

pITX-iMX8MP-Standalone

- Build and Install Linux System for pITX-MX8M-PLUS (Dual and Quad Core)
- Availability
- Carrier Board
- Basic Resources
- ARM Cross Compiler: GCC
- Generating SSH Keys
 - Step 1. Check for SSH keys
 - Step 2. Generate a new SSH key
 - Step 3. Add your SSH key to Embedian Gitlab Server
- Boot File: flash.bin
- Linux Kernel
- Root File System
- Setup microSD Card
 - Install Boot File
 - uEnv.txt based bootscript
 - Install Kernel Image
 - Install Kernel Device Tree Binary
- Install Root File System and Kernel Modules
 - Copy Root File System:
 - Copy Kernel Modules:
- Setup eMMC
 - Prepare for eMMC binaries from microSD card (or NFS):
 - Copy Binaries to eMMC from microSD card:
 - Install binaries for partition 1
 - Install Kernel Device Tree Binary
- Install Root File System
- Video Decoding
- WiFi

Build and Install Linux System for *pITX-MX8M-PLUS* (Dual and Quad Core)

This document provides instructions for advanced users how Embedian offers patches and builds a customized version of u-boot and linux kernel for Embedian's *pITX-MX8M-PLUS* product platform and how to install the images to bring the evaluation board up and running.

Our aim is to fully support our hardware through device drivers. We also provide unit tests so that testing a board is easy and custom development can start precisely.

The host Linux machine is recommended Ubuntu 20.04 or 22.04.

Once you have Ubuntu 20.04 or 22.04 LTS running, install the additional required support packages using the following console command:

```
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib build-essential chrpath socat cpio python python3 python3-pip python3-pexpect xz-utils debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa libsdl1.2-dev pylint xterm rsync curl zstd lz4 libssl-dev pv device-tree-compiler libghc-gnutls-dev
```

Availability

pITX-MX8M-PLUS at Embedian

Carrier Board

pITX-IOB-2201 (Expansion IO board for pITX-MX8M-PLUS) at Embedian

Basic Resources

- ARM Cross Compiler
 - ARM: <https://developer.arm.com/downloads/-/gnu-a>
- Bootloader
 - Das U-Boot – the Universal Boot Loader <http://www.denx.de/wiki/U-Boot>
 - Source – <http://git.denx.de/?p=u-boot.git;a=summary>
- Linux Kernel
 - Linus's Mainline tree: <http://git.kernel.org/?p=linux/kernel/git/torvalds/linux.git;a=summary>
 - NXP Linux source tree: [git://github.com/nxp-imx/linux-imx.git](https://github.com/nxp-imx/linux-imx.git)
 - NXP Yocto BSP meta layer: <https://github.com/nxp-imx/meta-imx/meta-bsp>
 - Freescale community BSP release: <https://github.com/Freescale/meta-freescale-distro>
 - Embedian pITX-MX8M-PLUS Linux kernel source tree: git@git.embedian.com:developer/smarc-fsl-linux-kernel.git or [git@github.com:embedian/smarc-fsl-linux-kernel.git](https://github.com/embedian/smarc-fsl-linux-kernel.git)
- ARM based rootfs
 - Debian Squeeze: <http://www.debian.org/>

ARM Cross Compiler: GCC

This is a pre-built (32bit) version of Linaro GCC that runs on generic linux, so 64bit users need to make sure they have installed the 32bit libraries for their distribution.

debian based	extra	pkgs: (sudo apt-get update ; sudo apt-get install xyz)
Ubuntu 20.04		ia32-libs
Debian 11 (Bullseye)	sudo dpkg --add-architecture i386	libc6:i386 libstdc++6:i386 libncurses5:i386 zli b1g:i386
Ubuntu 20.10 -> 22.04		libc6:i386 libstdc++6:i386 libncurses5:i386 zli b1g:i386
Red Hat/Centos/Fedora		libstdc++.i686 ncurses-devel.i686 zlib.i686
Red Hat based (rpm)	extra	pkgs: (yum install xyz)
Red Hat/Centos/Fedora		libstdc++.i686 ncurses-devel.i686 zlib.i686
Ubuntu 22.04		ia32-libs
Ubuntu 20.10 -> 22.04		libc6:i386 libstdc++6:i386 libncurses5:i386 zli b1g:i386

