AN2896

SAMA5D2 低功耗模式实现

范围

SAMA5D2 系列是基于 Arm® Cortex®-A5 处理器的高性能、超低功耗 MPU。

本应用笔记通过提供 Linux®和裸机软件示例以及硬件应用原理图来说明如何进入和退出 SAMA5D2 低功耗模式。本文档旨在帮助用户了解 SAMA5D2 的低功耗性能，并设计节能应用。

本应用笔记是 SAMA5D2 Series 数据手册的补充资料，应与以下参考文档结合使用。

参考文档

文档类型	标题	文档编号	下载
数据手册	SAMA5D2 Series	DS60001476	http://www.microchip.com
用户指南	SAMA5D2 Xplained Ultra 评估工具包	DS50002691B_CN	http://www.microchip.com
应用笔记	SAMA5D2 Discrete Power Supply Solution	AN44059	http://www.microchip.com
数据手册	ACT8945A	-	http://www.active-semi.com

目录

1. SAMA5D2 低功耗模式概述

1.1 SAMA5D2 低功耗模式

SAMA5D2 器件具有五种低功耗模式：备用、备用自刷新（Backup Self-Refresh，BSR）、超低功耗 0（Ultra Low-Power 0，ULP0）、超低功耗 1（Ultra Low-power 1，ULP1）和空闲。

这些模式提供广泛的功耗性能（从几微安到几毫安）和唤醒时间（从几微秒到几百毫秒），以满足不同的应用需求。以下各节详细介绍了每种低功耗模式下的器件操作。

1.1.1 备用模式

在应用中，备用模式对应于处理器在一段较长的时间内处于掉电状态。在此模式下，处理器及其外设和存储器均未上电。只有器件的备用区域保持上电和运行状态，以便让实时时钟（Real Time Clock，RTC）、备用寄存器、备用 SRAM 和安全模块保持运行状态。安全模块通过专用监视器保护应用免受篡改引脚 PIOBU0-7 的篡改，在 SAMA5D23 和 SAMA5D28 中，可防止频率（f）、温度（T）和电压（V）超出工作范围。所有篡改检测均通过 RTC 添加时间戳。

在进入或退出备用模式时，最好使用处理器的关断控制器（SHDWC）来帮助管理应用的电源单元。

该外设控制电路板上后续会使用的 SHDN 引脚，以使能或禁止电源通道。通常，在进入备用模式时，软件会将 SHDN 置为低电平。当发生唤醒事件时，SHDWC 会自动将 SHDN 切换回高电平。可以通过 RTC 事件、WKUP0、WKUP2 至 WKUP9 引脚事件、低功耗异步接收器（RXLP）事件或模拟比较器控制器（Analog Comparator Controller，ACC）事件退出备用模式。

当处于备用模式时，SAMA5D2 在 VDDBU（备用区域电源）下的电流消耗降至几微安，因此可以通过超级电容或锂锰纽扣电池为 VDDBU 供电。为了进一步降低存储元件的电流消耗，SAMA5D2 提供了一个电源开关，以便从 VDDBU 或 VDDANA（模拟轨电源）中选择备用区域的电源。当存在 VDDANA 时，可以通过设置 SFRBU_PSWBUCTRL 中的 SCTRL 和 SSWCTRL 位从 VDDANA 为备用区域供电。下表给出了根据应用备用电池的占空比使用情况估算电池使用寿命的示例。

表 1-1. VDDBU 上常见储能元件的典型使用寿命估算

储能元件	电容	备用模式电流消耗（25°C）		电池自放电电流（10 年）	应用处于备用模式的时间百分比	估算的平均电流消耗(2)	使用寿命	
		备用模式	运行模式				小时	年
CR2032 锂锰电池	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
充电电压为 3.3V 的 0.2F 超级电容	(3.3V-1.7V) * 0.2F = 0.32C	4.5 µA	1.5 µA	N/A	N/A	4.5 µA	20(3)	–

注：

1. 这是仅基于电池容量计算的理论使用寿命。实际使用寿命会受到电池老化效应的限制。
2. 平均电流消耗（µA）= (4.5 × d + 1.5 × (1-d)) + 0.2，其中 d 是应用程序处于备用模式的时间百分比。
3. 在使用超级电容的情况下，假设该元件在两个备用周期之间已完全再充电。这里的 20 小时是指使用此元件时，SAMA5D2 可保持处于备用模式的时间。

1.1.2 备用自刷新（BSR）模式

此模式是备用模式的扩展。在此模式下，应用程序上下文保存在以自刷新模式工作的外部 DDR 存储器中，方便更加快速地重新启动应用。

在此模式下，必须保持供应 VDDBU 和 VDDIODDR 电源输入，以及 DDR 元件的电源输入。

BSR 的系统功耗主要是 DDR 的功耗，因此 DDR 类型的选择至关重要。同理，选择在 BSR 模式下保持供应 DDR 电源的稳压器时，应优化低电流时的效率（见结论）。

1.1.3 ULP0、ULP1 和空闲模式

在这三种低功耗模式下，所有 SAMA5D2 电源均处于其工作范围内。这三种模式通过降低处理器和/或其外设的频率或者停止时钟信号的方式来实现节能目的。

下面分别介绍了这三种模式。为了便于理解，给出了大致的唤醒时间。如需了解确切时间，请参见 SAMA5D2 Series 数据手册的“Electrical Characteristics”部分。

- 在空闲模式下，只有处理器时钟停止工作，所有外设仍保持工作状态。退出此模式时，处理器全速运行。通常，进入和退出此模式需要几个处理器时钟周期。在 Linux®环境中，与之对应的是“暂停到空闲”。
- 在 ULP0 模式下，处理器停止工作，其外设以极低频率（几 kHz 到几 MHz）工作。从此模式唤醒时，处理器以此极低频率重新启动。降低频率可以优化功耗，但唤醒时间会因此而延长。在此模式下，处理器处于等待中断（Wait-For-Interrupt，WFI）状态，因此任何中断源都可以触发恢复正常操作。
- 在 ULP1 模式下，处理器时钟和外设时钟均停止工作。在进入此模式并停止时钟之前，每个时钟源都会切换到以 12 MHz 典型值运行的主 RC 振荡器。之后进入 ULP1 模式时，电源管理控制器（Power Management Controller，PMC）会停止该振荡器。当发生唤醒事件时，PMC 会自动重启该振荡器，以将器件时钟恢复至 12 MHz。与 ULP0 不同的是，只有唤醒引脚、USB 恢复（见 SAMA5D2 Series 数据手册）等少数事件可以将器件从 ULP0 模式唤醒。在此模式下，VDDCORE（内核电源）下的电流消耗会降至非常低的水平（室温下通常小于 0.5 mW），而且唤醒时间短至几微秒。

下表总结了每种模式下内核、外设和 DDR 的供电和时钟驱动方式。

表 1-2. 内核和外设时钟与模式

模式	处理器内核	外设/内部 SRAM 存储器
空闲	无时钟驱动（WFI）	时钟驱动
ULP0	无时钟驱动（WFI）	低频时钟驱动
ULP1	无时钟驱动（WFE）	无时钟驱动

2. SAMA5D2 低功耗模式的电源实现

2.1 硬件实现简介

从电源角度来看，必须分两种情况来管理 SAMA5D2 低功耗模式：

- 在 ULP0、ULP1 和空闲模式下，器件的所有电源输入均在其指定的工作范围内。随着功耗的降低，电源电路可以切换到节能模式。
- 在 BSR 模式下，器件的所有电源输入均关闭，但 VDDBU、VDDIODDR 和存储器的电源输入必须保持工作状态。要管理这种情形，应用可以将 I^2C 命令发送到 PMIC 或使用 PIOBU（在备用区域中供电和进行时钟驱动）来通知电源进入特定的供电情形。

下图给出了进入与退出备用模式和 BSR 模式时的简化时序图。有关上电和掉电序列的完整信息，请参见 SAMA5D2 Series 数据手册。

图 2-1. 进入和退出备用模式的示例

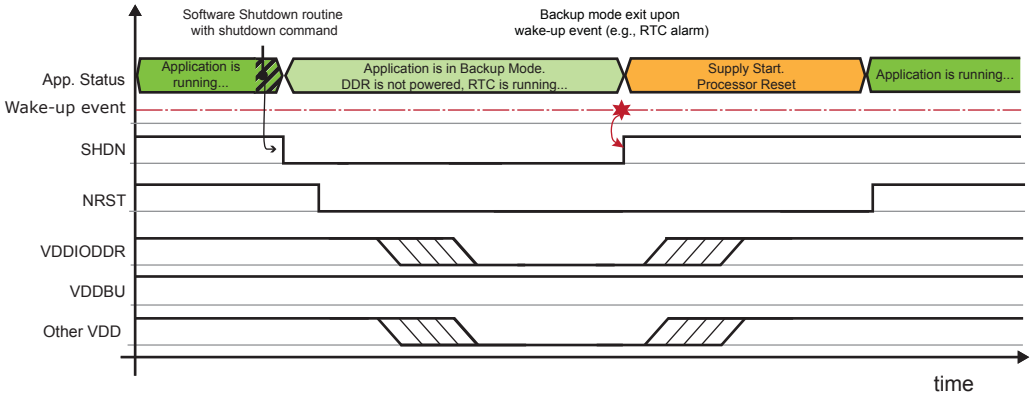
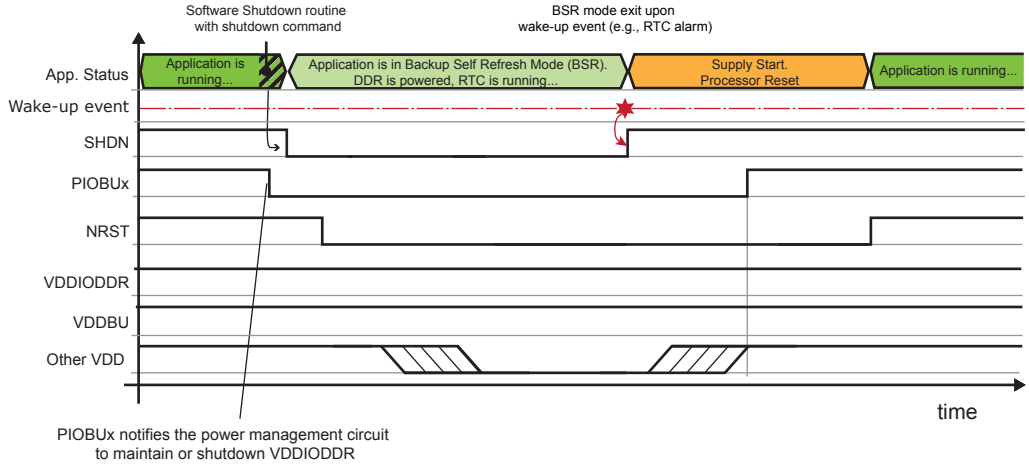


图 2-2. 使用 PIOBU 进入和退出 BSR 模式的示例



以下各节给出了使用 PMIC 器件或分立元件的硬件实现示例。

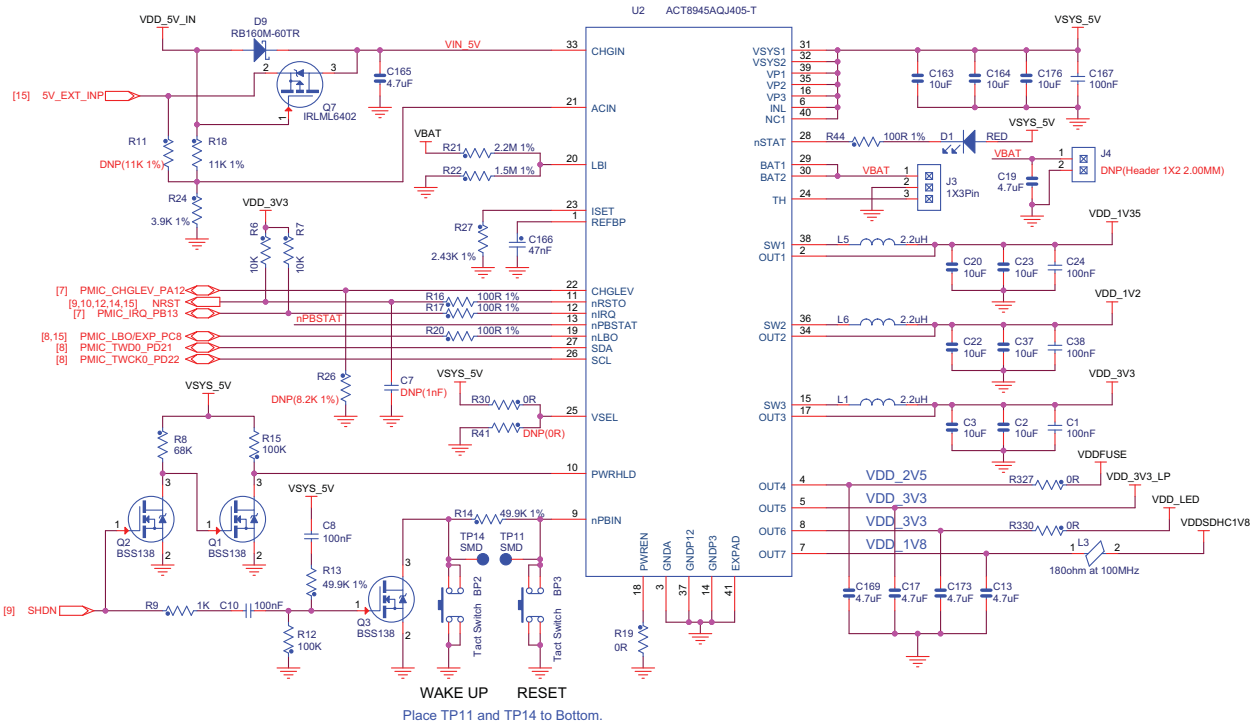
两个示例中提供的说明用于设置备用自刷新模式。

2.2 使用 PMIC 的 BSR 模式的硬件实现

下图所示为 SAMA5D2 Xplained 板电源。电源管理 IC ACT8945A 为处理器及其外部 DDR3L-SDRAM 存储器供电。在进入 BSR 模式之前，应用通过 I²C 接口向 PMIC 发送 BSR 模式下所需的通道配置。

