# AT91Bootstrap framework

**Reference: Lit #/File Name**

**Version: V1.0**

**Release Date: 09-Oct-2006**

# Revision History

| Revision | Date | Author | Comments |
|---|---|---|---|
| V1.0 | 09-Oct-2006 | NLe | Creation. |

| Date | 09-Oct-2006 | Name | AT91Bootstrap framework |
|---|---|---|---|
| Version | V1.0 | Reference | Lit #/File Name |

# Table of Contents

# Table of Contents

| Date | 09-Oct-2006 | Name | AT91Bootstrap framework |
|---|---|---|---|
| Version | V1.0 | Reference | Lit #/File Name |

# 1. Introduction

This document describes AT91Bootstrap application which can be considered as a first level bootloader. AT91Bootstrap is a modular application so it becomes easy to specialize the framework for a particular deployment strategy. AT91Bootstrap also provides clear examples, for a particular device, on how to perform some basic static configurations such as PMC or PIOs... AT91Bootstrap can be easily configured using a higher level protocol.

AT91Bootstrap integrates different sets of algorithms:

- Device initialization such as Clock Speed configuration, PIO settings...
- Peripheral drivers such as PIO, PMC or SDRAMC...
- Physical media algorithm such as DataFlash, NandFlash, Parallel Flash...
- File System drivers such as JFFS2 or FAT...
- Compression and Cipher algorithms
- Application Launcher for ELF, Linux...

Using this set of algorithms, it becomes easy to get a basic bootloader which, for example, is located in DataFlash and will be copied to internal SRAM by SAM-BA Boot. That bootloader could, for example, perform the processor initialization (PLLs, PIOs, SDRAMC, SPI) then could load U-Boot from DataFlash sectors to SDRAM and finally jump to it.

# 2. Peripheral drivers

Before using most of AT91SAM peripherals, AIC, PMC, and PIO must be configured to enable access to the device ressources to the peripheral (aka clock, pins, processor interrupt). All of these functions are located in the driver directory.

## 2.1 PIO

The following general purpose routines are used to interface with the PIO controller.

The basic idea for the PIO driver is to associate a unique number (pin_number) to each PIO line for all PIO controllers. Thus PIOA controller handles pin_number from 0 to 31, PIOB controller handles pin_number from 32 to 61, PIOC controller handles pin from 62 to 95 and so on.

A macro which returns the index of PIO controller for a given pin_number is straightforward:

```
/* Convert Pin Number into PIO controller index */
static inline unsigned pin_to_controller(unsigned pin)
{
      return  (pin) / 32;
}
```

Note: This implementation is close to the standard open source implementation found in Linux to set an example.

### 2.1.1 pio_desc structure

This C structure is used to specify the PIO configuration for a pin. An array of pio_desc structure is given as an argument to the pio_setup() function. For each element of the array, the pio_setup() function will configure the PIO controller according to the pin configuration. This array must be ended by a NULL element. pio_desc instances can be declared as const elements to be residant in CODE segments.

**Table 2-1.** pio_desc structure

| Field | Type | Description |
|---|---|---|
| pin_name | const char * | If NULL, mark the last element of an array |
| pin_num | unsigned int | pin_number as described in the introduction |
| dft _value | unsigned int | Default value when configured as an output 1 = HIGH; 0 = LOW |
| attribute | unsigned char | |
| type | enum | As defined in the pio_type enumeration |

**Table 2-2.** pio_type enumeration

| Name | Description |
|---|---|
| PIO_PERIPH_A | The pin corresponds to the peripheral A signal |

**Table 2-2.** pio_type enumeration

| Name | Description |
|---|---|
| PIO_PERIPH_B | The pin corresponds to the peripheral B signal |
| PIO_INPUT | The pin is in input |
| PIO_OUTPUT | The pin is in output |

**Table 2-3.** PIO attributes

| Name | Value | Description |
|---|---|---|
| PIO_DEFAULT | 0 | Default configuration |
| PIO_PULLUP | (1 << 0) | Enable the PIO internal pull-up |
| PIO_DEGLITCH | (1 << 1) | Enable the PIO glitch filter (mostly used with IRQ handling) |
| PIO_OPENDRAIN | (1 << 2) | Enable the PIO multi-driver.<br>This is only valid for output and allows the output pin to run as an open drain output. |

In the following example, pins 9 and 10 of the PIO controller A are configured to be connected with the DBGU internal signals RXD and TXD:

```
const struct pio_desc hw_pio[] = {
  {"RXD", AT91C_PIN_PA(9), 0, PIO_DEFAULT, PIO_PERIPH_A},
  {"TXD", AT91C_PIN_PA(10), 0, PIO_DEFAULT, PIO_PERIPH_A},
  {"", 0, 0, PIO_DEFAULT, PIO_PERIPH_A},
};
```

This array can be used as an argument to the pio_setup() function.

