GLOBAL EDGE
Intelligence Of Things

Power Management – Session 1

Zaheer R M

Contents

- Introduction
- Motivation for Power Management
- Power Management Principles
- Power Management Implementation with an example

Introduction

- Power management is the process by which the overall consumption of power by a computer is limited based on user requirements and policy.

- Power management (PM) software is a crucial component in battery-powered systems, such as PDAs and laptops, because it helps conserve power when the system is inactive. As a simple example, power may be conserved by switching off the display when a system is inactive for some time. Conserving power in this manner extends battery life, so one can work more hours before having to recharge the battery

Motivation for Power Management

- Reducing overall power consumption to save cost
- Prolong battery life for portable and embedded systems
- Reduce cooling requirements
- Lower power consumption also means lower heat dissipation, which increases system stability, and less energy use, which saves money and reduces the impact on the environment.

Power Management Principles

- Power consumed is proportional to frequency and square of voltage. Small reductions in voltage can be very significant.
- Keeping devices powered on consumes a lot of power. Cut or reduce power to entire device or idle device portions.
- For proper operation, you should run at one of the recommended operating voltage and frequency combinations (OPPs)
  - Too low voltages can result in data propagation errors and system failure
  - Too high voltages result in excessive power consumption.
  - Hardware timing closure results in a few PROVEN operating
  - performance points (OPPs) which guarantee the system to work properly.

Power Management Implementation

- Before implementing power management, it is important to understand what hardware support is available for saving power. One of the important goals of power management software is to keep all devices in their low power states as much as possible
- A possible approach for implementing power management is first to define a power state transition diagram. This defines several power states for the system and also defines the rules and events governing state transitions.

Power Management Implementation

- As an example, consider a PDA that has the following devices: Intel SA1110 CPU, real-time clock, DRAM, Flash, LCD, front light, UART, audio codec, touchscreen, keys and power button. The Intel SA1110 CPU supports several power-saving features, including frequency scaling, where the core clock frequency can be configured by software. Lowering clock frequency reduces the CPU's power consumption, but at the cost of reduced CPU speed. This CPU also supports several modes of operation:

Power Management Implementation

Run mode: the normal state of operation for the SA1110 when it is executing code. All power supplies are enabled, all clocks are running and every on-chip resource is functional.

Idle mode: allows software to stop the CPU when not in use. In this mode, the CPU clock is stopped, representing some savings in power. All other on-chip resources are active. When an interrupt occurs, the CPU is reactivated.

Sleep mode: offers the greatest power savings and consequently the lowest level of available functionality. In this mode, power is switched off to the majority of the processor. Some preprogrammed event, such as a power button press, wakes up the CPU from this mode

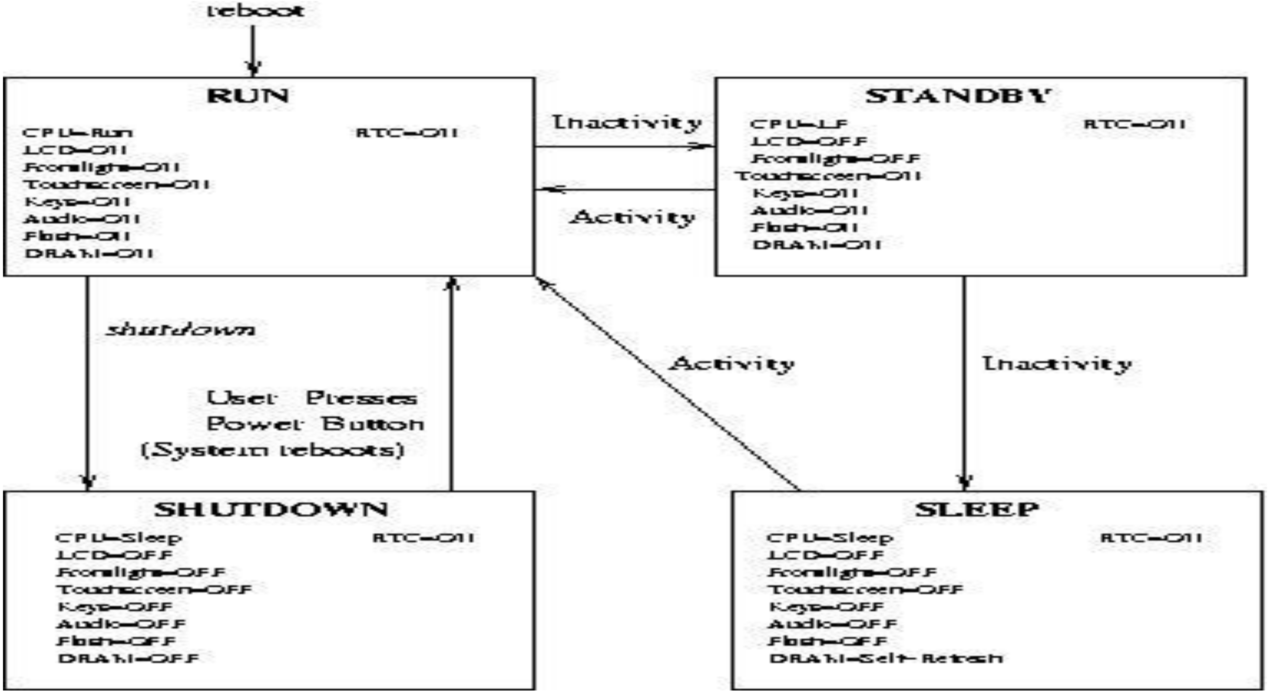
Power Management Implementation

As you can see, software is responsible for transitioning the CPU either to idle mode or sleep mode.

In such a PDA, DRAM cells normally are refreshed periodically by the memory controller logic present inside the CPU. In sleep state, however, the majority of the CPU is shut off, which results in DRAM cells not being refreshed, which in turn leads to loss of data in DRAM. To avoid this loss, most DRAMs support a mode called self-refresh wherein the DRAM itself takes care of refreshing its cells. In such cases, software can put DRAM in its self-refresh mode by writing to a few control registers before transitioning the CPU to its sleep mode, thereby preserving the DRAM contents.

The top power-hungry devices in this PDA can be the CPU, DRAM and display back light. Hence, they should be kept in their low power states as much as possible.

Power Management Implementation



reboot

RUN

CPU=Run
LCD=On
Frontlight=On
Touchscreen=On
Keys=On
Audio=On
Flash=On
DRAM=On
RTC=On

Inactivity

Activity

STANDBY

CPU=LF
LCD=OFF
Frontlight=OFF
Touchscreen=On
Keys=On
Audio=On
Flash=On
DRAM=On
RTC=On

shutdown

User Presses
Power Button
(System reboots)

Activity

Inactivity

SHUTDOWN

CPU=Sleep
LCD=OFF
Frontlight=OFF
Touchscreen=OFF
Keys=OFF
Audio=OFF
Flash=OFF
DRAM=OFF
RTC=On

SLEEP

CPU=Sleep
LCD=OFF
Frontlight=OFF
Touchscreen=OFF
Keys=OFF
Audio=OFF
Flash=OFF
DRAM=Self Refresh
RTC=On

Power Management Implementation

Figure shows a possible power state transition diagram for this PDA. Here is a brief description of the power states:

Run state: system falls into this default state when it reboots. Power consumption is maximum in this state, as all devices are turned on or active.

Standby state: system falls into this state due to inactivity. LCD and display back light are turned off, and CPU clock speed is reduced to save some power.

Power Management Implementation

Sleep state: system falls into this state due to continued inactivity. Power is conserved aggressively by putting the CPU in sleep mode, which in turn powers off most devices. DRAM, however, is put in its self-refresh mode to preserve the machine state (system and application text/data loaded in memory) while the system is sleeping. The system awakens from sleep state when a preprogrammed event occurs. When it wakes up, it transitions to the run state and machine state is restored.

Shutdown state: system falls into this state when the shutdown command is issued. The system reboots when it exits from this state. This means it is not necessary to preserve the machine state in DRAM, and hence DRAM can be powered off. The shutdown state then represents the lowest power consumption state of all.

Power Management Implementation

Implementing power management in any system is a complex task. Here's how to manage your system's transitions from normal run state to power-saving modes.

The real-time clock is kept on in all power states to retain system time.

It is clear from this diagram that detecting inactivity and putting the devices in their low power states forms the heart of power management software.

