# SMARC-iMX8M-BSP-Rocko

**On this page:**



## Building NXP/Embedian’s Yocto Rocko BSP Distribution


**Eric Lee**


version 1.0a, 4/10/2019

## Introduction

This document describes how Embedian builds a customized version of NXP’s i.MX8M official Yocto Rocko BSP release for Embedian's *SMARC-iMX8M* product platform. The approach is to pull from Embedian's public facing GIT repository and build that using bitbake. The reason why we use this approach is that it allows co-development. The build output is comprised of binary images, feed packages, and an SDK for *SMARC-iMX8M* specific development.

Freescale makes their i.MX series official bsp build scripts available via the following GIT repository:

```
git://git.freescale.com/imx/meta-fsl-bsp-release
```

Freescale community BSP release build script is available via the following repository:

```
git://git.freescale.com/imx/fsl-arm-yocto-bsp.git
```

It is this repository that actually pulls in the fsl-bsp-release project to perform the Linux BSP builds for Freescale's i.MX8M ARM Cortext-A53 chips.

# Generating SSH Keys

We recommend you use SSH keys to establish a secure connection between your computer and Embedian Gitlab server. The steps below will walk you through generating an SSH key and then adding the public key to our Gitlab account.

## Step 1. Check for SSH keys

First, we need to check for existing ssh keys on your computer. Open up Git Bash and run:

```
$ cd ~/.ssh
$ ls
# Lists the files in your .ssh directory
```

Check the directory listing to see if you have a file named either `id_rsa.pub` or `id_dsa.pub`. If you don't have either of those files go to **step 2** . Otherwise, you already have an existing keypair, and you can skip to **step 3**.

## Step 2. Generate a new SSH key

To generate a new SSH key, enter the code below. We want the default settings so when asked to enter a file in which to save the key, just press enter.

```
$ ssh-keygen -t rsa -C "your_email@example.com"
# Creates a new ssh key, using the provided email as a label
# Generating public/private rsa key pair.
# Enter file in which to save the key (/c/Users/you/.ssh/id_rsa): [Press enter]
$ ssh-add id_rsa
```

Now you need to enter a passphrase.

```
Enter passphrase (empty for no passphrase): [Type a passphrase]
Enter same passphrase again: [Type passphrase again]
```

Which should give you something like this:

```
Your identification has been saved in /c/Users/you/.ssh/id_rsa.
Your public key has been saved in /c/Users/you/.ssh/id_rsa.pub.
The key fingerprint is:
01:0f:f4:3b:ca:85:d6:17:a1:7d:f0:68:9d:f0:a2:db your_email@example.com
```

## Step 3. Add your SSH key to Embedian Gitlab Server

Copy the key to your clipboard.

```
$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAABDAQABAAABAQDQUEnh8uGpfxaZVU6+uE4bsDrs/tEE5/BPW7jMAxak
6qgOh6nUrQGBWS+VxMM2un3KzwvLRJSj8G4TnTK2CSmlBvR+X8ZeXNTyAdaDxULs/StVhH+QRtFEGy4o
iMIzvIlTyORY89jzhIsgZzwr01nqoSeWWASd+59JWtFjVy0nwVNVtbek7NfuIGGAPaijO5Wnshr2uChB
Pk8ScGjQ3z4VqNXP6CWhCXTqIk7EQl7yX2GKd6FgEFrzae+5Jf63Xm8g6abbE3ytCrMT/jYy5OOj2XSg
6jlxSFnKcONAcfMTWkTXeG/OgeGeG5kZdtqryRtOlGmOeuQe1dd3I+Zz3JyT your_email@example.c
om
```

Go to Embedian Git Server. At Profile Setting --> SSH Keys --> Add SSH Key

Paste your public key and press "Add Key" and your are done.

