1. Introduction
The AT91Bootstrap application is a first level bootloader for Atmel® AT91SAM9 ARM® Thumb®-based microcontrollers. It is a modular application and thus can be used to customize the framework for a particular deployment strategy. AT91Bootstrap also provides clear examples, for a particular device, on how to perform basic static configurations, such as PMC and PIOs. AT91Bootstrap can be easily configured using a higher level protocol.

AT91Bootstrap integrates several sets of algorithms:
- Device initialization such as clock speed configuration, PIO settings, etc.
- Peripheral drivers such as PIO, PMC, SDRAMC, etc.
- Physical media algorithms such as DataFlash®, NANDFlash, Parallel Flash, etc.
- File System drivers such as JFFS2, FAT, etc.
- Compression and Cipher algorithms
- Application Launcher for ELF, Linux®, etc.

For example, using this set of algorithms, it is possible to obtain a basic bootloader that is located in DataFlash and is copied to internal SRAM by SAM-BA™ Boot. The bootloader performs the processor initialization (PLLs, PIOs, SDRAMC, SPI), loads U-Boot from DataFlash sectors to SDRAM and then jumps to it.
2. Peripheral Drivers

Before using AT91SAM peripherals, the AIC, PMC and PIO must be configured to enable access to the device resources of the peripheral (e.g., 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 PIO driver associates 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.

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

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

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

2.1.1 pio_desc Structure

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

Table 2-1. pio_desc Structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pin_name</td>
<td>const char *</td>
<td>If NULL, mark the last element of an array</td>
</tr>
<tr>
<td>pin_num</td>
<td>unsigned int</td>
<td>pin_number as described in the introduction</td>
</tr>
<tr>
<td>dft_value</td>
<td>unsigned int</td>
<td>Default value when configured as an output 1 = HIGH; 0 = LOW</td>
</tr>
<tr>
<td>attribute</td>
<td>unsigned char</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>enum</td>
<td>As defined in the pio_type enumeration</td>
</tr>
</tbody>
</table>

Table 2-2. pio_type Enumeration

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIO_PERIPH_A</td>
<td>The pin corresponds to the peripheral A signal</td>
</tr>
<tr>
<td>PIO_PERIPH_B</td>
<td>The pin corresponds to the peripheral B signal</td>
</tr>
<tr>
<td>PIO_INPUT</td>
<td>The pin is in input</td>
</tr>
<tr>
<td>PIO_OUTPUT</td>
<td>The pin is in output</td>
</tr>
</tbody>
</table>
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:

```c
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 configures PIOs depending on 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>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIO_DEFAULT</td>
<td>0</td>
<td>Default configuration</td>
</tr>
<tr>
<td>PIO_PULLUP</td>
<td>(1 &lt;&lt; 0)</td>
<td>Enable the PIO internal pull-up</td>
</tr>
<tr>
<td>PIO_DEGLITCH</td>
<td>(1 &lt;&lt; 1)</td>
<td>Enable the PIO glitch filter (mostly used with IRQ handling)</td>
</tr>
<tr>
<td>PIO_OPENDRAIN</td>
<td>(1 &lt;&lt; 2)</td>
<td>Enable the PIO multi-driver. This is only valid for output and allows the output pin to run as an open drain output.</td>
</tr>
</tbody>
</table>

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:

```c
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 configures PIOs depending on 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>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIO_DEFAULT</td>
<td>0</td>
<td>Default configuration</td>
</tr>
<tr>
<td>PIO_PULLUP</td>
<td>(1 &lt;&lt; 0)</td>
<td>Enable the PIO internal pull-up</td>
</tr>
<tr>
<td>PIO_DEGLITCH</td>
<td>(1 &lt;&lt; 1)</td>
<td>Enable the PIO glitch filter (mostly used with IRQ handling)</td>
</tr>
<tr>
<td>PIO_OPENDRAIN</td>
<td>(1 &lt;&lt; 2)</td>
<td>Enable the PIO multi-driver. This is only valid for output and allows the output pin to run as an open drain output.</td>
</tr>
</tbody>
</table>

