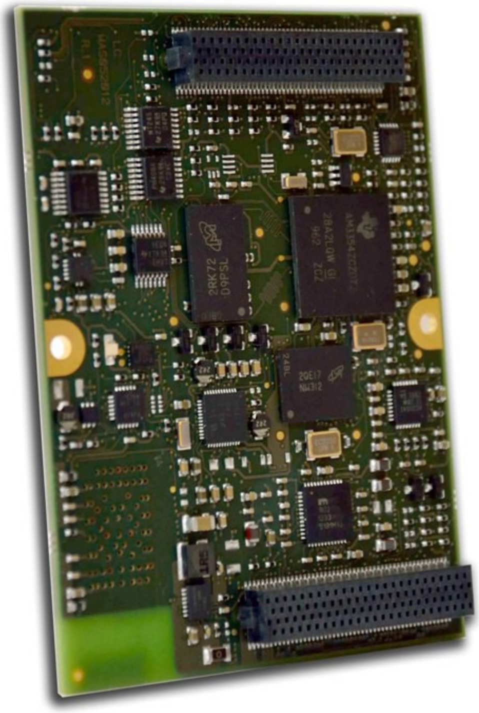
FIAMMA MCAM335x
Linux User's Guide



Mas Elettronica Sas
Via Risorgimento 16/C
35030 Selvazzano Dentro (PD)
Italy.

Revision History

Date	Version	Description	Author
10/05/2013	1.0	First Release	N.Convertino
15/09/2014	1.1	Second Release: updated external links and commands to programm CPU board	F.Molon

About this Manual

This guide describes how to install Mas Elettronica's Linux Support Package (LSP) for the MCAM335X processor. This LSP provides a fundamental software platform for development, deployment and execution on MCAM335X processor. It abstracts the functionality provided by the hardware. In this context, the guide contains instructions to:

- Install the release on a development machine.
- Build the sources included in this release.
- Installing the binaries on the MCAM335X.
- Booting the MCAM335x.

Installation

Introduction

Mas Elettronica supports its MCAM335x with linux kernel 3.2.0 and Arago file-systems for reference. The Linux kernel provides support for all on-board peripherals. The linux folder on the FTP consists of:

- Ready-to-run pre-built binaries: Arago file system, Linux kernel, U-boot, Xloader.
- Sources files.

Prerequisites

Before starting the installation of the package, make sure below system requirements are met:

- Host machine running a version of Windows OS such as Windows XP / 7 or a Linux such as Ubuntu.

The linux folder on the Virtual Machine consists of:

  - Recompiling U-Boot and kernel.
  - Hosting the NFS server to boot the EVM with NFS as root filesystem.

Either of Windows or Linux host can be used for:

- Hosting the TFTP server required for downloading the kernel and file-system images from U-Boot using network Ethernet.

- Running a serial console terminal application.

Install LSP packages

Mas Elettronica LSP consists of :

- Linux kernel source code
- U-Boot source code
- Linux root file-system binaries examples

Extract the contents of the LSP release packages on a Linux host machine with the following commands:

```
$ mkdir SDK-MAS-<kit-name>.R1
$ tar -xzvf linux-3.2.0-sdk.mas_r1.tar.gz -C SDK-MAS-<kit-name>.R1
$ tar -xzvf u-boot-sdk.mas_r1.tar.gz -C SDK-MAS-<kit-name>.R1
$ tar -xzvf binary-sdk.mas_r1.tar.gz -C SDK-MAS-<kit-name>.R1
```

In the Virtual Machine the direcory are already extracted. These creates a directory **MCAM335X -LSP** with the following contents:

```
 \--- SDK-MAS.R1-<kit-name>.R1
 +----linux-3.2.0-sdk.mas_r1/
 |     +----linux source code
 +----u-boot-sdk.mas_r1/
|     +----uboot source code
 +----binary-sdk.mas_r1/
|     |----base-rootfs-mcam335x.tar.gz
|     |----boot/
|     |     |----MLO
|     |     |----u-boot.img
|     |     |----uImage
```

