

# Yocto Project Tools

Bryan Thomas

Field Applications Engineer

---

October 2018 | AMF-AUT-T3399



SECURE CONNECTIONS  
FOR A SMARTER WORLD

Company External – NXP, the NXP logo, and NXP secure connections for a smarter world are trademarks of NXP B.V. All other product or service names are the property of their respective owners. © 2018 NXP B.V.

# Agenda

---

- BitBake
- Layers & BSP Layers
- Kernel Development Workflow
- Autotools Recipes
- Finding Help



# BitBake



## Yocto Project: BitBake

BitBake is a build engine that follows recipes in a specific format in order to perform sets of tasks. BitBake is a core component of the Yocto Project.



# Yocto Project: BitBake Operation

- bitbake must always be executed from within the <project> directory
- When bitbake runs
  - Parses recipes and tasks
  - Determines task queue dependencies
  - Prepares and executes a run queue of tasks, which perform the steps needed to obtain the desired result, e.g. image generation
- Any required earlier tasks will be run first (e.g. source will be installed before compilation)
- To speed up subsequent builds, generated <pkg>.rpm's are saved to the binary cache folders in : <project>/tmp/deploy/rpm

For more info:

- <http://www.yoctoproject.org/docs/1.6/bitbake-user-manual/bitbake-user-manual.html#bitbake-user-manual-execution>

# Yocto Project: BitBake Syntax

\$ bitbake -h

Usage: bitbake [options] [recipeName/target ...]

Executes the specified task (default is 'build') for a given set of target recipes (.bb files).  
It is assumed there is a conf/bblayers.conf available in cwd or in BBPATH which  
will provide the layer, BBFILES and other configuration information.

Options:

- version        show program's version number and exit
- h, --help       show this help message and exit
- k, --continue   Continue as much as possible after an error. While the  
                  target that failed and anything depending on it cannot  
                  be built, as much as possible will be built before  
                  stopping.
- f, --force      Force the specified targets/task to run (invalidating  
                  any existing stamp file).
- c CMD, --cmd=CMD Specify the task to execute. The exact options  
                  available depend on the metadata. Some examples might  
                  be 'compile' or 'populate\_sysroot' or 'listtasks' may  
                  give a list of the tasks available.
- D, --debug      Increase the debug level. You can specify this more  
                  than once.

# Yocto Project: BitBake Common Tasks

- Listing Bitbake tasks for a recipe or image
- `$bitbake <package> -c listtasks`
- Useful tasks that you can run manually for most packages:

build	clean	cleansstate	compile
configure	install	listtasks	patch
populate_sysroot	rm_work		

- Task sequence run for generic bitbake `<package>` `bitbake <package>` :
- `fetch > unpack > patch > configure > compile > install > package > package_write`

# Yocto Project: BitBaking an Image

## Example

```
b35938@b35938-13:~/projects/fsl/yocto/build-imx6qsabreauto$ bitbake fsl-image-fb
Loading cache: 100%
|#####|
#####
#####| ETA: 00:00:00
Loaded 2005 entries from dependency cache.
```

