AN2896

SAMA5D2 Low-Power Modes Implementation

Scope

The SAMA5D2 series is a high-performance, ultra-low power Arm® Cortex®-A5 processor-based MPU.

This application note describes how to enter and exit SAMA5D2 low-power modes by providing Linux and bare metal software examples, as well as hardware application schematics.. The purpose of this document is to help users understand the low-power performance of SAMA5D2, and design power-efficient applications.

This application note is a supplement to the SAMA5D2 Series data sheet. It should be used in conjunction with the following reference documents.

Reference Documents

Document Type	Title	Lit. No.	Available
Data Sheet	SAMA5D2 Series	DS60001476	http://www.microchip.com
User's Guide	SAMA5D2 Xplained Ultra Evaluation Kit	DS50002691	http://www.microchip.com
Application Note	SAMA5D2 Discrete Power Supply Solution	AN44059	http://www.microchip.com
Data Sheet	ACT8945A	–	http://www.active-semi.com

Table of Contents

1. SAMA5D2 Low Power Modes Overview

1.1 SAMA5D2 Low-Power Consumption Modes

The SAMA5D2 devices feature five low-power modes: Backup, Backup Self-refresh (BSR), Ultra Low-power 0 (ULP0), Ultra Low-power 1 (ULP1) and Idle.

These modes provide a wide range of power consumption performances (from a few microamps to a few milliamps) and wake-up times (from a few microseconds to a few hundred milliseconds) to accommodate very different application needs. The following sections give a detailed description of the device operation in each low-power mode.

1.1.1 Backup Mode

In an application, Backup mode corresponds to an extended power-down period of the processor. In this mode, the processor, its peripherals and its memories are unpowered. Only the backup area of the device remains powered and operating, thus maintaining the Real Time Clock (RTC), the backup registers, the backup SRAM and the security module running. The security module protects the application against tampering through tamper pins PIOBU0-7, and in the SAMA5D23 and SAMA5D28, against out-of-range operation in terms of frequency (f), temperature (T) and voltage (V) through dedicated monitors. All tamper detections are time-stamped via the RTC.

At Backup mode entry or exit, it is good practice to use the Shutdown Controller (SHDWC) of the processor to help manage the power supply unit of the application.

This peripheral controls the SHDN pin that is further used on the board to enable or disable power supply channels. Typically, the software asserts SHDN low when entering Backup mode, and the SHDWC automatically toggles SHDN back to high upon a wake-up event. Backup mode exit is possible through RTC events, WKUP0, WKUP2 to WKUP9 pin events, Low-power Asynchronous Receiver (RXLP) events or Analog Comparator Controller (ACC) events.

When in Backup mode, the SAMA5D2 current consumption is reduced to a few micro-amps in VDDBU (backup area power) and it is therefore possible to supply VDDBU from a supercapacitor or from a LiMn coin cell battery. To further reduce current consumption in the storage element, the SAMA5D2 features a power switch that selects the power source of the backup area either from VDDBU or from VDDANA (analog rail power). When VDDANA is present, the backup area can be powered from VDDANA by setting the SCTRL and SSWCTRL bits in SFRBU_PSWBUCTRL. The following table gives an example of battery lifetime estimation depending on the duty cycle usage of the application backup battery.

Table 1-1. Typical Lifetime Estimation for Common Storage Elements on VDDBU

Storage Element	Capacity	Backup Consumption at 25°C		Battery Self-Discharge Current (10 years)	% Time the Application is in Backup Mode	Estimated Average Consumption(2)	Lifetime	
		Backup Mode	Running Mode				Hours	Years
CR2032 LiMin battery	210 mAh	4.5 µA	1.5 µA	0.2 µA	10%	2 µA	105K	12(1)
					50%	3.2 µA	66K	7.5
					90%	4.4 µA	48K	5.4
0.2F Super Cap charged at 3.3V	(3.3V-1.7V) * 0.2F = 0.32C	4.5 µA	1.5 µA	N/A	N/A	4.5 µA	20(3)	–

Note:

1. This is a theoretical lifetime calculation based on the battery capacity only. In practice, aging effects of the battery may limit this number.
2. Average consumption (µA)= (4.5 × d + 1.5 × (1-d)) + 0.2 where d is the percentage of time the application is in Backup mode.
3. In the case of a supercapacitor, it is assumed that this element is fully recharged between two backup periods. Here, 20 hours is the time the SAMA5D2 can stay in Backup mode with this element.

1.1.2 Backup Self-Refresh (BSR) Mode

This mode is an extension of the previous mode with the application context saved in the external DDR memory operating in Self-refresh mode, in the perspective to restart the application faster.

In this mode, the VDDBU and VDDIODDR power inputs must be maintained, as well as the power inputs of the DDR component.

The system power consumption in BSR is mainly that of the DDR, therefore the choice of the DDR type is of prime importance. In a similar way, the choice of the regulator(s) that maintain(s) DDR supplies in BSR mode should optimize efficiency at low current (see Conclusion).

1.1.3 ULP0, ULP1 and Idle Modes

In these three low-power modes, all SAMA5D2 power supplies are applied within their operating range. Power saving is achieved by reducing the frequency or stopping the clock signals of the processor and/or its peripherals.

Each mode is described below. An approximate wake-up time is given to ease understanding. For accurate values, refer to SAMA5D2 Series data sheet, section Electrical Characteristics.

- In Idle mode, only the processor clock is stopped and all peripherals are still operating. When exiting this mode, the processor operates at full speed. Typically, a few processor clock cycles are needed to enter and exit this mode. In a Linux® environment, this corresponds to Suspend-to-Idle.
- In ULP0 mode, the processor is stopped and its peripherals operate at a very low frequency (from a few kHz to a few MHz). At wake-up from this mode, the processor restarts at this very low frequency. Power consumption can be optimized by reducing the frequency at the expense of a longer wake-up time. In this mode, the processor is placed in Wait-For-Interrupt (WFI) state, therefore any interrupt source can trigger return to normal operation.
- In ULP1 mode, the processor clock and the peripheral clocks are stopped. Prior to entering this mode and to stop the clocks, the source of every clock is switched to the Main RC oscillator running at a typical 12 MHz. The Power Management Controller (PMC) then stops this oscillator at ULP1 entry. Upon a wake-up event, the PMC automatically restarts this oscillator, thus clocking back the device to 12 MHz. Unlike ULP0, only a few events like wake-up pins, USB resume and others (see the SAMA5D2 Series data sheet) can wake up the device from ULP0. This mode achieves both a very low current consumption on VDDCORE (power of the core, typically less than 0.5 mW at room temperature) and a fast wake-up time of a few microseconds.

The following table summarizes how the core, peripherals and DDR are powered and clocked in each mode.

Table 1-2. Core and Peripheral Clock Versus Mode

Mode	Processor Core	Peripherals/Internal SRAM Memory
Idle	Not Clocked (WFI)	Clocked

.....continued		
Mode	**Processor Core**	**Peripherals/Internal SRAM Memory**
ULP0	Not Clocked (WFI)	Clocked at low frequency
ULP1	Not Clocked (WFE)	Not clocked

2. Power Supply Implementation for SAMA5D2 Low-Power Modes

2.1 Introduction to Hardware Implementations

From a power supply perspective, two cases must be considered to manage the SAMA5D2 low-power modes:

- In ULP0, ULP1 and Idle modes, all the device's power inputs operate within their specified range. As the power consumption is reduced, the power supply circuit can be switched to a power-saving mode.
- In BSR, all power supply inputs of the device are turned off, except VDDBU, VDDIODDR and those of the memory, that must be maintained. To manage this case, the application can either send an I^2C command to the PMIC or use a PIOBU (powered and clocked in the backup area) to signal the power supply to enter a specific powering case.

The following figures provide simplified timing diagrams at entry and exit of Backup and BSR modes. See the SAMA5D2 Series data sheet for complete information about power-up and power-down sequences.

Figure 2-1. Example of Backup Mode Entry and Exit

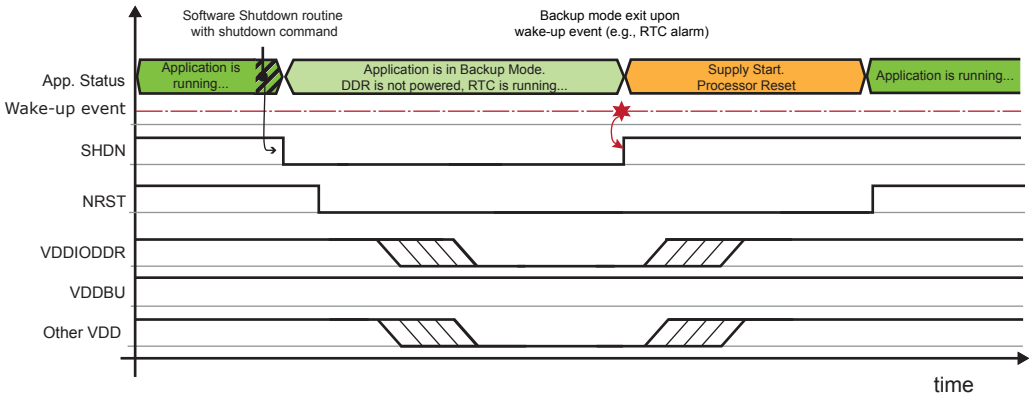
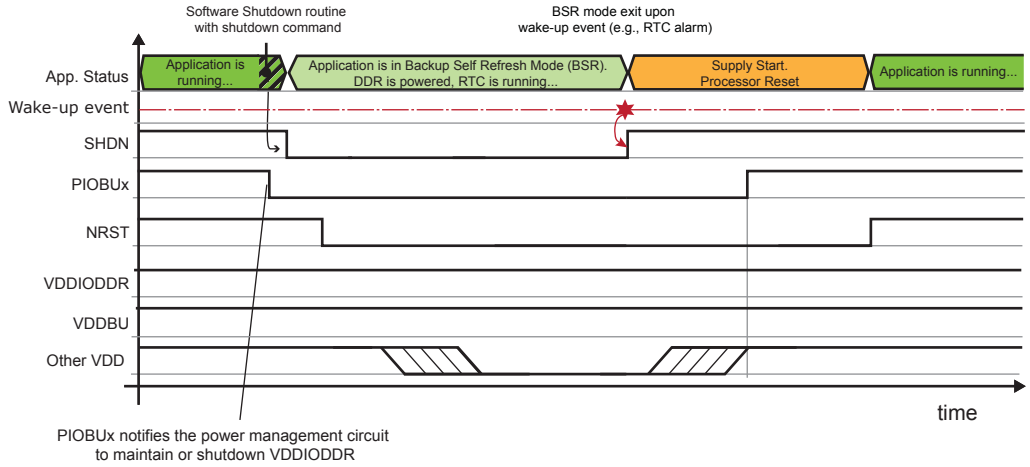


Figure 2-2. Example of BSR Mode Entry and Exit Using PIOBU



The following sections give an example of hardware implementation using a PMIC device or discrete components.

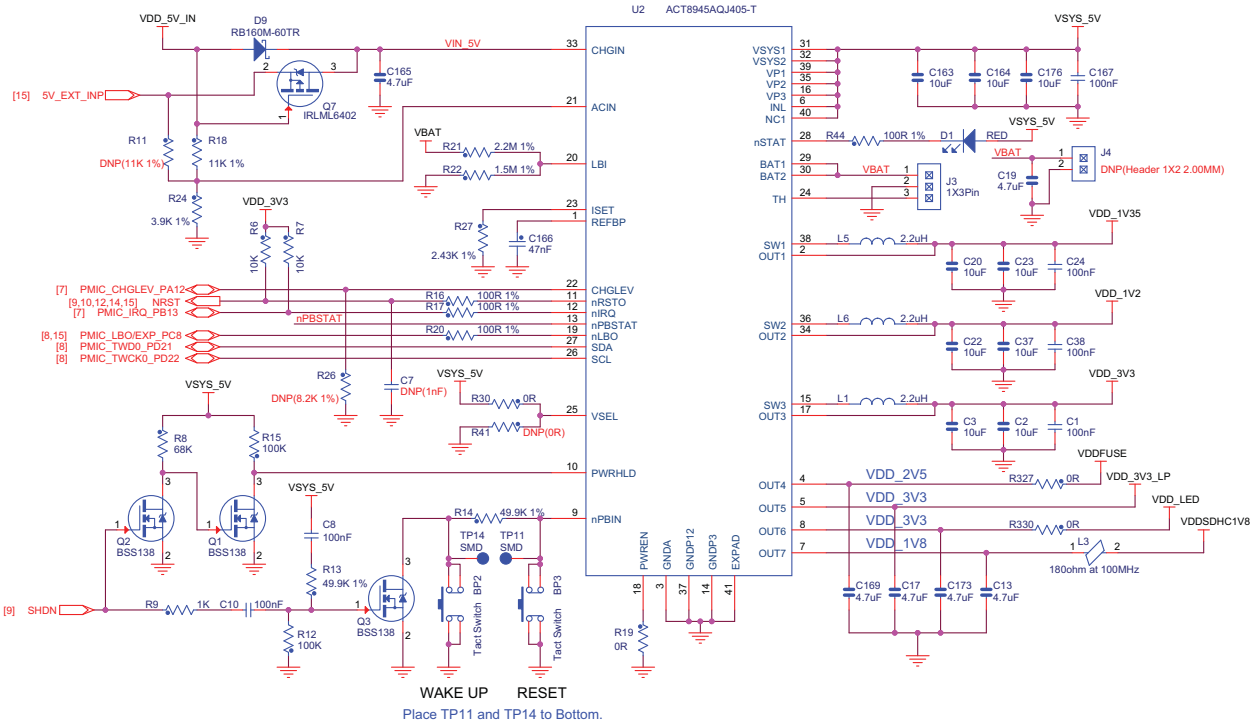
In both examples, the description aims at setting the Backup Self-refresh mode.