U-Boot Source Code

- U-Boot sources directory is at **u-boot-sdk.mas_r1**

Linux Kernel Source Code

- Linux kernel source directory is at **linux-3.2.0-sdk.mas_r1**

Linux Root File-System

To boot-up Linux, a target file-system is needed. Two Arago based file-systems are included in the LSP binaries package. It has some basic utilities installed but is intended to be rather small and light weight. Demo filesystem (~300MB). This file system is created by taking the base file system and adding all the additional SDK components such as 3D graphics, matrix, profiling tools, etc... Further explanation about customizing these file-systems can be found to following site:

http://processors.wiki.ti.com/index.php/AMSDK_File_System_Optimization/Customization

Toolchain

GNU toolchain for ARM processors from Arago is recommended. Arago Toolchain can be found in the linux-devkit directory of the SDK here: http://aragoproject.org/wiki/index.php/Setting_Up_Build_Environment. Arago Toolchain is already installed in this O.S.

Environment Setup

After installing the toolchain, the environment in the Linux host needs to be setting. Set the environment variable PATH to contain the binaries of the Arago cross-compiler tool-chain. For example, in bash:

```
$ export PATH=/opt/toolchain/arm-arago-gcc-4.5.3/bin/:$PATH
```

Add the location of u-boot tools directory to the PATH environment variable (required for mkimage utility that is built as part of U-Boot build process and is needed to generate uImage when building the kernel). For example, in bash:

```
$ export PATH=/opt/u-boot/tools:$PATH
```

NOTE: Actual commands to be used for setting the environment variables will depend upon the shell and location of the tools. Another to help get started quickly, the LSP package comes with pre-built binaries. However, after making any changes to U-Boot and/or Linux Kernel, they have to be cross-compiled and the new binaries that are generated should be used.

NOTE: To help get started quickly, the LSP package comes with pre-built binaries. However, after making any changes to U-Boot and/or Linux Kernel, they have to be cross-compiled and the new binaries that are generated should be used.

U-Boot

In AM335x the ROM code serves as the 1st stage bootloader. The 2nd and the 3rd stage bootloaders are based on U-Boot. The binary for the 2nd stage is referred to as SPL and the binary for the 3rd stage as simply U-Boot. SPL is a non-interactive loader and is a specially built version of U-Boot. It is built concurrently when building U-Boot. The ROM code can load the SPL image from the NAND or SDMMC devices.

Quickly compile uboot

Before you compile the source file, recall the toolchain installed in the VM. The toolchian is installed in /opt/arm-2009q1 directory. Run the following command to recall the toolchain:

```
$ . /home/mas/MAS/cross_omap_2009
```

To compile the u-boot enter in u-boot-sdk.mas_r1 directory and run the follow command:

```
$ make am335x_evm_config
$ make
```

At the end of compilation you find the MLO and u-boot.img file in u-boot-sdk.mas_r1 directory. Copy these files in boot partition of your SD Card.

Building U-Boot

Change to the base of the U-Boot directory.

```
$ cd <new-name-dir>/u-boot-sdk.mas_r1
```

NOTE: Building into a separate object directory with the "O=" parameter to make is strongly recommended.

Build

```
$ [ -d ./mcam335x ] && rm -rf ./mcam335x
$ make O=mcam335x CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm mcam335x_config
$ make O=mcam335x CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm
```

This will generate two binaries in the mcam335x directory, MLO and u-boot.img along with other intermediate binaries that may be needed in some cases. For information on installing the kernel into NAND on the SOM please see the Installing the Linux Kernel section.

U-Boot Environment Settings

The MCAM335X U-Boot has default environment settings that allows boot from NAND, SD/MMC card and Ethernet. By default the boot device is NAND, for more information about boot options go to Boot section.

Linux Kernel

Cleaning the Kernel Sources

Prior to compiling the Linux kernel make sure that the kernel sources are clean. Enter linux kernel directory:

```
$ cd <new-name-dir>/linux-3.2.0-sdk.mas_r1
```

NOTE: The next step will delete any saved .config file in the kernel tree as well as the generated object files. If you have done a previous configuration and do not wish to lose your configuration file you should save a copy of the configuration file before proceeding. The command to clean the kernel is:

```
$ make clean ARCH=arm
```

If you want delete the kernel configuration use the following command:

```
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- mrproper
```

Configuring the Kernel

Before compiling the Linux kernel it needs to be configured to select which components will become part of the kernel image:

Using Default Configurations

To build the default configuration for the MCAM335X:

```
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi-
```

Customizing the configuration kernel

For configuring the kernel run:

```
$ make menuconfig
```

Once the configuration window is open you can select which kernel components will be included in the build. Exiting the configuration will save your selections to a file in the root of the kernel tree called .config.