To build Embedian's *pITX-MX8M-PLUS* u-boot and linux kernel, you will need to install the following ARM compiler:

For **u-boot 2022.04**, you need to use the following Arm compiler.

```
$ wget -c https://developer.arm.com/-/media/Files/downloads/gnu-a/9.2-2019.12/binrel/gcc-arm-9.2-2019.12-x86_64-aarch64-none-linux-gnu.tar.xz

$ sudo tar -Jxvf gcc-arm-9.2-2019.12-x86_64-aarch64-none-linux-gnu.tar.xz -C /opt

$ export CC=/opt/gcc-arm-9.2-2019.12-x86_64-aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu-
```

Test:

If this test fails, verify that you have the 32bit libraries installed on your development system.

```
$ ${CC}gcc --version
arm-none-linux-gnueabi-gcc (GNU Toolchain for the A-profile Architecture 9.2-2019.12 (arm-9.10)) 9.2.1
20191025
```

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 ed25519 -C "your_email@example.com"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/eric/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/eric/.ssh/id_ed25519
Your public key has been saved in /home/eric/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:SS9opo/QHxT2cCwLX+ulhn3ZUVdhdG88vvliOVHJ/6c your_email@example.com
The key's randomart image is:
+--[ED25519 256]--+
|      . . . .+B|
|      = . . .O+|
|      = = . . O.=|
|      . O * O O.=O|
|      = S * O .O.|
|      . = O . . +|
|      . O .      =.|
|      . + .      = +|
|      . O      .E+O|
+-----[SHA256]-----+
```

Step 3. Add your SSH key to Embedian Gitlab Server

Copy the key to your clipboard.

```
$ cat ~/.ssh/id_ed25519
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDQUEnh8uGpfxaZVU6+uE4bsDrs/tEE5/BPW7jMAxak
6qgOh6nUrQGBWS+VxMM2un3KzwwLRJSj8G4TnTK2CSm1BvR+X8ZeXNTyAdaDxULs/StVhH+QRtFEGy4o
iMIzvIlTyORY89jzhIsgZzwr0lnqoSeWWASd+59JWtFjVy0nwVNVtbek7NfuIGGAPaijO5Wnshr2uChB
Pk8ScGjQ3z4VqNXP6CWhCXTqIk7EQ17yX2GKd6FgEFrzae+5Jf63Xm8g6abbE3ytCrMT/jYy500j2XSg
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 you are done.

Boot File: flash.bin

The boot file is called **flash.bin**. It is made up of some pieces of programs. This section instruct you how to generate flash.bin.

1. Download the imx-mkimage tool and apply Embedian's patch to accept Embedian's device tree blob.

```
$ git clone https://github.com/nxp-imx/imx-mkimage -b lf-5.15.71_2.2.0
$ cd imx-mkimage
$ wget -c ftp://ftp.embedian.com/public/pitxmx8mp/0002-change-uboot-device-tree-name.patch
$ patch -p1 <0002-change-uboot-device-tree-name.patch
$ cd ../
```

2. Get and Build the ARM Trusted firmware and copy bl31.bin to imx-mkimage/iMX8MP directory.

```
$ git clone https://github.com/nxp-imx/imx-atf -b lf_v2.6
$ cd imx-atf

$ wget -c ftp://ftp.embedian.com/public/pitxmx8mp/0002-imx-atf-change-console-to-ttymxc3.patch

$ patch -p1 <0002-imx-atf-change-console-to-ttymxc3.patch
$ make CROSS_COMPILE=${CC} PLAT=imx8mp bl31
$ cp build/imx8mp/release/bl31.bin ../imx-mkimage/iMX8M/
$ cd ../
```