### 2.1.2 pio_setup() function

This function configure PIOs according to the pin description given as an argument.

Note: If a pin is declared as being an input, the clock of the corresponding PIO is not automatically configured and must be previously enabled in the PMC.

**Table 2-4.** pio_setup()function

| `int pio_setup (const struct pio_desc *pio_desc)` | |
|---|---|
| **Input Arguments** | |
| **Name** | **Description** |
| pio_desc | Pointer to a pio_desc array ended by a NULL element |
| **Output Result** | |
| **Type** | **Description** |
| int | Return the number of pin configured |

Example

In the following example, pins are configured to output PCK signals on the pins controlled by the pin 7 and 8 of the PIO controller A.

```
const struct pio_desc hw_pio[] = {
        {"PCK0", AT91C_PIN_PA(7), 0, PIO_DEFAULT, PIO_PERIPH_B},
        {"PCK1", AT91C_PIN_PA(8), 0, PIO_DEFAULT, PIO_PERIPH_B},
        {"", 0, 0, PIO_DEFAULT, PIO_PERIPH_A},
};
/* Configure the PIO controller to output PCK0 */
pio_setup(hw_pio);
```

### 2.1.3 pio_get_value() function

This function is used to capture the state of the corresponding pin associated with the pin_number given in argument.

The pin must have been configured in input as previously described.

Note: The clock of the corresponding PIO must be enabled in the PMC in order to have a correct value returned by this function.

**Table 2-5.** `pio_get_value()function`

| `int pio_get_value(unsigned pin)` | |
|---|---|
| **Input Arguments** | |
| **Name** | **Description** |
| pin | pin number as decribed above |
| **Output Result** | |
| **Type** | **Description** |
| int | 0: the pin is low<br>1: the pin is high<br>-1 if the pin-number does not exist |

### 2.1.4 pio_set_value() function

This function is used to force the state of the pin associated with the pin_number given in argument.

The pin must have been configured in output as previously described.

**Table 2-6.** `pio_get_gpio_value()function`

| `int pio_set_value(unsigned pin, int value)` | |
|---|---|
| **Input Arguments** | |
| **Name** | **Description** |
| pin | pin number as decribed above |
| value | 0 forces the corresponding pin level to low<br>Else forces the corresponding pin to high |
| **Output Result** | |
| **Type** | **Description** |
| int | If the pin does not exist returns -1<br>Else returns 0 |

Example

In the following example, pins corresponding to PIO_TEST_OUT and PIO_TEST_IN are supposed to be connected by an external strap. PIO_TEST_OUT is configured in output and its default state is low. PIO_TEST_IN is configured in input. It is used to check the PIO_TEST_OUT level.

```
/* Device specific... */
#define PIO_TEST_OUT  AT91C_PIN_PA(7)
#define PIO_TEST_IN  AT91C_PIN_PA(8)

/* Device non specific... */
const struct pio_desc hw_pio[] = {
      {"TEST_OUT", PIO_TEST_OUT, 0, PIO_DEFAULT, PIO_OUTPUT},
      {"TEST_IN", PIO_TEST_IN, 0, PIO_DEGLITCH, PIO_INPUT},
      {"", 0, 0, PIO_DEFAULT, PIO_PERIPH_A},
};
/* Configure the PIO controller to output PCK0 */
pio_setup(hw_pio);
/* Test the default value */
if (pio_get_value(PIO_TEST_IN) == 1)
      return 0; /* Return failed */
/* Force a high level on PIO_TEST_OUT pin */
pio_set_value(PIO_TEST_OUT, 1);
/* Test the default value */
if (pio_get_value(PIO_TEST_IN) == 0)
      return 0; /* Return failed */
return 1; /* Success */
```

## 2.2 PMC

The goal of these routines is to configure the Clock Generator (CKGR) and the Power Management Controller (PMC) in order to clock the core and the peripherals with the correct frequencies.

In order to reduce mathematical operations such as divisions, most of the configuration is done though constant macros.

Constant values can be generated with an external tool from the device specification and application requirements.

