# SMARC-FiMX6-BSP2

**On this page:**


- Building Freescale/Embedian's Yocto BSP Distribution
- Introduction
- 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
- Overview of the meta-smarcfimx6 Yocto Layer
- Setting Up the Tools and Build Environment
- Setup SD Card
  - Install Bootloader
  - uEnv.txt based bootscript
  - Install Kernel zImage
  - Install Kernel Device Tree Binary
- Install Root File System
  - Copy Root File System:
- Feed Packages
- Writing Bitbake Recipes
  - Example HelloWorld recipe using autotools
  - Example HelloWorld recipe using a single source file
- Setup eMMC
  - Prepare for eMMC binaries from SD card (or NFS):
  - Copy Binaries to eMMC from SD card:
  - Install binaries for partition 1
  - Install Kernel Device Tree Binary
- Install Root File System


## Building Freescale/Embedian's Yocto BSP Distribution


**Eric Lee**


version 1.0a, 3/23/2015

## Introduction

This document describes how Embedian builds a customized version of Freescale's i.MX6 official BSP 2.0 release for Embedian's *SMARC-FiMX6* 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-FiMX6* 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.MX6 ARM Cortext-A9 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-smarcfimx6 Yocto Layer

The supplied meta-emb-smarc Yocto compliant layer has the following organization:

```
.
|-- conf
|   |-- layer.conf
|   |-- site.conf
|   |-- machine
|       |-- imx6solosmarc.conf
|       |-- imx6u1gsmarc.conf
|       |-- imx6q1gsmarc.conf
|   |   `-- imx6q2gsmarc.conf
|-- README
|-- recipes-bsp
|   |-- u-boot
|       `-- u-boot-smarcfimx6_2015.04-smarcfimx6.bb
|-- recipes-connectivity
|   |-- lftp
|       `–lftp_4.6.3a.bb
|-- recipes-core
|   |-- busybox
|       `-- busybox_%.bbappend
|       |   |-- busybox
|       |   |   `-- defconfig
|   |-- init-ifupdown
|       `-- init-ifupdown_%.bbappend
|       |-- init-ifupdown
|       |   |-- interfaces
|           `-- init
|   |-- initscripts
|       `-- initscripts_%.bbappend
|       |-- arm
|       |   |-- aarch64
|           `-- mx6
|   |-- ncurses
|       `-- ncurses_5.9.bbappend
|   |-- sysvinit
|       `-- sysvinit-inittab_2.88dsf.bbappend
|       |   |-- sysvinit-inittab
|       |   |   |-- mx6
|       |   |   |   |-- rc_gpu.S
|       |   |   |   `-- rc_mxc.S
|-- recipes-fsl
|   |-- images
|   |   |-- imx6solosmarc-fsl-image-gui.bb
|   |   |-- imx6solosmarc-fsl-image-qt5.bb
|   |   |-- imx6u1gsmarc-fsl-image-gui.bb
|   |   |-- imx6u1gsmarc-fsl-image-qt5.bb
|   |   |-- imx6q1gsmarc-fsl-image-gui.bb
|   |   |-- imx6q1gsmarc-fsl-image-qt5.bb
|   |   |-- imx6q2gsmarc-fsl-image-gui.bb
|   |   `-- imx6q2gsmarc-fsl-image-qt5.bb
|-- recipes-devtools
|   |-- nodejs
|   |   |-- nodejs_0.10.11.bb
|   |   |-- nodejs_0.10.17.bb
|   |   |-- nodejs_0.10.4.bb
|   |   |-- nodejs_0.8.14.bb
|   |   `-- nodejs_0.8.21.bb
|-- recipes-kernel
|   |-- linux
|       |-- linux-smarcfimx6_4.1.15.bb
|       |-- linux-smarcfimx6_%.bbappend
|       `-- linux-smarcfimx6-4.1.15
```

```
|           `-- defconfig
|   |-- linux-libc-headers
|       |-- linux-libc-headers.in
|       `-- linux-libc-headers_4.1.bb
|   |-- kernel-module-imx-gpu-viv
|       `-- kernel-module-imx-gpu-viv_5.0.11.p8.3.bb
`-- recipes-support
|   |-- boost
|   |   |-- boost_1.53.0.bb
|   |   |-- boost.inc
|   |   `-- files
    `-- ntp
        |-- files
        |-- ntp_4.2.6p5.bb
        `-- ntp.inc
```

#### Notes on *meta-emb-smarc* layer content

`conf/machine/*`

This folder contains the machine definitions for the ***imx6solosmarc|imx6u1gsmarc|imx6q1gsmarc|imx6q2gsmarc*** platform and backup repository in Embedian. These select the associated kernel, kernel config, u-boot, u-boot config, and tar.gz image settings.