## Overview of the *meta-smarcimx8m-rocko* Yocto Layer

The supplied meta-smarcimx8m-rocto Yocto compliant layer has the following organization:

```
.
|-- conf
|   |-- layer.conf
|   |-- site.conf
|   |-- machine
|       |-- smarcimx8m2g.conf
|       |-- smarcimx8m2gind.conf
|   |   `-- smarcimx8m4g.conf
|-- README
|-- recipes-bsp
|   |-- u-boot
|       `-- u-boot-smarcimx8m_2017.03.bb
|   |-- imx-atf
|       |   |-- imx-atf
|       |   |   |-- 0001-ATF-support-to-different-LPDDR4-configurations.patch
|       |   |   `-- atf-0001-1-add-noc-tuning-smc-case-lower-cpu-vpu-memory-acces.patch
|       `-- imx-atf_1.4.1.bbappend
|   |-- pm-utils
|       `-- pm-utils_%.bbappend
|   |-- imx-mkimage
|       `-- imx-boot_0.2.bbappend
|-- recipes-connectivity
|   |-- connman
|       `–connman_%.bbappend
|       |   |-- connman
|       |   |   |-- connman
|       |   |   |-- connmand-env
|       |   |   `-- connman-env.service
|-- recipes-core
|   |-- busybox
|       `-- busybox_%.bbappend
|       |   |-- busybox
|       |   |   |-- ftpget.cfg
|       |   |   `-- defconfig
|   |-- packagegroups
|       `-- packagegroup-core-tools-testapps.bbappend
|-- recipes-multimedia
|   |-- gst-plugins-good
|   |   |-- files
|           `-- increase_min_buffers.patch
|   |-- pulseaudio
|   |   |-- pulseaudio
|       |   |   |-- default.pa
|       |   |   |-- init
|       |   |   |-- pulseaudio-bluetooth.conf
|       |   |   |-- pulseaudio.service
|       |   |   `-- system.pa
|       `-- pulseaudio_%.bbappend
```

```
`-- recipes-kernel
|   |-- linux
|   |   |-- linux-imx-src.inc
|       `-- linux-smarcimx8m_4.9.51.bb
```

#### Notes on ***meta-smarcimx8m-rocko*** layer content

`conf/machine/*`

This folder contains the machine definitions for the *smarcimx8m2g/smarcimx8m2gind/smarcimx8m4g* platform and backup repository in Embedian. These select the associated kernel, kernel config, u-boot, u-boot config, and tar.bz2 image settings.

`recipes-bsp/u-boot/*`

This folder contains recipes used to build DAS U-boot for *smarcimx8m2g/smarcimx8m2gind/smarcimx8m4g* platform.

`recipes-bsp/imx-atf/*`

This folder contains ARM Trusted firmware for *smarcimx8m2g/smarcimx8m2gind/smarcimx8m4g* platform.

`recipes-bsp/imx-mkimage/*`

This folder contains imx-mkimage tool for *smarcimx8m2g/smarcimx8m2gind/smarcimx8m4g* platform.

`recipes-core/busybox/*`

This folder remove telnetd from bysybox for *smarcimx8m2g/smarcimx8m2gind/smarcimx8m4g* platform.

`recipes-kernel/linux/*`

Contains the recipes needed to build the *smarcimx8m2g/smarcimx8m2gind/smarcimx8m4g* Linux kernels.

`recipes-security/optee-imx/*`

This folder contains Trusted Execution Environment for *smarcimx8m2g/smarcimx8m2gind/smarcimx8m4g* platform.

# Setting Up the Tools and Build Environment

To build the latest Freescale i.MX8M fsl-bsp-release, you first need an Ubuntu 16.04 LTS installation. Since bitbake does not accept building images using root privileges, please **do not** login as a root user when performing the instructions in this section.

Once you have Ubuntu 16.04 LTS running, install the additional required support packages using the following console command:

```
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo build-essential chrpath libsdl1.2-dev
xterm python-m2crypto bc libsdl1.2-dev
```

If you are using a 64-bit Linux, then you'd also need to install 32-bit support libraries, needed by the pre-built Linaro toolchain and other binary tools.

```
$ sudo dpkg --add-architecture i386
$ sudo apt-get update
$ sudo apt-get install curl g++-multilib gcc-multilib lib32z1-dev libcrypto++9:i386 libcrypto++-dev:i386
liblzo2-dev:i386 libusb-1.0-0:i386 libusb-1.0-0-dev:i386 uuid-dev:i386
```



If you saw error like the following after running "sudo dpkg --add-architecture i386"

```
pkg: error: unknown option --add-architecture
```

make sure the only file present in /etc/dpkg/dpkg.cfg.d/ is "multiarch"

```
ls /etc/dpkg/dpkg.cfg.d/
```

if output is

```
multiarch
```

execute the following commands as it is else replace "multiarch" with the name of file present in that directory.

```
$ sudo sh -c "echo 'foreign-architecture i386' > /etc/dpkg/dpkg.cfg.d/multiarch"
```

The above command will add i386 architecture.

