sstate to new directory periodically and delete old sstate dir to maintain bounded size. There may be packages or package versions that are no longer used and just take up space.
- Although it is possible to use other protocols for the sstate such as HTTP and FTP, you should avoid these. Using HTTP limits the sstate to read-only and FTP provides poor performance.
- Additionally, a missing sstate file on http/ftp server cause wget to hang for a long time due to the retries and timeout

# Sstate: Known issues

- **A new changeset may invalidate part or all of the sstate cache occasionally - this is by design**
- **Native package's sstate-cache can't be reused among 32-bit and 64-bit host**
- **Native sstate dependent on glibc version and native sstate compiled with newer glibc version cannot be used on systems with older glibc**
- **The calculated signatures are not always perfect. For example, if a recipe copies files directly into its install directory and those files are not otherwise registered in the FILES directive, any file changes will be invisible to the signature, and even though the package builds the image will be populated by the original sstate version**
- **Users with different PR Servers can cause package extra-version incrementing that is inconsistent, resulting in false wins**
- **Never the less, the sstate-cache feature will save you a tremendous amount of time, and it has proven to be very stable. Use it!**

# Debugging Sstate Issues

- There are two important tools to help debug sstate issues

**Build History:** tells you 'what' changed

**bitbake-diffsigs:** helps you figure out the 'why'

- **Build History**

- The buildhistory class exists to help you maintain the quality of your build output. You can use the class to highlight unexpected and possibly unwanted changes in the build output. When you enable build history, it records information about the contents of each package and image and then commits that information to a local Git repository where you can examine the information.

- **bitbake-diffsigs**

- This tool is a BitBake task signature data comparison utility
- You can use this tool to check any changes between sstate cache and new produced one.



# Debugging Sstate Issues

- **Build History**

- Enabling: build history is disabled by default. To enable it, add the following at the end of your conf/local.conf file:

```
INHERIT += "buildhistory"  
BUILDHISTORY_COMMIT = "1"
```

- The build history information is kept in BUILDHISTORY\_DIR , generally equal to “\${[TOPDIR](#)}/buildhistory”
- The directory tracks build information into image, packages, and SDK subdirectories.

- **bitbake-diffSIGs**

- The usage is:

```
bitbake-diffSIGs -t recipename taskname  
bitbake-diffSIGs sigdatafile1 sigdatafile2  
bitbake-diffSIGs sigdatafile1
```

- The signature files are in the stamps directory, for example:

```
tmp/stamps/armv5e-poky-linux-gnueabi/busybox/*sig*
```





## Activity Six

**Lunch!**



# Activity Seven

## Toaster Deep Dive Belen Barros Pena

# Toaster: It's Magic!





# Activity Eight

**Poky-tiny**  
**Khem Raj**



# Activity Nine

**PR Service**  
**Mark Hatle**

# Advanced Topic: PR Service Introduction

- **Package Managers install/upgrade using:**
  - PKGN-PKGE:PKG V-PKGR
  - PKGN – Package Name
  - PKGE – Package Epoch
  - PKGV – Package Version
  - PKGR – Package Revision
- **Example:**
  - bash-3.2-r0.armv5l.rpm
  - bash-3.2-r1.armv5l.rpm

# Advanced Topic: PR Service Introduction

- **PKG\_N set by recipe author**
  - Default to PN which defaults to recipe filename
- **PKG\_V set by recipe author**
  - Default to PV which defaults to recipe filename
  - Changes to match upstream community
  - PKGE only used when the PV 'scheme' changes
- **PKGR numbers can be manually updated**
  - Based in part on PR
  - Error prone – easy to forget to update
  - What happens when a rebuild was due to dependency change? (PR doesn't change)

# Advanced Topic: PR Service

## PR Numbers

- **PR Server**
  - The build system knows when a package is going to be built
  - It also knows the unique signature of the package steps and dependencies – as well as the PN and PE:PV.
  - This is everything we need to know if the PR should increment
- **Enable with this option:**

```
# add to conf/local.conf:  
PRSERV_HOST = "localhost:0"
```



# Advanced Topic: PR Service

## PR Numbers

- **PR Service uses the PN-PE:PV-PR, package arch and signature to determine the PR value to return**
- **Server is a network service**
  - XML RPC API:
  - `getPR(version, pkgarch, checksum)`, `quit`, `ping`,  
`export(version=None, pkgarch=None, checksum=None, colinfo=True)`, `importone(version, pkgarch, checksum, value)`
- **Values stored in an sqlite DB**
  - Table: `PRMAIN_nohist`
  - Columns: `version`, `pkgarch`, `checksum`, `value`

# Advanced Topic: PR Service

## PR Numbers

- Effectively what happens is the system sends to the server (getPR)  `${PN} - ${EXTENDPE} ${PV} - ${PR} ,  
 ${PACKAGE_ARCH} , ${BB_TASKHASH}`
- PR Server matches each of these and returns the value if known.
- If unknown it matches what it can and assigns the next integer value for that name/version and stores that value for the future and returns the value.
- See `package.bbclass: python package_get_auto_pr()`

## Advanced Topic: PR Service PR Numbers

- **As implemented, values from the PR service are included into the PR field as an addition of the form ".X" so r0 becomes r0.1, r0.2 and so on.**
- **This allows existing PR values to be used for whatever reasons allowing manual PR bumps should it be necessary.**
- **As of Yocto Project 1.7 -- PR numbers are no longer updated in recipes. Existing PR numbers are only removed on a PV change.**