Build Configuration: <removed for clarity>

NOTE: Preparing runqueue

NOTE: Executing SetScene Tasks

NOTE: Executing RunQueue Tasks

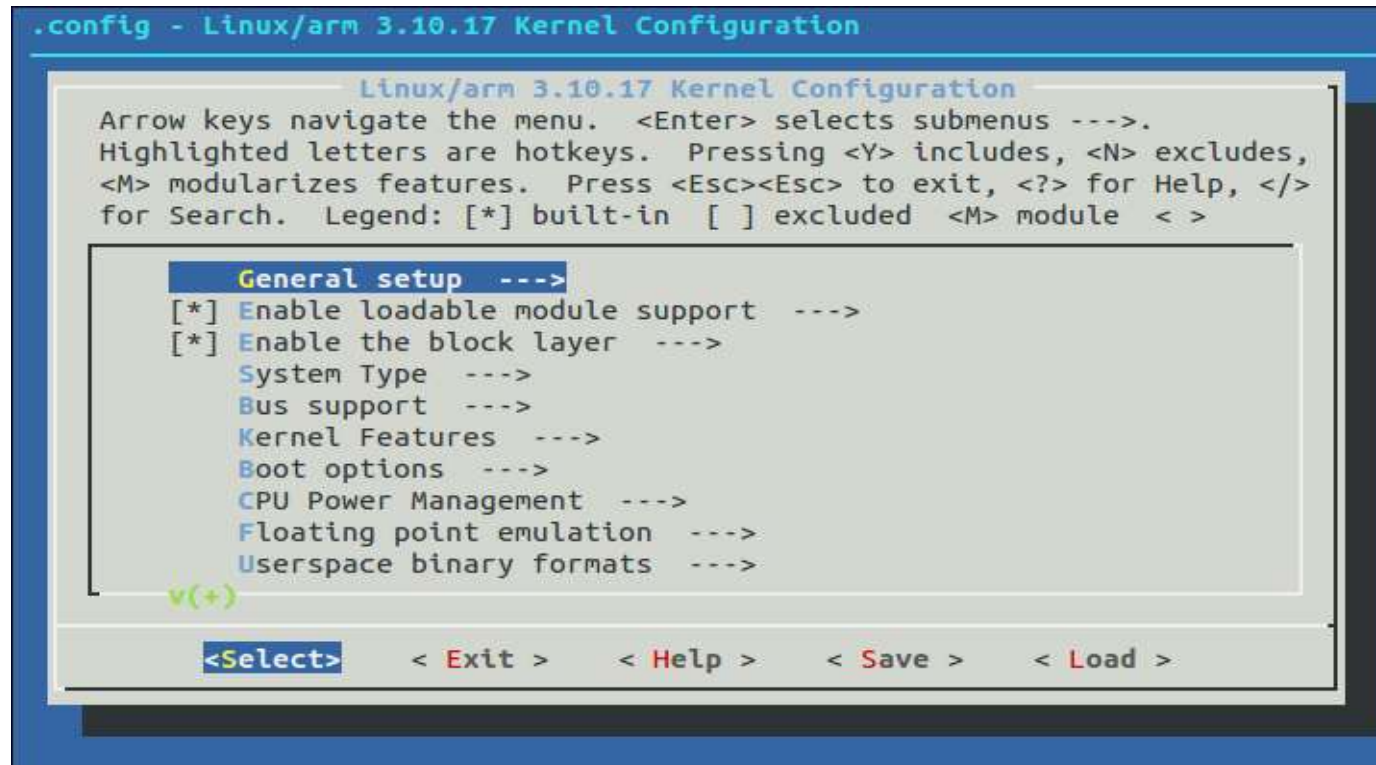
Currently 1 running tasks (3851 of 4000):



# Yocto Project : BitBake to Configure the Kernel

## Examples

- Bitbake linux-imx –c menuconfig



```
.config - Linux/arm 3.10.17 Kernel Configuration

Linux/arm 3.10.17 Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module < >

General setup --->
[*] Enable loadable module support --->
[*] Enable the block layer --->
System Type --->
Bus support --->
Kernel Features --->
Boot options --->
CPU Power Management --->
Floating point emulation --->
Userspace binary formats --->

v(+)

<Select> < Exit > < Help > < Save > < Load >
```

# Yocto Project : BitBake to Compile

## Examples

```
b35938@b35938-13:~/projects/fsl/yocto/build-imx6qsabreauto$ bitbake linux-imx -c compile
```

```
Loading cache: 100%
```

```
|#####  
#####  
#####| ETA: 00:00:00
```

```
Loaded 2005 entries from dependency cache.
```

```
NOTE: Resolving any missing task queue dependencies
```

```
Build Configuration: <removed for clarity>
```

```
NOTE: Preparing runqueue
```

```
NOTE: Executing SetScene Tasks
```

```
NOTE: Executing RunQueue Tasks
```

```
Currently 1 running tasks (249 of 249):
```

```
0: linux-imx-3.10.17-r0 do_compile (pid 17655)
```

# Yocto Project : BitBake to Start Fresh

## Examples

```
b35938@b35938-13:~/projects/fsl/yocto/build-imx6qsabreauto$ bitbake linux-imx -c cleansstate
```

```
Loading cache: 100%
```

```
|#####  
#####  
#####| ETA: 00:00:00
```

```
Loaded 2005 entries from dependency cache.
```

```
NOTE: Resolving any missing task queue dependencies
```

```
Build Configuration: <removed for clarity>
```

```
NOTE: Preparing runqueue
```

```
NOTE: Executing SetScene Tasks
```