To get the BSP you need to have 'repo' installed and use it as:

Install the 'repo' utility:

```
$ mkdir ~/bin
$ curl http://commondatastorage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
$ chmod a+x ~/bin/repo
$ PATH=${PATH}:~/bin
```

Download the BSP Yocto Project Environment.

```
$ mkdir ~/smarc-imx8m-rocko-release

$ cd ~/smarc-imx8m-rocko-release

$ repo init -u https://source.codeaurora.org/external/imx/imx-manifest -b imx-linux-rocko -m
imx-4.9.88-2.0.0_ga.xml

$ repo sync
```

Download the Embedian Yocto build script and meta layer.

```
$ wget
ftp://ftp.embedian.com/public/dev/minfs/smarc-imx8m-bsp-release/fsl-smarcimx8m-rocko-setup-release.sh

$ chmod 444 fsl-smarcimx8m-rocko-setup-release.sh

$ cd sources

$ git clone git@git.embedian.com:developer/meta-smarcimx8m-rocko.git -b smarcimx8m-rocko

$ cd ~/smarc-imx8m-rocko-release

$ DISTRO=fsl-imx-wayland MACHINE=smarcimx8m2g source fsl-smarcimx8m-rocko-setup-release.sh -b
imx8m-build-qt5wayland
```

Choose "y" to accept EULA.

This script will create and bring you to ~/smarc-imx8m-rocko-release/imx8m-build-qt5wayland directory.

**Note**
The last line of the above script

```
$ DISTRO=<distro name> MACHINE=<machine name> source fsl-smarcimx8m-rocko-setup-release.sh -b <build
dir>
```

1. <distro name>
   - fsl-imx-x11 - Only X11 graphics
   - fsl-imx-wayland - Wayland weston graphics
   - fsl-imx-xwayland - Wayland graphics and X11. X11 applications using EGL are not supported
   - fsl-imx-fb - Frame Buffer graphics - no X11 or Wayland (Frame Buffer DISTRO is not supported on i.MX8M.)
2. <machine name>
   - *smarcimx8m2g* - if your board is SMARC-iMX8M-D-2G, SMARC-iMX8M-L-2G or SMARC-iMX8M-Q-2G.
   - *smarcimx8m2gind* - if your board is SMARC-iMX8M-D-2G-I, SMARC-iMX8M-L-2G-I or SMARC-iMX8M-Q-2G.

- *smarcimx8m4g* - if your board is SMARC-iMX8M-Q-4G or SMARC-iMX8M-Q-4G-I.

The default console debug port is *SER3.*

In this document, we will use *smarcimx8m2g* as the example of machine name. Users need to change different machine name if you have different SMARC card variants.

### Building the target platforms

To build Embedian/Freescale Yocto BSP, use the following commands:

```
$ MACHINE=smarcimx8m2g bitbake -k fsl-image-qt5-validation-imx
or
$ MACHINE=smarcimx8m2g bitbake -k fsl-image-validation-imx
```

**Note**

fsl-image-validation-imx provides a gui image without QT5.

fsl-image-qt5-validation-imx provides a Qt5 image for X11, wayland or FB backends depending on your distro name.

If your machine name is smarcimx8m2g and your gui image is without QT5 , the following command gives you as an example.

```
$ MACHINE=smarcimx8m2g bitbake -k fsl-image-validation-imx
```

The first build takes time.

Once it done, you can find all required images under ~/smarc-imx8m-rocko-release/<build directory>/tmp/deploy/images/<machine name>/

You may want to build programs that aren't installed into a root file system so you can make them available via a feed site (described below.) To do this you can build the package directly and then build the package named package-index to add the new package to the feed site.