# Advanced Topic: PR Service

## PR Numbers

- How does bitbake handle this?

```
# meta/conf/bitbake.conf
PF = "${PN}-${EXTENDPE}${PV}-${PR}"
EXTENDPE = "${@[',','${PE}_']}" \
           "[int(d.getVar('PE', True) or 0) > 0]"

PKGVS = "${PV}"
PKGR = "${PR}${EXTENDPRAUTO}"
PKGE = "${@[',','${PE}']}[int(d.getVar('PE', True) or 0) > 0]"

PRAUTOINX = "${PF}"

EXTENDPRAUTO = "${@[',','${PRAUTO}']}" \
              "[d.getVar('PRAUTO', True) is None]"
```

## Advanced Topic: PR Service

### PR Numbers and sstate-cache

- **PR information is not maintained as part of the sstate packages.**
- **If you maintain a sstate feed, it's expected that either you run builders contributing to the sstate feed with a shared PR service, or you don't run a shared feed!**
- **A shared feed without a PR service will lead to problems!**
  - Bad, Possibly no re-use of binary packages
  - Worse, re-use of binary packages with names that don't match expected
  - Even worse, PR numbers do not increase

# Advanced Topic: PR Service Exercises

- **-exercise-**
- **Manually inspect the PR server database**
  - `sqlite <db>`
- **Start a PR service multiple builders can use**
- **Show a shared sstate w/o a PR server**
- **Show a shared sstate w/ a PR server**



# Activity Ten

## Users and Groups

Mark Hatle

# Advanced Topic: Users and Groups

## base-passwd

- **It all starts with:**
  - meta/recipes-core/base-passwd/base-passwd\_3.5.29.bb
- **Based on the debian base-passwd package**
- **Includes passwd.master and group.master**
- **update-passwd utility**
- **Differences:**
  - Change default shell to /bin/sh
  - Input user
  - Shutdown user



## Advanced Topic: Users and Groups base-passwd

- **When the base-passwd package installs it installs the passwd.master and group.master files into /usr/share**
- **Post install script “update-passwd” runs.**
- **update-passwd (8):**
  - handles updates of /etc/passwd, /etc/shadow and
  - /etc/group on running Debian systems. It compares the
  - current files to master copies ... and updates all
  - entries in the global system range (that is, 0-99).
- **On a new system it copies the master files as the starting point.**

## Advanced Topic: Users and Groups password policy

- **The Debian policy guide specifies the following:**
  - 0-99: Globally allocated ids (distribution specific)
  - 100-999: Dynamically allocated system ids. These ids are specific to each system install.
  - 1000-59999: Dynamically allocated user accounts
  - 60000-64999: Globally allocated static ids (special purpose)
  - 65000-65533: Reserved
  - 65534: User 'nobody'.
  - 65535: (uid\_t)(-1) == (gid\_t)(-1) may not be used

**See (9.2):**

**<https://www.debian.org/doc/debian-policy/ch-opersys.html>**

## Advanced Topic: Users and Groups base-passwd

- **base-passwd should only contain distribution specific globally allocated static ids**
- **To define additional uid/gid specific to your distribution, create a .bbappend and patch the passwd.master/group.master file**
- **This ensures that upgrading the package will work properly**

# Advanced Topic: Users and Groups

## base-passwd

- **-exercise-**
- **Create .bbappend for base-passwd**

# Advanced Topic: Users and Groups

## useradd.bbclass

- **This class is responsible for adding dynamic uid/gid to the system.**
- **Inherits common functions from useradd\_base**
  - perform\_groupadd(), perform\_useradd(), perform\_groupmems(), perform\_groupdel(), perform\_userdel(), perform\_groupmod(), perform\_usermod()
- **Class adds “preinstall” scripts to recipes (USERADD\_PACKAGES)**
  - useradd\_preinst via pkg\_preinst\_<package>

# Advanced Topic: Users and Groups

## useradd.bbclass

- **Special Variables**
  - USERADD\_PACKAGES (specifies packages w/ params)
  - GROUPADD\_PARAM
  - USERADD\_PARAM
  - GROUPMEMS\_PARAM
- **Each of these takes the same parameters as if the user called groupadd, useradd, or groupmems by itself.**
- **See groupadd(8), useradd(8), groupmems(8)**

# Advanced Topic: Users and Groups Recipes

- Recipes, generally you want recipes to dynamically generated users/groups (security)
- In the case where software requires a static uid/gid one can be added using standard arguments (see policy)
- **Example:**

```
inherit useradd
```

```
USERADD_PACKAGES = "${PN}"
```

```
GROUPADD_PARAM_${PN} = "--system shutdown"
```

```
USERADD_PARAM_${PN} = "--create-home \  
                        --groups video,tty,audio,input,shutdown,disk \  
                        --user-group xuser"
```

## Advanced Topic: Users and Groups pseudo/fakeroot

- When building recipes, or images 'pseudo' intercepts calls to emulate a root like environment
- 'pseudo' has the ability to also emulate users and groups settings
- The sysroot gets a passwd/group file installed which assists
- `bitbake -c devshell` allows you to inspect values



# Advanced Topic: Users and Groups Recipes

- **-exercise-**
- **Add a new recipe w/ customer user/group**
  - Show package pre-install script
  - Show how adding it to an image adjusts the passwd/group
- **Inspect with bitbake –c devshell (vs outside)**

# Advanced Topic: Users and Groups Images

- Often you want image or system specific users/groups.
- `extrausers.bbclass` implements this feature

```
INHERIT += "extrausers"  
  
EXTRA_USERS_PARAMS = "\nuseradd -p '' tester; \  
groupadd developers; \  
userdel nobody; \  
groupdel -g video; \  
groupmod -g 1020 developers; \  
usermod -s /bin/sh tester; \  
"
```