Session 2 Contents

- Power Management Standards
- Power Management Techniques

Power Management – Session 2

Zaheer R M

Contents

- Power Management Standards
- Power Management Techniques

Power Management Standards

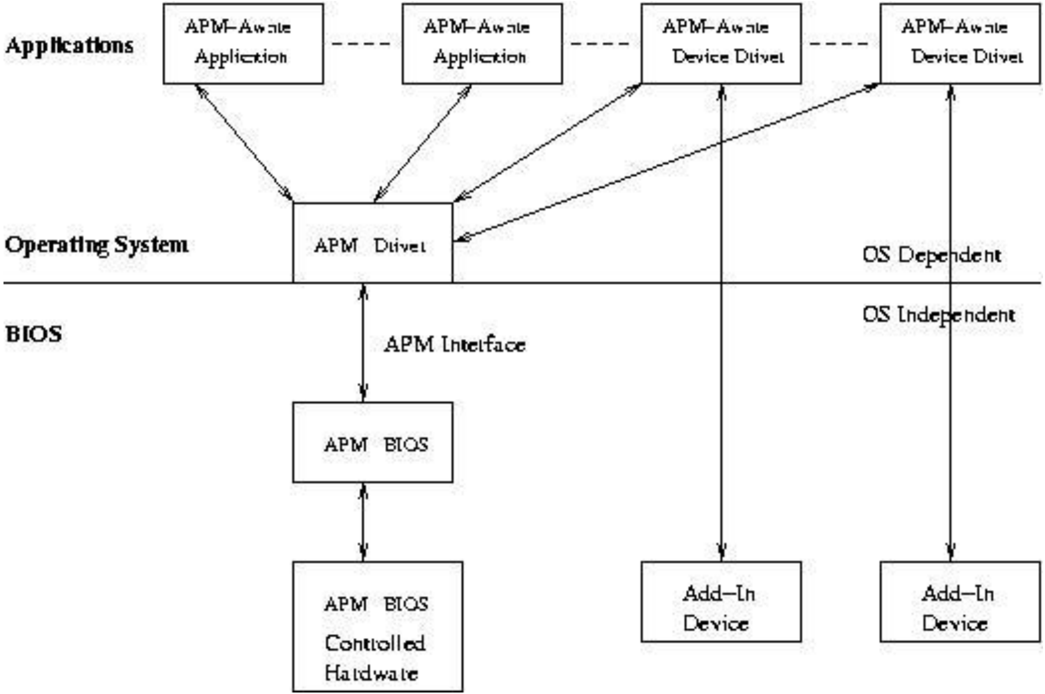
Two popular power management standards :

- Advanced Power Management (APM)
- Advanced Configuration and Power Interface (ACPI)

APM

- Control resides in BIOS
- Uses activity timeouts to determine when to power down a device
- BIOS rarely used in Embedded Systems
- Makes power management decisions without informing OS or Application
- No Knowledge of add-in cards or new devices

Power Management Standards - APM



Power Management Standards - APM

Figure shows the APM model. The important components of this model are:

APM BIOS: software interface to the motherboard and its power managed devices and components. It is the lowest level of PM software in the system.

APM driver: implements APM in a particular operating system.

APM-aware device drivers and applications: APM driver interacts with them for all PM events.

Power Management Standards - APM

APM BIOS detects and reports various PM events, including low battery, power status change, system standby, system resume and so on.

The APM driver uses polling function calls to the APM BIOS to gather information about PM events. It then processes these events in association with APM-aware drivers and applications.

The APM driver in Linux exposes two interfaces for an application's use

first /proc/apm, holds information on the system power. It specifies whether the system is running on A/C power or battery. If running on battery, it also specifies the battery charge and time left for the battery to drain completely.

Power Management Standards - APM

The second interface,

/dev/apm-bios, allows applications to know of and participate in PM events. It also allows them to initiate power state transitions by themselves, by issuing suitable ioctl calls. Read calls issued against this file will block until the next PM event occurs. When the read call returns, it carries information regarding the PM event about to occur.

Some of the applications that have opened /dev/apm_bios may be running with root privileges. Such applications are special to the APM driver. For some of the events, such as standby or suspend transition, APM driver informs all applications that have opened /dev/apm_bios about the event. In addition, it waits for approval from those few applications running with root privileges before the system actually is put in standby/suspend state. This approval comes when applications issue suitable ioctls.

Power Management Standards - APM

The following ioctls normally are supported:

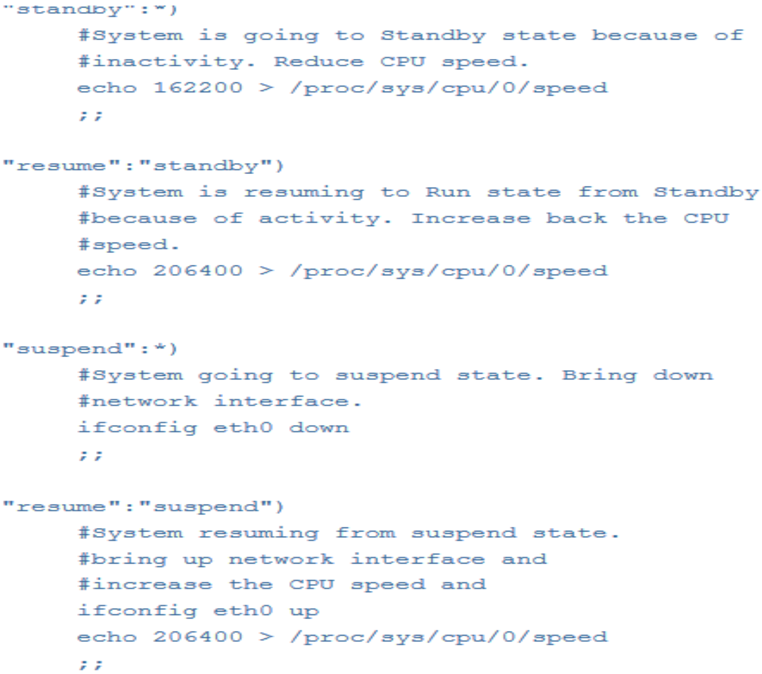
APM_IOC_STANDBY: puts the system in standby state.

APM_IOC_SUSPEND: puts the system in suspend state.

APM also comes with two user-space utilities. The apm command interacts with the APM subsystem in the kernel. Depending on the arguments passed, it can display system power status, or it can be used to initiate system standby/suspend transition. The apmd dæmon reports and processes various PM events and logs all PM events to /var/log/messages. In addition to logging, apmd also can take some specific actions for each type of PM event. These actions are specified in a script file (usually called apmd_proxy). This script file is invoked by the apmd dæmon with one or two arguments indicating the PM event about to occur.

Power Management Standards - APM

The following is a sample script file:



```
"standby":*)
        #System is going to Standby state because of
        #inactivity. Reduce CPU speed.
        echo 162200 > /proc/sys/cpu/0/speed
        ;;

"resume":"standby")
        #System is resuming to Run state from Standby
        #because of activity. Increase back the CPU
        #speed.
        echo 206400 > /proc/sys/cpu/0/speed
        ;;

"suspend":*)
        #System going to suspend state. Bring down
        #network interface.
        ifconfig eth0 down
        ;;

"resume":"suspend")
        #System resuming from suspend state.
        #bring up network interface and
        #increase the CPU speed and
        ifconfig eth0 up
        echo 206400 > /proc/sys/cpu/0/speed
        ;;
```

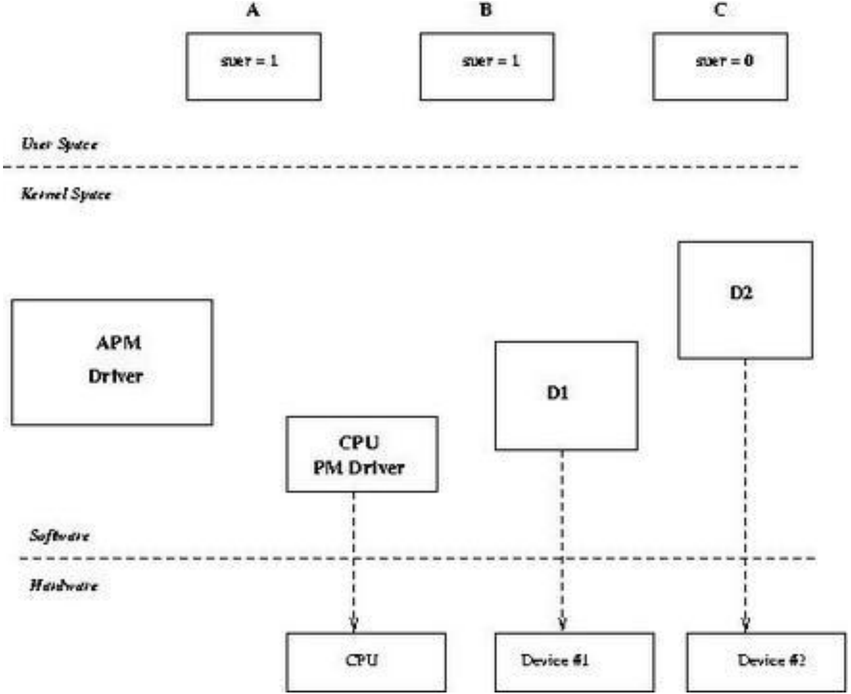
Power Management Standards - APM

Implementing power management in any system is a complex task. Here's how to manage your system's transitions from normal run state to power-saving modes

Example Power State Transition

Some of the complexities involved in power state transitions can be understood by taking the example of a state transition involving both drivers and applications. Assume the system has two drivers, D1 and D2, registered with PM and three applications, A, B and C, also participating in PM (by way of opening /dev/apm_bios). Of the three applications, A and B are running with superuser privileges, and C is not. Figure 8 depicts this scenario.