The following example builds the minicom program and makes it available on the feed site:

```
$ MACHINE=smarcimx8m2g bitbake tcpdump
$ MACHINE=smarcimx8m2g bitbake package-index
```

Once the build(s) are completed you'll find the resulting images, rpm and licenses in folder *~/smarc-imx8m-rocto-release/<build directory>/tmp/deploy*.

deploy/images/<machine name>/*

This folder contains the binary images for the root file system and the Embedian *SMARC-iMX8M* specific version of the boot file, Image and device tree file. Specifically the images are:

deploy/images/<machine name>/imx-boot-<machine name>-sd.bin-flash_evk

This boot file binary for *SMARC-iMX8M* if your device has HDMI

deploy/images/<machine name>/imx-boot-<machine name>-sd.bin-flash_evk_no_hdmi

This boot file binary for *SMARC-iMX8M* if your device has no HDMI

deploy/images/<machine name>/Image

The kernel Image for *SMARC-iMX8M*.

deploy/images/<machine name>/<device tee file>

***DCSS vs LCDIF***

i.MX8M comes with 2 display controllers: DCSS and LCDIF.

DCSS can be connected to either HDMI or MIPI-DSI (to LVDS bridge) and supports resolutions up to 4K.

LCDIF can be connected only to MIPI-DSI and supports resolutions up to 1080p.

Selecting display configuration is a matter of selecting an appropriate DTB file under deploy/images/<machine name>/<device tee file>

All available DTB files are listed in the table below.

| DTB File Name | Description |
|---|---|
| *fsl-smarcimx8mq.dtb* | Device tree blob for no display configuration. |
| *fsl-smarcimx8mq-hdmi.dtb* | Device tree blob for HDMI display configuration (DCSS). |
| *fsl-smarcimx8mq-hdmi-4k.dtb* | Device tree blob for HDMI 4k display configuration (DCSS). |
| *fsl-smarcimx8mq-lcdif-lvds.dtb* | Device tree blob for LCDIF LVDS display configuration. |
| *fsl-smarcimx8mq-dcss-lvds.dtb* | Device tree blob for DCSS LVDS display configuration. |
| *fsl-smarcimx8mq-dual-display.dtb* | Device tree blob for dual LVDS+HDMI display configuration. |

`deploy/images/<machine name>/fsl-image-validation-imx-<machine name>.*`

Embedian root file system images for software development on Embedian's *SMARC-iMX8M* platforms without QT5.

`deploy/images/<machine name>/fsl-image-qt5-validation-imx-<machine name>.*`

Embedian root file system images for software development on Embedian's *SMARC-iMX8M* with QT5.

`deploy/rpm/*`

This folder contains all the packages used to construct the root file system images. They are in **rpm** format (similar format to Fedora packages) and can be dynamically installed on the target platform via a properly constructed *feed* file. Here is an example of the feed file (named *imx8m_qt5wayland_update.repo*) that is used internally at Embedian to install upgrades onto a ***imx8msmarc*** QT5 platform directly on framebuffer without reflashing the file system:

```
[noarch]
type = rpm-md
baseurl = http://www.embedian.com/smarcimx8m-rocko-feed/imx8msmarc/wayland/qt5/noarch
[aarch64]
type = rpm-md
baseurl = http://www.embedian.com/smarcimx8m-rocko-feed/imx8msmarc/wayland/qt5/aarch64
[aarch64_mx8mq]
type = rpm-md
baseurl = http://www.embedian.com/smarcimx8mq-rocko-feed/imx8msmarc/wayland/qt5/aarch64_mx8mq
[smarcimx8m2g]
type = rpm-md
baseurl = http://www.embedian.com/smarcimx8m-rocko-feed/imx8msmarc/wayland/qt5/smarcimx8m2g
```

`deploy/licenses/*`

A database of all licenses used in all packages built for the system.

# Setup SD Card

For these instruction, we are assuming: DISK=/dev/mmcblk0, "lsblk" is very useful for determining the device id.

```
$ export DISK=/dev/mmcblk0
```

Erase SD card:

```
$ sudo dd if=/dev/zero of=${DISK} bs=1M count=16
```

Create Partition Layout: Leave 2MB offset for boot file.

