SMARC T335X SDK6

On this page:

Building TI/Embedian's Arago SDK 6.0 Distribution

Eric Lee

version 1.0b, 8/8/2014

Introduction

This document describes how Embedian builds a customized version of TI's am335x EZSDK 6.0 release for Embedian's SMARC T335X 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 T335X specific development.

TI makes their EZSDK 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 EZSDK 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 EZSDK builds for TI's entire suite of ARM CortextA chips. In this document we are only concerned with the oe-layersetup project.

Overview of the meta-embedian Yocto Layer

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

```
.
|-- conf
|   |-- bblayers.conf.sample
|   |-- layer.conf
|   |-- local.conf.sample
|   |-- machine
|   |   `-- smarct335x.conf
| |-- site.conf
|   `-- site.conf.sample
|-- README
|-- recipes-bsp
|   |-- u-boot
| | |-- u-boot_2014.04-smarct335x.bb
| `-- u-boot_2013.10-smarct335x.bb
|-- recipes-connectivity
|   |-- lftp
|       `-- lftp_4.0.5.bb
|-- recipes-core
|   |-- busybox
|       `-- busybox_1.20.2.bbappend
|   |-- images
|   |   |-- am335xevm-rootfs-image.bb
|   |   |-- arago-base-smarct335x-image.bb
|   |   |-- arago-base-tisdk-image.bb
|   |   |-- arago-image.inc
|   |   |-- smarct335x-amsdk-image.bb
|   |   |-- meta-toolchain-smarc-tisdk.bb
|   |   |-- smarct335x-initramfs-image.bb
|   |   `-- smarct335x-rootfs-image.bb
|   |-- initscripts
|   |   `-- initscripts_1.0.bbappend
|   |-- meta
|   |   `-- meta-toolchain-smarct335x-sdk.bb
|   |-- netbase
|   |   |-- netbase-5.0
|   |   `-- netbase_5.0.bbappend
|   |-- sysvinit
|   |   |-- sysvinit-inittab
|   |   |   `-- inittab
|   `-- packagegroups
|   |-- packagegroup-arago-smarct335x-base.bb
|   |-- packagegroup-arago-smarct335x-connectivity.bb
|   |-- packagegroup-arago-smarct335x-console.bb
|   |-- packagegroup-arago-smarct335x-sdk.bb
|   |-- packagegroup-arago-smarct335x-sdk-target.bb
|   |-- packagegroup-arago-smarct335x-test.bb
|   |-- packagegroup-arago-tisdk-matrix-no-bt-wifi-demos.bb
|   |-- packagegroup-initramfs-boot.bb
|   `-- packagegroup-arago-toolchain-tisdk-target.bbappend
|-- 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
|   |-- ltp-ddt
|   |   |--ltp-ddt
| | `-- ltp-ddt_0.0.4.bbappend
|   |-- pinmux-utility
|   |   `-- pinmux-utility_2.5.2.0.bbappend
```

```
|-- recipes-kernel
|   `-- linux
|       |-- linux.inc
|       |-- linux-smarct335x-3.2
|       |-- linux-smarct335x_3.2.bb
|       `-- linux-tools.inc
|-- recipes-multimedia
|   `-- amsdk-av-files
|       |-- amsdk-av-files_1.1.bbappend
`-- recipes-support
|   |-- boost
|   |   |-- boost_1.53.0.bb
|   |   |-- boost.inc
|   |   `-- files
|   |-- ntp
|   |   |-- files
|   |   |-- ntp_4.2.6p5.bb
|   |   `-- ntp.inc
```

Notes on ***meta-embedian*** layer content

`conf/machine/*`

This folder contains the machine definitions for the ***smarct335x*** platform. 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 ***smarct335x*** platform.

`recipes-connectivity/lftp/*`

This folder adds lftp ftp client utility for ***smarct335x*** 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 ***smarct335x*** platform:

```
MACHINE=smarct335x bitbake -k smarct335x-rootfs-image
```

`recipes-core/initscripts/*`

This recipe is used to amend TI's initialization scripts for the platform.

`recipes-core/netbase/*`

This recipe is used to amend TI's network configuration data for the platform.

`recipes-core/sysvinit/*`

This recipe is used to amend TI's console debug port configuration from ttyO0 to ttyO3 for the platform.