It allows a clear separation from the configuration algorithm with the configuration values:

```
/* Automatically generated using ... */
#define PMC_PLLAR 0x12345678 /* MOSC = 18.423Mhz,MUL = ,DIV = */
#define PMC_PLLBR 0x12345678 /* MOSC = 18.423Mhz, MUL =,DIV = */
#define PMC_MCKR  0x12345678 /* MCK = PLLA */
#define PLL_TIMEOUT 1000000

/* Configure PLLA */
if (!pmc_cfg_plla(PMC_PLLAR, PLL_TIMEOUT))
      goto pmc_error;

/* Switch MCK on PLLA output PCK = PLLA = 2 * MCK */
if (!pmc_cfg_mck(PMC_MCKR, PLL_TIMEOUT))
      goto pmc_error;

/* Configure PLLB */
if (!pmc_cfg_pllb(PMC_PLLBR, PLL_TIMEOUT))
     goto pmc_error;

pmc_error:
    /* Reset the device*/
```

### 2.2.1 pmc_cfg_plla() function

This function configures the PMC_PLLAR register with the value provided in argument and waits for the lock of the PLL. If the timeout is reached, -1 is returned.

**Table 2-7.** `pmc_cfg_plla()function`

| `int pmc_cfg_plla(unsigned int pmc_pllar, unsigned int timeout)` | |
|---|---|
| **Input Arguments** | |
| **Name** | **Description** |
| `pmc_pllar` | Value to indicate in the PMC_PLLAR Register |
| timeout | Initial value of a timeout counter |
| **Output Result** | |
| **Type** | **Description** |
| int | -1 Timeout error<br>0 Success |

### 2.2.2 pmc_cfg_pllb() function

This function configures the PMC_PLLBR register with the value provided in argument and waits for the lock of the PLL. If the timeout is reached, -1 is returned.

**Table 2-8.** `pmc_cfg_pllb()function`

| `int pmc_cfg_pllb(unsigned int pmc_pllbr, unsigned int timeout)` | |
|---|---|
| **Input Arguments** | |
| **Name** | **Description** |
| `pmc_pllbr` | Value to indicate in the PMC_PLLBR Register |
| timeout | Initial value of a timeout counter |
| **Output Result** | |
| **Type** | **Description** |
| int | -1 Timeout error<br>0 Success |

### 2.2.3 pmc_cfg_mck() function

This function configures the PMC_MCKR register with the value provided in argument and waits for MCK to be ready. If the timeout is reached, -1 is returned.

**Table 2-9.** `pmc_cfg_mck()function`

| `int pmc_cfg_mck(unsigned int pmc_mckr, unsigned int timeout)` | |
|---|---|
| **Input Arguments** | |
| **Name** | **Description** |
| `pmc_mckr` | Value to indicate in the PMC_MCKR Register |
| timeout | Initial value of a timeout counter |
| **Output Result** | |
| **Type** | **Description** |
| int | -1 Timeout error<br>0 Success |

### 2.2.4 pmc_cfg_pck() function

This function configures PMC_PCKRx register with the value provided in argument and waits for PCKx flag to be ready.

**Table 2-10.** `pmc_cfg_mck()function`

| `int pmc_cfg_mck(unsigned char x, unsigned int clk_sel, unsigned int prescaler)` | |
|---|---|
| **Input Arguments** | |
| **Name** | **Description** |
| x | PCKx |
| clk_sel | Clock Source Selection (CSS field of PMC_PCKx register) |
| prescaler | PCK Prescaler (PRES field of PMC_PCKx register) |
| **Output Result** | |
| **Type** | **Description** |
| int | 0 Success |

## 2.3 SDRAMC

The goal of these routines is to configure the SDRAM Controller (SDRAMC) in order to use the SDRAM at the specified frequency.

### 2.3.1 Prerequisite

- CFG_SDRAM and CFG_HW_INIT flags must be defined in your board project header file.

Example: Define these flags in board/at91sam9260ek/dataflash/at91sam9260ek.h header file.

- Create a function sdramc_hw_init() in your board source file.

Example: Define this function in board/at91sam9260ek/at91sam9260ek.c source file.

### 2.3.2 sdramc_hw_init() function

That function has to be implemented in your board source file. It simply configures the PIOs used by the SDRAMC controller.

Prototype: `void sdramc_hw_init(void)`

Note: Even if no PIO needs to be configured, the function has to be defined as empty.

