

AT-501

Cortex-A5

System on Module



Software user guide

- Linux

Table of Contents

1	Overview	4
2	Linux SW overview and installation guide	4
2.1	Linux overview	4
2.2	Supported Linux versions.....	6
2.3	SW Installation steps.....	6
2.4	SAM-BA activation process	6
2.4.1	Bootstrap and U-boot installation steps.....	8
2.4.2	Linux Kernel, device tree and File system installation.....	10
2.5	Installation via TFTP	10
2.5.2	Using SD CARD	14
3	SW version creation and modification.....	16
3.1	Cross Compilation	16
3.1.1	Introduction	16
3.1.2	Setting Cross Compilation Environment	16
3.1.3	Cross Compiling V1.5	17
3.1.4	Cross Compiling V1.4	19
3.2	Running User Applications on ShiraTech Board.....	22
3.2.1	Cross compiling C code	22
3.2.2	Cross compiling Java code.....	23
4	Utilizing the eMMC device.....	24
4.1	USING RAMDISK & TFTP.....	24
4.1.1	Creating ramdisk.gz in host Linux PC	24
4.1.2	Boot from SD-CARD and mount compressed ramdisk.....	24
4.2	Formatting the eMMC device using SAM-BA	26
4.3	Using NAND flash	27
5	CB-20 Carrier board description	28

5.1	Overview	28
5.2	CB20 interfaces mapping	29
5.3	Interrupt & I/O Table	30
5.4	Jumpers	31

Document Revision History

Revision	Date	Description
1.0	26.6.2013	Initial version
1.1	11.9.2013	Update document to support hardware version 2.0 of the CB-20
1.2	17.2.2014	Update boot arg, pictures, download links
1.3	18.9.2014	Add support for version 1.4
1.4	8.9.2015	Add support for version 1.5, adding information for utilizing the eMMC device

1 Overview

The software user guide includes 3 parts:

- **Section 2** includes a description of the SW files required, their location in the memory along with an explanation on how to download them to the target device using the tools provided by Atmel.
- **Section 3** describes how to create, modify and compile the various parts of the SW.
- **Section 4** describes how to utilize the eMMC device
- **Section 5** describes the CB-20 evaluation board providing details on the available ports and a hardware/software guide.

2 Linux SW overview and installation guide

2.1 Linux overview

Linux software required for operating the AT-501 consists of following parts:

1. **Boot Program** (located in the internal ROM of the processor) – it checks if a valid bootstrap is present in SD card or NAND FLASH and in case it find it, the bootstrap is downloaded it into internal SRAM.
2. **AT91Bootstrap** - In charge of hardware configuration, download U-Boot binary from FLASH to SDRAM, start the boot loader.
3. **U-Boot** - The boot loader, in charge of download kernel binaries from FLASH, network, USB key, etc... and for starting the kernel.
4. **Linux kernel** - The operating system kernel.
5. **Device tree** - The Device Tree is a binary file containing the description of the SoC and the board.
6. **Root File-system** - Contains applications which are executed on the target, using the OS kernel services.

The Linux SW can be located in the NAND flash or alternatively in an SD card. If located in the NAND flash it will be located as described in figure 1. If an SD card is used, it should be divided

to two partitions. A FAT partition for Boot, U-boot, Kernel and device tree and an additional ext2/3/4 partition for Linux file-system see figure 2.

Note: When the AT-501 is equipped with both a NAND flash and valid SD card, the processor will use the SD card.

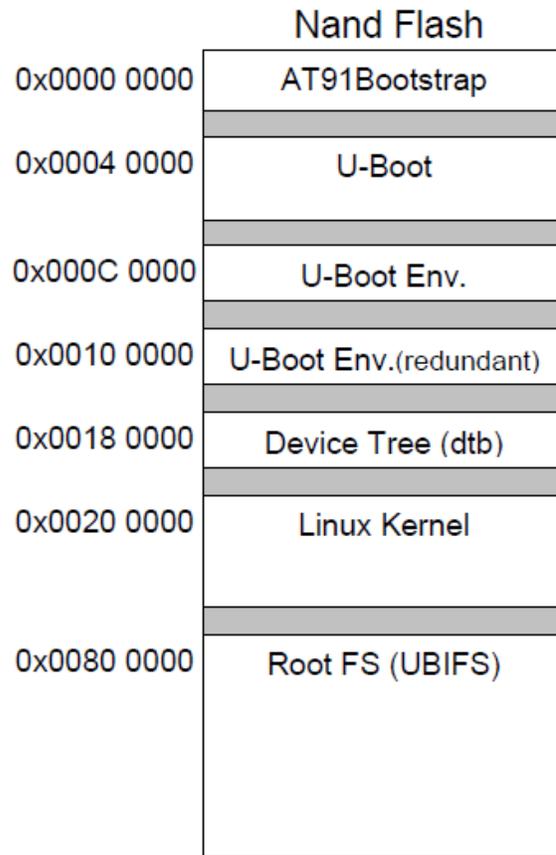


Figure 1 – NAND Flash memory mapping

<p>16MB FAT partition</p> <p>boot.bin (AT91bootstrap) u-boot.bin (U-boot) ulmage (kernel image) dtd (device tree)</p>	<p>ext2 (Linux) partition</p> <p>buildroot root filesystem</p>
---	---

Figure 2 – SD card structure

Note: It is recommended to use Ext3 for enhanced robustness of the solution.

2.2 Supported Linux versions

Version 1.4 delivered by Shiratech is based on the following:

- Linux Kernel version - 3.6.9
- Linux file system - DEBIAN Version 6.0.4

Version 1.5 delivered by Shiratech is based on the following:

- Linux Kernel version - 3.10
- Linux file system - DEBIAN Version 7

2.3 SW Installation steps

The following steps describes the how to install boot, U-boot, Linux Kernel, device tree and file system in to the NAND Flash. The boot and U-boot will be supplied with the SoM but the following process can be done in case of memory corruption, or in case of an upgrade. For these cases we offer to use the SAM-BA tools supplied by Atmel (see paragraph 2.3.1).

Kernel, device tree and file system can be installed in one of the following ways:

- Via the SAM-BA tool.
- Via TFTP from a host PC.

The following paragraphs will describe each option.

In case of using an SD card, the boot, U-boot, Linux Kernel and device tree files should be copied to the FAT (16 or 32) part of the SD while the file system should be placed in the ext part (see figure 2).

2.4 SAM-BA activation process

ATMEL provides a software tool called SAM-BA to burn the boot loader in the Flash memory along with other initialization services. To start the SAM-BA you should download the Software from ATMEL website, from the following link:

<http://www.atmel.com/tools/ATMELSAM-BAIN-SYSTEMPROGRAMMER.aspx>

Note: the version should be SAM-BA 2.12 Patch 2a or higher for proper operation.

