SMARC T4378 ProcessorSDK02000107

On this page:

Building TI/Embedian's Yocto BSP Distribution

Eric Lee

version 1.0a, 1/26/2016

Introduction

This document describes how Embedian builds a customized version of TI's AM437X official BSP release for Embedian's *SMARC-T4378* 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-T4378* specific development.

TI makes their Processor-SDK-02.00.01.07 Arago build scripts available via the following GIT repository:

```
http://arago-project.org/git/projects/oe-layersetup.git
```

If you're interested in TI's overall Processor SDK build and test process you should analyze the following repository:

```
http://arago-project.org/git/projects/tisdk-build-scripts.git
```

It is this repository that actually pulls in the oe-layersetup project to perform the Linux Processor-SDK builds for TI's entire suite of ARM CortextA chips. In this document we are only concerned with the oe-layersetup project.

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-smarct437x-sdk-02.00.01.07 Yocto Layer

The supplied meta-embedian-sdk7 Yocto compliant layer has the following organization:

```
.
|-- conf
|   |-- layer.conf
|   |-- site.conf
|   |-- machine
|   |   `-- smarct437x.conf
|-- README
|-- recipes-bsp
|   |-- u-boot
|       `-- u-boot_2015.07-smarct437x.bb
|-- recipes-connectivity
|   |-- lftp
|       `–lftp_4.6.3a.bb
|-- recipes-core
|   |-- base-files
|       `-- base-files_%.bbappend
|   |-- busybox
|       |-- busybox_1.23.1.bbappend
|       |   |-- busybox
|       |   |   `-- defconfig
|   |-- images
|   |   |-- arago-base-smarct437x-image.bb
|   |   |-- arago-image.inc
|   |   |-- meta-toolchain-smarc-tisdk.bb
|   |   |-- smarct437x-initramfs-image.bb
|   |   `-- smarct437x-rootfs-image.bb
|   |-- init-ifupdown
|       |-- init-ifupdown_%.bbappend
|       |   |-- init-ifupdown
|       |   |   `-- interfaces
|   |-- initscripts
|   |   `-- initscripts_1.0.bbappend
|   `-- packagegroups
|       |-- packagegroup-arago-smarct437x-addons.bb
|       |-- packagegroup-arago-smarct437x-base.bb
|       |-- packagegroup-arago-smarct437x-console.bb
|       |-- packagegroup-arago-smarct437x-matrix.bb
|       |-- packagegroup-arago-smarct437x-sdk.bb
|       `-- packagegroup-initramfs-boot.bb
|-- recipes-devtools
|   |-- cloud9
|   |   |-- cloud9_0,6.bb
|   |   |-- cloud9
|   |   |   `-- cloud9-avahi.service
|   |-- 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
|   |-- ltp-ddt
|   |   |-- ltp-ddt_1.0.0.bbappend
|   |   |-- ltp-ddt
|           `-- 0001-add-smarct437x-platform-support.patch
|   |-- pinmux-utility
```

```
|   |   `-- pinmux-utility_2.5.2.0.bbappend
|-- recipes-kernel
|   |-- linux
|       |-- cmem.inc
|       |-- linux-dtb.inc
|       |-- copy-defconfig.inc
|       |-- setup-defconfig.inc
|       |-- linux-smarct437x-staging_4.1.bb
|       `-- linux-smarct437x-staging-4.1
|       |   |-- defconfig
|       |   |-- configs
|       |       `-- systest
|   |-- cryptodev
|       |-- cryptodev-module_1.6.bbappend
|       `-- cryptodev_1.6.bbappend
|-- recipes-support
|   |-- boost
|   |   |-- boost_1.53.0.bb
|   |   |-- boost.inc
|   |   `-- files
|   |-- ntp
|   |   |-- files
|   |   |-- ntp_4.2.6p5.bb
|   |   `-- ntp.inc
`-- recipes-tisdk
    `-- ti-tisdk-makefile
        `-- ti-tisdk-makefile_1.0.bbappend
```

Notes on ***meta-smarct437x-sdk-02.00.01.07*** layer content

`conf/machine/*`

This folder contains the machine definitions for the ***smarct437x*** platform and backup repository in Embedian. These select the associated kernel, kernel config, u-boot, u-boot config, and UBI image settings.