通常，软件请求 PMIC 保持供应 VOUT1（VDD_1V35，DDR3L 存储器的电源）并关闭其他通道。当 SHDN 引脚被处理器清零时，执行该配置。

图 2-3. PMIC 原理图

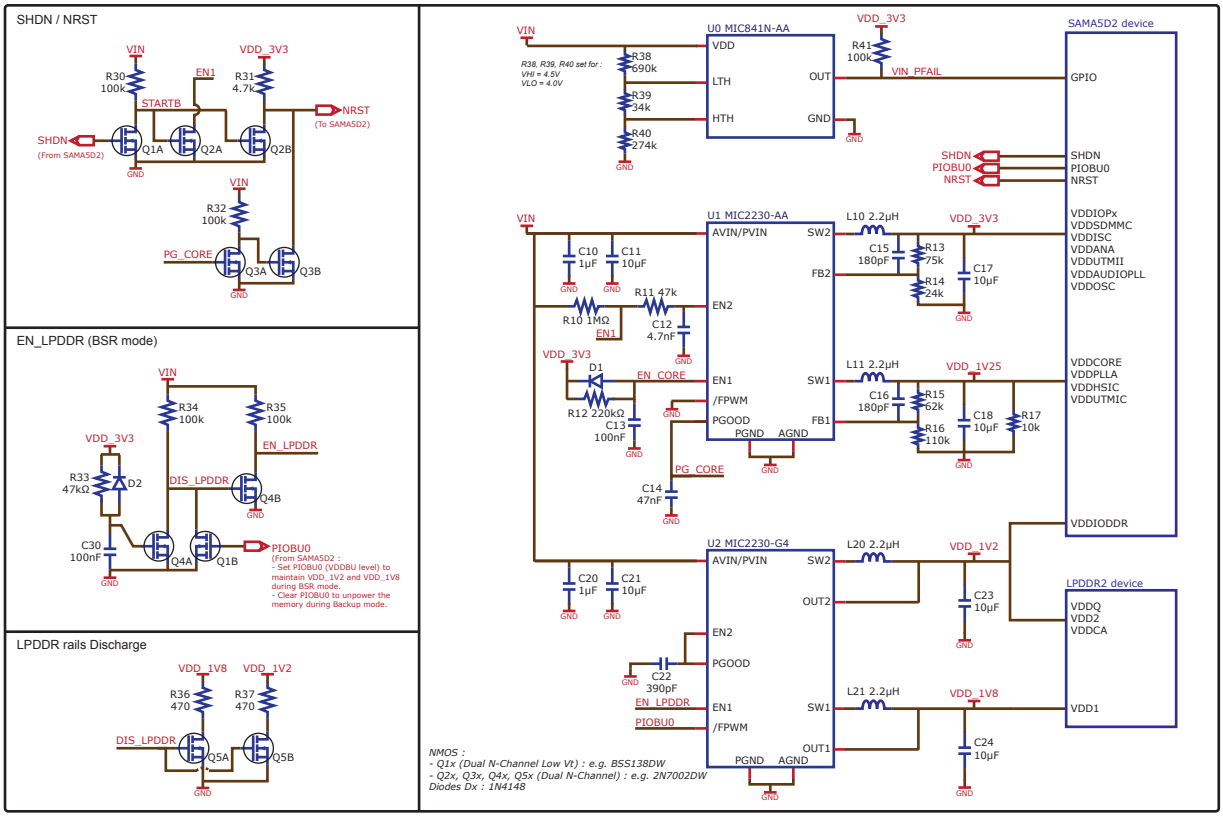


2.3 使用分立元件的 BSR 模式的硬件实现

下图提供了针对配有 LPDDR2 存储器的 SAMA5D2 系统进行优化的分立元件电源原理图。

输入电压 VIN 的范围为 3.0V 至 5.5V。该原理图使用了应用笔记“SAMA5D2 Discrete Power Supply Solution”中描述的大多数原理（定序、复位置为有效等）（见 参考文档）。关断控制器（SHDWC）的 SHDN 引脚用于启动和停止电源通道，PIOBU0 引脚专用于 BSR 模式管理。当 PIOBU0 设置为高电平时，无论 SHDN 上的电平如何，EN_LPDDR 信号都保持为高电平。因此，应保持供应 VDD_1V2 和 VDD_1V8 电源轨（分别连接到 LPDDR2 器件和连接到 SAMA5D2 的 VDDIODDR）。由于 LPDDR2 器件具有严格的掉电要求，因此添加 U0 来检测早期电源输入损耗，Q5A/Q5B 有助于使 VDD_1V2 和 VDD1V8 立即放电。

图 2-4. 分立元件原理图


SHDN / NRST
VIN
VDD_3V3
R30 100k
EN1
R31 4.7k
STARTB
NRST
(To SAMA5D2)
SHDN
(From SAMA5D2)
Q1A
Q2A
Q2B
GND
VIN
R32 100k
PG_CORE
Q3A
Q3B
EN_LPDDR (BSR mode)
VIN
R34 100k
R35 100k
VDD_3V3
EN_LPDDR
R33 47kΩ
D2
DIS_LPDDR
Q4B
C30 100nF
Q4A
Q1B
PIOBU0
(From SAMA5D2 :
- Set PIOBU0 (VDDBU level) to maintain VDD_1V2 and VDD_1V8 during BSR mode.
- Clear PIOBU0 to unpower the memory during Backup mode.
LPDDR rails Discharge
VDD_1V8
VDD_1V2
R36 470
R37 470
DIS_LPDDR
Q5A
Q5B
U0 MIC841N-AA
VDD
LTH
HTH
GND
OUT
R38 690k
R39 34k
R40 274k
R41 100k
VIN_PFAIL
SAMA5D2 device
GPIO
SHDN
PIOBU0
NRST
U1 MIC2230-AA
AVIN/PVIN
SW2
FB2
EN2
EN1
/FPWM
PGOOD
PGND
AGND
SW1
FB1
L10 2.2µH
L11 2.2µH
C10 1µF
C11 10µF
C12 4.7nF
R10 1MΩ
R11 47k
C15 180pF
R13 75k
R14 24k
C17 10µF
D1
R12 220kΩ
C13 100nF
EN_CORE
PG_CORE
C14 47nF
C16 180pF
R15 62k
R16 110k
C18 10µF
R17 10k
VDD_1V25
VDDIOPx
VDDSDMMC
VDDISC
VDDANA
VDDUTMII
VDDAUDIOPLL
VDDOSC
VDDCORE
VDDPLLA
VDDHSIC
VDDUTMIC
VDDIODDR
U2 MIC2230-G4
OUT2
OUT1
L20 2.2µH
L21 2.2µH
C20 1µF
C21 10µF
C22 390pF
C23 10µF
C24 10µF
VDD_1V2
VDD_1V8
LPDDR2 device
VDDQ
VDD2
VDDCA
VDD1
R38, R39, R40 set for :
VHI = 4.5V
VLO = 4.0V
NMOS :
- Q1x (Dual N-Channel Low Vt) : e.g. BSS138DW
- Q2x, Q3x, Q4x, Q5x (Dual N-Channel) : e.g. 2N7002DW
Diodes Dx : 1N4148

3. 关于将系统设置为低功耗模式的一般建议

由于 SAMA5D2 在某些模式下的功耗可低至几微安，因此确保系统级器件外部没有泄漏电流丢失至关重要。在进入其中一种低功耗模式之前，以下一般建议适用：

- 验证在低功耗模式下将保持供电的外部元件的状态。可能需要将其中一些设置为待机模式或低功耗模式，甚至关闭。在备用或 BSR 模式下，可以方便地为此模式下未使用的元件断开电源。应根据具体情况对此进行评估。
- 验证器件每个 IO 的状态。需要特别注意的是，连接到 SAMA5D2 的任何元件（包括上拉和下拉电阻）都可能形成泄漏路径。此外，即使将 I/O 配置为输入，强制此线路采用活动时钟（例如，来自串行链路上的主器件的串行时钟，或来自以太网 PHY 的时钟），VDDCORE 域也难免会产生一定的功耗。
- 为避免 VDDBU 电源域中出现泄漏，除非经过仔细验证，否则不得将属于 VDDBU 电源域的 MPU 的 I/O（WKUP、PIOBUx、RXD、COMPP、COMPN 和 SHDN）直接连接到外部元件（例如 PMIC）的 I/O。最好通过外部缓冲器（例如简单的 NMOS 晶体管）隔离这些线路。在直接连接的情况下，可以通过这些 I/O 的 ESD 保护二极管在 VDDBU 电源域与主电源域之间形成泄漏路径。请参见以下示例。

图 3-1. SHDN 引脚连接

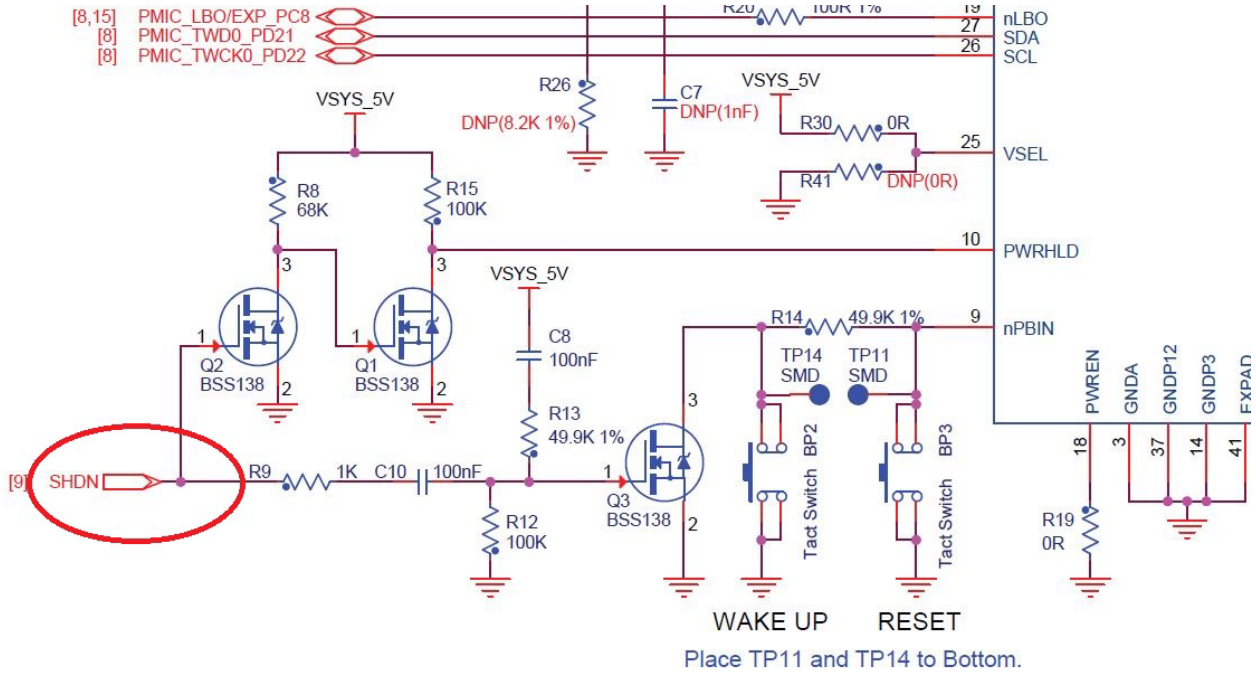
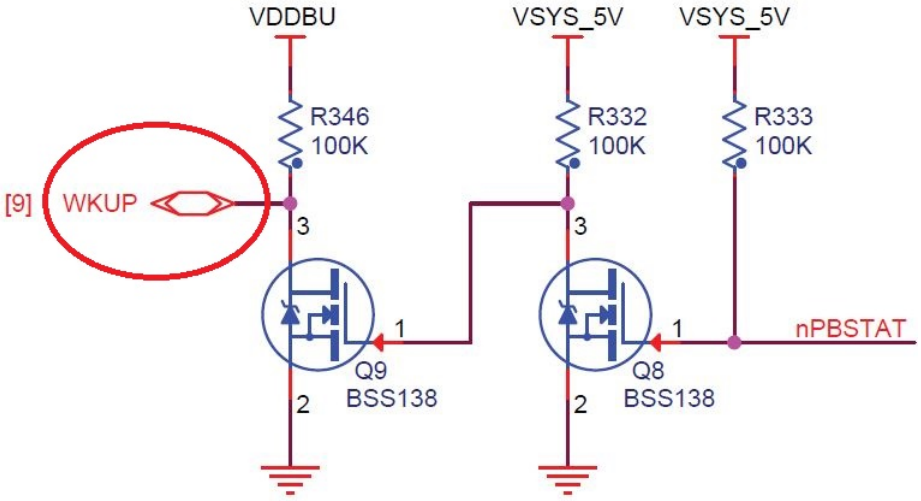


图 3-2. WKUP 引脚连接



- 在进入备用模式或 BSR 模式（电源因此而关闭）之前，最好降低器件工作速度，以免因达到最大执行速度而导致电源崩溃。为此，建议将主时钟（MCK）源切换为慢速时钟。
- 如果电源电路具有特定的低功耗模式，则进入此模式可能会比较合适。
- 在空闲模式下，若要减少因使用 USB 而产生的额外功耗(1)，必须通过将 SFR_OHCIICR.SUSPEND_x 位置 1 强制使其他端口进入暂停模式。

注：

1. USB 设备：在 UDPHS_CTRL 寄存器中将 DETACH 位置 1，将 PULLD_DIS 位清零。这两个位的任何其他组合均可能导致出现额外的功耗。
   USB 主机：在 USB 暂停程序结束时，通过将 SFR_OHCIICR 寄存器中的 SUSPEND_A、SUSPEND_B 和 SUSPEND_C 位置 1，强制所有 USB 主机端口暂停工作。

4. 裸机软件实现

以下裸机示例提供了进入每个 SAMA5D2 低功耗模式的详细步骤。本章中使用的项目基于 SAMA5D2 软件包（IAR7.40）。

一些降低功耗的方法包括：

- 在运行时调整系统时钟
- 根据器件的使用情况适当将其置于休眠模式
- 将存储器置于自刷新模式
- 将系统置于备用模式

本章将详细介绍上述方法，其中低功耗模式的电源管理分为三个部分：

- 备用模式
- 超低功耗模式
- 空闲模式

4.1 备用和备用自刷新模式

4.1.1 如何进入备用模式

通过关闭除 VDDBU 之外的所有电源轨进入备用模式，仅备用区域处于运行状态。关闭内核、外设和内部存储器，在 SAMA5D2 Xplained 板的特定情况下，所有外部元件均未上电。