# Advanced Topic: Users and Groups Images

- **-exercise-**
- **Rebuild image with custom project/image specific user/group**

# Advanced Topic: Users and Groups

## Problems with dynamically generated users/groups

- **Problem: Multiple images end up with different passwd/group values**
- **Each installation will contain slightly different passwd/group configurations**
  - Generally not a problem
  - Shared filesystems or reproducible images (image level upgrades) need consistent values

## Advanced Topic: Users and Groups useradd-staticids.bbclass

- **Solution: useradd-staticids.bbclass**
- **Can be included in the project local.conf, or distribution.conf – depending on how you want to define the values.**
- **The class rewrites values:**
  - GROUPADD\_PARAM
  - USERADD\_PARAM
  - GROUPMEMS\_PARAM
- **Values are then hard coded into the packages**

## Advanced Topic: Users and Groups

### `useradd-staticids.bbclass`

- See `update_useradd_static_config(d)` function
- The user defines `USERADD_UID_TABLES` and `USERADD_GID_TABLES`. One or more entries (full path or search the 'bbpath').
- **Format of the file is traditional passwd or group files**
  - Note: password fields are IGNORED. Packages should *never* have hardcoded passwords in them. (Including hashed passwords.)
  - Multiple files, earlier entries are overwritten
  - 'Blank' values use previously defined or recipe values

# Advanced Topic: Users and Groups

## `useradd-staticids.bbclass`

- `update_useradd_static_config()` function
- Read the `USERADD_*_TABLES`.
- Parse the values of `*_PARAM`.
  - The entries set the default values.
  - The table entry is loaded based on user or group name
- Write a new `*_PARAM` using the combine entries.  
This ensures that the specific `passwd/group` entry will be generated as defined... but still left to be dynamic and only installed when the package is.

## Advanced Topic: Users and Groups

### `useradd-staticids.bbclass`

- **Easiest way to generate the file is to build a full image and start with the generated passwd/group files**
  - Tailor them as necessary to define the basic settings
  - Pass them back in via the `USERADD_*_TABLE` variables
- **It's a good idea to split up the file into recipe components if appropriate**
  - Can make layer management easier



## Advanced Topic: Users and Groups useradd-staticids.bbclass

- **USERADD\_ERROR\_DYNAMIC = '1'**
- **Trigger a failure if the system encounters a user or group that has not been defined in a USERADD\_\*\_TABLE.**

# Advanced Topic: Users and Groups

## useradd-staticids

- **-exercise-**
  - Generate base file (start with previous image)
  - Modify the uid/gid and put it in your local layer, set TABLES
  - Rebuild image and inspect
  - Enable the no dynamic ids mode (add a pkg w/ dynamic id)
  - Add to another uid/gid and show the error



# Activity Eleven


## Advanced Kernel Topics

### Bruce Ashfield



yocto  
PROJECT™

## Questions and Answers



**Thank you for your  
participation!**

**yocto** ·  
PROJECT

 THE  
**LINUX**  
FOUNDATION



## **Bonus Activity Twelve**

### **Tune Files - Toolchain Flags (CFLAGS, LDFLAGS, ASFLAGS, etc)**

**Mark Hatle**

# Advanced Topic: Tunings and Toolchain Flags

## The environment

- **When building a recipe the environment is configured with a series of special environment variables:**
  - AS, AR, CC, CPP, CXX, LD, CCLD, ...
  - CPPFLAGS – C Pre-Processor flags
  - CFLAGS – C Compiler flags
  - CXXFLAGS – C++ Compiler flags
  - LDFLAGS – Linker flags
- **Flags are defined by the build system, recipes, or the software being compiled.**

# Advanced Topic: Tunings and Toolchain Flags

## The environment: Tools

- **Defined in meta/conf/bitbake.conf, controlled by overrides and bbclasses.**

```
CC = "${CCACHE}${HOST_PREFIX}gcc ${HOST_CC_ARCH}${TOOLCHAIN_OPTIONS}"
CXX = "${CCACHE}${HOST_PREFIX}g++ ${HOST_CC_ARCH}${TOOLCHAIN_OPTIONS}"
CPP = "${HOST_PREFIX}gcc -E${TOOLCHAIN_OPTIONS} ${HOST_CC_ARCH}"
LD = "${HOST_PREFIX}ld${TOOLCHAIN_OPTIONS} ${HOST_LD_ARCH}"
CCLD = "${CC}"
AR = "${HOST_PREFIX}ar"
AS = "${HOST_PREFIX}as ${HOST_AS_ARCH}"

TOOLCHAIN_OPTIONS = " --sysroot=${STAGING_DIR_TARGET}"

HOST_CC_ARCH = "${TARGET_CC_ARCH}"
HOST_LD_ARCH = "${TARGET_LD_ARCH}"
HOST_AS_ARCH = "${TARGET_AS_ARCH}"
```



# Advanced Topic: Tunings and Toolchain Flags

## The environment: Tools

```
TARGET_CC_ARCH = "${TUNE_CCARGS}"  
TARGET_LD_ARCH = "${TUNE_LDARGS}"  
TARGET_AS_ARCH = "${TUNE_ASARGS}"
```

- **TUNE\_\*ARGS** is the required arguments to produce binaries with the right ABI, instruction set and instruction optimization
- Defines the minimum requires arguments for the tool
- Does NOT contain general optimizations
- Based on 'TUNE\_FEATURES', which are arch specific

# Advanced Topic: Tunings and Toolchain Flags

## Tune Files

- **Tune files are located in meta/conf/machine/include**
  - See README for more details
- **Selected by MACHINE .conf file and DEFAULTTUNE**

```
DEFAULTTUNE ?= "core2-64"  
require conf/machine/include/tune-core2.inc
```