Power Management Standards - APM



Power Management Standards - APM

Figure shows Example Power State Transition

Now, with this setup, let's consider a case where the system wants to transition to sleep state from run state. The sequence of steps involved in this case begins with informing applications A, B and C about the pending transition to sleep state. This allows them to take whatever actions are necessary for this transition. Also, because A and B have superuser privileges, we have to wait for them to say okay to this sleep transition before it proceeds any further.

When A and B are done with whatever work they need to perform before the system transitions to a sleep state, they give the go-ahead to the APM driver. Now, the APM driver is ready to put the system into sleep state. It sends a PM_SUSPEND message to D1 and D2. D1 and D2 put their respective devices into sleep state and say okay to APM. After D1 and D2 are finished processing this transition, APM informs the CPU PM driver to put the CPU in sleep state. At this stage, the system transition to sleep state is complete.

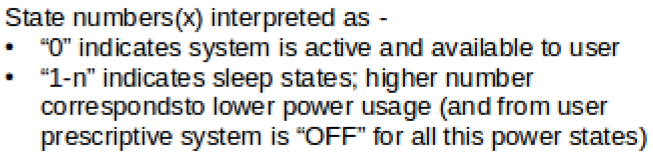
Power Management Standards - ACPI

ACPI

- Control divided between BIOS and OS
- Decisions managed through the OS
- Power Management is global in system unlike APM where it is hidden
- ACPI defines power management states and required functionality at multiple levels of system

ACPI System States

- Global state - Gx
- System state - Sx
- Processor/CPU state -Cx
- Devices state -Dx



State numbers(x) interpreted as -

- “0” indicates system is active and available to user
- “1-n” indicates sleep states; higher number correspondsto lower power usage (and from user prescriptive system is “OFF” for all this power states)

Power Management Standards - ACPI

Advanced Configuration and Power Interface (**ACPI**) specification provides an open standard that the operating systems can use for computer hardware discovery, configuration, power management, and monitoring

First released in December 1996, ACPI defines platform-independent interfaces for hardware discovery, configuration, power management, and monitoring, and is designed to replace Advanced Power Management (APM), the MultiProcessor Specification, and the Plug and Play BIOS (PnP) Specification

ACPI brings the power management under the control of the operating system, as opposed to the previous BIOS-centric system that relied on platform-specific firmware to determine power management and configuration policies

The specification is central to *Operating System-directed configuration and Power Management* (OSPM), a system implementing ACPI which removes device management responsibilities from legacy firmware interfaces

Power Management Standards - ACPI

Intel, Microsoft and Toshiba originally developed the standard, while HP and Phoenix also participated later

In October 2013, the original developers of the ACPI standard agreed to transfer all assets to the UEFI Forum, in which all future development will take place

The latest version of the standard is "Revision 6.0", which was published by the UEFI Forum in April 2015

Once an OSPM-compatible operating system activates ACPI, it takes exclusive control of all aspects of power management and device configuration. The OSPM implementation must expose an ACPI-compatible environment to device drivers, which exposes certain system, device and processor states

Power Management Standards - ACPI

Global states

The ACPI specification defines the following four global "Gx" states and six sleep "Sx" states for an ACPI-compliant computer-system:

G0 (S0), Working: "Awaymode" is a subset of S0, where monitor is off but background tasks are running.

G1, Sleeping: Divided into four states, S1 through S4:

S1, Power on Suspend (POS): Processor caches are flushed, and the CPU(s) stops executing instructions. The power to the CPU(s) and RAM is maintained. Devices that do not indicate they must remain on may be powered off.

S2: CPU powered off. Dirty cache is flushed to RAM.

S3, commonly referred to as Standby, Sleep, or Suspend to RAM (STR): RAM remains powered.

S4, Hibernation or Suspend to Disk: All content of the main memory is saved to non-volatile memory such as a hard drive, and the system is powered down.

Power Management Standards - ACPI

G2 (S5), Soft Off: G2/S5 is almost the same as G3 Mechanical Off, except that the power supply unit (PSU) still supplies power, at a minimum, to the power button to allow return to S0. A full reboot is required. No previous content is retained. Other components may remain powered so the computer can "wake" on input from the keyboard, clock, modem, LAN, or USB device.

G3, Mechanical Off: The computer's power has been totally removed via a mechanical switch (as on the rear of a PSU). The power cord can be removed and the system is safe for disassembly (typically, only the real-time clock continues to run using its own small battery).

Power Management Standards - ACPI

Device states

The device states D0–D3 are device dependent:

D0 or Fully On is the operating state.

D1 and D2 are intermediate power-states whose definition varies by device.

D3: The D3 state is further divided into D3 Hot (has aux power), and D3 Cold (no power provided):

- Hot: A device can assert power management requests to transition to higher power states.

- Cold or Off has the device powered off and unresponsive to its bus.

Power Management Standards - ACPI

Processor states

The CPU power states C0–C3 are defined as follows:

C0 is the operating state.

C1 (often known as Halt) is a state where the processor is not executing instructions, but can return to an executing state essentially instantaneously. All ACPI-conformant processors must support this power state. Some processors, such as the Pentium 4, also support an Enhanced C1 state (C1E or Enhanced Halt State) for lower power consumption.[25]

C2 (often known as Stop-Clock) is a state where the processor maintains all software-visible state, but may take longer to wake up. This processor state is optional.

C3 (often known as Sleep) is a state where the processor does not need to keep its cache coherent, but maintains other state. Some processors have variations on the C3 state (Deep Sleep, Deeper Sleep, etc.) that differ in how long it takes to wake the processor. This processor state is optional.

Power Management Standards - ACPI

Performance states

While a device or processor operates (D0 and C0, respectively), it can be in one of several power-performance states. These states are implementation-dependent. Though, P0 is always the highest-performance state; with P1 to P*n* being successively lower-performance states, up to an implementation-specific limit of *n* no greater than 16.

P-states have become known as SpeedStep in Intel processors, as PowerNow! or Cool'n'Quiet in AMD processors, and as PowerSaver in VIA processors.

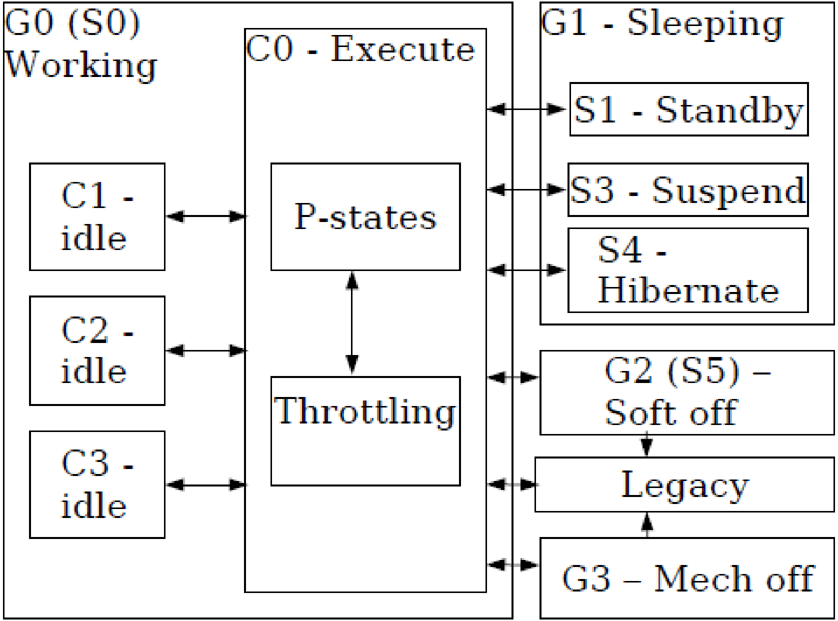
P0 max power and frequency

P1 less than *P0*, voltage and frequency scaled

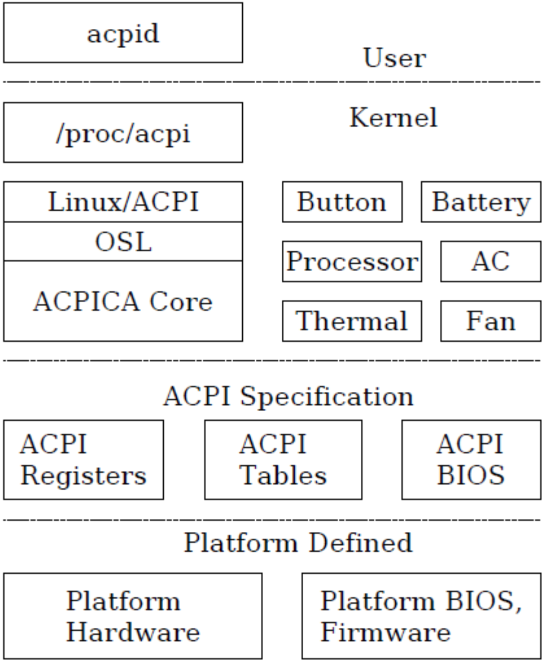
P2 less than *P1*, voltage and frequency scaled ....