Quickly compiling the Kernel

Enter in **linux-3.2.0-sdk.mas_r1** direcory and recall the arago toolchain with the following command:

```
$ . /home/mas/MAS/cross_omap_2009
```

To compile the kernel source run the following command:

```
$ make uImage ARCH=arm
```

At the end of compilation, will be generated the uImage file in **arch/arm/boot** directory. Copy uImage file in boot partition of SD Card.

Compiling the Kernel

Once the kernel has been configured compile kernel:

```
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- uImage
```

This will result in a kernel image file being created in the **arch/arm/boot/** directory called uImage. This file can be used by u-boot to boot your device. If you selected any components of the kernel to be build as dynamic modules you must issue an additional command to compile those modules. The command is:

```
$ make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabi- modules
```

This will result in .ko (kernel object) files being placed in the kernel tree. These .ko files are the dynamic kernel modules. The next section will cover how to install these modules.

Installing the Kernel

Once the Linux kernel and modules have been compiled they must be installed. In the case of the kernel image this can be installed by copying the uImage file to the location for downloading using TFTP, or put in an SD-card. For example: when using TFTP boot, /tftpboot directory is the common location, whereas when booting from SD card, file shoudl be put in the first FAT partition. To install the kernel modules, provide teh rootfs location, see below. This command will create a directory tree in that location: lib/modules/<kernel version> which will contain the dynamic modules corresponding to this version of the kernel. The base location should usually be the root of your target file system. The general format of the command is:

```
$ make ARCH=arm CROSS_COMPILE=arm-arago-none-gnueabi- INSTALL_MOD_PATH=<path to root of file system> modules_install
```

For example if you are installing the modules to an NFS share located at /home/user/targetNFS you would do:

```
$ make ARCH=arm CROSS_COMPILE=arm-arago-none-gnueabi- INSTALL_MOD_PATH=/home/user/targetNFS modules_install
```

Out-of-tree Kernel Modules

NOTE: Some drivers like the SGX and WLAN drivers are delivered as modules outside of the kernel tree. These drivers binaries are already included in the pre-build root file-systems provided by Mas Elettronica.

Boot

The Kernel and root the file-system can be booted either from NAND, SD-Card or can be retrieved via ethernet to RAM using TFTP. Nand Flash root file-system is UBIFS based which is the most recommended filesystem for nand flashes. Following sections describe various kernel boot options possible.

Boot from MMC/SD

For creating a bootable SD , follow the below instruction on creating an SD described in the paragraph **Nand Recovery.** To boot the Linux, type:

```
U-Boot# run mmc_boot
```

Boot from NAND

The MCAM335X can boot from NAND. The SPL, U-Boot and kernel uImage can be flashed on the NAND flash.

Flash Images to NAND

Replacing Nand Flash images can be done from either Linux user space or U-Boot.

<u>From U-Boot</u>

Get the images to U-Boot via TFTP or MMC/SD.

```
U-Boot# mmc rescan
U-Boot# nandecc hw 2
U-Boot# fatload mmc 0 0x82000000 MLO
U-Boot# nand erase 0x0 0x20000
U-Boot# nand write.i 0x82000000 0x0 0x20000
U-Boot# mmc rescan
U-Boot# nandecc hw 2
U-Boot# fatload mmc 0 0x82000000 u-boot.img
U-Boot# nand erase 0x80000 0x40000
U-Boot# nand write.i 0x82000000 0x80000 0x40000
U-Boot# mmc rescan
U-Boot# nandecc hw 2
U-Boot# fatload mmc 0 0x82000000 uImage
U-Boot# nand erase 0x280000 0x500000
U-Boot# nand write.i 0x82000000 0x280000 0x480000
```

<u>From Linux</u>

Put the images on the file-system.

```
<< Install SPL >>
$ flash_erase /dev/mtd0 0 0
$ flash_erase /dev/mtd1 0 0
$ flash_erase /dev/mtd2 0 0
$ flash_erase /dev/mtd3 0 0
$ nandwrite -p /dev/mtd0 <MLO file>
$ nandwrite -p /dev/mtd1 <MLO file>
$ nandwrite -p /dev/mtd2 <MLO file>
$ nandwrite -p /dev/mtd3 <MLO file> << Install U-Boot >>
$ flash_erase /dev/mtd4 0 0
$ flash_erase /dev/mtd5 0 0
$ nandwrite -p /dev/mtd4 <u-boot.img file> << Install Kernel >>
$ flash_erase /dev/mtd6 0 0
$ nandwrite -p /dev/mtd6 <uImage file>
```