To activate the SAM-BA perform the following steps:

1. Connecting the SAM-BA

Connect the USB cable to the Micro USB (USB A) port of the board (See port location in the picture below). Additionally by connecting the Debug port, debug messages of the SAM-BA can be viewed using a hyper terminal application.

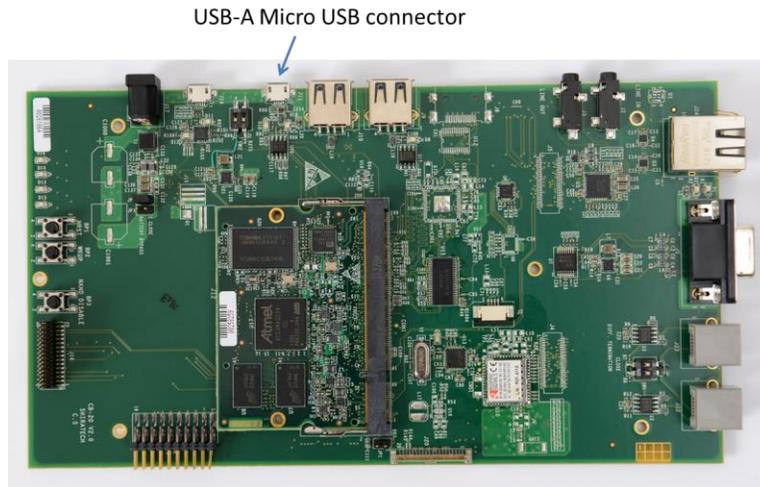


Figure 3 – Micro USB port for SAM-BA

2. Disable the NAND

If the NAND flash contains a valid BOOT, it should be disabled during power up in order to enable the operation of the SAM-BA. Disabling the NAND is done pressing the NAND disable button during power up. Once the system is up the SAM-BA can be activated. If you are using an SD card it should be plugged out.

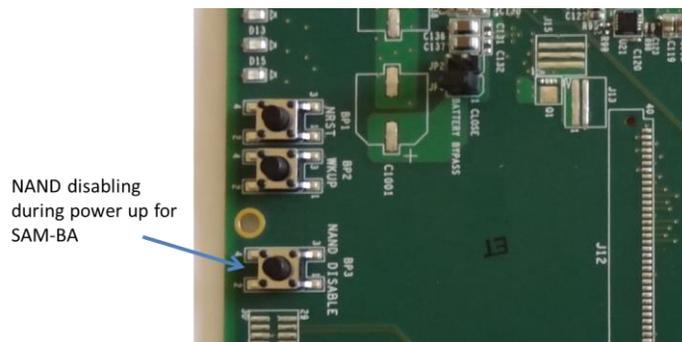


Figure – 4 NAND disable push button

Double click on SAM-BA and it will open the following window:

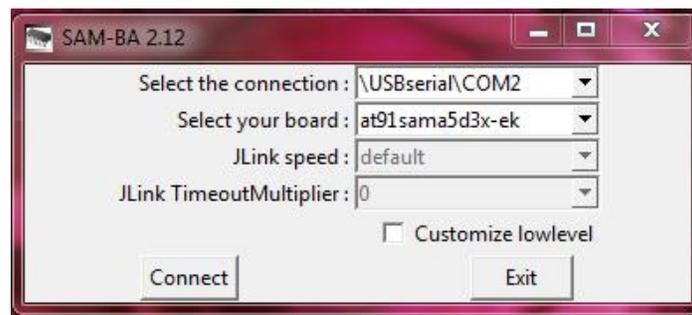


Figure 5 - SAM-BA connecting screen

The virtual COM port number (COM2) may vary from PC to PC. Click Connect button and it will open a window.

2.4.1 Bootstrap and U-boot installation steps

The following paragraph describes the download procedure for the following components:

- Boot
- U-boot

2.4.1.1 Boot file download procedure:

1. Choose the NandFlash media tab in the SAM-BA GUI.
2. Initialize the media choosing the Enable NandFlash action in the Scripts rolling menu and press Execute (If the Flash is not empty it is recommended to use the Erase all option to clean the Flash).
3. Select Send Boot File in the Scripts rolling menu and press Execute. Then select the boot file and press Open, the media is written down.

2.4.1.2 U-boot file download procedure:

To download the U-boot to the NAND flash the following steps should be done:

1. Choose the NandFlash media tab in the SAM-BA GUI.
2. Initialize the media choosing the Enable NandFlash in the rolling menu and press Execute.
3. If the software version used is 1.4 or above skip this step

Select Enable OS PMECC parameters in the Scripts rolling menu and press Execute. The default ECC Configuration should be ok (note: in patch 5 the default configuration has change, the PMECC should be configured according to the window below) so you should have this pop-up appearing:

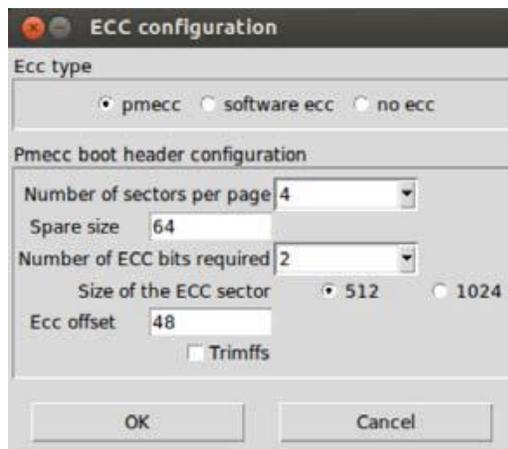


Figure 6 - SAM-BA ECC configuration screen

Select the right configuration and press the OK button.

4. To erase only the U-Boot part into the NAND FLASH, type this command after the SAM-BA prompt:

```
NANDFLASH::EraseBlocks 0x40000 0xBFFFF
```

Then press the Enter Key.

Note: The SAM-BA EraseBlocks command take two parameters: a start and an end address of FLASH that you want to erase.

5. Press *Send File Name Browse* button and choose your U-Boot binary file.

6. Enter the proper address on media in the Address text field. **Its value should be 0x40000.**

7. Press Send File button.

Close SAM-BA, remove the USB cable and restart the device. Once the boot and U-boot files are installed power up the board. You will get the Boot loader prompt (**U-boot>**).