Pn less than *P(n-1)*, voltage and frequency scaled

Power Management Standards - ACPI Global, CPU, and Sleep states



Power Management Standards – ACPI Implementation Architecture



Power Management Techniques

Processor optimizes system power consumption by employing two main techniques: managing active system power consumption and managing standby system power consumption.

- **Active power consumption management** – Used to optimize the power a system draws while processing data to achieve useful results. Here, three basic methods exist:
  - Dynamic voltage and frequency scaling (DVFS)
  - Adaptive voltage scaling (AVS)
  - Dynamic power switching (DPS)
- **Static power consumption management** – Keeps an idle system in a power-efficient state until further processing is required. Managed by a technique known as static leakage management (SLM), static power consumption can result in several low-power modes, from standby to a deep sleep mode, which mimics the power-off state but has faster wake-up latency.

Power Management Techniques - Dynamic voltage and frequency scaling (DVFS)

What is DVFS?

Dynamic Voltage and Frequency scaling is a framework to change the frequency and/or operating voltage of a processor(s) based on system performance requirements at the given point of time.

Frequency Scaling

Frequency scaling is achieved using CPUFreq framework.

What is CPUFreq?

CPUfreq is a linux kernel framework that monitors the performance requirements of a processor(s) and takes decisions to increase or decrease operating frequency in order to save power and/or reduce leakage power.

Power Management Techniques - Dynamic voltage and frequency scaling (DVFS)

CpuFreq Architecture

CPUFreq consists two elements

The Governor - that makes decisions

The Driver - acts based on the decisions made by the governor

Policy and Governor

Policy

Policy is set of rules the system is bound by such as min and max frequency for each cpu, availability of a frequency. Policy for a cpu is created during the CPUFreq framework initialization based on the frequency table.

Power Management Techniques - Dynamic voltage and frequency scaling (DVFS)

Frequency Table

Frequency table consists of available frequencies for all cup's in the system. Frequency table is generated/populated based on the Operating Performance Point(OPP) list for each cpu.

OPP List

Operating performance Point(OPP) is a tuple consisting a frequency value and voltage required to run at the frequency. OPP table contains OPPs with a cpu/device name they are applicable to and an availability flag. OPP information of each device is added to OPP list.

Governor

Governor continuously monitors the system performance requirements and when the requirement to change the frequency arises it checks the current cpu policy for frequency limits and requests the driver to change the frequency.

Power Management Techniques - Dynamic voltage and frequency scaling (DVFS)

Driver

Multiple drivers can exist in the kernel but there will be only one scaling driver which performs actions based on governors decision. When the governor request the driver to change the frequency to a target value, driver checks the frequency availability in the OPP list. If its found it scales the device to new frequency.

Power Management Techniques - Dynamic voltage and frequency scaling (DVFS)

Governors provided by CPUFreq

The following governors are available in CPUFreq frame work

Performance

Powersave

Ondemand

Userspace

Conservative

Power Management Techniques - Dynamic voltage and frequency scaling (DVFS)

Voltage Scaling

Voltage scaling is achieved using voltage layer and regulator framework(driver). When the CPUFreq driver scales the device frequency, voltage corresponding to the frequency(target_voltage) is looked-up in the opp list. The device scale function requests the voltage layer to scale the device voltage to the target_voltage.

Power Management Techniques - Adaptive voltage scaling (AVS)

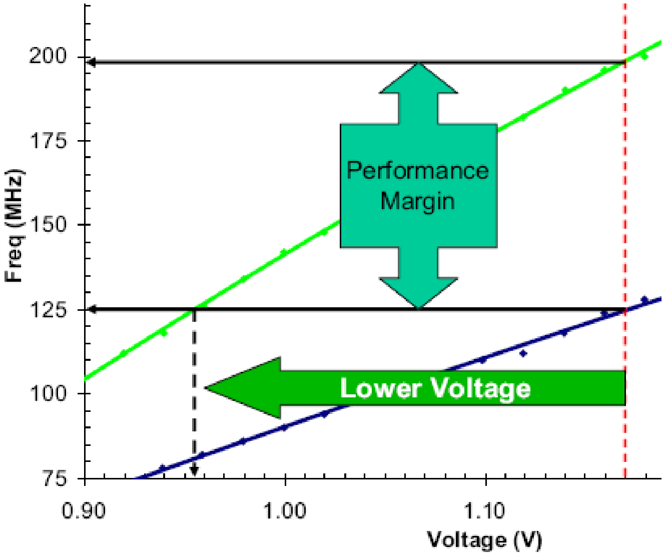
Active PM Technique

SmartReflex (trademark of Texas Instruments)

- Silicon manufacturing process yields a distribution of performance capability
- For a given frequency requirement:
  - Hot/strong/fast devices can meet this at a lower voltage
  - Cold/weak/slow devices needs higher voltage
- Simple system will set the higher voltage for operating all devices
- **Smarter system will adapt operating voltage per device!**

Green line: “hot” device

Blue line: “cold” device



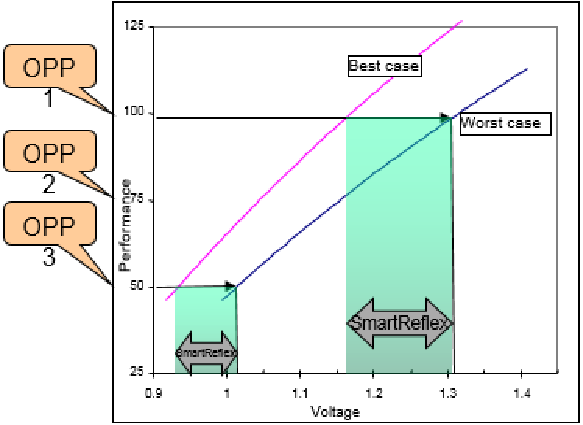
Power Management Techniques - DVFS vs AVS

DVFS:

- Select OPP from available OPPs based on MIPS requirement in scenario
- Lower frequency requirement -> lower voltage
- All die treated the same for a particular application scenario

	OPP	ARM MHz	Vdd1
AM335x	OPPTurbo	720	1.26
	120	600	1.2
	100	500	1.1
	50	275	0.95

OPP	L3 MHz	Vdd2
100	200	1.1
50	100	0.95



AVS (SmartReflex)

- **After** selection of OPP, scale voltage based on device-specific properties (process capability, temperature)
- Faster (also stronger/warmer) process -> lower voltage
- Each die treated differently

DVFS vs AVS

DVFS:

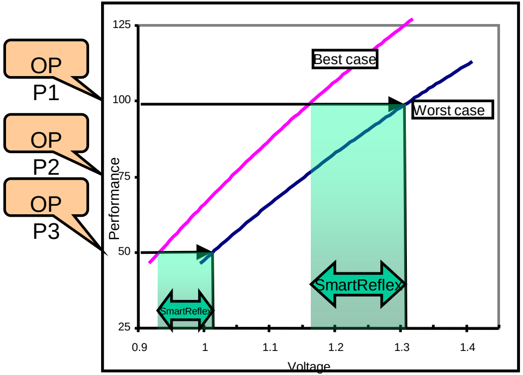
Select OPP from available OPPs based on MIPS requirement in scenario

Lower frequency requirement -> lower voltage

All die treated the same for a particular application scenario

	OPP	ARM MHz	Vdd1
AM335x	OPPTurbo	720	1.26
	120	600	1.2
	100	500	1.1
	50	275	0.95

OPP	L3 MHz	Vdd2
100	200	1.1
50	100	0.95



AVS (SmartReflex)

- **After** selection of OPP, scale voltage based on device-specific properties (process capability, temperature)
- Faster (also stronger/warmer) process -> lower voltage
- Each die treated differently

Power Management Techniques - Adaptive voltage scaling (AVS)

Dedicated, on-chip SmartReflex technology-based hardware implements a feedback loop that does not require processor intervention, which dynamically optimizes voltage levels to account for variations in process results, temperature and silicon degradation (Figure 2, next page). Software sets up the SmartReflex technology-based hardware for each OPP. The hardware control algorithm then sends commands to external voltage regulators over an I2C bus, lowering the appropriate regulators' outputs in incremental steps until the processor just exceeds target frequency requirements. **SmartReflex technology-based AVS works in concert with DVFS to provide additional power savings beyond those obtainable by DVFS alone.**