`recipes-bsp/u-boot/*`

This folder contains recipes used to build DAS U-boot for ***smarct437x*** platform.

`recipes-connectivity/lftp/*`

This folder adds lftp ftp client utility for ***smarct437x*** platform.

`recipes-core/base-files/*`

This recipe is used to amend the device hostname for the platform.

`recipes-core/busybox/*`

This recipe modifies TI's BusyBox configuration to remove telnet from the image.

`recipes-core/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 ***smarct437x*** platform:

```
  MACHINE=smarct437x bitbake -k smarct437x-rootfs-image
```

`recipes-core/init-ifupdown*`

This recipe is used to amend device network interfaces

`recipes-devtools/nodejs/*`

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

`recipes-kernel/linux/*`

Contains the recipes needed to build the ***smarct437x*** 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.

`recipes-tisdk/ti-tisdk-makefile/*`

Add smarct437x device tree into Makefile.

Setting Up the Tools and Build Environment

To build the latest TI AM437X Processor-SDK-02.00.01.07, you first need an 64-bit Unbuntu Linux 12.04LTS or Ubuntu 14.04LTS installation because of support for 32-bit host is dropped as Linaro toolchain is available only for 64-bit machines. A x86_64 ubuntu 14.04 is highly recommended. 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 12.04 LTS or 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 dos2unix
```

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

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


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

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

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

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

if output is

```
multiarch
```

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

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

The above command will add i386 architecture.

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 build TI's am437x Processor-SDK-02.00.01.07 you will need to install the Linaro arm compiler that TI used for the release:

```
$ wget -c https://releases.linaro.org/15.05/components/toolchain/binaries/arm-linux-gnueabihf/gcc-linaro
-4.9-2015.05-x86_64_arm-linux-gnueabihf.tar.xz
$ sudo tar -C /opt -xJf gcc-linaro-4.9-2015.05-x86_64_arm-linux-gnueabihf.tar.xz
```

Add the following PATH definition to the `.bashrc` file in your `$HOME` directory:

```
$ export PATH=/opt/gcc-linaro-4.9-2015.05-x86_64_arm-linux-gnueabihf/bin:$PATH
```

Next clone and intialize TI's am437x SDK build process:

```
$ git clone http://arago-project.org/git/projects/oe-layersetup.git smarct4x-processor-sdk-02.00.01.07
$ cd smarct4x-processor-sdk-02.00.01.07
$ git checkout -b smarct4x-processor-sdk-02.00.01.07 4cd35d71756a59b6d538d774ae6b359b2aba1294
$ ./oe-layertool-setup.sh -f configs/processor-sdk/processor-sdk-02.00.01.07-config.txt
```

Add the Embedian's *`meta-smarct437x-sdk-02.00.01.07`* layer to the build process.

```
$ cd ~/smarct4x-processor-sdk-02.00.01.07/sources
$ git clone git@git.embedian.com:developer/meta-smarct437x-sdk-02-00-01-07.git
$ cd ~/smarct4x-processor-sdk-02.00.01.07/build
```

Edit the ~/smarct4x-processor-sdk-02.00.01.07/build/conf/bblayers.conf file to include the *meta-smarct437x-sdk-02-00-01-07* layer in the layer list. It should look something like this (the example reflects the absolute paths on my machine):

```
# This template file was created by taking the oe-core/meta/conf/bblayers.conf