2.2 Hardware Implementation of BSR Mode Using a PMIC

The following figure illustrates the SAMA5D2 Xplained board power supply. The power management IC ACT8945A provides the power to the processor and to its external DDR3L-SDRAM memory. Prior to entering BSR mode, the application sends to the PMIC the desired channel configuration in BSR mode over the I²C interface.

Typically, the software requests the PMIC to maintain VOUT1 (VDD_1V35, power of the DDR3L memory) and to turn off other channels. This configuration is executed when the SHDN pin is cleared by the processor.

Figure 2-3. PMIC Schematic

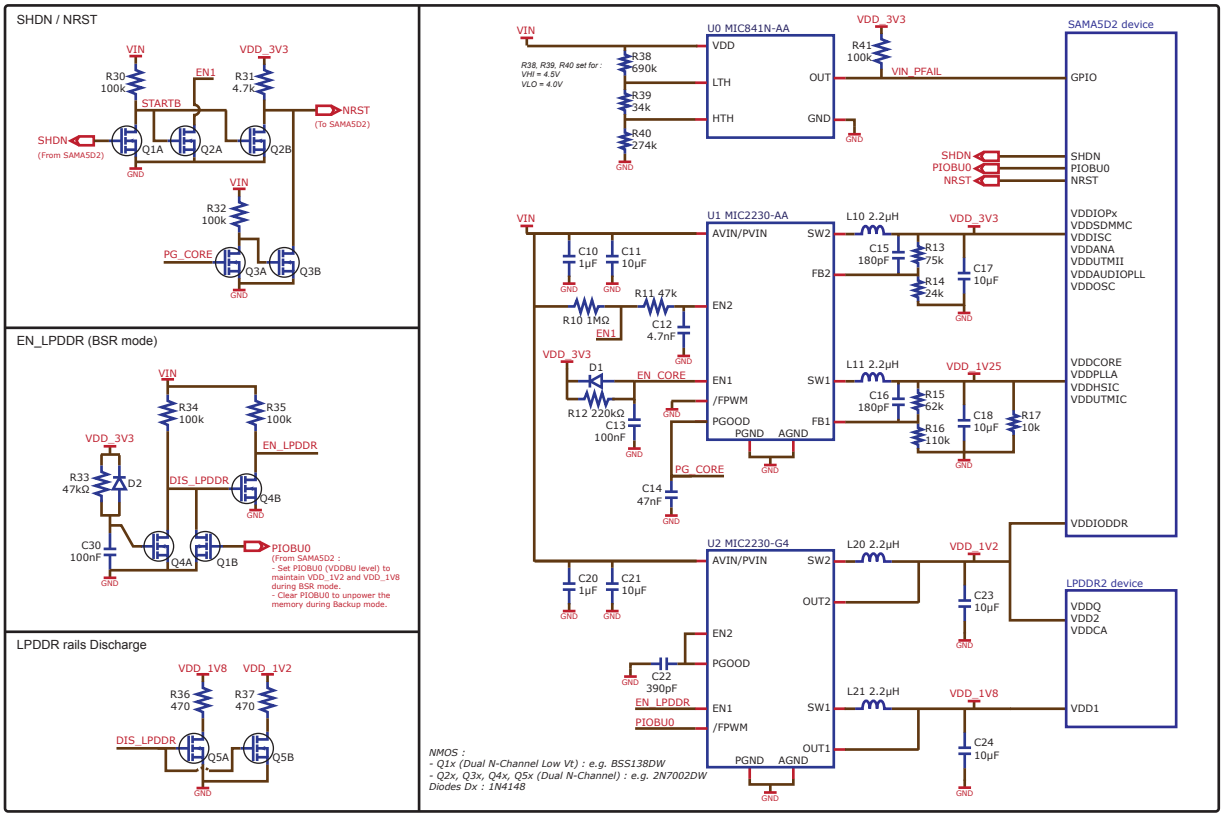


2.3 Hardware Implementation of BSR Mode Using Discrete Components

The following figure provides a discrete components power supply schematic optimized for SAMA5D2 systems equipped with LPDDR2 memories.

The input voltage VIN can range from 3.0V to 5.5V. This schematic uses most of the principles (sequencing, reset assertion, etc.) described in the application note “SAMA5D2 Discrete Power Supply Solution” (see Reference Documents). The SHDN pin of the Shutdown Controller (SHDWC) is used to start and stop the supply channels, and the PIOBU0 pin is dedicated to the BSR mode management. When PIOBU0 is set to high level, the EN_LPDDR signal is maintained high whatever the level on SHDN. Therefore, both the VDD_1V2 and VDD_1V8 rails connected to the LPDDR2 device and to VDDIODDR of the SAMA5D2 are maintained. As the LPDDR2 devices have stringent power-off requirements, U0 is added to detect early power input loss, and Q5A/Q5B help discharge VDD_1V2 and VDD1V8 promptly.

Figure 2-4. Discrete Components Schematic



3. Generic Recommendations to Set the System to Low-Power Modes

As the SAMA5D2 power consumption can be as low as a few micro-amps in some modes, it is of prime importance to ensure that no leakage current is lost outside the device at system level. The following generic recommendations apply prior to entering one of the low-power modes:

- Verify the state of the external components that will remain powered during the low-power mode. It may be necessary to set some of those in Standby mode or Low-power mode, or even turn them off. In Backup or BSR mode, it may be convenient to remove the power to the components that are not used in this mode. This should be assessed on a case-by-case basis.
- Verify the state of each IO of the device. In particular, any component connected to the SAMA5D2, including pull-up and pull-down resistors, may create a leakage path. In addition, even though an I/O is configured as an input, forcing this line with an active clock (e.g. the serial clock from a master on a serial link, or the clock from an Ethernet PHY) would result in some power consumption in the VDDCORE domain.
- To avoid leakages in the VDDBU power domain, the I/Os of the MPU belonging to the VDDBU power domain (WKUP, PIOBUx, RXD, COMPP, COMPN and SHDN) must not be directly connected to the I/Os of an external component (e.g. PMIC) unless carefully verified. It is good practice to isolate those lines with an external buffer (e.g. a simple NMOS transistor). In case of direct connection, leakage paths from the VDDBU power domain to the main power domain may be created through the ESD protection diodes of these I/Os. See examples below.

Figure 3-1. SHDN Pin Connection

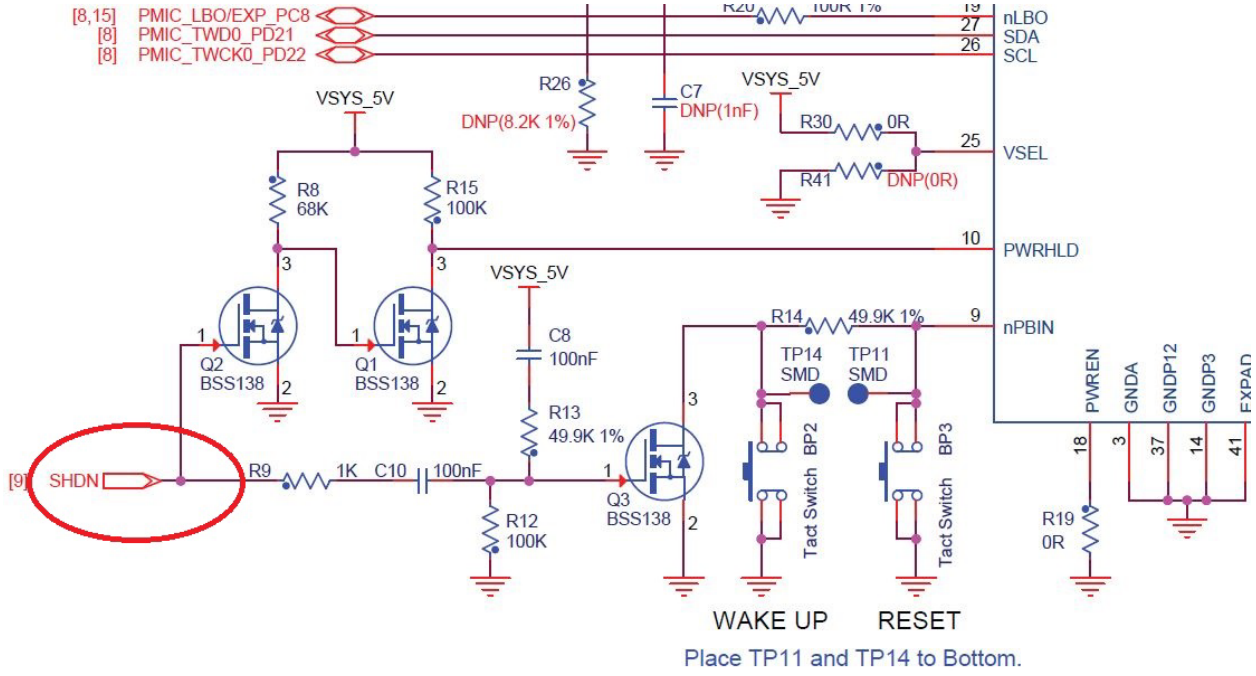
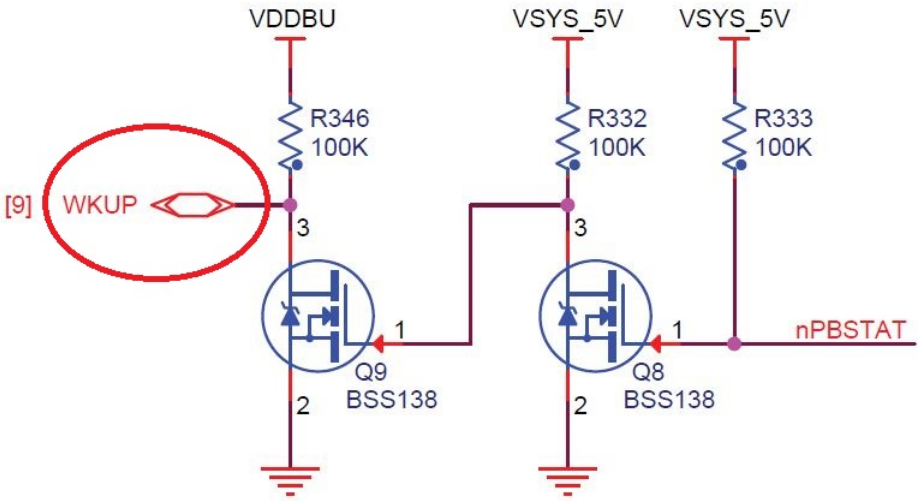


Figure 3-2. WKUP Pin Connection



- Before entering Backup mode or BSR mode, and thus turning off the power supplies, it is good practice to slow down the device operation, to avoid maximum execution speed at power supply collapse. To do so, it is recommended to switch the master clock (MCK) source to the slow clock.
- If the power supply circuit features a specific low-power mode, it may be appropriate to enter this mode.
- To reduce extra power consumption when using the USB in Idle mode(1), the different ports must be forced to Suspend mode by setting the SFR_OHCIICR.SUSPEND_x bit.

Note:

1. USB device: set the DETACH bit to 1 and the PULLD_DIS bit to 0 in register UDPHS_CTRL. Any other combinations of these two bits may cause additional consumption.
   USB host: force all USB host ports to suspend by setting the SUSPEND_A, SUSPEND_B and SUSPEND_C bits to 1 in register SFR_OHCIICR, at the end of the USB suspend routine.

4. Bare Metal Software Implementation

The following bare metal examples provide the detailed procedure to enter each SAMA5D2 low-power mode. The project used in this section is based on the SAMA5D2 Software Package (IAR7.40).

Some of the methods to reduce power consumption are:

- Adjust the system clocks at run time
- Put devices in Sleep mode based on their usage
- Put the memory in Self-refresh mode
- Put the system in Backup mode

The above methods are explained in detail in this section, where power management for low-power mode is split in three parts:

- Backup modes
- Ultra Low-power modes
- Idle mode

4.1 Backup and Backup Self-Refresh Modes

4.1.1 How to Enter Backup Mode

Backup mode is entered by shutting down all power rails except VDDBU, so that only the backup area is running. The core, peripherals and internal memory are turned off and, in the specific case of SAMA5D2 Xplained board, all external components are unpowered.

1. Configure the wake-up sources in the Shutdown Controller.
2. Slow down the device operation by switching the master clock (MCK) to Slow Clock.
3. Enter Backup mode by asserting the SHDN pin to notify the PMIC or discrete components that all powers except VDDBU can be turned off.

Sample code:

```
// Switch back to VDDBU power on backup area instead of VDDANA.
SFRBU->SFRBU_PSWBUCTRL = SFRBU_PSWBUCTRL_WPKEY_PASSWD;
/* config the wakeup */
shdwc_configure_wakeup();
/* clear status */
(void)shdwc_get_status();
/* PCK = MCK = 32 kHz */
/* Select Slow Clock as input clock for PCK and MCK */
PMC->PMC_MCKR = (PMC->PMC_MCKR & ~PMC_MCKR_
/* enter backup mode */
shdwc_do_shutdown();
```

4.1.2 How to Exit Backup Mode

Backup mode exit is triggered by any of the following events:

- WKUP0
- WKUP1 (Security Module event)
- WKUP2 to WKUP9 pins (PIOBU0 to PIOBU7, level transition, configurable debouncing)
- Character received on a low-power UART receiver (RXLP)
- Analog comparison

- RTC alarm

When a wake-up event is detected, the SHDN pin is driven high automatically by the Shutdown Controller. On the SAMA5D2 Xplained board, this makes the PMIC turn on all SAMA5D2 power supplies.