Power Management Techniques - Adaptive voltage scaling (AVS)

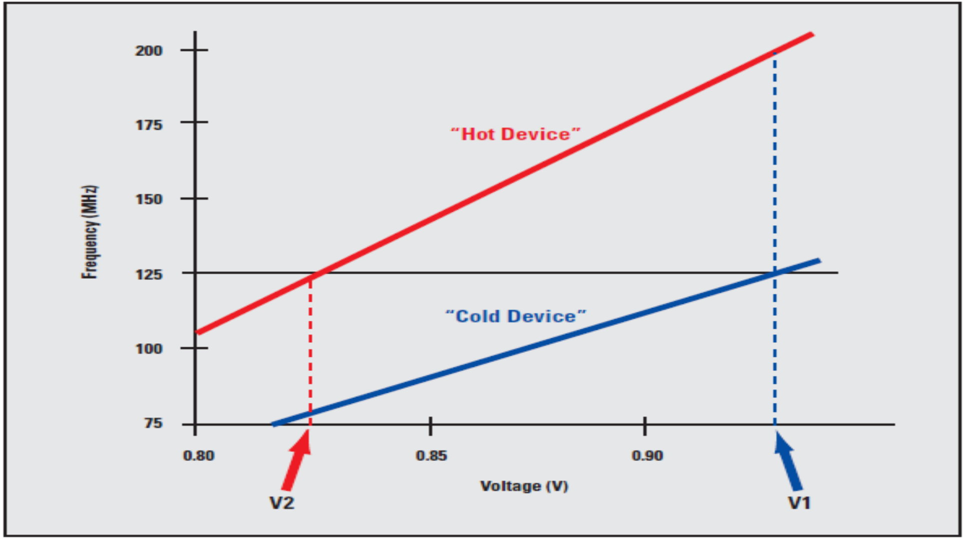


Figure 2. Processors come off the fab line with different performance characteristics. Here, a "cold" OMAP35x device needs 0.94 V to run at 125 MHz, whereas a "hot" device needs just 0.83 V to run at the same frequency. Adaptive voltage scaling (AVS) includes a SmartReflex technology-based feedback loop that can lower supply voltages from the safe 0.95-V level (set initially under the DVFS policy) to the frequency levels individual devices need to run specific processing tasks.

Power Management Techniques - Adaptive voltage scaling (AVS)

For example, you might start by designing in a voltage that fits all cases; for 125 MHz, that is 0.95 V(V1 in Figure 2). If a hot OMAP35x device with SmartReflex technology is inserted in the system, however, the on-chip feedback mechanism can automatically lower the voltage of the ARM to 0.85 V or less, yet have the same 125-MHz performance (V2 in Figure 2).

Power Management Techniques - Dynamic power switching (DPS)

Active PM Technique

Allows certain modules to idle when not being used.

Example use cases: during MP3 playback, we could idle CPSW (among others)

These modules operate in a high performance state to complete its tasks as fast as possible, then dynamically switch to a low-power state ("retention" or "off state")

Context save/restore may be necessary if memory is lost → additional overhead

Acceptable wakeup latencies on the order of microseconds

Software support: In Linux, handled by the "Runtime PM" framework. This is driver-local suspend/resume, managed entirely by individual device drivers operation independently. Drivers can relinquish clocks, resulting in portions of the device in a "clock stop" state

Power Management Techniques - Dynamic power switching (DPS)

Active PM Technique

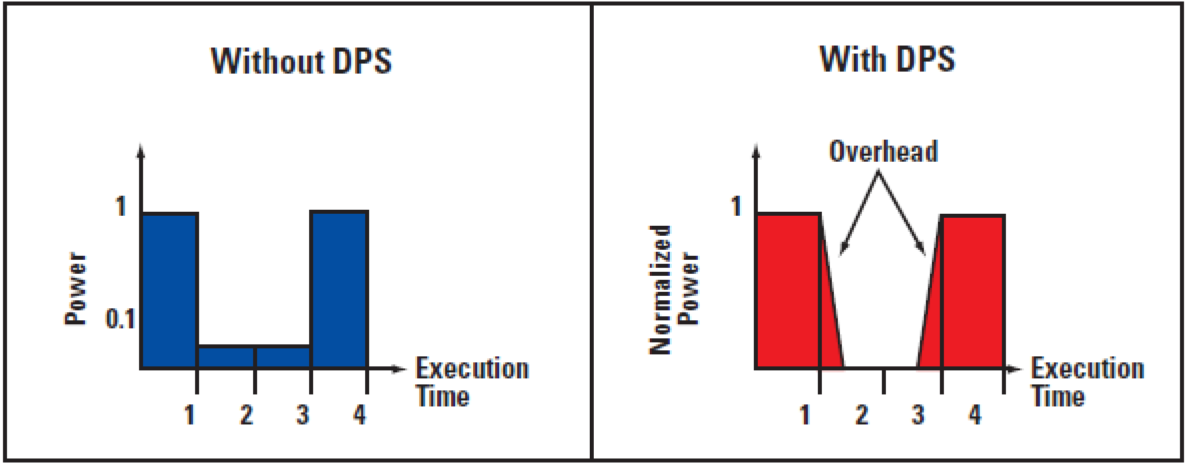


Figure 3. In dynamic power switching (DPS), parts of an OMAP35x device switch to a low-power state after completing their tasks.

Power Management Techniques - Static/Passive Power Management (SLM)

Idle PM Technique

Device switches into low-power system modes automatically or under user requests when no application is running

- Example: Media player shuts off display and enters standby after 10-seconds of on-time with no processing and no user input
- Typically whole device is in some sort of standby mode
- Acceptable latencies on the order of milliseconds. Deeper sleep state == longer wakeup latency.

Devices have multiple sleep states (or C-states)

AM335x supports two C-states

- C1 – MPU WFI
- C2 – MPU WFI + DDR Self Refresh (lowest power)

Power Management Techniques - Static/Passive Power Management (SLM)

Idle PM Technique

Software support:

- CPUIdle algorithm automatically chooses from available sleep states based upon the inactivity period and current HW state. Useful for power savings during shorter periods of inactivity.

- Suspend/Resume support available for long inactivity periods (Deep sleep state)

Session 3

- Power Management Case Study
- Power management in Linux

Power Management – Session 3

Zaheer R M

Contents

- ReCap
- Power Management Case Study
- Linux Power Management

Recap

Why power management?

End Users

- ✓ Smaller, sleeker products
- ✓ Cooler device
- ✓ More exciting, power hungry features like multimedia, streaming video, etc.
- ✓ Improved experience (longer battery life)
- ✓ Lower cost

ReCap - Power Management Principles

1) Power consumed is proportional to frequency and square of voltage.

- Small reductions in voltage can be very significant
- Applicable PM Techniques:

  Dynamic Voltage and Frequency Scaling (DVFS)

  SmartReflex (aka Adaptive Voltage Scaling or AVS)

2) Keeping devices powered on consumes a lot of power. Cut or reduce power to entire device or idle device portions.

- Applicable PM Techniques:

  Dynamic Power Switching (DPS)

  - Turn off what you're not using!
  - At a granular, per-module level
  - Called "Runtime PM" in Linux

ReCap - Power Management Principles

Static Leakage Management (SLM)

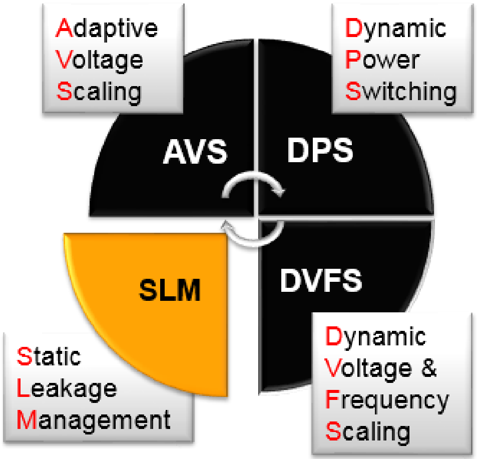
- System level idle and/or suspend

3) For proper operation, you should run at one of the recommended operating voltage and frequency combinations (OPPs)

- Too low voltages can result in data propagation errors and system failure
- Too high voltages result in excessive power consumption
- Hardware timing closure results in a few PROVEN operating performance points (OPPs) which guarantee the system to work properly.

ReCap - Power Management Techniques: Basics

PM Techniques can be categorized as “ ” or “Idle”:



- Active PM Technique

Performed while device is on and in use. Involves all systems components: HW modules, device drivers, OS & apps. Techniques: AVS (SmartReflex), DPS & DVFS.

