Android Devkit Power Management Porting Guide

From Texas Instruments Wiki

Content is no longer maintained and is being kept for reference only!

Contents

About this manual

Content is no longer maintained and is being kept for reference only!

Features

Linux PM Features

States

DVFS

SmartReflex

Idle PM

Android PM Features

Wakelocks

Early Suspend

PM Framworks

Linux Framework

Wake up enabled devices

There are two requirements for a device to be wake up capable. Firstly the wake up event from the device should trigger any of the pins (GPIO bank 0 or any other configured wake up source) connected to the wake up domain of the cpu.

Secondly the device must be configured stay partially awake in suspended state in order to trigger the wake up event.

When the system enters a low power state, each device's driver is asked to suspend the device by putting it into state compatible with the target system state. The wakeup-enabled devices will usually stay partly functional in order to wake the system.

This capability of a device is initialized by bus or device driver code using device_init_wakeup(dev,can_wakeup). This is also controlled through one of the sysfs entry "/sys/devices/.../power/wakeup". The "can_wakeup" flag just records whether the device (and its driver) can physically support wakeup events. When that flag is clear, the sysfs "wakeup" file is empty, and "device_may_wakeup()" returns false.

Usually a wake up capable device will invoke device_init_wakeup(dev,pdata->wakeup) during its .probe routine with the pdata is being its device specific data. While .remove routine the device invoke device_init_wakeup(dev,0).

The device's .suspend function will contain the call to enable_irq_wake for the particular irq line and the .resume function will contain a call to disable_irq_line. So device can be partially functional in suspended state.

EXAMPLE: matrix keypad driver

---

matrix keypad is a simple gpio based keypad

Let's see platform-dependent keypad data of this driver

```
struct matrix_keypad_platform_data {
    .......................
    bool          wakeup;
};
```

It has a field wakeup controls whether the device should be set up as wakeup source.

This field can be set us "true" in the board initialization file of the device.

Now lets explore the ".probe" and ".remove" functions of the driver

```
static int __devinit matrix_keypad_probe(struct platform_device *pdev) {

    const struct matrix_keypad_platform_data *pdata;
    ...............................
    pdata = pdev->dev.platform_data;
    ...............................
    device_init_wakeup(&pdev->dev, pdata->wakeup);


}
```

```
static int __devexit matrix_keypad_remove(struct platform_device *pdev) {

    struct matrix_keypad *keypad = platform_get_drvdata(pdev);
    const struct matrix_keypad_platform_data *pdata = keypad->pdata;
    int i;
    device_init_wakeup(&pdev->dev, 0);
    ...............................
}
```

Also in the suspend routine of the driver, enable_irq_wake is called for the irq lines corresponding to the gpio keys

```
static int matrix_keypad_suspend(struct device *dev)
{
     struct platform_device *pdev = to_platform_device(dev);
     ...............................
     if (device_may_wakeup(&pdev->dev))
             for (i = 0; i < pdata->num_row_gpios; i++)
                     enable_irq_wake(gpio_to_irq(pdata->row_gpios[i]));

     return 0;
}
```