1. 在关断控制器中配置唤醒源。
2. 通过将主时钟（MCK）切换为慢速时钟来降低器件的工作速度。
3. 通过将 SHDN 引脚置为有效进入备用模式，通知 PMIC 或分立元件可以关闭除 VDDBU 之外的所有电源。

示例代码：

```
//切换回备用区域中的 VDDBU 电源而非 VDDANA。
SFRBU->SFRBU_PSWBUCTRL = SFRBU_PSWBUCTRL_WPKEY_PASSWD;
/*配置唤醒*/
shdwc_configure_wakeup();
/*清除状态*/
(void)shdwc_get_status();
/* PCK = MCK = 32 kHz */
/*选择慢速时钟作为 PCK 和 MCK 的输入时钟*/
PMC->PMC_MCKR = (PMC->PMC_MCKR & ~PMC_MCKR_
/*进入备用模式*/
shdwc_do_shutdown();
```

4.1.2 如何退出备用模式

通过以下任一事件触发退出备用模式：

- WKUP0
- WKUP1（安全模块事件）
- WKUP2 至 WKUP9 引脚（PIOBU0 至 PIOBU7，电平跳变，可配置去抖动）
- 在低功耗 UART 接收器（RXLP）上接收到字符
- 模拟比较
- RTC 报警

当检测到唤醒事件时，关断控制器会自动将 SHDN 引脚驱动为高电平。在 SAMA5D2 Xplained 板上，这使得 PMIC 可以接通所有 SAMA5D2 电源。

一旦 SAMA5D2 上电且 NRST 引脚被释放，就会执行 ROM 代码并将自举程序从存储介质（eMMC 和 SD 卡等）装入内部 SRAM。然后从内部 SRAM 执行自举程序。

4.1.3 如何进入备用自刷新（BSR）模式

BSR 模式是备用模式的扩展。要进入此模式，必须首先将 DDR 存储器设置为自刷新模式，并且必须为 VDDBU 和 VDDIODDR 供电。下面详细介绍了进入 BSR 模式的步骤。步骤 1 和步骤 2 可以从 DDR 执行，而步骤 3 至步骤 8 必须从内部 SRAM 执行。

1. 软件保存所有要恢复的上下文信息（取决于应用程序）。
2. 复制到 SRAM 并从 SRAM 执行程序，以将 DDR 设置为自刷新模式并关断器件（一旦 DDR 处于自刷新状态，就不可再访问 DDR）。
3. 将 DDR 置于自刷新模式并等待达到自刷新状态（请参见 SAMA5D2 Series 数据手册中的 MPDDRC 部分）。
4. 将 SFRBU_DDRBUMCR 寄存器中的 BUMEN 位置 1，以将 DDR I/O 段与 VDDCORE 关断相隔离。
5. 在关断控制器中配置唤醒源。
6. 将系统时钟切换为慢速时钟。
7. 将 PIOBU0 置为有效或将 I^2C 控制信号发送到 PMIC（这会配置 PMIC，以便在下一个关断周期保持供应 VDDIODDR）。
8. 通过将 SHDN 引脚置为有效进入 BSR 模式。

必须关闭除 DDR 存储器之外的所有外部元件。

使用 PMIC 电源解决方案时，应首先将 PMIC 配置为在关断之前保持供应适当的电源轨（例如，SAMA5D2 Xplained 板上的 VDD_1V35）。

使用分立元件解决方案时，必须控制 PIOBU 以关断除 VDDIODDR 和 VDD_1V35 之外的所有电源。

示例代码：

```
//进入 BSR 模式之前要保存的
//数据和配置（取决于应用程序）*/
...
/*将 DDR 设置为自刷新模式*/
MPDDRC->MPDDRC_LPR = MPDDRC_LPR_LPCB_SELFREFRESH;
//检查是否完成自刷新；如果未完成，则继续。
while(!(MPDDRC->MPDDRC_LPR & MPDDRC_LPR_SELF_DONE));
//使能 DDR 备用模式
SFRBU->SFRBU_DDRBUMCR = SFRBU_DDRBUMCR_BUMEN;
//在外设的 PMC 级禁止 DDR 控制器时钟信号
PMC->PMC_PCR = ( PMC_PCR_CMD | PMC_PCR_GCKCSS_MCK_CLK | (ID_MPDDRC));
//禁止 ddrclk
PMC->PMC_SCDR |= PMC_SCDR_DDRCK;
//将 PMIC 配置为 BSR 模式
board_cfg_pmic_ulpm(selfrefresh, backup);
/*配置唤醒*/
shdwc_configure_wakeup();
/*清除状态*/
(void)shdwc_get_status();
//将备用区域切换回由 VDDBU 供电。
SFRBU->SFRBU_PSWBUCTRL = SFRBU_PSWBUCTRL_WPKEY_PASSWD;
/* PCK = MCK = 32 kHz */
/*选择慢速时钟作为 PCK 和 MCK 的输入时钟*/
PMC->PMC_MCKR = (PMC->PMC_MCKR & ~PMC_MCKR_CSS_Msk) | PMC_MCKR_CSS_SLOW_CLK;
    while (!(PMC->PMC_SR & PMC_SR_MCKRDY));
```

```
/*进入备用模式*/
shdwc_do_shutdown();
```

4.1.4 如何退出 BSR 模式

通过以下任一事件启动退出 BSR 模式：

- WKUP0
- WKUP1（安全模块事件）
- WKUP2 至 WKUP9 引脚（PIOBU0 至 PIOBU7，电平跳变，可配置去抖动）
- 在低功耗 UART 接收器（RXLP）上接收到字符
- 模拟比较
- RTC 报警

当检测到唤醒事件时，关断控制器会自动驱动 SHDN 引脚。这将使电源重新启动，当释放 NRST 引脚时，将启动 ROM 引导序列。

执行 ROM 代码并将客户自举程序装入内部 SRAM 中。自举程序必须检查 SFRBU_DDRBUMCR 寄存器中 BUMEN 位的状态，并重新初始化 DDR 控制器。当 DDR 存储空间中发生访问时，DDR 存储器会自动退出自刷新模式。必须执行以下序列将 DDR 焊盘连接到 CPU 域。

示例代码：

```
if ((SFRBU->SFRBU_DDRBUMCR & SFRBU_DDRBUMCR_BUMEN) != 0)
/*将 DDR 焊盘连接到 CPU 域 VCCCORE */
SFRBU->SFRBU_DDRBUMCR &= ~SFRBU_DDRBUMCR_BUMEN;
```

4.2 超低功耗模式

超低功耗（Ultra Low-Power，ULP）模式包括两个子模式：ULP0 模式和 ULP1 模式。

如表 1-2 中所述，ULP0 和 ULP1 之间的差异在于系统中存在（ULP0）还是不存在（ULP1）时钟。在这两种模式下，SAMA5D2 都是完全供电的（即，所有电源输入均已正确供电）。

为了进一步降低 ULP0 或 ULP1 模式下的系统功耗，可以将这些 SAMA5D2 ULP 模式与应用于系统中其他元件的一些节能技术相结合。在某些情况下，甚至可能使其中一些元件掉电。

4.2.1 如何进入 ULP0 模式

下面详细介绍了进入 ULP0 模式的步骤。进入此模式时所使用的代码必须在内部 SRAM 中执行。

1. 将 DDR 置于自刷新模式。
2. 设置中断以唤醒系统。
3. 禁止所有未使用的外设时钟。
4. 将 I/O 设置为适当的状态。如果未使用 USB 收发器，则将其禁止（见 SAMA5D2 Series 数据手册中的特殊功能寄存器（SFR）部分）。
5. 将系统时钟切换为慢速时钟。
6. 禁止 PLL、主晶振和 12 MHz RC 振荡器。
7. 进入等待中断模式并在 PMC_SCDR 寄存器中禁止 PCK 时钟。

示例代码：

```
 /*备份 IO 和 USB 收发器*/
read_reg[0] = PMC->PMC_PCSR0;
```

```
read_reg[1] = PMC->PMC_PCSR1;
read_reg[2] = PMC->PMC_SCSR;
read_reg[3] = PMC->CKGR_UCKR;

/*将 DDR 设置为自刷新模式*/
MPDDRC->MPDDRC_LPR = MPDDRC_LPR_LPCB_SELFREFRESH;
//检查自刷新模式是否完成；如果未完成，则继续。
while(!(MPDDRC->MPDDRC_LPR & MPDDRC_LPR_SELF_DONE));

/*禁止 USB 收发器和所有外设时钟*/
board_save_misc_power();

/*配置唤醒*/
shdwc_configure_wakeup();

/*清除状态*/
(void)shdwc_get_status();

/*配置唤醒源和有效极性*/
pmc_set_fast_startup_polarity(0, PMC_FSPR_FSTP0);
pmc_set_fast_startup_mode(PMC_FSMR_FSTT0 | PMC_FSMR_FSTT2 | PMC_FSMR_LPM);

/*选择慢速时钟作为 PCK 和 MCK 的输入时钟*/
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");
/*进入 ULP0 模式*/
asm("WFI");

/*恢复默认 PCK 和 MCK */
pmc_set_custom_pck_mck(&clock_test_setting[0]);
_restore_console();

/*恢复 IO 和 USB 收发器*/
PMC->PMC_PCER0 = read_reg[0];
PMC->PMC_PCER1 = read_reg[1];
PMC->PMC_SCER  = read_reg[2];
PMC->CKGR_UCKR = read_reg[3];

//将备用区域中的 VDDBU 电源切换为 VDDANA，降低 VDDBU 电池的功耗
SFRBU->SFRBU_PSWBUCTRL = SFRBU_PSWBUCTRL_WPKEY_PASSWD | SFRBU_PSWBUCTRL_SSWCTRL;
/
```

4.2.2 如何退出 ULP0 模式

任何允许的中断都会触发从 ULP0 模式唤醒。在此示例中，按钮 BP1 用作唤醒源。软件的恢复方式是使用命令 asm（“cpsid if”）配置 Arm 内核，以执行 WFI 指令后面的指令。因此，在前面的示例代码中，执行的第一行代码是“pmc_set_custom_pck_mck”，以确保 DDR 存储器使用了正确的时钟频率。PMC 寄存器采用进入 ULP0 模式之前的配置，之后自刷新模式被禁止，代码可以继续执行并在正常模式下访问 DDR。

4.2.3 如何进入 ULP1 模式

下面详细介绍了进入 ULP1 模式的步骤。进入此模式时所使用的代码必须在内部 SRAM 中执行。

1. 将 DDR 置于自刷新模式。
2. 设置事件以使能系统唤醒。
3. 禁止所有外设时钟。
4. 将 I/O 设置为适当的状态并禁止 USB 收发器。
5. 将系统时钟切换到 12 MHz RC 振荡器。
6. 禁止 PLL 和主振荡器。
7. 通过以下任一方式进入 ULP1 模式：

- 将 CKGR_MOR.WAITMODE 位置 1，或
- 将 PMC_FSMR.LPM 位置 1 并执行处理器 WaitForEvent（WFE）指令。

然后，在将 WAITMODE 位置 1 或使用 WFE 指令后立即等待 PMC_SR.MCKRDY 位置 1。

示例代码：

```
/*备份 IO 和 USB 收发器*/
read_reg[0] = PMC->PMC_PCSR0;
read_reg[1] = PMC->PMC_PCSR1;
read_reg[2] = PMC->PMC_SCSR;
read_reg[3] = PMC->CKGR_UCKR;
/*将 DDR 设置为自刷新模式*/
MPDDRC->MPDDRC_LPR = MPDDRC_LPR_LPCB_SELFREFRESH;
//检查自刷新是否完成；如果未完成，则继续。
while(!(MPDDRC->MPDDRC_LPR & MPDDRC_LPR_SELF_DONE));
/*禁止 USB 收发器和所有外设时钟*/
board_save_misc_power();
/*配置唤醒*/
shdwc_configure_wakeup();
/*清除状态*/
(void)shdwc_get_status();

/*配置唤醒源和有效极性*/
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);

/*禁止 PLL 和主振荡器*/
/*超低功耗模式 1，RC12 选为主时钟*/
PMC->CKGR_MOR = (PMC->CKGR_MOR & ~CKGR_MOR_KEY_Msk) | CKGR_MOR_MOSCRCEN | CKGR_MOR_KEY_PASSWD;
/*等待内部 12 MHz RC 起振时间以使时钟稳定*/
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));

/*将主时钟切换为内部 12 MHz 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;

/*进入 ULP1 */
asm("WFE");
asm("WFE");

/*等待 PMC_SR.MCKRDY 位置 1。*/
while ((PMC->PMC_SR & PMC_SR_MCKRDY) == 0);

/*恢复默认 PCK 和 MCK */
pmc_set_custom_pck_mck(&clock_test_setting[0]);
_restore_console();

/*恢复 IO 和 USB 收发器*/
PMC->PMC_PCER0 = read_reg[0];
PMC->PMC_PCER1 = read_reg[1];
PMC->PMC_SCER  = read_reg[2];
PMC->CKGR_UCKR = read_reg[3];

//将备用区域中的 VDDBU 电源切换为 VDDANA，以降低 VDDBU 电池的功耗
   SFRBU->SFRBU_PSWBUCTRL = SFRBU_PSWBUCTRL_WPKEY_PASSWD | SFRBU_PSWBUCTRL_SSWCTRL;
```

4.2.4 如何退出 ULP1 模式

退出 ULP1 模式的方法与上述退出 ULP0 模式的方法类似。

在上面的示例代码中，RTC 用作唤醒源。

软件通过执行跟在 WFE 指令后的指令恢复。因此，在前面的示例代码中，执行的第一行代码是“pmc_set_custom_pck_mck”，以确保 DDR 存储器使用了正确的时钟频率。PMC 寄存器采用进入 ULP1 模式之前的配置，之后自刷新模式被禁止，代码可以继续执行并在正常模式下访问 DDR。