Boot over Network (Ethernet)

NOTE: When setting a MAC address please ensure that the LS-bit of the 1st byte is not 1 i.e. when setting the MAC address: y in x**y**:ab:cd:ef:gh:jk has to be an even number. For more info this refer to the wiki page http://en.wikipedia.org/wiki/MAC_address.

When kernel image and root file-system are fetched from a TFTP/NFS server:

- Ensure that the SOM is connected to network with DHCP and TFTP server set up.
- If the TFTP server supports negotiation between client and server, disable it.
- Copy 'uImage' kernel image to TFTP server's root directory.
- Set 'ethaddr' U-Boot environment variable with proper ethernet address in format 'xx:xx:xx:xx:xx:xx' (replace 'xx' with proper hexadecimal values).
- Setup NFS server and export one of the provided pre-build root file-system.
- Execute following commands at U-Boot prompt. Assuming kernel image name as 'uImage':

```
U-Boot# setenv serverip <Server IP address>
```

```
U-Boot# setenv rootpath <Path of the exported root file-system on the NFS server>
U-Boot# run net_boot
```

UBIFS

UBIFS is used for Linux root file-system on the MCAM335X NAND Flash.

Compilling UBIFS Tools

The MTD and UBI user-space tools are available from the the following git repository:

```
$ git clone git://git.infradead.org/mtd-utils.git
$ cd mtd-utils/
$ git checkout v1.5.0
$ make
```

IMPORTANT

Tested wuth mtd-utils version is 1.5.0. For instructions on compiling MTD-utils, refer MTD-Utils Compilation: http://processors.wiki.ti.com/index.php/MTD_Utilities#MTD-Utils_Compilation

Creating UBIFS

This section describes steps for creating a UBI rootfs image to be flashed to the MCAM335X NAND Flash.

- mkfs.ubifs

```
$ mkfs.ubifs/mkfs.ubifs -r rootFS/ -F -o system_ubifs.img -m 2048 -e 126976 -c 1960
```

Where:

-m 2KiB (or 2048)

The minimum I/O size of the underlying UBI and MTD devices. In our case, we are running the flash with no sub-page writes, so this is a 2KiB page.

-e 124KiB (or 126976)

Erase Block Size: UBI requires 2 minimum I/O units out of each Physical Erase Block (PEB) for overhead: 1 for maintaining erase count information, and 1 for maintaining the Volume ID information. The PEB size for our flash is 128KiB, so this leads to each Logical Erase Block (LEB) having 124KiB available for data.

-c 1960

The maximum size, in LEBs, of our file system.

-r rootFS

Use the contents of the '**rootFS/**' directory to generate the initial file system image.

-F

File-system free space has to be fixed up on first mount (http://www.linux-mtd.infradead.org/faq/ubifs.html#L_free_space_fixup)

-o system_ubifs.img

Output file.

NOTE: On AM335x, -F option is required when creating ubifs image. If this option is not used, Kernel may crash while loading the Filesystem from UBI partition.

```
The output of the above command, 'system_ubifs.img' is fed into the 'ubinize' program to wrap it
into a UBI image. The images produced by mkfs.ubifs are later used by the ubinize tool to create a
UBI image is flashed to the raw flash to be used a UBI partition.
```

- Create ubinize.cfg file and write the bellow contents into it:

```
[rootfs]
mode=ubi
image=system_ubifs.img
vol_id=0
vol_size=220MiB
vol_type=dynamic
vol_name=rootfs
vol_flags=autoresize
```

- ubinize

```
$ ubi-utils/ubinize -o rootfs-mcam335x.ubi.img -m 2048 -p 128KiB -s 2048 -O 2048 ubinize.cfg
```

Where:

-o rootfs-mcam335x.ubi.img

Output file.

-m 2KiB (or 2048)

Minimum flash I/O size of 2KiB page.

-p 128KiB

Size of the physical erase block of the flash this UBI image is created for

-O 2048

offset if the VID header from start of the physical erase block

The output of the above command, '**rootfs-mcam335x.ubi.img'** is the required image.