# file and removing the BBLAYERS section at the end.

# Layers configured by oe-core-setup script
BBLAYERS += " \
/home/linux/smarct4x-processor-sdk-02.00.01.07/sources/meta-smarct437x-sdk-02-00-01-07 \
/home/linux/smarct4x-processor-sdk-02.00.01.07/sources/meta-processor-sdk \
/home/linux/smarct4x-processor-sdk-02.00.01.07/sources/meta-arago/meta-arago-distro \
/home/linux/smarct4x-processor-sdk-02.00.01.07/sources/meta-arago/meta-arago-extras \
/home/linux/smarct4x-processor-sdk-02.00.01.07/sources/meta-qt5 \
/home/linux/smarct4x-processor-sdk-02.00.01.07/sources/meta-openembedded/meta-networking \
/home/linux/smarct4x-processor-sdk-02.00.01.07/sources/meta-openembedded/meta-ruby \
/home/linux/smarct4x-processor-sdk-02.00.01.07/sources/meta-openembedded/meta-python \
/home/linux/smarct4x-processor-sdk-02.00.01.07/sources/meta-openembedded/meta-oe \
/home/linux/smarct4x-processor-sdk-02.00.01.07/sources/meta-ti \
/home/linux/smarct4x-processor-sdk-02.00.01.07/sources/meta-linaro/meta-linaro-toolchain \
/home/linux/smarct4x-processor-sdk-02.00.01.07/sources/oe-core/meta \
"
```

Note:
The default kernel is for SMARC-T4378-01G. If you are using SMARC-T4378-800, modify the kernel recipe under *sources/recipes-kernel/linux/linux-smarct437x-staging_4.1.bb* as follows.

```
# For 4.1y Kernel and running at 1Ghz
# BRANCH = "smarct4x-processor-sdk-linux-02.00.01"
# For 4.1y Kernel and running at 800MHz
BRANCH = "smarct4x-800-processor-sdk-linux-02.00.01"

# Corresponds to tag smarct4x-processor-sdk-linux-02.00.01, CPU running at 1Ghz
# SRCREV = "9a2fcea53700bf118fc7d98a9db75131f404c5d0"
# Corresponds to tag smarct4x-800-processor-sdk-linux-02.00.01, CPU running at 800Mhz
SRCREV = "966328fc05582f1e092ec6688445bbc4ea5b0b90"
```

Building the target platforms

To build the Embedian SMARC-T437X developer board images, respectively, use the following commands:

```
$ cd ~/smarct4x-processor-sdk-02.00.01.07/build
$ source conf/setenv
$ MACHINE=smarct437x bitbake -k smarct437x-rootfs-image
```

Note

1. The first clean build might take more than 10 hours. If you met errors during the building process, let it finish and usually build again should be fine.
2. During the very first build, if you saw the following error.
   ERROR: Task 1978 (/home/linux/smarct4x-processor-sdk-02.00.01.07/sources/meta-qt5/recipes-qt/qt5/qtmultimedia_git.bb, do_compile) failed with exit code '1'

Simply run $ MACHINE=smarct437x bitbake -k smarct437x-rootfs-image again

Once it done, you can find all required images under ~/smarct4x-processor-sdk-02.00.01.07/build/arago-tmp-external-linaro-toolchain/deploy/images/smarct437x/

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=smarct437x bitbake minicom
$ MACHINE=smarct437x bitbake package-index
```

Once the build(s) are completed you'll find the resulting images, feeds and licenses in folder `~/smarct4x-processor-sdk-02.00.01.07/build/arago-tmp-external-linaro-toolchain/deploy/`

`deploy/images/smarct437x/*`

This folder contains the binary images for the root file system and the Embedian SMARC-T437X specific version of the am437x SDK. Specifically the images are:

`deploy/images/smarct437x/u-boot.img`

This u-boot bootloader binary for SMARC T437X

`deploy/images/smarct437x/MLO.byteswap`

The "Stage 1 SPI NOR flash Boot Loader" for SMARC-T437X. Its purpose is load the Stage 2 Boot Loader (u-boot.img).

`deploy/images/smarct437x/zImage`

The kernel zImage for SMARC-T437X.

`deploy/images/smarct437x/zImage-am437x-smarct437x.dtb`

The device tree binary file for SMARC-T437X.

`deploy/images/smarct437x/smarct437x-rootfs-image-smarct437x*`

Embedian root file system images for software development on Embedian's SMARC-T437X platforms.