`recipes-devtools/nodejs/*`

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

`recipes-graphics/libgles/*`

This recipe is to add smarct335xevm machine id when loads sgx module in SMARC T3354.

`recipes-kernel/linux/*`

Contains the recipes needed to build the ***smarct335x*** Linux kernels.

`recipes-multimedia/amsdk-av-files/*`

This recipe is to add smarct335x compatible machine for the platform.

`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 TI am335x 6.0 EZSDK you first need an Unbuntu Linux 12.04LTS installation (preferably 32bit). 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. Only Yocto version 1.6 supports Ubuntu 14.04 LTS. TI SDK6 is based on Yocto 1.5 and if your host PC is Ubuntu 14.04 LTS, please use SDK7 instead.

Once you have Ubuntu 12.04 LTS running, install the additional required support packages using the following console command:

```
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo build-essential chrpath
libsdl1.2-dev xterm python-m2crypto bc
```

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 apt-get install ia32-libs-multiarch
```

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 am335x SDK you will need to install the Linaro arm compiler that TI used for the release:

```
$ wget http://releases.linaro.org/archive/13.04/components/toolchain/binaries/gcc-linaro-arm-linux-g
nueabihf-4.7-2013.04-20130415_linux.tar.xz
$ sudo tar -C /opt -xJf gcc-linaro-arm-linux-gnueabihf-4.7-2013.04-20130415_linux.tar.xz
```

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

```
export PATH=/opt/gcc-linaro-arm-linux-gnueabihf-4.7-2013.04-20130415_linux/bin:$PATH
```

Next clone and intialize TI's am335x SDK build process:

```
$ git clone http://arago-project.org/git/projects/oe-layersetup.git
$ cd oe-layersetup
$ git checkout -b tisdk-06-00 4eae4e6c5eb39a54c6bd8b41eaf972931f4af279
$ ./oe-layertool-setup.sh -f configs/amsdk/amsdk-06.00.00.00-config.txt
```


Update (12-07-2015)
Some repo locations have been moved by TI. You need to apply the following patch first.
$ cd ~/oe-layersetup-smarct33-sdk7/source/meta-arago/
$ wget -c http://developer.embedian.com/download/attachments/1245249/0001-sdk6-fix-repo-location.patch
$ patch -p1 <fix-binutils-textinfo-ubuntu14.04.patch
$ rm 0001-sdk6-fix-repo-location.patch

Add the Embedian's `meta-embedian` layer to the build process.

```
$ cd ~/oe-layersetup/sources
$ git clone git@git.embedian.com:developer/meta-embedian.git
$ cd ~/oe-layersetup/build
```

Edit the ~/oe-layersetup/build/conf/bblayers.conf file to include the meta-embedian 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.


# LAYER_CONF_VERSION is increased each time build/conf/bblayers.conf

# changes incompatibly

LCONF_VERSION = "5"


BBPATH = "${TOPDIR}"

BBFILES ?= ""


# Layers configured by oe-core-setup script

BBLAYERS += " \

/home/oeeric/oe-layersetup/sources/meta-embedian \
/home/oeeric/oe-layersetup/sources/meta-arago/meta-arago-distro \
/home/oeeric/oe-layersetup/sources/meta-arago/meta-arago-extras \
/home/oeeric/oe-layersetup/sources/meta-openembedded/toolchain-layer \
/home/oeeric/oe-layersetup/sources/meta-openembedded/meta-oe \
/home/oeeric/oe-layersetup/sources/meta-linaro \
/home/oeeric/oe-layersetup/sources/meta-ti \
/home/oeeric/oe-layersetup/sources/oe-core/meta \
"
```

Note:
If your platform is SMARC T3354 (with sgx features), you can simply add "sgx" in the MACHINE_FEATURES to have the powervr driver loaded by editing the ~/oe-layersetup/source/meta-embedian/conf/machine/smarct335x.conf file. Find MACHINE_FEATURES and add "sgx" in the very bottom:

```
MACHINE_FEATURES = "kernel26 alsa usbgadget usbhost apm vfat ext2 screen touchscreen ethernet sgx"
```

Please do not enable the sgx MACHINE FEATURES if your platform is SMARC T3352.

