APPLICATION NOTE

U-Boot Design for SAMA5D3/D4 Based Board

Atmel | SMART SAMA5D3/D4 Series

Scope

The Atmel® | SMART SAMA5D3 and SAMA5D4 series are high-performance, power-efficient embedded MPUs based on the ARM® Cortex®-A5 processor.

Atmel provides Linux® support on the SAMA5D3/D4. This application note introduces how to use U-Boot (Universal Boot Loader) as the first step in starting Linux, provides a brief guide on how to modify the U-Boot to adapt the customized board based on SAMA5D3/D4, and shows some key functions of U-Boot, which are useful in debugging the hardware level drivers and Linux kernel.

This document gives the main tracks to add support for a customized board in U-Boot.

Reference Documents

- SAMA5D3 Series Datasheet (Atmel literature No. 11121)
- SAMA5D3 Series Evaluation Kit User Guide (Atmel literature No. 11180)
- SAMA5D3 Xplained User Guide (Atmel literature No. 11269)
- SAMA5D4 Series Datasheet (Atmel literature No. 11238)
- SAMA5D4 Series Evaluation Kit User Guide (Atmel literature No. 11294)
- SAMA5D4 Xplained User Guide (Atmel literature No. 44005)

1. U-Boot Project Overview

U-Boot (Universal Boot Loader) is an open source boot loader for hardware platforms. U-Boot is widely used for embedded systems and supports many processor architectures including ARM. Users can easily configure U-Boot to strike the right balance between a rich feature set and a small binary footprint. Users also can port U-Boot onto different development boards with the processors that U-Boot supports.

U-Boot project is an open source project and is compliant with General Public License (GPL). Any project that uses U-Boot should also abide by the GPL.

1.1 How to Get and Compile the U-Boot Source Code

All relative documentation about U-Boot can be found on the official website www.denx.de.

The U-Boot source code is put in Git repository: git.denx.de.

Customers can download the U-Boot source code with the dedicated Atmel ARM eMPU support from https://github.com.

The sequence of steps to download the U-Boot source code is the following:

```
$ git clone git://github.com/linux4sam/u-boot-at91.git
Cloning into 'u-boot-at91'...
remote: Counting objects: 264544, done.
remote: Compressing objects: 100% (48990/48990), done.
remote: Total 264544 (delta 210999), reused 264544 (delta 210999)
Receiving objects: 100% (264544/264544), 65.04 MiB | 816 KiB/s, done.
Resolving deltas: 100% (210999/210999), done.
$ cd u-boot-at91/
```

The source code in the GitHub is updated frequently to support the latest Atmel ARM eMPU products.

1.2 Source Code Structure Overview

After downloading the U-Boot code, the user can find many folders and files in this project. Table 1-1 provides a brief introduction to the top level directory.

Table 1-1. Introduction of Top Level Directory

Directory	Description
api	U-Boot machine/arch independent API for external applications
arch	Architecture specific files such as ARM, ARV32, MIPS, etc.
board	Board dependent files or directories
common	Misc architecture independent functions
disk	Code for disk drive partition handling
doc	Basic documentation files
drivers	Device drivers for common peripherals
dts	Contains Makefile for building internal U-Boot fdt.
examples	Example code for standalone applications
fs	Common file system support
include	Header files
lib	Files generic for all architectures
net	Networking support (bootp, tftp, rarp, nfs, and so on)

Table 1-1. Introduction of Top Level Directory (Continued)

Directory	Description
post	Power on self-test
tools	Tools for building s-record files, U-Boot image, and so on
boards.cfg	Board configuration file
README	README file where users can found more information

The folders and files dedicated for SAMA5D3/D4 are also introduced, as shown in Table 1-2.