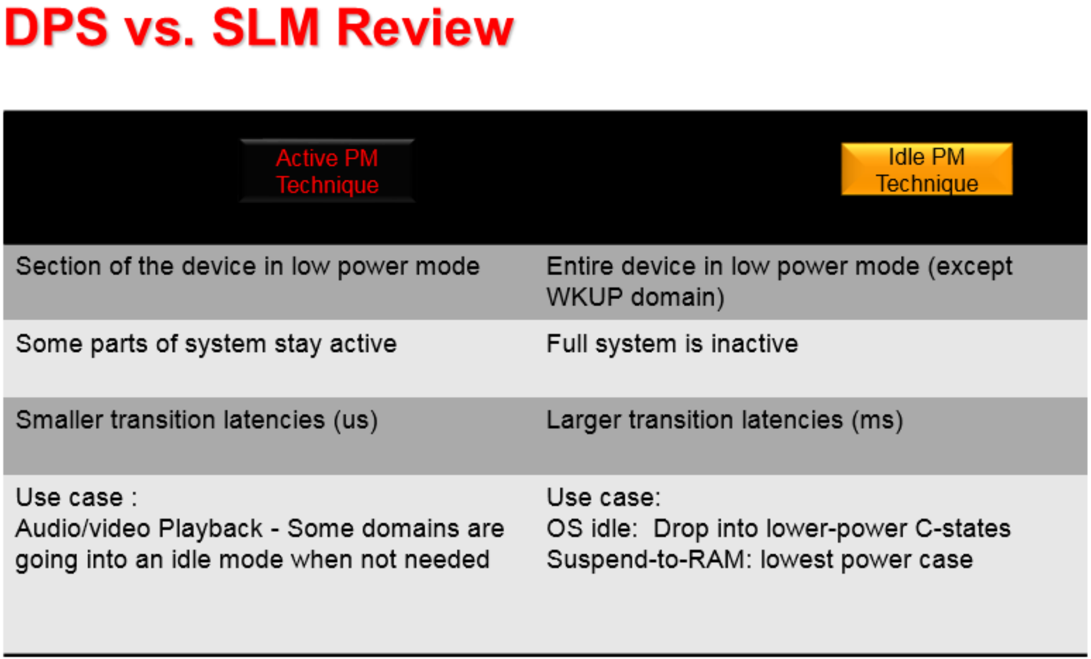
Idle PM Technique

Entire device is idled. Various idle states selected based on trade-off of power consumption and wakeup latency.

Techniques: SLM

➔ Power Management is centralized and is handled by the PRCM (Power, Reset and Clock Management) module.

ReCap - Power Management Techniques: Basics



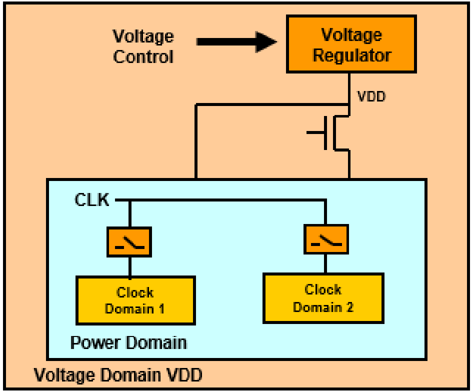
DPS vs. SLM Review

Active PM Technique	Idle PM Technique
Section of the device in low power mode	Entire device in low power mode (except WKUP domain)
Some parts of system stay active	Full system is inactive
Smaller transition latencies (us)	Larger transition latencies (ms)
Use case : Audio/video Playback - Some domains are going into an idle mode when not needed	Use case: OS idle: Drop into lower-power C-states Suspend-to-RAM: lowest power case

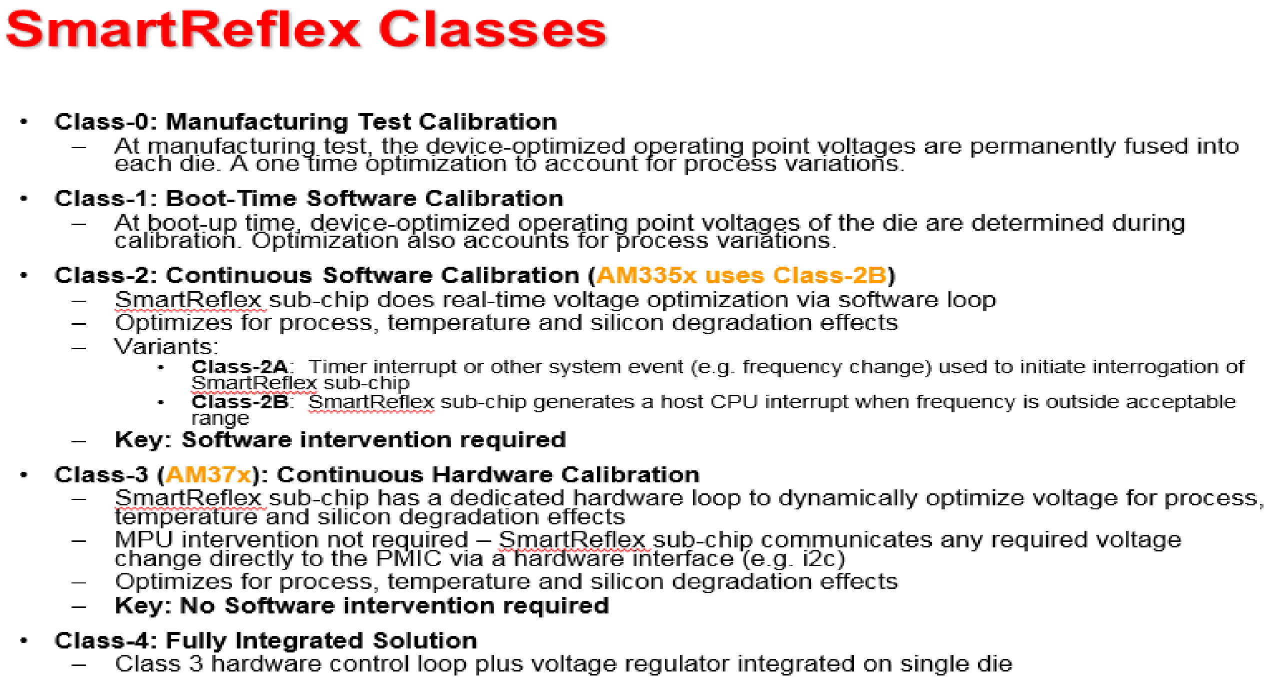
ReCap - PM Architecture Basics

Devices are partitioned into the following architectural blocks:

- Clock Domain:
  - One or more modules fed with the same clock
  - Clock has independent gating control
- Power Domain:
  - One or more modules fed with the same power rail
  - Power rail controlled by independent power switches
- Voltage Domain
  - Group of modules supplied by the same voltage regulator (embedded or external)
  - "VDDn" sometimes referred to in documentation as a "power rail"
- Domains hierarchy is generally:
  - Voltage domains contain one or more Power Domain
  - Power domains contain one or more Clock Domain
- Control of this infrastructure is done by the Power Reset and Clock Manager (PRCM), which is split in two entities:
  - PRM: Handles voltage, power, reset, wake-up management, system clock source control and clock generation
  - CM: Handles the clock generation, distribution and management



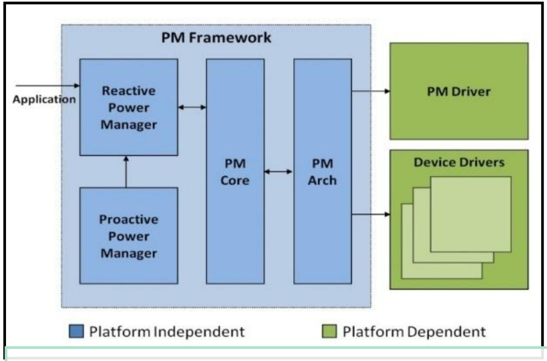
ReCap - Adaptive Voltage Scaling (SmartReflex Classes)



SmartReflex Classes

- **Class-0: Manufacturing Test Calibration**
  - At manufacturing test, the device-optimized operating point voltages are permanently fused into each die. A one time optimization to account for process variations.
- **Class-1: Boot-Time Software Calibration**
  - At boot-up time, device-optimized operating point voltages of the die are determined during calibration. Optimization also accounts for process variations.
- **Class-2: Continuous Software Calibration (AM335x uses Class-2B)**
  - SmartReflex sub-chip does real-time voltage optimization via software loop
  - Optimizes for process, temperature and silicon degradation effects
  - Variants:
    - **Class-2A**: Timer interrupt or other system event (e.g. frequency change) used to initiate interrogation of SmartReflex sub-chip
    - **Class-2B**: SmartReflex sub-chip generates a host CPU interrupt when frequency is outside acceptable range
  - **Key: Software intervention required**
- **Class-3 (AM37x): Continuous Hardware Calibration**
  - SmartReflex sub-chip has a dedicated hardware loop to dynamically optimize voltage for process, temperature and silicon degradation effects
  - MPU intervention not required – SmartReflex sub-chip communicates any required voltage change directly to the PMIC via a hardware interface (e.g. i2c)
  - Optimizes for process, temperature and silicon degradation effects
  - **Key: No Software intervention required**
- **Class-4: Fully Integrated Solution**
  - Class 3 hardware control loop plus voltage regulator integrated on single die