See also the picture below for the exact steps required

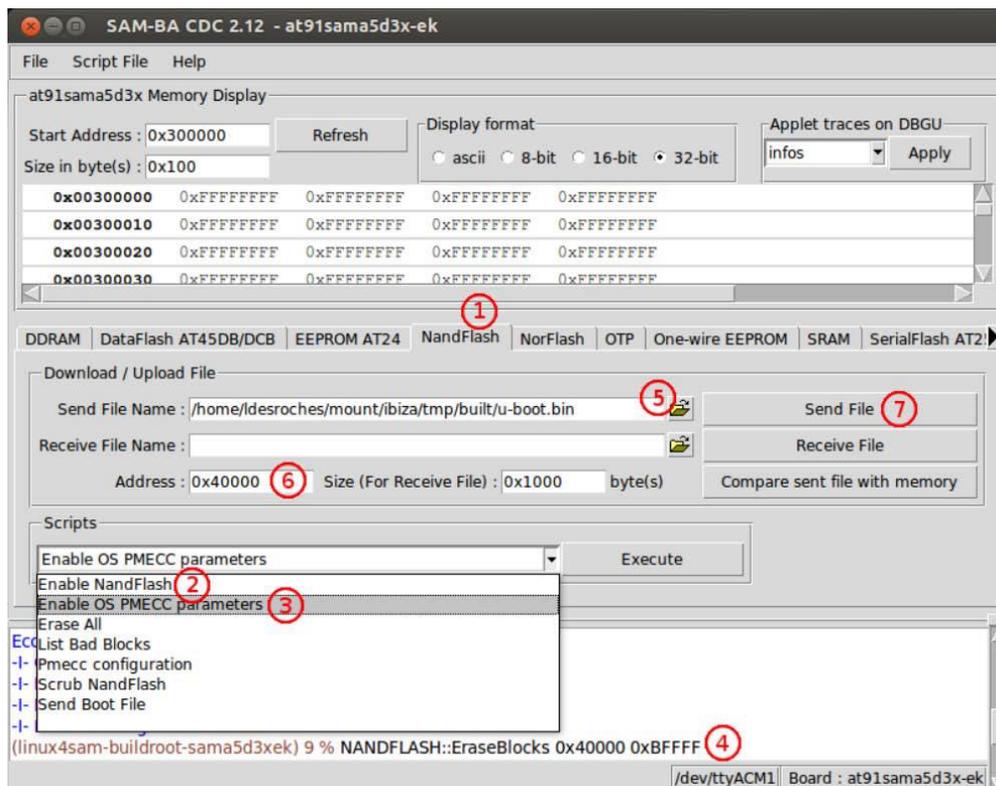


Figure 7 – U-boot download procedure

2.4.2 Linux Kernel, device tree and File system installation

The Linux Kernel, device tree and File system can be downloaded either via the SAM-BA or alternatively using TFTP over the Ethernet port. The TFTP configuration is done using the U-boot menu.

2.5 Installation via TFTP

Once the boot loader is installed on board, the Linux kernel, device tree and file system can be installed by the u-boot prompt using TFTP commands.

To install the SW files via TFTP do the following steps:

- Open HyperTerminal on the PC. Connect a USB cable to the debug port (Micro USB port with build in RS232 to USB converter).
- Power up the board.
- Open hyper terminal application; select the virtual com detected by the PC.
 - Set the rate to 115,200 Bps, data – 8 bits, party – none, stop – 1bit and flow control – none.
- Enable a TFTP server the host PC.
- Connect the Ethernet port of the board to the network

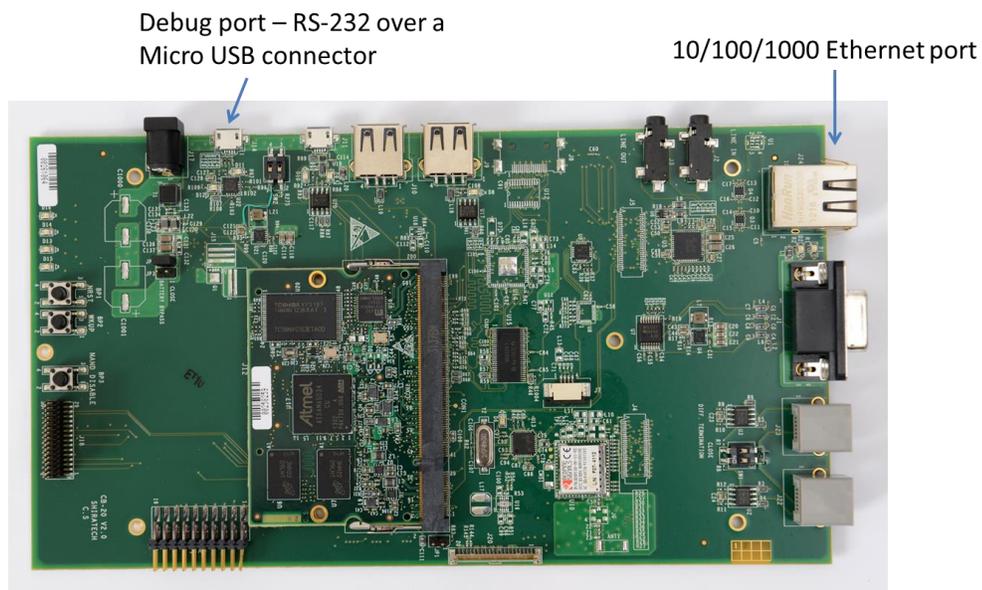


Figure 8 – Connections for TFTP

- Configure the IP address of host, IP address of board and Ethernet address of board by using

u-boot commands as follows (The IP addresses can varied according to the setup used)

```
u-boot > setenv serverip 10.0.0.10
```

“10.0.0.10” is IP address of host PC.

```
u-boot>setenv ipaddr 10.10.10.5
```

“10.10.10.5” will be IP address of board.

```
u-boot>setenv ethaddr 12:34:56:78:9a:bc
```

12:34:56:78:9a:bc will be the MAC address used.

```
u-boot>saveenv
```

Save the parameters in the Flash.

- In order to check the configuration you can use the **printenv** command.
- A reset should be before starting the TFTP process.

Note: that once the MAC address is set it cannot be overwrite through the u-boot menu.

Kernel and device tree Installation

- Put the file “**ulmage**” (Kernel) and the “**sama5d34ek.dtb**” (device tree) on host PC on which TFTP is configured. Disable firewall in host pc.
- Run the following commands from u-boot prompt
- For the device tree file:

```
u-boot> tftp 0x21000000 sama5d34ek.dtb
```

```
u-boot> nand erase 0x180000 0x1FFFFFF
```

```
u-boot> nand write 0x21000000 0x180000 0x50E1 (size of the file – this value can be taken for the results of the TFTP action)
```

- For the Kernel file ulmage:

```
u-boot> tftp 0x22000000 ulmage
```

```
u-boot> nand erase 0x200000 0x600000
```

```
u-boot> nand write 0x22000000 0x200000 0x250000 (size of the file – this value can be taken for the results of the TFTP action)
```

- Configure the booting arguments.

```
u-boot> setenv bootcmd 'nand read 0x21000000 0x180000 0x50E1; nand read 0x22000000 0x200000 0x26A1A8; bootm 0x22000000 - 0x21000000'
```

Note: In the Shiratech U-boot version this parameter is already defined

```
u-boot>saveenv
```

- Boot the Linux kernel using following command.

```
u-boot> boot
```

After boot it will show error proper file system couldn't be mounted, since the file system was not installed.