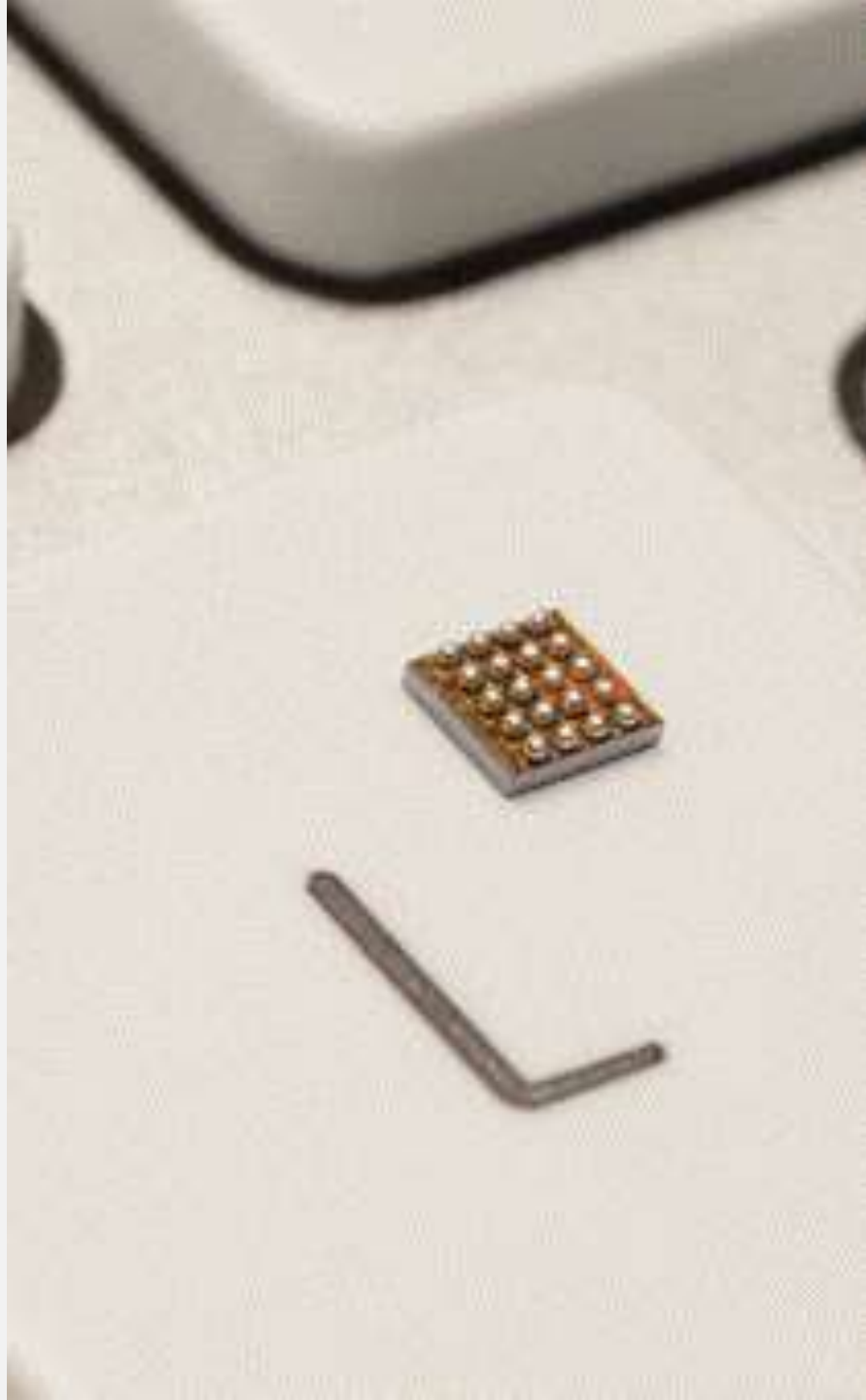
```
NOTE: Executing RunQueue Tasks
```

```
Currently 1 running tasks (249 of 249):
```

```
0: linux-imx-3.10.17-r0 do_cleansstate (pid 18465)
```

# Layers & BSP Layers





## Yocto Project: Layers

- Layers are a collection of metadata like recipes, classes, and machine configurations
- Layers allow you to partition your development for better reuse
- A Layer that contains machine support is referred to as a BSP Layer
- The SOURCE directory already contains examples of both types of Layers
- Layers are the best way to customize the build system to meet project needs

# Yocto Project : Layers in NXP Yocto Release

These layers are present after you setup the NXP Yocto Release on your host machine:

Layer Name	Layer Function
meta-fsl-arm	Provides support for NXP Reference Boards
meta-fsl-arm-extra	Provides support for 3 <sup>rd</sup> party development boards
meta-fsl-demos	QT Graphics and other user interface examples
meta-fsl-bsp-release	Provides updates to the meta-fsl-arm and meta-fsl-demos layers

# Yocto Project : Layer Management

- The BitBake layer management tool provides a view into the structure of recipes across a multi-layer project

\$ bitbake-layers <command> [arguments]

The following list describes the available commands:

- *help*: Displays general help or help on a specified command.
- *show-layers*: Shows the current configured layers.
- *show-recipes*: Lists available recipes and the layers that provide them.
- *show-overlayed*: Lists overlayed recipes
- *show-appends*: Lists .bbappend files and the recipe files to which they apply.
- *show-cross-depends*: Lists dependency relationships between recipes that cross layer boundaries.