# Advanced Topic: Tunings and Toolchain Flags

## Tune Files

```
# tune-core2.inc
# Include the previous tune to pull in PACKAGE_EXTRA_ARCHS
require conf/machine/include/tune-i586.inc

# Extra tune features
TUNEVALID[core2] = "Enable core2 specific processor optimizations"
TUNE_CCARGS .= "${@bb.utils.contains("TUNE_FEATURES", "core2",
    " -march=core2 -mtune=core2 -msse3 -mfpmath=sse", "", d)}"

AVAILTUNES += "core2-64"
TUNE_FEATURES_tune-core2-64 = "${TUNE_FEATURES_tune-x86-64} core2"
BASE_LIB_tune-core2-64 = "lib64"
TUNE_PKGARCH_tune-core2-64 = "core2-64"
PACKAGE_EXTRA_ARCHS_tune-core2-64 = \
    "${PACKAGE_EXTRA_ARCHS_tune-x86-64} core2-64"
```

# Advanced Topic: Tunings and Toolchain Flags

## Tune Files

```
# tune-i586.inc
require conf/machine/include/x86/arch-x86.inc

# tune-x86.inc
# ELF64 ABI
TUNEVALID[m64] = "IA32e (x86_64) ELF64 standard ABI"
TUNECONFLICTS[m64] = "m32 mx32"
TUNE_ARCH .= "${@bb.utils.contains("TUNE_FEATURES", "m64", \
    "x86-64", "", ,d)}"
TUNE_CCARGS .= "${@bb.utils.contains("TUNE_FEATURES", "m64", \
    "-m64", "", ,d)}"

AVAILTUNES += "x86-64"
TUNE_FEATURES_tune-x86-64 = "m64"
BASE_LIB_tune-x86-64 = "lib64"
TUNE_PKGARCH_tune-x86-64 = "x86_64"
PACKAGE_EXTRA_ARCHS_tune-x86-64 = "${TUNE_PKGARCH_tune-x86-64}"
```

# Advanced Topic: Tunings and Toolchain Flags

## The environment: Flags

- **Defined in meta/conf/bitbake.conf**

```
CPPFLAGS = "${TARGET_CPPFLAGS}"
TARGET_CPPFLAGS = ""

CFLAGS = "${TARGET_CFLAGS}"
TARGET_CFLAGS = "${TARGET_CPPFLAGS} ${SELECTED_OPTIMIZATION}"

CXXFLAGS = "${TARGET_CXXFLAGS}"
TARGET_CXXFLAGS = "${TARGET_CFLAGS}"

LDFLAGS = "${TARGET_LDFLAGS}"
TARGET_LDFLAGS = "-Wl,-O1 ${TARGET_LINK_HASH_STYLE}"
```

# Advanced Topic: Tunings and Toolchain Flags

## The environment: Flags

- **Defined in meta/conf/bitbake.conf**

```
SELECTED_OPTIMIZATION = "${@d.getVar(['FULL_OPTIMIZATION',  
    'DEBUG_OPTIMIZATION'])[d.getVar('DEBUG_BUILD', True)  
    == '1'], True)}"  
  
FULL_OPTIMIZATION = "-O2 -pipe ${DEBUG_FLAGS}"  
DEBUG_OPTIMIZATION = "-O -fno-omit-frame-pointer ${DEBUG_FLAGS} -pipe"  
  
DEBUG_FLAGS ?= "-g -feliminate-unused-debug-types"
```

- **DEBUG\_BUILD switches behavior**

# Advanced Topic: Tunings and Toolchain Flags

## The environment: Flags

- **Linux kernel recipe**
- **Kernel ignores all userspace CFLAGS and arguments**
- ***If* they are needed, a special KERNEL\_CC or KERNEL\_LD is required.**

```
meta/recipes-kernel/linux/linux-yocto.inc:
```

```
KERNEL_CC_append_aarch64 = " ${TOOLCHAIN_OPTIONS}"
```

```
KERNEL_LD_append_aarch64 = " ${TOOLCHAIN_OPTIONS}"
```

# Advanced Topic: Tunings and Toolchain Flags

## Exercise

- **-exercise-**
- **Change the MACHINE 'defaulttune' watch the change (bitbake -e)**
- **Adjust the DEBUG\_BUILD value (bitbake -e)**
- **Adjust DEBUG FLAGS, and/or the optimizations**
- **Generate an SDK, see how the settings expand into the SDK**





# Appendix

## Board Setup Quick Start

# Beaglebone Black - Setup

- **Create project directory, update local.conf and bblayers.conf**

```
$ export INSTALL_DIR=`pwd`
$ git clone -b jethro git://git.yoctoproject.org/poky
$ source poky/oe-init-build-env `pwd`/build_beagle
$ echo 'MACHINE = "beaglebone"' >> conf/local.conf
$ echo 'IMAGE_INSTALL_append = " gdbserver openssh"' \
  >> conf/local.conf
$ echo 'EXTRA_IMAGEDEPENDS_append = " gdb-cross-arm"' \
  >> conf/local.conf
$ bitbake core-image-base
```

- **Nothing to change in bblayers.conf, beaglebone is supported in meta-yocto-bsp**