**With util-linux v2.26, sfdisk was rewritten and is now based on libfdisk**.

**sfdisk**
```
$ sudo sfdisk --version
sfdisk from util-linux 2.27.1
```

Create Partitions:

**sfdisk >=2.26.x**
```
$ sudo sfdisk ${DISK} <<-__EOF__
2M,48M,0x83,*
50M,,,
__EOF__
```

**sfdisk <=2.25**
```
$ sudo sfdisk --in-order --Linux --unit M ${DISK} <<-__EOF__
2,48,0x83,*
,,,-
__EOF__
```

Format Partitions:

```
for: DISK=/dev/mmcblk0
$ sudo mkfs.vfat -F 16 ${DISK}p1 -n boot
$ sudo mkfs.ext4 ${DISK}p2 -L rootfs

for: DISK=/dev/sdX
$ sudo mkfs.vfat -F 16 ${DISK}1 -n boot
$ sudo mkfs.ext4 ${DISK}2 -L rootfs
```

Mount Partitions:

**On some systems, these partitions may be auto-mounted...**

```
$ sudo mkdir -p /media/boot/
$ sudo mkdir -p /media/rootfs/

for: DISK=/dev/mmcblk0
$ sudo mount ${DISK}p1 /media/boot/
$ sudo mount ${DISK}p2 /media/rootfs/

for: DISK=/dev/sdX
$ sudo mount ${DISK}1 /media/boot/
$ sudo mount ${DISK}2 /media/rootfs/
```

## Install Boot File (`imx-boot-<machine name>-sd.bin-flash_evk` or imx-boot-<machine name>-sd.bin-flash_evk_no_hdmi)

Boot file is factory default flashed at on-module eMMC flash.

**If on-module eMMC Flash is empty**

In some cases, when eMMC flash is erased or the u-boot is under development, we need a way to boot from SD card first. Users need to shunt cross the ***TEST#*** pin to ground. In this way, *SMARC-iMX8M* will always boot up from SD card.

Fuse flash.bin to the SD card.

**~/smarc-imx8m-rocto-release/<build dir>/tmp/deploy/images/<machine name>/**

$ *sudo dd if=<boot file> of=${DISK} bs=1024 seek=33*

**If on-module eMMC Flash is not empty**

The <*boot file*> is pre-installed in on-module eMMC flash at factory default. SMARC-iMX8M is designed to always boot up from on-module eMMC flash and to load Image, device tree blob and root file systems based on the setting of *BOOT_SEL*. If users need to fuse your own flash.bin or perform u-boot upgrade. This section will instruct you how to do that.

Copy <boot file> to the second partition home directory of your SD card and boot into SD card. Go to home directory and you should see flash.bin file.

**~/smarc-imx8m-rocto-release/<build dir>/tmp/deploy/images/<machine name>/**

```
$ sudo cp -v <boot file> /media/rootfs/home/root/
```

Fuse <boot file> to the on-module eMMC flash. (The eMMC flash is emulated as /dev/mmcblk0 in SMARC-iMX8M)

**home directory**

```
$ sudo dd if=<boot file> of=/dev/mmcblk0 bs=1024 seek=33
```



1. If your u-boot hasn't been finalized and still under development, it is recommended to shunt cross the test pin and boot directly from SD card first. Once your u-boot is fully tested and finalized, you can fuse your <boot file> to eMMC flash.
2. When *TEST#* pin of SMARC-iMX8M is not shunt crossed, it will always boot up from on-module eMMC flash. U-boot will read the *BOOT_SEL* configuration and determine where it should load Image and device tree blob. When TEST# is shunt crossed (pull low), it will always boot up from SD card.

## uEnv.txt based bootscript

Create "uEnv.txt" boot script: ($ vim uEnv.txt)

**~/uEnv.txt**

```
optargs="video=HDMI-A-1:1920x1080-32@60 consoleblank=0"
#optargs="video=HDMI-A-1:3840x2160-32@30 consoleblank=0"
#optargs="video=HDMI-A-1:3840x2160-32@60 consoleblank=0"
#console port SER3
console=ttymxc0,115200 earlycon=ec_imx6q,0x30860000,115200
#console port SER2
#console=ttymxc1,115200 earlycon=ec_imx6q,0x30890000,115200
#console port SER1
#console=ttymxc2,115200 earlycon=ec_imx6q,0x30880000,115200
#console port SER0
#console=ttymxc3,115200 earlycon=ec_imx6q,0x30A60000,115200
mmcdev=1
mmcpart=1
image=Image
loadaddr=0x40480000
fdt_addr=0x43000000
mmcroot=/dev/mmcblk1p2 rw
usbroot=/dev/sda2 rw
mmcrootfstype=ext4 rootwait fixrtc
netdev=eth0
ethact=FEC0
ipaddr=192.168.1.150
serverip=192.168.1.53
gatewayip=192.168.1.254
mmcargs=setenv bootargs console=${console} root=${mmcroot} rootfstype=${mmcrootfstype} ${optargs}
uenvcmd=run loadimage; run loadfdt; run mmcboot
```

```
# USB Boot
#usbargs=setenv bootargs console=${console} root=${usbroot} rootfstype=${mmcrootfstype} ${optargs}
#uenvcmd=run loadusbimage; run loadusbfdt; run usbboot
```