# Yocto Project: BitBake Layer Management

- Example of bitbake-layers show-layers
- Higher priority numbers take precedence over lower numbers

```
b35938@b35938-13:~/projects/fsl/yocto/build-imx6qsabreauto$ bitbake-layers show-layers
WARNING: Host distribution "Ubuntu-14.04" has not been validated with this version of the build system; yo
ion.
layer                path                                                    priority
-----
meta                 /home/b35938/projects/fsl/yocto/sources/poky/meta      5
meta-yocto           /home/b35938/projects/fsl/yocto/sources/poky/meta-yocto 5
meta-oe              /home/b35938/projects/fsl/yocto/sources/meta-openembedded/meta-oe 6
meta-fsl-arm         /home/b35938/projects/fsl/yocto/sources/meta-fsl-arm    5
meta-fsl-arm-extra   /home/b35938/projects/fsl/yocto/sources/meta-fsl-arm-extra 4
meta-fsl-demos       /home/b35938/projects/fsl/yocto/sources/meta-fsl-demos  4
meta-fsl-arm         /home/b35938/projects/fsl/yocto/sources/meta-fsl-bsp-release/imx/meta-fsl-arm 0
meta-fsl-demos       /home/b35938/projects/fsl/yocto/sources/meta-fsl-bsp-release/imx/meta-fsl-demos 8
meta-browser         /home/b35938/projects/fsl/yocto/sources/meta-browser    7
meta-gnome           /home/b35938/projects/fsl/yocto/sources/meta-openembedded/meta-gnome 7
meta-networking      /home/b35938/projects/fsl/yocto/sources/meta-openembedded/meta-networking 5
b35938@b35938-13:~/projects/fsl/yocto/build-imx6qsabreauto$
```



# Yocto Project: Layer Creation

- The Yocto Project provides a script that can be used to create a new layer called yocto-layer
- Layers can also be created manually
- The script will walk you through setting up the new layer
  - The layer priority
  - Whether or not to create a sample recipe.
  - Whether or not to create a sample append file.

Use the yocto-layer create sub-command to create a new general layer. In its simplest form, you can create a layer as follows:

```
$ yocto-layer create mylayer
```

# Yocto Project: Layer Creation Example

- `b35938@b35938-13:~/projects/fsl/yocto/sources$ yocto-layer create dwf-layer`
- Please enter the layer priority you'd like to use for the layer: [default: 6]
- Would you like to have an example recipe created? (y/n) [default: n] y
- Please enter the name you'd like to use for your example recipe: [default: example]
- Would you like to have an example bbappend file created? (y/n) [default: n] y
- Please enter the name you'd like to use for your bbappend file: [default: example]
- Please enter the version number you'd like to use for your bbappend file (this should match the recipe you're appending to): [default: 0.1]
- New layer created in meta-dwf-layer.
- Don't forget to add it to your BBLAYERS (for details see meta-dwf-layer\README).

# Yocto Project : Layer Layout After Creation

Here is the tree of the new layer that we have created:

```
b35938@b35938-13:~/projects/fsl/yocto/sources$ tree meta-dwf-layer/
```

```
meta-dwf-layer/
```

```
|— conf
|   |— layer.conf
|— COPYING.MIT
|— README
|— recipes-example
|   |— example
|       |— example-0.1
|           |— example.patch
|           |— helloworld.c
|       |— example_0.1.bb
|— recipes-example-bbappend
|   |— example-bbappend
|       |— example-0.1
|           |— example.patch
|       |— example_0.1.bbappend
```

# Yocto Project : Layer Configuration File

Here is the configuration of the new layer that we have created:

```
b35938@b35938-13:~/projects/fsl/yocto/sources$ cat meta-dwf-layer/conf/layer.conf
```

```
# We have a conf and classes directory, add to BBPATH
```

```
BBPATH .= ":{LAYERDIR}"
```

```
# We have recipes-* directories, add to BBFILES
```

```
BBFILES += "${LAYERDIR}/recipes-*/*/*.bb \  
           ${LAYERDIR}/recipes-*/*/*.bbappend"
```

```
BBFILE_COLLECTIONS += "dwf-layer"
```

```
BBFILE_PATTERN_dwf-layer = "^${LAYERDIR}/"
```

```
BBFILE_PRIORITY_dwf-layer = "6"
```

# Yocto Project: Adding the Layer to bblayers.conf

Add the new layer to <build>/conf/bblayers.conf to let bitbake find the layer:

```
b35938@b35938-13:~/projects/fsl/yocto/build$ cat conf/bblayers.conf
```

```
LCONF_VERSION = "6"
```

```
BBPATH = "${TOPDIR}"
```

```
BSPDIR := "${@os.path.abspath(os.path.dirname(d.getVar('FILE', True)) + '/../..')}"
```

```
BBFILES ?= ""
```

```
BBLAYERS = " \
```

```
    ${BSPDIR}/sources/poky/meta \
```

```
    ${BSPDIR}/sources/poky/meta-yocto \
```

```
    \
```

```
    ${BSPDIR}/sources/meta-openembedded/meta-oe \
```

```
    \
```

```
    ${BSPDIR}/sources/meta-fsl-arm \
```

```
    ${BSPDIR}/sources/meta-fsl-arm-extra \
```

```
    ${BSPDIR}/sources/meta-fsl-demos \
```

```
    ${BSPDIR}/sources/meta-dwf-layer \
```

```
"
```