# BeagleBone Black - MicroSD

```
# Format blank SD Card for Beaglebone Black
$ export DISK=/dev/sd[c] <<<Use dmesg to find the actual device name
$ sudo umount ${DISK}1 <<<Note the addition of the '1'
$ sudo dd if=/dev/zero of=${DISK} bs=512 count=20
$ sudo sfdisk --in-order --Linux --unit M ${DISK} <<-__EOF__
1,12,0xE,*
,,-
__EOF__
$ sudo mkfs.vfat -F 16 ${DISK}1 -n boot
$ sudo mkfs.ext4 ${DISK}2 -L rootfs

# Now unplug and replug your SD Card for automount
$ cd tmp/deploy/images/beaglebone
$ sudo cp -v MLO-beaglebone /media/guest-mXlApE/BOOT/MLO
$ sudo cp -v u-boot.img /media/guest-mXlApE/BOOT/
$ sudo tar xf core-image-base-beaglebone.tar.bz2 \
  -C /media/guest-mXlApE/rootfs
$ sync (flush to device, not necessary, but illustrative)
$ umount /media/guest-mXlApE/rootfs /media/guest-mXlApE/boot
```

# BeagleBone Black: GPIO Layout

P9

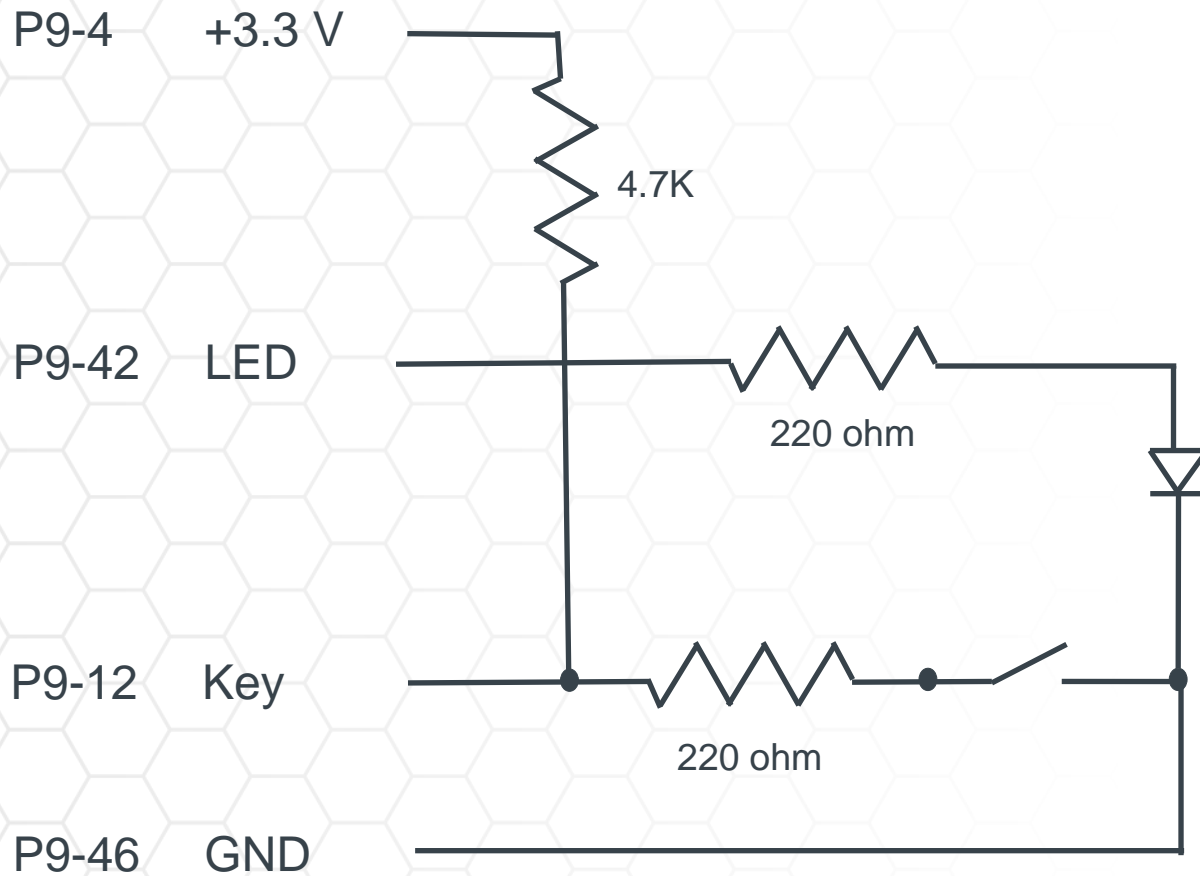
DGND	1	2	DGND	
VDD_3V3	3	4	VDD_3V3	--- +3.3V
VDD_5V	5	6	VDD_5V	
SYS_5V	7	8	SYS_5V	
PWR_BTN	9	10	SYS_RESETN	
GPIO_30	11	12	GPIO_60	--- KEY
GPIO_31	13	14	GPIO_40	
GPIO_48	15	16	GPIO_51	
GPIO_4	17	18	GPIO_5	
I2C2_SCL	19	20	I2C2_SDA	
GPIO_3	21	22	GPIO_2	
GPIO_49	23	24	GPIO_15	
GPIO_117	25	26	GPIO_14	
GPIO_125	27	28	GPIO_123	
GPIO_121	29	30	GPIO_122	
GPIO_120	31	32	VDD_ADC	
AIN4	33	34	GNDA_ADC	
AIN6	35	36	AIN5	
AIN2	37	38	AIN3	
AIN0	39	40	AIN1	
GPIO_20	41	42	GPIO_7	--- LED
DGND	43	44	DGND	
DGND	45	46	DGND	--- GND