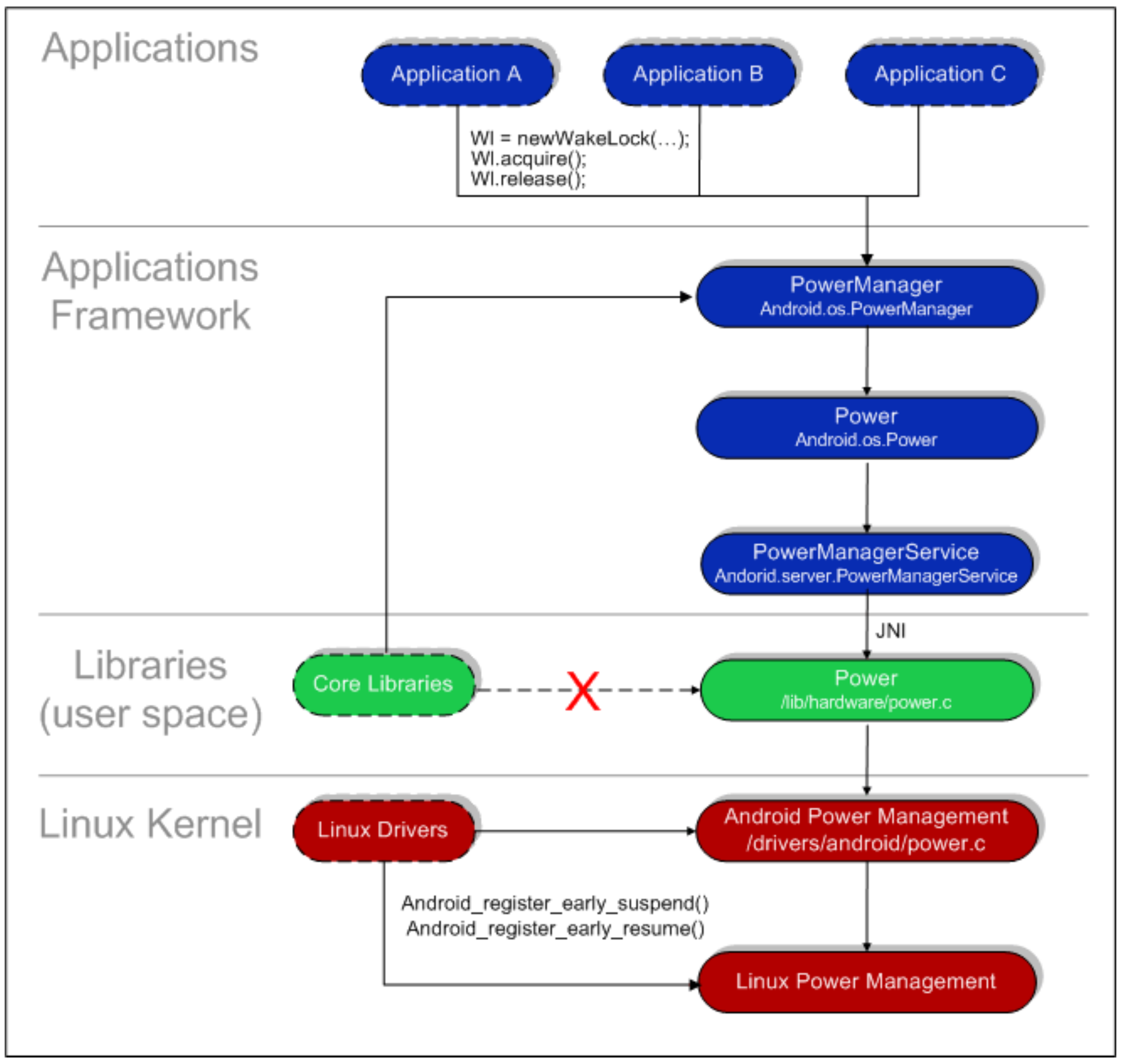
```
static int matrix_keypad_resume(struct device *dev)
{
     struct platform_device *pdev = to_platform_device(dev);
     ...............................
     if (device_may_wakeup(&pdev->dev))
             for (i = 0; i < pdata->num_row_gpios; i++)
                     disable_irq_wake(gpio_to_irq(pdata->row_gpios[i]));
     ...............................
}
```

Android Framework

Android supports its own Power Management (on top of the standard Linux Power Management) designed with the premise that the CPU shouldn't consume power if no applications or services require power.

Android requires that applications and services request CPU resources with "wake locks" through the Android application framework and native Linux libraries. If there are no active wake locks, Android will shut down the CPU.

The image below illustrates the Android power management architecture.


Applications
Application A
Application B
Application C
Wl = newWakeLock(…);
Wl.acquire();
Wl.release();
Applications Framework
PowerManager Android.os.PowerManager
Power Android.os.Power
PowerManagerService Andorid.server.PowerManagerService
JNI
Libraries (user space)
Core Libraries
X
Power /lib/hardware/power.c
Linux Kernel
Linux Drivers
Android Power Management /drivers/android/power.c
Android_register_early_suspend()
Android_register_early_resume()
Linux Power Management

Application Framework

By default, Android will go through different power states depending on the user activity and time. Please look at the diagrams in Android PM Flow. But applications can keep the system in different states by acquiring different kind of wake locks.

The API's provided by PowerManager class will allow applications to do this. For reference read http://developer.android.com/reference/android/os/PowerManager.html
Also, you can take a look at the mobile app development company (http://www.apptraction.com/).