4.3 空闲模式

由于空闲模式的目的是降低器件功耗，因此所有电源都在指定范围内工作。

此模式下的功耗取决于应用，但可以通过在 L2 高速缓存控制器中使能动态时钟门控（设置 L2CC_POWCR.DCKGATEN = 1）来降低功耗。

4.3.1 如何进入空闲模式

要进入空闲模式，请禁止 PCK 并执行等待中断（WFI）指令。

示例代码：

```
//禁止 PCK
PMC->PMC_SCDR = PMC_SCDR_PCK;
//进入空闲模式
asm(“wfi”);
```

4.3.2 如何退出空闲模式

可以通过中断将处理器从空闲模式唤醒。系统恢复到进入 WFI 模式之前的状态。

示例代码：

```
static void configure_buttons(void)
{
   int i = 0;
   for (i = 0; i < ARRAY_SIZE(button_pins); ++i){
   //将 PIO 配置为输入。
   pio_configure(&button_pins[i], 1);
   //调节 PIO 去抖滤波器参数，这里我们设置为 10 Hz 滤波器，
   //这是示例。
   pio_set_debounce_filter(&button_pins[i], 10);
   //通过处理程序初始化 PIO 中断，
   //请参见 board.h 中的 PIO 定义。
   pio_configure_it(&button_pins[i]);
   pio_add_handler_to_group(button_pins[i].group,
   button_pins[i].mask, pio_handler);
   //允许 PIO 线中断。
   pio_enable_it(button_pins);
   }
}
```

在上述示例代码中，按钮 BP1 用作唤醒源， 因此在进入空闲模式之前，相应 PIO 必须配置为中断源。

5. Linux 软件实现

以下部分基于 Linux4SAM 版本 5.7。请参见 http://www.linux4sam.org。

Linux 电源管理（Power Management，PM）框架提供了多种方法来实现节能目的。一些降低功耗的方法包括：

- 使用无时标内核
- 在运行时调整系统时钟
- 根据设备的使用情况将其置于休眠模式
- 将系统暂停并保存到存储器中
- 将系统置于冬眠模式

Linux 电源管理选项分为两个主要类别：

- 运行时电源管理
- 系统休眠模型

运行时电源管理是指将设备切换到休眠状态（通过禁止时钟或切换到高级电源管理硬件相关的状态）。

系统休眠电源管理模型是指通过将整个系统置于节能状态来执行与系统相关的电源管理程序。

第 5.2 节中讨论的电源管理选项是系统休眠电源管理模型的一部分。运行 Linux 操作系统且采用基于 SAMA5D2 的 SoC 的系统可以切换到低功耗模式，节能效果则取决于选择的模式。

本应用笔记仅介绍系统休眠模型，因为此模型与第 1.1 节中介绍的低功耗模式相关。

5.1 Linux 电源管理内核（系统休眠模型）

电源管理内核实现提供了基础结构，允许系统从活动运行模式切换到低功耗模式（暂停操作）以及从低功耗模式切换到活动模式（恢复操作），而不会影响系统功能。

标准 Linux 实现提供了四种节能方法。Linux 休眠状态包括：

- 暂停到空闲
- 上电暂停
- 暂停到 RAM
- 暂停到磁盘

本节介绍了上电暂停和暂停到 RAM 节能模式。

5.1.1 文件系统接口

Linux 文件系统接口使用户能够切换到所需的电源管理状态、设置电源管理参数和检索统计信息。

用于电源管理的主要 Linux 文件系统接口包括：

```
/sys/powernull
/sys/kernel/debug/sleep_timenull
/sys/kernel/debug/suspend_statsnull
/sys/kernel/debug/wakeup_sourcesnull
/sys/devices/.../power/wakeup
```

/sys/power是主接口，用于控制到低功耗休眠状态的切换。选项包括：

```
ls -l /sys/power/null
-rw-r--r-- 1 root root 4096 Jan 12 17:37 pm_asyncnull
-rw-r--r-- 1 root root 4096 Jan 12 17:37 pm_freeze_timeoutnull
-rw-r--r-- 1 root root 4096 Jan 12 17:37 pm_print_timesnull
-rw-r--r-- 1 root root 4096 Jan 12 17:37 pm_testnull
-r--r--r-- 1 root root 4096 Jan 12 17:37 pm_wakeup_irqnull
-rw-r--r-- 1 root root 4096 Jan 12 17:37 statenull
-rw-r--r-- 1 root root 4096 Jan 12 17:37 wakeup_count
```

/sys/power/state 接口是主要的电源管理文件系统接口，用于触发到节能状态的转换。在基于 SAMA5D2 Linux 的系统中，受支持的休眠状态包括：

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

要切换到其中一种状态，可将所需状态对应的字符串写入/sys/power/state 文件，如表 5-1 所示。

表 5-1. Linux 休眠状态和命令

Linux 休眠状态	进入休眠状态的命令
暂停到空闲	echo freeze > /sys/power/state
上电暂停或待机	echo standby > /sys/power/state
暂停到 RAM	echo mem > /sys/power/state
暂停到硬盘或冬眠	/sys/power/state 中不存在字符串“disk”时为 NA

有关 Linux 休眠状态的概述，请查看 https://www.kernel.org 上的文档部分。

下面简要说明了文件系统接口：

- /sys/power/pm_async：允许暂停和恢复某些设备的回调，这些回调彼此间并行执行并且与主暂停线程并行执行
- /sys/power/pm_freeze_timeout：指定冻结所有可冻结进程所花费的时间
- /sys/power/pm_print_times：用于输出设备执行暂停和恢复操作所花费的时间
- /sys/power/pm_test：用于测试电源管理选项
- /sys/power/pm_wakeup_irq：输出唤醒 IRQ
- /sys/power/wakeup_count：读操作返回唤醒事件的数量，写操作中止到休眠状态的当前转换

出于调试目的，debugfs 文件系统中提供了下列文件：

- /sys/kernel/debug/sleep_time 输出休眠操作花费的时间（以秒为单位）。
- /sys/kernel/debug/suspend_stats 输出暂停统计信息。
- /sys/kernel/debug/wakeup_sources 输出当前注册的唤醒源。

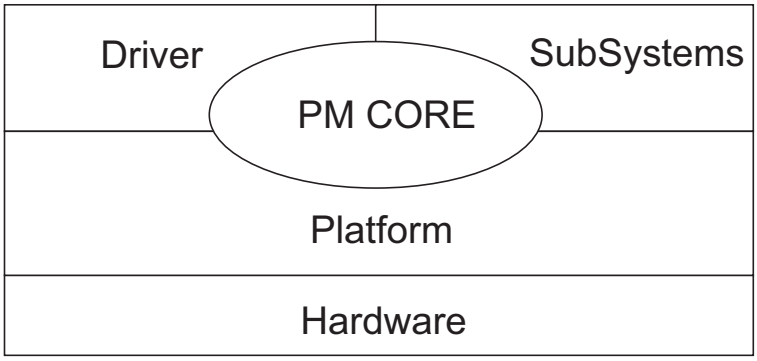
可以基于从设备或外部引脚接收的事件从低功耗状态恢复。可以设置设备，使其用作唤醒源。Linux 文件系统提供了一个可以为设备设置唤醒功能的接口：

- /sys/devices/.../power/wakeup 文件可以通过写入“enabled”或“disabled”字符串来设置，以将设备用作唤醒源或不用作唤醒源。

5.1.2 内核实现

下图总结了 Linux 电源管理实现的主要元素。

图 5-1. Linux 电源管理架构概览



在 Linux 上下文中，上图中使用的术语定义如下：

- 驱动程序：用于控制硬件（IP/MCU/MPU）的计算机程序
- 子系统：同类驱动程序共用的代码。例如：I^2C 驱动程序共用构成 Linux I^2C 子系统的通用代码并实现特定于 I^2C 设备的通用算法。
- 平台：实现平台特定的算法的代码（例如，SAMA5D2 实现其自己的平台代码，此代码控制 SAMA5D2 特定的功能，如备份和自刷新功能）
- PM 内核：电源管理代码，实现通用电源管理算法并激活子系统、驱动程序和平台电源管理逻辑
- 硬件：受驱动程序、子系统、平台和 PM 内核采取的操作影响的 IP/MCU/MPU

PM 内核包含在进入/退出节能状态时执行的通用电源管理代码。冻结或暂停到空闲状态会冻结用户空间进程并使硬件处于工作状态。其他节能状态会将硬件切换到更节能的模式。

如图 5-1 所示，为了能够切换到更高级的节能状态，下列三个元素必须提供可由 PM 内核在暂停/恢复操作中访问的 PM Hook 函数：

- 设备驱动程序
- 注册设备驱动程序的子系统
- 平台代码

硬件必须支持高级节能模式。

在 Linux 的平台初始化阶段，平台代码必须将平台特定的电源暂停/恢复代码注册到 PM 内核。方法是通过向 PM 内核提供如下所示的结构：

```
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);
};
```

平台 PM 相关代码在暂停操作的最后一步和恢复操作的第一步中执行。根据节能模式，平台代码会将 CPU 置于不同的节能模式。

Linux 内核设备模型引入了设备、类和总线。PM 内核利用这些概念来选择和应用正确的 PM 策略。此外，父设备和子设备的概念在暂停/恢复过程中非常有用，有助于以正确的顺序暂停/恢复设备。在 Linux 中，驱

动程序实现了 struct devices 类型的结构，子系统实现了 struct class 或 struct bus_type 类型的结构。设备驱动程序模型的每个结构均实现了一个 struct dev_pm_ops 类型的结构，如下所示：

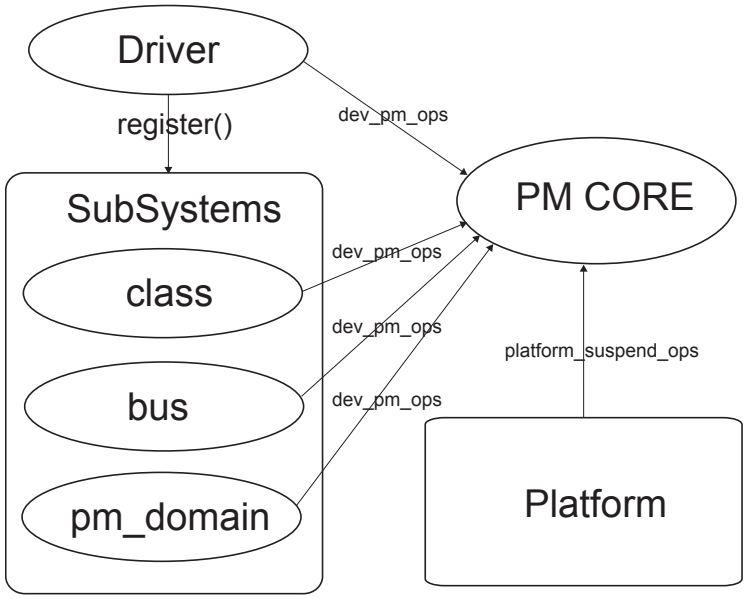
```
struct dev_pm_ops {
int (*prepare)(struct device *dev);
void (*complete)(struct device *dev);
int (*suspend)(struct device *dev);
int (*resume)(struct device *dev);
int (*freeze)(struct device *dev);
int (*thaw)(struct device *dev);
int (*poweroff)(struct device *dev);
int (*restore)(struct device *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);
};
```

驱动程序和子系统实现 struct dev_pm_os 类型的结构，PM 内核按顺序执行正确的硬件相关设置，以将外设切换到高级节能模式。每个子系统或驱动程序通过该结构将暂停/恢复功能提供给 PM 内核（图 5-2），PM 内核可将外设切换到正确的电源状态。请注意，旧平台驱动程序仅通过 struct platform_driver 将暂停和恢复功能提供给 PM 内核。PM 内核执行这些功能。

除了 Linux 设备驱动程序模型外，PM 内核还引入了电源域的概念。电源域包含共用参考时钟或电源的一个或多个设备。设备可以是电源域的一部分，电源域可以嵌套。此外，电源域还通过 struct dev_pm_ops 结构向 PM 内核提供 PM 功能（图 5-2）。

如图 5-2 所示，驱动程序和子系统将 PM 操作注册到 PM 内核，如果需要实现特定的节能模式，则还会将与平台相关的 PM 操作注册到 PM 内核。

图 5-2. Linux 电源管理实现概览



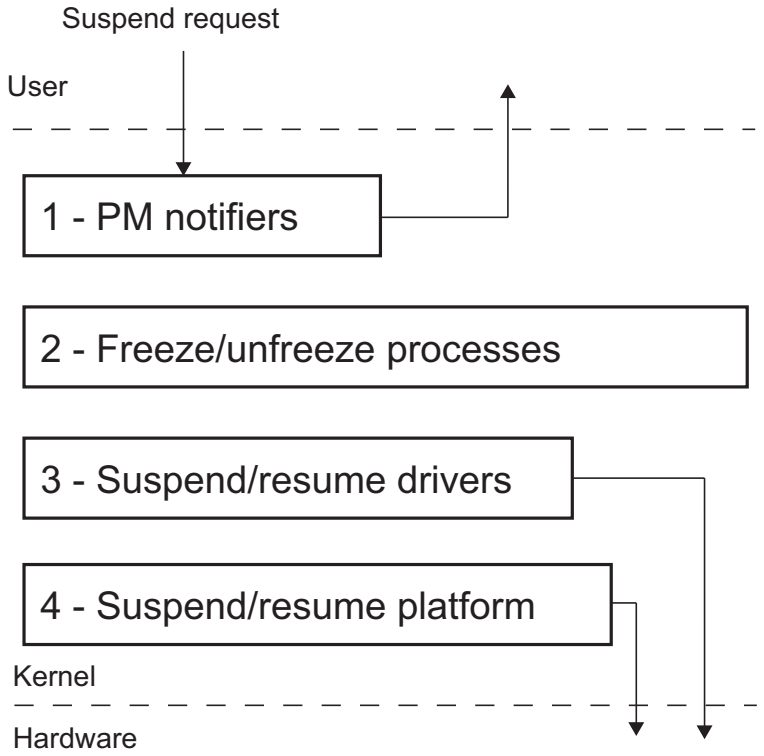
如图 5-2 所示，根据 Linux 设备驱动程序模型，可通过以下方式为一个设备注册多个 PM 操作：