FILE SYSTEM Installation

To install the Debian file system follow the following steps:

- Put the compressed file system image “**rootfs.ubi**” in tftpboot folder of host PC (the file name should be the valid file used). The following steps are used to install Debian file system from boot loader prompt.

```
u-boot> tftp 0x22000000 rootfs.ubi
```

```
u-boot> nand erase 0x800000 0xf800000
```

```
u-boot> nand write.trimffs 0x22000000 0x800000 0x3860000 (size of the file – can be  
view in the results of the TFTP download, give the file size in bytes)
```

Reboot the board. After reboot you will get a Linux login prompt.

2.5.1.1 Installation via SAM-BA

For downloading the Kernel, device tree and file system via the SAM-BA follow the steps detailed in 2.2.1 for connecting the SAM-BA then:

1. Choose the NandFlash media tab in the SAM-BA GUI.
2. Initialize the media choosing the Enable NandFlash action in the Scripts rolling menu and press Execute.
3. ***If version 1.4 or higher is used skip this step.***

Select Enable OS PMECC parameters in the Scripts rolling menu and press Execute. The default ECC configuration should be ok(note in patch 5 the default has change the configuration should be as viewed in 2.3.2.2). Then press the OK button.

- To download the Kernel image:
 1. To erase only the kernel image into the NAND flash, type this command after the SAM-BA prompt:

```
NANDFLASH::EraseBlocks 0x200000 0x600000
```

2. Enter the proper address on media in the Address text field. The value for the kernel image is 0x200000.
3. Press Send File Name browse button and choose your kernel image.

4. Press Send File button.

- For download the device tree file:

1. To erase only the device tree into the NAND flash, type this command after the SAM-BA prompt:

NANDFLASH::EraseBlocks 0x180000 0x1FFFFFF

2. Enter the proper address in the Address text field. The value for the device tree is 0x180000.

3. Press Send File Name browse button and choose your device tree binary file.

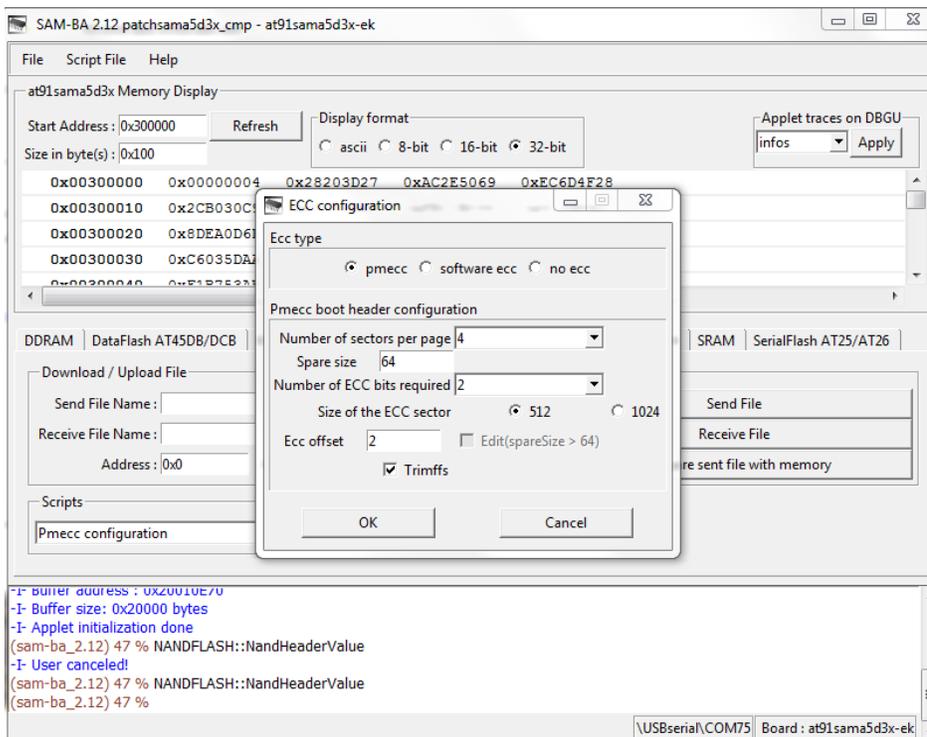
4. Press Send File button.

- For download the file system:

1. To erase only the NAND flash rootfs partition, type this command after the SAM-BA prompt:

NANDFLASH::EraseBlocks 0x800000 0xFFFFFFFF

2. Select the Pmecc configuration option and check the trimfss option



3. Press Send File Name browse button and choose your UBI file system.

4. Enter the proper address on media in the Address text field. The value for the file system is 0x800000.

4. Press Send File button. Wait for the end of the flashing process.

5. Close SAM-BA, remove the USB cable.

6. Restart the system and set through the U-boot menu the boot arguments according to the

following paragraph if needed.

2.5.1.2 Boot command and arguments for NAND

Boot Command for **NAND**:

```
setenv bootcmd 'nand read 0x21000000 0x180000 0x50E1; nand read 0x22000000 0x200000 0x283D70; bootm 0x22000000 - 0x21000000'
```

Boot argument for **jffs2**:

```
setenv bootargs = "console=ttyS0,115200 mtdparts=atmel_nand:8M(bootstrap/uboot/kernel)ro,-(rootfs) root=/dev/mtdblock1 rw rootfstype=jffs2 :rootfs"
```

Boot argument for **ubi**:

```
setenv bootargs = "console=ttyS0,115200 mtdparts=atmel_nand:8M(bootstrap/uboot/kernel)ro,-(rootfs) rw rootfstype=ubifs ubi.mtd=1 root=ubi0:rootfs"
```

2.5.2 Using SD CARD

The SD card slot is located on the bottom of the CB-20. The slot is for Micro SD format only.

