pITX-MX8M-PLUS-BSP-Gatesgarth

On this page:

Building NXP/Embedian’s Yocto Gatesgarth BSP Distribution

Eric Lee

version 1.0a, 08/18/2022

Introduction

This document describes how Embedian builds a customized version of NXP’s i.MX8M Plus official Yocto Gatesgarth BSP release for Embedian's *pITX-MX8M-PLUS* 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 *pITX-MX8M-PLUS* 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 NXP's i.MX8M Plus 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-pitximx8mp-gatesgarth* Yocto Layer

The supplied meta-pitximx8mp-gatesgarth Yocto compliant layer has the following organization:

```
.
|-- conf
|   |-- layer.conf
|   |-- site.conf
|   |-- machine
|       |-- pitximx8mp2g.conf
|       |-- pitximx8mp4g.conf
|   |   `-- pitximx8mp6g.conf
|-- README
|-- recipes-bsp
|   |-- u-boot
|       `-- u-boot-potximx8mp_2020.04.bb
|   |-- imx-vpu-hantro-vc
|       `-- imx-vpu-hantro-vc_1.3.0.bbappend
|   |-- alsa-state
|       |   |-- alsa-state
|       |   |   `-- asound.state
|       `-- alsa-state.bbappend
|   |-- pm-utils
|       `-- pm-utils_%.bbappend
|   |-- imx-mkimage
|       `-- imx-boot_1.0.bbappend
|-- recipes-core
|   |-- busybox
|       `-- busybox_%.bbappend
|       |   |-- busybox
|       |   |   |-- ftpget.cfg
|       |   |   `-- defconfig
|   |-- base-files
|       `-- base-files_%.bbappend
|       |   |-- base-files
|       |   |   |-- issue
|       |   |   `-- issue.net
|   |-- packagegroups
|       `-- packagegroup-core-tools-testapps.bbappend
|   |-- psplash
|       `-- psplash_git.bbappend
|       |   |-- files
|       |   |   |-- 0001-psplash-Change-colors-for-the-Embedian-Yocto-logo.patch
|       |   |   |-- psplash-poky.png
|       |   |   `-- psplash-bar.png
|   |-- udev
|       |   |-- files
|       |   |   |-- pitximx8mp2g
|       |   |   |   `-- usb-power.rules
|       |   |   |-- pitximx8mp4g
|       |   |   |   `-- usb-power.rules
|       |   |   |-- pitximx8mp6g
```

```
|       |   |   |   `-- usb-power.rules
|       `-- udev-rules-imx.bbappend
|-- recipes-support
|   |-- vim
|   |   `-- vim_%.bbappend
|-- recipes-kernel
|   |-- linux
|       `-- linux-pitximx8mp_5.10.bb
`-- scripts
|   `-- emb_mk_yocto_sdcard
```

Notes on ***meta-pitximx8mp-gatesgarth*** layer content

`conf/machine/*`

This folder contains the machine definitions for the *pitximx8mp2g/pitximx8mp4g/pitximx8mp6g* 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 *pitximx8mp2g/pitximx8mp4g/pitximx8mp6g* platform.

`recipes-bsp/alsa-state/*`

This folder contains sgtl5000 sound chip default state for *pitximx8mp2g/pitximx8mp4g/pitximx8mp6g* platform.

`recipes-bsp/imx-mkimage/*`

This folder contains imx-mkimage tool for *pitximx8mp2g/pitximx8mp4g/pitximx8mp6g* platform.

`recipes-bsp/imx-vpu-hantro-vc/*`

This folder adds compatible machine for *pitximx8mp2g/pitximx8mp4g/pitximx8mp6g* platform

`recipes-core/busybox/*`

This folder remove telnetd from bysybox for *pitximx8mp2g/pitximx8mp4g/pitximx8mp6g* platform.

`recipes-core/psplash/*`

This folder customized Yocto boot psplash for *pitximx8mp2g/pitximx8mp4g/pitximx8mp6g* platform.