# Yocto Project: Adding the Layer to bblayers.conf

Run bitbake-layers show-layers:

Higher numbers are higher priority

```
b35938@b35938-13:~/projects/fsl/yocto/build$ bitbake-layers show-layers
WARNING: Host distribution "Ubuntu-14.04" has not been validated with this version of the build system.
layer                path                                                    priority
=====
meta                 /home/b35938/projects/fsl/yocto/sources/poky/meta      5
meta-yocto           /home/b35938/projects/fsl/yocto/sources/poky/meta-yocto 5
meta-oe              /home/b35938/projects/fsl/yocto/sources/meta-openembedded/meta-oe 6
meta-fsl-arm         /home/b35938/projects/fsl/yocto/sources/meta-fsl-arm   5
meta-fsl-arm-extra   /home/b35938/projects/fsl/yocto/sources/meta-fsl-arm-extra 4
meta-fsl-demos       /home/b35938/projects/fsl/yocto/sources/meta-fsl-demos 4
meta-dwf-layer       /home/b35938/projects/fsl/yocto/sources/meta-dwf-layer 6
b35938@b35938-13:~/projects/fsl/yocto/build$
```

# Yocto Project: Change a Layer into a BSP Layer

- For a layer to be considered a BSP layer you have to add a machine folder in the layer's conf directory
- An example of the meta-fsl-arm directory is below:

```
b35938@b35938-13:~/projects/fsl/yocto/sources/meta-fsl-arm/conf$ tree -L 1
.
├── layer.conf
└── machine

1 directory, 1 file
```

- More files and directories are recommended in the Yocto documentation to maintain consistency between layers

# Yocto Project: BSP Layer Directories

- A complete listing of the top directory of the meta-fsl-arm directory
- This directory contains a README file that explains how to use the Layer
- A stub README is placed when using bitbake-layer create <layer>

```
b35938@b35938-13:~/projects/fsl/yocto/sources/meta-fsl-arm$ tree -L 1
.
├── classes
├── conf
├── EULA
├── qt5-layer
├── README
├── recipes-bsp
├── recipes-core
├── recipes-graphics
├── recipes-kernel
├── recipes-multimedia
└── recipes-qt

9 directories, 2 files
```



# Yocto Project: Working with an Empty Layer

- Use this custom layer to create a custom BSP
- Examples of items that can be done in a layer
  - Create a MACHINE for custom hardware
  - Create an image specific to project requirements
  - Add a Linux kernel append recipe for custom hardware support
  - Add a u-boot append recipe for custom hardware support
  - Add a custom application recipe and include them in the image

# Yocto Project: Adding a Machine

Use a machine from another layer that is similar to your board. Many designs are derivatives of the reference designs

```
b35938@b35938-13:~/projects/fsl/yocto/sources/meta-dwf-layer/conf/machine$ tree
```

```
.
├── imx6q-custom-dwf.conf
└── include
    ├── fsl-default-providers.inc
    ├── fsl-default-settings.inc
    ├── fsl-default-versions.inc
    ├── imx6sabresd-common.inc
    └── imx-base.inc
```

1 directory, 6 files

## Yocto Project : Adding a new Image

- Create a custom image for the custom board by adding a directory called images in a recipes folder typically named after the layer

```
b35938@b35938-13:~/projects/fsl/yocto/sources/meta-dwf-layer/recipes-dwf$ tree
```

```
.
├── images
│   └── custom-dwf-image-fb.bb
```

1 directory, 1 file

# Yocto Project : Image Recipe Example

This image recipe can be borrowed from another layer and modified to meet the project needs

- DESCRIPTION = "DwF Custom Image Frame Buffer Image"
- IMAGE\_FEATURES += "splash"
- LICENSE = "MIT"
- inherit core-image
- inherit distro\_features\_check
- CONFLICT\_DISTRO\_FEATURES = "x11 wayland directfb"
- DISTRO\_FEATURES += "pulseaudio "
- WEB = "web-webkit"
- # Add extra image features
- EXTRA\_IMAGE\_FEATURES += "\
- nfs-server \
- tools-debug \
- tools-profile \
- ssh-server-dropbear \
- "
- SOC\_IMAGE\_INSTALL = ""
- SOC\_IMAGE\_INSTALL\_mx6 = "gpu-viv-bin-mx6q gpu-viv-g2d fsl-gpu-sdk"
- IMAGE\_INSTALL += "\
- \${SOC\_IMAGE\_INSTALL} \
- cpufrequtils \
- nano \
- packagegroup-fsl-tools-testapps \
- packagegroup-fsl-tools-benchmark \
- "
- export IMAGE\_BASENAME = "custom-dwf-image-fb"