3. Get the DDR firmware and copy to imx-mkimage/iMX8M/ directory.

```
$ wget https://www.nxp.com/lgfiles/NMG/MAD/YOCTO/firmware-imx-8.18.bin
$ chmod a+x firmware-imx-8.18.bin

$ ./firmware-imx-8.18
enter "y" to accept EULA

$ cd firmware-imx-8.18

$ cp firmware/ddr/synopsys/lpddr4_pmu_train_1d_dmem_202006.bin ../imx-mkimage/iMX8M/
$ cp firmware/ddr/synopsys/lpddr4_pmu_train_1d_imem_202006.bin ../imx-mkimage/iMX8M/
$ cp firmware/ddr/synopsys/lpddr4_pmu_train_2d_dmem_202006.bin ../imx-mkimage/iMX8M/
$ cp firmware/ddr/synopsys/lpddr4_pmu_train_2d_imem_202006.bin ../imx-mkimage/iMX8M/

$ cp firmware/hdmi/cadence/signed_hdmi_imx8m.bin ../imx-mkimage/iMX8M/
```

```
$ cd ../
```

4. Clone the U-Boot source code from [Embedian Git Server](#) and copy related files to imx-mkimage/iMX8M/ directory.

Download:

For u-boot v2022.04:

```
$ git clone git@git.embedian.com:developer/smarc-t335x-uboot.git v2022.04 -b emb_1f_v2022.04
```

Configure and Build:

```
$ make ARCH=arm CROSS_COMPILE=${CC} distclean
$ make ARCH=arm CROSS_COMPILE=${CC} pitxmx8mp_4g_defconfig
$ make ARCH=arm CROSS_COMPILE=${CC}
```

Note1:

If the board is 2GB LPDDR4 in commercial or industrial temperature, use

```
$ make ARCH=arm CROSS_COMPILE=${CC} pitxmx8mp_2g_defconfig
```

If the board is 4GB LPDDR4, use

```
$ make ARCH=arm CROSS_COMPILE=${CC} pitxmx8mp_4g_defconfig
```

If the board is 6GB LPDDR4, use

```
$ make ARCH=arm CROSS_COMPILE=${CC} pitxmx8mp_6g_defconfig
```

Note 2:

The *pITX-MX8M-PLUS* module always boot up from the on-module *eMMC* flash. The factory default will be *flash.bin* pre-installed. In some cases when the *eMMC* flash is empty or needs to be upgraded. Users can set SW2 port 1-3 as (ON ON ON). In this way, the *pITX-MX8M-PLUS* module will boot up to carrier microSD card. The *flash.bin* image for eMMC and microSD card is the same, the difference is how you flash *flash.bin* and how SW2 port 1-3 is set. This will be explained in the "Setup microSD card" section.

Copy u-boot-nodtb.bin spl/u-boot-spl.bin arch/arm/dts/imx8mp-pitx.dtb to imx-mkimage/iMX8M directory and copy tools/mkimage to imx-mkimage/iMX8M/mkimage_uboot

```
$ cp u-boot-nodtb.bin spl/u-boot-spl.bin arch/arm/dts/imx8mp-pitx.dtb ../imx-mkimage/iMX8M/
$ cp tools/mkimage ../imx-mkimage/iMX8M/mkimage_uboot
```

5. Generate flash.bin file.

```
$ cd ../imx-mkimage
$ make CROSS_COMPILE=${CC} SOC=IMX8MP clean
$ make CROSS_COMPILE=${CC} SOC=IMX8MP flash_evk
```

The flash.bin file will be located at imx-mkimage/iMX8M directory. Go to "Setup SD Card" section to instruct you how to flash this file into microSD card.

Linux Kernel

Download:

For 5.15.71 (Based on NXP imx_1f-5.15.y official release):

```
$ git clone git@git.embedian.com:developer/smarc-fsl-linux-kernel.git v5.15.71 -b emb_imx_1f-5.15.y
```

Configure and Build:

```
$ make ARCH=arm CROSS_COMPILE=${CC} distclean
$ make ARCH=arm CROSS_COMPILE=${CC} emb_imx_v8_defconfig
$ make ARCH=arm CROSS_COMPILE=${CC} Image modules dtbs
```

Selecting display configuration is a matter of selecting an appropriate DTB file under `arch/arm64/boot/dts/embedian`.