`deploy/ipk/*`

This folder contains all the packages used to construct the root file system images. They are in ***opkg*** 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 ***smarct437x*** platform without reflashing the file system:

```
src/gz smarct335x http://www.embedian.com/core-sdk/smarct437x/processor-sdk-02.00.00.00/deploy/ipk/all

src/gz cortexa9hf-vfp-neon http://www.embedian.com/core-sdk/smarct437x/smarct4x-processor-sdk-02.00.00.00/deploy/ipk/cortexa9hf-vfp-neon
```

```
src/gz smarct437x http://www.embedian.com/core-sdk/smarct437x/smarct4x-processor-sdk-02.00.00.00/deploy/ipk/smarct437x
```

`deploy/licenses/*`
: A database of all licenses used in all packages built for the system.

`deploy/sdk/arago-2015.12-cortexa9-linux-gnueabi-tisdk.sh`
: The installer for ARM toolchain that was created for the target platform. In Embedian's case that means that the headers for the Boost libraries are baked into the tools. (Generate by meta-toolchain-smarc-tisdk image)

Setup SD Card

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

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

Erase SD card:

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

Create Partition Layout:

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__
1M,48M,0xE,*
,,,-
__EOF__
```

sfdisk <=2.25
```
$ sudo sfdisk --in-order --Linux --unit M ${DISK} <<-__EOF__
1,48,0xE,*
,,,-
__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 MLO.byteswap and u-boot.img is pre-installed in SPI NOR flash at factory default. SMARC-T4378 is designed to always boot upi from SPI NOR flash and to load zImage 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 MLO.byteswap/u-boot.img to the boot partition.

build/arago-tmp-external-linaro-toolchain/deploy/images/smarct437x

```
$ sudo cp -v MLO.byteswap /media/boot/
$ sudo cp -v u-boot.img /media/boot/spi-u-boot.img
```

Fuse MLO.byteswap/u-boot.img to the SPI NOR flash.

Stop at U-Boot command prompt (Press any key when booting up).

u-boot command prompt

```
U-Boot# sf probe
U-Boot# sf erase 0 400000
U-Boot# mmc rescan
U-Boot# fatload mmc 0 ${loadaddr} MLO.byteswap
U-Boot# sf write ${loadaddr} 0 ${filesize}
U-Boot# fatload mmc 0 ${loadaddr} spi-u-boot.img
U-Boot# sf write ${loadaddr} 0x20000 ${filesize}
```

⚠ MLO.byteswap and u-boot.img are from smarct437x_evm_spi_uart3_defconfig

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-T437X will always boot up from SD card.

Copy MLO/u-boot.img to the boot partition

build/arago-tmp-external-linaro-toolchain/deploy/images/smarct437x

```
$ sudo cp -v MLO /media/boot/
$ sudo cp -v u-boot.img /media/boot/
```

⚠ *MLO* and *u-boot.img* will be from *smarct437x_evm_uart3_defconfig*. Modify the config file under *sources/meta-smarct437x-sdk-02-00-01-07/conf/machine/smarct437x.conf*

UBOOT_MACHINE = "smarct437x_evm_spi_uart3_defconfig"
# If Test# pin is shunt
# UBOOT_MACHINE = "smarct437x_evm_uart3_defconfig"

Also modify *sources/meta-smarct437x-sdk-02-00-01-07/recipes-bsp/u-boot/u-boot-smarct437x_2015.07-smarct437x.bb*

If TEST# pin is shunt

```
# SPL_BINARY = "MLO"
SPL_BINARY = "MLO.byteswap"
```

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 make smarct437x_evm_spi_uart3_defconfig again fuse your u-boot to SPI NOR flash.