Note: GPIO\_20 at least is not actually free

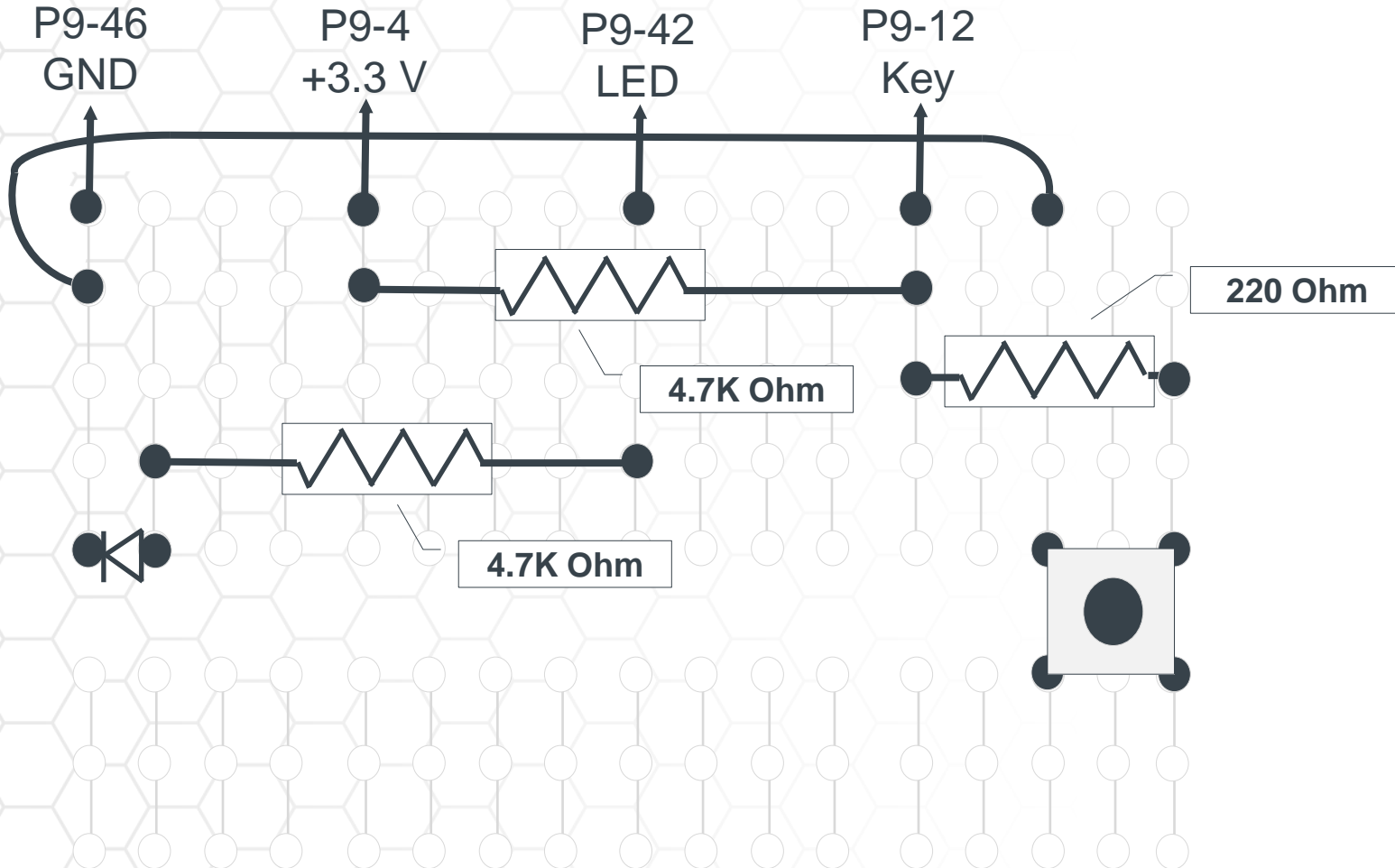
P9

# Schematic: Beaglebone Black LED and Button



220 ohm=red,red,brown 4.7K=yellow,violet,orange

# Sample Breadboard Layout: Beaglebone Black



220 ohm=red,red,brown 4.7K=yellow,violet,orange

# Minnowboard Turbot/Max - Setup

- **Create project directory, update local.conf**

```
$ export INSTALL_DIR=`pwd`
$ git clone -b jethro git://git.yoctoproject.org/poky
$ git clone -b jethro git://git.yoctoproject.org/meta-intel
$ source poky/oe-init-build-env `pwd`/build_minnow
$ echo 'BBLAYERS += "$INSTALL_DIR/meta-intel"' >> conf/bblayers.conf
$ echo 'MACHINE = "intel-corei7-64"' >> conf/local.conf
$ echo 'IMAGE_INSTALL_append = " gdbserver openssh"' \
  >> conf/local.conf
$ echo 'EXTRA_IMAGEDEPENDS_append = " gdb-cross-x86_64"' \
  >> conf/local.conf
$ bitbake core-image-base
```

- **The Minnowboard Turbot and Max use the same BSP**

# Minnowboard Turbot/Max - MicroSD

- **To build a brand new microSD or USB Flash drive image:**

- Get the .hddimg from your build:

```
$ cp tmp/deploy/images/intel-corei7-64/core-image-minimal-intel-corei7-64.hddimg .
```

- Insert your microSD or USB flash drive:

```
$ umount /media/boot
```

```
$ dd if=core-image-base-intel-corei7-64.hddimg of=/dev/sd(n)
```

<<<replace with proper dev

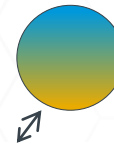


# Minnowboard Turbot/Max - Bootup

- **Booting the board**
    - Insert the new MicroSD card
- ```
Shell> connect -r  
Shell> map -r  
Shell> fs0:  
Shell> bootx64
```
- **That should get you to a Linux login prompt**