All available DTB files are listed in the table below.

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

Root File System

Debian 11 Bullseys:

User	Password
root	root

Debian 11 Bullseye Download:

```
$ wget -c ftp://ftp.embedian.com/public/dev/minfs/debian/bulleyes/pitximx8mp-bulleys-arm64.tar.gz
```

Verify:

```
$ md5sum pitximx8mp-bulleys-arm64.tar.gz
a446a2a0191b64f895a56cf2dd6ee04c  pitximx8mp-bulleys-arm64.tar.gz
```

Yocto Kirkstone Build Root File System:

User	Password
root	N/A

Find the yocto pre-built root file systems here at [Embedian's ftp site](#) based on your module CPU variants.

```
$ wget -c ftp://ftp.embedian.com/public/test/fsl-image-qt6-validation-imx-pitximx8mp4g.tar.bz2
$ md5sum fsl-image-qt6-validation-imx-pitximx8mp4g.tar.bz2
16f7312faa60b3600f4e99cbb1ff7c5e  fsl-image-qt6-validation-imx-pitximx8mq4g.tar.bz2
```

Setup microSD Card

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:

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

```
sfdisk  
$ sudo sfdisk --version  
sfdisk from util-linux 2.34
```

Create Partitions:

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

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 SW2 port 1-3 and (ON ON ON). In this way, *piTX-iMX8M-PLUS* will always boot up from microSD card.

Fuse flash.bin to the microSD card.

~/imx-mkimage

```
$ sudo dd if=iMX8M/flash.bin of=${DISK} bs=1024 seek=32
```

If on-module eMMC Flash is not empty

The *flash.bin* is pre-installed in on-module eMMC flash at factory default. *p/ITX-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 *flash.bin* 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 (The *flash.bin* file is located at *imx-mkimage/iMX8M/* directory).

```
$ sudo cp -v imx-mkimage/iMX8M/flash.bin /media/rootfs/home/root/
```

Fuse *flash.bin* to the on-module eMMC flash. (The eMMC flash is emulated as */dev/mmcblk2* in *p/ITX-MX8M-PLUS*)

home directory

```
$ sudo dd if=flash.bin 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 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 *flash.bin* to eMMC flash.
2. When SW2 port1-3 on *p/ITX-MX8M-PLUS* is set as (OFF ON OFF), it will always boot up from on-module eMMC flash. U-boot will read the SW2 port 1-3 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=ttymx3,115200 earlycon=ec_imx6q,0x30a60000,115200
mmcdev=1
mmcpart=1
image=Image
loadaddr=0x40480000
fdt_addr=0x43000000
mmccroot=/dev/mmcblk1p2 rw
usbroot=/dev/sda2 rw
mmccrootfstype=ext4 rootwait fixrtc
netdev=eth0
ethact=FEC0
ipaddr=192.168.1.150
serverip=192.168.1.53
gatewayip=192.168.1.254
mmcbargs=setenv bootargs ${mcore_clk} console=${console} root=${mmccroot} rootfstype=${mmccrootfstype}
${optargs}
uenvcmd=run loadimage; run loadfdt; run mmcbboot
# USB Boot
#usbargs=setenv bootargs console=${console} root=${usbroot} rootfstype=${mmccrootfstype} ${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:

~/v5.15.71

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

Install Kernel Device Tree Binary

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

```
$ sudo cp -v arch/arm/boot/dts/<device tree file> /media/boot/dtbs
```

All available DTB files are listed in the table below.

DTB File Name	Description
<i>imx8mp-pitx.dtb</i>	Device tree blob for no display configuration.
<i>imx8mp-pitx-hdmi.dtb</i>	Device tree blob for HDMI display configuration.
<i>imx8mp-pitx-lvds.dtb</i>	Device tree blob for LVDS display configuration.
<i>imx8mp-pitx-m7.dtb</i>	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 and Kernel Modules

Copy Root File System:

Yocto Pre-Built Rootfs:

directory where your root file system is

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

Debian 11 Bulleyes:

directory where your root file system is

```
$ sudo tar xvfz imx8mp-bulleyes-arm64.tar.gz -C /media/rootfs
```

Copy Kernel Modules:

~/smarc-fsl-linux-kernel