Once the SAMA5D2 is powered and the NRST pin is released, the ROM code is executed and loads the bootstrap from a storage media (eMMC, SD card, etc.) in the internal SRAM. The bootstrap is then executed out of the internal SRAM.

4.1.3 How to Enter Backup Self-Refresh (BSR) Mode

BSR mode is an extension of Backup mode. To enter this mode, the DDR memory must first be set to Self-refresh mode, and both VDDBU and VDDIODDR must be powered. The sequence to enter BSR mode is detailed below. Steps 1 and 2 can be executed from the DDR, while steps 3 to 8 must be executed from the internal SRAM.

1. Software saves all context information to resume (application-dependent).
2. Copy to SRAM and execute out of SRAM the routine to set the DDR to Self-refresh mode and to shut down the device (as soon as the DDR is in Self-refresh, accessing the DDR is no longer possible).
3. Put the DDR in Self-refresh mode and wait until the self-refresh status is OK (refer to the MPDDRC section in the SAMA5D2 Series data sheet).
4. Set the BUMEN bit, in the SFRBU_DDRBUMCR register, to isolate the DDR I/O segment from the VDDCORE shutdown.
5. Configure the wake-up sources in the Shutdown Controller.
6. Switch the system clock to Slow Clock.
7. Assert PIOBU0 or send I^2C controls to the PMIC (this configures the PMIC so that it maintains VDDIODDR for the next shutdown period).
8. Enter BSR mode by asserting the SHDN pin.

All external components must be turned off, except the DDR memory.

When a PMIC power solution is used, the PMIC should be first configured to maintain the appropriate rails (e.g. VDD_1V35 on SAMA5D2 Xplained board) before being shut down.

When a discrete components solution is used, the PIOBU must be controlled to shut down all power except VDDIODDR and VDD_1V35.

Sample code:

```
// Data and configuration to be saved prior
// to entering BSR mode (Application dependent) */
...
/* Set the DDR in Self Refresh */
MPDDRC->MPDDRC_LPR = MPDDRC_LPR_LPCB_SELFREFRESH;
//Check if self-refresh is done; if not, continue.
while(!(MPDDRC->MPDDRC_LPR & MPDDRC_LPR_SELF_DONE));
// Enable DDR Backup Mode
SFRBU->SFRBU_DDRBUMCR = SFRBU_DDRBUMCR_BUMEN;
// Disable the DDR Controller clock signal at PMC level for the periph
PMC->PMC_PCR = ( PMC_PCR_CMD | PMC_PCR_GCKCSS_MCK_CLK | (ID_MPDDRC));
//Disable ddrclk
PMC->PMC_SCDR |= PMC_SCDR_DDRCK;
// Configure PMIC to be in BSR
board_cfg_pmic_ulpm(selfrefresh, backup);
/* config the wakeup */
shdwc_configure_wakeup();
/* clear status */
(void)shdwc_get_status();
// Switch back to VDDBU power on backup area.
SFRBU->SFRBU_PSWBUCTRL = SFRBU_PSWBUCTRL_WPKEY_PASSWD;
```

```
/* PCK = MCK = 32 kHz */
/* Select Slow Clock as input clock for PCK and MCK */
PMC->PMC_MCKR = (PMC->PMC_MCKR & ~PMC_MCKR_CSS_Msk) | PMC_MCKR_CSS_SLOW_CLK;
    while (!(PMC->PMC_SR & PMC_SR_MCKRDY));
/* enter backup mode */
shdwc_do_shutdown();
```

4.1.4 How to Exit BSR Mode

Exiting BSR mode is initiated by any of the following events:

- WKUP0
- WKUP1 (Security Module event)
- WKUP2 to WKUP9 pins (PIOBU0 to PIOBU7, level transition, configurable debouncing)
- Character received on a low-power UART receiver (RXLP)
- Analog comparison
- RTC alarm

When a wake-up event is detected, the Shutdown Controller drives the SHDN pin automatically. This makes the power supply restart and, when the NRST pin is released, the ROM boot sequence is started.

The ROM code is executed and loads the customer bootstrap in internal SRAM. The bootstrap must check the state of the BUMEN bit in the SFRBU_DDRBUMCR register and re-initialize the DDR controller. The DDR memory automatically exits Self-refresh mode when an access in the DDR memory space occurs. The following sequence must be executed to connect DDR pads to the CPU domain.

Sample code:

```
if ((SFRBU->SFRBU_DDRBUMCR & SFRBU_DDRBUMCR_BUMEN) != 0)
/* Connect the DDR Pads to the CPU domain, VCCCORE */
SFRBU->SFRBU_DDRBUMCR &= ~SFRBU_DDRBUMCR_BUMEN;
```

4.2 Ultra Low-Power Mode

Ultra Low-power mode (ULP) includes two submodes: ULP0 mode and ULP1 mode.

As described in Table 1-2, the difference between ULP0 and ULP1 is the presence (ULP0) or the absence (ULP1) of clocks in the system. In both modes, the SAMA5D2 is fully powered (i.e., all its power inputs are properly supplied).

To further decrease the system power consumption in ULP0 or ULP1, these SAMA5D2 ULP modes may be combined with some power saving techniques applied to other components in the system. In some cases, it may even be possible to power down some of these components.

4.2.1 How to Enter ULP0 Mode

The sequence to enter ULP0 mode is detailed below. The code used to enter this mode must be executed out of the internal SRAM.

1. Set the DDR to Self-refresh mode.
2. Set the interrupts to wake up the system.
3. Disable all unused peripheral clocks.
4. Set the I/Os to an appropriate state, and disable the USB transceivers if they are not used (refer to the Special Function Registers (SFR) section in the SAMA5D2 Series datasheet).
5. Switch the system clock to Slow Clock.
6. Disable the PLLs, the main crystal oscillator and the 12 MHz RC oscillator.

7. Enter the Wait for Interrupt mode and disable the PCK clock in the PMC_SCDR register.

Sample code:

```
 /* Back up IOs and USB transceivers */
read_reg[0] = PMC->PMC_PCSR0;
read_reg[1] = PMC->PMC_PCSR1;
read_reg[2] = PMC->PMC_SCSR;
read_reg[3] = PMC->CKGR_UCKR;

/* Set the DDR in sSelf Refresh */
MPDDRC->MPDDRC_LPR = MPDDRC_LPR_LPCB_SELFREFRESH;
//Check if self-refresh is done; if not, continue.
while(!(MPDDRC->MPDDRC_LPR & MPDDRC_LPR_SELF_DONE));

/* Disable the USB transceivers and all peripheral clocks */
board_save_misc_power();

/* config the wakeup */
shdwc_configure_wakeup();

/* clear status */
(void)shdwc_get_status();

/* config wake up sources and active polarity */
pmc_set_fast_startup_polarity(0, PMC_FSPR_FSTP0);
pmc_set_fast_startup_mode(PMC_FSMR_FSTT0 | PMC_FSMR_FSTT2 | PMC_FSMR_LPM);

/* Select Slow Clock as input clock for PCK and MCK */
PMC->PMC_MCKR = (PMC->PMC_MCKR & ~PMC_MCKR_CSS_Msk) | PMC_MCKR_CSS_SLOW_CLK;
while (!(PMC->PMC_SR & PMC_SR_MCKRDY));

//arch_irq_disable();
asm("cpsid if");
/* enter ULP0 mode */
asm("WFI");

/* Restore default PCK and MCK */
pmc_set_custom_pck_mck(&clock_test_setting[0]);
_restore_console();

/* Restore IOs and USB transceivers */
PMC->PMC_PCER0 = read_reg[0];
PMC->PMC_PCER1 = read_reg[1];
PMC->PMC_SCER  = read_reg[2];
PMC->CKGR_UCKR = read_reg[3];

// Switch VDDBU power on backup areato VDDANA decreasing the power consumption on VDDBU
battery
SFRBU->SFRBU_PSWBUCTRL = SFRBU_PSWBUCTRL_WPKEY_PASSWD | SFRBU_PSWBUCTRL_SSWCTRL;
/
```

4.2.2 How to Exit ULP0 Mode

The wake-up from ULP0 mode is triggered by any enabled interrupt. In this example, Push Button BP1 is used as the wake-up source. The software resumes by executing the instruction following the WFI instruction by configuring the Arm core with the command asm(”cpsid if”); . Hence, in the previous sample code, the first line of code executed is “pmc_set_custom_pck_mck” to make sure that the DDR memory is clocked at the correct frequency. The PMC registers are configured as they were before entering ULP0 mode, then Self-refresh mode is disabled and the code can continue and access the DDR in normal mode.

4.2.3 How to Enter ULP1 Mode

The sequence to enter ULP1 mode is detailed below. The code used to enter this mode must be executed out of the internal SRAM.

1. Set the DDR to Self-refresh mode.

2. Set the events to enable a system wake-up.
3. Disable all peripheral clocks.
4. Set the I/Os to an appropriate state and disable the USB transceivers.
5. Switch the system clock to the 12 MHz RC oscillator.
6. Disable the PLLs and the main oscillator.
7. Enter ULP1 mode by either:
   – setting the CKGR_MOR.WAITMODE bit, or
   – setting the PMC_FSMR.LPM bit and executing the processor WaitForEvent (WFE) instruction.

Then, immediately after setting the WAITMODE bit or using the WFE instruction, wait for the PMC_SR.MCKRDY bit to be set.

Sample code:

```
/* Back up IOs and USB transceivers */
read_reg[0] = PMC->PMC_PCSR0;
read_reg[1] = PMC->PMC_PCSR1;
read_reg[2] = PMC->PMC_SCSR;
read_reg[3] = PMC->CKGR_UCKR;
/* Set the DDR in Self Refresh */
MPDDRC->MPDDRC_LPR = MPDDRC_LPR_LPCB_SELFREFRESH;
//Check if self-refresh is done; if not, continue.
while(!(MPDDRC->MPDDRC_LPR & MPDDRC_LPR_SELF_DONE));
/* Disable the USB transceivers and all peripheral clocks */
board_save_misc_power();
/* config the wakeup */
shdwc_configure_wakeup();
/* clear status */
(void)shdwc_get_status();

/* config wake up sources and active polarity */
pmc_set_fast_startup_polarity(0, PMC_FSPR_FSTP0);
pmc_set_fast_startup_mode(PMC_FSMR_FSTT0 | PMC_FSMR_FSTT2 | PMC_FSMR_RTCAL | PMC_FSMR_LPM);

/* Disable the PLLs and the main oscillator */
/* ultra low power mode 1, RC12 is selected for Main Clock */
PMC->CKGR_MOR = (PMC->CKGR_MOR & ~CKGR_MOR_KEY_Msk) | CKGR_MOR_MOSCRCEN | CKGR_MOR_KEY_PASSWD;
/* Wait internal 12MHz RC Startup Time for clock stabilization */
while (!(PMC->PMC_SR & PMC_SR_MOSCRCS));

PMC->PMC_MCKR = (PMC->PMC_MCKR & ~PMC_MCKR_CSS_Msk) | PMC_MCKR_CSS_MAIN_CLK;
while (!(PMC->PMC_SR & PMC_SR_MCKRDY));

/* switch MAIN clock to internal 12MHz RC */
PMC->CKGR_MOR = (PMC->CKGR_MOR & ~(CKGR_MOR_MOSCSEL | CKGR_MOR_KEY_Msk)) |
CKGR_MOR_KEY_PASSWD;
PMC->CKGR_MOR = (PMC->CKGR_MOR & ~(CKGR_MOR_MOSCXTEN | CKGR_MOR_MOSCXTBY | CKGR_MOR_KEY_Msk))
| CKGR_MOR_KEY_PASSWD;

/* enter ULP1 */
asm("WFE");
asm("WFE");

/* wait for the PMC_SR.MCKRDY bit to be set. */
while ((PMC->PMC_SR & PMC_SR_MCKRDY) == 0);

/* Restore default PCK and MCK */
pmc_set_custom_pck_mck(&clock_test_setting[0]);
_restore_console();

/* Restore IOs and USB transceivers */
PMC->PMC_PCER0 = read_reg[0];
PMC->PMC_PCER1 = read_reg[1];
PMC->PMC_SCER  = read_reg[2];
PMC->CKGR_UCKR = read_reg[3];

// Switch VDDBU power on backup areato VDDANA decreasing the power consumption on VDDBU
```

```
battery
   SFRBU->SFRBU_PSWBUCTRL = SFRBU_PSWBUCTRL_WPKEY_PASSWD | SFRBU_PSWBUCTRL_SSWCTRL;
```

4.2.4 How to Exit ULP1 Mode

The method to exit ULP1 mode is similar to the method described for ULP0 mode.

In the above sample code, the RTC is used as the wake-up source.

The software resumes by executing the instruction following the WFE instruction. Hence, in the previous sample code, the first piece of code executed is “pmc_set_custom_pck_mck” to make sure that the DDR memory is clocked at the correct frequency. The PMC registers are configured as they were before entering ULP1 mode, then Self-refresh mode is disabled and the code can continue and access the DDR in normal mode.