# Minnowboard Max: GPIO Layout

- [http://www.elinux.org/Minnowboard:MinnowMax#Low\\_Speed\\_Expansion\\_.28Top.29](http://www.elinux.org/Minnowboard:MinnowMax#Low_Speed_Expansion_.28Top.29)

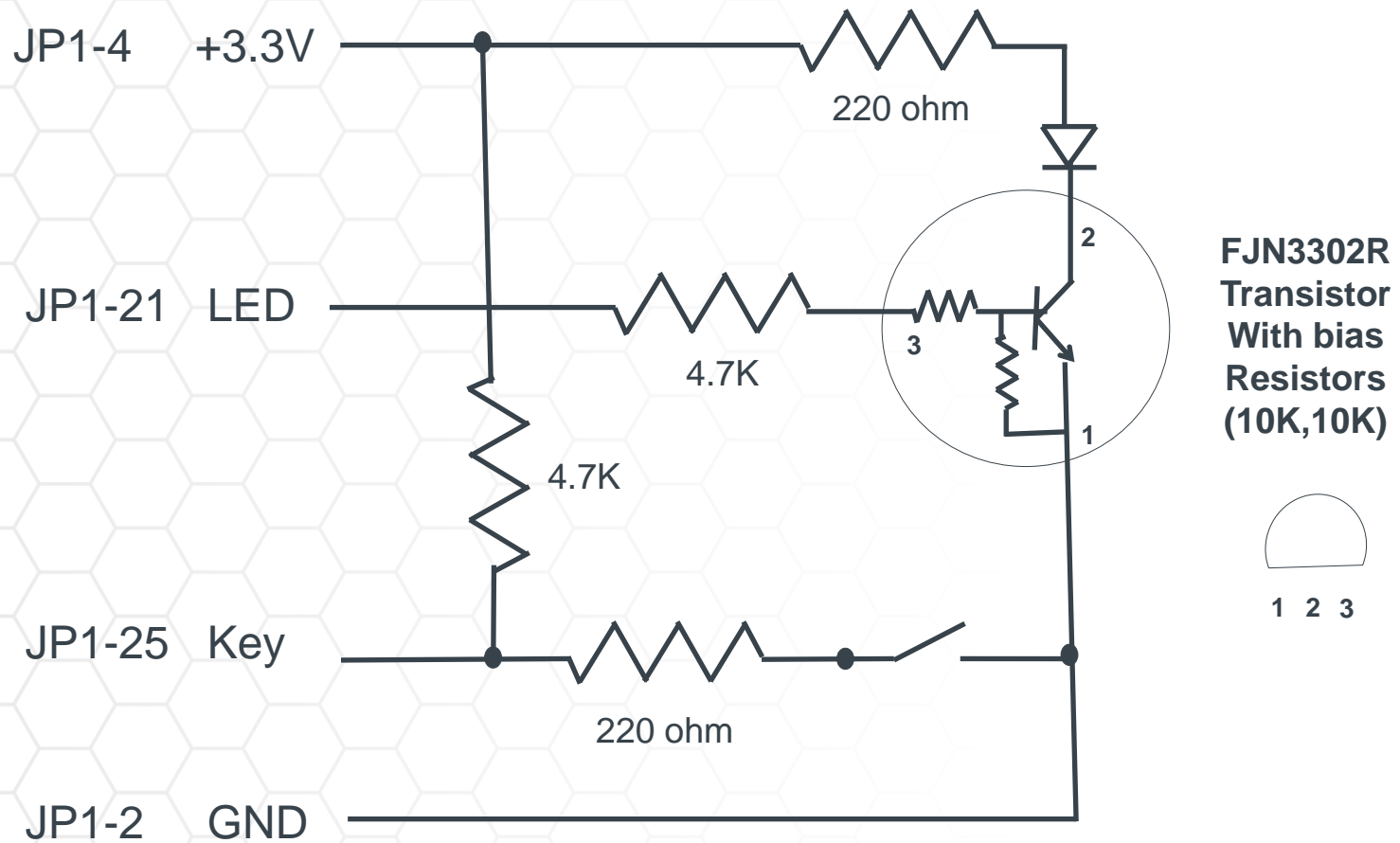


← Power Connector

| Description           | Name                   | Pin | Linux  | Linux  | Pin | Name           | Description     |
|-----------------------|------------------------|-----|--------|--------|-----|----------------|-----------------|
|                       |                        |     | GPIO # | GPIO # |     |                |                 |
| Ground                | Gnd                    | 1   |        |        | 2   | Gnd            | Ground          |
| +5V Power             | VCC                    | 3   |        |        | 4   | +3V3           | + 3.3V Power    |
| SPI Chip Select 1     | GPIO_SPI_CS#           | 5   | 220    | 225    | 6   | GPIO_UART1_TXD | UART Transmit   |
| Master In / Slave Out | GPIO_SPI_MISO          | 7   | 221    | 224    | 8   | GPIO_UART1_RXD | UART Receive    |
| Master Out / Slave In | GPIO_SPI_MOSI          | 9   | 222    | 227    | 10  | GPIO_UART1_CTS | CTS / GPIO      |
| SPI Clock             | GPIO_SPI_CLK           | 11  | 223    | 226    | 12  | GPIO_UART1_RTS | RTS / GPIO      |
| Clock / GPIO          | GPIO_I2C_SCL (I2C #5)  | 13  | 243    | 216    | 14  | GPIO_I2S_CLK   | Clock / GPIO    |
| Data / GPIO           | GPIO_I2C_SDA (I2C #5)  | 15  | 242    | 217    | 16  | GPIO_I2S_FRM   | Frame / GPIO    |
| UART Transmit / GPIO  | GPIO_UART2_TXD         | 17  | 229    | 219    | 18  | GPIO_I2S_DO    | Data Out / GPIO |
| UART Receive / GPIO   | GPIO_UART2_RXD         | 19  | 228    | 218    | 20  | GPIO_I2S_DI    | Data In / GPIO  |
| GPIO / Wakeup         | <b>LED--</b> GPIO_S5_0 | 21  | 82     | 248    | 22  | GPIO_PWM0      | PWM / GPIO      |
| GPIO / Wakeup         | GPIO_S5_1              | 23  | 83     | 249    | 24  | GPIO_PWM1      | PWM / GPIO      |
| GPIO / Wakeup         | <b>KEY--</b> GPIO_S5_4 | 25  | 84     | 208    | 26  | GPIO_IBL_8254  | Timer / GPIO    |