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.

```c
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 the 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>
<thead>
<tr>
<th>Table 2-5. <strong>pio_get_value() Function</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>int pio_get_value(unsigned pin)</td>
</tr>
</tbody>
</table>

### Input Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pin</td>
<td>pin number as described above</td>
</tr>
</tbody>
</table>

### Output Result

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| int  | 0: the pin is low  
1: the pin is high  
-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 the argument.

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

<table>
<thead>
<tr>
<th>Table 2-6. <strong>pio_set_gpio_value() Function</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>int pio_set_value(unsigned pin, int value)</td>
</tr>
</tbody>
</table>

### Input Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pin</td>
<td>Pin number as described above</td>
</tr>
</tbody>
</table>
| value| 0 forces the corresponding pin level to low  
Else forces the corresponding pin to high |

### Output Result

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| int  | If the pin does not exist, returns **-1**  
Else returns 0 |

2.1.4.1 **Example**

In the following example, pins corresponding to PIO_TEST_OUT and PIO_TEST_IN are 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.

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

```c
/* 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_pll_ar(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_pll() 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_pll() function

<table>
<thead>
<tr>
<th>Input Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>pmc_pll</td>
<td>Value to indicate in the PMC_PLLAR Register</td>
</tr>
<tr>
<td>timeout</td>
<td>Initial value of a timeout counter</td>
</tr>
</tbody>
</table>

Output Result

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>-1 Timeout error</td>
</tr>
<tr>
<td></td>
<td>0 Success</td>
</tr>
</tbody>
</table>

2.2.2 pmc_cfg_pll() 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_pll() function

<table>
<thead>
<tr>
<th>Input Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>pmc_pllbr</td>
<td>Value to indicate in the PMC_PLLBR Register</td>
</tr>
<tr>
<td>timeout</td>
<td>Initial value of a timeout counter</td>
</tr>
</tbody>
</table>

Output Result

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>-1 Timeout error</td>
</tr>
<tr>
<td></td>
<td>0 Success</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Input Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pmc_mckr</td>
<td>Value to indicate in the PMC_MCKR Register</td>
</tr>
<tr>
<td>timeout</td>
<td>Initial value of a timeout counter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>-1 Timeout error</td>
</tr>
<tr>
<td></td>
<td>0 Success</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Input Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>PCKx</td>
</tr>
<tr>
<td>clk_sel</td>
<td>Clock Source Selection (CSS field of PMC_PCKx register)</td>
</tr>
<tr>
<td>prescaler</td>
<td>PCK Prescaler (PRES field of PMC_PCKx register)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>0 Success</td>
</tr>
</tbody>
</table>

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 sdrmc_hw_init() in your board source file. Example: Define this function in board/at91sam9260ek/at91sam9260ek.c source file.

2.3.2 **sdrmc_hw_init() Function**

This function must be implemented in your board source file. It 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.

### 2.3.2.1 Example

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

```c
#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 the argument.

**Table 2-11. sdramc_init() Function**

<table>
<thead>
<tr>
<th>Input Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sdramc_cr</td>
<td>Value to indicate in the SDRAMC_CR Register</td>
</tr>
<tr>
<td>sdramc_tr</td>
<td>Value to indicate in the SDRAMC_TR Register</td>
</tr>
</tbody>
</table>

**Output Result**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>0 Success</td>
</tr>
</tbody>
</table>

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 a 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>baudrate</td>
<td>Value to indicate in the US_BRGR Register</td>
</tr>
</tbody>
</table>

2.4.3 dbg_print() Function

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

Table 2-13. dbg_print() Function

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ptr</td>
<td>Pointer to the string to send.</td>
</tr>
</tbody>
</table>

2.4.3.1 Example

```c
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

This function must be implemented in your board source file. It 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.
2.5.2.1 Example

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

```c
#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

<table>
<thead>
<tr>
<th>Input Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pcs</td>
<td>Select corresponding SPI Chip Select</td>
</tr>
<tr>
<td>img_addr</td>
<td>Image Address in the DataFlash</td>
</tr>
<tr>
<td>img_size</td>
<td>Image Size in the DataFlash</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>int</td>
<td>-1 Error</td>
</tr>
<tr>
<td></td>
<td>0 Success</td>
</tr>
</tbody>
</table>

Note: load_df() function calls df_hw_init() function

2.6 NAND Flash

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

It configures:

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

Prototype: `void nandflash_hw_init(void)`

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

```c
#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
This function must be implemented in your board source file. It is only used when a 16-bit NAND Flash has been detected by the bootloader.

