QorIQ Linux SDK 1.6 Working With Yocto

Yocto is an open source collaboration project that provides templates, tools and methods for building custom Linux-based systems for embedded products. SDK 1.6 is based on the Yocto 1.6 “daisy” release.

1. QorIQ SDK 1.6 Installation

The Yocto Project supports typical Linux distributions , and QorIQ SDK 1.6 is preferred to installed on one of the following test host distributions:
CentOS 6.5(32/64 bit)
Fedora Core 20(64bit)
OpenSUSE 13.1 (64 bit)
Mint 15(64bit)
Ubuntu 14.04(32/64bit)

The source ISO QorIQ-SDK-V1.6-SOURCE-20140619-yocto.iso contains recipes, source tar balls, and set up scripts, it supports full non-cache builds from source for any core. The Cache binary ISO QorIQ-SDK-V1.6-<core>-20140619-yocto.iso contains Cache binaries to accelerate the building to avoid build everything from the scratch, SDK 1.6 provides pre-build cache binaries per core type e500V2, e500MC, e5500, e5500-64b, e6500, e6500-64b.

It is preferred to install both source and Cache binary ISOs.

$ sudo mount -o loop QorIQ-SDK-V1.6-[SOURCE|<core>]-<date>-yocto.iso /mnt/cdrom

Run the install script with a non-root user.

$ /mnt/cdrom/install

After installing ISOs, it needed to run the script to prepare the build environment, some necessary packages are installed and internet access may be required, and you need to run this script to enable sudo root permission.

$ cd QorIQ-SDK-V1.6-20140619-yocto
$ ./scripts/host-prepare.sh

The SDK contains two QorIQ specific layers of software components.

meta-fsl-ppc : public QorIQ software components upstreamed to the community

meta-fsl-networking : private software components, not available upstream available for users through the SDK only.

2. Create a Build Environment

The script fsl-setup-poky in SDK install directory sets up a build environment for choosing a target machine.

$ source ./fsl-setup-poky -m p4080ds -j 4 -t 2 -l

The project directory build_<machine>_<release> will be created. To resume working with an existing project run the script.

$ source <project>/SOURCE_THIS

This also changes the shell's current working directory to build_<machine>_<release>.

Local Configuration File <project>/conf/local.conf

 # Machine Selection

MACHINE = "p4080ds" ***# Set by: -m <machine>***

 # Parallelism Options

BB_NUMBER_THREADS = "2"

PARALLEL_MAKE = "-j 4" ***# Set by: -t <threads> -j <jobs>***

 # delete sources after build

INHERIT += "rm_work" ***# Set by: -l(lite mode)***

3. Working with a Build Environment

A build environment has a ./tmp directory containing all the build output.

Deploy/images: Generated image files including Kernel, u-boot, rcw, rootfs, etc.

Deploy/rpm: Produced installable .rpm files of built packages.

Sysroots: Shared header files and libraries.

Work/<machine>-fsl-linux: Board specific target side packages such as rcw, Kernel , u-boot.

Work/<core>-fsl-linux : Non-board specific packages, compiled for the target architecture.

3.1 Image Generation

Images are generated by invoking "bitbake" for an image recipe, e.g.

$ bitbake fsl-image-core

SDK 1.6 contains the following image recipes.

fsl-image-minimal : Basic just packages to boot up a board.

fsl-image-core : fsl-image-minimal + FSL-specific packages.

fsl-image-flash: A small image programming into the flash on the target. It doesn't contain Freescale special SDK packages.

fsl-image-full : All packages + self-hosted toolchain.

fsl-image-virt : fsl-image-minimal + KVM + QEMU + libvirt

fsl-toolchain: the cross compiler binary package

An image recipe can specify multiple image types to be generated simultaneously.

Edit IMAGE_FSTYPES variable in the recipe meta-fsl-networking/images/fsl-image-minimal.bb.

IMAGE_FSTYPES ?= "tar.gz ext2.gz.u-boot jffs2"

The *.rootfs.tar.gz image contains an archive of the filesystem, and suitable for deploy external media like hard drive or NFS mounted rootfs.

3.2 Machine Configuration File

Machine configuration files are located in meta-fsl-ppc/conf/, <core>.inc files define shared hardware tuning definitions, <machine>.conf file specifies BSP information.

For example for P4080DS.

meta-fsl-ppc/conf/machine/p4080ds.conf

require e500mc.inc

UBOOT_MACHINES = "P4080DS P4080DS_SECURE_BOOT P4080DS_SDCARD P4080DS_SPIFLASH"

KERNEL_DEVICETREE = "p4080ds.dtb p4080ds-usdpaa.dtb"

KERNEL_DEFCONFIG = "${S}/arch/powerpc/configs/corenet32_smp_defconfig"

JFFS2_ERASEBLOCK = "0x10000"

UBOOT_MACHINES : enumeration of u-boot configs to build, the available u-boot configs can be got from board.cfg file of u-boot source tree.