```
$ sudo make ARCH=arm64 CROSS_COMPILE=${CC} INSTALL_MOD_PATH=/media/rootfs modules_install
```

**Note**

1. 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.
2. If your rootfs is yocto built, the kernel modules will be included in the rootfs.

Networking:

Edit: /etc/network/interfaces

```
$ sudo vim /media/rootfs/etc/network/interfaces
```

Add:

/media/rootfs/etc/network/interfaces

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp
```

Remove microSD card:

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

Setup eMMC

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 SD 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:

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



Note

1. If your rootfs is Debian Bullseyes, copy to [/media/rootfs/home/user](#) instead of [/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=ttymx3,115200 earlycon=ec_imx6q,0x30a60000,115200
mmcdev=2
mmcpart=1
image=Image
loadaddr=0x40480000
fdt_addr=0x43000000
mmccroot=/dev/mmcblk2p2 rw
usbroot=/dev/sda2 rw
mmccrootfstype=ext4 rootwait fixrtc
netdev=eth0
ethact=FEC0
ipaddr=192.168.1.150
serverip=192.168.1.53
gatewayip=192.168.1.254
mmccargs=setenv bootargs ${mcore_clk} console=${console} root=${mmccroot} rootfstype=${mmccrootfstype}
${optargs}
uenvcmd=run loadimage; run loadfdt; run mmccboot
# USB Boot
#usbargs=setenv bootargs console=${console} root=${usbroot} rootfstype=${mmccrootfstype} ${optargs}
#uenvcmd=run loadusbimage; run loadusbfdt; run usbboot
```

Copy device tree blob to rootfs partition:

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

Copy boot file to rootfs partition:

```
~/imx-mkimage
$ sudo cp -v iMX8M/flash.bin /media/rootfs/home/root/flash.bin
```

Copy real rootfs to rootfs partition:

Yocto Built Root File Systems

```
$ pushd /media/rootfs
```

```
$ sudo tar cvfz ~/pitximg8mp-emmc-rootfs.tar.gz .  
$ sudo mv ~/pitximg8mp-emmc-rootfs.tar.gz /media/rootfs/home/root  
$ popd
```

Debian Bullseyes Root File Systems

```
$ pushd /media/rootfs  
$ sudo tar cvfz ~/pitximg8mp-emmc-rootfs.tar.gz .  
$ sudo mv ~/pitximg8mp-emmc-rootfs.tar.gz /media/rootfs/home/root  
$ popd
```

Unmount microSD card.

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

It is safe to remove microSD card now.

Copy Binaries to eMMC from microSD card:

Insert this microSD card into your pITX-MX8M-PLUS device and boot into SD card.

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

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

Erase microSD card:

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

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 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
```

Flash boot file

```
$ sudo dd if=flash.bin of=${DISK} bs=1024 seek=32
```

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
```

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

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

```
root@pitximax8mp4g:~# 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.pl6-MXM5X16214-GPL-(FP92)
[ 35.966307] wlan: Driver loaded successfully
root@pitximax8mp4g:~#
```

Verify that the module is now visible to the system.

```
root@pitximax8mp4g:~# ifconfig -a
can0: flags=128<NOARP> mtu 16
unspec 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 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
```

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

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

Associate the Wi-Fi with config

[illegible]

```
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@pitxmx8mp4g:~#
```

[illegible]

```
root@pitxmx8mp4g:~# udhcpd -i mlan0
udhcpd: started, v1.32.0
udhcpd: sending discover
udhcpd: sending select for 192.168.1.57
udhcpd: lease of 192.168.1.57 obtained, lease time 86400
/etc/udhcpd.d/50default: Adding DNS 192.168.1.254
root@pitxmx8mp4g:~#
```

```
root@pitxlmx8mp4g:~# 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
```

```
root@pitxmx8mp4g:~# vim /etc/resolv.conf
```

```
nameserver 8.8.8.8
nameserver 8.8.4.4
```



```
root@pitxmx8mp4g:/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, 08/10/2023

Last updated 2023-08-10