uEnv.txt based bootscript

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

~/uEnv.txt

```
optargs="consoleblank=0 mem=512M"
#u-boot eMMC specific overrides; Angstrom Distribution (SMARC-T437X) 2014-05-20
kernel_file=zImage
initrd_file=initrd.img

loadaddr=0x82000000
initrd_addr=0x88080000
fdtaddr=0x88000000
fdtfile=am437x-smarct437x.dtb

initrd_high=0xffffffff
fdt_high=0xffffffff

loadimage=load mmc ${mmcdev}:${mmcpart} ${loadaddr} ${kernel_file}
loadinitrd=load mmc ${mmcdev}:${mmcpart} ${initrd_addr} ${initrd_file}; setenv initrd_size ${filesize}
loadfdt=load mmc ${mmcdev}:${mmcpart} ${fdtaddr} /dtbs/${fdtfile}
#

##Un-comment to enable systemd in Debian Wheezy
#optargs=quiet init=/lib/systemd/systemd

console=ttyS4,115200n8
mmcroot=/dev/mmcblk1p2 ro
mmcrootfstype=ext4 rootwait fixrtc

mmcargs=setenv bootargs console=${console} root=${mmcroot} rootfstype=${mmcrootfstype} ${optargs}

#zImage:
uenvcmd=run loadimage; run loadfdt; run mmcargs; bootz ${loadaddr} - ${fdtaddr}

#zImage + uInitrd: where uInitrd has to be generated on the running system.
#boot_fdt=run loadimage; run loadinitrd; run loadfdt
#uenvcmd=run boot_fdt; run mmcargs; bootz ${loadaddr} ${initrd_addr}:${initrd_size} ${fdtaddr}

###Begin Rootfs from NFS
#serverip=192.168.1.51
#rootpath=/srv/nfs/smarct335x/ubuntu1204/
#nfsopts=nolock,acdirmin=60
#netargs=setenv bootargs console=${console} ${optargs} root=/dev/nfs nfsroot=${serverip}:${rootpath},${nfsopts} rw ip=dhcp
##netboot=echo Loading kernel from SDCARD and booting from NFS ...; run loadimage; run netargs; bootz ${loadaddr} - ${fdtaddr}
##uenvcmd=run netboot
###End Rootfs from NFS

###Begin Load kernel from TFTP
#netmask=255.255.255.0
#ipaddr=192.168.1.65
#serverip=192.168.1.51
#netboot=echo Loading kernel and device tree from TFTP and booting from NFS ...; setenv autoload no; tftp ${loadaddr} ${kernel_file}; tftp
${fdtaddr} ${fdtfile}; run netargs; bootz ${loadaddr} - ${fdtaddr}
#uenvcmd=run netboot
###End Load kernel from TFTP
```


1. If you use SMARC-T4378-800, mem=512M in optargs. Otherwise, mem has to change to 1024M in optargs.
2. mmcroot=/dev/mmcblk1p2 when SDMMC eMMC is not present on carrier board. If there is an eMMC present on carrier board. SD card will be emulated as /dev/mmcblk2 and mmcroot=/dev/mmcblk2p2.

Install Kernel zImage

Copy zImage to the boot partition:

build/arago-tmp-external-linaro-toolchain/deploy/images/smarct437x

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

Install Kernel Device Tree Binary

build/arago-tmp-external-linaro-toolchain/deploy/images/smarct437x

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

$ sudo cp -v zImage-am437x-smarct437x.dtb /media/boot/dtbs/am437x-smarct437x.dtb
```

Install Root File System and Kernel Modules

build/arago-tmp-external-linaro-toolchain/deploy/images/smarct437x

```
$ sudo tar xvfz smarct437x-rootfs-image-smarct437x.tar.gz -C /media/rootfs
```

⚠ Kernel modules are built into root filesystems.

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. eMMC on module will be always emulated as /dev/mmcblk0. Setting up eMMC now is nothing but changing the device descriptor.

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

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

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

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

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

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

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__
1,48,0xE,*
,,,-
__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/
```

Copy zImage to rootfs partition:

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

~/uEnv.txt

```
optargs="consoleblank=0 mem=512M"
#u-boot eMMC specific overrides; Angstrom Distribution (SMARC-T437X) 2014-05-20
kernel_file=zImage
initrd_file=initrd.img

loadaddr=0x82000000
initrd_addr=0x88080000
fdtaddr=0x88000000
fdtfile=am437x-smarct437x.dtb

initrd_high=0xffffffff
fdt_high=0xffffffff

loadimage=load mmc ${mmcdev}:${mmcpart} ${loadaddr} ${kernel_file}
loadinitrd=load mmc ${mmcdev}:${mmcpart} ${initrd_addr} ${initrd_file}; setenv initrd_size ${filesize}
loadfdt=load mmc ${mmcdev}:${mmcpart} ${fdtaddr} /dtbs/${fdtfile}
#

##Un-comment to enable systemd in Debian Wheezy
#optargs=quiet init=/lib/systemd/systemd

console=ttyS4,115200n8
mmcroot=/dev/mmcblk0p2 ro
mmcrootfstype=ext4 rootwait fixrtc
```

```
mmcargs=setenv bootargs console=${console} root=${mmcroot} rootfstype=${mmcrootfstype} ${optargs}

#zImage:
uenvcmd=run loadimage; run loadfdt; run mmcargs; bootz ${loadaddr} - ${fdtaddr}

#zImage + uInitrd: where uInitrd has to be generated on the running system.
#boot_fdt=run loadimage; run loadinitrd; run loadfdt
#uenvcmd=run boot_fdt; run mmcargs; bootz ${loadaddr} ${initrd_addr}:${initrd_size} ${fdtaddr}

###Begin Rootfs from NFS
#serverip=192.168.1.51
#rootpath=/srv/nfs/smarct335x/ubuntu1204/
#nfsopts=nolock,acdirmin=60
#netargs=setenv bootargs console=${console} ${optargs} root=/dev/nfs nfsroot=${serverip}:${rootpath},${nfsopts} rw ip=dhcp
##netboot=echo Loading kernel from SDCARD and booting from NFS ...; run loadimage; run netargs; bootz ${loadaddr} - ${fdtaddr}
##uenvcmd=run netboot
###End Rootfs from NFS

###Begin Load kernel from TFTP
#netmask=255.255.255.0
#ipaddr=192.168.1.65
#serverip=192.168.1.51
#netboot=echo Loading kernel and device tree from TFTP and booting from NFS ...; setenv autoload no; tftp ${loadaddr} ${kernel_file}; tftp
${fdtaddr} ${fdtfile}; run netargs; bootz ${loadaddr} - ${fdtaddr}
#uenvcmd=run netboot
###End Load kernel from TFTP
```


1. If you use SMARC-T4378-800, mem=512M in optargs. Otherwise, mem has to change to 1024M in optargs.
2. The uEnv.txt is exactly the same as that is created in SD card except the eMMC device descriptor now is mmcroot=/dev/mmcblk0p2.

Copy device tree blob to rootfs partition:

```
$ sudo cp -v /media/boot/dtbs/am437x-smarct437x.dtb /media/rootfs/home/root/am437x-smarct437x.dtb
```

Copy final rootfs to rootfs partition:

```
$ pushd /media/rootfs

$ sudo tar cvfz ~/smarct437x-emmc-rootfs.tar.gz .

$ sudo mv ~/smarct437x-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-T437X device (carrier board).

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

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

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 am437x-smarct437x.dtb /media/boot/dtbs
```

Install Root File System

```
$ sudo tar -zxvf smarct437x-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 SPI NOR flash and import u-boot environmental parameters and load kernel zImage and device tree blob from eMMC now.

Feed Packages

The following procedure can be used on a Embedian *SMARC-T437X* device to download and utilize the feed file show above to install the *minicom* terminal emulation program:

```
# vim /etc/opkg/base-feeds.conf
```

Only keep the following three lines:

```
src/gz all http://www.embedian.com/core-sdk/smarct437x/smarct4x-processor-sdk-02.00.00.00/deploy/ipk/all
src/gz cortexa9hf-vfp-neon http://www.embedian.com/core-sdk/smarct437x/smarct4x-processor-sdk-02.00.00.00/deploy/ipk/cortexa9hf-vfp-neon
src/gz smarct437x http://www.embedian.com/core-sdk/smarct437x/smarct4x-processor-sdk-02.00.00.00/deploy/ipk/smarct437x
```

```
# opkg update

# opkg upgrade

# opkg install minicom
```

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/arago-tmp-external-linaro-toolchain/work/cortexa9hf-vfp-neon-linux-gnueabi/<package name and version>` for most packages. If the package is machine-specific (rather than generic for the armv7ahf architecture), it may be located in the smarct437x-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.

-- End of Document --

version 1.0a, 1/26/2016

Last updated 2016-02-10