Copy uEnv.txt to the boot partition:

**~/**

```
$ sudo cp -v ~/uEnv.txt /media/boot/
```

## Install Kernel Image

Copy Image to the boot partition:

**~/smarc-imx8m-rocko-release/<build dir>/tmp/deploy/images/<machine name>/**

```
$ sudo cp -v Image /media/boot
```

## Install Kernel Device Tree Binary

**~/smarc-imx8m-rocko-release/<build dir>/tmp/deploy/images/<machine name>/**

```
$ sudo mkdir -p /media/boot/dtbs

$ sudo cp -v <device tree name> /media/boot/dtbs/fsl-smarcimx8mq.dtb
```

All available DTB files are listed in the table below.

| DTB File Name | Description |
|---|---|
| *fsl-smarcimx8mq.dtb* | Device tree blob for no display configuration. |
| *fsl-smarcimx8mq-hdmi.dtb* | Device tree blob for HDMI display configuration (DCSS). |
| *fsl-smarcimx8mq-hdmi-4k.dtb* | Device tree blob for HDMI 4k display configuration (DCSS). |
| *fsl-smarcimx8mq-lcdif-lvds.dtb* | Device tree blob for LCDIF LVDS display configuration. |
| *fsl-smarcimx8mq-dcss-lvds.dtb* | Device tree blob for DCSS LVDS display configuration. |
| *fsl-smarcimx8mq-dual-display.dtb* | Device tree blob for dual LVDS+HDMI display configuration. |

The device tree name in your SD card has be to fsl-smarcimx8mq.dtb

# Install Root File System

## Copy Root File System:

**Yocto Built Rootfs:**

**~/smarc-imx8m-rocko-release/<build dir>/tmp/deploy/images/<machine name>/**

```
$ sudo tar jxvf <filename.tar.bz2> -C /media/rootfs
```

**Note**

1. *SMARC-iMX8M* always boots up from on-module eMMCI flash first. The firmware in eMMC flash is factory pre-installed from Embedian. It will read the *BOOT_SEL* configuration that defined by SMARC specification on your carrier board and load Image and device tree blob from the partition one of the device (could be SD card, eMMC, GBE,..etc) that you selected.
2. MAC address is factory pre-installed at on board I2C EEPROM at offset 60 bytes. It starts with Embedian's vendor code *10:0D:32*. u-boot will read it and pass this parameter to kernel.
3. The kernel modules is included in the Yocto rootfs.

Remove SD card:

```
$ sync
$ sudo umount /media/boot
$ sudo umount /media/rootfs
```

# Feed Packages

The following procedure can be used on a Embedian *SMARC-iMX8M* device to download and utilize the feed file show above to install the *tcpdump* Ethernet packet analyzer program:

```
$ smart channel -y --add http://www.embedian.com/smarcimx8m-rocko-feed/imx8m_qt5wayland_update.repo

$ smart update

$ smart install tcpdump
```

# Writing Bitbake Recipes

In order to package your application and include it in the root filesystem image, you must write a BitBake recipe for it.

When starting from scratch, it is easiest to learn by example from existing recipes.

### Example HelloWorld recipe using autotools

For software that uses autotools (./configure; make; make install), writing recipes can be very simple:

```
DESCRIPTION = "Hello World Recipe using autotools"
HOMEPAGE = "http://www.embedian.com/"
SECTION = "console/utils"
PRIORITY = "optional"
LICENSE = "GPL"
PR = "r0"

SRC_URI = "git://git@git.embedian.com/developer/helloworld-autotools.git;protocol=ssh;tag=v1.0"
S = "${WORKDIR}/git"

inherit autotools
```

`SRC_URI` specifies the location to download the source from. It can take the form of any standard URL using http://, ftp://, etc. It can also fetch from SCM systems, such as git in the example above.

`PR` is the package revision variable. Any time a recipe is updated that should require the package to be rebuilt, this variable should be incremented.