`recipes-support/vim/*`

This folder fixes xwayland build error for *pitximx8mp2g/pitximx8mp4g/pitximx8mp6g* platform.

`recipes-kernel/linux/*`

Contains the recipes needed to build the *pitximx8mp2g/pitximx8mp4g/pitximx8mp6g* Linux kernels.

Setting Up the Tools and Build Environment

To build the latest NXP *i.MX8M* Plus fsl-bsp-release, you first need an Ubuntu 16.04 0r 18.04LTS 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 or 18.04 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 pv
```

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++9v5:i386 libcrypto++-dev:i3
86 liblzo2-dev:i386 libusb-1.0-0:i386 libusb-1.0-0-dev:i386 uuid-dev:i386
```

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 ~/pitx-imx8mp-gatesgarth-release

$ cd ~/pitx-imx8mp-gatesgarth-release

$ repo init -u https://source.codeaurora.org/external/imx/imx-manifest -b imx-linux-gatesgarth -m
imx-5.10.9-1.0.0.xml

$ repo sync
```

Download the Embedian Yocto build script and meta layer.

```
$ wget ftp://ftp.embedian.com/public/dev/minfs/pitx-imx8mp-bsp-release/pitximx8mp-setup-release.sh

$ chmod 444 pitximx8mp-setup-release.sh

$ cd sources

$ git clone git@git.embedian.com:developer/meta-pitximx8mp.git meta-pitximx8mp-gatesgarth -b
pitx8mp_5.10.9_1.0.0_gatesgarth

$ cd ~/pitx-imx8mp-gatesgarth-release

$ DISTRO=fsl-imx-wayland MACHINE=pitximx8mp4g source pitximx8mp-setup-release.sh -b build-wayland
```

Choose "y" to accept EULA.

This script will create and bring you to ~/pitx-imx8mp-gatesgarth-release/build-wayland directory.

Note
The last line of the above script

```
$ DISTRO=<distro name> MACHINE=<machine name> source pitximx8mp-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 Plus.)
2. <machine name>
   - *pitximx8mp2g - if your board is with 2GB LPDDR4.*
   - *pitximx8mp4g* - if your board is with 4GB LPDDR4.
   - *pitximx8mp6g* - if your board is with 6GB LPDDR4.

In this document, we will use *pitximx8mp4g* as the example of machine name. Users need to change different machine name if you have different LPDDR4 variants.

Building the target platforms

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

```
$ MACHINE=pitximx8mp4g bitbake -k fsl-image-qt5-validation-imx
or
$ MACHINE=pitximx8mp4g 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 pitximx8mp4g and your gui image is without QT5 , the following command gives you as an example.

```
$ MACHINE=pitximx8mp4g bitbake -k fsl-image-validation-imx
```

The first build takes time.

Once it done, you can find all required images under ~/pitx-imx8mp-gatesgarth-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 tcpdump program and makes it available on the feed site:

```
$ MACHINE=pitximx8mp4g bitbake tcpdump
$ MACHINE=pitximx8mp4g bitbake package-index
```

Once the build(s) are completed you'll find the resulting images, rpm and licenses in folder *~/pitx-imx8mp-gatesgarth-release/<build directory>/tmp/deploy*.

`deploy/images/<machine name>/*`

This folder contains the binary images for the root file system and the Embedian pITX-*MX8M-PLUS* 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 *pITX-MX8M-PLUS*

`deploy/images/<machine name>/Image`

The kernel Image for *pITX-MX8M-PLUS*.

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

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
imx8mp-pitx.dtb	Device tree blob for no display configuration.
imx8mp-pitx-hdmi.dtb	Device tree blob for HDMI display configuration.
imx8mp-pitx-lvds.dtb	Device tree blob for LVDS display configuration.
imx8mp-pitx-m7.dtb	Device tree blob for Cortex-M7 co-processor configuration.

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

Embedian root file system images for software development on Embedian's *pITX-MX8M-PLUS* 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 *pITX-MX8M-PLUS* with QT5.

deploy/deb/*

This folder contains all the packages used to construct the root file system images. They are in ***deb*** format (similar format to Debian 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 base-feeds.conf) that is used internally at Embedian to install upgrades onto a ***pITX-MX8M-PLUS*** platform without reflashing the file system:

```
src/gz all http://<ip address>/all

src/gz aarch64 http://<ip address>/aarch64

src/gz aarch64-mx8mp http://<ip addresss>/aarch64-mx8m

src/gz pitximx8mp4g http://<ip addresss>/pitximx8mp4g
```

`deploy/licenses/*`

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

Setup microSD Card Manually

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

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

Erase microSD card:

```
$ sudo dd if=/dev/zero of=${DISK} bs=1M count=160
```

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`)

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 microSD card first. Users need to set the SW2 port 1-3 as (ON ON ON). In this way, *pITX-MX8M-PLUS* will always boot up from SD card.

Fuse flash.bin to the microSD card.

~/smarc-imx8mp-gatesgarth-release/<build dir>/tmp/deploy/images/<machine name>/

$ sudo dd if=<boot file> of=${DISK} bs=1024 seek=32

If on-module eMMC Flash is not empty

The <*boot file*> is pre-installed in on-module eMMC flash at factory default. *pITX-MX8M-PLUS* 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 SW2 port 1-3. 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 microSD card and boot into microSD card. Go to home directory and you should see flash.bin file.

~/smarc-imx8mp-gatesgarth-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/mmcblk2 in *pITX-MX8M-PLUS*)

home directory

$ sudo dd if=<boot file> of=/dev/mmcblk2 bs=1024 seek=32

1. If your u-boot hasn't been finalized and still under development, it is recommended to set the SW2 port 1-3 as (ON ON ON) and boot directly from microSD card first. Once your u-boot is fully tested and finalized, you can fuse your <boot file> to eMMC flash.
2. When SW2 port 1-3 is set (OFF ON OFF) in *pITX-MX8M-PLUS*, it will always boot up from on-module eMMC flash. U-boot will read the *SW2* configuration and determine where it should load Image and device tree blob. When SW2 port 1-3 is set (ON ON ON) it will always boot up from microSD 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=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-imx8mp-gatesgarth-release/<build dir>/tmp/deploy/images/<machine name>/

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

Install Kernel Device Tree Binary

~/smarc-imx8mp-gategarth-release/<build dir>/tmp/deploy/images/<machine name>/

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

$ sudo cp -v <device tree name> /media/boot/dtbs/imx8mp-pitx.dtb
```

All available DTB files are listed in the table below.

DTB File Name	Description
imx8mp-pitx.dtb	Device tree blob for no display configuration.
imx8mp-pitx-hdmi.dtb	Device tree blob for HDMI display configuration.
imx8mp-pitx-lvds.dtb	Device tree blob for LVDS display configuration.
imx8mp-pitx-m7.dtb	Device tree blob for Cortex-M7 co-processor configuration.

The device tree name in your microSD card has be to *imx8mp-pitx.dtb*

Install Root File System

Copy Root File System:

Yocto Built Rootfs:

~/smarc-imx8mp-gatesgarth-release/<build dir>/tmp/deploy/images/<machine name>/

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

Note

1. *pITX-MX8M-PLUS* always boots up from on-module eMMC flash first. The firmware in eMMC flash is factory pre-installed from Embedian. It will read the *SW2 port 1-3* configuration on your carrier board and load Image and device tree blob from the partition one of the device (could be microSD card or eMMC) 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 microSD card:

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

Setup microSD Card Automatically

This section tells you how to set up an microSD card automatically. It mainly uses a script to do all the steps in the above section.

```
$ cd ~/pitx-imx8mp-gatesgarth-release

$ sudo MACHINE=pitximx8mp4g
sources/meta-pitximx8mp-gatesgarth/scripts/emb_mk_yocto_sdcard/emb-create-yocto-sdcard.sh /dev/sdX
```

The default output display is HDMI. If you need to use other device tree output, change "DISPLAY" parameters in the script.

