TI81XX PSP PM SUSPEND RESUME User Guide

TI81XX PSP PM SUSPEND RESUME User Guide

Linux PSP

IMPORTANT
TI81XX refers to TI816X, TI814X and TI813X.

About this manual

This document gives an overview of suspend to memory feature available as part of linux psp package for TI81XX devices.It also details the configuration changes and commands required to use Suspend/Resume feature. This document is applicable to **TI814x and TI813x** devices only.

Read this first:

- This document is applicable for TI814x release 04.01.00.06 onwards, which includes suspend/resume support.
- This document is applicable for TI813x release 04.04.00.01 onwards, which includes suspend/resume support.
- DDR self-refresh support is added from 04.01.00.07 release onwards, for more info refer DDR Self-refresh
- Suspend/resume is **not** supported on PG 1.x revisions of the chip.
- Lowest 1 KB(1024 Bytes from the end) of OCMC RAM is used by Suspend code to execute from OCMC when DDR is placed in Self-refresh mode during suspend/resume.This memory area will be over-written by Suspend code when kernel is loaded, during SRAM initialization so drivers should avoid using the lowest 1KB of OCMC RAM

Acronyms and Definitions

Acronyms and Definitions

Acronym	Expanded
WFI	Wait For Interrupt
DS	Deep Sleep
DDR RAM	Double Data Rate Random Access Memory
OCMC RAM	On Chip Memory Controller Random Access Memory

Introduction

Suspend enables the system to enter low power state to reduce power consumption whenever user desires to do so.
Following are the low power modes defined by hardware:

```
* StandBy
* DeepSleep
```

Only **Standby** mode is currently supported as part of suspend to memory.

DeepSleep mode is an extra power saving state in addition to what is being saved in StandBy mode.
The default suspend mode is StandBy.

Implementation

Suspend enables the user to force the system to enter a low power state whenever user desires to do so:

Suspend Sequence:WFI/StandBy and DeepSleep

1. User triggers suspend through sysfs entry - /sys/power/state
2. Kernel invokes device suspend - drivers execute their suspend calls I2c, USB, MMC, etc.
3. drivers disable/release the clocks they acquired
4. when all the clocks in a domain are disabled the domains is disabled
5. System suspend
    1. check if timer is set
    2. Set Power domains state to OFF based on a flag - PDs with no clk activity will be turned off.
    3. Enter DDR SelfRefresh state - Jump to OCMC
    4. Gate DDR PHY clocks, Bypass DDR PLL
    5. If DeepSleep is enabled - enter Deepsleep
    6. Else WFI (ARM/A8 cpu waits for Interrupt)

Wake-up event:

WFI/StandBy

1. User Triggered - Uart wakeup- User input on Uart wakes up the system, to keep UART live we keep uart clock enabled there by l4s_clock domain alive. This needs a flage "no-console_suspend" in bootargs.
2. Timer expires - User sets a timer through debugfs entries before entering suspend, once the timer expires an interrupt is generated to Arm.

DeepSlep

1. OSC_WAKE switch pressed - User enable the OSC clock by pressing OSC_WAKE

Resume Sequence:

1. Release DDR PLL from Bypass, enable PHY clocks
2. Exit DDR Self refresh
3. Jump to DDR
4. Check whaich power domains did not enter OFF state - if turnoff pds flag was set
5. Kernel Resumes Devices - Drivers resume their operation, enable clocks
6. Control comes back to user

Assumptions

1. All module clocks are disabled by drivers.
2. All PHYs and I/Os will be handled by drivers.

- Refer the respective driver documentation for suspend/resume support

OUT OF SCOPE:

1. Suspend hooks/drivers for controlling slave processors.

- Current suspend implementation does not provide any drivers or hooks for drivers to manage above modules during suspend and resume.