`inherit autotools` brings in support for the package to be built using autotools, and thus no other instructions on how to compile and install the software are needed unless something needs to be customized.

`S` is the source directory variable. This specifies where the source code will exist after it is fetched from SRC_URI and unpacked. The default value is `${WORKDIR}/${PN}-${PV}`, where `PN` is the package name and `PV` is the package version. Both `PN` and `PV` are set by default using the filename of the recipe, where the filename has the format `PN_PV.bb`.

#### Example HelloWorld recipe using a single source file

This example shows a simple case of building a helloworld.c file directly using the default compiler (gcc). Since it isn't using autotools or make, we have to tell BitBake how to build it explicitly.

```
DESCRIPTION = "HelloWorld"
SECTION = "examples"
LICENSE = "GPL"

SRC_URI = "file://helloworld.c"

S = "${WORKDIR}"

do_compile() {
    ${CC} ${CFLAGS} ${LDFLAGS} helloworld.c -o helloworld
}

do_install() {
    install -d ${D}${bindir}
    install -m 0755 helloworld ${D}${bindir}
}
```

In this case, `SRC_URI` specifies a file that must exist locally with the recipe. Since there is no code to download and unpack, we set `S` to `WORKDIR` since that is where helloworld.c will be copied to before it is built.

`WORKDIR` is located at `${OETREE}/<build directory>/tmp/work/aarch64-poky-linux/<package name and version>` for most packages. If the package is machine-specific (rather than generic for the aarxh64 architecture), it may be located in the aarch64-mx8mq-poky-linux subdirectory depending on your hardware (this applies to kernel packages, images, etc).

`do_compile` defines how to compile the source. In this case, we just call gcc directly. If it isn't defined, `do_compile` runs `make` in the source directory by default.

`do_install` defines how to install the application. This example runs `install` to create a bin directory where the application will be copied to and then copies the application there with permissions set to 755.

`D` is the destination directory where the application is installed to before it is packaged.

`${bindir}` is the directory where most binary applications are installed, typically `/usr/bin`.

For a more in-depth explanation of BitBake recipes, syntax, and variables, see the Recipe Chapter of the OpenEmbedded User Manual.

# Setup eMMC

Setting up eMMC usually is the last step at development stage after the development work is done at your SD card or NFS environments. From software point of view, eMMC is nothing but a non-removable SD card on board. For *SMARC-iMX8M*, the SD card is always emulated as /dev/mmcblk1 and on-module eMMC is always emulated as /dev/mmcblk0. Setting up eMMC now is nothing but changing the device descriptor.

This section gives a step-by-step procedure to setup eMMC flash. Users can write a shell script your own at production to simplify the steps.

First, we need to backup the final firmware from your SD card or NFS.

## Prepare for eMMC binaries from SD card (or NFS):

Insert SD card into your Linux PC. For these instructions, we are assuming: DISK=/dev/mmcblk0, "lsblk" is very useful for determining the device id.

For these instruction, we are assuming: DISK=/dev/mmcblk0, "lsblk" is very useful for determining the device id.

```
$ export DISK=/dev/mmcblk0
```

Mount Partitions:

**On some systems, these partitions may be auto-mounted...**

```
$ sudo mkdir -p /media/boot/
```

```
$ sudo mkdir -p /media/rootfs/

for: DISK=/dev/mmcblk0
$ sudo mount ${DISK}p1 /media/boot/
$ sudo mount ${DISK}p2 /media/rootfs/

for: DISK=/dev/sdX
$ sudo mount ${DISK}1 /media/boot/
$ sudo mount ${DISK}2 /media/rootfs/
```

**Copy Image to rootfs partition:**

**~/smarc-imx8m-rocko-release/<build dir>/tmp/deploy/images/<machine name>**

```
$ sudo cp -v Image /media/rootfs/home/root
```