`recipes-bsp/u-boot/*`

This folder contains recipes used to build DAS U-boot for ***imx6solosmarc|imx6u1gsmarc|imx6q1gsmarc|imx6q2gsmarc*** platform.

`recipes-connectivity/busybox/*`

This folder remove telnetd from bysybox for ***imx6solosmarc|imx6u1gsmarc|imx6q1gsmarc|imx6q2gsmarc*** platform.

`recipes-fsl/images/*`

These recipes are used to create the final target images for the devices. When you run Bitbake one of these recipes would be specified. For example, to build the root file system for the ***imx6q1gsmarc*** platform:

```
MACHINE=imx6q1gsmarc bitbake -k imx6q1gsmarc-fsl-image-qt5
```

`recipes-core/init-ifupdown*`

This recipe is used to amend device network interfaces

`recipes-core/init-sysvinit*`

This recipe is used to amend device console interfaces

recipes-devtools/nodejs/*

These recipes build the Node.js Javascript server execution environment.

`recipes-kernel/linux/*`

Contains the recipes needed to build the ***imx6solosmarc|imx6u1gsmarc|imx6q1gsmarc|imx6q2gsmarc*** Linux kernels.

`recipes-support/boost/*`

Adds Boost to the images. Boost provides various C++ libraries that encourage cross-platform development.

`recipes-support/ntp/*`

Network time protocol support.

# Setting Up the Tools and Build Environment

To build the latest Freescale i.MX6 fsl-bsp-release, you first need an Ubuntu 14.04LTS 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 14.04LTS 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
```

⚠ 1. If built under Linux Mint 18.2 64-bit Virtual Box, use libcrytpo++9v5:i386 instead of libcrypto++9:i386.

2. 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.

You'll also need to change the default shell to bash from Ubuntu's default dash shell (select the <No> option):

```
$ sudo dpkg-reconfigure dash
```

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-fsl-bsp-release
$ cd ~/smarc-fsl-bsp-release
$ repo init -u git://git.freescale.com/imx/fsl-arm-yocto-bsp.git -b imx-4.1.15-1.0.0_ga
$ repo sync
```

Download the Embedian Yocto build script and meta layer.

```
$ wget
ftp://ftp.embedian.com/public/dev/minfs/smarc-fimx6-bsp-release/fsl-smarcfimx6-jethro-setup-release.sh
$ chmod 444 fsl-smarcfimx6-jethro-setup-release.sh
$ cd sources
$ git clone git@git.embedian.com:developer/meta-smarcfimx6.git
$ cd ~/smarc-fsl-bsp-release
$ MACHINE=imx6q1gsmarc source fsl-smarcfimx6-jethro-setup-release.sh -b imx6q1g-build-qt5fb -e fb
```

This script will create and bring you to ~/smarc-fsl-bsp-release/imx6q1g-build-qt5fb directory.

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

$ MACHINE= <*machine name*> source emb-setup-release.sh <build directory> -e <*fb/x11/dfb*>

1. <machine name> is *imx6q1gsmarc* if your SMARC-FiMX6 is dual or quad core and 1GB DDR3 memory.
   <machine name> is *imx6q2gsmarc* if your SMARC-FiMX6 is dual or quad core and 2GB DDR3 memory.
   <machine name> is *imx6u1gsmarc* if your SMARC-FiMX6 is dual lite core and 1GB DDR3 memory.
   <machine name> is *imx6solosmarc* if your SMARC-FiMX6 is solo core.
2. "*fb*" stands for the software GUI layer is on framebuffer directly.
   "*x11*" stands for the software GUI layer is on X11.
   "*dfb*" stand for the software GUI layer is on direct FB (DFB)

The default console debug port is *SER3.*

In this document, we will use *imx6q1gsmarc* 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=imx6q1gsmarc bitbake -k imx6q1gsmarc-fsl-image-qt5
or
$ MACHINE=imx6q1gsmarc bitbake -k imx6q1gsmarc-fsl-image-gui
```

**Note**
*imx6q1gsmarc-fsl-image-gui* provides a gui image without QT5.

*imx6q1gsmarc-fsl-image-qt5* provides a Qt5 image for X11 and FB backends.

If your machine name is *imx6u1gsmarc* and your gui image is without QT5 , the following command gives you as an example.