- 设备电源域、
- 设备类、
- 设备总线、
- 设备驱动程序。

在暂停/恢复时，PM 内核按照上面指定的顺序检查已注册的 PM 操作，并执行匹配的第一个 PM 操作。已注册的 PM 操作是互斥的，这意味着 PM 内核在暂停/恢复序列中仅执行设备的电源域、类、总线或驱动程序的 PM 操作。

图 5-3 给出了一般的暂停/恢复步骤。

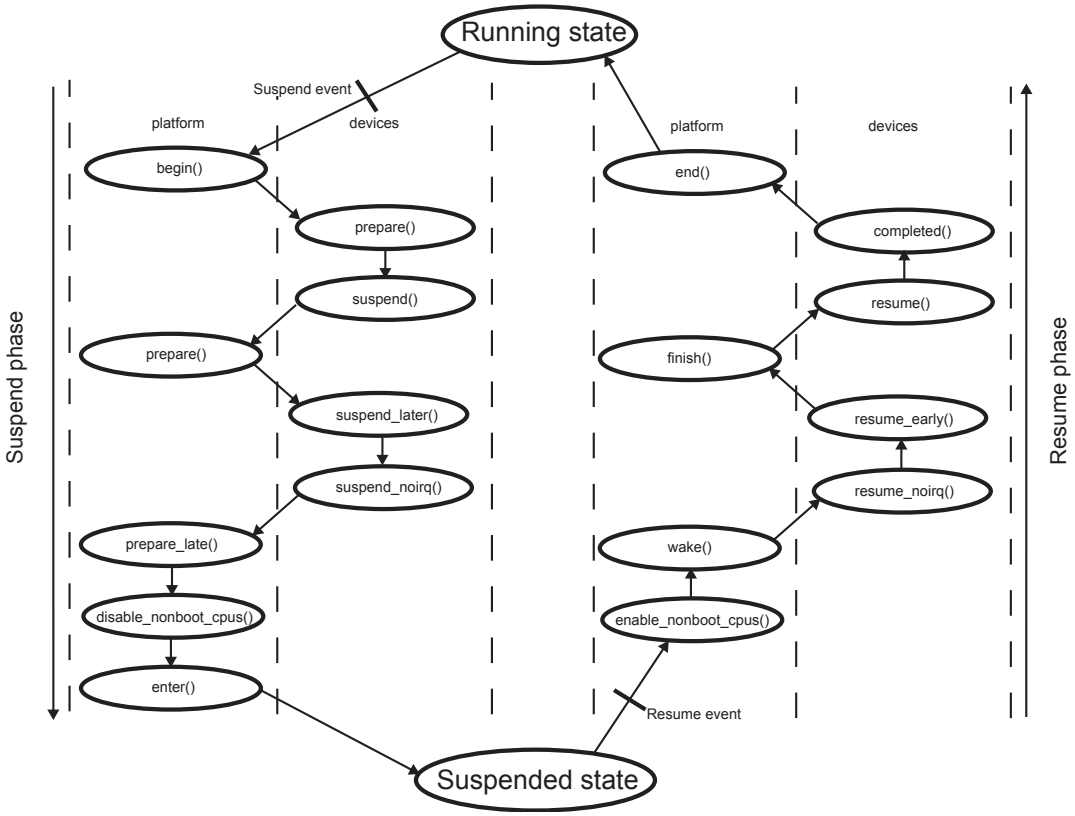
图 5-3. 一般的暂停/恢复阶段



如图 5-3 中所示，暂停请求从用户空间开始初始化。应用程序可以注册在第一个暂停阶段调用的 PM 相关通知程序。在此之后，内核会冻结所有进程（内核和用户空间）并依次暂停设备和平台。恢复过程将经过图 5-3 所述的状态，但顺序相反。

图 5-4 详细说明了图 5-3 中指定的设备和平台的暂停和恢复操作。图中给出了平台初始化代码和驱动程序探测函数中指定的平台 PM 操作（见 `struct platform_suspend_ops`）和设备 PM 操作（见 `struct dev_pm_ops`）的执行顺序。

图 5-4. 暂停/恢复详细阶段



5.2 SAMA5D2 上的电源管理实现

5.2.1 支持的模式

SAMA5D2 SoC 支持的 Linux 电源管理模式如下：

- 空闲
- ULP0
- ULP1
- 备用自刷新（BSR）

在空闲模式下，CPU 内核将处于等待中断（WFI）状态。

BSR、ULP0 和 ULP1 模式已在第 4.1 节和第 4.2 节中进行了介绍。

本节将介绍 BSR 和 ULP0/ULP1 模式的实现。

5.2.2 AT91Bootstrap 支持

在 BSR 模式下，CPU 内核（备用区域除外）关闭。为了能够恢复系统，添加了 AT91Bootstrap 支持。此支持基于以下事实：备用区域是系统暂停时惟一保持活动状态的部分（DDR 存储器除外，它仍处于自刷新模式）。Linux 和 AT91Bootstrap 通过备用区域（SECURAM）进行通信。

下图显示了 AT91Bootstrap 和 Linux 在 BSR 模式下的暂停和恢复过程中如何相互交互。

图 5-5. BSR 模式下的暂停和恢复过程中的 Linux 和 AT91Bootstrap 通信

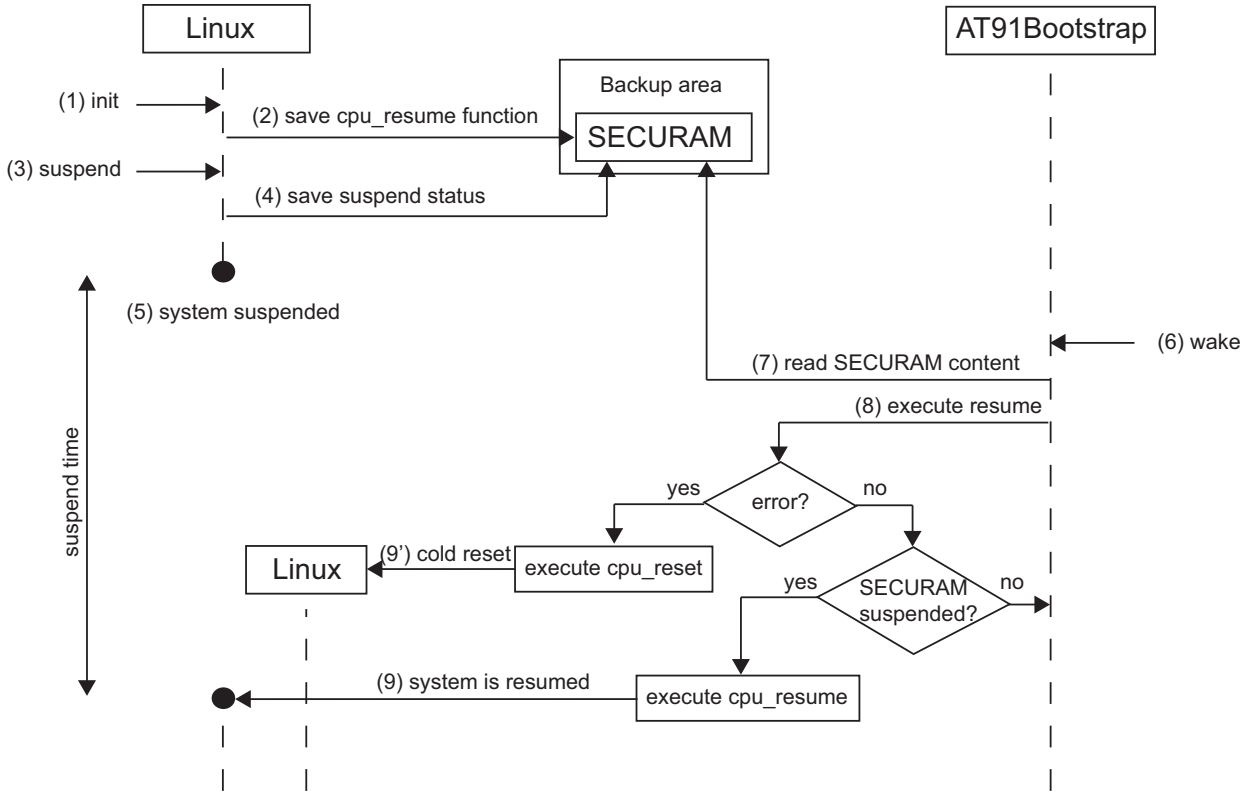


图 5-5 给出了 BSR 模式的标准暂停/恢复序列。如图中所示，当初始化暂停请求时，Linux 会将恢复代码和暂停状态保存在 SECURAM 中，同时内核电源会关闭。当检测到唤醒事件时，AT91Bootstrap 会启动并检查暂停状态以及寄存器内容是否已更改。根据从 SECURAM 读取的暂停状态，AT91Bootstrap 将跳转到 Linux 执行程序，以执行恢复代码。如果 AT91Bootstrap 在恢复过程中检测到错误，则执行冷复位。

5.2.3 Linux 内核参数

根据前面几节所述，Linux 的四种休眠状态与 SAMA5D2 的四种电源管理模式无法一对一匹配。SAMA5D2 SoC 电源管理模式可视为在 Linux“上电暂停”休眠模式的基础之上经过略微调整而形成的模式。

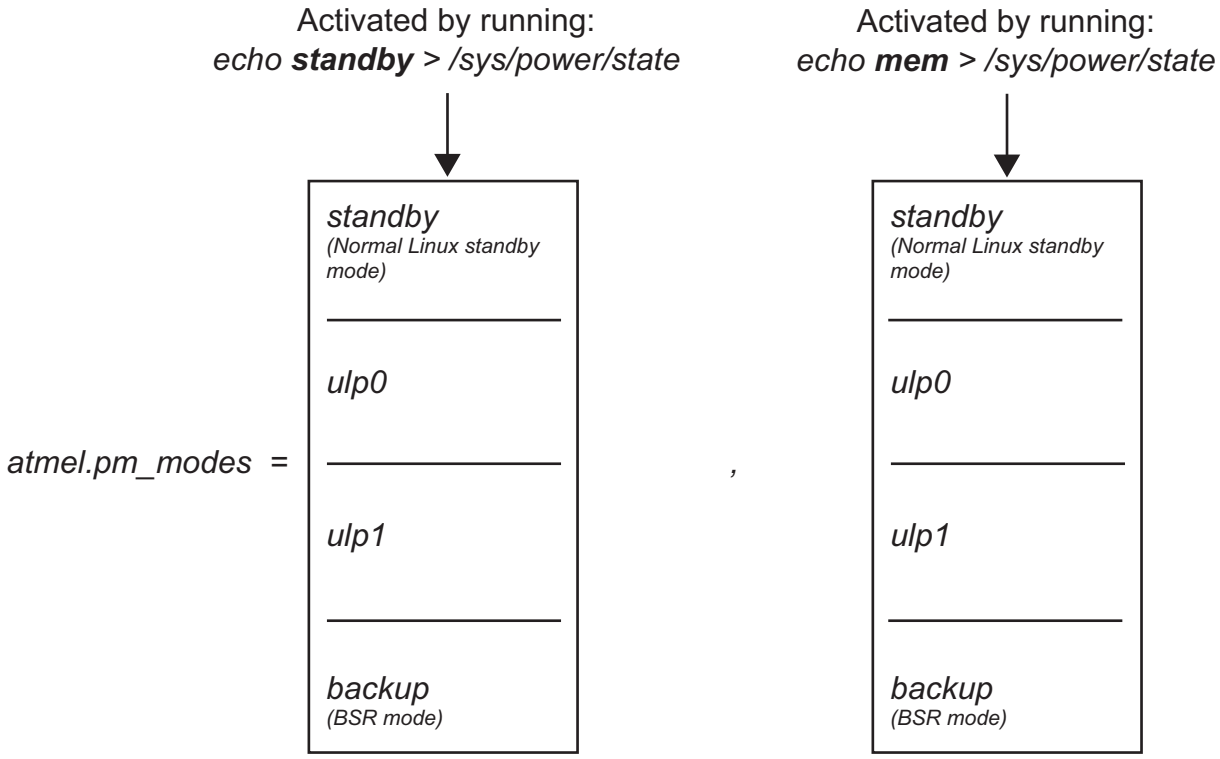
目前的技术能够让用户选择将哪种 SAMA5D2 电源管理模式映射到一个特定的 Linux 休眠模式。此映射将在系统初始化阶段使用 Linux 内核命令行参数以静态方式完成。这些参数与任何其他内核参数一样，均可以在自举程序启动 Linux 时传送。

U-Boot 用于 Linux4SAM 参考发行版，新参数附加到 `bootargs` U-Boot 变量。

对于所选的两种 SAMA5D2 低功耗模式，新的内核参数 `atmel.pm_modes` 需要两个参数。

`atmel.pm_modes` 格式如下：

图 5-6. “atmel.pm_modes”命令格式



例如，要在正常的 Linux 待机模式和 SAMA5D2 BSR 模式下初始化系统，应对 `atmel.pm_modes` 使用以下值：

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

可以使用以下命令从运行的 Linux 系统中获取 `atmel.pm_modes` 的值：

```
cat /proc/cmdline
```

这样，用户便可从 Linux 中检查 Linux 休眠状态（上电暂停和暂停到 RAM）和 SAMA5D2 电源管理模式（空闲、ULP0、ULP1 和 BSR）之间的映射。

5.2.4 Linux 支持

通过为驱动程序代码和平台代码添加暂停/恢复支持，实现了第 5.3.1 节所列的模式（见图 5-1 和图 5-2）。在驱动程序上，实现了 `struct dev_pm_ops` 类型的结构。在平台上，实现了 `struct platform_suspend_ops` 类型的结构。

5.2.5 驱动程序实现

如 5.1.2 内核实现所述，Linux 电源管理通过调用 `struct dev_pm_ops` 类型（旧设备驱动程序为 `platform_driver`）对象中驱动程序提供的函数来暂停和恢复 SoC 外设。要实现 Linux 系统休眠模型，驱动程序必须 `struct dev_pm_ops`（或 `struct platform_driver`）类型的对象中实现暂停和恢复成员。

由于在备用和自刷新模式下会切断 CPU 电源，因此必须保存外设状态（寄存器内容）。由于存储器保持自刷新模式，因此会保留其内容，并且可以保存外设状态。在驱动程序的恢复部分中，此状态从存储器中恢复。