Set SW2 port 1-3 as (ON ON ON). The module will boot up from microSD card.

Feed Packages

You need to setup Apache2 web server on your development host machine first.

The Apache server default web page directory is **/var/www/html** .We need to populate it with a link pointing to our deb **package** repository.

```
sudo ln -s /path/to/build-yocto/tmp/deploy/deb /var/www/html/deb
```

The following procedure can be used on a Embedian *pITX-MX8M-PLUS* device to download and utilize the feed file show above to install the *tcpdump* terminal emulation program:

```
# vim /etc/apt/sources.list.d/yocto.list
```

Only keep the following four lines:

```
deb https://<ip address>/all ./

deb http://<ip address>/cortexa53-crypto ./

deb http://<ip address>/cortexa53-crypto-mx8mp ./

deb http://<ip address>/pitximx8mp4g ./
```

```
# apt-get update

# apt-get upgrade

# apt-get 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/cortexa53-crypto-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 cortexa53-crypto-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 Manually

Setting up eMMC usually is the last step at development stage after the development work is done at your microSD card or NFS environments. From software point of view, eMMC is nothing but a non-removable microSD card on board. For *pITX-MX8M-PLUS*, the microSD card is always emulated as /dev/mmcblk1 and on-module eMMC is always emulated as /dev/mmcblk2. 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 microSD card or NFS.

Prepare for eMMC binaries from microSD card (or NFS):

Insert microSD 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-imx8mp-gatesgarth-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=ttymxc3,115200 earlycon=ec_imx6q,0x30a60000,115200
mmcdev=2
mmcpart=1
image=Image
loadaddr=0x40480000
fdt_addr=0x43000000
mmcroot=/dev/mmcblk2p2 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-imx8mp-gatesgarth-release/<build dir>/tmp/deploy/images/<machine name>

```
$ sudo cp -v <device tree blob> /media/rootfs/home/root/imx8mp-pitx.dtb
```

Copy real rootfs to rootfs partition:

```
$ pushd /media/rootfs

$ sudo tar cvfz ~/pitximx8mp-emmc-rootfs.tar.gz .

$ sudo mv ~/pitximx8mp-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 microSD card:

Insert this microSD card into your *pITX-MX8M-PLUS* device.

Now it will be almost the same as you did when setup your microSD card, but the eMMC device descriptor is */dev/mmcblk2* now. Booting up the device.

```
$ export DISK=/dev/mmcblk2
```

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 imx8mp-pitx.dtb /media/boot/dtbs/
```

Install Root File System

```
$ sudo tar -zxvf pitximx8mp-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.

Setup eMMC Automatically

Boot up the module from microSD card and run the following script. The Yocto images will be written into on-module eMMC.

```
$ emb-create-yocto-emmc.sh /dev/mmcblk2 >/dev/null 2>&1
```

Shutdown the device. Set SW2 port 1-3 as (OFF ON OFF). The module will boot up from on-module eMMC.

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

WiFi

The BSP includes NXP 88W8997 wifi chipset. Users can choose mPCIe or M.2 key E form factor wifi modules based on NXP 88W8997 chipset.

M.2 Form Factor:

- AzureWave P/N: AW-CM276MA-PUR
- Laird Connectivity P/N: 60-2230C
- Embedded Artists 1YM M.2 Module

mPCIe Factor:

- Globascale Technologies NXP 88W8997 2x2 WiFi 802.11ac+BT 5.0 mini PCIe Card w/ Two External SMA Antennas

Get 88W8997 Firmware

```
$ git clone https://github.com/NXP/imx-firmware.git -b lf-5.10.y_1.0.0
```

Copy the firmware imx-firmware/nxp/FwImage_8997/pcieuart8997_combo_v4.bin into device /lib/firmware/nxp/ directory. (Replace the original one)