Table 1-2. Introduction of Directories/Files Dedicated for SAMA5D3/D4

Directory/File	Description
board/atmel	Include all configuration files for Atmel evaluation boards
board/atmel/sama5d3xek/sama5d3xek.c board/atmel/sama5d3_xplained/sama5d3_xplained.c	SAMA5D3 Evaluation Kit and processor initialization
board/atmel/sama5d4ek/sama5d4ek.c board/atmel/sama5d4_xplained/sama5d4_xplained.c	SAMA5D4 Evaluation Kit and processor initialization
arch/arm/cpu/armv7/at91/cpu.c	Set up CPU with interrupts, stack, MMU and cache, etc.
arch/arm/cpu/armv7/at91/clock.c	Clock initialization and setting
arch/arm/cpu/armv7/at91/reset.c	Reset CPU operation
arch/arm/cpu/armv7/at91/timer.c	Timer and counter register settings
arch/arm/cpu/armv7/at91/sama5d3_devices.c	SAMA5D3 relative peripherals initialization and settings
arch/arm/cpu/armv7/at91/sama5d4_devices.c	SAMA5D4 relative peripherals initialization and settings
arch/arm/include/asm/arch-at91	Include all the Atmel CPUs and the registers and functions definition for relative peripherals
include/configs/sama5d3xek.h include/configs/sama5d3_xplained.h	Include variable definitions for SAMA5D3
include/configs/sama5d4ek.h include/configs/sama5d4_xplained.h	Include variable definitions for SAMA5D4
drivers	In each subfolder under this directory, there is an Atmel dedicated driver named atmel_xxx.c

2. Create a New Board Based on SAMA5D3 (Example)

The U-Boot version that Atmel provides supports the Atmel SAMA5D3 Series Evaluation Kit (SAMA5D3x-EK), SAMA5D3 Xplained Evaluation Kit, SAMA5D4 Series Evaluation Kit (SAMA5D4-EK) and SAMA5D4 Xplained Ultra Evaluation Kit. If users make their own boards based on SAMA5D3/D4 and want U-Boot to work on these boards, U-Boot modifications must be made. Here we take SAMA5D3 as an example.

The steps to create a new board based on SAMA5D3 are the following:

1. Clean the source code tree.
   ```
   make distclean
   ```
2. Copy the contents of the *sama5d3xek* folder to the new folder for the user board named such as *sama5d3xek_ctm* under the same directory *board/atmel*.
   ```
   cp –r sama5d3xek sama5d3xek_ctm
   ```
3. In the new folder *sama5d3x_ctm*:
   3.1. Change file name: *sama5d3xek.c* to *sama5d3xek_ctm.c*
   ```
   mv sama5d3xek.c sama5d3xek_ctm.c
   ```
   3.2. Open *Makefile*, change *sama5d3xek.o* to *sama5d3xek_ctm.o*
   – Use “Vi” command for editing
4. Create a new header file named such as *sama5d3x_ctm.h* under the directory *include/configs*. Copy the content of the *sama5d3x.h* file into the new header file.
   ```
   cp sama5d3xek.h sama5d3xek_ctm.h
   ```
5. Go the top level directory, open the *boards.cfg* file and add the following lines for the new board:
   ```
   Active arm   armv7 at91  atmel sama5d3xek_ctm     sama5d3xek_ctm_mmc
   sama5d3xek_ctm:SAMA5D3,SYS_USE_MMC
   Active arm   armv7 at91  atmel sama5d3xek_ctm     sama5d3xek_ctm_nandflash
   sama5d3xek_ctm:SAMA5D3,SYS_USE_NANDFLASH
   Active arm   armv7 at91  atmel sama5d3xek_ctm     sama5d3xek_ctm_spiflash
   sama5d3xek_ctm:SAMA5D3,SYS_USE_SERIALFLASH
   ```
6. Build the corresponding board configuration according to the binary that you want to generate:
   ```
   make sama5d3xek_ctm_nandflash_config; //The binary is for NAND Flash boot
   make sama5d3xek_ctm_spiflash_config; //The binary is for serial Flash boot
   make sama5d3xek_ctm_mmc_config; //The binary is for SD card boot
   ```
7. Compile the project and generate the binary *u-boot.bin*:
   ```
   make CROSS_COMPILE=<path_to_cross-compiler/cross-compiler-prefix->
   ```
   Note: *path_to_cross-compiler* is only needed if it is not in your PATH.
   Usually *cross-compiler-prefix-* looks like *arm-linux-* or *arm-elf-*.

2.1 Customize the Key Components

Based on the new board folders, there will be some key components such as System Clock, NAND Flash, and Serial Flash, that need to be initialized before the system boots up.

Open the *include/configs/sama5d3xek_ctm.h* file to display definitions of components. For example, the System Clock definition is provided as shown below:

```
/* ARM asynchronous clock */
#define CONFIG_SYS_AT91_SLOW_CLOCK      32768
#define CONFIG_SYS_AT91_MAIN_CLOCK      12000000 /* From 12 MHz crystal */
```

And the base address for binary is defined as below:

```
#define CONFIG_SYS_TEXT_BASE          0x26f00000
```

The user can set the slow clock and on-board crystal value according the customized board.