PM Flow

Linux PM Flow

Linux Suspend Resume Sequence

Linux Suspending Process include three parts

1) Freezing Tasks 2) Calling each driver's suspend routine 3) Calling Device specific suspend routine to shut down the power

In omap3evm, you can the suspend the device by writing the desired state string to the sysfs entry, "/sys/power/state"

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

Now let's see what happens after this. Checkout the following files from arago linux-omap3 kernel source code for the reference.

```
linux-omap3/kernel/power/suspend.c
linux-omap3/arch/arm/mach-omap2/pm34xx.c
```

After some checking related with the validity of the requested state, the control jump to the enter_state function of linux-omap3/kernel/power/suspend.c

```
int enter_state(suspend_state_t state)
{
    int error;

    if (!valid_state(state))
        return -ENODEV;

    if (!mutex_trylock(&pm_mutex))
        return -EBUSY;

    printk(KERN_INFO "PM: Syncing filesystems ... ");
    sys_sync();
```

```
	printk("done.\n");

	pr_debug("PM: Preparing system for %s sleep\n", pm_states[state]);
	error = suspend_prepare();
	if (error)
		goto Unlock;

	if (suspend_test(TEST_FREEZER))
		goto Finish;

	pr_debug("PM: Entering %s sleep\n", pm_states[state]);
	error = suspend_devices_and_enter(state);

 Finish:
	pr_debug("PM: Finishing wakeup.\n");
	suspend_finish();
 Unlock:
	mutex_unlock(&pm_mutex);
	return error;
}
```

First task is syncing file system. After this suspend_prepare function is called. This function take care of all device independent steps for suspending the device. Finally suspend_devices_and_enter is called to suspend all the devices and enter into low power state of the system.

suspend_prepare

```
static int suspend_prepare(void)
{
	int error;

	if (!suspend_ops || !suspend_ops->enter)
		return -EPERM;

	pm_prepare_console();

	error = pm_notifier_call_chain(PM_SUSPEND_PREPARE);
	if (error)
		goto Finish;

	error = usermodehelper_disable();
	if (error)
		goto Finish;

	error = suspend_freeze_processes();
	if (!error)
		return 0;

	suspend_thaw_processes();
	usermodehelper_enable();
 Finish:
	pm_notifier_call_chain(PM_POST_SUSPEND);
	pm_restore_console();
	return error;
}
```

This function does all the preparation work before entering into low-power state. It runs suspend notifiers, allocate a console and stop all processes If any step fails in between it restores all the state which was there before. suspend_ops is the platform specific global suspend method table set by the function suspend_set_ops.

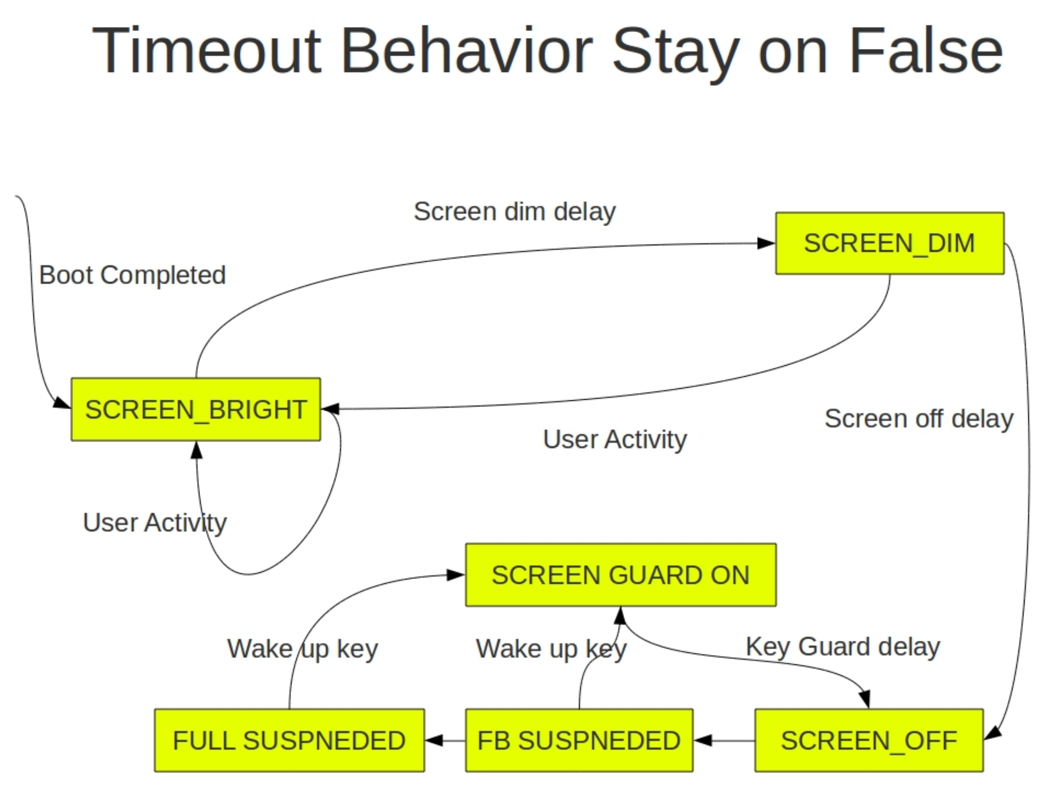
- Direct kernel messages to a newly created console (SUSPEND_CONSOLE
- notifies all the blocks registered to pm blocking chain about the event PM_SUSPEND_PREPARE
- prevents user land processes from being created
- suspend_freeze_processes freezes all processes to RAM