It configures the SMC in 16-bit Data Bus Width mode.
2.6.3.1 Example

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

```c
#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

<table>
<thead>
<tr>
<th>Input Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>img_addr</code></td>
<td>Image Address in the NANDFlash</td>
</tr>
<tr>
<td><code>img_size</code></td>
<td>Image Size in the NANDFlash</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int</code></td>
<td>-1 Error</td>
</tr>
<tr>
<td></td>
<td>0 Success</td>
</tr>
</tbody>
</table>

Note: load_nandflash() function calls nandflash_hw_init() function

2.6.5 NAND Flash Features

2.6.5.1 Large Blocks vs. Small Blocks

To summarize, NAND Flash blocks are divided into two categories:

- Large Block NAND Flash: parts bigger than 1 Gbit
- Small Block NAND Flash: parts smaller than 1 Gbit

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

NAND Flash Large Block algorithms are used by default.

To use NAND Flash Small Block 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.

```c
#define CFG_NANDFLASH /* Enable DataFlash Download */
#define NANDFLASH_SMALL_BLOCKS /*Small Block NandFlash used */
```
2.6.5.2 Adding a New NAND Flash Support in AT91Bootstrap

NAND Flash IDs and information are located in the include/nand_ids.h header file with the following format:

```c
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;
```

Add new NAND Flash information in the NandFlash_InitInfo[] table to add support for your new NAND Flash 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:

1. Initialize the Main Oscillator (if not already done).
2. Switch MCK to the main oscillator (only if MCK is at Slow Clock).
3. C VARIABLES INIT.
4. Branch to the C code.

This routine is given in the crt0_gnu.s assembly file.

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

3.1 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)

3.1.1 Generating DataFlashBoot

The following example shows how to generate a bootstrap in DataFlash to launch U-Boot.

The required flags are defined in the board/at91sam9260ek/dataflash/at91sam9260ek.h header file.

```c
/* Download Settings */
```


```c
#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 */
#ifndef 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.

```c
    # DataFlashBoot Configuration for AT91SAM9260EK

    # Target name (case sensitive!!!)
    TARGET=AT91SAM9260
    # Board name (case sensitive!!!)
    BOARD=at91sam9260ek
    # Link Address and Top_of_Memory
    LINK_ADDR=0x20000
    # Stack pointer
    TOP_OF_MEMORY=0x301000
    # Name of current directory (case sensitive!!!)
    PROJECT=dataflash
```

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

```bash
    > 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.

### 3.1.2 Generating NANDFlashBoot

The following example shows how to to generate a bootstrap in NANDFlash to launch U-Boot. The required flags are defined in the board/at91sam9260ek/nandflash/at91sam9260ek.h header file.

```c
/* 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
# Stack pointer
TOP_OF_MEMORY=0x301000
# Name of current directory (case sensitive!!!)
PROJECT=nandflash

Now the project parameters are set correctly and the 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.

3.1.3 AT91SAM9260 BootStrap Code Size Constraint

Bootstrap binary image size must be less than 4K bytes which corresponds to the internal available
SRAM size for the AT91SAM9260.

Depending on the toolchain which is used, resulting code size may be higher than the allowed
4K bytes. In such a case, either update your toolchain to a more recent one or do not use the
provided gpio driver to configure SDRAM PIOs for example. Replace sdrmc_hw_init() function
in board/at91sam9260/at91sam9260ek.c source file by:

#ifdef CFG_SDRAM

void sdrmc_hw_init(void)
{
    /* Configure the PIO controller to enable 32-bits SDRAM */

#endif
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 4K bytes code size without having to re-compile a complete GCC toolchain.

3.1.4 AT91SAM9260-EK Bootstrap Recovery Procedure

AT91SAM9260 ROM code boots on:

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

There is no jumper on DataFlash NCS1 or on NANDFlash on the AT91SAM9260-EK board and so it is not possible to program the target unless the content of the first page of the device is erased. A simple software recovery procedure has been implemented in AT91Bootstrap for the AT91SAM9260-EK board.

If BP4 is pressed during the boot sequence, it is automatically detected and DataFlash first page (or NAND Flash first block) is erased in order to boot correctly next time. The recovery procedure is as follows:

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

AT91SAM9260 ROM code should now start correctly.