Boot up the device and load the driver modules in the kernel.

```
root@pitximx8mp4g:~# modprobe moal mod_para=nxp/wifi_mod_para.conf
[ 33.834782] can2-stby: disabling
[ 33.838051] VSD1_3V3: disabling
```

```
[ 33.979809] wlan: Loading MWLAN driver
[ 33.984701] wlan_pcie 0000:01:00.0: enabling device (0000 -> 0002)
[ 33.991014] Attach moal handle ops, card interface type: 0x204
[ 34.000829] PCIE8997: init module param from usr cfg
[ 34.005845] card_type: PCIE8997, config block: 0
[ 34.010483] cfg80211_wext=0xf
[ 34.013465] wfd_name=p2p
[ 34.016011] max_vir_bss=1
[ 34.018632] cal_data_cfg=none
[ 34.021611] drv_mode = 7
[ 34.024159] ps_mode = 2
[ 34.026604] auto_ds = 2
[ 34.029084] fw_name=nxp/pcieuart8997_combo_v4.bin
[ 34.033830] rx_work=1 cpu_num=4
[ 34.037010] Attach mlan adapter operations.card_type is 0x204.
[ 34.046917] Request firmware: nxp/pcieuart8997_combo_v4.bin
[ 35.013725] FW download over, size 627620 bytes
[ 35.879247] WLAN FW is active
[ 35.882226] on_time is 35807347500
[ 35.917890] fw_cap_info=0x18fcffa3, dev_cap_mask=0xffffffff
[ 35.923500] max_p2p_conn = 8, max_sta_conn = 8
[ 35.956580] wlan: version = PCIE8997-16.68.10.p16-MXM5X16214-GPL-(FP92)
[ 35.966307] wlan: Driver loaded successfully
root@pitximx8mp4g:~#
```

Verify that the module is now visible to the system.

```
root@pitximx8mp4g:~# ifconfig -a
can0: flags=128<NOARP> mtu 16
unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 txqueuelen 10 (UNSPEC)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
device interrupt 35

can1: flags=128<NOARP> mtu 16
unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 txqueuelen 10 (UNSPEC)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
device interrupt 36

eth0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
ether 10:0d:32:01:00:01 txqueuelen 1000 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth1: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
ether 10:0d:32:02:00:01 txqueuelen 1000 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
device interrupt 54

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 3452 bytes 216146 (211.0 KiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 3452 bytes 216146 (211.0 KiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

```
mlan0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
ether 4a:6b:15:b3:7f:a4 txqueuelen 1000 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

p2p0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
ether 2a:08:86:b1:27:cb txqueuelen 1000 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

uap0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
ether 5a:57:c4:46:2b:68 txqueuelen 1000 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@pitximx8mp4g:~#
```

In case you need to see which network and you can scan it and select the one you need.

```
root@pitximx8mp4g:~# iwlist mlan0 scan
mlan0 Scan completed :
Cell 01 - Address: D8:FE:E3:5F:68:98
ESSID:"Risetek"
Mode:Master
Frequency=2.412 GHz (Channel 1)
```

Identify the network and add it to the WPA supplicant file.

```
root@pitximx8mp4g:~# vim /etc/wpa_supplicant.conf
```

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
update_config=1

network={
scan_ssid=1
ssid="embedian"
psk="xxxxxxxxxx"
}
```