4.3 Idle Mode

As the purpose of Idle mode is to save the device power consumption, all power supplies operate within their specified range.

Power consumption in this mode is application-dependent, but it can be reduced by enabling the Dynamic Clock Gating in the L2 Cache Controller (set L2CC_POWCR.DCKGATEN = 1).

4.3.1 How to Enter Idle Mode

To enter Idle mode, disable PCK and execute the Wait for Interrupt (WFI) instruction.

Sample code:

```
//Disable PCK
PMC->PMC_SCDR = PMC_SCDR_PCK;
//Enter Idle mode
asm(“wfi”);
```

4.3.2 How to Exit Idle Mode

The processor can be awakened from Idle mode by an interrupt. The system resumes where it was before entering WFI mode.

Sample code:

```
static void configure_buttons(void)
{
   int i = 0;
   for (i = 0; i < ARRAY_SIZE(button_pins); ++i){
   //Configure PIOs as inputs.
   pio_configure(&button_pins[i], 1);
   //Adjust PIO debounce filter parameters, here we have set a 10 Hz filter,
   //this is an example..
   pio_set_debounce_filter(&button_pins[i], 10);
   //Initialize PIOs interrupt with its handlers,
   //see PIO definition in board.h.
   pio_configure_it(&button_pins[i]);
   pio_add_handler_to_group(button_pins[i].group,
   button_pins[i].mask, pio_handler);
   //Enable PIO line interrupts.
   pio_enable_it(button_pins);
   }
}
```

In the above sample code, Push Button BP1 is used as the wake-up source, so, before entering Idle mode, the corresponding PIO must be configured as an interrupt source.

5. Linux Software Implementation

The following section is based on Linux4SAM version 5.7. Refer to http://www.linux4sam.org.

The Linux Power Management (PM) framework provides multiple ways of saving power. Some of the methods used to reduce power consumption are:

- Using a tickless kernel
- Adjusting system clocks at run time
- Putting devices to sleep based on their usage
- Suspending system to memory
- Putting the system in Hibernate mode

The Linux power management options are split in two main categories:

- Runtime power management
- System sleep model

Runtime power management refers to switching devices to sleep states (by disabling clocks or switching to advanced power management hardware-related states).

The system sleep power management model refers to executing system-related power management routines by putting the whole system in a power saving state.

The power management options discussed in Section 5.2 are part of the system sleep power management model. The system with a SAMA5D2-based SoC running Linux operating system can switch to a low-power mode and save various amounts of power depending on the mode chosen.

This application note describes only the system sleep model, since this is the one related to the low-power modes described in Section 1.1.

5.1 Linux Power Management Core (System Sleep Model)

The power management core implementation offers the infrastructure to allow systems to be switched from an active running mode to a low-power mode (Suspend operation), and from a low-power mode to an active mode (Resume operation) without affecting system functionality.

Standard Linux implementation provides four ways to save power. The Linux sleep states are:

- Suspend-to-Idle
- Power-On Suspend
- Suspend-to-RAM
- Suspend-to-Disk

This section describes the Power-On Suspend and Suspend-to-RAM power saving modes.

5.1.1 File System Interfaces

Linux file system interfaces enable the user to switch to the required power management state, to set power management parameters and to retrieve statistics.

The main Linux file system interfaces for power management are:

```
/sys/power
/sys/kernel/debug/sleep_time
/sys/kernel/debug/suspend_stats
```

```
/sys/kernel/debug/wakeup_sources
/sys/devices/.../power/wakeup
```

```
/sys/power
```

is the main interface used to control switching to a low-power sleep state. The options are:

```
ls -l /sys/power/
-rw-r--r-- 1 root root 4096 Jan 12 17:37 pm_async
-rw-r--r-- 1 root root 4096 Jan 12 17:37 pm_freeze_timeout
-rw-r--r-- 1 root root 4096 Jan 12 17:37 pm_print_times
-rw-r--r-- 1 root root 4096 Jan 12 17:37 pm_test
-r--r--r-- 1 root root 4096 Jan 12 17:37 pm_wakeup_irq
-rw-r--r-- 1 root root 4096 Jan 12 17:37 state
-rw-r--r-- 1 root root 4096 Jan 12 17:37 wakeup_count
```

The `/sys/power/state` interface is the principal power management file system interface and triggers the transition to a power saving state. On SAMA5D2 Linux based systems, the supported sleep states are:

```
cat /sys/power/state
freeze standby mem
```

To switch to one of these states, write the string corresponding to the required state into the `/sys/power/state` file, as indicated in Table 5-1.

Table 5-1. Linux Sleep States and Commands

Linux Sleep States	Command to Enter Sleep State
Suspend-to-Idle	`echo freeze > /sys/power/state`
Power-On Suspend or Standby	`echo standby > /sys/power/state`
Suspend-to-RAM	`echo mem > /sys/power/state`
Suspend-to-Disk or Hibernation	NA as string “disk” not present in `/sys/power/state`

For an overview of the Linux sleep states, check the Documentation section on https://www.kernel.org.

A brief description of the file system interfaces is as follows:

- `/sys/power/pm_async`: allows suspend and resume callbacks of some devices to be executed in parallel with each other and in parallel with the main suspend thread
- `/sys/power/pm_freeze_timeout`: specifies how long it will take to freeze all freezable processes
- `/sys/power/pm_print_times`: used to print the time taken by devices to execute suspend and resume operations
- `/sys/power/pm_test`: used to test power management options
- `/sys/power/pm_wakeup_irq`: prints the wake-up IRQ
- `/sys/power/wakeup_count`: reading returns the number of wake-up events and writing aborts the current transition to a sleep state

For debugging purposes, files are provided in the `debugfs` file system:

- `/sys/kernel/debug/sleep_time` prints the time (in seconds) spent by the Sleep operation.
- `/sys/kernel/debug/suspend_stats` prints Suspend statistics.
- `/sys/kernel/debug/wakeup_sources` prints the current registered wake-up sources.

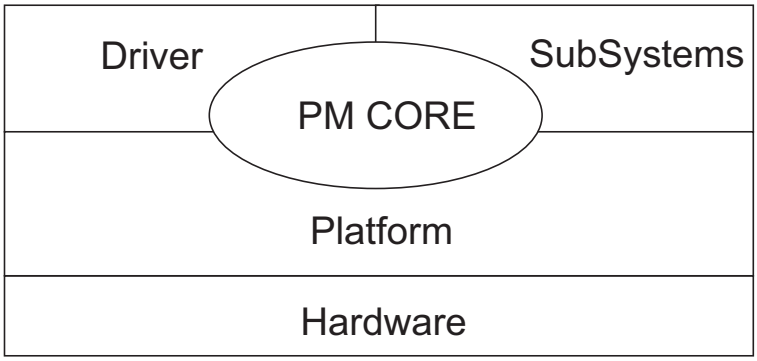
Resuming from a low-power state is performed based on events received from devices or external pins. Devices can be set up to act as wake-up sources. The Linux file system offers an interface to enable setting the wake-up capability for devices:

- `/sys/devices/.../power/wakeup` files can be set by writing “enabled” or “disabled” strings so that the device acts or not as a wake-up source.

5.1.2 Core Implementation

The following figure summarizes the main elements of Linux power management implementation.

Figure 5-1. Linux Power Management Architecture Overview



Terms used in the above figure are defined as follows, in a Linux context:

- Driver: a computer program used to control the hardware (IP/MCU/MPU)
- Subsystem: a code shared by drivers of the same type. Ex.: I^2C drivers share a common code that forms the Linux I^2C subsystem and implements common algorithms specific to I^2C devices
- Platform: code that implements platform-specific algorithms (e.g. SAMA5D2 implements its own platform code that controls SAMA5D2-specific features, e.g. Backup and Self-refresh features)
- PM core: power management code that implements common power management algorithms and activates subsystems, drivers and platform power management logic
- Hardware: the IP/MCU/MPU which is affected by the actions taken by drivers, subsystem, platform and PM core

The PM core contains generic power management code that is executed when entering/leaving a power saving state. The Freeze or Suspend-To-Idle state freezes the user space processes and leaves hardware in a working state. The other power saving states switch hardware to more power saving modes.

As shown in Figure 5-1, to enable switching to more advanced power saving states, three elements must provide PM hooks that will be accessed by the PM core in suspend/resume operations:

- device drivers
- subsystems where device drivers were registered
- platform code

The hardware must provide support for advanced power saving modes.

In the platform initialization phase of Linux, the platform code must register the platform-specific power suspend/resume code to the PM core. This is done by providing a structure such as the following to the PM core:

```
struct platform_suspend_ops {
int (*valid)(suspend_state_t state);
```

```
int (*begin)(suspend_state_t state);
int (*prepare)(void);
int (*prepare_late)(void);
int (*enter)(suspend_state_t state);
void (*wake)(void);
void (*finish)(void);
bool (*suspend_again)(void);
void (*end)(void);
void (*recover)(void);
};
```

The platform PM related code is executed in the final step of the suspend operation and in the first step of the resume operation. Depending on the power saving mode, the platform code puts the CPUs in different power saving modes.

The Linux kernel device model introduces devices, classes and buses. The PM core makes use of these notions to choose and apply the correct PM policies. In addition, the notions of parent and child devices are very useful in the suspend/resume procedure to suspend/resume the devices in the right order. In Linux, drivers implement a structure of type `struct devices`, and subsystems implement a structure of type `struct class` or `struct bus_type`. Every structure from the device driver model implements a structure of type `struct dev_pm_ops` as follows:

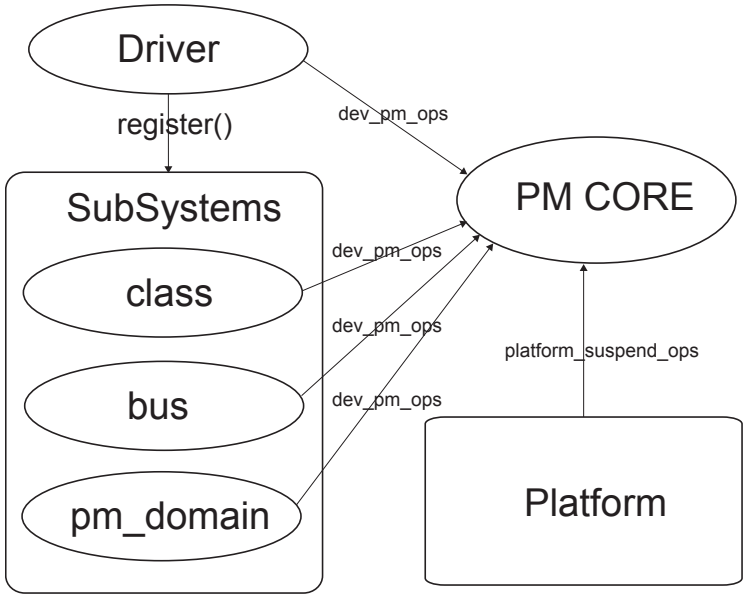
```
struct dev_pm_ops {
int (*prepare)(struct device *dev);
void (*complete)(struct device *dev);
int (*suspend)(struct device *dev);
int (*resume)(struct device *dev);
int (*freeze)(struct device *dev);
int (*thaw)(struct device *dev);
int (*poweroff)(struct device *dev);
int (*restore)(struct device *dev);
int (*suspend_late)(struct device *dev);
int (*resume_early)(struct device *dev);
int (*freeze_late)(struct device *dev);
int (*thaw_early)(struct device *dev);
int (*poweroff_late)(struct device *dev);
int (*restore_early)(struct device *dev);
int (*suspend_noirq)(struct device *dev);
int (*resume_noirq)(struct device *dev);
int (*freeze_noirq)(struct device *dev);
int (*thaw_noirq)(struct device *dev);
int (*poweroff_noirq)(struct device *dev);
int (*restore_noirq)(struct device *dev);
int (*runtime_suspend)(struct device *dev);
int (*runtime_resume)(struct device *dev);
int (*runtime_idle)(struct device *dev);
};
```

With drivers and subsystems implementing structures of type `struct dev_pm_os`, the PM core executes the correct hardware-related settings in order to switch peripherals to advanced power saving modes. Every subsystem or driver provides suspend/resume functionality via this structure to the PM core (Figure 5-2), allowing the PM core to switch peripherals to the right power state. Note that old platform drivers are only providing suspend and resume functions to the PM core via `struct platform_driver`. The PM code executes them.

In addition to the Linux device driver model, the PM core introduces the concept of power domain. A power domain contains one or more devices sharing reference clocks or power resources. Devices can be part of a power domain and power domains can be nested. Power domains also offer PM functionalities to the PM core via the `struct dev_pm_ops` structure (Figure 5-2).

Figure 5-2 shows that drivers and subsystems register PM operations to the PM core, and that also platform-related PM operations are registered to the PM core if specific power saving modes need to be implemented.