Power Management Case Study



Features

- Dynamic Voltage and Frequency Scaling
- Processor Power Save Modes
- Clock Domain Control
- Device Power Management
- Pro-active & Re-active Mechanisms
- Wake on RTC/LAN/USB/GPIO

PM Arch

- HAL Interfaces to PM Core
- HAL Interfaces to Platform Specific Modules

PM Core

- System Power State Management
- Device Power State Management

Re-active Power Manager

- Interfaces to User/ Application
- State Change Notification

Pro-active Power Manager

- System Load based Power Save Control
- State Change Notification

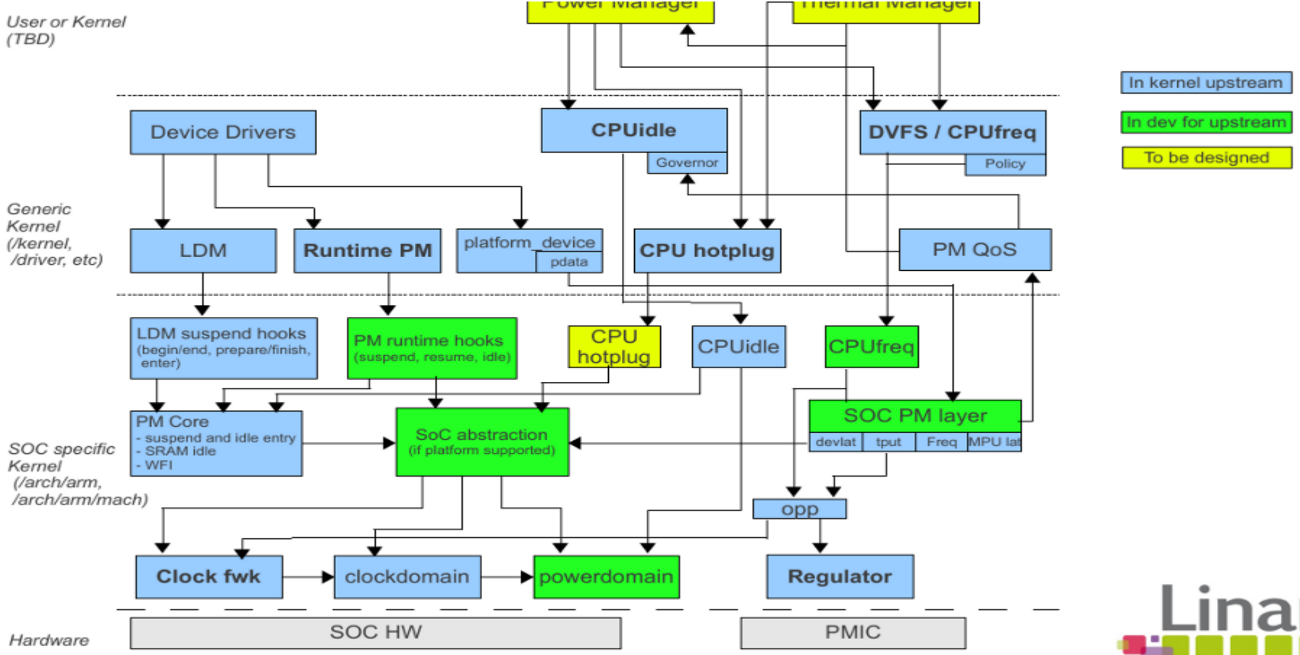
PM Driver

- Interfaces for Hardware Control
- Hardware specific Power Management

Device Drivers

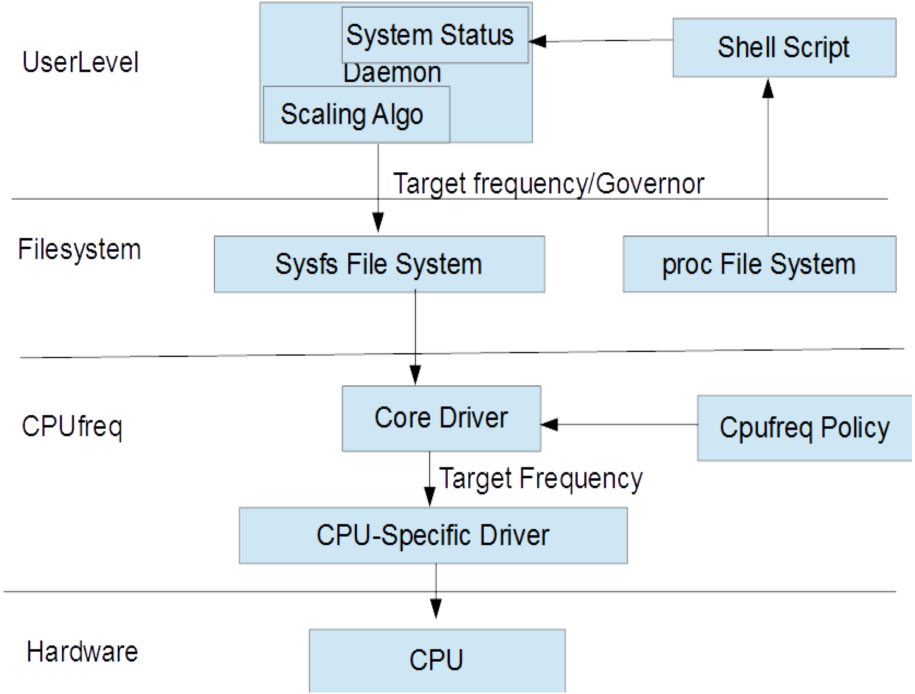
- Interfaces for Peripheral Control
- Device specific Power Management

Linux Power Management



Linaro

Linux Power Management



Linux Power Management – PM Building Blocks

Several power management « building blocks »

- Suspend and resume
- CPUidle
- Runtime power management
- Frequency and voltage scaling
- Applications

Independent « building blocks » that can be improved gradually during development

Session 4

- Linux Power Management Continuation
- Linux Test Project (Power Management Test Cases)

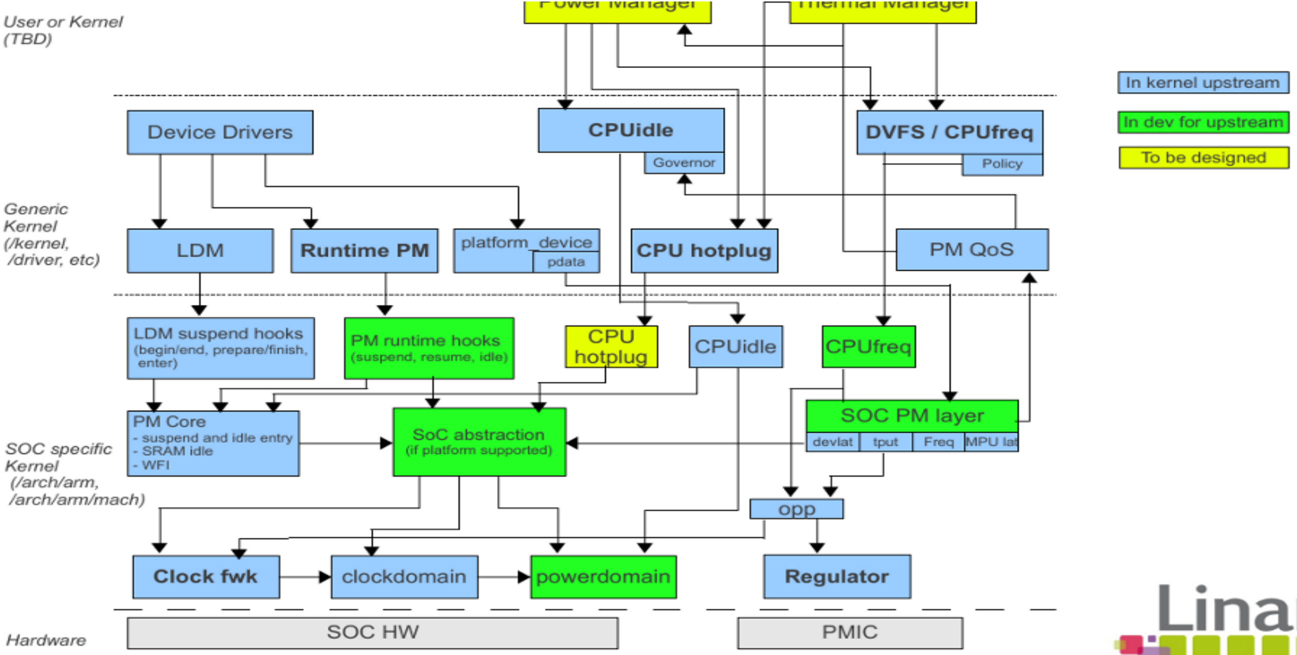
Power Management – Session 4

Zaheer R M

Contents

- Linux Power Management Continuation
  - PM Framework
  - Example ***PM Implementation with IEGD***
  - **Suspend to RAM**
  - **Resume/Wake-up**
- Linux Test Project (Power Management Test Cases)

Linux Power Management Continuation



Linux Power Management

Some examples of hardware PM include:

- CPU: HLT, Stop Clock, SpeedStep®
- Display: Blanking, dimming, Energy Star, power-save mode.
- Graphics Controller: Powering down (completely off or into an intermediate power state).
- Hard Drive/CD-ROM: Spin down
- Network/NIC: Wake on LAN
- USB: Device power state transitions (mouse, USB drives, speakers); Wake on access (mouse movement, R/W access, input signal)
- BIOS: ACPI S3/S4 platform behaviors, video repost, CPU PM settings (states enabled, CPU fan throttling), and platform settings (e.g., thermal low/high watermarks, chassis fan throttling).