# Yocto Project : Sourcing the custom MACHINE

Now that we have added our custom MACHINE and image recipe we can use them to create a new build directory for our custom platform:

```
b35938@b35938-13:~/projects/fsl/yocto$ MACHINE=imx6q-custom-dwf . fsl-setup-release.sh build-custom-dwf
Configuring for imx6q-custom-dwf
```

The Yocto Project has extensive documentation about OE including a reference manual which can be found at:

<http://yoctoproject.org/documentation>

For more information about OpenEmbedded see their website:

<http://www.openembedded.org/>

You can now run 'bitbake <target>'

Common targets are:

- core-image-minimal
- meta-toolchain
- meta-toolchain-sdk
- adt-installer
- meta-ide-support

Your build environment has been configured with:

- MACHINE=imx6q-custom-dwf
- SDKMACHINE=i686
- DISTRO=poky
- EULA=1

# Yocto Project : Building the Custom Image

We can now build the custom image with bitbake. Note that we need to add the moon-buggy recipe or a layer that contains the recipe before we can successful build the new image:

```
b35938@b35938-13:~/projects/fsl/yocto/build-custom-dwf$ bitbake custom-dwf-image-fb
```

```
Parsing recipes: 100% |#####| Time: 00:01:04
```

```
Parsing of 1608 .bb files complete (0 cached, 1608 parsed). 2007 targets, 159 skipped, 0 masked, 0 errors.
```

```
Build Configuration:
```

```
BB_VERSION      = "1.20.0"
```

```
BUILD_SYS       = "x86_64-linux"
```

```
NATIVELSBSTRING = "Ubuntu-14.04"
```

```
TARGET_SYS      = "arm-poky-linux-gnueabi"
```

```
MACHINE         = "imx6q-custom-dwf"
```

```
DISTRO          = "poky"
```

```
DISTRO_VERSION   = "1.5.3"
```

```
TUNE_FEATURES    = "armv7a vfp neon callconvention-hard cortexa9"
```

```
TARGET_FPU       = "vfp-neon"
```

```
meta
```

```
meta-dwf-layer
```

```
NOTE: Preparing runqueue
```

```
NOTE: Executing SetScene Tasks
```

```
NOTE: Executing RunQueue Tasks
```

# Kernel Development Workflow



# Yocto Project : Adding a Custom Kernel

- Add a custom Kernel to your layer by adding a bbappend recipe that appends the 3.10.17 recipe in the NXP release layer

```
$ cd ../sources/meta-dwf-custom
```

```
$ mkdir -p recipes-kernel/linux/linux-imx
```

- This will make the folder for our recipe file and for our patches that we use for our custom board

```
$ cd recipes-kernel/linux
```

```
$ touch linux-imx_3.10.17.bbappend
```

```
$ vim linux-imx_3.10.17.bbappend
```



# Yocto Project : Adding a Custom Kernel Recipe

File: linux-imx\_3.10.17.bbappend

# Linux Kernel bbappend example

# Bryan Thomas 2014 - NXP

# This points to where our extra files are like patches and kernel config files

FILESEXTRAPATHS\_prepend := "\${THISDIR}/\${PN}:"

# add our custom patches for our kernel build

SRC\_URI += "file://0001-patch-file.patch"

SRC\_URI += "file://0002-patch-file.patch"

# use our custom config for this kernel build

SRC\_URI += "file://defconfig"

SRC\_URI += "file://feature.conf"

# Yocto Project: Kernel Patching Workflow with GIT

- Multiple ways to work and create patches (GIT, quilt, manual diff)
- Can work inside the context of Yocto or generate tool chain to work in external directory

```
$ bitbake linux-imx -c patch
```