Figure 5-2. Linux Power Management Implementation Overview



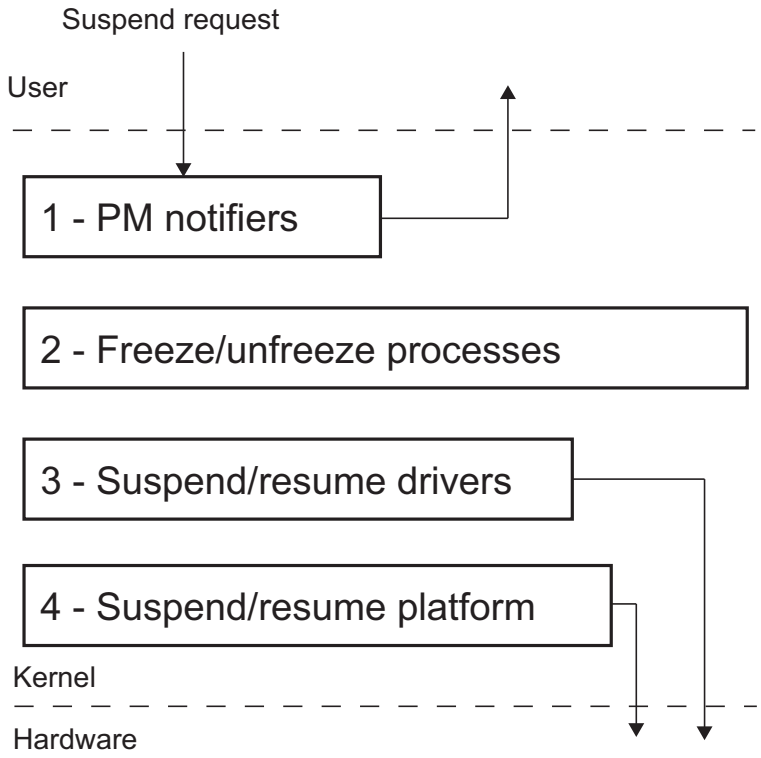
As shown in Figure 5-2, and depending on the Linux device driver model, more than one PM operation can be registered for a device from:

- the device power domain,
- the device class,
- the device bus,
- the device driver.

At suspend/resume, the PM core checks the registered PM operations in the order specified above, and executes the first PM operations matched. The registered PM operations are mutually exclusive, which means that only the PM operations for the device's power domain, class, bus or driver, is executed by the PM core in a suspend/resume sequence.

General suspend/resume steps are described in Figure 5-3.

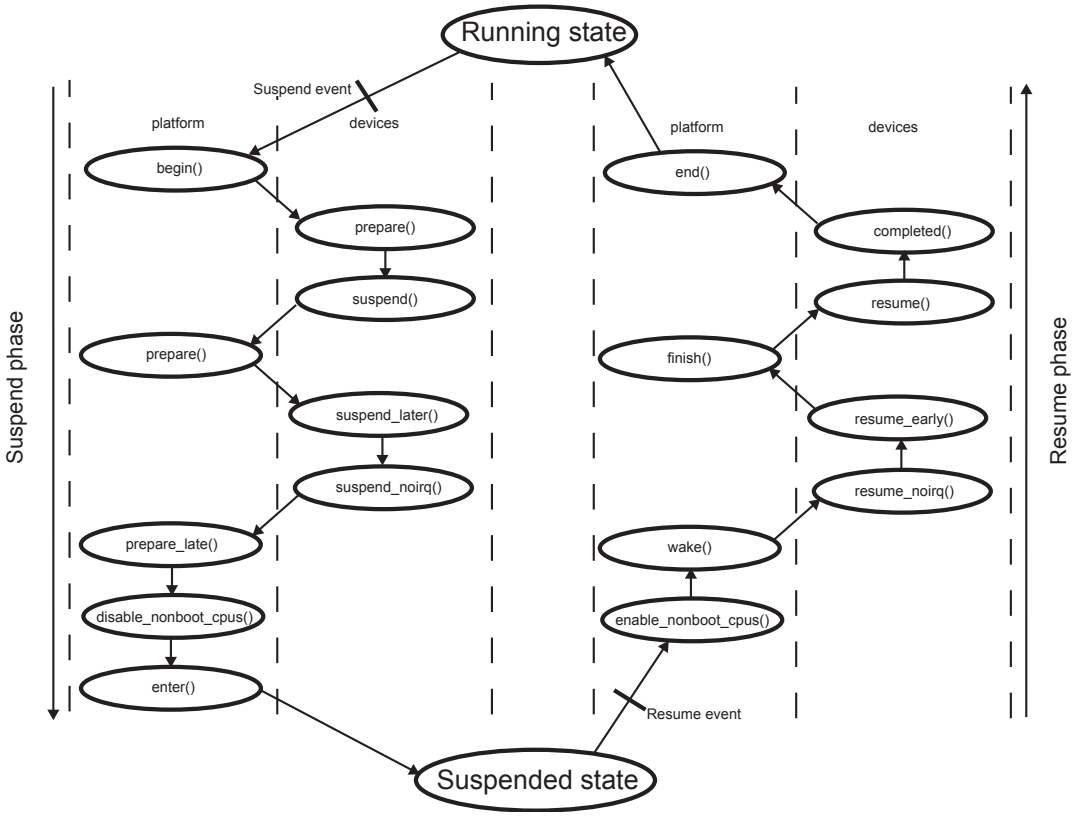
Figure 5-3. Suspend/Resume General Phases



As specified in Figure 5-3, the suspend request is initialized from the user space. Applications can register PM-related notifiers that are called in the first suspend phase. After this, the kernel freezes all processes (kernel and user space) and suspends devices and then the platform. The resume goes through the states described in Figure 5-3, but in reverse order.

Figure 5-4 details the suspend and resume of devices and platform specified in Figure 5-3. The figure shows the order of execution for platform PM operations (see `struct platform_suspend_ops`) and device PM operations (see `struct dev_pm_ops`) specified in platform initialization code and drivers' probe functions.

Figure 5-4. Suspend/Resume Detailed Phases



5.2 Power Management Implementation on SAMA5D2

5.2.1 Supported Modes

The Linux power management modes supported by SAMA5D2 SoCs are as follows:

- Idle
- ULP0
- ULP1
- Backup Self-refresh (BSR)

Idle mode puts the CPU core in Wait-For-Interrupt (WFI) state.

BSR, ULP0 and ULP1 modes are described in Section 4.1 and Section 4.2.

This section describes the implementation for BSR and ULP0/ULP1 modes.

5.2.2 AT91Bootstrap Support

In BSR mode, the CPU core (except the backup area) is shut down. To enable resuming the system, AT91Bootstrap support has been added. This support is based on the fact that the backup area is the only part that remains active while the system is suspended (except the DDR memory, which remains in Self-refresh mode). Linux and AT91Bootstrap communicate via the backup area (SECURAM).

The following figure shows how AT91Bootstrap and Linux interact with each other in the process of suspend and resume in BSR mode.

Figure 5-5. Linux and AT91Bootstrap Communication for Suspend/Resume for BSR Mode

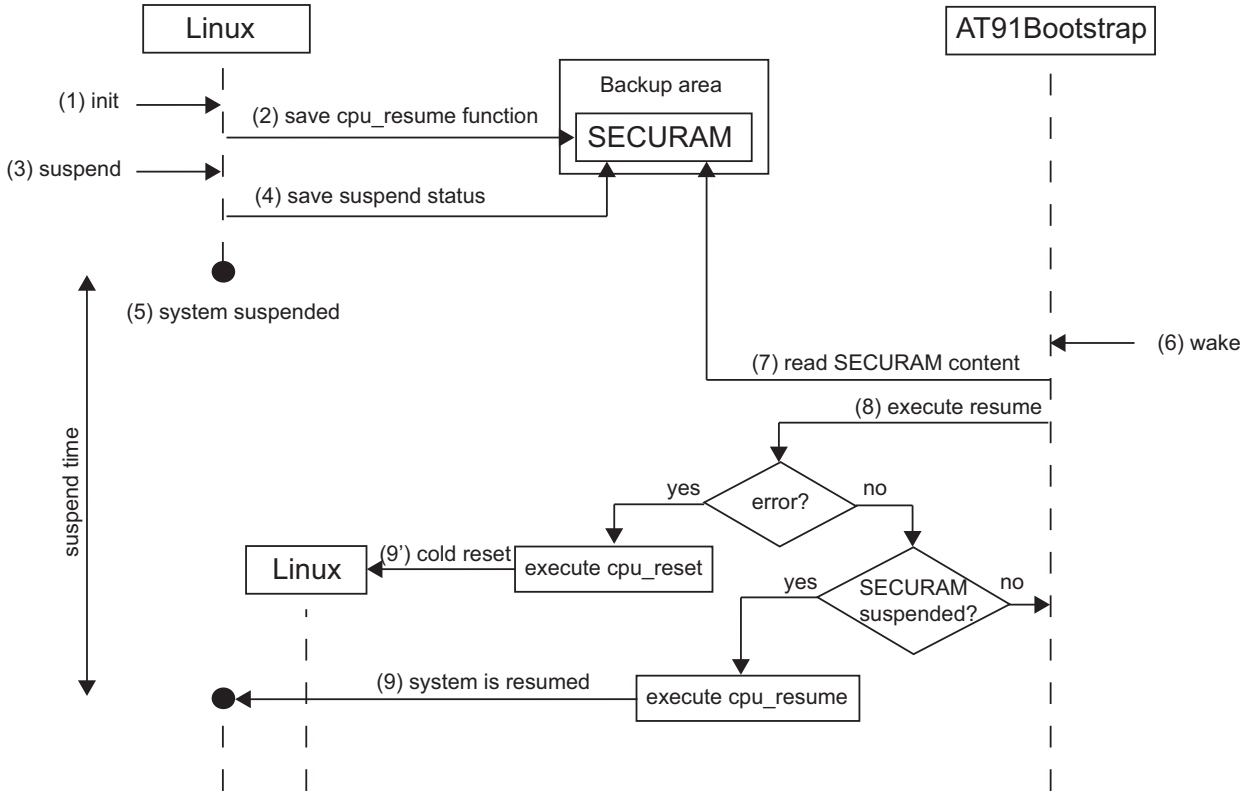


Figure 5-5 describes a standard suspend/resume sequence for BSR mode. As shown, when a suspend request is initialized, Linux saves the resume code and suspend status in SECURAM, and the core's power is turned off. When a wake-up event is detected, AT91Bootstrap starts and checks the suspend status and whether the memory content has changed. Depending on the suspend status read from SECURAM, AT91Bootstrap jumps to Linux execution, leading to executing the resume code. If AT91Bootstrap detects errors in the resume procedure, a cold reset is performed.

5.2.3 Linux Kernel Parameter

Previous sections explained that the Linux four sleep states and the four SAMA5D2 power management modes cannot be matched one to one. The SAMA5D2 SoC power management modes can be considered as slight variations of the same Linux “Power-On Suspend” sleep mode.

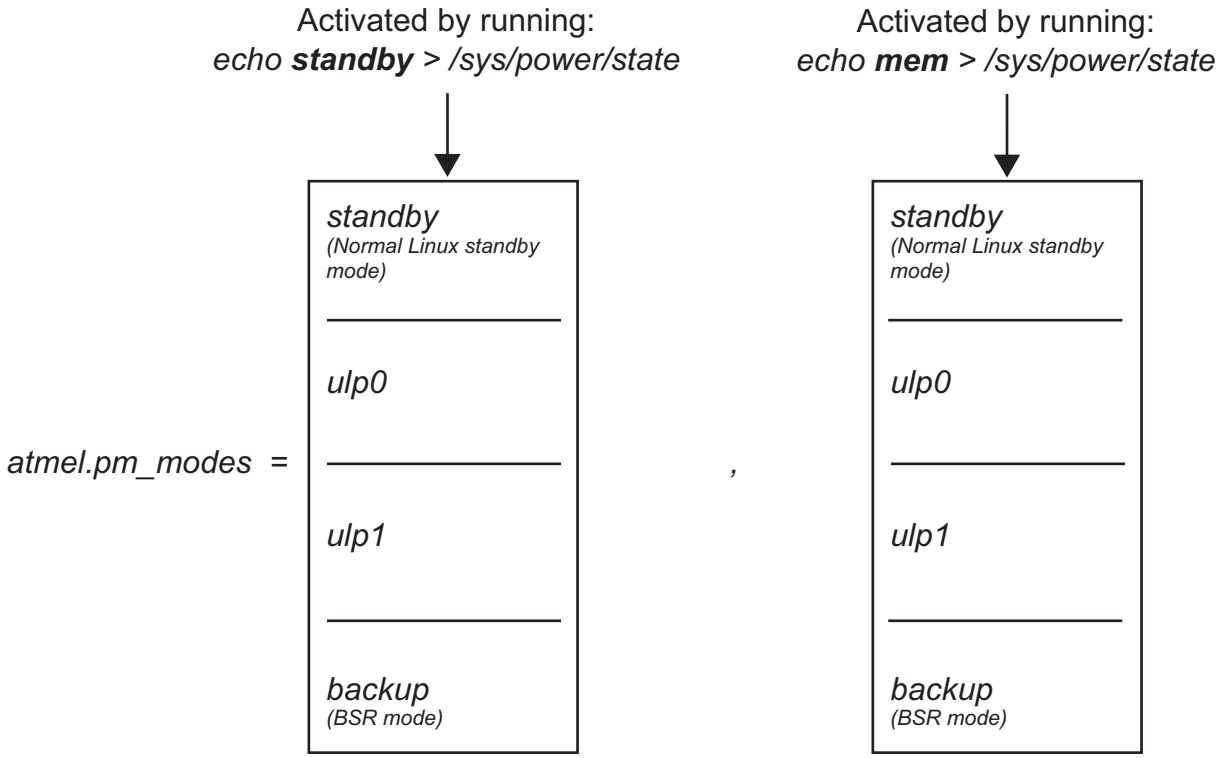
A technique has been developed to allow the user to choose which SAMA5D2 power management mode to map to one particular Linux sleep mode. This mapping is done statically at system initialization by using Linux kernel command line parameters. Those parameters, like any other kernel parameters, can be passed while starting Linux by the bootloader.