Note: If the clock is not correctly configured, the system cannot boot successfully.

If a NAND Flash module is used, a NAND Flash driver must be included. Firstly, ensure that the *include/configs/sama5d3xek_ctm.h* file includes the line *#define CONFIG_CMD_NAND*.

If the NAND Flash is defined, configure the NAND Flash settings listed below according to the customized board:

```
#ifdef CONFIG_CMD_NAND
#define CONFIG_NAND_ATMEL
#define CONFIG_SYS_MAX_NAND_DEVICE          1
#define CONFIG_SYS_NAND_BASE                ATMEL_BASE_CS3
/* Our ALE is AD21 */
#define CONFIG_SYS_NAND_MASK_ALE            (1 << 21)
/* Our CLE is AD22 */
#define CONFIG_SYS_NAND_MASK_CLE            (1 << 22)
#define CONFIG_SYS_NAND_ONFI_DETECTION
/* PMECC & PMERRLOC */
#define CONFIG_ATMEL_NAND_HWECC
#define CONFIG_ATMEL_NAND_HW_PMECC
#define CONFIG_PMECC_CAP                    4
#define CONFIG_PMECC_SECTOR_SIZE            512
#define CONFIG_CMD_NAND_TRIMFFS
#endif
```

Secondly, check the *drivers/mtd/nand/nand_ids.c* file to see if the ID of NAND Flash on the customized board is in the support list. If it is missing, add it to the list.

For example:

```
const struct nand_flash_dev nand_flash_ids[] = {

#ifdef CONFIG_MTD_NAND_MUSEUM_IDS
        {“NAND 1MiB 5V 8-bit”, 0x6e, 256, 1, 0x1000, 0},
        {“NAND 2MiB 5V 8-bit”, 0x64, 256, 2, 0x1000, 0},
        {“NAND 4MiB 5V 8-bit”, 0x6b, 512, 4, 0x2000, 0},
        {“NAND 1MiB 3,3V 8-bit”, 0xe8, 256, 1, 0x1000, 0},
        {“NAND 1MiB 3,3V 8-bit”, 0xec, 256, 1, 0x1000, 0},
        ...
#endif
        ...
};

const struct nand_manufacturers nand_manuf_ids[] = {
        {NAND_MFR_TOSHIBA, "Toshiba"},
        {NAND_MFR_SAMSUNG, "Samsung"},
        {NAND_MFR_FUJITSU, "Fujitsu"},
        {NAND_MFR_NATIONAL, "National"},
        {NAND_MFR_RENESAS, "Renesas"},
        {NAND_MFR_STMICRO, "ST Micro"},
        {NAND_MFR_HYNIX, "Hynix"},
        {NAND_MFR_MICRON, "Micron"},
        {NAND_MFR_AMD, "AMD/Spansion"},
        {NAND_MFR_MACRONIX, "Macronix"},
        {NAND_MFR_EON, "Eon"},
        {0x0, "Unknown"}
};
```

If a serial Flash module other than the one provided in the SAMA5D3x-EK is used, check the *drivers/mtd/spi/sf_params.c* file to ensure that the corresponding serial Flash memory is listed.

The serial Flash driver must be included. Firstly, check the *include/configs/sama5d3xek_ctm.h* file to ensure that file includes the line *#define CONFIG_CMD_SF*. In addition, verify that the SPI driver is listed and that it supports the Serial Flash.

```
#ifdef CONFIG_CMD_SF
#define CONFIG_ATMEL_SPI
#define CONFIG_SPI_FLASH
#define CONFIG_SPI_FLASH_ATMEL
#define CONFIG_SF_DEFAULT_SPEED       30000000
#endif
```

If booting from an SD card, make sure that the HSMCI driver is included. Firstly, ensure the *include/configs/sama5d3xek_ctm.h* file includes the line *#define CONFIG_CMD_MMC*, and then define the MCI device which is connected on the customized board, such as:

```
#ifdef CONFIG_CMD_MMC
#define CONFIG_MMC
#define CONFIG_GENERIC_MMC
#define CONFIG_GENERIC_ATMEL_MCI
#define ATMEL_BASE_MMCI                 ATMEL_BASE_MCI0
#endif
```