**Copy uEnv.txt to rootfs partition:**

Copy and paste the following contents to /media/rootfs/home/root ($ sudo vim /media/rootfs/home/root/uEnv.txt)

```
optargs="video=HDMI-A-1:1920x1080-32@60 consoleblank=0"
#optargs="video=HDMI-A-1:3840x2160-32@30 consoleblank=0"
#optargs="video=HDMI-A-1:3840x2160-32@60 consoleblank=0"
#console port SER3
console=ttymxc0,115200 earlycon=ec_imx6q,0x30860000,115200
#console port SER2
#console=ttymxc1,115200 earlycon=ec_imx6q,0x30890000,115200
#console port SER1
#console=ttymxc2,115200 earlycon=ec_imx6q,0x30880000,115200
#console port SER0
#console=ttymxc3,115200 earlycon=ec_imx6q,0x30A60000,115200
mmcdev=0
mmcpart=1
image=Image
loadaddr=0x40480000
fdt_addr=0x43000000
mmcroot=/dev/mmcblk0p2 rw
usbroot=/dev/sda2 rw
mmcrootfstype=ext4 rootwait fixrtc
netdev=eth0
ethact=FEC0
ipaddr=192.168.1.150
serverip=192.168.1.53
gatewayip=192.168.1.254
mmcargs=setenv bootargs console=${console} root=${mmcroot} rootfstype=${mmcrootfstype} ${optargs}
uenvcmd=run loadimage; run loadfdt; run mmcboot
# USB Boot
#usbargs=setenv bootargs console=${console} root=${usbroot} rootfstype=${mmcrootfstype} ${optargs}
#uenvcmd=run loadusbimage; run loadusbfdt; run usbboot
```

**Copy device tree blob to rootfs partition:**

**~/smarc-imx8m-rocko-release/<build dir>/tmp/deploy/images/<machine name>**

```
$ sudo cp -v <device tree blob> /media/rootfs/home/root/fsl-smarcimx8mq.dtb
```

**Copy real rootfs to rootfs partition:**

```
$ pushd /media/rootfs

$ sudo tar cvfz ~/smarcimx8mq-emmc-rootfs.tar.gz .

$ sudo mv ~/smarcimx8mq-emmc-rootfs.tar.gz /media/rootfs/home/root

$ popd
```

Remove SD card:

```
$ sync
$ sudo umount /media/boot
$ sudo umount /media/rootfs
```

## Copy Binaries to eMMC from SD card:

Insert this SD card into your SMARC-iMX8M device.

Now it will be almost the same as you did when setup your SD card, but the eMMC device descriptor is */dev/mmcblk0* now. Booting up the device.

```
$ export DISK=/dev/mmcblk0
```

Erase eMMC:

```
$ sudo dd if=/dev/zero of=${DISK} bs=2M count=16
```

Create Partition Layout:

```
$ sudo sfdisk ${DISK} <<-__EOF__
2M,48M,0x83,*
50M,,,
__EOF__
```

Format Partitions:

```
$ sudo mkfs.vfat -F 16 ${DISK}p1 -n boot
$ sudo mkfs.ext4 ${DISK}p2 -L rootfs
```

Mount Partitions:

```
$ sudo mkdir -p /media/boot/
$ sudo mkdir -p /media/rootfs/
$ sudo mount ${DISK}p1 /media/boot/
$ sudo mount ${DISK}p2 /media/rootfs/
```

## Install binaries for partition 1

Copy uEnv.txt/Image/*.dtb to the boot partition

```
$ sudo cp -v Image uEnv.txt /media/boot/
```

## Install Kernel Device Tree Binary

```
$ sudo mkdir -p /media/boot/dtbs
$ sudo cp -v fsl-smarcimx8mq.dtb /media/boot/dtbs/
```

# Install Root File System

```
$ sudo tar -zxvf smarcimx8mq-emmc-rootfs.tar.gz -C /media/rootfs
```

Unmount eMMC:

```
$ sync
$ sudo umount /media/boot
$ sudo umount /media/rootfs
```

Switch your Boot Select to eMMC and you will be able to boot up from eMMC now.

# Video Decoding

For playing video, we can use three solutions to support it.

a) # gplay-1.0 <video file>

b) # gst-launch-1.0 playbin uri=file://<video absolute path>

c) ( i ) if video container on .mp4 format

# gst-launch-1.0 filesrc location=<file name.mp4> typefind=true ! video/quicktime ! qtdemux ! queue max-size-time=0 ! vpudec ! queue max-size-time=0 ! kmssink force-hantrope=true sync=false &

( ii ) if video container on .ts format

# gst-launch-1.0 filesrc location=<file name.ts> typefind=true ! video/mpegts ! tsdemux ! queue max-size-time=0 ! vpudec ! queue max-size-time=0 ! waylandsink

version 1.0a, 4/10/2019

Last updated 2019-04-10