U-Boot is used in the Linux4SAM reference distribution and the new parameter is appended to the `bootargs` U-Boot variable.

The new kernel parameter, `atmel.pm_modes`, requires two arguments for the two SAMA5D2 low-power modes chosen.

The `atmel.pm_modes` format is as follows:

Figure 5-6. "atmel.pm_modes" Command Format



For example, to initialize the system with normal Linux standby and SAMA5D2 BSR mode, use the following value for atmel.pm_modes:

```
atmel.pm_modes=standby,backup
```

The value of `atmel.pm_modes` can be retrieved from a running Linux system by using the following command:

```
cat /proc/cmdline
```

This way, the user can check from Linux the mapping between Linux sleep states (Power-On Suspend, Suspend-to-RAM) and SAMA5D2 power management modes (Idle, ULP0, ULP1, BSR).

5.2.4 Linux Support

Implementation of the modes listed in Section 5.3.1 was done by adding suspend/resume support for both the driver's code and the platform's code (see Figure 5-1 and Figure 5-2). On drivers, structures of type `struct dev_pm_ops` were implemented. On platforms, a structure of type `struct platform_suspend_ops` was implemented.

5.2.5 Drivers Implementation

As explained in 5.1.2 Core Implementation, Linux power management suspends and resumes SoC peripherals by calling the functions provided by drivers in objects of type `struct dev_pm_ops` (or `platform_driver` for old device drivers). To implement the Linux system sleep model, drivers must implement the suspend and resume members in objects of type `struct dev_pm_ops` (or `struct platform_driver`).

As the CPU power is cut off in Backup and Self-refresh modes, the peripheral state (register contents) must be saved. Because memory remains in Self-refresh mode, its content is preserved, and the peripheral state can be saved. In the resume part of the driver, this state is restored from memory.

The UART driver is an example. As explained in 5.1.2 Core Implementation, the driver must provide to the PM core suspend and resume functions. In Linux kernel source code, the UART driver is located at `drivers/tty/serial/atmel_serial.c` and the suspend and resume functions are `atmel_serial_suspend` and `atmel_serial_suspend`. These functions are provided to the Linux PM core via an object of type `struct platform_driver`, as shown below:

```
static struct platform_driver atmel_serial_driver = {
   .probe = atmel_serial_probe,
   .remove = atmel_serial_remove,
   .suspend = atmel_serial_suspend,
   .resume = atmel_serial_resume,
   .driver = {
      .name = "atmel_usart",
      .of_match_table = of_match_ptr(atmel_serial_dt_ids),
   },
};
```

in the probe phase of the UART driver (see probe function, `atmel_serial_probe`) via the following line of code:

```
ret = platform_driver_register(&atmel_serial_driver);
```

Since in BSR mode, the CPU power is turned off and the DDR is kept in Self-refresh mode, the IP's state is saved in the DDR memory. The UART driver saves IP's state with the following code in the `atmel_serial_suspend` function:

```
if (atmel_is_console_port(port) && !console_suspend_enabled) {
/* Cache register values as we won't get a full shutdown/startup
* cycle*/
atmel_port->cache.mr = atmel_uart_readl(port, ATMEL_US_MR);
atmel_port->cache.imr = atmel_uart_readl(port, ATMEL_US_IMR);
atmel_port->cache.brgr = atmel_uart_readl(port, ATMEL_US_BRGR);
atmel_port->cache.rtor = atmel_uart_readl(port,atmel_port->rtor);
atmel_port->cache.ttgr = atmel_uart_readl(port, ATMEL_US_TTGR);
atmel_port->cache.fmr = atmel_uart_readl(port, ATMEL_US_FMR);
atmel_port->cache.fimr = atmel_uart_readl(port, ATMEL_US_FIMR);
}
```

In the resume phase, this state is restored as can be seen in the `atmel_serial_resume` function:

```
if (atmel_is_console_port(port) && !console_suspend_enabled) {
/* Cache register values as we won't get a full shutdown/startup
* cycle*/
atmel_port->cache.mr = atmel_uart_readl(port, ATMEL_US_MR);
atmel_port->cache.imr = atmel_uart_readl(port, ATMEL_US_IMR);
atmel_port->cache.brgr = atmel_uart_readl(port, ATMEL_US_BRGR);
atmel_port->cache.rtor = atmel_uart_readl(port,atmel_port->rtor);
atmel_port->cache.ttgr = atmel_uart_readl(port, ATMEL_US_TTGR);
atmel_port->cache.fmr = atmel_uart_readl(port, ATMEL_US_FMR);
atmel_port->cache.fimr = atmel_uart_readl(port, ATMEL_US_FIMR);
}
```

5.2.6 Platform Implementation

On the platform side, code has been added to deal with the modes specified in Section 5.3.1 (mainly for BSR and ULP0/ULP1 modes).

During the Linux initialization phase, the command line parameter is read to check if the platform code needs to deal with those modes.

The power management platform initialization code checks the device tree for:

- Shutdown Controller (SHDWC)
- Special Function Registers Backup (SFRBU)
- SECURAM
- DRAM controller
- Power Management Controller (PMC)

and memory maps these devices. The memory-mapped regions are used in the suspend/resume procedure (see `at91_pm_suspend_in_sram` function).

During the platform-related PM initialization phase, SECURAM is written with the suspend status and address of the resume function. During the resume process, AT91Bootstrap uses this information to resume Linux as required (see Figure 5-5).

After the initialization phase, the platform is ready for suspend/resume operations. The platform suspend/resume is the last/first phase of suspend/resume (see `platform enter()` state and `platform end()` state in Figure 5-4).

The last suspend phase is executed from the internal SRAM. The corresponding code is copied to SRAM at platform initialization and after each resume operation.

Depending on the suspend mode (BSR or ULP0/ULP1 modes), different levels of power saving are implemented in the last suspend phase.

The last/first phase of the suspend/resume procedure is written in assembly code because this phase will be executed from SRAM as the main memory (DDR) will be in Self-refresh mode while executing power management instructions. The main entry point to the last suspend phase is the `at91_pm_suspend_in_sram()` function. The argument received by this function is an object of type `struct at91_pm_data` (initialized in platform initialization code) that keeps all data necessary in the process of last/first phase of suspend/resume.

```
struct at91_pm_data {
   void __iomem *pmc;
   void __iomem *ramc[2];
   unsigned long uhp_udp_mask;
   unsigned int memctrl;
   unsigned int mode;
   void __iomem *shdwc;
   void __iomem *sfrbu;
   unsigned int standby_mode;
   unsigned int suspend_mode;
};
```

The last phase of the suspend procedure are detailed in the steps below (as seen in the `at91_pm_suspend_in_sram()` function):

- Save registers on stack
- Store the content of the object of type `struct at91_pm_data` on the corresponding registers
- Activate DDR Self-refresh mode
- Switch to the selected power management mode

Depending on the selected power management mode, further settings will be done as follows:

1. Idle mode: CPU switches to WFI.
2. ULP0/ULP1 modes:
   – Master clock settings are saved.

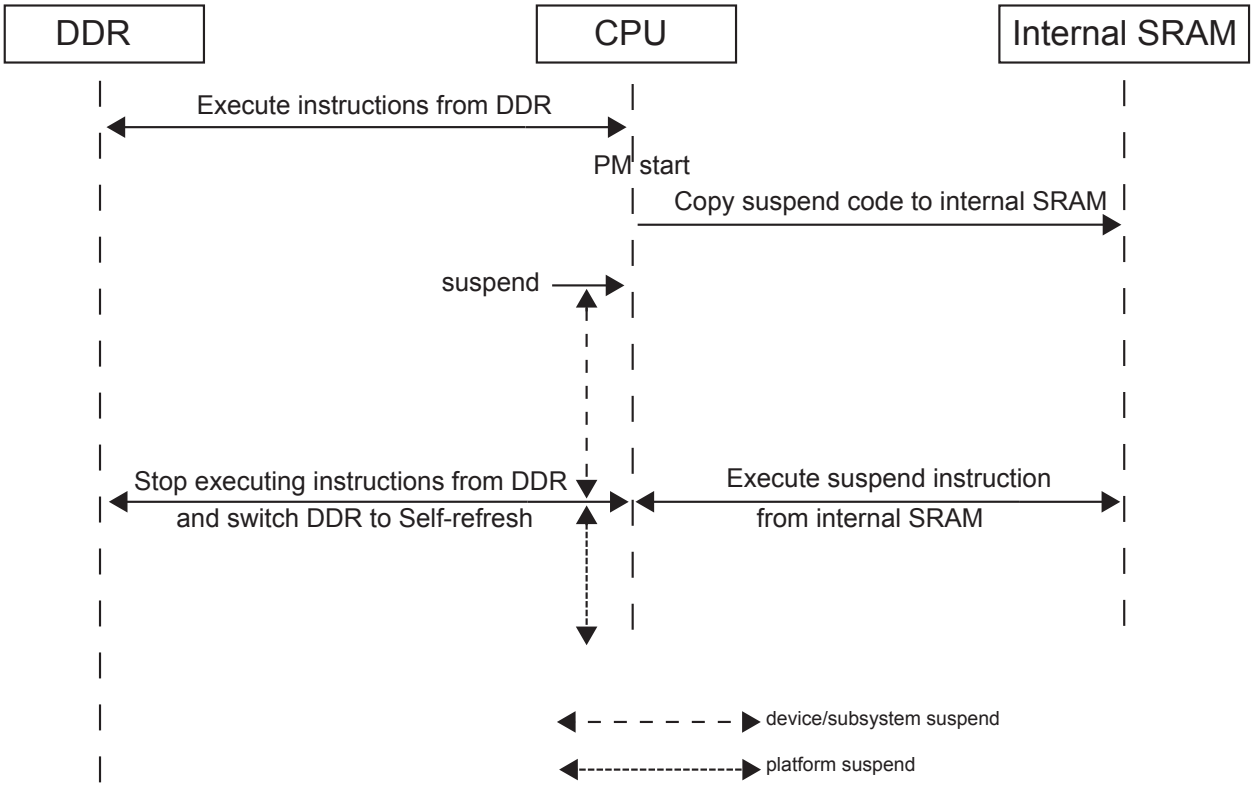
– Master clock source is switched to slow clock.
– PLLA settings are saved, then PLLA is disabled.

2.1. ULP0 mode:
- Crystal oscillator is turned off.
- CPU switches to WFI.

2.2. ULP1 mode:
- Main clock source is switched to 12 MHz RC oscillator.
- Crystal oscillator is disabled.
- Master clock source is switched to main clock.
- Enter ULP1 mode: CKGR_MOR.WAITMODE=1.

3. BSR mode:
   – Shut down the CPU.

Figure 5-7. Last Suspend Phase



In BSR mode, the CPU core is powered off (that is why the CPU and internal SRAM timelines are interrupted after platform suspend in Figure 5-7). For ULP0/ULP1 modes, processor power is kept, so the CPU and internal SRAM timelines continue after the platform suspend code was executed.

The resume phase performs the settings in reverse order compared to the suspend phase. Depending on the suspend mode:

1. Idle mode: the memory Self-refresh mode is deactivated and stack is restored.
2. ULP0/ULP1 modes:
   2.1. ULP0 mode:

- Crystal oscillator is turned on.

2.2. ULP1 mode:

- Crystal oscillator is turned on.
- Master clock source is switched to slow clock.
- Main clock source is switched to crystal oscillator.
- Master clock source is switched to main clock.

2.3. ULP0/ULP1 modes:

- PLLA settings are restored.
- Master clock settings are restored.

3. Backup mode: `cpu_resume` function loads the stack saved by the suspend process and the execution continues from the point where it was before suspend.

6. Measurement Results

This section describes how to perform measurements to evaluate consumption in the different modes. The described measurements were performed at room temperature.

Power consumption measurement is based on the SAMA5D2 Xplained board. On the SAMA5D2 Xplained board, all SAMA5D2 power rails are supplied by a power management IC ACT8945AQJ405 (refer to the ACT8945A datasheet). Figure 2-3 is the PMIC part schematic (refer to the Schematics section in SAMA5D2 Xplained Ultra Evaluation Kit User's Guide).

6.1 Conditions

6.1.1 Board Setup for Measurement

To perform the different measurements on the SAMA5D2 Xplained board, the bare metal or Linux application clock settings were set to:

- 498 MHz for the CPU clock
- 166 MHz for the peripherals/master clock

The board offers several points of measurement, all of them feeding a specific power rail. Figure 6-1 is a top view of the SAMA5D2 Xplained board and shows the jumper to access all these power rails. The board was modified to enable access to two power rails that are not connected to jumpers on the board. See Table 6-1 and Figure 6-2. In addition, only one type of DDR memory was used on this board. Custom bare metal application and Linux OS were used and configured according to the board.