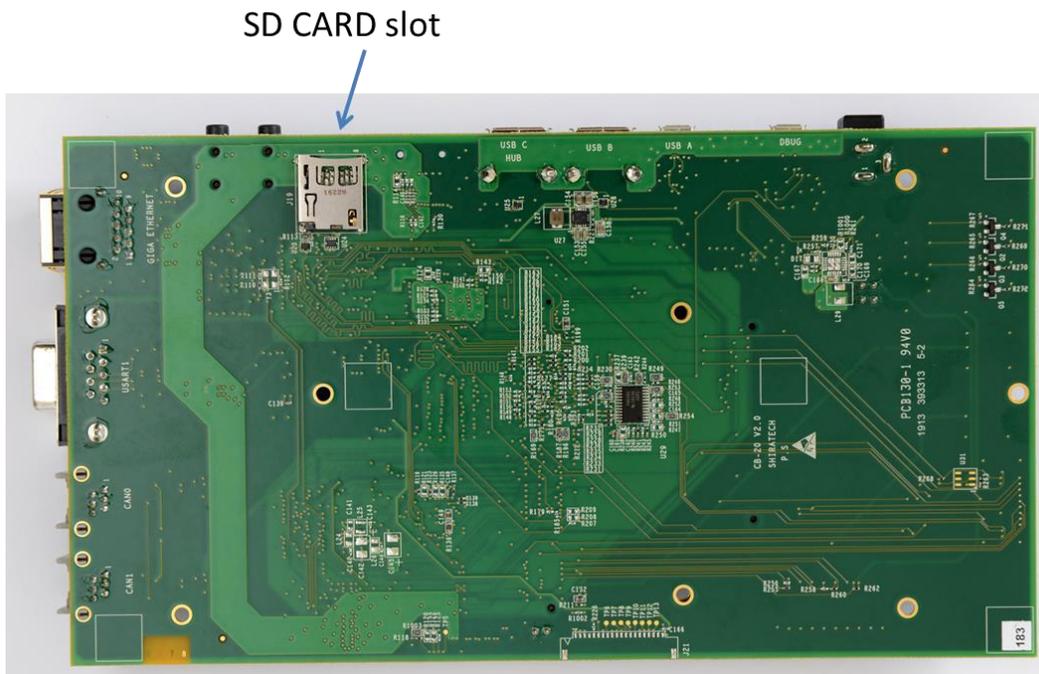


Figure 9 – SD card slot

The SD card should be divided to two partitions one relatively small for the boot, u-boot, device tree and kernel, should be in FAT 16 or 32 format. **For proper operation the file names should be boot, u-boot, ulmage and dtb.**

The other partition will be ext and will be used for the file system.

The following link provides details on how to create two partitions on the SD:

<http://www.at91.com/linux4sam/bin/view/Linux4SAM/SDCardBootNotice>.

Note: it is recommended to use ext3 or ext4 only.

3 SW version creation and modification

The following section describe how to compile the various SW parts, how to modify the file system and how to add user applications.

The following process was done on a PC with Ubuntu 10.04 and 12.04.

3.1 Cross Compilation

The following paragraphs describes how build the environment for cross compiling along with instruction on how to cross compile each part of the SW

3.1.1 Introduction

The Linux platform includes several parts Bootstrap, boot loader(u-boot) Kernel and file system. The AT91Bootstrap does some minimal initialization of SDRAM and clock. Then it will check the presence of boot loader (u-boot). The Boot loader will initialize UART (serial) and Ethernet. Then u-boot will check the presence of Linux kernel image. If it is present u-boot will load it from flash to DDR and starts execution. If Linux kernel image is not present, u-boot prompt will be shown e.g: **u-boot>** along with an error message that the kernel is not found.

The U-boot provides a rich set of commands to get hardware information, downloading images to memory, writing images to flash etc.

The Linux kernel will start execution and it will load all device drivers. Then Linux kernel will search for a root file system in compressed format on flash. The file system provided here is based on Debian.

3.1.2 Setting Cross Compilation Environment

For setting up the cross compilation environment see the following link for exact details.

<http://eewiki.net/display/linuxonarm/SAMA5D3#SAMA5D3-ARMCrossCompiler:GCC>

Note that that the cross compile process is different for V1.4 and V1.5, the following paragraph provides description for each version.

3.1.3 Cross Compiling V1.5

3.1.3.1 Cross Compiling Bootstrap

The following steps will lead you how to cross compilation a Bootstrap:

Set the cross compilation environment as per the steps in section 2.6.2. Download the bootstrap source code from ShiraTech site. Before building Bootstrap, you have to configure it for 5series boards and to indicate where you want to store the environment. The bootstrap can be built to be used in an internal NAND or in an SD.

For creating Bootstrap for SD card:

```
$ cd at91bootstrap/  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- distclean  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-  
sama5d3xeksd_uboot_defconfig  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
```

For creating Bootstrap for NAND flash:

```
$ cd at91bootstrap/  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- distclean  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-  
sama5d3xeknf_uboot_defconfig  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
```

The result of this operation is a new Bootstrap binary located in the "binaries" directory and called *at91sama5d3xek-xxxboot-uboot-3.6.2.bin*.

To Customize the Bootstrap configuration

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
```

3.1.3.2 Cross Compiling U-Boot

For creating a U-boot configuration for SD-CARD

```
$ cd u-boot-at91/  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- distclean  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- sama5d3xek_sdcard  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
```

For creating a U-boot configuration for NAND flash

```
$ cd u-boot-v2012.10-sama5d3_M5_v1.4/  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- distclean  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-  
sama5d3xek_nandflash  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
```

The result of this operation is a fresh u-boot binary called *u-boot.bin* in the same directory.

3.1.3.3 Boot arguments setting for u-boot

This u-boot version has support for bootz command which is for supporting zImage. In order to set zImage for booting instead of ulmage set the following arguments should be set via the u-boot prompt:

For SD-CARD

```
setenv bootcmd "mmcinfo;fatload mmc 0:1 0x21000000 dtb;fatload mmc 0:1 0x22000000 zImage;bootz 0x22000000 - 0x21000000"
```

```
setenv bootargs "console=ttyS0,115200 earlyprintk root=/dev/mmcblk0p2 rw rootdelay=2"
```

```
saveenv
```

For NAND

```
setenv bootcmd "nand read 0x21000000 0x180000 0x80000;nand read 0x22000000 0x200000 0x600000;bootz 0x22000000 - 0x21000000"
```

```
saveenv
```

Ethernet port selection for D36

When using an AT-501 with D36 which has two Ethernet ports, use following to choose which one to use:

```
setenv ethact macb0,gmacb0
setenv ethprime gmacb0
```

3.1.3.4 Cross Compiling Linux Kernel

The following process is used for configure and build the Kernel.

Note for 1 wire device support and for audio support use the following read me files available at the Shiratech web site:

- [ONEWIRE_RV1.5_readme.txt](#)
- [README_enabling_audio_kernel.txt](#)

```
$ cd linux-at91/
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- distclean
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- sama5_defconfig
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- sama5d31ek.dtb
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- sama5d34ek.dtb
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- sama5d35ek.dtb
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- sama5d36ek.dtb
```

The result of this operation is kernel images *zImage* will be created in the directory: "arch/arm/boot/" and *sama5d31ek.dtb*, *sama5d34ek.dtb*, *sama5d35ek.dtb*, *sama5d36ek.dtb* will be created in the directory: "arch/arm/boot/dts/"

3.1.3.5 UBI Filesystem

To create UBIFS File-system for 256MB NAND-FLASH:

```
$ mkfs.ubifs -m 2048 -e 126976 -c 244682752 -r rootfs_dir_location/ -x lzo -o rootfs.img --max-leb-cnt=1170
```

The result of this operation is UBI file-system created in the name *rootfs.img*

To create UBI file:

Copy the script "ubinize.cfg" to the directory where rootfs.img generated from previous step, for 256MB NAND-FLASH use the following command:

```
$ ubinize -o rootfs.ubi -p 131072 -m 2048 -s 2048 -O 2048 -e 3 ubinize.cfg
```

The result of this operation is creation of rootfs.ubi

3.1.4 Cross Compiling V1.4

3.1.4.1 Cross Compiling Bootstrap

The following steps will lead you how to cross compilation a Bootstrap:

Set the cross compilation environment as per the steps in section 2.6.2. Download the bootstrap source code from ShiraTech site. Before building Bootstrap, you have to configure it for 5series boards and to indicate where you want to store the environment. The bootstrap can be built to be used in an internal NAND or in an SD.

For creating Bootstrap for SD card:

```
$ cd at91bootstrap/  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- distclean  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-  
at91sama5d3xeksd_uboot_defconfig  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
```

Note: For proper operation the file should be renamed to boot.bin and should be located in the FAT part of the SD.

Configure and build the Bootstrap source for NAND:

```
$ cd at91bootstrap/  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- distclean  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-  
at91sama5d3xeknf_uboot_defconfig  
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
```

The result of this operation is a new Bootstrap binary located in the "binaries" directory and called *at91sama5d3xek-xxxboot-uboot-3.4.bin*.

To Customize the Bootstrap configuration

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf- menuconfig
```

3.1.4.2 Cross Compiling U-boot

The following steps will lead you how to cross compilation a U-boot:

Set the arm cross compilation environment as per the steps in section 2.4.2. Download the u-boot source from ShiraTech site and extract the source code from it. Perform the following process for either NAND or SD version:

Configure and build the U-boot source for SDCARD:

```
$ cd u-boot-v2012.10-sama5d3_M5_v1.4/
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf- distclean
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf-
at91sama5ek_sdcard_config
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf-
```

Configure and build the U-boot source for NAND:

```
$ cd u-boot-v2012.10-sama5d3_M5_v1.4/
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf- distclean
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf-
at91sama5ek_nandflash_config
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf-
```

The result of this operation is a fresh u-boot binary called u-boot.bin in the same directory.

3.1.4.3 Cross Compiling Linux Kernel

The following steps will lead you how to cross compilation a Linux Kernel:

Set the arm cross compilation environment as per the steps in section 2.4.2.

Download the Linux kernel source from ShiraTech site. Extract the sources and perform the following steps for generating new kernel:

Configure and build the Kernel source for SDCARD

```
$ cd kernel-source-rv_1.4
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf- distclean
```

Then, copy the file "at91sama5ek_sdcard_config" in the directory "kernel_config_rv_1.4" to the kernel source directory "kernel-source-rv_1.4" as ".config"

```
$ cp -v kernel_config_rv_1.4/at91sama5ek_sdcard_config .config
```

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf- menuconfig
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf- uImage
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf- sama5d31ek.dtb
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf- sama5d34ek.dtb
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf- sama5d35ek.dtb
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf- sama5d36ek.dtb
```

Configure and build the Kernel source for NAND

```
$ cd kernel-source-rv_1.4
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf- distclean
```

Then, copy the file "at91sama5ek_nand_config" in the directory "kernel_config_rv_1.4" to the kernel source directory "kernel-source-rv_1.4" as ".config"

```
$ cp -v kernel_config_rv_1.4/at91sama5ek_nand_config .config
```

When building for 1GB or 2GB ONFI NAND flash for example based on D34:

```
$ cp -v kernel_patch/sama5d3cm_d34_2gb.dtsi
arch/arm/boot/dts/sama5d3cm.dtsi
```

When building for SOMs with 256MB NAND flash

```
$ cp -v kernel_patch/sama5d3cm.dtsi arch/arm/boot/dts/sama5d3cm.dtsi

$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf- menuconfig
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf- uImage
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf- sama5d31ek.dtb
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf- sama5d34ek.dtb
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf- sama5d35ek.dtb
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-hf- sama5d36ek.dtb
```

The result of this operation is kernel images ulmage, sama5d31ek.dtb, sama5d34ek.dtb, sama5d35ek.dtb, sama5d36ek.dtb created in the directory "arch/arm/boot/"

3.1.4.4 Cross Compiling File system

To create UBIFS File-system use the following process according to the target required:

For 256MB NAND-FLASH:

```
$ mkfs.ubifs -m 2048 -e 126976 -c 244682752 -r rootfs_dir_location/ -
x lzo -o rootfs.img --max-leb-cnt=2000
```

For 2GB NAND-FLASH:

```
$ mkfs.ubifs -m 4096 -e 516096 -c 244682752 -r rootfs_dir_location/ -
x lzo -o rootfs.img --max-leb-cnt=2000
```

The result of this operation is UBI file-system created in the name rootfs.img

To create UBI File-system use the following process according to the target required:

Copy the script "ubinize.cfg" to the directory where rootfs.img generated from previous step and run the following:

For 256MB NAND-FLASH:

```
$ ubinize -o rootfs.ubi -p 131072 -m 2048 -s 2048 -O 2048 -e 3
ubinize.cfg
```

For 2GB NAND-FLASH:

```
$ ubinize -v -o rootfs.ubi -p 524288 -m 4096 -s 2048 -O 4096 -e 3
ubinize.cfg
```

The result of this operation is creation of rootfs.ubi

3.2 Running User Applications on ShiraTech Board

The following paragraph explains how a user can compile a C or JAVA coded application.

3.2.1 Cross compiling C code

Follow the below mentioned steps to cross compile C source code application:

Follow the steps mentioned in section 3.1.1 (Setting Cross Compilation Environment) to install the toolchain before compiling the application.

Create a sample C program. For e.g. source.c

Compile the program using:

```
$arm-linux-gcc source.c -o source_output -static
```

Then the result will be a binary named "source_output".

Transfer the binary using one of the following options:

Using scp, transfer the source_output binary to the file-system:

```
$scp -r source_output root@<shiratech_board_IPaddress>:/root/
```

Using tftp, transfer the source_output binary to the file-system using,

```
$ tftp -g -r source_output tftp_server_ip
```

Give execution permission to "source_output" using:

```
$chmod +x source_output
```

Run the binary using:

```
$./source_output
```

3.2.2 Cross compiling Java code

Follow the below mentioned steps to cross compile Java source code application

For compiling java applications, we need java jdk version 1.6.0_24 in Ubuntu Linux machine

Install java jdk version 1.6.0_24, using below command.