suspend_devices_and_enter

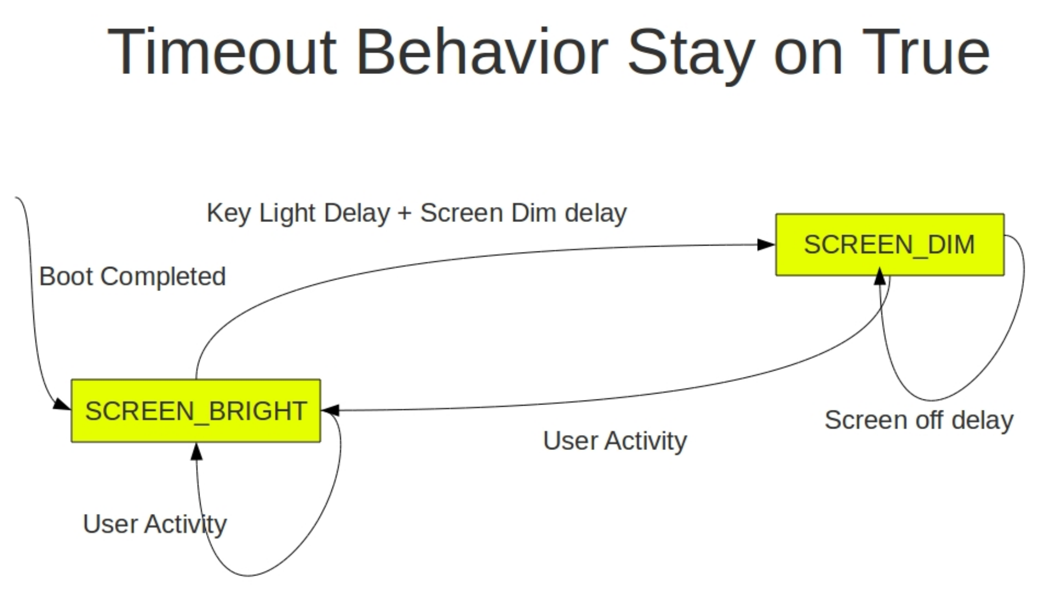
If begin function of the global method table is defined, it will be invoked first. In omap3evm the begin function disable UART interrupts.