Figure 6-1. Power Jumpers on SAMA5D2 Xplained Board

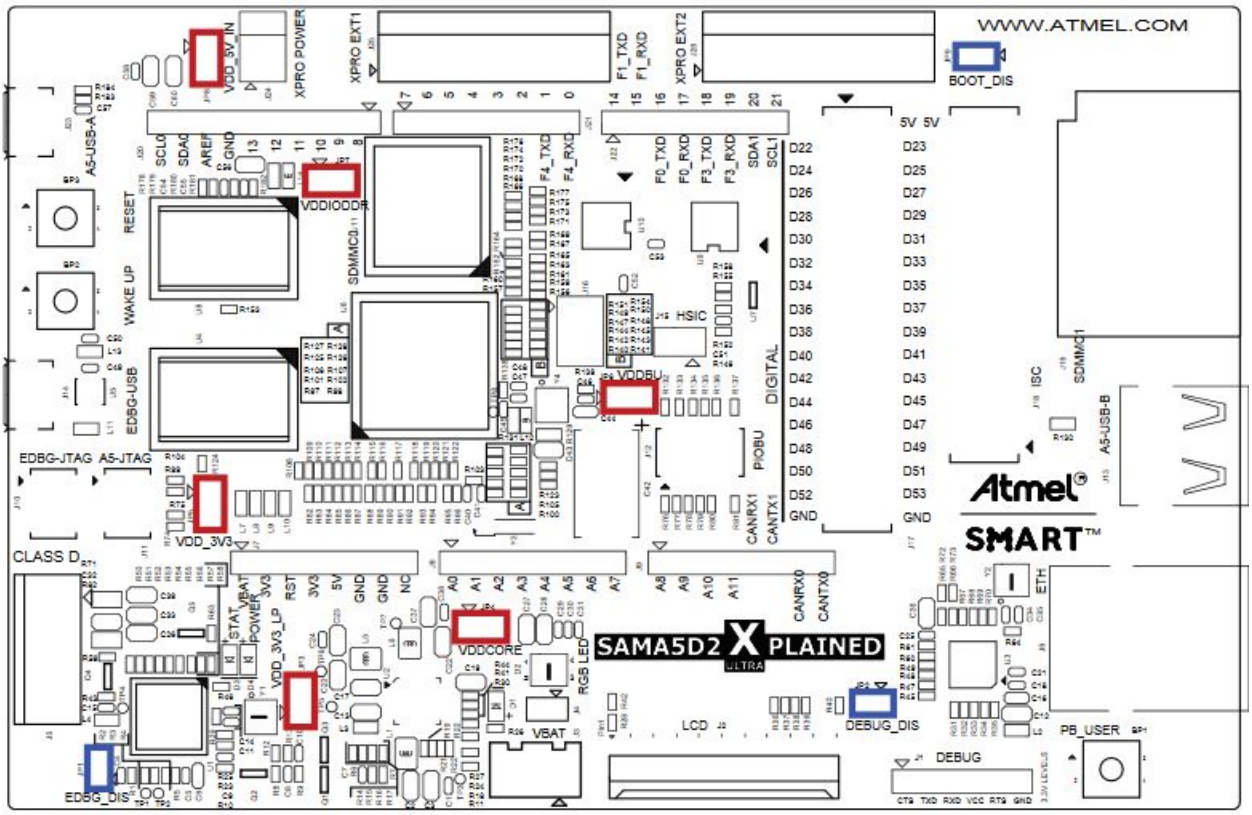


Figure 6-2. VDD_1V2 and VDD_1V35 Power Rail Access

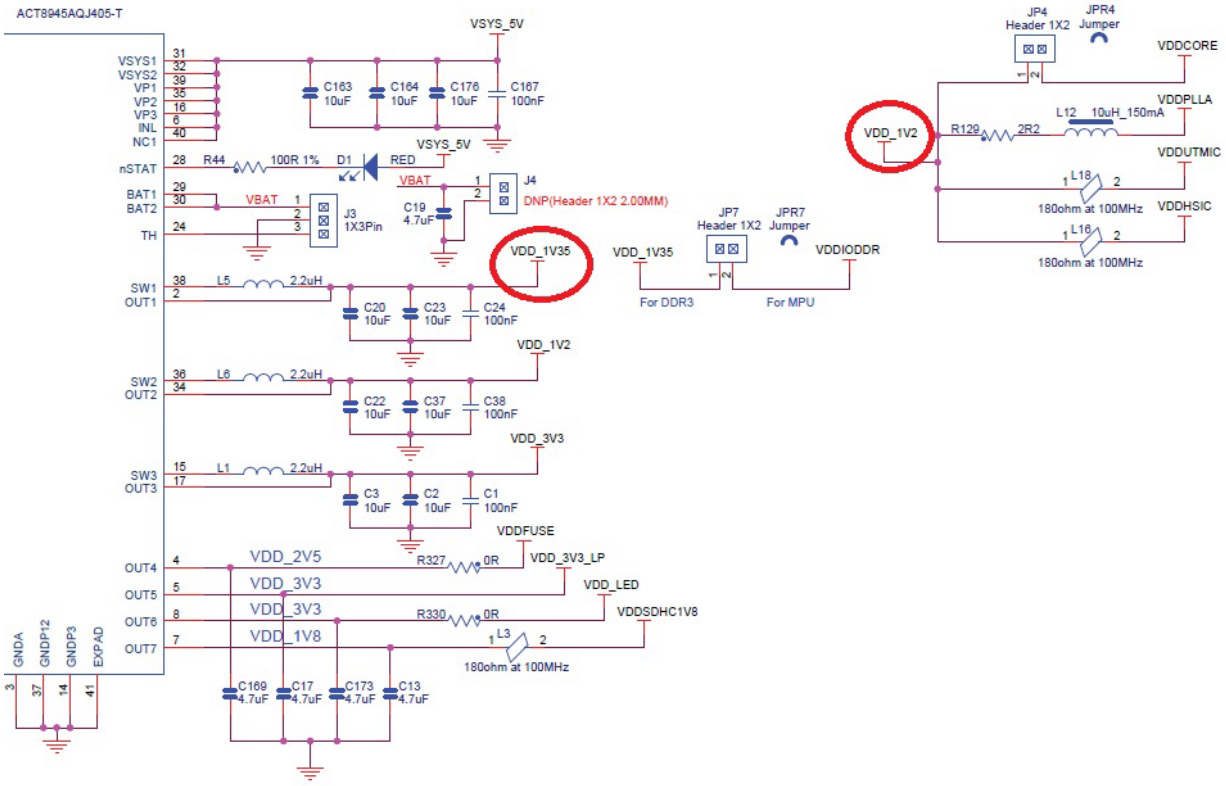


Table 6-1 gives a correspondence between jumpers and power rails, as well as a description of the different power rails connected to these jumpers.

Table 6-1. Power Rail Measurement Access and Feeding

Power Rail	Jumper	Power Rail Feeding
VDDBU	JP6	Backup area
VDDCORE	JP4	VDDCORE rail
VDD_1V2	N/A(1)	VDDCORE, VDDPLLA, VDDUTMIC & VDDHCSIC
VDD_5V_IN	JP8	Main power of the PMIC
VDDIODDR	JP7	VDDIODDR rail
VDD_1V35	N/A(1)	DDR memory
VDD_3V3_LP	JP3	VDDOSC, VDDUTMII, VDDANA & VDDAUDIOPLL
VDD_3V3	JP5	VDDIOP0, VDDIOP1, VDDIOP2 & VDDISC

Note:

1. Before performing those measurements, a SAMA5D2 Xplained board was modified to enable independent access to:
   - The VDDCORE power consumption and the VDD_1V2 power consumption (VDDCORE, VDDPLLA, VDDUTMIC and VDDHCSIC)
   - The DDR memory power consumption powered by VDD_1V35 and the VDDIODDR power rail power consumption (VDD_1V35 was taken directly on the memory)

In addition to jumpers for measurement purpose, the following jumpers must be left open:

- JP1 (Disable EDBG)
- JP2 (Disable Debug)
- JP9 (Disable CS of SPI/QSPI memory) when the application is not stored in an SDMMC or in an eMMC

For more information on the SAMA5D2 Xplained board, refer to the SAMA5D2 Xplained Ultra Evaluation Kit User's Guide (see Reference Documents).

6.1.2 Linux-Specific Code Changes

The measurements were performed with a SAMA5D2 Xplained board while monitoring PB5 pin toggling. The Linux SD card image from the file *linux4sam-poky-sama5d2_xplained-5.7.img.bz2*(1) was patched with the GIT patch given in Appendix A (see Appendix A. Linux Code Patch for Time Measurement).

Note:

1. Available on ftp://www.at91.com/pub/demo/linux4sam_5.7.

6.2 Suspend/Wake-up Time Measurement

This section specifies the suspend/wake-up times for SAMA5D2 Low-Power modes.

6.2.1 Measurement Conditions

To measure suspend times, a pin was toggled before the suspend instruction and after the suspend was finished. The same principle was used to measure wake-up time: the pin was toggled at the instant of resume start and after resume was finished.

In bare metal applications, suspend measurements were started at the beginning of each bare metal sample code in this application note (see sections 4.1.1, 4.1.3, 4.2.1 and 4.2.3) and finished just before entering Low-Power mode. Wake-up time measurements were started immediately after the instruction to enter the Low-Power mode until the end of the sample code.

6.2.2 Measured Values

Tests were done on bare metal and Linux operating systems. Five series of five measurements each were performed, and the board was rebooted every five measurements.

Main test differences:

- Linux OS tests:
  - Applications corresponding to Linux4sam version 5.7 were executing a simple embedded distribution running from a root filesystem in SD Card.
  - Most of the peripheral device states were saved and restored in suspend and resume processes.

- Bare metal OS tests:
  - Only one application (LED blink) was running.
  - Peripheral devices were not treated in suspend and resume processes.

The following table summarizes results.

Table 6-2. Bare Metal and Linux Suspend and Wake-Up Time Measurements

Mode	Operating System	Time to Enter Low-power Mode	Wake-up Time
Idle	Bare metal	<10 µs(1)	<1 µs(1)
	Linux	N/A	NA
ULP0	Bare metal	5.6 ms	4 ms
	Linux	184 ms	246 ms
ULP1	Bare metal	350 µs	15 µs
	Linux	201 ms	238 ms
BSR	Bare metal	1.4 ms	600 µs
	Linux	175 ms	2123 ms(2)

Note:

1. Indicative timings. Actual timings (ns range) cannot be measured by the sample code given in this application note.
2. Starting from AT91Bootstrap version 3.8.12, the wake-up time for BSR is decreased to approx. 800 ms.

Note: To wake up the system from Low-Power modes, the 5V power supply on the SAMA5D2 Xplained board must be kept connected.

In the Linux world, timing is strongly dependent on the save and restore processes of the peripheral states. These results could be significantly improved by removing some of these unnecessary actions, as the major part of the context is maintained in ULP0/ULP1/IDLE modes.

6.3 Consumption Measurement

This section describes results of the SAMA5D2 Xplained board power consumption tests using a bare metal application and a Linux application.

6.3.1 Measured Values

Table 6-3 gives power consumption on each rail when the application is in Running mode.

For Linux, the Linux4SAM 5.7 distribution is started, the Root filesystem is in SD Card and the kernel runs with a few daemons.

Table 6-3. Applications in Running Mode

Power Rail	Application	Consumption	Comment
VDD_1V35	Linux	18 mA	The DDR is powered.
	Bare metal		

.....continued			
Power Rail	**Application**	**Consumption**	**Comment**
VDDIODDR	Linux	8 mA	The VDDIODDR rail is powered so that it can communicate with the DDR.
	Bare metal		
VDDCORE	Linux	65 mA	This difference shows that the Linux scheduler uses Idle mode to reduce its power consumption.
	Bare metal	86 mA	
VDDBU	Linux	1.5 µA	SFRBU_PSWBUCTRL is set to the power backup area with VDDANA instead of VDDBU and saves battery life time.
	Bare metal		
VDD3V3	Linux	1.5 mA	On the Linux side, a real application is running, so there is activity on Ethernet PHY and mass storage.
	Bare metal	220 µA	As this is a bare metal application, there is no activity on the GPIO. Power consumption is at the minimum.
VDD3V3LP	Linux	30 mA	On the Linux side, the peripherals powered by VDDUTMI, VDDAUDIOPLL and VDDANA (UTMI transceiver, audio PLL, analog, etc.) are enabled, whereas they are disabled on the bare metal devices.
	Bare metal	1.7 mA	
VDD5V	Linux	120 mA	The application on the Linux side is much more complex than on the bare metal side.
	Bare metal	98 mA	

Table 6-4 gives the different power consumptions on each rail when the application is in Idle mode.

For Linux, the Idle mode is selected by using the following command line parameter:

```
atmel.pm_modes=standby,ulp0
```

and running the following command:

```
echo standby > /sys/power/state
```

Table 6-4. Applications in Idle Mode

Power Rail	Application	Consumption	Comment
VDD_1V35	Linux	18 mA	The DDR is powered.
	Bare metal		
VDDIODDR	Linux	8 mA	The VDDIODDR rail is powered so that it can communicate with the DDR.
	Bare metal		
VDDCORE	Linux	46 mA	–
	Bare metal	35 mA	–
VDDBU	Linux	1.5 µA	SFRBU_PSWBUCTRL is set to the power backup area with VDDANA instead of VDDBU and saves battery life time.
	Bare metal		

.....continued			
Power Rail	**Application**	**Consumption**	**Comment**
VDD3V3	Linux	1.1 mA	–
	Bare metal	2.3 mA	–
VDD3V3LP	Linux	6.4 mA	–
	Bare metal	1.7 mA	–
VDD5V	Linux	68 mA	The core is powered but not clocked. Peripherals are powered and clocked. On the Linux side, proper management of external components (Standby mode of Ethernet PHY) leads to a reduced total power consumption.
	Bare metal	78 mA	

Table 6-5 gives the different power consumptions on each rail when the application is in ULP0 mode.