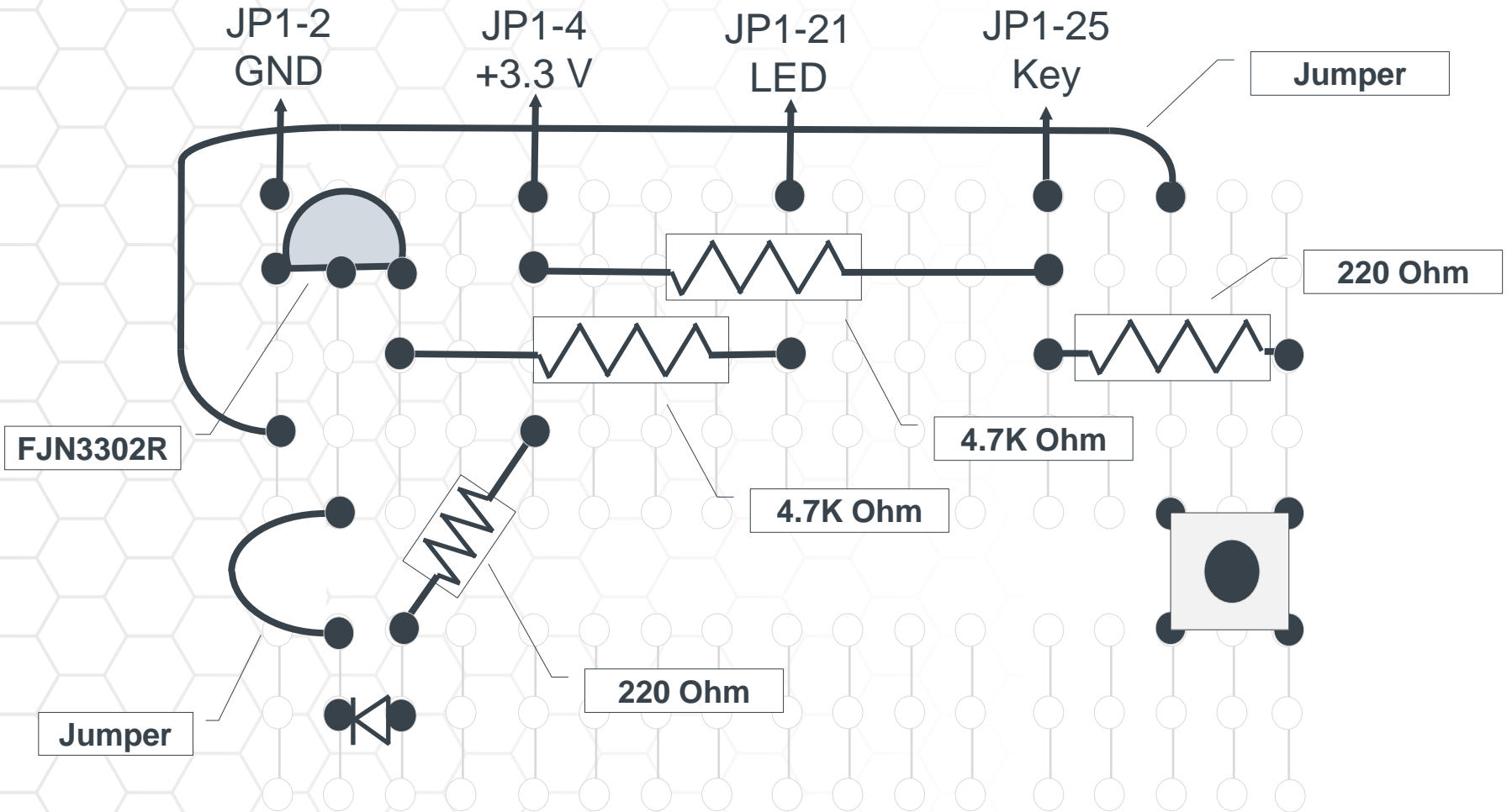
Note: The pins 21, 23, and 25 are the free GPIO pins.

# Schematic: Minnowboard Max LED and Button



220 ohm=red,red,brown 4.7K=yellow,violet,orange

# Sample Breadboard Layout: Minnowboard Max



220 ohm=red,red,brown 4.7K=yellow,violet,orange

# Dragonboard 410c - Setup

- **The Dragon Board is new to Yocto Project. See this URL to see instructions on how to install Jethro.**

<https://github.com/96boards/documentation/wiki/Dragonboard-410c-OpenEmbedded-and-Yocto>

- **To get a serial boot console, you will need to get a specialized FTDI cable. Here are some sources:**

<https://www.96boards.org/products/accessories/debug/>

- **For the slow GPIO bus (at 1.8V), it is recommended to use a protected and/or voltage shifting shield, for example the new Grove baseboard for the Dragonboard**