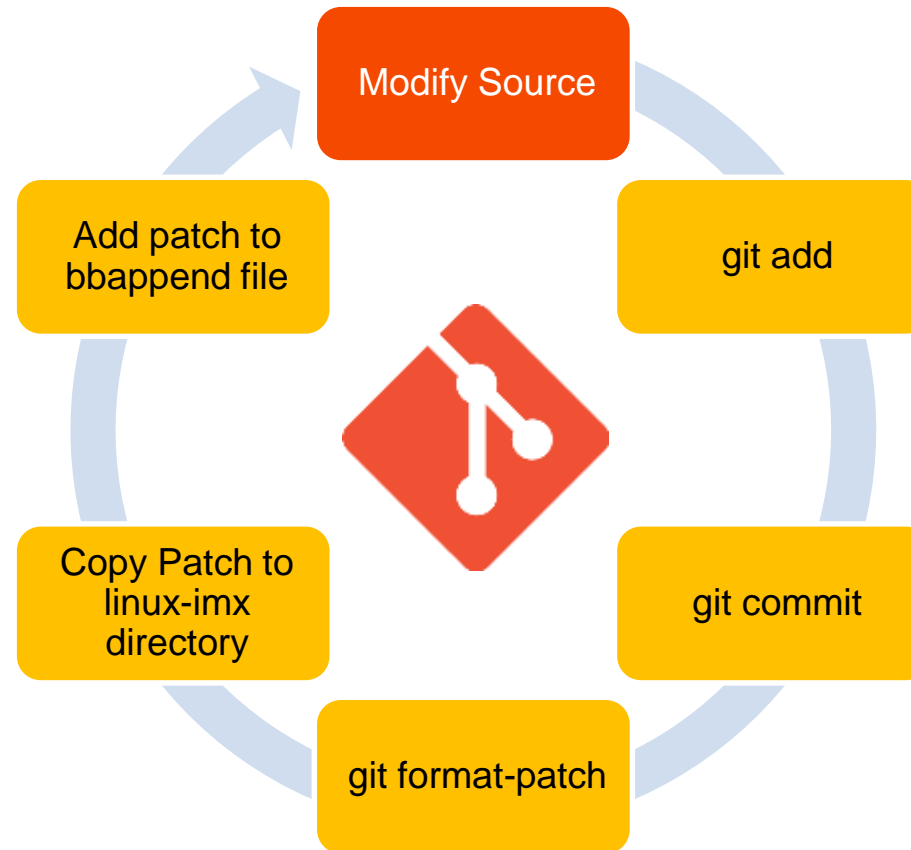
- This extracts the kernel sources to your work directory under your machine name

Example Path:

```
<build>/tmp/work/imx6-dwf-custom-poky-linux-gnueabi/linux-imx/3.10.17-r0/git/
```

- From here we can edit files and use GIT to generate patches using a iterative workflow

# Yocto Project: Kernel Patching Workflow with GIT



# Yocto Project: Adding the Patches to Recipe

File: linux-imx\_3.10.17.bbappend

```
# Linux Kernel bbappend example
```

```
# Bryan Thomas 2014 - NXP
```

```
# This points to where our extra files are like patches and kernel config files
```

```
FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"
```

```
# add our custom patches for our kernel build
```

```
SRC_URI += "file://0001-added-extra-version-information-to-makefile.patch"
```

```
# use our custom config for this kernel build
```

```
SRC_URI += "file://defconfig"
```

# Autotools Recipes



# Yocto Project: Adding an Autotools Recipe

- The GNU build system, also known as the Autotools, is a suite of programming tools designed to assist in making source code packages portable to many Unix-like systems.
- Many programs for Unix-like systems are configured and compiled with Autotools
- These programs commonly have a read me file that contains the following sequence as instructions to compile and install the program.