System Status during Suspend

System Status during Suspend

Module	Status in StandBy/WFI	Status in DeepSleep
Clocks	Clock domains not in use are forced to sleep, drivers disable the clocks that were enabled by them	Clock domains not in use are forced to sleep, drivers disable the clocks that were enabled by them
DDR	Self Refresh	Self Refresh
Power Domains	All switchable(on board power switch) Power domains are switched off.(if "turnoff_idle_powerdomains" enabled) ALWON remains on.	All switchable(on board power switch) Power domains are switched off.(if "turnoff_idle_powerdomains" enabled) ALWON remains on.
Voltage Domains	--No Change--	--No Change--
DPLL	--No Change--	Reference clock to all PLLs is gated at the source
Peripherals	All peripherals in ALWON domain remain on as they dont have power switches. UART stays clk enabled, rest all are disabled	All peripherals in ALWON domain remain on as they dont have power switches. UART stays clk enabled, rest all are disabled
Entry	ARM executes WFI	DeepSleep logic is enabled
Wake-up	UART interrupt/key bord and Timer interrupt	OSC_WAKE switch

User Guide

The suspend operation results in the system transitioning to the lowest power state being supported. The drivers implement the `suspend()` function defined in the LDM. When the suspend for the system is asserted, the `suspend()` function is called for all drivers. The drivers release the clocks to reach the desired low power state. The actual transition to suspend is implemented in the function `ti814x_pm_suspend()`.

Configuration

The default EVM configuration (ti8148_evm_defconfig) has "Suspend to RAM and standby" support enabled.

To disable/enable Suspend to RAM support, start the *Linux Kernel Configuration* tool.

```
$ make menuconfig
```

Select *Power management options* from the main menu.

```
...
...
Boot options  --->
CPU Power Management  --->
Floating point emulation  --->
Userspace binary formats  --->
Power management options  --->
[*] Networking support  --->
Device Drivers  --->
...
...
```

Power Management support should be enabled by default. In addition, select *Suspend to RAM and standby* by pressing 'y' or SPACE key. To disable, press 'n' key.

```
-*- Power Management support
[*]   Power Management Debug Support
[ ]     Extra PM attributes in sysfs for low-level debugging/testing
[*]     Verbose Power Management debugging
[*] Suspend to RAM and standby
< > Advanced Power Management Emulation
-*- Run-time PM core functionality
```

Debugging support in Power Management

Start the *Linux Kernel Configuration* tool.

```
$ make menuconfig
```

Select *Power management options* from the main menu.

```
...
...
Floating point emulation  --->
Userspace binary formats  --->
Power management options  --->
[*] Networking support  --->
```

```
Device Drivers  --->
...
...
```

Select *Power Management Debug support* from the next menu.

```
-*- Power Management support
[*]   Power Management Debug Support
[ ]     Extra PM attributes in sysfs for low-level debugging/testing
[*]     Verbose Power Management debugging
[*] Suspend to RAM and standby
< > Advanced Power Management Emulation
-*- Run-time PM core functionality
```

Enabling debug filesystem

Start the *Linux Kernel Configuration* tool.

```
$ make menuconfig
```

Select *Kernel hacking* from the main menu.

```
File systems  --->
Kernel hacking  --->
Security options  --->
-*- Cryptographic API  --->
```

Debug Filesystem is already selected

```
[ ] Enable unused/obsolete exported symbols
-*- Debug Filesystem
[ ] Run 'make headers_check' when building vmlinux
[*] Kernel debugging
```

User Interface

Suspend

Notes:

- You must determine and configure a supported wakeup source before proceeding to suspend otherwise system won't come out of sleep. To use UART as a wake-up source "no_console_suspend" needs to be added to bootargs, refer to Uart Wakeup.
- If MMC/SD card boot mode is used then following kernel config option must be enabled to prevent hang during suspend to memory:

```
Device Drivers-->
    MMC/SD/SDIO Card support -->
          [*]   Assume MMC/SD cards are non-removable (DANGEROUS)
```

- Drivers that do not support suspend may cause DDR corruption during suspen to Memory, Refer to DDR Self-refresh to avoid DDR corruption.
- To **suspend system to RAM** execute following command:

```
 $ echo -n "mem" > /sys/power/state
```

