SMARC-FiMX7-Standalone

Build and Install Linux System for *SMARC-FiMX7* (Solo and Dual 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 *SMARC-FiMX7* 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
```

Availability

SMARC-FiMX7 at Embedian

Carrier Board

SBC-SMART-BEE (module and carrier board) at Embedian

SBC-SMART-MEN (module and carrier board) at Embedian

EVK-STD-CARRIER-S20 (universal carrier board for all SMARC 1.1 and 2.0 modules) 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
  - 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 SMARC-FiMX7 Linux kernel source tree: git@git.embedian.com:developer/smarc-fsl-linux-kernel.git or git@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 zlib1g:i386
Ubuntu 20.10 -> 22.04		libc6:i386 libstdc++6:i386 libncurses5:i386 zlib1g: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 zlib1g:i386

To build Embedian's *SMARC-FiMX7* 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-arm-none-linux-gnueabihf.tar.xz

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

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

Test:

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

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

```
20191025
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

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 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.

Bootloader: U-Boot

Clone the U-Boot source code from Embedian Git Server.

Download:

For u-boot v2022.04:

```
$ git clone git@git.embedian.com:developer/smarc-t335x-uboot.git v2022.04 -b emb_lf_v2022.04
```

Configure and Build:

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

$ make ARCH=arm CROSS_COMPILE=${CC} smarcfimx7d_ser3_defconfig

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

Note
Note1:

If the board is SMARC-FiMX7-S (Solo Core), use
`$ make ARCH=arm CROSS_COMPILE=${CC} smarcfimx7s_ser3_defconfig`

If the board is SMARC-FiMX7-D-2G (Dual Core with 2GB DDR3L), use
`$ make ARCH=arm CROSS_COMPILE=${CC} smarcfimx7d_2g_ser3_defconfig`

Note 2:

"*ser3*" stands for console debug port. In this example, we uses SER3 as debug port. If user uses SER0 as your debug port, make change to "*ser0*" instead. Same as SER1 and SER2.

Note 3:

The *SMARC-FiMX7* module always boot up from the on-module *SPI NOR* flash. The factory default will be *u-boot.imx* pre-installed with SER3 as console output. In some cases when the *SPI NOR* flash is empty or needs to be upgraded. Users can shunt crossed the *TEST#* to ground. In this way, the *SMARC-FiMX7* module will boot up to carrier SD card, if *TEST#* pin is shunt crossed. The *u-boot.imx* image are the same, the difference is how you flash *u-boot.imx*. This will be explained in the "Setup SD card" section.

Linux Kernel

Download:

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

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

Configure and Build:

```
$ make ARCH=arm CROSS_COMPILE=${CC} distclean
$ make ARCH=arm CROSS_COMPILE=${CC} emb_imx_v7_defconfig
$ make ARCH=arm CROSS_COMPILE=${CC} zImage modules dtbs
```

All available DTB files are listed in the table below.

DTB Name	Description
imx7s-smarc.dtb	Device tree blob for i.mx7 solo core configuration.
imx7d-smarc-wvga.dtb	Device tree blob for i.mx7 dual core configuration.

Note1:

If the board is solo core, the device tree blob is imx7s-smarc.dtb

If the board is dual core, the device tree blob is imx7d-smarc.dtb

Root File System

Ubuntu 16.04:

User	Password
root	root
ubuntu	temppwd

Ubuntu 16.04 Download:

```
$ wget -c
ftp://ftp.embedian.com/public/dev/minfs/ubuntu/xenial/imx7-ubuntu-16.04.2-armhf-2017-03-02.tar.gz
```

Verify:

```
$ md5sum imx7-ubuntu-16.04.2-armhf-2017-03-02.tar.gz
4604f42c0525d6a5406bfed8ce61a892  imx7-ubuntu-16.04.2-armhf-2017-03-02.tar.gz
```

Debian 8.7:

User	Password
root	root
debian	temppwd

Debian 8 Download:

```
$ wget -c ftp://ftp.embedian.com/public/dev/minfs/debian/jessie/imx7-debian-8.7-armhf-2017-03-02.tar.gz
```

Verify:

```
$ md5sum imx7-debian-8.7-armhf-2017-03-02.tar.gz
beb77ef08400cb9f1780e8a80f47add6  imx7-debian-8.7-armhf-2017-03-02.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.

For dual core i.MX7,

```
$ wget -c ftp://ftp.embedian.com/public/test/fsl-image-qt6-validation-imx-smarcfimx7d.tar.bz2
$ md5sum fsl-image-qt6-validation-imx-smarcfimx7d.tar.bz2
dbb5a67698a150cc655f5d021e8fe4aa  fsl-image-qt6-validation-imx-smarcfimx7d.tar.bz2
```

For solo core i.MX7,

```
$ wget -c ftp://ftp.embedian.com/public/test/fsl-image-validation-imx-smarcfimx7s.tar.bz2
$ md5sum fsl-image-validation-imx-smarcfimx7s.tar.bz2
bae9b432e1c17dca72370191c1d8048a  fsl-image-validation-imx-smarcfimx7s.tar.bz2
```

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=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__
1M,48M,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 Bootloader

If SPI NOR Flash is not empty

The *u-boot.imx* is pre-installed in SPI NOR flash at factory default. SMARC-FiMX7 is designed to always boot up from SPI NOR flash and to load zImage, device tree blob and root file systems based on the setting of *BOOT_SEL*. If users need to fuse their own u-boot or perform u-boot upgrade. This section will instruct you how to do that.

Copy u-boot.imx to the boot partition. (Note: Rename u-boot-dtb.img to u-boot.img for u-boot is v2022.04)

~/v2022.04