以 UART 驱动程序为例。如 5.1.2 内核实现所述，驱动程序必须向 PM 内核提供暂停和恢复函数。在 Linux 内核源代码中，UART 驱动程序位于 drivers/tty/serial/atmel_serial.c 中，暂停和恢复函数分别为 atmel_serial_suspend 和 atmel_serial_resume。这些函数通过 struct platform_driver 类型的对象提供给 Linux PM 内核，如下所示：

```
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),
   },
};
```

在 UART 驱动程序的探测阶段（见探测函数 atmel_serial_probe），通过以下代码行实现：

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

由于在 BSR 模式下，CPU 电源关闭并且 DDR 保持自刷新模式，因此 IP 的状态保存在 DDR 存储器中。UART 驱动程序使用 atmel_serial_suspend 函数中的以下代码保存 IP 的状态：

```
if (atmel_is_console_port(port) && !console_suspend_enabled) {
/*缓存寄存器值，因为我们不会获得完整的关断/启动周期*/
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);
}
```

在恢复阶段，恢复此状态，如 atmel_serial_resume 函数所示：

```
if (atmel_is_console_port(port) && !console_suspend_enabled) {
/*缓存寄存器值，因为我们不会获得完整的关断/启动周期*/
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 平台实现

在平台方面，添加了相应的代码以处理第 5.3.1 节中指定的模式（主要用于 BSR 和 ULP0/ULP1 模式）。

在 Linux 初始化阶段，通过读取命令行参数检查平台代码是否需要处理这些模式。

电源管理平台初始化代码检查设备树中的以下各项：

- 关断控制器（SHDWC）
- 特殊功能寄存器备用（SFRBU）
- SECURAM
- DRAM 控制器
- 电源管理控制器（PMC）

存储器映射这些设备。存储器映射区域用于暂停/恢复过程（见 at91_pm_suspend_in_sram 函数）。

在与平台相关的 PM 初始化阶段，会将恢复函数的暂停状态和地址写入 SECURAM。在恢复过程中，AT91Bootstrap 根据需要使用此信息恢复 Linux（见图 5-5）。

在初始化阶段之后，平台可随时执行暂停/恢复操作。平台暂停/恢复是暂停/恢复的最后一个阶段/第一个阶段（见图 5-4 中的 `platform enter()` 状态和 `platform end()` 状态）。

最后一个暂停阶段从内部 SRAM 执行。在平台初始化时和每次恢复操作之后，相应的代码将复制到 SRAM。

根据暂停模式（BSR 或 ULP0/ULP1 模式）的不同，将在最后一个暂停阶段实现不同程度的节能。

暂停/恢复过程的最后一个阶段/第一个阶段是用汇编代码编写的，因为这一阶段将从 SRAM 执行（执行电源管理指令时，主存储器（DDR）将处于自刷新模式）。最后一个暂停阶段的主要入口点是 `at91_pm_suspend_in_sram()` 函数。此函数接收的参数是 `struct at91_pm_data` 类型的对象（在平台初始化代码中完成初始化），用于保存暂停/恢复的最后一个阶段/第一个阶段所需的所有数据。