Generally, there is an external SDRAM device on the board. Users can configure the SDRAM base address and size in the *include/configs/sama5d3xek_ctm.h* file.

```
#define CONFIG_NR_DRAM_BANKS          1
#define CONFIG_SYS_SDRAM_BASE         ATMEL_BASE_DDRCS
#define CONFIG_SYS_SDRAM_SIZE         0x20000000
```

2.2 Make Drivers Ready

After modifying the code for the customized board, which ensures that the system can be correctly initialized and boot up, users need to tailor the peripheral drivers according to their boards to make sure that the generated binary includes the necessary drivers and excludes the unnecessary drivers.

All the peripheral drivers are located in the *drivers* folder. For example, the EMAC and GMAC driver is in *drivers/net/macb.c*; the USB driver is in *drivers/usb/host/ehci-atmel.c*; the DBGU driver is in *drivers/serial/atmel_usart.c*.

Generally users do not need to change the drivers.

In the board configuration file *include/configs/sama5d3xek_ctm.h*, users can tailor the drivers they need by defining or undefining the corresponding peripherals.

For example:

```
/* Serial console */
#define CONFIG_ATMEL_USART
#define CONFIG_USART_BASE       ATMEL_BASE_DBGU
#define CONFIG_USART_ID         ATMEL_ID_DBGU
/* USB */
#define CONFIG_CMD_USB
```

If a peripheral is defined in the above header file, its corresponding initialization function in the *board/atmel/sama5d3xek_ctm/sama5d3xek_ctm.c* file will be executed.

```
#ifdef CONFIG_CMD_USB
static void sama5d3xek_usb_hw_init(void)
{
      at91_set_pio_output(AT91_PIO_PORTD, 25, 0);
      at91_set_pio_output(AT91_PIO_PORTD, 26, 0);
      at91_set_pio_output(AT91_PIO_PORTD, 27, 0);
}
#endif
```

The hardware level initialization function is defined in the *arch/arm/cpu/armv7/at91/sama5d3_device.c* file.

Users can change the functions according to the corresponding peripheral settings and connections on the customized boards.

2.3 How to Use U-Boot for Debugging

When U-Boot binary is generated, the user can load the binary into the corresponding memory device and begin to use U-Boot[(1)]. In this application note, the binary in NAND Flash serves as an example. The Atmel SAM-BA® programming tool can load the image into the NAND Flash.

The SAM-BA software can be downloaded from www.atmel.com.

For information on how to use SAM-BA to load the image, please refer to the section *Load U-Boot on AT91 boards* available on www.at91.com.

With U-Boot, users can perform several operations, such as checking the system information, setting environment variables, debugging the memory, transferring files, and booting the application in memory.

When U-Boot starts up, the system prints the following information if *#define CONFIG_DISPLAY_CPUINFO* is enabled in the *include/configs/sama5d3xek_ctm.h* file:

```
U-Boot 2014.07-00043-g87076e8 (Nov 19 2014 - 17:03:23)

CPU: SAMA5D34
Crystal frequency:       12 MHz
CPU clock        :      528 MHz
Master clock     :      132 MHz
DRAM:  512 MiB
NAND:  256 MiB
MMC:   mci: 0
In:    serial
Out:   serial
Err:   serial
Net:   gmac0
Warning: failed to set MAC address
, usb_ether
Hit any key to stop autoboot:  0
```

The displayed information provides the CPU part number, CPU clock setting, DDR size, NAND Flash size, MCI device number and GMAC availability.

Notes: 1. AT91Bootstrap binary needs to be downloaded to the NAND Flash on the board before the U-Boot. For AT91Bootstrap information, please refer to the guide available on www.at91.com.

After U-Boot runs, users can use U-Boot commands to perform debugging as shown in the following examples:

- Check the U-Boot version information

```
U-Boot> version

U-Boot 2014.07-00043-g87076e8 (Nov 19 2014 - 17:03:23)
arm-none-linux-gnueabi-gcc (Sourcery CodeBench Lite 2013.05-24) 4.7.3
GNU ld (Sourcery CodeBench Lite 2013.05-24) 2.23.52.20130219
```

Users can use the following commands to debug and to determine if the network is working:

- Set up TFTP (Trivial File Transfer Protocol) server IP address

```
U-Boot> setenv serverip 10.0.0.10
```

- Set up the target board's IP address

```
U-Boot> setenv ipaddr 10.0.0.20
```

- Set up the target board's ethernet address

```
U-Boot> setenv ethaddr 3a:1f:34:08:54:54
```

Note: If U-Boot prints '*Can't overwrite "ethaddr"*', it means *ethaddr* has already been set. There is no need to set it again.