```
$ sudo cp -v u-boot-dtb.imx /media/boot/u-boot.imx
```

Fuse u-boot.imx to the SPI NOR flash.

Stop at U-Boot command prompt (Press any key when booting up). Copy and Paste the following script under u-boot command prompt.

u-boot command prompt

```
U-Boot# mmc rescan; mmc dev; load mmc 0:1 0x90800000 u-boot.imx; sf probe; sleep 2; sf erase 0 0xc0000;
sf write 0x90800000 0x400 c0000
```

If SPI NOR Flash is empty

In some cases, when SPI NOR 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-FiMX7* will always boot up from SD card.

Copy u-boot.imx to the boot partition. (Note: Rename u-boot-dtb.img to u-boot.img for u-boot is v2022.04)

~/v2022.04

```
$ sudo dd if=u-boot.imx of=${DISK} bs=512 seek=2
```

⚠ 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 u-boot to SPI NOR flash.
2. When *TEST#* pin of SMARC-FiMX7 is not shunt crossed, it will always boot up from SPI NOR flash. U-boot will read the *BOOT_SEL* configuration and determine where it should load zImage 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

```
console=ttymxc2,115200
mmcdev=0
mmcpart=1
image=zImage
loadaddr=0x80800000
fdt_addr=0x83000000
mmcroot=/dev/mmcblk0p2 ro
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 loadzimage; run loadfdt; run mmcboot
```

Copy uEnv.txt to the boot partition:

~/

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

Install Kernel zImage

Copy zImage to the boot partition:

~/v5.15.71

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

Install Kernel Device Tree Binary

```
$ sudo mkdir -p /media/boot/dtbs
$ sudo cp -v arch/arm/boot/dts/imx7d-smarc.dtb arch/arm/boot/dts/imx7s-smarc /media/boot/dtbs
```

All available DTB files are listed in the table below.

DTB Name	Description
imx7s-smarc.dtb	Device tree blob for i.mx7 solo core configuration.
imx7d-smarc-wvga.dtb	Device tree blob for i.mx7 dual core configuration.

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
```

Ubuntu 16.04:

directory where your root file system is

```
$ sudo tar xvfz imx7-ubuntu-16.04.2-armhf-2017-03-02.tar.gz -C /media/rootfs
```

Copy Kernel Modules:

~/smarc-fsl-linux-kernel

```
$ sudo make ARCH=arm INSTALL_MOD_PATH=/media/rootfs modules_install
```

Note

1. After compiled u-boot, it will generated u-boot-dtb.imx and u-boot-dtb.bin. The only difference is IVT header that will tell i.MX7 internal ROM where to load u-boot. If the firmware in SPI flash need to be update or empty. Users could pull the *TEST#* pin on carrier board to ***low***. In this way, *SMARC-FiMX7* will boot up to SD card first. The u-boot we need to use now will be u-boot-dtb.imx. Please rename it as *u-boot.imx*. The command to copy u-boot.imx to SD card now is:
   $ sudo dd if=u-boot.imx of=${DISK} bs=512 seek=2
   In this case, user will only need to copy *uEnv.txt*, *zImage* and *device tree blob* to partition one of your boot device.
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. 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 SD 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 SD card or NFS environments. From

software point of view, eMMC is nothing but a non-removable SD card on board. For *SMARC-FiMX7*, the SD card is always emulated as /dev/mmcblk0 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 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 zImage to rootfs partition:

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

Note

1. If your rootfs is Ubuntu 16.04, copy to */media/rootfs/home/ubuntu* 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)

```
console=ttymxc2,115200
mmcdev=1
mmcpart=1
image=zImage
loadaddr=0x80800000
fdt_addr=0x83000000
mmcroot=/dev/mmcblk2p2 ro
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 loadzimage; run loadfdt; run mmcboot
```

Copy device tree blob to rootfs partition:

```
$ sudo cp -v /media/boot/dtbs/imx7d-smarc.dtb /media/rootfs/home/root/imx7d-smarc.dtb
$ sudo cp -v /media/boot/dtbs/imx7s-smarc.dtb /media/rootfs/home/root/imx7s-smarc.dtb
```

Copy real rootfs to rootfs partition:

Yocto Built Root File Systems

```
$ pushd /media/rootfs

$ sudo tar cvfz ~/smarcfimx7-emmc-rootfs.tar.gz .

$ sudo mv ~/smarcfimx7-emmc-rootfs.tar.gz /media/rootfs/home/root

$ popd
```

Ubuntu 16.04 Root File Systems

```
$ sudo vim /media/rootfs/etc/udev/rules.d/70-persistent-net.rules

Delete all contents starting with "SUBSYSTEM=="

$ pushd /media/rootfs

$ sudo tar cvfz ~/smarcfimx7-emmc-rootfs.tar.gz .

$ sudo mv ~/smarcfimx7-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-FiMX7 device and boot into SD card.

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

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

Erase SD card:

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

Create Partition Layout:

```
$ sudo sfdisk ${DISK} <<-__EOF__
1M,48M,0x83,*
,,,-
__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/zImage/*.dtb to the boot partition

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

Install Kernel Device Tree Binary

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

$ sudo cp imx7d-smarc.dtb imx7s-smarc.dtb /media/boot/dtbs
```

Install Root File System

```
$ sudo tar -zxvf smarcfimx7-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.

---

version 1.0a, 08/08/2023

Last updated 2023-08-08