```
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;
};
```

暂停过程最后一个阶段的详细步骤如下 （如 `at91_pm_suspend_in_sram()` 函数所示）：

- 将寄存器内容保存在堆栈中
- 将 `struct at91_pm_data` 类型对象的内容存储到相应的寄存器上
- 激活 DDR 自刷新模式
- 切换到所选的电源管理模式

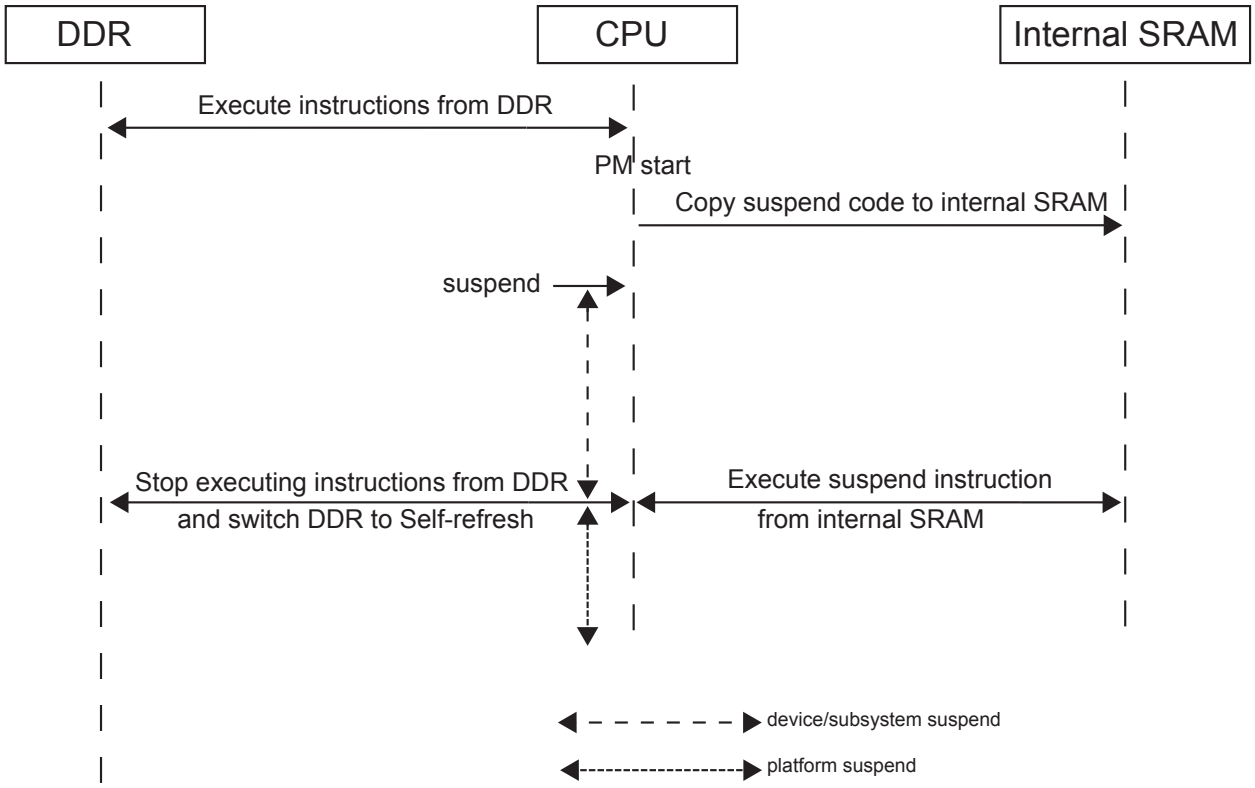
根据所选的电源管理模式进一步设置，具体如下：

1. 空闲模式：CPU 切换到 WFI。
2. ULP0/ULP1 模式：
   - 保存主时钟设置。
   - 将主时钟源切换到慢速时钟。
   - 保存 PLLA 设置，然后禁止 PLLA。

   2.1. ULP0 模式：
   - 关闭晶振。
   - CPU 切换到 WFI。

   2.2. ULP1 模式：
   - 将主时钟源切换到 12 MHz RC 振荡器。
   - 禁止晶振。
   - 将主时钟源切换到主时钟。
   - 进入 ULP1 模式：CKGR_MOR.WAITMODE=1。
3. BSR 模式：
   - 关断 CPU。

图 5-7. 最后一个暂停阶段



在 BSR 模式下，CPU 内核掉电（这就是在图 5-7 中的平台暂停后中断 CPU 和内部 SRAM 时间线的原因）。对于 ULP0/ULP1 模式，处理器电源保持供应，因此在执行平台暂停代码后，CPU 和内部 SRAM 时间线将继续。

恢复阶段执行的设置与暂停阶段相反。具体取决于暂停模式：

1. 空闲模式：禁止存储器自刷新模式，并恢复堆栈。
2. ULP0/ULP1 模式：
   - 2.1. ULP0 模式：
     - 开启晶振。
   - 2.2. ULP1 模式：
     - 开启晶振。
     - 将主时钟源切换到慢速时钟。
     - 将主时钟源切换到晶振。
     - 将主时钟源切换到主时钟。
   - 2.3. ULP0/ULP1 模式：
     - 恢复 PLLA 设置。
     - 恢复主时钟设置。
3. 备用模式：`cpu_resume` 函数加载由暂停过程保存的堆栈内容，然后从暂停之前的位置继续执行。

6. 测量结果

本章将介绍如何通过执行测量来评估不同模式下的功耗。所述测量在室温条件下进行。

功耗测量基于 SAMA5D2 Xplained 板。在 SAMA5D2 Xplained 板上，所有 SAMA5D2 电源轨均由电源管理 IC ACT8945AQJ405 供电（见 ACT8945A 数据手册）。图 2-3 是 PMIC 器件原理图（见 SAMA5D2 Xplained Ultra 评估工具包用户指南中的原理图部分）。

6.1 条件

6.1.1 测量相关的电路板设置

为了在 SAMA5D2 Xplained 板上执行不同的测量，裸机或 Linux 应用程序时钟的设置如下：

- CPU 时钟为 498 MHz
- 外设/主器件时钟为 166 MHz

电路板上提供了多个测量点，所有这些测量点均由特定的电源轨供电。图 6-1 是 SAMA5D2 Xplained 板的俯视图，其中显示了用于接入所有这些电源轨的跳线。为了能够接入两个未连接到电路板上跳线的电源轨，电路板进行了相应的修改。请参见表 6-1 和表 6-2。此外，该电路板上只使用了一种 DDR 存储器。使用和配置的裸机应用程序和 Linux OS 是根据电路板而定制的。

图 6-1. SAMA5D2 Xplained 板上的电源跳线

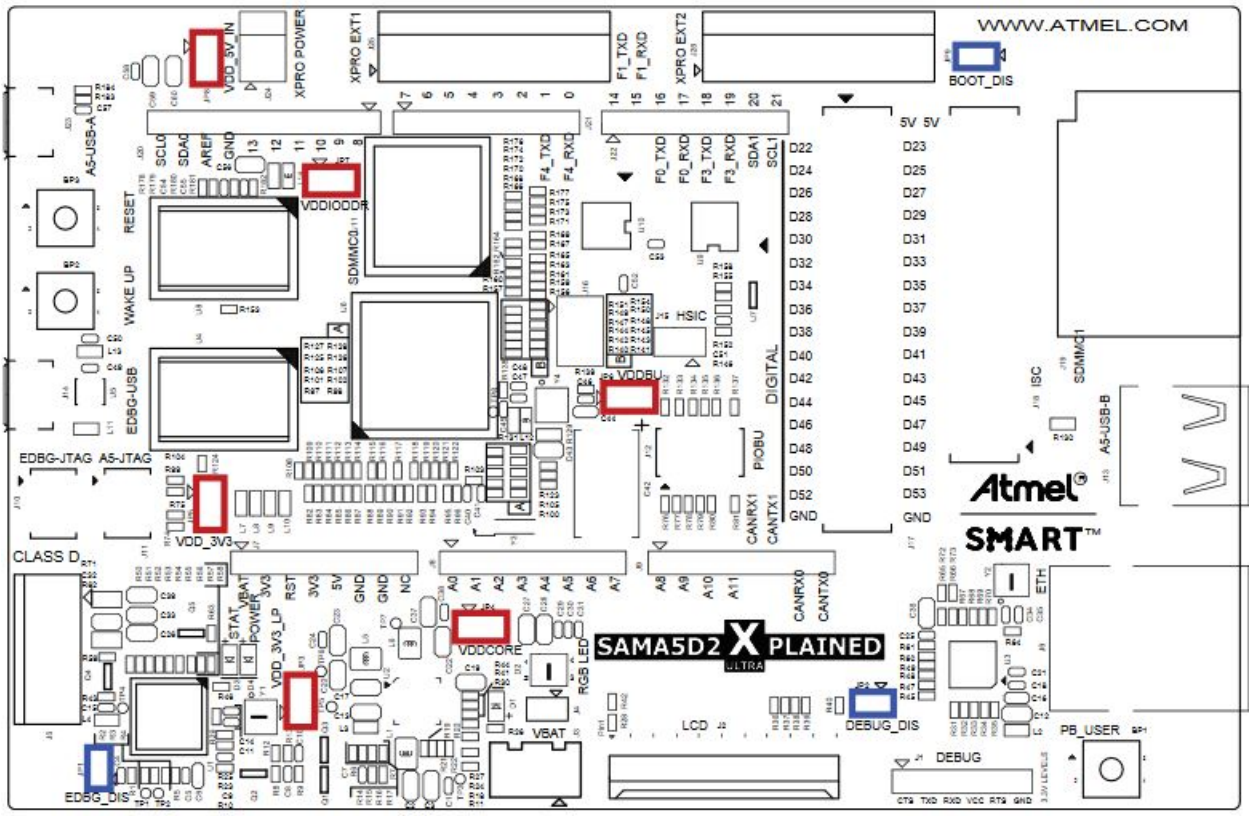


图 6-2. VDD_1V2 和 VDD_1V35 电源轨接入

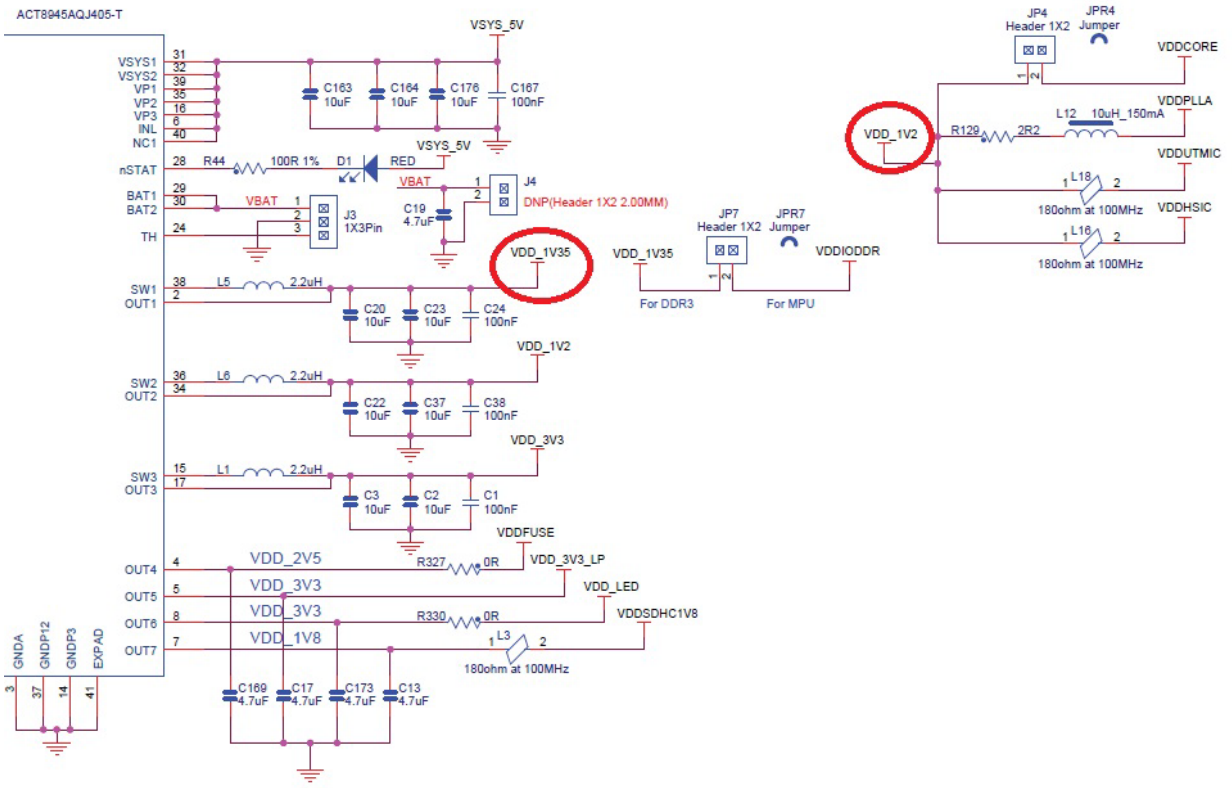


表 6-1 给出了跳线和电源轨之间的对应关系，以及连接到这些跳线的不同电源轨的说明。

表 6-1. 电源轨测量接入和供电

电源轨	跳线	电源轨供电对象
VDDBU	JP6	备用区域
VDDCORE	JP4	VDDCORE 电源轨
VDD_1V2	N/A(1)	VDDCORE、VDDPLLA、VDDUTMIC 和 VDDHCSIC
VDD_5V_IN	JP8	PMIC 的主电源
VDDIODDR	JP7	VDDIODDR 电源轨
VDD_1V35	N/A(1)	DDR 存储器
VDD_3V3_LP	JP3	VDDOSC、VDDUTMII、VDDANA 和 VDDAUDIOPLL
VDD_3V3	JP5	VDDIOP0、VDDIOP1、VDDIOP2 和 VDDISC

注：

1. 在执行这些测量之前，SAMA5D2 Xplained 板经过了修改，方便单独接入：
   - VDDCORE 功耗和 VDD_1V2 功耗（VDDCORE、VDDPLLA、VDDUTMIC 和 VDDHCSIC）
   - 由 VDD_1V35 供电的 DDR 存储器功耗和 VDDIODDR 电源轨功耗（VDD_1V35 直接在存储器上获取）

除了用于测量目的的跳线之外，以下跳线必须断开：

- JP1（禁止 EDBG）
- JP2（禁止调试）
- JP9（禁止 SPI/QSPI 存储器的 CS），当应用程序未存储在 SDMMC 或 eMMC 中

有关 SAMA5D2 Xplained 板的更多信息，请参见 SAMA5D2 Xplained Ultra 评估工具包用户指南（见参考文档）。

6.1.2 Linux 特定的代码更改

使用 SAMA5D2 Xplained 板进行测量，同时监视 PB5 引脚切换。来自文件 *linux4sam-poky-sama5d2_xplained-5.7.img.bz2*[(1)]的 Linux SD 卡映像通过附录 A 中给出的 GIT 补丁进行修补（见附录 A. 时间测量相关的 Linux 代码补丁）。

注：

1. 下载地址：ftp://www.at91.com/pub/demo/linux4sam_5.7。

6.2 暂停/唤醒时间测量

本节规定了 SAMA5D2 低功耗模式的暂停/唤醒时间。

6.2.1 测量条件

要测量暂停时间，需在暂停指令之前和暂停完成之后切换引脚。该原理同样适用于测量唤醒时间：在恢复启动的瞬间和恢复完成后切换引脚。

在裸机应用程序中，暂停测量在本应用笔记中的每个裸机示例代码开始时启动（见 4.1.1、4.1.3、4.2.1 和 4.2.3 节），在进入低功耗模式之前完成。唤醒时间测量在进入低功耗模式的指令之后立即启动，直到示例代码结束。

6.2.2 测量值

测试是在裸机和 Linux 操作系统上完成的。总共进行五组测量，每组五次，每完成五次测量重启一次电路板。

主要测试差异：

- Linux OS 测试：
  - 与 Linux4sam 版本 5.7 相对应的应用程序正在执行从 SD 卡中的根文件系统运行的简单嵌入式发行版。
  - 大多数外设状态均在暂停和恢复过程中进行保存和恢复。
- 裸机 OS 测试：
  - 只有一个应用程序（LED 闪烁）正在运行。
  - 外设未在暂停和恢复过程中进行处理。

下表汇总了测量结果。

表 6-2. 裸机和 Linux 的暂停/唤醒时间测量

模式	操作系统	进入低功耗模式的时间	唤醒时间
空闲	裸机	<10 µs[(1)]	<1 µs[(1)]
	Linux	N/A	NA

.....（续）

模式	操作系统	进入低功耗模式的时间	唤醒时间
ULP0	裸机	5.6 ms	4 ms
	Linux	184 ms	246 ms
ULP1	裸机	350 µs	15 µs
	Linux	201 ms	238 ms
BSR	裸机	1.4 ms	600 µs
	Linux	175 ms	2123 ms(2)

注：

1. 指示性时间。使用本应用笔记中给出的示例代码无法测得实际时间（ns 范围）。
2. 从 AT91Bootstrap 版本 3.8.12 开始，BSR 的唤醒时间缩短至 800 ms 左右。

注： 要从低功耗模式唤醒系统，SAMA5D2 Xplained 板上的 5V 电源必须保持连接。

在 Linux 操作系统中，时间很大程度上取决于外设状态的保存和恢复过程。通过删除一些不必要的操作可以显著改善上述测量结果，因为上下文的主要部分是在 ULP0/ULP1/IDLE 模式下进行维护的。

6.3 功耗测量

本节将介绍使用裸机应用程序和 Linux 应用程序进行的 SAMA5D2 Xplained 电路板功耗测试的结果。

6.3.1 测量值

表 6-3 给出了应用程序处于运行模式时每个电源轨上的功耗。

对于 Linux，启动 Linux4SAM 5.7 发行版，Root 文件系统在 SD 卡中，内核运行一些守护进程。

表 6-3. 应用程序处于运行模式

电源轨	应用程序	功耗	备注
VDD_1V35	Linux	18 mA	DDR 上电。
	裸机		
VDDIODDR	Linux	8 mA	VDDIODDR 电源轨上电，因此可以与 DDR 通信。
	裸机		
VDDCORE	Linux	65 mA	这种差异表明 Linux 调度程序使用空闲模式来降低其功耗。
	裸机	86 mA	
VDDBU	Linux	1.5 µA	通过 SFRBU_PSWBUCTRL 将电源备用区域设置为由 VDDANA 而非 VDDBU 供电，这样可延长电池使用寿命。
	裸机		

.....（续）

电源轨	应用程序	功耗	备注
VDD3V3	Linux	1.5 mA	在 Linux 方面，将实际运行一个应用程序，因此以太网 PHY 和海量存储设备上会有活动。
	裸机	220 µA	由于这是裸机应用程序，因此 GPIO 上没有活动。功耗处于最低水平。
VDD3V3LP	Linux	30 mA	在 Linux 方面，由 VDDUTMI、VDDAUDIOPLL 和 VDDANA 供电的外设（UTMI 收发器、音频 PLL 和模拟器件等）处于使能状态，但这些外设在裸机中处于禁止状态。
	裸机	1.7 mA	
VDD5V	Linux	120 mA	Linux 方面的应用程序要比裸机方面复杂得多。
	裸机	98 mA	

表 6-4 给出了应用程序处于空闲模式时每个电源轨上的不同功耗。

对于 Linux，使用以下命令行参数选择空闲模式：

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

并运行以下命令：

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

表 6-4. 应用程序处于空闲模式

电源轨	应用程序	功耗	备注
VDD_1V35	Linux	18 mA	DDR 上电。
	裸机		
VDDIODDR	Linux	8 mA	VDDIODDR 电源轨上电，因此可以与 DDR 通信。
	裸机		
VDDCORE	Linux	46 mA	-
	裸机	35 mA	-
VDDBU	Linux	1.5 µA	通过 SFRBU_PSWBUCTRL 将电源备用区域设置为由 VDDANA 而非 VDDBU 供电，这样可延长电池使用寿命。
	裸机		
VDD3V3	Linux	1.1 mA	-
	裸机	2.3 mA	-

.....（续）

电源轨	应用程序	功耗	备注
VDD3V3LP	Linux	6.4 mA	-
	裸机	1.7 mA	-
VDD5V	Linux	68 mA	内核上电但不受时钟驱动。外设上电且受时钟驱动。 在 Linux 方面，适当管理外部元件（以太网 PHY 的待机模式）可以降低总功耗。
	裸机	78 mA	

表 6-5 给出了应用程序处于 ULP0 模式时每个电源轨上的不同功耗。

对于 Linux，使用以下命令行参数选择 ULP0 模式：

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

并运行以下命令：

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

表 6-5. 应用程序处于 ULP0 模式

电源轨	应用程序	功耗	备注
VDD_1V35	Linux	11 mA	DDR 电源保持供应，以便在跳转到 ULP0 模式之前继续装载代码。
	裸机		
VDDIODDR	Linux	190 µA	VDDIODDR 电源轨保持上电状态，以便能够在 SAMA5D2 器件退出 ULP0 模式后立即重新启动代码执行。
	裸机		
VDDCORE	Linux	250 µA	内核上电且受 512 Hz 时钟驱动。
	裸机		
VDDBU	Linux	1.5 µA	通过 SFRBU_PSWBUCTRL 将电源备用区域设置为由 VDDANA 而非 VDDBU 供电，这样可延长电池使用寿命。
	裸机		
VDD3V3	Linux	1.2 mA	-
	裸机		-
VDD3V3LP	Linux	340 µA	-
	裸机		-
VDD5V	Linux	39 mA	内核上电但不受时钟驱动。外设上电且受 512 Hz 时钟驱动。 在 Linux 方面，适当管理外部元件（以太网 PHY 的待机模式）可以降低功耗。
	裸机	67 mA	

表 6-6 给出了应用程序处于 ULP1 模式时每个电源轨上的不同功耗。

对于 Linux，使用以下命令行参数选择 ULP1 模式：

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

并运行以下命令：

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

表 6-6. 应用程序处于 ULP1 模式

电源轨	应用程序	功耗	备注
VDD_1V35	Linux	11 mA	DDR 电源保持供应，以便在跳转到 ULP1 模式之前继续装载代码。
	裸机		
VDDIODDR	Linux	190 µA	VDDIODDR 轨保持上电状态，以便能够在 SAMA5D2 器件退出 ULP1 模式后立即重新启动代码执行。
	裸机		
VDDCORE	Linux	250 µA	内核上电但不受时钟驱动。
	裸机		
VDDBU	Linux	1.5 µA	通过 SFRBU_PSWBUCTRL 配置电源备用区域，使其由 VDDANA 而非 VDDBU 供电，从而延长电池寿命。
	裸机		
VDD3V3	Linux	1.2 mA	-
	裸机		-
VDD3V3LP	Linux	340 µA	-
	裸机		-
VDD5V	Linux	39 mA	内核和外设均上电但未受时钟驱动。 在 Linux 方面，适当管理外部元件（以太网 PHY 的待机模式）可以降低功耗。
	裸机	67 mA	

表 6-7 给出了应用程序处于 BSR 模式时每个电源轨上的不同功耗。

对于 Linux，使用以下命令行参数选择 BSR 模式：

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

并运行以下命令：

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

表 6-7. 应用程序处于备用自刷新（BSR）模式

电源轨	应用程序	功耗	备注
VDD_1V35	Linux	11 mA	DDR 电源保持供应，以便在跳转到 BSR 模式之前继续将代码装载到 DDR 中。
	裸机		

.....（续）

电源轨	应用程序	功耗	备注
VDDIODDR	Linux	190 µA	VDDIODDR 电源轨保持上电状态，以便能够在主电源施加到 SAMA5D2 后立即重启。
	裸机		
VDD5V	Linux	52 mA	只有 SAMA5D2 器件的备用区域上电，DDR 处于 BSR 模式，PMIC 上电。
	裸机		
VDDBU	Linux	3.8 µA	只有 VDDBU 为备用区域供电。
	裸机		
VDDCORE、VDD3V3 和 VDD3V3LP	Linux	0	备用和 BSR 模式下不提供这些电源输入。
	裸机		

无论裸机应用程序还是 Linux 应用程序，只有 VDDBU 和 VDD_1V35 上电（VDDIODDR 来自 VDD_1V35），因此这两个应用程序的功耗完全相同。

7. 结论

在任何现代系统中，功耗方面都不容忽视。优化应用程序功耗性能的一种方法是使能系统可以设置为低功耗模式的时间段。SAMA5D2 器件具有多种低功耗模式，这些低功耗模式具有特定的功耗水平和相关的暂停/唤醒时间，可满足这种需求。

本应用笔记介绍了这些特定配置的低功耗模式（带有 DDR3L 存储器的 SAMA5D2 Xplained 板）。请注意，使用不同的存储器类型可以获得明显不同的结果。

下表给出了两种存储器类型和三种低功耗模式的功耗估算结果。如存储器数据手册中所述，给出的值是在 85°C 下得到的。预计 25°C 时的值将降低 3 至 5 倍。

表 7-1. 功耗估算

SAMA5D2 低功耗模式(1)		存储器(3)		总功耗(2)
模式	SAMA5D2	LPDDR2	DDR3L	
ULP0 模式	3.1 mW	3 mW	-	6.1 mW
		-	16 mW	19.1 mW
ULP1 模式	2.9 mW	3 mW	-	5.9 mW
		-	16 mW	18.9 mW
BSR 模式	0	3 mW	-	3 mW
		-	16 mW	16 mW

注：

1. 所有值均取自 SAMA5D2 Series 数据手册，并且是在 85°C 下得到的。