```
$ MACHINE=imx6u1gsmarc bitbake -k imx6u1gsmarc-fsl-image-gui
```

The first build takes time.

Once it done, you can find all required images under ~/smarc-fsl-bsp-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=imx6q1gsmarc bitbake tcpdump
$ MACHINE=imx6q1gsmarc bitbake package-index
```

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

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

This folder contains the binary images for the root file system and the Embedian *SMARC-FiMX6* specific version of the u-boot, zImage and device tree file. Specifically the images are:

`deploy/images/<machine name>/u-boot.imx`

This u-boot bootloader binary for *SMARC-FiMX6*

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

The kernel zImage for *SMARC-FiMX6*.

`deploy/images/<machine name>/zImage-imx6q-smarcfimx6.dtb`

The device tree binary file for *SMARC-FiMX6* Dual and Quad core.

deploy/images/<machine name>/zImage-imx6dl-smarcfimx6.dtb

The device tree binary file for *SMARC-FiMX6* Solo and Dual Lite core.

`deploy/images/<machine name>/imx6u1gsmarc-fsl-image-gui-imx6u1gsmarc.*`

Embedian root file system images for software development on Embedian's *SMARC-FiMX6* Dual Lite core platforms without QT5.

deploy/images/<machine name>/imx6u1gsmarc-fsl-image-qt5-imx6u1gsmarc.*

Embedian root file system images for software development on Embedian's *SMARC-FiMX6* Dual Lite core platforms with QT5.

`deploy/images/<machine name>/imx6q1gsmarc-fsl-image-gui-imx6q1gsmarc.*`

Embedian root file system images for software development on Embedian's *SMARC-FiMX6* Dual and Quad core platforms without QT5.

deploy/images/<machine name>/imx6q1gsmarc-fsl-image-qt5-imx6q1gsmarc.*

Embedian root file system images for software development on Embedian's *SMARC-FiMX6* Dual and Quad core platforms with QT5.

`deploy/images/<machine name>/imx6solosmarc-fsl-image-gui-imx6solosmarc.*`

Embedian root file system images for software development on Embedian's *SMARC-FiMX6* Solo core platforms without QT5.

deploy/images/<machine name>/imx6solosmarc-fsl-image-qt5-imx6solosmarc.*

Embedian root file system images for software development on Embedian's *SMARC-FiMX6* Solo core platforms with QT5.

deploy/images/<machine name>/imx6q2gsmarc-fsl-image-gui-imx6q2gsmarc.*

Embedian root file system images for software development on Embedian's *SMARC-FiMX6* Dual and Quad core platforms and 2GB DDR3 without QT5.

deploy/images/<machine name>/imx6q2gsmarc-fsl-image-qt5-imx6q2gsmarc.*

Embedian root file system images for software development on Embedian's *SMARC-FiMX6* Dual and Quad core platforms and 2GB DDR3 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 *imx6q1g_qt5fb_update.repo*) that is used internally at Embedian to install upgrades onto a ***imx6q1gsmarc*** QT5 platform directly on framebuffer without reflashing the file system:

```
[all]
type = rpm-md
baseurl = http://www.embedian.com/smarcfimx6-yocto-feed/imx6q1gsmarc/fb/qt5/all
[cortexa9hf_vfp_neon]
type = rpm-md
baseurl = http://www.embedian.com/smarcfimx6-yocto-feed/imx6q1gsmarc/fb/qt5/cortexa9hf_vfp_neon
[cortexa9hf_vfp_neon_mx6]
type = rpm-md
baseurl = http://www.embedian.com/smarcfimx6-yocto-feed/imx6q1gsmarc/fb/qt5/cortexa9hf_vfp_neon_mx6
[imx6q1gsmarc]
type = rpm-md
baseurl = http://www.embedian.com/smarcfimx6-yocto-feed/imx6q1gsmarc/fb/qt5/imx6q1gsmarc
```

`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 Partitions:

**sfdisk >=2.26.x**

```
$ sudo sfdisk ${DISK} <<-__EOF__
1M,48M,0x83,*
,,,-
__EOF__
```

**sfdisk <=2.25**

```
$ sudo sfdisk --in-order --Linux --unit M ${DISK} <<-__EOF__
1,48,0x83,*
,,,-
__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 Bootloader

**If SPI NOR Flash is not empty**

The *u-boot.imx* is pre-installed in SPI NOR flash at factory default. SMARC-FiMX6 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.

**~/smarc-fs-bsp-release/<build dir>/tmp/deploy/images/<machine name>/**