Mount debugfs

- Create a directory to mount *debugfs* or use */sys/kernel/debug*, execute following command

```
$mount -t debugfs debugfs /sys/kernel/debug
```

Turn OFF Power domains

To turn off power domains with power switches if they are idle(no clock is active) during suspend enable following flag:

```
$echo 1 > /sys/kernel/debug/pm_debug/turnoff_idle_powerdomains
```

To disable this:

```
$echo 0 > /sys/kernel/debug/pm_debug/turnoff_idle_powerdomains
```

*NOTE:*If any of the power domains with switches(mentioned above) is active during suspend(any clock(s) running) then the power domain will not be powerd down i.e. the transition to OFF(0) state will not be successful, this will be reported during resume as shown below. e.g.if dsp clock was running then you will see following print:

```
 gem_pwrdm did not enter OFF mode, current state = 3
```

- state 3 stands for ON.

DDR Self-refresh

This section is applicable to 04.01.00.07 and above releases only.
In order to save power and reduce leakage current DDR is put in to self-refresh during suspend and brought out of self-refresh during resume. As part of self-refresh sequence, control of execution moves to OCMC, DDR is put in to self-refresh, EMIF clocks are gated,DMM is configured for No emif access and DDR PLL is pleced in bypass mode.During DDR self-refresh there should not be any access to DDR contents.
Drivers must ensure to suspend all activity before entering suspend(or as part of drivers suspend)and must not make any access to DDR when system is in suspend.

Devices without suspend support

Devices without linux drivers/drivers that do not support suspend may try to access DDR (dma transfers) when DDR in self-refresh, this leads to DDR data corruption and cause system instabilities.Hence the user must ensure to suspend all activities to be suspended/stopped that might result in accessing DDR in turn cause data corruption.

Wake-up sources

- Uart
- Timer

Uart Wakeup

- To use Uart as a wake-up source add "no_console_suspend" to boot args as shown below

```
set bootargs 'console=ttyO0,115200n8 root=/dev/nfs nfsroot=172.24.190.72:/srv/nfs,nolock rw ip=dhcp mem=128M earlyprintk no_console_suspend'
```

To wake up, tap any key on the serial console(teraterm/hyperterminal).

Timer Wakeup

To resume system after certain time use following debugfs entries

```
1. timer_wakeup_seconds
2. timer_wakeup_milliseconds
```

To set these variables debugfs needs to be mounted first,

```
$mount –t debugfs debugfs /sys/kernel/debug
```

eg: To set 5 seconds and 100 milliseconds

```
$echo 5 > /sys/kernel/debug/pm_debug/wakeup_timer_seconds

$echo 100 > /sys/kernel/debug/pm_debug/wakeup_timer_milliseconds
```

To enter suspend:

```
$echo –n "mem" > /sys/power/state
```

Timer sends an interrupt once the count is reached and the system will be resumed.

Limitations

1. Voltage Management during suspend is not available
2. DeepSleep mode is not supported

Future Work

1. Voltage management using regulator framework

Adding suspend/resume support to Drivers

- For successful suspend and maximum power savings Device drivers must provide suspend/resume functionality
- In suspend, driver must disable all the clocks that were enabled during probe(as part of driver code).

These clocks can be enabled in resume if required.

- Suspend fails when driver (with suspend support) fails to suspend a device

PM hooks for drivers

- For device driver:

```
struct device_driver {
        const char              *name;
        struct bus_type         *bus;
         .....................
        const struct dev_pm_ops *pm;
        struct driver_private *p;
};
```

- For bus driver:

```
struct bus_type {
   const char              *name;
   struct bus_attribute    bus_attrs;
   ......................
    const struct dev_pm_ops *pm;
    struct bus_type_private *p;
};
```

similar hooks present in struct device_type , struct class

- Device PM operations structure

```
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 (*suspend_noirq)(struct device *dev);
int (*resume_noirq)(struct device *dev);
...
};
```