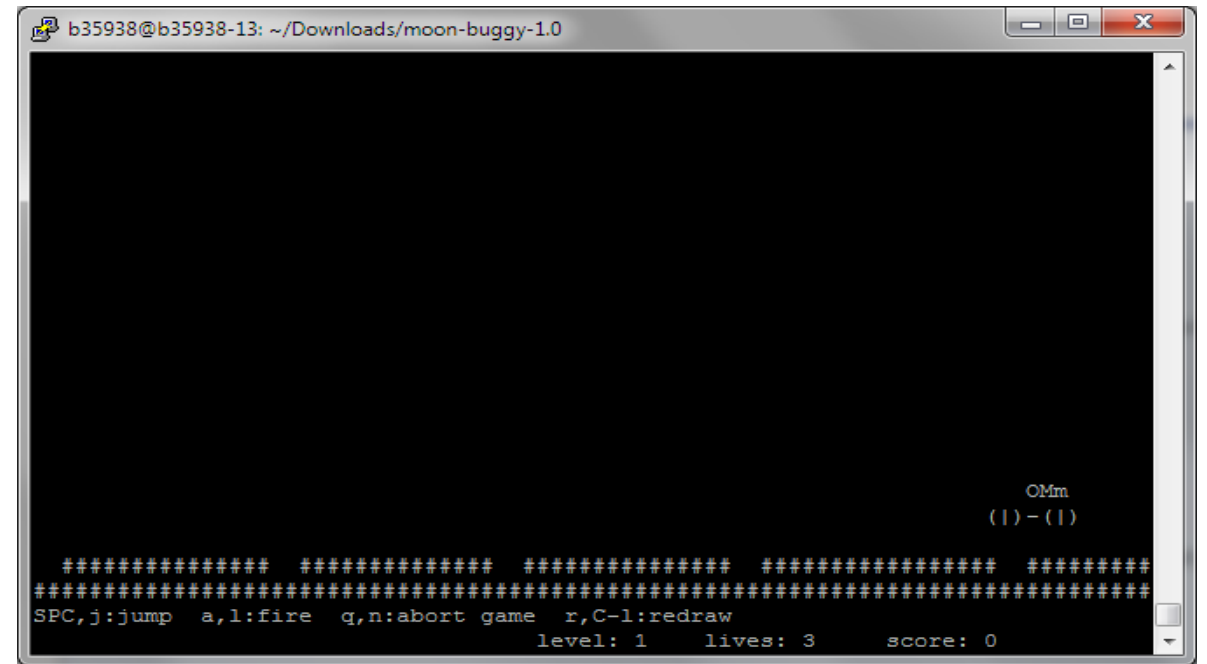
**\$ ./configure**

**\$ make**

**\$ make install**

# Yocto Project: Adding an Autotools Recipe

- Yocto can handle recipes that use the autotools suite
- Inherit autotools in the recipe
- For our example we will add the game moon buggy using an autotools style recipe



A screenshot of a terminal window titled "b35938@b35938-13: ~/Downloads/moon-buggy-1.0". The terminal displays the "moon-buggy" game. The game screen is mostly black with a white border. At the bottom, there is a status bar with the following text: "SPC,j:jump a,l:fire q,n:abort game r,C-l:redraw level: 1 lives: 3 score: 0". Above the status bar, there is a line of text: "OMm (1) - (1)".

# Yocto Project: Adding an Autotools Recipe

Add recipes-games folder to our meta-dwf-custom layer

```
b35938@b35938-13:~/projects/fsl/yocto/sources/meta-dwf-layer/recipes-games/moon-buggy$ tree
```

```
.
├── moon-buggy
│   ├── 0001-removed-creating-highscore-on-install.patch
│   └── 0002-remove-c-from-am-file-to-stop-from-running-at-instal.patch
└── moon-buggy_1.0.bb
```

\* We added another folder called moon-buggy for our patches needed to complete compilation on ARM.



# Yocto Project: Autotools Recipe

```
b35938@b35938-13:~/projects/fsl/yocto/sources/meta-dwf-layer/recipes-games/moon-buggy$ cat moon-  
buggy_1.0.bb
```

```
DESCRIPTION = "Moon Buggy command line game"
```

```
FILESEXTRAPATHS_prepend := "${THISDIR}/${PV}:"
```

```
LICENSE = "GPLv2"
```

```
LIC_FILES_CHKSUM = "file://COPYING;md5=94d55d512a9ba36caa9b7df079bae19f "
```

```
PR = "r0"
```

```
SRC_URI = "http://m.seehuhn.de/programs/moon-buggy-${PV}.tar.gz"
```

```
SRC_URI[md5sum] = "4da97ea40eca686f6f8b164d8b927e38"
```

```
SRC_URI[sha256sum] = "f8296f3fabd93aa0f83c247fbad7759effc49eba6ab5fdd7992f603d2d78e51a"
```

```
SRC_URI += "file://0001-removed-creating-highscore-on-install.patch"
```

```
SRC_URI += "file://0002-remove-c-from-am-file-to-stop-from-running-at-instal.patch"
```

```
inherit autotools gettext
```

```
b35938@b35938-13:~/projects/fsl/yocto/sources/meta-dwf-layer/recipes-games/moon-buggy$
```

# Yocto Project: Adding our Recipe Output

Once we create our recipe for moon-buggy we need to include into our custom image

```
DESCRIPTION = "DwF Custom Image Frame Buffer Image"
IMAGE_FEATURES += "splash"
LICENSE = "MIT"
inherit core-image
inherit distro_features_check
CONFLICT_DISTRO_FEATURES = "x11 wayland directfb"
DISTRO_FEATURES += "pulseaudio "
WEB = "web-webkit"
# Add extra image features
EXTRA_IMAGE_FEATURES += " \
    nfs-server \
    tools-debug \
    tools-profile \
    ssh-server-dropbear \
"
SOC_IMAGE_INSTALL = ""
SOC_IMAGE_INSTALL_mx6 = "gpu-viv-bin-mx6q gpu-viv-g2d fsl-gpu-sdk"
IMAGE_INSTALL += " \
    ${SOC_IMAGE_INSTALL} \
    cpufrequtils \
    nano \
    moon-buggy \
    packagegroup-fsl-tools-testapps \
    packagegroup-fsl-tools-benchmark \
"
export IMAGE_BASENAME = "custom-DwF-image-fb"
```

# Finding Help



## Yocto Project: Finding Help

- community.NXP.com
- meta-NXP mailing list
- Yocto Project Documentation
- Presenters contact: Bryan Thomas - [bryan.thomas@nxp.com](mailto:bryan.thomas@nxp.com)



SECURE CONNECTIONS  
FOR A SMARTER WORLD

[www.nxp.com](http://www.nxp.com)

NXP, the NXP logo, and NXP secure connections for a smarter world are trademarks of NXP B.V. All other product or service names are the property of their respective owners. © 2018 NXP B.V.