Example

This is the function defined in board/at91sam9260ek/at91sam9260ek.c source file.

```
#ifdef CFG_SDRAM
void sdramc_hw_init(void)
{
	const struct pio_desc sdramc_pio[] = {
		{"D0", AT91C_PIN_PC(16), 0, PIO_DEFAULT, PIO_PERIPH_A},
		{"D1", AT91C_PIN_PC(17), 0, PIO_DEFAULT, PIO_PERIPH_A},
		{"D2", AT91C_PIN_PC(18), 0, PIO_DEFAULT, PIO_PERIPH_A},
		{"D3", AT91C_PIN_PC(19), 0, PIO_DEFAULT, PIO_PERIPH_A},
		{"D4", AT91C_PIN_PC(20), 0, PIO_DEFAULT, PIO_PERIPH_A},
		{"D5", AT91C_PIN_PC(21), 0, PIO_DEFAULT, PIO_PERIPH_A},
		{"D6", AT91C_PIN_PC(22), 0, PIO_DEFAULT, PIO_PERIPH_A},
		{"D7", AT91C_PIN_PC(23), 0, PIO_DEFAULT, PIO_PERIPH_A},
		{"D8", AT91C_PIN_PC(24), 0, PIO_DEFAULT, PIO_PERIPH_A},
		{"D9", AT91C_PIN_PC(25), 0, PIO_DEFAULT, PIO_PERIPH_A},
		{"D10", AT91C_PIN_PC(26), 0, PIO_DEFAULT, PIO_PERIPH_A},
		{"D11", AT91C_PIN_PC(27), 0, PIO_DEFAULT, PIO_PERIPH_A},
		{"D12", AT91C_PIN_PC(28), 0, PIO_DEFAULT, PIO_PERIPH_A},
		{"D13", AT91C_PIN_PC(29), 0, PIO_DEFAULT, PIO_PERIPH_A},
		{"D14", AT91C_PIN_PC(30), 0, PIO_DEFAULT, PIO_PERIPH_A},
		{"D15", AT91C_PIN_PC(31), 0, PIO_DEFAULT, PIO_PERIPH_A},
		{"", 0, 0, PIO_DEFAULT, PIO_PERIPH_A},
	};
	/* Configure the PIO controller to enable 32-bits SDRAM */
    pio_setup(sdramc_pio);

}
#endif
```

### 2.3.3 sdramc_init() function

This function configures the SDRAM Controller with the values provided in argument.

**Table 2-11.** `sdramc_init()function`

| `int sdramc_init(unsigned int sdramc_cr, unsigned int sdramc_tr)` | |
|---|---|
| **Input Arguments** | |
| **Name** | **Description** |
| sdramc_cr | Value to indicate in the SDRAMC_CR Register |
| sdramc_tr | Value to indicate in the SDRAMC_TR Register |
| **Output Result** | |
| **Type** | **Description** |
| int | 0 Success |

Note: sdramc_init() function calls sdramc_hw_init() function

## 2.4 DBGU

The goal of these routines is to configure the Debug Unit (DBGU) in order to send message on a console.

### 2.4.1 Prerequisite

- CFG_DEBUG and CFG_HW_INIT flags must be defined in your board project header file.

Example: Define these flags in board/at91sam9260ek/dataflash/at91sam9261ek.h header file.

### 2.4.2 dbg_init() function

This function configures the Debug Unit with the provided baudrate value.

**Table 2-12.** `dbg_init()function`

| `void dbg_init(unsigned int baudrate)` | |
|---|---|
| **Input Arguments** | |
| **Name** | **Description** |
| baudrate | Value to indicate in the US_BRGR Register |

### 2.4.3 dbg_print() function

This function sends a message on a console through the Debug Unit.

**Table 2-13.** `dbg_print()function`

| `void dbg_print(const char * ptr)` | |
|---|---|
| **Input Arguments** | |
| **Name** | **Description** |
| ptr | Pointer to the string to send. |

Example

```
dbg_print(”Send this message”);
```

## 2.5 DataFlash

### 2.5.1 Prerequisite

- CFG_DATAFLASH flag must be defined in your board project header file.

Example: Define this flag in board/at91sam9260ek/dataflash/at91sam9260ek.h header file.

- Create a function df_hw_init() in your board source file

Example: Define this function in board/at91sam9260ek/at91sam9260ek.c source file.

### 2.5.2 df_hw_init() function

That function has to be implemented in your board source file. It simply configures the PIOs used by the SPI DataFlash.

Prototype: `void df_hw_init(void)`

Note: Even if no PIO needs to be configured, the function has to be defined as empty.

Example

This is the function defined in board/at91sam9260ek/at91sam9260ek.c source file.

```
#ifdef CFG_DATAFLASH
void df_hw_init(void)
{
      const struct pio_desc df_pio[] = {
            {"MISO", AT91C_PIN_PA(0), 0, PIO_DEFAULT, PIO_PERIPH_A},
            {"MOSI", AT91C_PIN_PA(1), 0, PIO_DEFAULT, PIO_PERIPH_A},
            {"SPCK", AT91C_PIN_PA(2), 0, PIO_DEFAULT, PIO_PERIPH_A},
            {"NPCS0", AT91C_PIN_PA(3), 0, PIO_DEFAULT, PIO_PERIPH_A},
            {"", 0, 0, PIO_DEFAULT, PIO_PERIPH_A},
      };
      /* Configure the PIO controller to enable SPI DataFlash*/
     pio_setup(df_pio);

}
#endif
```

### 2.5.3 load_df() function

This function configures the SPI Interface and downloads an image stored in DataFlash at the defined JUMP_ADDR. If an error happens during the process, -1 is returned.