NOTE: Suspend() and Resume() are mandatory for successful suspend/resume of the device.All Other operations, such as Prepare() and Complete(), are Optional.
For details such as * the function calling sequence and * what each function is meant for refer to the section "Calling Drivers to Enter and Leave System Sleep States" in Kernel documentation @ <kernel_source>Documentation/power/devices.txt

Example driver code

- **DSP Power Management Driver**

NOTE: Refer to the attached patch for complete code .

```
static int ti814x_dsp_power_probe(struct platform_device *pdev)
{
        struct ti814x_dsp_power_dev *dev;
        int r;
```

```
.........
.........

        dev->ick = clk_get(NULL, "gem_ick");
        if (!dev->ick) {
                r = -ENODEV;
                goto err_free_mem;
        }
        clk_enable(dev->ick);
        dev->fck = clk_get(NULL, "gem_fck");
        if (!dev->fck) {
                r = -ENODEV;
                goto err_free_mem;
        }
        clk_enable(dev->fck);

..........
..........
}


#ifdef CONFIG_SUSPEND
static int ti814x_dsp_power_prepare(struct device *dev)
{
        /*XXX: Prepare DSP for suspended */
        return 0;
}

static void ti814x_dsp_power_complete(struct device *dev)
{
        /*XXX: Do post suspend operations */
}

static int ti814x_dsp_power_suspend(struct device *dev)
{
        struct platform_device *pdev = to_platform_device(dev);
        struct ti814x_dsp_power_dev *dsp_dev =
platform_get_drvdata(pdev);

        clk_disable(dsp_dev->ick);
        clk_disable(dsp_dev->fck);
        return 0;
}

static int ti814x_dsp_power_resume(struct device *dev)
{
        struct platform_device *pdev = to_platform_device(dev);
```

```
        struct ti814x_dsp_power_dev *dsp_dev =
platform_get_drvdata(pdev);

        clk_enable(dsp_dev->ick);
        clk_enable(dsp_dev->fck);

        return 0;
}

static const struct dev_pm_ops ti814x_dsp_power_pm = {
        .prepare        = ti814x_dsp_power_prepare,
        .complete       = ti814x_dsp_power_complete,
        .suspend        = ti814x_dsp_power_suspend,
        .resume         = ti814x_dsp_power_resume,
};

#define ti814x_dsp_power_pm_ops (&ti814x_dsp_power_pm)
#else
#define ti814x_dsp_power_pm_ops NULL
#endif
static struct platform_driver ti814x_dsp_power_driver = {
        .probe          = ti814x_dsp_power_probe,
        .remove         = ti814x_dsp_power_remove,
        .driver         = {
                .name   = "ti814x_dsp_power",
                .owner  = THIS_MODULE,
                .pm     = ti814x_dsp_power_pm_ops,
        },
};

static int __init ti814x_dsp_power_init_driver(void)
{
        return platform_driver_probe(&ti814x_dsp_power_driver,
                                     ti814x_dsp_power_probe);
}
subsys_initcall(ti814x_dsp_power_init_driver);

static void __exit ti814x_dsp_power_exit_driver(void)
{
        platform_driver_unregister(&ti814x_dsp_power_driver);
}
module_exit(ti814x_dsp_power_exit_driver);
```

References

- Refer to the attached patch for complete code .
- I2C driver is a good reference for implementing suspend functionality.Refer to the file : **<kernel_source>drivers/i2c/busses/i2c-davinci.c**.

Revision History

Revision History

Revision	Date	Modified By

Article Sources and Contributors

TI81XX PSP PM SUSPEND RESUME User Guide *Source*: http://processors.wiki.ti.com/index.php?oldid=124078 *Contributors*: RK

Image Sources, Licenses and Contributors

Image:TIBanner.png *Source*: http://processors.wiki.ti.com/index.php?title=File:TIBanner.png *License*: unknown *Contributors*: Nsnehaprabha

License

8. Miscellaneous

a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
b. Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.
c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.
f. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.