For Linux, the ULP0 mode is selected by using the following command line parameter:

```
atmel.pm_modes=standby,ulp0
```

and running the following command:

```
echo mem > /sys/power/state
```

Table 6-5. Applications in ULP0 Mode

Power Rail	Application	Consumption	Comment
VDD_1V35	Linux	11 mA	DDR power is maintained so that the code remains loaded before jumping to ULP0 mode.
	Bare metal		
VDDIODDR	Linux	190 µA	The VDDIODDR rail is maintained powered to be able to restart code execution as soon as the SAMA5D2 device exits ULP0 mode.
	Bare metal		
VDDCORE	Linux	250 µA	The core is powered and clocked at 512 Hz.
	Bare metal		
VDDBU	Linux	1.5 µA	SFRBU_PSWBUCTRL is set to the power backup area with VDDANA instead of VDDBU and saves battery life time.
	Bare metal		
VDD3V3	Linux	1.2 mA	–
	Bare metal		–
VDD3V3LP	Linux	340 µA	–
	Bare metal		–
VDD5V	Linux	39 mA	The core is powered but not clocked. Peripherals are powered and clocked at 512 Hz. On the Linux side, a proper management of external components (Standby mode of Ethernet PHY) leads to a reduced power consumption.
	Bare metal	67 mA	

Table 6-6 gives the different power consumptions on each rail when the application is in ULP1 mode.

For Linux, the ULP1 mode is selected by using the following command line parameter:

```
atmel.pm_modes=standby,ulp1
```

and running the following command:

```
echo mem > /sys/power/state
```

Table 6-6. Application in ULP1 Mode

Power rail	Application	Consumption	Comment
VDD_1V35	Linux	11 mA	DDR power is maintained so that the code remains loaded before jumping to ULP1 mode.
	Bare metal		
VDDIODDR	Linux	190 µA	The VDDIODDR rail is maintained powered to be able to restart code execution as soon as the SAMA5D2 device exits ULP1 mode
	Bare metal		
VDDCORE	Linux	250 µA	The core is powered but not clocked.
	Bare metal		
VDDBU	Linux	1.5 µA	SFRBU_PSWBUCTRL configures the power backup area so that it is fed with VDDANA instead of VDDBU to save battery life time.
	Bare metal		
VDD3V3	Linux	1.2 mA	–
	Bare metal		–
VDD3V3LP	Linux	340 µA	–
	Bare metal		–
VDD5V	Linux	39 mA	Core and peripherals are powered but not clocked. On the Linux side, a proper management of external components (Standby mode of Ethernet PHY) leads to a reduced power consumption.
	Bare metal	67 mA	

Table 6-7 gives the different power consumptions on each rail when the application is in BSR mode.

For Linux, the BSR mode is selected by using the following command-line parameter:

```
atmel.pm_modes=standby,backup
```

and running the following command:

```
echo mem > /sys/power/state
```

Table 6-7. Applications in Backup Self-Refresh (BSR) Mode

Power Rail	Application	Consumption	Comment
VDD_1V35	Linux	11 mA	The DDR power is maintained to be able to keep the code loaded in DDR prior to jump in BSR mode.
	Bare metal		

.....continued

Power Rail	Application	Consumption	Comment
VDDIODDR	Linux	190 µA	The VDDIODDR rail is maintained powered to be able to restart as soon as the main power is applied to the SAMA5D2.
	Bare metal		
VDD5V	Linux	52 mA	Only the backup area of the SAMA5D2 device is powered, the DDR is in BSR, the PMIC is powered.
	Bare metal		
VDDBU	Linux	3.8 µA	Only VDDBU is powering the backup area.
	Bare metal		
VDDCORE, VDD3V3, VDD3V3LP	Linux	0	These power inputs are not supplied in Backup and BSR modes.
	Bare metal		

Whatever the application, bare metal or Linux, only the VDDBU and the VDD_1V35 are powered (VDDIODDR is taken from the VDD_1V35), so the power consumption is exactly the same for each application.

7. Conclusion

In any modern system, power consumption aspects cannot be ignored. One way to optimize an application power performance is to enable periods of time when the system can be set to a low-power mode. The SAMA5D2 device features several low-power modes with specific power consumption levels and associated suspend/wake-up times to serve this need.

This application note demonstrates these low-power modes for a specific configuration (SAMA5D2 Xplained board with DDR3L memory). Note that significantly different results can be obtained with different memory types.

The following table gives an estimation of the power consumption for two memory types and three low-power modes. The values given are at 85°C, as provided in the memory data sheets. Values at 25°C are expected to be 3 times to 5 times lower.

Table 7-1. Power Consumption Estimations

SAMA5D2 Low-power Modes(1)		Memory(3)		Total(2)
Mode	SAMA5D2	LPDDR2	DDR3L	
ULP0 mode	3.1 mW	3 mW	-	6.1 mW
		-	16 mW	19.1 mW
ULP1 mode	2.9 mW	3 mW	-	5.9 mW
		-	16 mW	18.9 mW
BSR mode	0	3 mW	-	3 mW
		-	16 mW	16 mW

Note:

1. All values are taken from the SAMA5D2 Series data sheet at 85°C.
2. The total power consumption includes DDR power + VDDCORE. The real total power consumption depends on the efficiency of the power supply.
3. It is assumed that the self-refresh power consumption at 85°C is approximately 16 mW for a DDR3L memory and 3 mW for a LPDDR2 memory. These are the data sheet values available at the time this application note is written.

This table shows mainly that:

- for a system equipped with a DDR3L memory, very little benefit is obtained in terms of power consumption when using BSR mode instead of ULP0/1,
- for a system equipped with an LPDDR2 memory, about 50% of power consumption is saved by using BSR mode.

Finally, this application note shows significant differences for suspend and wake-up times in ULP1 mode between a complex operating system such as Linux (that saves and restores the full context, which is more than needed) and a simple bare metal application. Thus, specific improvements are needed for a complex OS to leverage the full ULP1 performance.

8. Appendix A. Linux Code Patch for Time Measurement

This is the patch to be applied in addition to Linux4SAM 5.7 in order to measure suspend and wake-up times. The process consists basically in toggling a GPIO.

```
diff --git a/arch/arm/boot/dts/at91-sama5d2_xplained_common.dtsi b/arch/arm/boot/dts/at91-
sama5d2_xplained_common.dtsi
index a9bf487feb21..85b909c9b875 100644
--- a/arch/arm/boot/dts/at91-sama5d2_xplained_common.dtsi
+++ b/arch/arm/boot/dts/at91-sama5d2_xplained_common.dtsi
@@ -734,7 +734,7 @@
         compatible = "gpio-leds";
         pinctrl-names = "default";
         pinctrl-0 = <&pinctrl_led_gpio_default>;
-        status = "okay"; /* conflict with pwm0 */
+        status = "disabled"; /* conflict with pwm0 */

         red {
             label = "red";
diff --git a/arch/arm/mach-at91/pm.c b/arch/arm/mach-at91/pm.c
index e756bc71ffce..aa0d8e65c17e 100644
--- a/arch/arm/mach-at91/pm.c
+++ b/arch/arm/mach-at91/pm.c
@@ -489,6 +489,12 @@ static void __init at91_pm_backup_init(void)
     struct device_node *np;
     struct platform_device *pdev = NULL;

+    np = of_find_compatible_node(NULL, NULL, "atmel,sama5d2-pinctrl");
+    if (!np)
+        return;
+    pm_data.gpioc = of_iomap(np, 0);
+    of_node_put(np);
+
     if ((pm_data.standby_mode != AT91_PM_BACKUP) &&
         (pm_data.suspend_mode != AT91_PM_BACKUP))
         return;
diff --git a/arch/arm/mach-at91/pm.h b/arch/arm/mach-at91/pm.h
index f39679b39d5c..9ef80de101d2 100644
--- a/arch/arm/mach-at91/pm.h
+++ b/arch/arm/mach-at91/pm.h
@@ -35,6 +35,7 @@ struct at91_pm_data {
     unsigned int mode;
     void __iomem *shdwc;
     void __iomem *sfrbu;
+    void __iomem *gpioc;
     unsigned int standby_mode;
     unsigned int suspend_mode;
 };
diff --git a/arch/arm/mach-at91/pm_data-offsets.c b/arch/arm/mach-at91/pm_data-offsets.c
index c0a73e62b725..dc98d3be399b 100644
--- a/arch/arm/mach-at91/pm_data-offsets.c
+++ b/arch/arm/mach-at91/pm_data-offsets.c
@@ -11,6 +11,7 @@ int main(void)
     DEFINE(PM_DATA_MODE,        offsetof(struct at91_pm_data, mode));
     DEFINE(PM_DATA_SHDWC,        offsetof(struct at91_pm_data, shdwc));
     DEFINE(PM_DATA_SFRBU,        offsetof(struct at91_pm_data, sfrbu));
+    DEFINE(PM_DATA_GPIOC,        offsetof(struct at91_pm_data, gpioc));

     return 0;
 }
diff --git a/arch/arm/mach-at91/pm_suspend.S b/arch/arm/mach-at91/pm_suspend.S
index 0f639102f4ef..81d1da65ab0e 100644
--- a/arch/arm/mach-at91/pm_suspend.S
+++ b/arch/arm/mach-at91/pm_suspend.S
@@ -18,10 +18,32 @@

 #define    SRAMC_SELF_FRESH_ACTIVE        0x01
 #define    SRAMC_SELF_FRESH_EXIT        0x00
+#define ATMEL_PIO_PB5_SODR        0x50
+#define ATMEL_PIO_PB5_CODR        0x54
+

 pmc    .req    r0
```

```
 tmp1    .req    r4
 tmp2    .req    r5
+gpio    .req    r6
+
+/*
+ * Turn off green led
+ */
+    .macro turn_off_led
+    ldr    gpio, .gpioc
+    mov    tmp2, #32
+    str    tmp2, [gpio, #ATMEL_PIO_PB5_SODR]
+    .endm
+
+/*
+ * Turn on green led
+ */
+    .macro turn_on_led
+    ldr    gpio, .gpioc
+    mov    tmp2, #32
+    str    tmp2, [gpio, #ATMEL_PIO_PB5_CODR]
+    .endm

 /*
  * Wait until master clock is ready (after switching master clock source)
@@ -68,9 +90,14 @@ tmp2    .req    r5
     mov    tmp1, #AT91_PMC_PCK
     str    tmp1, [pmc, #AT91_PMC_SCDR]

+    turn_off_led
+
     dsb

     wfi        @ Wait For Interrupt
+
+    turn_on_led
+
 #else
     mcr    p15, 0, tmp1, c7, c0, 4
 #endif
@@ -116,6 +143,11 @@ ENTRY(at91_pm_suspend_in_sram)
     cmp    tmp1, #0
     ldrne    tmp2, [tmp1, #0x10]

+    ldr    tmp1, [r0, #PM_DATA_GPIOC]
+    str    tmp1, .gpioc
+    cmp    tmp1, #0
+    ldrne    tmp2, [tmp1, #0x12]
+
     /* Active the self-refresh mode */
     mov    r0, #SRAMC_SELF_FRESH_ACTIVE
     bl    at91_sramc_self_refresh
@@ -283,6 +315,9 @@ ENTRY(at91_backup_mode)
     ldr    r0, .shdwc
     mov    tmp1, #0xA5000000
     add    tmp1, tmp1, #0x1
+
+    turn_off_led
+
     str    tmp1, [r0, #0]
 ENDPROC(at91_backup_mode)

@@ -343,8 +378,13 @@ ENDPROC(at91_backup_mode)
     orr    tmp1, tmp1, #AT91_PMC_WAITMODE
     bic    tmp1, tmp1, #AT91_PMC_KEY_MASK
     orr    tmp1, tmp1, #AT91_PMC_KEY
+
+    turn_off_led
+
     str    tmp1, [pmc, #AT91_CKGR_MOR]

+    turn_on_led
+
     wait_mckrdy

     /* Enable the crystal oscillator */
@@ -451,6 +491,8 @@ ENDPROC(at91_ulp_mode)
```

```
      .word 0
 .sfr:
      .word 0
+.gpioc:
+     .word 0
 .memtype:
      .word 0
 .pm_mode:
diff --git a/kernel/power/main.c b/kernel/power/main.c
index 281a697fd458..544dc29a085c 100644
--- a/kernel/power/main.c
+++ b/kernel/power/main.c
@@ -15,6 +15,7 @@
 #include <linux/workqueue.h>
 #include <linux/debugfs.h>
 #include <linux/seq_file.h>
+#include <linux/gpio.h>

 #include "power.h"

@@ -358,6 +359,8 @@ static ssize_t state_store(struct kobject *kobj, struct kobj_attribute
*attr,
      suspend_state_t state;
      int error;

+     gpio_direction_output(37, 0);
+
      error = pm_autosleep_lock();
      if (error)
          return error;
@@ -377,6 +380,9 @@ static ssize_t state_store(struct kobject *kobj, struct kobj_attribute
*attr,

   out:
      pm_autosleep_unlock();
+
+     gpio_direction_output(37, 1);
+
      return error ? error : n;
 }

--
2.7.4
```

9. Revision History

9.1 Rev. A - 12/2018

First issue.

The Microchip Web Site

Microchip provides online support via our web site at http://www.microchip.com/. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQ), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Customer Change Notification Service

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip web site at http://www.microchip.com/. Under "Support", click on "Customer Change Notification" and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or Field Application Engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: http://www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.

- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable.”

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip’s code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Trademarks

ISBN: 978-1-5224-3947-9

Quality Management System Certified by DNV

ISO/TS 16949

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: http://www.microchip.com/support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4450-2828 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-67-3636 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-72884388 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820