Associate the Wi-Fi with config

```
root@pitximx8mp4g:~# wpa_supplicant -B -i mlan0 -c /etc/wpa_supplicant.conf
Successfully initialized wpa_supplicant
nl80211: kernel reports: Match already configured
nl80211: kernel reports: Match already configured
nl80211: kernel reports: Match already configured
nl80211: kernel reports: Match already configured
nl80211: kernel reports: Match already configured
nl80211: kernel reports: Match already configured
nl80211: kernel reports: Match already configured
nl80211: kernel reports: Match already configured
nl80211: kernel reports: Match already configured
nl80211: kernel reports: Match already configured
nl80211: kernel reports: Match already configured
```

```
nl80211: kernel reports: Match already configured
nl80211: kernel reports: Match already configured
nl80211: kernel reports: Match already configured
nl80211: kernel reports: Match already configured
nl80211: kernel reports: Match already configured
nl80211: kernel reports: Match already configured
nl80211: kernel reports: Match already configured
nl80211: kernel reports: Match already configured
nl80211: kernel reports: Match already configured
nl80211: kernel reports: Match already configured
nl80211: kernel reports: Match already configured
rfkill: Cannot open RFKILL control device
root@pitximx8mp4g:~#
```

Check if you have right SSID associated.

```
root@pitximx8mp4g:~# iwconfig mlan0
mlan0 IEEE 802.11-DS ESSID:"embedian" [14]
Mode:Managed Frequency=5.745 GHz Access Point: 48:EE:0C:ED:D7:38
Bit Rate:6.5 Mb/s Tx-Power=24 dBm
Retry limit:9 RTS thr=2347 B Fragment thr=2346 B
Encryption
key:****-****-****-****-****-****-****-****-****-****-****-****-****-****-****-****-****-****-****-****-
****-****-****-****-****-****-****-****-****-**** Security mode:open
Power Management:off
Link Quality=3/5 Signal level=-66 dBm Noise level=-91 dBm
Rx invalid nwid:0 Rx invalid crypt:0 Rx invalid frag:27439
Tx excessive retries:8 Invalid misc:24 Missed beacon:0

root@pitximx8mp4g:~#
```

Use DHCP to get IP

```
root@pitximx8mp4g:~# udhcpc -i mlan0
udhcpc: started, v1.32.0
udhcpc: sending discover
udhcpc: sending select for 192.168.1.57
udhcpc: lease of 192.168.1.57 obtained, lease time 86400
/etc/udhcpc.d/50default: Adding DNS 192.168.1.254
root@pitximx8mp4g:~#
```

You should be able to ping local network now.

```
root@pitximx8mp4g:~# ping 192.168.1.10
PING 192.168.1.10 (192.168.1.10) 56(84) bytes of data.
64 bytes from 192.168.1.10: icmp_seq=1 ttl=64 time=2141 ms
64 bytes from 192.168.1.10: icmp_seq=2 ttl=64 time=1120 ms
64 bytes from 192.168.1.10: icmp_seq=3 ttl=64 time=95.7 ms
64 bytes from 192.168.1.10: icmp_seq=4 ttl=64 time=1.63 ms
```

Modify /etc/resolv.conf of your preference, you will be able to ping out.

```
root@pitximx8mp4g:~# vim /etc/resolv.conf
```

```
nameserver 8.8.8.8
nameserver 8.8.4.4
```

```
root@pitximx8mp4g:/etc# ping www.google.com
PING www.google.com (172.217.163.36) 56(84) bytes of data.
64 bytes from maa05s01-in-f4.1e100.net (172.217.163.36): icmp_seq=1 ttl=117 time=7.23 ms
64 bytes from tsa01s13-in-f4.1e100.net (172.217.163.36): icmp_seq=2 ttl=117 time=39.7 ms
64 bytes from maa05s01-in-f4.1e100.net (172.217.163.36): icmp_seq=3 ttl=117 time=7.50 ms
64 bytes from tsa01s13-in-f4.1e100.net (172.217.163.36): icmp_seq=4 ttl=117 time=5.29 ms
64 bytes from tsa01s13-in-f4.1e100.net (172.217.163.36): icmp_seq=5 ttl=117 time=4.65 ms
64 bytes from tsa01s13-in-f4.1e100.net (172.217.163.36): icmp_seq=6 ttl=117 time=5.01 ms

--- www.google.com ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5010ms
rtt min/avg/max/mdev = 4.649/11.560/39.682/12.623 ms
```

version 1.0a, 8/18/2022

Last updated 2022-08-18