JFFS2_ERASEBLOCK : the flash JFFS2 erase block size

KERNEL_DEFCONFIG: the default Kernel defconfig , common defconfig each for corenet32 and corenet64 machines.

KERNEL_DEVICETREE : Kernel device tree files.

4. Working with BitBake

Bitbake command must be executed in the <project> directory, it parses recipes, determines task queue dependencies, performs the steps to obtain the desired result.

4.1 Running Specific BitBake Tasks

The command bitbake is invoked to run a specific task specified in the recipe, optionally with - c <CMD> indicate a specific task.

$ bitbake [-c <CMD>] [options] <recipe>

The following is bitbake tasks for most of recipes.

build, clean, cleansstate, compile, compile, configure, install, patch, rm_work

clean: remove the work folder of the package

cleansstate: clean + delete the cached binary

patch: install source including all patches

menuconfig : run kernel menuconfig

The most common sequence of bitbake task is as the following.

fetch->unpack->patch->configure->compile->install->package->package_write

Bitbake executing logs are written to <project>/tmp/work/<folder>/<pkg>/<ver>/temp.

4.2 Configure and Rebuild Linux kernel

The following procedure is about how to exact Kernel source, reconfigure Linux Kernel and rebuild Linux Kernel.

1. Clean the current cache and build temporary folder.

   $ bitbake virtual/kernel -c cleansstate

2. Extract the Kernel source and apply related Kernel patches.
   $ bitbake virtual/kernel -c patch

   Go to Kernel source folder and users could “git format-patch” or “git log” to get Kernel

   patch list.

   $ cd tmp/work/p4080ds-fsl-linux/linux-qoriq-sdk/3.12-r0/git/

3. Change the Kernel defconfig by updating KERNEL_DEFCONFIG variable in
   meta-fsl-ppc/conf/machine/<machine>.conf
   Change dts by updating KERNEL_DEVICETREE variable in
   meta-fsl-ppc/conf/machine/<machine>.conf

4. Go to Kernel source folder in step 2 and do menuconfig.
   $ make ARCH=powerpc menuconfig

5. Rebuild Kernel image and deploy Kernel image.
   $ cd build_<machine>_release
   $ bitbake virtual/kernel -c compile -f
   $ bitbake virtual/kernel

4.3 Configure and Rebuild the u-boot

1. Clean u-boot Cache and get u-boot source code.

   $ bitbake -c cleansstate u-boot

   $ bitbake -c patch u-boot

2. Go to u-boot source folder and modify u-boot source code.

Use the command “bitbake -e u-boot | grep ^S=” to get u-boot source folder and got to the source folder to modify the source code.

3. Modify u-boot configuration

Modify the variable UBOOT_MACHINES in meta-fsl-ppc/conf/machine/<machine>.conf.

4. Rebuild U-Boot image
$ cd build_<machine>_release
$ bitbake -c compile -f u-boot
$ bitbake u-boot

5. Customize Root File System

The following procedure introduces how to define a new custom layer, modify from an existing image recipe and modify RFS content after package installation.

1. Make a new layer directory <sdk-install-dir>/meta-custom, and create a new layer.conf file copying from meta-fsl-ppc/conf/layer.conf and modify as the following.
[meta-custom/conf/layer.conf]
 # We have a packages directory, add to BBFILES
BBPATH := "${BBPATH}:${LAYERDIR}"
BBFILES += "${LAYERDIR}/recipes-/*/*.bb*"*
BBFILES += "${LAYERDIR}/images/.bb*"*
*BBFILE_COLLECTIONS += "**custom**"*
*BBFILE_PATTERN_**custom** := "^${LAYERDIR}/"*
*BBFILE_PRIORITY_**custom** = "**6**"*

2. *Edit the conf/bblayers.conf file in the build project*
[conf/bblayers.conf]
*BBLAYERS = " *
*/opt/yt_sdks/QorIQ-SDK-V1.6-20140619-yocto/meta *
*/opt/yt_sdks/QorIQ-SDK-V1.6-20140619-yocto/meta-yocto *
…
***/opt/yt_sdks/QorIQ-SDK-V1.6-20140619-yocto/meta-custom** *
"

3. Users could use either of the following methods to modify from the existing image recipe.
   a. $ cp meta-fsl-networking/images/fsl-image-core.bb \
      meta-custom/images/custom-image-core.bb
      Add/remove packages from the IMAGE_INSTALL list in custom-image-core.bb.
      Rebuild images.
      *IMAGE_INSTALL += " *
      *bridge-utils *
      *coreutils *
      [...]
      *perf *
      *psmisc *
      *tcpdump *
      "
   b. Edit the image recipe to require the settings from a pre-existing image recipe.
      require images/fsl-image-core.bb

IMAGE_INSTALL_append = " bridge-utils"

IMAGE_INSTALL_remove = " net-tools"