Building the target platforms

To build the Embedian SMARC T335X and am335x-evm developer board images, respectively, use the following commands:

```
$ cd ~/oe-layersetup/build
$ source conf/setenv
```

```
$ MACHINE=smarct335x bitbake -k smarct335x-rootfs-image
$ MACHINE=am335x-evm bitbake -k tisdk-rootfs-image
```

> **Note**
> 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.

Once it done, you can find all required images under ~/oe-layersetup/build/arago-tmp-external-linaro-toolchain/deploy/images/

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=smarct335x bitbake minicom
$ MACHINE=smarct335x bitbake package-index
```

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

`deploy/images/*`

This folder contains the binary images for the root file system and the Embedian SMARCT335X specific version of the am335x SDK. Specifically the images are:

`deploy/images/u-boot.img`

This u-boot bootloader binary for SMARC T335X

`deploy/images/MLO`

The "Stage 1 Boot Loader" for SMARC T335X. Its purpose is load the Stage 2 Boot Loader (u-boot.img).

`deploy/images/zImage`

The kernel zImage for SMARC T335X.

`deploy/images/smarct335x-rootfs-image-smarct335x*`

Embedian root file system images for software development on Embedian's SMARC T335X platforms.

`deploy/images/smarct335x-amsdk-image*`

These files contain the entire TI EZSDK augmented with Boost and other options to simplify Embedian SMARC T335X software development.

`deploy/images/arago-base-tisdk-image-smarct335x*`

These images are used to create the SMARC T335X EZSDK (see `smarct335x-amsdk-image-smarct335x*`).

`deploy/images/tisdk-rootfs-image-am335x-evm*`

TI Arago root file system images for software development on TI's am335x-evm and BeagleBone 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 plaform via a properly constructed *feed* file. Here is an example of the feed file (named arago-smarct335x-feed.conf) that is used internally at Embedian to install upgrades onto a **smarct335x** platform without reflashing the file system:

```
src/gz smarct335x http://www.embedian.com/arago/smarct335x

src/gz armv7ahf_vfp-neon http://www.embedian.com/arago/armv7ahf-vfp-neon-3.2
```

```
src/gz all http://www.embedian.com/arago/all
```

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

`deploy/sdk/arago-2013.05-armv7a-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:

```
$ 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/dtbs
$ 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

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

deploy/images

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

uEnv.txt based bootscript

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

For v3.2:

~/uEnv.txt

```
optargs="consoleblank=0 mem=512M"
bootfile=zImage
loadaddr=0x80200000
#fdtaddr=0x80F80000
#fdtfile=am335x-smarct335x.dtb
console=ttyO3,115200n8
mmcroot=/dev/mmcblk0p2 rw
mmcrootfstype=ext4 rootwait fixrtc

#To boot old v3.2.x based kernel enable: (SMARC T335X and BeagleBone)
uenvcmd=run loadimage; run mmc_classic_boot

#For u-boot 13.10
#uenvcmd=run loadzimage; run mmc_classic_boot

###Begin Rootfs from NFS
#serverip=192.168.1.51
#rootpath=/srv/nfs/smarct335x/arago6/
#nfsopts=nolock
#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 loaduimage; run netargs; bootz ${loadaddr}
##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 from TFTP and booting from NFS ...; setenv autoload no; tftp ${loadaddr} ${bootfile}; run netargs; bootz
${loadaddr}
#uenvcmd=run netboot
###End Load kernel from TFTP
```

Copy uEnv.txt to the boot partition:

~/

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

Install Kernel zImage

Copy zImage to the boot partition:

deploy/images

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

Install Root File System

Copy Root File System:

Arago:

deploy/images

```
$ sudo tar xvfz smarct335x-rootfs-image-smarct335x.tar.gz -C /media/rootfs
```

Note:

Kernel modules are included in root file systems.