- Save the environment variables

```
U-Boot> saveenv
```

- Test the network connection to server

```
U-Boot> ping $(serverip)
gmac0: PHY present at 7
gmac0: Starting autonegotiation...
gmac0: Autonegotiation complete
gmac0: link up, 100Mbps full-duplex (lpa: 0x41e1)
Using gmac0 device
host 10.0.0.10 is alive
```

- Dump, modify or erase the contents of the memory to check if the memory works correctly

```
U-boot> md [.b, .w, .l] address [number of objects] //Dump the memory contents
U-boot> mm[.b, .w, .l] address                      //Modify the memory contents
```

- Run the application under U-Boot

```
U-boot> bootm [addr [arg…]]     //Boot application image from memory
U-boot> go addr[arg]            //Start application at the specific address
```

Print "?" to list all the available U-Boot commands.

2.4 U-Boot Application Demo

One of the most useful functions of U-Boot is loading the Linux kernel with TFTP. When recompiling the Linux kernel, users can put the files on the TFTP server on the PC side and load them through the network instead of re-flashing these binaries into the Flash memory every time. This prolongs the lifetime of Flash memory and also makes the debugging highly efficient.

Here we provide a brief example on how to load the Linux kernel with TFTP protocol and how to use NFS (Network File System).

- Load Linux kernel by TFTP

  To load Linux kernel by TFTP protocol, firstly make sure that a TFTP server is installed on the PC side and then put the Kernel file under the main folder of the TFTP server. When the environment setup is done, execute the following commands:

```
U-boot> setenv serverip xxx.xxx.xxx.xxx  //Set up the TFTP server's IP address
U-boot> setenv ipaddr xxx.xxx.xxx.xxx    //Set up the target board's IP address
U-boot> setenv ethaddr xx:xx:xx:xx:xx:xx //Set up the target board's Ethernet
address
U-boot> setenv bootcmd 'tftp 0x22200000 uImage; bootm 0x22200000'
//Set "bootcmd" variable, the command that will be executed when running "boot"
command
U-boot> saveenv                          //Save all variables into flash
U-boot> boot                             //Run boot command
```

  The system will load the Linux kernel to the dedicated address and start the kernel.

- Use NFS as Root File System

  As for how to use NFS as root file system, firstly make sure that an NFS server is installed on the PC side, which contains the whole root file system. Then access an additional disk space (e.g., USB disk) through the network. This storage space, located on the development host, will be mounted on the target board by using NFS. Finally, use U-Boot to pass parameters to the kernel, so that the kernel boot will use the NFS as the root file system.

  IMPORTANT: The kernel must be compiled with the following options to support NFS as the root files system: CONFIG_IP_PNP, CONFIG_ROOT_NFS.

  On the PC side, after we make sure that the network connection and NFS server work well, we need to mount the root file system via NFS (configuration details on how to set up NFS server on PC side are available on https://wiki.debian.org).

```
sudo mkdir /opt/rootfs/buildroot
//Create a folder in the NFS export folder. This folder is used to store a root
file system.
tar xjvf /media/…/xxx.tar.bz2  -C /opt/rootfs/buildroot
//Extract the root file system "xxx.tar.bz2" from the USB disk under the folder
just created
```

  On the target board side, we should set the serverip, ipaddr and ethaddr correctly (the same way as TFTP mode), and then use the following commands. In this way, the Linux on target board can access the NFS root file system.

```
U-boot> setenv bootargs 'mem=256M console=ttyS0, 115200 root=/dev/nfs rw
nfsroot=10.0.0.10:/opt/rootfs/buildroot
ip=10.0.0.20:10.0.0.10::255.0.0.0:::::
U-boot> boot
```

  Note: 10.0.0.10 is the IP address of PC server.
  10.0.0.20 is the IP address of the SAMA5D3 based target board.

  When the kernel boots up, it will use the IP address we set to access the NFS root file system.

Revision History

Table 2-1. Revision History

Doc. Rev.	Date	Changes
A	28-Nov-2014	First release

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**