```
$ sudo cp -v u-boot.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 0x10800000 u-boot.imx; sf probe; sleep 2; sf erase 0 0xc0000;
sf write 0x10800000 0x400 86000
```

**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-FiMX6* will always boot up from SD card.

Copy u-boot.imx to the boot partition

**~/smarc-fs-bsp-release/<build dir>/tmp/deploy/images/<machine name>/**

```
$ 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-FiMX6 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**

```
#####HDMI######
#optargs="video=mxcfb0:dev=hdmi,1280x720M@60,if=RGB24,bpp=32 consoleblank=0"
#####LVDS#####
#optargs="video=mxcfb0:dev=ldb,if=RGB24,bpp=32 consoleblank=0 fbmem=24M vmalloc=400M"
#####Parallel LCD Setting#####
#optargs="video=mxcfb0:dev=lcd,CLAA-WVGA,if=RGB24,bpp=32 consoleblank=0 fbmem=24M vmalloc=400M"
#####Parallel LCD to CH7055A (VESA Timing Format) Setting #####
#optargs="video=mxcfb0:dev=lcd,768x576M@75,if=RGB24,bpp=32 consoleblank=0"
#optargs="video=mxcfb0:dev=lcd,1280x1024M@60,if=RGB24,bpp=32 consoleblank=0"
#optargs="video=mxcfb0:dev=lcd,640x480M@60,if=RGB24,bpp=32 consoleblank=0"

console=ttymxc4,115200
mmcdev=0
mmcpart=1
image=zImage
loadaddr=0x12000000
fdt_addr=0x18000000
mmcroot=/dev/mmcblk1p2 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:

**~/smarc-fs-bsp-release/<build dir>/tmp/deploy/images/<machine name>/**

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

### Install Kernel Device Tree Binary

**~/smarc-fs-bsp-release/<build dir>/tmp/deploy/images/<machine name>/**

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

$ sudo cp -v zImage-imx6q-smarcfimx6.dtb /media/boot/dtbs/imx6q-smarcfimx6.dtb

$ sudo cp -v zImage-imx6dl-smarcfimx6.dtb /media/boot/dtbs/imx6dl-smarcfimx6.dtb
```

**Note, ~/smarc-fs-bsp-release/<build dir>/tmp/deploy/images/<machine name>**

1. If you are using LVDS panel, copy the corresponding device tree blob into SD card as follows.

***For WVGA (800x480) LVDS panel:***

```
$ sudo cp -v zImage-imx6q-smarcfimx6-wvga.dtb /media/boot/dtbs/imx6q-smarcfimx6.dtb
$ sudo cp -v zImage-imx6dl-smarcfimx6-wvga.dtb /media/boot/dtbs/imx6dl-smarcfimx6.dtb
```

***For XGA (1024x768) LVDS panel:***

```
 $ sudo cp -v zImage-imx6q-smarcfimx6-xga.dtb /media/boot/dtbs/imx6q-smarcfimx6.dtb
 $ sudo cp -v zImage-imx6dl-smarcfimx6-xga.dtb /media/boot/dtbs/imx6dl-smarcfimx6.dtb
```

***For WXGA (1366x768) LVDS panel:***

```
 $ sudo cp -v zImage-imx6q-smarcfimx6-wxga.dtb /media/boot/dtbs/imx6q-smarcfimx6.dtb
 $ sudo cp -v zImage-imx6dl-smarcfimx6-wxga.dtb /media/boot/dtbs/imx6dl-smarcfimx6.dtb
```

***For 1080p (1920x1080) LVDS panel:***

```
 $ sudo cp -v zImage-imx6q-smarcfimx6-1080p.dtb /media/boot/dtbs/imx6q-smarcfimx6.dtb
 $ sudo cp -v zImage-imx6dl-smarcfimx6-1080p.dtb /media/boot/dtbs/imx6dl-smarcfimx6.dtb
```

# Install Root File System

## Copy Root File System:

**Yocto Built Rootfs:**

**~/smarc-fs-bsp-release/<build dir>/tmp/deploy/images/<machine name>/**

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

**Note**

1. *SMARC-FiMX6* always boots up to SPI flash first. The firmware in SPI 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 u-boot.bin from the partition one of the device (could be SD card, eMMC, SATA,..etc) that you selected to memory.
2. Embedian Yocto BSP u-boot will only generated u-boot.bin. If people are interested in knowing u-boot.imx. Please see Linux

on SMARC-FiMX6 for details.
3. 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.
4. 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-FiMX6* 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/smarcfimx6-yocto-feed/imx6q1g_qt5fb_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/cortexa9hf-vfp-neon-poky-linux-gnueabi/<package name and version>` for most packages. If the package is machine-specific (rather than generic for the cortexa9hf architecture), it may be located in the imx6q1gsmarc-poky-linux-gnueabi 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-FiMX6*, the SD card is always emulated as /dev/mmcblk1 and on-module eMMC is always emulated as /dev/mmcblk3. 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:**