Using UBIFS

We can Flash UBIFS image from either Linux Kernel or U-Boot.

From U-boot

Get the UBIFS image to U-Boot from tftp or MMC/SD. Since we copy the data to NAND, Empty/Erase the required RAM. Then, get the UBIFS image to U-Boot

```
u-boot# mw.b ${loadaddr} 0xFF <filesystem_image_size> <=== filesystem image size is upward aligned
to NAND block size(128k).
u-boot# mmc rescan
u-boot# fatload mmc 0 ${loadaddr} base-rootfs-mcam335x.ubi.img
u-boot# nandecc hw 2
u-boot# nand erase 0x00780000 0xF880000
u-boot# nand write.i ${loadaddr} 0x780000 0xFC0000
```

From Linux

```
$ flash_erase /dev/mtd7 0 0
$ ubiformat /dev/mtd7 -f base-rootfs-mcam335x.ubi.img -s 2048 -O 2048
```

NAND Recovery

As an easy and fast way to recover the MCAM335X NAND flash, Mas Elettronica provides a recovery SD card image that can be used to install the pre-built Linux and Android systems.This SD card image includes a script (nand-recovery.sh) that installs all the boot images and root file-system.

Preparing rescue SD-Card

- Plug your SD card to your Linux machine, run dmesg and see what device is added (i.e. /dev/sd**X**)
- gunzip am33-som-nand-recovery-sd.v10.img.gz
- dd if=am33-som-nand-recovery-sd.v10.img of=/dev/sd**X** bs=128k

Recover Nand Flash

- Insert the SD card into the SD/MMC slot of the custom board

- Press and hold the boot select switch while powering ON the board
- Login as root (no password)
- From Linux command line, type: "**nand-recovery.sh**". (This will install Linux on the NAND)
- Unplug the SD card and reboot

NAND recovery script usage:

```
usage: /sbin/nand-recovery.sh options

This script install Linux/Android binaries in mcam335x NAND.

OPTIONS:
   -h Show this message
   -o <Linux|Android> OS type (defualt: Linux).
```

Reference Documentation

How to Flash LinuxSystem from U-boot
AMSDK U-Boot User's Guide
AM335X Flash Programming Guide
AMSDK Linux User's Guide
AM335X PSP User's Guide
UBIFS Support

Rohs compliance

The MCAM335x Standalone Embedded CPU Board comply with the European Union’s Directive 2002/95/EC: "Restrictions of Hazardous Substances".

Warranty Terms

MAS Elettronica guarantees hardware products against defects in workmanship and material for a period of one (1) year from the date of shipment. Your sole remedy and MAS Elettronica’s sole liability shall be for MAS Elettronica, at its sole discretion, to either repair or replace the defective hardware product at no charge or to refund the purchase price. Shipment costs in both directions are the responsibility of the customer. This warranty is void if the hardware product has been altered or damaged by accident, misuse or abuse.

Disclaimer of Warranty
THIS WARRANTY IS MADE IN LIEU OF ANY OTHER WARRANTY, WHETHER EXPRESSED, OR IMPLIED, OF MERCHANTABILITY, FITNESS FOR A SPECIFIC PURPOSE, NON-INFRINGEMENT OR THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION, EXCEPT THE WARRANTY EXPRESSLY STATED HEREIN. THE REMEDIES SET FORTH HEREIN SHALL BE THE SOLE AND EXCLUSIVE REMEDIES OF ANY PURCHASER WITH RESPECT TO ANY DEFECTIVE PRODUCT.
Limitation on Liability
UNDER NO CIRCUMSTANCES SHALL MAS ELETTRONICA BE LIABLE FOR ANY LOSS, DAMAGE OR EXPENSE SUFFERED OR INCURRED WITH RESPECT TO ANY DEFECTIVE PRODUCT. IN NO EVENT SHALL MAS ELETTRONICA BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES THAT YOU MAY SUFFER DIRECTLY OR INDIRECTLY FROM USE OF ANY PRODUCT.

Contact Informations

Headquarters

Mas Elettronica Sas
Via Rossi 1
35030 Rubano (PD)
Italy.
Tel +39 0498687469
Fax +39 0498687469

Sales : amm@maselettronica.com
Support: info@maselettronica.com