**Table 2-14.** `load_df()function`

| `int load_df(unsigned int pcs, unsigned int img_addr, unsigned int img_size)` | |
|---|---|
| **Input Arguments** | |
| **Name** | **Description** |
| `pcs` | Select corresponding SPI Chip Select |
| img_addr | Image Address in the DataFlash |
| img_size | Image Size in the DataFlash |
| **Output Result** | |
| **Type** | **Description** |
| int | -1 Error<br>0 Success |

Note: load_df() function calls df_hw_init() function

## 2.6 NandFlash

### 2.6.1 Prerequisite

- CFG_NANDFLASH flag must be defined in your board project header file.

Example: Define this flag in board/at91sam9260ek/nandflash/at91sam9260ek.h header file.

- Create nandflash_hw_init() and nandflash_cfg_16bits_dbw_init() in your board source file

Example: Define this function in board/at91sam9260ek/at91sam9260ek.c source file.

### 2.6.2 nandflash_hw_init() function

That function has to be implemented in your board source file.

It simply configures:

- PIOs used by the NandFlash
- SMC timings
- HMatrix or Chip Configuration User Interface

Prototype: `void nandflash_hw_init(void)`

Example

This is the function defined in board/at91sam9260ek/at91sam9260ek.c source file.

```
#ifdef CFG_NANDFLASH
void nandflash_hw_init(void)
{
      const struct pio_desc nand_pio[] = {
            {"RDY_BSY", AT91C_PIN_PC(13), 0, PIO_PULLUP, PIO_PERIPH_A},
            {"NANDCS", AT91C_PIN_PC(14), 0, PIO_PULLUP, PIO_PERIPH_A},
            {"", 0, 0, PIO_DEFAULT, PIO_PERIPH_A},
      };


    /* Setup Smart Media, first enable the address range of CS3 in HMATRIX
   user interface */
    writel(readl(AT91C_BASE_CCFG + CCFG_EBICSA) | AT91C_EBI_CS3A_SM,
                  AT91C_BASE_CCFG + CCFG_EBICSA);
    /* Configure SMC CS3 */
    writel((AT91C_SM_NWE_SETUP | AT91C_SM_NCS_WR_SETUP | AT91C_SM_NRD_SETUP|
           AT91C_SM_NCS_RD_SETUP), AT91C_BASE_SMC + SMC_SETUP3);
    writel((AT91C_SM_NWE_PULSE | AT91C_SM_NCS_WR_PULSE | AT91C_SM_NRD_PULSE|
           AT91C_SM_NCS_RD_PULSE), AT91C_BASE_SMC + SMC_PULSE3);
    writel((AT91C_SM_NWE_CYCLE | AT91C_SM_NRD_CYCLE),
           AT91C_BASE_SMC + SMC_CYCLE3);
    writel((AT91C_SMC_READMODE | AT91C_SMC_WRITEMODE |
           AT91C_SMC_NWAITM_NWAIT_DISABLE | AT91C_SMC_DBW_WIDTH_EIGTH_BITS|
           AT91C_SM_TDF), AT91C_BASE_SMC + SMC_CTRL3);
```

```
    /* Configure the PIO controller */
    writel((1 << AT91C_ID_PIOC), PMC_PCER + AT91C_BASE_PMC);
    pio_setup(nand_pio);
}
#endif
```

### 2.6.3 nandflash_cfg_16bits_dbw_init() function

That function has to be implemented in your board source file. It is only used when a 16-bits NandFlash has been detected by the bootloader.

It configures the SMC in 16-bits Data Bus Width mode.

Example

This is the function defined in board/at91sam9260ek/at91sam9260ek.c source file.

```
#ifdef CFG_NANDFLASH
void nandflash_cfg_16bits_dbw_init(void)
{
   writel(readl(AT91C_BASE_SMC + SMC_CTRL3) |
   AT91C_SMC_DBW_WIDTH_SIXTEEN_BITS, AT91C_BASE_SMC + SMC_CTRL3);
}
#endif
```

### 2.6.4 load_nandflash() function

This function configures the NandFlash Interface and downloads an image stored in Nand-Flash at the defined JUMP_ADDR. If an error happens during the process, -1 is returned.

**Table 2-15.** `load_nandflash()function`

| `int load_nandflash(unsigned int img_addr, unsigned int img_size)` | |
|---|---|
| **Input Arguments** | |
| **Name** | **Description** |
| img_addr | Image Address in the NandFlash |
| img_size | Image Size in the NandFlash |
| **Output Result** | |
| **Type** | **Description** |
| int | -1 Error<br>0 Success |

Note: load_nandflash() function calls nandflash_hw_init() function

### 2.6.5 NandFlash features

#### *2.6.5.1 Large Blocks vs Small Blocks*

To summarize, NandFlash are divided into two categories:

- Large Blocks NandFlash: parts having a size higher than 1Gbit
- Small Blocks NandFlash: parts having a size smaller than 1Gbit

Note: For 1Gbit NandFlash parts, it is recommended to verify in the NandFlash datasheet if the part is considered as a Small or Large Blocks part.