4. Add ROOTFS_POSTPROCESS_COMMAND variable to image recipe to modify RFS content after package installation and before image generation. (Optional)
   *ROOTFS_POSTPROCESS_COMMAND += " rm -rf ${IMAGE_ROOTFS}/boot ; *
   *rm -rf ${IMAGE_ROOTFS}/usr/include ; *
   *rm -rf ${IMAGE_ROOTFS}/usr/share/info ; *
   *(find ${IMAGE_ROOTFS} -type d -name "man" | xargs rm -rf) ; *
   *(find ${IMAGE_ROOTFS} -type d -name "src" | xargs rm -rf) ; *
   *(find ${IMAGE_ROOTFS} -type d -name "doc" | xargs rm -rf) ; *
   *(find ${IMAGE_ROOTFS} -name "*python*" | xargs rm -rf) ; *
   (find ${IMAGE_ROOTFS} -name "elf_*86" | xargs rm -rf) ; *
   (find ${IMAGE_ROOTFS} -name "elf_*64" | xargs rm -rf) ; *
   *(find ${IMAGE_ROOTFS} -name "*openbios*" | xargs rm -rf) ; *
   (find ${IMAGE_ROOTFS} -name "powerpc-fsl-" | xargs rm -rf) ; *
   "
5. *Add extra space in image recipe.(Optional)*
   IMAGE_ROOTFS_EXTRA_SPACE = "<size_in_KB>"

6. *Build the customized image with the following command.*
   $ bitbake custom-image-core

6. Merge Files to Root File System

Users could use merge-files recipe to add their own files into root file system.

1. Copy desired files and directories in meta-fsl-networking/recipes-tools/merge-files/files/merge directory.
   For example KVM deployment:
   $ mkdir -p ../meta-fsl-networking/recipes-tools/merge-files/files/merge/home/root
   $ cp tmp/deploy/images/fsl-image-minimal-p4080ds.rootfs.ext2.gz ../meta-fsl-networking/recipes-tools/merge-files/files/merge/home/root/guest.rootfs.ext2.gz
2. After populating the merge directory with the desired files, rebuild the rootfs.
   $ bitbake -c install -f merge-files
   $ bitbake merge-files
   $ bitbake fsl-image-core

7. Create a New Package in Yocto build envrionment

Make a directory (e.g. meta-custom/recipes-custom) to group recipes for the new package, then make a sub-directory for each recipe <pkg>_<version>.bb, the structure is as the following.

```
meta-custom
├── conf
│   └── layer.conf
└── recipes-custom
    └── simple
        ├── files
        │   └── simple.c
        └── simple_1.0.bb
```

Create recipe for a package with local source Files

[meta-custom/recipes-custom/simple/simple_1.0.bb]
DESCRIPTION = "Simple application"
SECTION = "examples"
LICENSE = "MIT"
LIC_FILES_CHKSUM =
"file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"
PR = "r0"
SRC_URI = "file://simple.c"
S = "${WORKDIR}/simple"
do_compile() {
${CC} ${WORKDIR}/simple.c -o ${S}/simple
}
do_install() {
install -d ${D}${bindir}
install -m 0755 ${S}/simple ${D}${bindir}
}

For Make file based packages, please refer to mtd-utils as an example.

[meta-custom/recipes-custom/mtd-utils_1.5.0/mtd-utils_1.5.0.bb]
DESCRIPTION = "Tools for managing memory technology devices."
SECTION = "base"
HOMEPAGE = "http://www.linux-mtd.infradead.org/"
LICENSE = "GPLv2"
*LIC_FILES_CHKSUM = "file://COPYING;md5=0636e73ff0215e8d672dc4c32c317bb3 *
file://include/common.h;beginline=1;endline=17;md5=ba05b07912a44ea2bf81ce409380049c"
DEPENDS = "zlib lzo e2fsprogs util-linux"
SRC_URI = "git://git.infradead.org/mtd-utils.git;protocol=git;tag=v${PV}"
S = "${WORKDIR}/git/"
***EXTRA_OEMAKE = "'CC=${CC}' 'CFLAGS=${CFLAGS} -I${S}/include ***
-DWITHOUT_XATTR' 'BUILDDIR=${S}'"
do_install () {
*oe_runmake install DESTDIR=${D} SBINDIR=${sbindir} MANDIR=${mandir} *
INCLUDEDIR=${includedir}
install -d ${D}${includedir}/mtd/
for f in ${S}/include/mtd/.h; do*
install -m 0644 $f ${D}${includedir}/mtd/
done
}

The Makefile based package recipe lists source archive in the SRC_URI variable, store additional make options in the EXTRA_OEMAKE variable, provide manually written do_install task
The example pulls mtd-utils v1.5.0 from upstream git and builds the package using its Makefile
DEPENDS : build time dependencies between .bb files
SRC_URI : list of source files - local or remote