AndroidPM Flow

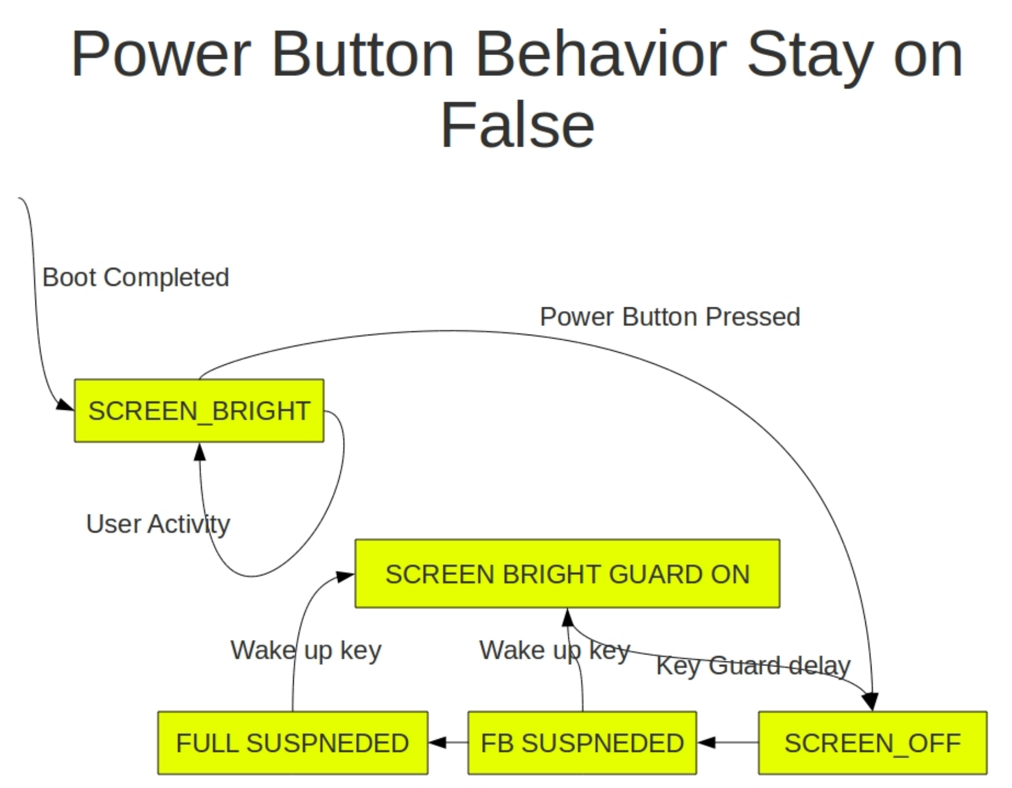
Here are four state diagrams which relates time and user events to various power states.



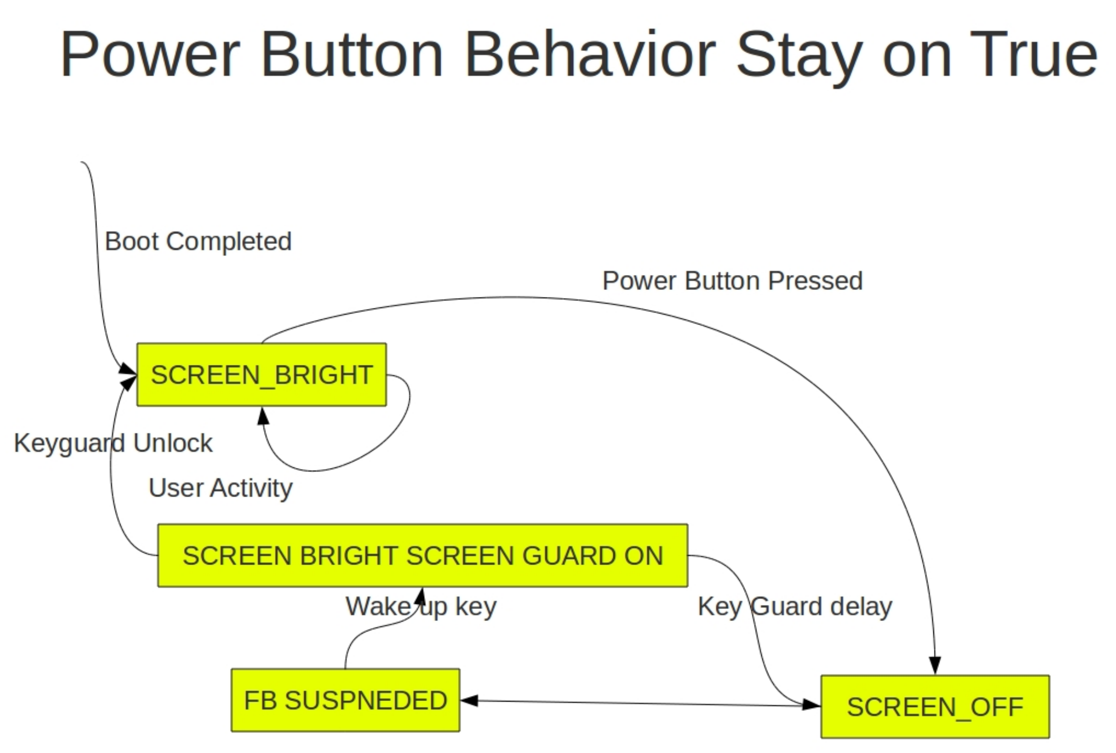
Time out behavior when we have set Stay Awake option is false