To use NandFlash Small Blocks algorithms, NANDFLASH_SMALL_BLOCKS flag must be defined in your board project header file.

Example: Define this flag in board/at91sam9260ek/nandflash/at91sam9260ek.h header file.

```
#define CFG_NANDFLASH /* Enable DataFlash Download */
#define NANDFLASH_SMALL_BLOCKS /*Large Block NandFlash used instead*/
```

### 2.6.5.2 *Adding a new NandFlash support in AT91Bootstrap*

NandFlash IDs and informations are located in the *include/nand_ids.h* header file according to the following format:

```
typedef struct SNandInitInfo
{
  unsigned int uNandID; /* Nand Chip ID */
  unsigned int uNandNbBlocks;  /* Total Number of Blocks */
  unsigned int uNandBlockSize; /* Block Size*/
  unsigned int uNandSectorSize;/* Page Size */
  unsigned int uNandSpareSize; /* Number of Spare bytes for a Page */
  unsigned int uNandBusWidth; /* 0=8 bits /// 1=16 bits */
  char      name[40]; /* Nand Name */
} SNandInitInfo, *PSNandInitInfo;
```

Simply add new NandFlash informations in the NandFlash_InitInfo[] table to add support for your new NandFlash device.

# 3. Device Hardware Initialization

The C startup routine is the same for all the AT91SAM devices. This routine performs the following actions before it jumps to the main function:

- Initialize the Main Oscillator (if not already done)
- Switch MCK to the main oscillator (only if MCK is at Slow Clock)
- Initialize the .BSS segment
- Branch to the C code

This routine is written in crt0_gnu.s assembly file.

Note: To reduce the boot time, the main oscillator is enabled as soon as possible.

## 3.1 AT91SAM9261 Initialization

The following operations are performed during the AT91SAM9261 hardware initialization:

- Disable Watchdog
- Configure PLLA
- Switch MCK to PLLA/2
- Configure PLLB
- Enable I-Cache
- Configure PIOs
- Configure Matrix (only when using SDRAMC)

### 3.1.1 Generating DataFlashBoot

To generate a bootstrap in DataFlash allowing to launch U-Boot for example, please proceed that way.

Every necessary flag is defined in the board/at91sam9261ek/dataflash/at91sam9261ek.h header file.

```
/* Download Settings */

#define IMG_ADDR 0x8000 /* U-Boot Address in DataFlash */
#define IMG_SIZE 0x30000 /* U-Boot size in DataFlash */
#define JUMP_ADDR 0x23F00000 /* U-Boot Link Address in SDRAM */

/* BootStrap Settings */
#undef CFG_DEBUG /* Disable DBGU messages (optional) */
#define CFG_DATAFLASH /* Enable DataFlash Download */
#define CFG_HW_INIT /* Enable Low Level Configuration (PIOs,PMC...)*/
#define CFG_SDRAM /* Enable SDRAM Controller Configuration */
```

Some project parameters needs to be defined in the Makefile. Makefile is located in the board/at91sam9261ek/dataflash/ project directory.

```
# DataFlashBoot Configuration for AT91SAM9261EK

# Target name (case sensitive!!!)
TARGET=AT91SAM9261
# Board name (case sensitive!!!)
BOARD=at91sam9261ek
# Link Address and Top_of_Memory
LINK_ADDR=0x300000
TOP_OF_MEMORY=0x328000
# Name of current directory (case sensitive!!!)
PROJECT=dataflash
```

Now project parameters are set correctly, project can be compiled and linked.

```
> cd board/at91sam9261ek/dataflash
> make
```

Your image, named dataflash_at91sam9261ek.bin, should be ready now. Simply download the binary file in DataFlash with SAM-BA Application for example.

### 3.1.2 Generating NandFlashBoot

To generate a bootstrap in NandFlash allowing to launch U-Boot for example, please proceed that way.