**~/smarc-fs-bsp-release/<build dir>/tmp/deploy/images/<machine name>**

```
$ sudo cp -v zImage /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)

```
#####HDMI######
#optargs="video=mxcfb0:dev=hdmi,1280x720M@60,if=RGB24,bpp=32 consoleblank=0"
#####LVDS#####
#optargs="video=mxcfb0:dev=ldb,if=RGB24,bpp=32 consoleblank=0 fbmem=24M vmalloc=400M"
#####Parallel LCD Setting#####
#optargs="video=mxcfb0:dev=lcd,CLAA-WVGA,if=RGB24,bpp=32 consoleblank=0 fbmem=24M vmalloc=400M"
#####Parallel LCD to CH7055A (VESA Timing Format) Setting #####
#optargs="video=mxcfb0:dev=lcd,768x576M@75,if=RGB24,bpp=32 consoleblank=0"
#optargs="video=mxcfb0:dev=lcd,1280x1024M@60,if=RGB24,bpp=32 consoleblank=0"
#optargs="video=mxcfb0:dev=lcd,640x480M@60,if=RGB24,bpp=32 consoleblank=0"

console=ttymxc4,115200
mmcdev=2
mmcpart=1
image=zImage
loadaddr=0x12000000
fdt_addr=0x18000000
mmcroot=/dev/mmcblk3p2 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:**

**~/smarc-fs-bsp-release/<build dir>/tmp/deploy/images/<machine name>**

```
$ sudo cp -v zImage-mx6q-smarcfimx6.dtb /media/rootfs/home/root/imx6q-smarcfimx6.dtb


$ sudo cp -v zImage-imx6dl-smarcfimx6.dtb /media/rootfs/home/root/imx6dl-smarcfimx6.dtb
```

⚠ **Note, ~/smarc-fs-bsp-release/<build dir>/tmp/deploy/images/<machine name>**

1. If you are using LVDS panel, copy the corresponding device tree blob into SD card as follows.

***For WVGA (800x480) LVDS panel:***

```
$ sudo cp -v zImage-imx6q-smarcfimx6-wvga.dtb /media/boot/dtbs/imx6q-smarcfimx6.dtb
$ sudo cp -v zImage-imx6dl-smarcfimx6-wvga.dtb /media/boot/dtbs/imx6dl-smarcfimx6.dtb
```

***For XGA (1024x768) LVDS panel:***

```
$ sudo cp -v zImage-imx6q-smarcfimx6-xga.dtb /media/boot/dtbs/imx6q-smarcfimx6.dtb
$ sudo cp -v zImage-imx6dl-smarcfimx6-xga.dtb /media/boot/dtbs/imx6dl-smarcfimx6.dtb
```

***For WXGA (1366x768) LVDS panel:***

```
$ sudo cp -v zImage-imx6q-smarcfimx6-wxga.dtb /media/boot/dtbs/imx6q-smarcfimx6.dtb
$ sudo cp -v zImage-imx6dl-smarcfimx6-wxga.dtb /media/boot/dtbs/imx6dl-smarcfimx6.dtb
```

***For 1080p (1920x1080) LVDS panel:***

```
$ sudo cp -v zImage-imx6q-smarcfimx6-1080p.dtb /media/boot/dtbs/imx6q-smarcfimx6.dtb
$ sudo cp -v zImage-imx6dl-smarcfimx6-1080p.dtb /media/boot/dtbs/imx6dl-smarcfimx6.dtb
```

-

**Copy real rootfs to rootfs partition:**

```
$ pushd /media/rootfs

$ sudo tar cvfz ~/smarcfimx6-emmc-rootfs.tar.gz .

$ sudo mv ~/smarcfimx6-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-FiMX6 device.

Now it will be almost the same as you did when setup your SD card, but the eMMC device descriptor is */dev/mmcblk3* now.

```
$ export DISK=/dev/mmcblk3
```

Erase SD card:

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

Create Partition Layout:

```
$ sudo sfdisk --in-order --Linux --unit M ${DISK} <<-__EOF__
1,48,0x83,*
,,,-
__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 -v imx6q-smarcfimx6.dtb imx6dl-smarcfimx6.dtb /media/boot/dtbs
```

## Install Root File System

```
$ sudo tar -zxvf smarcfimx6-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, 4/12/2016

Last updated 2016-04-12