The following procedure can be used on a Embedian SMARC T335X device to download and utilize the feed file show above to install the minicom terminal emulation program:

```
# cd /etc/opkg
# wget http://www.embedian.com/feedfiles/arago-smarct335x-feed.conf
# opkg update
# opkg upgrade
# opkg install minicom
```

Occasionally you may run into build problems with package ti-compat-wireless-wl12xx when rebuilding an image. If you do run into this, perform one of the following clean up commands by selecting the MACHINE target that is causing you problems:

```
$ MACHINE=am335x-evm bitbake -f -c cleanall ti-compat-wireless-wl12xx
$ MACHINE=smarct335x bitbake -f -c cleanall ti-compat-wireless-wl12xx
```

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/armv7ahf-vfp-neon-3.2-oe-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 smarct335x-oe-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. When booting from eMMC and SD card is present, SD card is emulated as /dev/mmcblk0 and eMMC is emulated as /dev/mmcblk1. On the other hand, when booting from eMMC and SD card is absent, eMMC will be emulated as /dev/mmcblk0 now. eMMC could be /dev/mmcblk0 or /dev/mmcblk1 depending on if SD card is inserted and the boot device become dynamic when booting from eMMC.

Initramfs is the successor of initrd and has many advantages over initrd. Linux kernel here will mount it as a temperately rootfs and starts the init process from here. The init script will check if the partition 2 of eMMC is exist and them mount the real rootfs.

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.

Get initramfs (assuming the home directory is /home/developer here)

```
$ cd ~/

$ wget http://developer.embedian.com/download/attachments/2883656/initramfs.tar.gz?version=1&modificationDate=1414480792916&api=v2

$ mkdir initramfs

$ sudo tar xvfz initramfs.tar.gz -C initramfs/
```

Note: The above initramfs is obtained by the following steps:

```
$ cd ~/oe-layersetup/build

$ source conf/setenv

$ MACHINE=smarct335x bitbake -k smarct335x-initramfs-image
```

You will find smarct335x-initramfs-image-smarct335x.tar.gz file
under ~/oe-layersetup/build/arago-tmp-external-linaro-toolchain/deploy/images/

Extract this tarball and add your own init script. Users can use Embedian's init script for references.

Prepare for initramfs zImage

```
$ MACHINE=smarct335x bitbake virtual/kernel -c menuconfig
```

Select

General setup -->

[*] Initial RAM filesystem and RAM disk (initramfs/initrd) support

() Initramfs source file(s)

Enter the directory where your initramfs is. In this example

/home/developer/initramfs

Save the kernel config and build again.

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 MLO to rootfs partition:

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

Copy u-boot.img to rootfs partition:

```
$ sudo cp -v /media/boot/u-boot.img /media/rootfs/home/root
```

Copy initramfs zImage to rootfs partition:

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

```
$ 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)

For kernel v3.2:

```
optargs="consoleblank=0 mem=512M"
bootfile=zImage
loadaddr=0x80200000
#fdtaddr=0x80F80000
#fdtfile=am335x-smarct335x.dtb
console=ttyO3,115200n8
#mmcroot=/dev/mmcblk0p2 rw
#mmcrootfstype=ext4 rootwait fixrtc
mmcroot=/dev/ram0
#To boot old v3.2.x based kernel enable: (SMARC T335X and BeagleBone)
uenvcmd=run loadimage; run mmc_classic_boot
#For u-boot 13.10
#uenvcmd=run loadzimage; run mmc_classic_boot
###Begin Rootfs from NFS
```

```
#serverip=192.168.1.51
#rootpath=/srv/nfs/smarct335x/arago6/
#nfsopts=nolock
#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 loaduimage; run netargs; bootz ${loadaddr}
##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 from TFTP and booting from NFS ...; setenv autoload no; tftp ${loadaddr} ${bootfile}; run netargs; bootz
${loadaddr}
#uenvcmd=run netboot
###End Load kernel from TFTP
```

Copy real rootfs to rootfs partition:

```
$ pushd /media/rootfs

$ sudo tar cvfz ~/smarct335x-emmc-rootfs.tar.gz .

$ sudo mv ~/smarct335x-emmc-rootfs.tar.gz /media/rootfs/home/root

$ popd
```

Remove SD card:

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

Insert this SD card into your SMARC T335X device.

Now it will be almost the same as you did when setup your SD card, but the eMMC device descriptor is /dev/mmcblk1 now.

```
$ export DISK=/dev/mmcblk1
```

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__
```

In SDK6, Arago rootfs will mount partition2 as /media/mmcblk1p2 automatically after executing the above command. Umount the partition first.

```
$ sudo umount /media/mmcblk1p2
```

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 MLO/u-boot.img/uEnv.txt/zImage to the boot partition

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

Install Root File System

```
$ sudo tar -zxvf smarct335x-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 1.0b, 1/13/2014

Last updated 2015-12-07