Time out behavior when we have set Stay Awake option is true



POWER Button behavior when we have set Stay Awake option is false



POWER Button behavior when we have set Stay Awake option is false

Linux features supported

Exercising linux features

Limitations

Android features supported

Below are the features supported in TI-Android-GingerBread-2.3-DevKit-1.0

AM37X, OMAP35X

- Change of LCD backlights based on Wake Locks and Screen Timeouts (DIM,OFF)
- LCD back light brightness control from Settings Application
- Suspending the device to Memory.The device can be suspended in two ways.

1. Pressing the POWER key on the keypad
2. Allowing the system go to suspend after a screen timeout

- Prevent Suspend based on Wake Locks
- System Resume on Key Press
- CPU Dynamic Voltage and Frequency Scaling (ondemand, performance, powersave and userspace governors)
- CPU Idle States
- Enabling system for hitting retention during idle
- Enabling system for hitting OFF

AM35X

- Suspending the device to Memory.The device can be suspended in two ways.

1. Pressing the POWER key on the keypad
2. Allowing the system go to suspend after a screen timeout

- Prevent Suspend based on Wake Locks
- System Resume on Key Press
- Android Early Suspend Feature for frame buffer driver

Exercising android features

Basic Settings

To go in suspend mode

- Press POWER key on the keypad

To resume from suspend mode

- Press any button on the key pad

To set the Screen timeout to go suspend

- Select Settings->Display-> Screen Timeout
- Select one of the options from the list

To set set the screen always on preventing suspend

- Select Settings-> Applications-> Development-> Stay awake

To set Screen brightness

- Select Settings->Display-> Brightness

Advanced Settings

To Disable Power Management

- Edit init.rc file on the root directory.
- Set the property hw.nopm to true
- This will prevent POWER key suspend
- Select Settings-> Applications-> Development-> Stay awake
- The above will prevent screen timeout suspend

Enabling system for hitting retention during idle

```
#echo 1 > /debug/pm_debug/sleep_while_idle
```

Enabling system for hitting OFF

```
#echo 1 > /debug/pm_debug/enable_off_mode
```

By default sleep_while_idle is set to false and enable_off_mode is set to true

CPU Dynamic Voltage Frequency Scaling settings

Enabling ondemand frequency governor

The ondemand governor enables DVFS(frequency/OPP) transitions based on CPU load.

```
#echo ondemand > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

Enabling performance frequency governor

The performance governor keeps the CPU always at the highest frequency.

```
#echo performance > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

Enabling powersave frequency governor

The powersave governor keeps the CPU always at the lowest frequency.

```
#echo powersave > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

Enabling userspace frequency governor

Once this governor is enabled, DVFS(frequency) transitions will be manually triggered by a userspace application by using the CPUfreq sysfs interface

```
#echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

See all the available operating points

```
#cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies
```

Application can select any of the available frequency from the above

```
#echo  <Desired Frequancy> > /sys/devices/system/cpu/cpu0/cpufreq/ scaling_setspeed
```

Checking CPU IDLE states usage

There are seven power states introduced by CPU Idle

The usage and time count for these different states can be checked via

```
#cat /sys/devices/system/cpu/cpu0/cpuidle/state*/time
```

```
#cat /sys/devices/system/cpu/cpu0/cpuidle/state*/usage
```

Limitations

- In beagle and am35x platforms there is no keypad connected to the main board. So wake up is not possible by pressing keys.To enable power management in am35x platform, set Settings -> Applications -> Development -> Stay Awake to true.

In this case one workaround in am35x platform for wake up is like below.

```
Press ENTER on Serial Port console and then press any key on the keypad to wake up the device.
```

- If the usb device connected to the USB EHCI port when suspending the device, it might not work properly after resume.

So Disconnect any usb device connected to USB EHCI port when suspending the device.

- After pressing the POWER key the device will go to early suspend mode. It will take a while to go to the complete suspend state.

Adding device specific changes

Files to edit POWER HA LAYER hardware/ti/omap3 LIGHTS HA LAYER hardware/ti/omap3/liblights

Limitations