2. 总功耗包括 DDR 电源 + VDDCORE。实际总功耗取决于电源的效率。
3. 假设在 85°C 下，DDR3L 存储器的自刷新功耗约为 16 mW，LPDDR2 存储器的自刷新功耗约为 3 mW。这些是编写本应用笔记时可用的数据手册值。

该表主要说明：

- 对于配备 DDR3L 存储器的系统，当使用 BSR 模式而不是 ULP0/1 时，在功耗方面获得的优势极少；
- 对于配备 LPDDR2 存储器的系统，使用 BSR 模式可节省约 50%的功耗。

最后，本应用笔记说明了在 ULP1 模式下，Linux 等复杂操作系统（保存和恢复完整的上下文，超出需要）和简单裸机应用程序之间的暂停和唤醒时间存在明显差异。因此，需要对复杂的操作系统进行特定的改进才能充分利用 ULP1 的完整性能。

8. 附录 A. 用于时间测量的 Linux 代码补丁

这是除 Linux4SAM 5.7 之外需要另外应用的补丁，作用是测量暂停和唤醒时间。过程主要包括切换 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"; /*与 pwm0 冲突*/
+        status = "disabled"; /*与 pwm0 冲突*/

         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
+
+/*
+ *熄灭绿色 led
+ */
+    .macro turn_off_led
+    ldr    gpio, .gpioc
+    mov    tmp2, #32
+    str    tmp2, [gpio, #ATMEL_PIO_PB5_SODR]
+    .endm
+
+/*
+ *点亮绿色 led
+ */
+    .macro turn_on_led
+    ldr    gpio, .gpioc
+    mov    tmp2, #32
+    str    tmp2, [gpio, #ATMEL_PIO_PB5_CODR]
+    .endm

 /*
  *等待主时钟就绪（在切换主时钟源之后）
@@ -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]
+
     /*激活自刷新模式*/
     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

     /*使能晶振*/
```

```
@@ -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. 版本历史

9.1 版本 A——2018 年 12 月

第一版。

Microchip 网站

Microchip 网站 http://www.microchip.com/为客户提供在线支持。客户可通过该网站方便地获取文件和信息。只要使用常用的互联网浏览器即可访问，网站提供以下信息：

- **产品支持**——数据手册和勘误表、应用笔记和示例程序、设计资源、用户指南以及硬件支持文档、最新的软件版本以及归档软件
- **一般技术支持**——常见问题（FAQ）、技术支持请求、在线讨论组以及 Microchip 顾问计划成员名单
- **Microchip 业务**——产品选型和订购指南、最新 Microchip 新闻稿、研讨会和活动安排表、Microchip 销售办事处、代理商以及工厂代表列表

变更通知客户服务

Microchip 的变更通知客户服务有助于客户了解 Microchip 产品的最新信息。注册客户可在他们感兴趣的某个产品系列或开发工具发生变更、更新、发布新版本或勘误表时，收到电子邮件通知。

欲注册，请登录 Microchip 网站 http://www.microchip.com/。在“支持”（Support）下，点击“变更通知客户”（Customer Change Notification）服务后按照注册说明完成注册。

客户支持

Microchip 产品的用户可通过以下渠道获得帮助：

- 代理商或代表
- 当地销售办事处
- 应用工程师（FAE）
- 技术支持

客户应联系其代理商、代表或应用工程师（FAE）寻求支持。当地销售办事处也可为客户提供帮助。本文档后附有销售办事处的联系方式。

也可通过以下网站获得技术支持： http://www.microchip.com/support

Microchip 器件代码保护功能

请注意以下有关 Microchip 器件代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术指标。
- Microchip 确信：在正常使用的情况下，Microchip 系列产品是当今市场上同类产品中最安全的产品之一。
- 目前，仍存在着恶意、甚至是非法破坏代码保护功能的行为。就我们所知，所有这些行为都不是以 Microchip 数据手册中规定的操作规范来使用 Microchip 产品的。这样做的人极可能侵犯了知识产权。
- Microchip 愿意与关心代码完整性的客户合作。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是“牢不可破”的。

代码保护功能处于持续发展中。Microchip 承诺将不断改进产品的代码保护功能。任何试图破坏 Microchip 代码保护功能的行为均可视为违反了《数字器件千年版权法案（Digital Millennium Copyright Act）》。如

果这种行为导致他人在未经授权的情况下，能访问您的软件或其他受版权保护的成果，您有权依据该法案提起诉讼，从而制止这种行为。

法律声明

本出版物中所述的器件应用信息及其他类似内容仅为您提供便利，它们可能由更新之信息所替代。确保应用符合技术规范，是您自身应负的责任。Microchip 对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保，包括但不限于针对其使用情况、质量、性能、适销性或特定用途的适用性的声明或担保。Microchip 对因这些信息及使用这些信息而引起的后果不承担任何责任。如果将 Microchip 器件用于生命维持和/或生命安全应用，一切风险由买方自负。买方同意在由此引发任何一切伤害、索赔、诉讼或费用时，会维护和保障 Microchip 免于承担法律责任，并加以赔偿。除非另外声明，否则在 Microchip 知识产权保护下，不得暗中或以其他方式转让任何许可证。

商标

Microchip 的名称和徽标组合、Microchip 徽标、AnyRate、AVR、AVR 徽标、AVR Freaks、BitCloud、chipKIT、chipKIT 徽标、CryptoMemory、CryptoRF、dsPIC、FlashFlex、flexPWR、Heldo、JukeBlox、KeeLoq、Kleer、LANCheck、LINK MD、maXStylus、maXTouch、MediaLB、megaAVR、MOST、MOST 徽标、MPLAB、OptoLyzer、PIC、picoPower、PICSTART、PIC32 徽标、Prochip Designer、QTouch、SAM-BA、SpyNIC、SST、SST 徽标、SuperFlash、tinyAVR、UNI/O 和 XMEGA 是 Microchip Technology Incorporated 在美国和其他国家或地区的注册商标。

ClockWorks、The Embedded Control Solutions Company、EtherSynch、Hyper Speed Control、HyperLight Load、IntelliMOS、mTouch、Precision Edge 和 Quiet-Wire 为 Microchip Technology Incorporated 在美国的注册商标。

Adjacent Key Suppression、AKS、Analog-for-the-Digital Age、Any Capacitor、AnyIn、AnyOut、BodyCom、CodeGuard、CryptoAuthentication、CryptoAutomotive、CryptoCompanion、CryptoController、dsPICDEM、dsPICDEM.net、Dynamic Average Matching、DAM、ECAN、EtherGREEN、In-Circuit Serial Programming、ICSP、INICnet、Inter-Chip Connectivity、JitterBlocker、KleerNet、KleerNet 徽标、memBrain、Mindi、MiWi、motorBench、MPASM、MPF、MPLAB Certified 徽标、MPLIB、MPLINK、MultiTRAK、NetDetach、Omniscient Code Generation、PICDEM、PICDEM.net、PICkit、PICtail、PowerSmart、PureSilicon、QMatrix、REAL ICE、Ripple Blocker、SAM-ICE、Serial Quad I/O、SMART-I.S.、SQI、SuperSwitcher、SuperSwitcher II、Total Endurance、TSHARC、USBCheck、VariSense、ViewSpan、WiperLock、Wireless DNA 和 ZENA 为 Microchip Technology Incorporated 在美国和其他国家或地区的商标。

SQTP 为 Microchip Technology Inc.在美国的服务标记。

Silicon Storage Technology 为 Microchip Technology Inc.在除美国外的国家或地区的注册商标。

GestIC 是 Microchip Technology Inc.的子公司 Microchip Technology Germany II GmbH & Co. KG 在除美国外的国家或地区的注册商标。

在此提及的所有其他商标均为各持有公司所有。

ISBN： 978-1-5224-4525-8

AMBA、Arm、Arm7、Arm7TDMI、Arm9、Arm11、Artisan、big.LITTLE、Cordio、CoreLink、CoreSight、Cortex、DesignStart、DynamIQ、Jazelle、Keil、Mali、Mbed、Mbed Enabled、NEON、

DNV 认证的质量管理体系

ISO/TS 16949

Microchip 位于美国亚利桑那州 Chandler 和 Tempe 与位于俄勒冈州 Gresham 的全球总部、设计和晶圆生产厂及位于美国加利福尼亚州和印度的设计中心均通过了 ISO/TS-16949:2009 认证。Microchip 的 PIC® MCU 和 dsPIC® DSC、KEELOQ®跳码器件、串行 EEPROM、单片机外设、非易失性存储器及模拟产品严格遵守公司的质量体系流程。此外，Microchip 在开发系统的设计和生产方面的质量体系也已通过了 ISO 9001:2000 认证。

全球销售及服务网点

美洲	亚太地区	亚太地区	欧洲
公司总部 2355 West Chandler Blvd. 钱德勒，亚利桑那州 85224-6199 电话：480-792-7200 传真：480-792-7277 技术支持： http://www.microchip.com/support 网址： www.microchip.com **亚特兰大** 德卢斯，佐治亚州 电话：678-957-9614 传真：678-957-1455 **奥斯汀，德克萨斯州** 电话：512-257-3370 **波士顿** 韦斯特伯鲁，马萨诸塞州 电话：774-760-0087 传真：774-760-0088 **芝加哥** 艾塔斯卡，伊利诺伊州 电话：630-285-0071 传真：630-285-0075 **达拉斯** 阿迪森，德克萨斯州 电话：972-818-7423 传真：972-818-2924 **底特律** 诺维，密歇根州 电话：248-848-4000 **休斯顿，德克萨斯州** 电话：281-894-5983 **印第安纳波利斯** 诺布尔斯维尔，印第安纳州 电话：317-773-8323 传真：317-773-5453 电话：317-536-2380 **洛杉矶** 米慎维荷，加利福尼亚州 电话：949-462-9523 传真：949-462-9608 电话：951-273-7800 **罗利，北卡罗来纳州** 电话：919-844-7510 **纽约，纽约州** 电话：631-435-6000 **圣何塞，加利福尼亚州** 电话：408-735-9110 电话：408-436-4270 **加拿大 - 多伦多** 电话：905-695-1980 传真：905-695-2078	**澳大利亚 - 悉尼** 电话：61-2-9868-6733 **中国 - 北京** 电话：86-10-8569-7000 **中国 - 成都** 电话：86-28-8665-5511 **中国 - 重庆** 电话：86-23-8980-9588 **中国 - 东莞** 电话：86-769-8702-9880 **中国 - 广州** 电话：86-20-8755-8029 **中国 - 杭州** 电话：86-571-8792-8115 **中国 - 香港特别行政区** 电话：852-2943-5100 **中国 - 南京** 电话：86-25-8473-2460 **中国 - 青岛** 电话：86-532-8502-7355 **中国 - 上海** 电话：86-21-3326-8000 **中国 - 沈阳** 电话：86-24-2334-2829 **中国 - 深圳** 电话：86-755-8864-2200 **中国 - 苏州** 电话：86-186-6233-1526 **中国 - 武汉** 电话：86-27-5980-5300 **中国 - 西安** 电话：86-29-8833-7252 **中国 - 厦门** 电话：86-592-2388138 **中国 - 珠海** 电话：86-756-3210040	**印度 - 班加罗尔** 电话：91-80-3090-4444 **印度 - 新德里** 电话：91-11-4160-8631 **印度 - 浦那** 电话：91-20-4121-0141 **日本 - 大阪** 电话：81-6-6152-7160 **日本 - 东京** 电话：81-3-6880- 3770 **韩国 - 大邱** 电话：82-53-744-4301 **韩国 - 首尔** 电话：82-2-554-7200 **马来西亚 - 吉隆坡** 电话：60-3-7651-7906 **马来西亚 - 槟榔屿** 电话：60-4-227-8870 **菲律宾 - 马尼拉** 电话：63-2-634-9065 **新加坡** 电话：65-6334-8870 **台湾地区 - 新竹** 电话：886-3-577-8366 **台湾地区 - 高雄** 电话：886-7-213-7830 **台湾地区 - 台北** 电话：886-2-2508-8600 **泰国 - 曼谷** 电话：66-2-694-1351 **越南 - 胡志明市** 电话：84-28-5448-2100	**奥地利 - 韦尔斯** 电话：43-7242-2244-39 传真：43-7242-2244-393 **丹麦 - 哥本哈根** 电话：45-4450-2828 传真：45-4485-2829 **芬兰 - 埃斯波** 电话：358-9-4520-820 **法国 - 巴黎** 电话：33-1-69-53-63-20 传真：33-1-69-30-90-79 **德国 - 加兴** 电话：49-8931-9700 **德国 - 哈恩** 电话：49-2129-3766400 **德国 - 海尔布隆** 电话：49-7131-72400 **德国 - 卡尔斯鲁厄** 电话：49-721-625370 **德国 - 慕尼黑** 电话：49-89-627-144-0 传真：49-89-627-144-44 **德国 - 罗森海姆** 电话：49-8031-354-560 **以色列 - 若那那市** 电话：972-9-744-7705 **意大利 - 米兰** 电话：39-0331-742611 传真：39-0331-466781 **意大利 - 帕多瓦** 电话：39-049-7625286 **荷兰 - 德卢内市** 电话：31-416-690399 传真：31-416-690340 **挪威 - 特隆赫姆** 电话：47-72884388 **波兰 - 华沙** 电话：48-22-3325737 **罗马尼亚 - 布加勒斯特** 电话：40-21-407-87-50 **西班牙 - 马德里** 电话：34-91-708-08-90 传真：34-91-708-08-91 **瑞典 - 哥德堡** 电话：46-31-704-60-40 **瑞典 - 斯德哥尔摩** 电话：46-8-5090-4654 **英国 - 沃金厄姆** 电话：44-118-921-5800 传真：44-118-921-5820