3.2 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)

3.2.1 Generating DataFlashBoot

The following example shows how to generate a bootstrap in DataFlash to launch U-Boot.

The required flags are defined in the board/at91sam9261ek/dataflash/at91sam9261ek.h header file.

/* Download Settings */

#define IMG_ADDR 0x8000 /* U-Boot Address in DataFlash */
The following example shows how to generate a bootstrap in NAND Flash to launch U-Boot. The required flags are 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_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
# Stack pointer
TOP_OF_MEMORY=0x328000
# Name of current directory (case sensitive!!!)
PROJECT=dataflash

Now project parameters are set correctly and the 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.

### 3.2.2 Generating NAND Flash Boot

The following example shows how to generate a bootstrap in NAND Flash to launch U-Boot.
```c
#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 need 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
# Stack pointer
TOP_OF_MEMORY=0x328000
# Name of current directory (case sensitive!!!)
PROJECT=nandflash
```

Now the project parameters are set correctly and the 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.

### 3.3 AT91SAM9263 Initialization

The following operations are performed during AT91SAM9263 hardware initialization:

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

### 3.3.1 Generating DataFlashBoot

The following example shows how to generate a bootstrap in DataFlash to launch U-Boot.

The required flags are defined in the board/at91sam9263ek/dataflash/at91sam9263ek.h header file.

```c
/* Download Settings */
```
```c
#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 need to be defined in the Makefile. Makefile is located in the board/at91sam9263ek/dataflash/ project directory.

```bash
# DataFlashBoot Configuration for AT91SAM9263EK

# Target name (case sensitive!!!)
TARGET=AT91SAM9263
# Board name (case sensitive!!!)
BOARD=at91sam9263ek
# Link Address and Top_of_Memory
LINK_ADDR=0x300000
# Stack pointer
TOP_OF_MEMORY=0x314000
# Name of current directory (case sensitive!!!)
PROJECT=dataflash
```

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

```bash
> cd board/at91sam9263ek/dataflash
> make
```

Your image, named dataflash_at91sam9263ek.bin, should be ready now. Simply download the binary file in DataFlash with SAM-BA Application for example.

## 4. AT91Bootstrap GNU Project

### 4.1 Compiling an AT91Bootstrap Project

#### 4.1.1 GNU ARM Toolchain

AT91Bootstrap is compiled with a GNU ARM Toolchain (Code Sourcery, GNU ARM, etc.)

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

#### 4.1.2 Make Command

To compile an AT91Bootstrap project:

1. Go into the board directory.
2. Select your board by going into the corresponding board directory.
3. Select your project by going into the corresponding project directory.
4. Configure your project (Makefile and header file).
5. Type make.

Example:

To compile a NANDFlash boot project for AT91SAM9260-EK board, type the following commands:

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

### 4.2 Adding a New Board to an AT91Bootstrap Project

To add a new board to the AT91Bootstrap project:

- Copy the board directory which is the most similar to your board, e.g. if your board is based on a AT91SAM9261 part:
  ```bash
cd board
cp -r at91sam9261ek/ my_board/
```
- Rename board-specific files.
  ```bash
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.
  ```bash
BOARD=my_board
PROJECT="project"
```
- Make your modifications and compile the project.

### 4.3 Adding a New Project to a Board

To add a new project to the AT91Bootstrap:

- Copy the board directory which is the most similar to your board.
  ```bash
cd board/my_board
mkdir new_project
```
- Copy a Makefile and my_board.h header file.
  ```bash
cp dataflash/* new_project/.
```
- Edit corresponding "new_project" Makefile and modify PROJECT variable accordingly.
  ```bash
PROJECT=new_project
```

### 4.4 Adding a New Driver to an AT91Bootstrap Project

To add a new driver to the AT91Bootstrap project:

- Add the new driver in the driver directory. Move the pieces of code which are product or board dependent into the board/your_board/ directory.
- If necessary, add an include file with the 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.
5. Revision History

Table 5-1.

<table>
<thead>
<tr>
<th>Document Ref.</th>
<th>Comments</th>
<th>Change Request Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>6277A</td>
<td>First issue.</td>
<td></td>
</tr>
<tr>
<td>6277B</td>
<td>Added Section 3.3 &quot;AT91SAM9263 Initialization&quot; on page 18.</td>
<td></td>
</tr>
</tbody>
</table>