```
$ sudo apt-get install openjdk-6-jdk
```

Use “java -version” command to get the current jdk version.

Compile the java application using

```
$ javac Source_name.java
```

The result of the above command will get like

```
$ Source_name.class
```

There are two options to transfer the application

Using

```
$ scp -r source_output root@<shiratech_board_IPaddress>:/root/
```

Using

```
$ tftp -g -r Source_name.class tftp_server_ip
```

Run java application using,

```
$ java Source_name
```

4 Utilizing the eMMC device

The EMMC device is connected to MCI0 interface of the SAMA5D3x which is the only MCI interface which can be used as a bootable interface.

The eMMC device can be format and programed via the following options:

1. Using an SD for boot and running RAM based boot for programming the eMMC.
2. Using SAM-BA.
3. Using NAND flash.

The following paragraphs provide the process required for each option.

4.1 USING RAMDISK & TFTP

An alternative option is to use a combination of booting from an SD card, starting a RAM based program, and handling the eMMC using the RAMdisk software. The process is done as follows:

4.1.1 Creating ramdisk.gz in host Linux PC

Copy the script "rootfs_compress_create.sh" (available on the Shiratech website)to the location where the "rootfs" directory is placed. Modify the "ROOTFS_DIR" variable within the script to your rootfs location. Run the script with root privileges as follows (the script can be downloaded from the Shiratech site):

```
$ sudo ./rootfs_compress_create.sh
```

This will generate "ramdisk" file. Compress the ramdisk file as follows:

```
$ gzip -c ramdisk | dd of=ramdisk.gz
```

Unmount and remove the directory "rootfs-loop".

```
$ umount rootfs-loop  
$ rm -rv rootfs-loop
```

Copy the kernel images and ramdisk.gz to the TFTP access location.

Note: A Ready-to-use RAMdisk is also available in the Shiratech web site.

4.1.2 Boot from SD-CARD and mount compressed ramdisk

Boot from SD-CARD and stop at the in u-boot prompt by pressing any key. Set the following environment variables in u-boot as following:

```
# setenv ethaddr 12:34:56:78:9a:bc (eth1addr for the FE port)  
# setenv eth1addr 12:34:56:78:9a:bb (for the GE port)  
# setenv serverip "tftp server ip address"  
  Ex : setenv serverip 10.0.0.23  
# setenv ipaddr "for board ip" (for FE port)  
  EX : setenv ipaddr 10.0.0.2  
# setenv memboot "setenv bootargs console=ttyS0,115200  
ramdisk_size=20480 root=/dev/ram0 rw initrd=0x23400000,20M"
```

```
# setenv ramboot "run memboot;tftp 0x22000000 zImage;tftp
0x21000000 dtb;tftp 0x23400000 ramd.gz;bootz 0x22000000 -
0x21000000"
```

In case a D36 module is used the following bootarg should be added in order to enable TFTP via the GE port.

```
setenv ethact macb0,gmacb0
setenv ethprime gmacb0
```

```
# saveenv
# run ramboot
```

At this stage **you must eject the SD card** to enable the software to detect the eMMC device. After the software has come up, the eMMC device will appear as `/dev/mmcblk0` and it will be possible to format it and download files to it.

The following paragraph provide a sample script for formatting the eMMC to two partitions and for copying the content out of a USB memory device, formatted to ext3 format.

copy necessary kernel and boot images in to the usb device as mentioned above(including `sdheader_512.bin`). Mount the USB dongle using teh following command:

```
mount -t vfat /dev/sda1 /mnt/usb
```

Run the following script:

```
# ./emmc_upgrade.sh
```

script structure:-

```
# Shell script to format eMMC with partition 1 of type FAT and
# partition 2 of type ext3
# Assume eMMC may contain 2 partions and delete them
(echo d; echo 1; echo d; echo d; echo w) | fdisk /dev/mmcblk0
# Create partition 1 of type FAT
(echo n; echo p; echo 1; echo 4; echo +64M; echo t; echo c; echo a;
echo 1; echo w) | fdisk /dev/mmcblk0
# Create partition 2 of type ext3
(echo n; echo p; echo 2; echo 1958; echo 118016; echo t; echo 2; echo
83; echo w) | fdisk /dev/mmcblk0
```

```
mkfs.vfat          /dev/mmcblk0p1
mke2fs_ext3 -j    /dev/mmcblk0p2
```

```
#/dev/mmcblk0p1 is the ist partion in emmc ,mount it on the fat
directory created under /mnt.
```

```
mount -t vfat /dev/mmcblk0p1 /mnt/fat
```

```
#/dev/mmcblk0p2 is the 2nd partion in emmc ,mount it on the ext3
directory created under /mnt.
mount -t ext3 /dev/mmcblk0p2 /mnt/ext3

#/dev/sda1 is external usb device, mount it on the usb directory
created under /mnt.
mount -t ext3 /dev/sda1 /mnt/usb

#copy the respective boot files in the fat partition(mmcblk0p1) of
emmc and rootfilesystem in ext partition (mmcblk0p2) of emmc.
cp -vr /mnt/usb/dtb /mnt/fat
cp -vr /mnt/usb/u-boot.bin /mnt/fat
cp -vr /mnt/usb/boot.bin /mnt/fat
cp -vr /mnt/usb/zImage /mnt/fat
cp -vr /mnt/usb/rootfs/* /mnt/ext3

#dd if=/mnt/usb/sdheader_512.bin of=/dev/mmcblk0

#Finally unmount all mounted devices..
umount /mnt/fat
umount /mnt/usb
umount /mnt/ext3
```

NOTE: Uncomment the line `#dd if=/mnt/usb/sdheader_512.bin of=/dev/mmcblk0` from the script in case facing any issue from booting through eMMC. That command will write the header section ahead of the eMMC partitioning.

4.2 Formatting the eMMC device using SAM-BA

The SAM_BA enables detecting eMMC on AT-501 SOM as USB Mass Storage Device, which means that using the regular PC tools you can format and program the eMMC device.

The process is as follows:

- Disable NAND and connect the board to SAM-BA(version 2.15 or above)
- Choose "DDR" media tab in the SAM-BA GUI and Initialize the media choosing the "Enable DDR" action in the scripts rolling menu and press "Execute"
- Press "Send File Name" browse button and choose the USB Mass storage example binary provided by Atmel "usb_masmstorage_sama5d3x_ek_sama5d3x-ddram.bin" (can be downloaded also from Shiratech site.
- Check the address on the media in the Address text field is "0x20000000"

- Press "Send File" button
- In the scripts dialogue to run the codes, run the following command:

```
go 0x20000000
```

Disconnect the USB cable from PC, close the SAM-BA tool and connect it back. An Atmel USB Mass storage device will be detected by the PC enabling full access to the eMMC device.

4.3 Using NAND flash

In case the AT-501 is equipped with both NAND flash and eMMC, the NAND flash can be programmed via the SAM-BA tool. Once booting out of the NAND the eMMC device will be detected and all the operations to format and load the eMMC can be done via the Linux prompt, see 2.1.2 for the required process and commands.

5 CB-20 Carrier board description

5.1 Overview

The CB-20 is a fully featured development and carrier board for ShiraTech's AT-501 system on modules. The board is used by software developer as a development platform with all relevant peripherals for simulating the target product functionalities. It can also be used as a reference design for hardware teams or as a ready for use control and display unit to be integrated in commercial products.

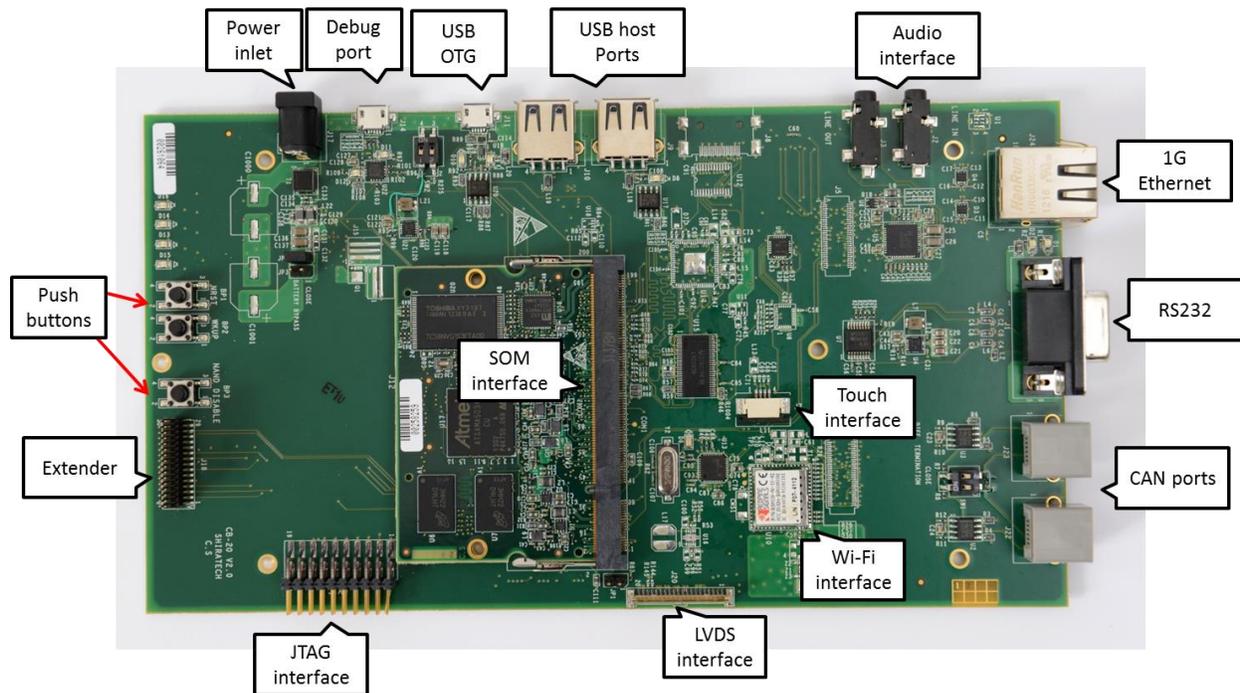


Figure 1 – CB-20 interfaces

Notes:

1. The debug port includes a build in RS-232 to USB converter. It can be connected to the PC using the Micro USB to USB cable supplied with the evaluation board. The board can be controlled using a hyper terminal application over USB. The port should be configured to the following parameters:

Rate to 115,200 Bps, data – 8 bits, parity – none, stop – 1bit and flow control – none.

2. The USB OTG port can be used for SAM-BA connectivity.

3. The JTAG interface can be used for connecting the **AT91SAM-ICE**. The jumper, JP-1, select between JTAG testing ("0") and ICE or normal operation mode.

Note: For both normal operation and ICE it should be left open.

5.2 CB20 interfaces mapping

The CB-20 offers a large variety of interfaces for supporting a large variety of applications. The following table provides mapping between the various interfaces and the internal ports of the CPU.

Note: On the PS (there is a label for each connector)

Interface	Connector name	Processor interface	Remarks
Debug port	J-14	PB-30,PB31	Micro USB Type A/B. An internal RS232 to USB converter by FTDI.
USB OTG	J-11	HHSDMA, HHSDPA	USB 2.0 Type-A Host/Device
USB host port 1	J-10	HHSDMB, HHSDPB	USB 2.0 Type-A Host
USB host port 2	J-9	HHSDMC, HHSDPC	USB 2.0 Type-A Host Interface from USB HUB
HDMI	J-8	LCD Interface	19 pins Type-A Receptacle
Audio interface 1	J-3	Audio Interface	Stereo Out
Audio interface 2	J-2	Audio Interface	Stereo Line in
Giga Ethernet port	J-24	Ethernetg Port 0	Rj-45
RS232 port	J-1	USART 1	DB-9
CAN interface 0	J-23	CAN0	RJ-11
CAN interface 1	J-22	CAN1	RJ-11
JTAG interface	J-16		20 pins 2.54 Header
Extender	J-18	SPI-1, USART-3, I2C from Extender, 10 GPI	30 pins 1.25 Header
Power inlet	J-17		Power Jack, 5-9 Volt

Table 1 – External interfaces mapping

Notes:

USB support:

- Port-C supports USB 2.0 Host is connected via a USB-HUB device to the external interface.
- The control and monitor of the USB is done using GPIOs listed in 4.3.

5.3 Interrupt & I/O Table

The following table describes the AT-501 I/O configuration used by the SW. To keep compatibility with the AT-501 SW these I/O must be used for the following interfaces:

Signal	I/O	Description	Remarks
E0 INTR	PB-25	Giga Ethernet 0 interrupt	Active Low
E1 INTR	PE-30	Fast Ethernet 1 interrupt	Active Low
MCI0 CD	PD-17	uSD card detect	0 – Card in 1 – No card
MCI1 CD	PD-18	SD card detect	0 – Card in 1 – No card
VBUS Sense	PD-29	USB port A power sense	0 – No power sensed 1 – Power sensed
OverCur USB	PD-28	USB port A or Port B over current	Open Drain
EN5V HDA#	PD-25	USB port A power drive enable	0 – Enable 1 – Disable (Default)
EN5V HDB#	PD-26	USB port B power drive enable	0 – Enable 1 – Disable (Default)
EN5V HDC#	PD-27	USB port C power drive enable	0 – Enable 1 – Disable (Default)

Table 2 – Configured I/O and Interrupts

NOTE – More interrupts can be available using the SAMA5D I/O pins can be configured according to the user application.

NOTE – When an interface is not in use its relevant I/O can be used as a general purpose I/O.

5.4 Jumpers

The CB-20 holds several jumpers for various applications:

JP-1 **JTAG enable** – if open Normal operation, JTAG emulator can be connected. If closed JTAG mode is enabled.

JP-2, JP-3 – **Battery bypass** - If closed normal operation, open Battery support

Note -that it is enough that one of them will be closed for normal operation.