Every necessary flag is defined in the board/at91sam9261ek/nandflash/at91sam9261ek.h header file.

```
/* Download Settings */

#define IMG_ADDR 0x20000 /* U-Boot Address in NandFlash */
#define IMG_SIZE 0x30000 /* U-Boot size in NandFlash */
#define JUMP_ADDR 0x23F00000 /* U-Boot Link Address in SDRAM */

/* BootStrap Settings */
#undef CFG_DEBUG /* Disable DBGU messages (optional) */

#define CFG_NANDFLASH /* Enable DataFlash Download */
#undef NANDFLASH_SMALL_BLOCKS /*Large Block NandFlash used instead*/

#define CFG_HW_INIT /* Enable Low Level Configuration (PIOs,PMC...)*/
#define CFG_SDRAM /* Enable SDRAM Controller Configuration */
```

Some project parameters needs to be defined in the Makefile. Makefile is located in the board/at91sam9261ek/nandflash/ project directory.

```
# NandFlashBoot Configuration for AT91SAM9261EK

# Target name (case sensitive!!!)
TARGET=AT91SAM9261
# Board name (case sensitive!!!)
BOARD=at91sam9261ek
# Link Address and Top_of_Memory
LINK_ADDR=0x300000
TOP_OF_MEMORY=0x328000
# Name of current directory (case sensitive!!!)
PROJECT=nandflash
```

Now project parameters are set correctly, project can be compiled and linked.

```
> cd board/at91sam9261ek/nandflash
> make
```

Your image, named nandflash_at91sam9261ek.bin, should be ready now. Simply download the binary file in NandFlash with SAM-BA Application for example.

## 3.2 AT91SAM9260 Initialization

The following operations are performed during the AT91SAM9260 hardware initialization:

- Disable Watchdog
- Configure PLLA
- Switch MCK to PLLA/2
- Configure PLLB
- Enable I-Cache
- Configure PIOs
- Configure Matrix (only when using SDRAMC)

### 3.2.1 Generating DataFlashBoot

To generate a bootstrap in DataFlash allowing to launch U-Boot for example, please proceed that way.

Every necessary flag is defined in the board/at91sam9260ek/dataflash/at91sam9260ek.h header file.

```
/* Download Settings */

#define IMG_ADDR 0x8000 /* U-Boot Address in DataFlash */
#define IMG_SIZE 0x30000 /* U-Boot size in DataFlash */
#define JUMP_ADDR 0x23F00000 /* U-Boot Link Address in SDRAM */
```

```
/* BootStrap Settings */
#undef CFG_DEBUG /* Disable DBGU messages (Recommended - see code size
part below) */
#define CFG_DATAFLASH /* Enable DataFlash Download */
#define CFG_HW_INIT /* Enable Low Level Configuration (PIOs,PMC...)*/
#define CFG_SDRAM /* Enable SDRAM Controller Configuration */
```

Some project parameters needs to be defined in the Makefile. Makefile is located in the board/at91sam9260ek/dataflash/ project directory.

```
# DataFlashBoot Configuration for AT91SAM9260EK

# Target name (case sensitive!!!)
TARGET=AT91SAM9260
# Board name (case sensitive!!!)
BOARD=at91sam9260ek
# Link Address and Top_of_Memory
LINK_ADDR=0x200000
TOP_OF_MEMORY=0x301000
# Name of current directory (case sensitive!!!)
PROJECT=dataflash
```

Now project parameters are set correctly, project can be compiled and linked.

```
> cd board/at91sam9260ek/dataflash
> make
```

Your image, named dataflash_at91sam9260ek.bin, should be ready now. Simply download the binary file in DataFlash with SAM-BA Application for example.

### 3.2.2 Generating NandFlashBoot

To generate a bootstrap in NandFlash allowing to launch U-Boot for example, please proceed that way.

Every necessary flag is defined in the board/at91sam9260ek/nandflash/at91sam9260ek.h header file.

```
/* Download Settings */

#define IMG_ADDR 0x20000 /* U-Boot Address in NandFlash */
#define IMG_SIZE 0x30000 /* U-Boot size in NandFlash */
#define JUMP_ADDR 0x23F00000 /* U-Boot Link Address in SDRAM */

/* BootStrap Settings */
#undef CFG_DEBUG /* Disable DBGU messages (Recommended - see code size
part below) */
```

```
#define CFG_NANDFLASH /* Enable DataFlash Download */
#undef NANDFLASH_SMALL_BLOCKS /*Large Block NandFlash used instead*/

#define CFG_HW_INIT /* Enable Low Level Configuration (PIOs,PMC...)*/
#define CFG_SDRAM /* Enable SDRAM Controller Configuration */
```

Some project parameters needs to be defined in the Makefile. Makefile is located in the board/at91sam9260ek/nandflash/ project directory.

```
# NandFlashBoot Configuration for AT91SAM9260EK

# Target name (case sensitive!!!)
TARGET=AT91SAM9260
# Board name (case sensitive!!!)
BOARD=at91sam9260ek
# Link Address and Top_of_Memory
LINK_ADDR=0x200000
TOP_OF_MEMORY=0x301000
# Name of current directory (case sensitive!!!)
PROJECT=nandflash
```

Now project parameters are set correctly, project can be compiled and linked.

```
> cd board/at91sam9260ek/nandflash
> make
```

Your image, named nandflash_at91sam9260ek.bin, should be ready now. Simply download the binary file in NandFlash with SAM-BA Application for example.

### 3.2.3 AT91SAM9260 BootStrap Code size constraint

Bootstrap binary image size must be less than 4kBytes as it is the AT91SAM9260 internal available SRAM size.

According the GCC toolchain which is used (GCC-3.4 Toolchain or less), resulting code size may be higher than the allowed 4kBytes. In such a case, either update your GCC toolchain to a more recent one (GCC-4.0 Toolchain or higher) or do not use the provided gpio driver to configure SDRAM PIOs for example. Indeed, replace sdramc_hw_init() function in board/at91sam9260/at91sam9260ek.c source file by:

```
#ifdef CFG_SDRAM
void sdramc_hw_init(void)
{
	/* Configure the PIO controller to enable 32-bits SDRAM */
	writel(0xFFFF0000, AT91C_BASE_PIOC + PIO_ASR(0));
	writel(0xFFFF0000, AT91C_BASE_PIOC + PIO_PDR(0));
}
```

```
#endif
```

Note: Code is less readable but it should be sufficient enough to have less than 4kBytes code size without having to re-compile a complete GCC toolchain.

### 3.2.4 AT91SAM9260-EK Bootstrap Recovery Procedure

AT91SAM9260 ROM code is able to boot on:

- DataFlash card on NCS0
- DataFlash device on NCS1 (recovery procedure available)
- NandFlash (recovery procedure available)

There is no jumper neither on DataFlash NCS1 nor on NandFlash on the AT91SAM9260-EK board. So it won't be possible to program your target anymore unless the first page content of the device is erased...

That is why a simple software recovery procedure has been implemented in at91bootstrap for the at91sam9260ek board.

If BP4 is pressed during the boot sequence, it is automatically detected and dataflash first page (or nandflash first block) is erased in order to boot correctly next time. Recovery procedure is performed the following way:

1. Press BP4 button (BP4 must be keeped pressed during steps 2 and 3)
2. Press Reset button
3. Let the application boot
4. Release BP4
5. Reset the board

AT91SAM9260 ROM Code should now start correctly...

# 4. AT91Bootstrap GNU project

## 4.1 Compiling an AT91Bootstrap project

### 4.1.1 CodeSourcery GNU ARM Toolchain

Sourcery G++ is a complete software development environment based on the GNU Toolchain. It is available for both Windows and Linux environments and can be downloaded at the following URL:

http://www.codesourcery.com/gnu_toolchains/arm/download.html

Note: AT91Bootstrap and Sourcery G++ for Windows are installed with the AT91-ISP package.

Once your toolchain is installed, install AT91Bootstrap in a directory and cd into it.

### 4.1.2 Make Command

To compile an AT91Bootstrap project:

- Go into the board directory
- Select your board by going into corresponding board directory
- Select your project by going into corresponding project directory
- Configure your project (Makefile and header file)
- Type make

Example:

To compile a NandFlashboot project for AT91SAM9260-EK board, just type the following commands:

```
> cd board/at91sam9260ek/nandflash
> make
```

## 4.2 Adding a new board to AT91Bootstrap project

Adding a new board to the AT91Bootstrap project is very easy:

- Copy the board directory which is the closest to your board

Example:

If your board is based on a AT91SAM9261 part,

```
cd board
cp -r at91sam9261ek/ my_board/
```

- Rename board specific files

```
cd my_board
mv at91sam9261ek.c my_board.c
mv “project“/at91sam9261ek.h “project“/my_board.h
```

- Edit corresponding “project“ Makefile and modify BOARD variable accordingly

```
BOARD=my_board
PROJECT=“project“
```

- Make your modifications and compile the project

## 4.3 Adding a new project to a board

Adding a new project to the AT91Bootstrap project is very easy:

- Copy the board directory which is the closest to your board

```
cd board/my_board
mkdir new_project
```

- Copy a Makefile and my_board.h header file

```
cp dataflash/* new_project/.
```

- Edit corresponding “new_project“ Makefile and modify PROJECT variable accordingly

```
PROJECT=new_project
```

## 4.4 Adding a new driver to AT91Bootstrap project

Adding a new driver to the AT91Bootstrap project is very easy:

- Add new driver in the driver directory. Move every pieces of code which are product or board dependent in the *board/your_board/* directory.
- If necessary, add an include file having exactly same driver name in the *include* directory.
- Use pre-processor instructions to insert your new feature into the AT91Bootstrap project.
- Add new driver file into project Makefile.