Linux Power Management

Linux supports two implementations of firmware power management: Advanced Power Management (APM) and Advanced Configuration and Power Interface (ACPI).

APM is an older power management technology focused on basic system and OS power management, with much of the power management policy in the BIOS. It requires an APM driver as the interface between the BIOS and the Linux operating system. With this method, power management events could pass between the BIOS and the OS. Devices can be notified of these events and respond appropriately. APM is still supported in the kernel, but most newer platforms support ACPI, the successor to APM.

Linux Power Management - **PM Framework**

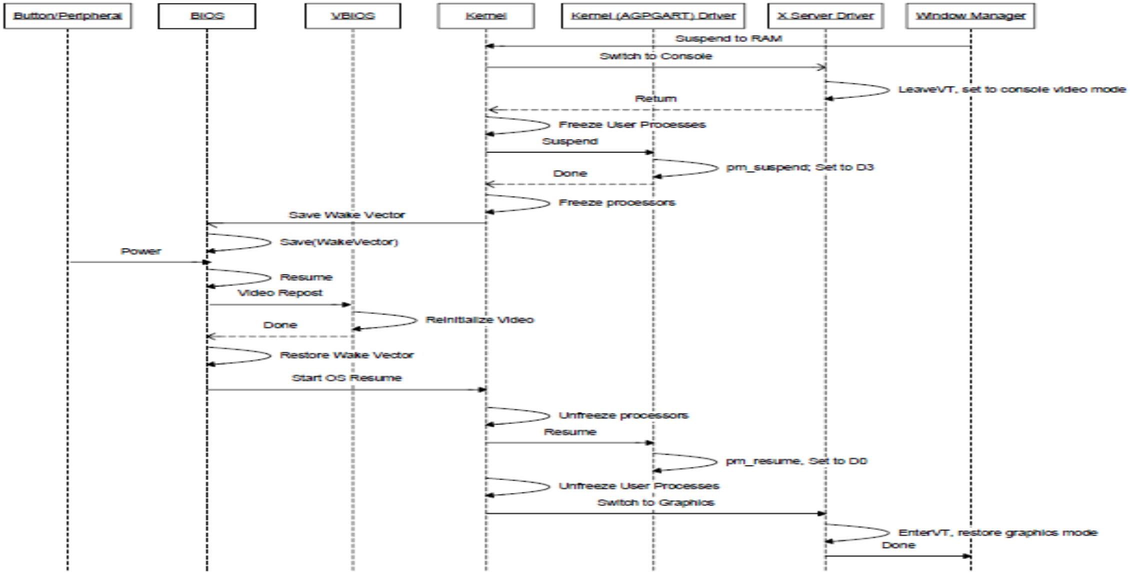
The Linux kernel has supported ACPI for a long time. ACPI provides greater power management state support, platform configuration support, and allows for platform independence and OS control over power management events. In those ways, it differs from APM, where BIOS was mostly in charge of power management policies. ACPI supports power management, but also thermal (fans), buttons/lids (events), CPU “C” states, and power source (battery, AC), etc.

Linux Power Management – Example ***PM Implementation with IEGD***

PM software manages state transitions along with device drivers and applications. Device drivers are responsible for saving device states before putting them into their low power states and then restoring the device state when the system becomes active. Generally, applications are not involved in power management state transitions. A few specialized applications that deal directly with some devices may want handle state transitions. This section explains the sequence of steps the Intel® Embedded Graphics Driver (IEGD) for Linux goes through during power management state transitions.

Linux Power Management – Example ***PM Implementation with IEGD***

Figure 1. Power Management Sequence Diagram



The following two sections correspond to the sequence diagram and describe Suspend to RAM and Resume/Wake-up.

Linux Power Management – Example ***PM Implementation with IEGD***

In Figure 1. Power Management Sequence Diagram, the sequence can be read by first understanding the top level blocks and then considering the events that occur over time from top to bottom. The blocks are described as:

- Button/Peripheral: Describes input from a physical device such as a button, key-press or mouse movement.
- BIOS: System BIOS responsible for platform configuration.
- VBIOS: Video BIOS in charge of pre-OS graphics initialization.
- Kernel: The Linux Kernel.
- Kernel (AGPGART) Driver: IEGD IKM (Intel Kernel Module) responsible for graphics device initialization and resource allocation.
- X-Server Driver: The X Windows System graphics driver for an Intel graphics device.
- Window Manager: The Graphical User Interface.

Linux Power Management – **Suspend to RAM**

When the system suspends or resumes from a power state such as S3, there is handoff among several variables (BIOS, VBIOS, kernel, drivers, windows manager and X) to put the system back to its previous state.

Examine what happens when the user puts the system into S3 suspend mode, i.e., Suspend to RAM.

The Suspend to RAM action starts when an “event” is triggered in the platform telling the OS that a power management event has occurred. The event can be in the form of a button press, such as holding down the power (or sleep) button on a laptop to trigger a sleep action or it can be through user interaction, such as clicking on a window manager option to put the platform into a suspended state. Certainly, one can execute a script at the terminal or have a timer setup to automatically put the system in a low power state as well. In the example above, assume the user chose a window manager option to signal the GUI to change to a lower power state: Suspend to RAM.

Linux Power Management – **Suspend to RAM**

When this action occurs, the window manager tells the kernel to trigger an S3 power management operation. How this happens is distribution- and implementation-specific. Newer Linux distributions have a platform management infrastructure that provides a protocol for the OS to communicate system or platform events between the software components and the kernel. By using such an infrastructure, power management events can be broadcast to other parts of the system through the kernel. Figure 1 shows that the window manager, after detecting the user-generated event, triggers the kernel to go to a lower power state. The kernel then starts the suspend to RAM procedure.

Linux Power Management – **Suspend to RAM**

Alternatively, the suspend action triggers a hardware interrupt that is handled by the embedded controller. The embedded controller triggers a register in the southbridge chipset, setting a flag that a general purpose event has just occurred. The OS notices this, and checks the DSDT for what must happen next. Normally, this just calls a notification event that returns to user space via /proc/acpi/events; the user space software determines what happens next. Finally, the string "mem" is written to /sys/power/state.

Linux Power Management – **Suspend to RAM**

The X display must switch to console before going into a lower power state. With ACPI, the Linux kernel first switches to console mode and then continues with its power management code. This is in contrast to APM where suspend/resume often occurs through messages that were trapped from within the X-Server. When X switches to the console, it calls a LeaveVT (“leave virtual terminal”) function that allows the switch to occur. This is where IEGD saves the graphics states.

Linux Power Management – **Suspend to RAM**

The process saves all graphics registers and state information so that on resume, it can restore the graphics mode and state. After saving all of the registers, the process restores the previous mode−the mode before the OS entered into the GUI, i.e., console/terminal display mode. After restoring the previous mode, the end-user sees a text mode display on the screen and the kernel/OS can display text, which helps when debugging power management issues.

Linux Power Management – **Suspend to RAM**

After restoring the display to the console, the Linux kernel freezes all user processes. This includes the X process and any other application running on the system (window manager, browsers, X applications, etc.). With the user applications frozen, the kernel can work with each device to ensure those devices support the power management operation correctly and then invoke the Suspend to RAM operation. After the user processes freeze, the kernel probes each device driver and calls its associated suspend function (if implemented). This gives each driver the opportunity to flush queues, stop interrupts, process any remaining interrupts, and lower the power state of the device. Some devices may or may not support putting the device into any other lower power state other than D3 (powered off). Other devices may support intermediate states, such as D1 or D2, i.e., the resulting power state depends on the device capabilities.

Linux Power Management – **Suspend to RAM**

In the case of IEGD, the driver puts the device into D3 no matter what kind of power management event occurs. However, the driver sets D3 behavior in the graphics controller by turning off the graphics device's clocks on D3. On a Suspend to RAM that makes a device operate in a lower power state.

Linux Power Management – **Suspend to RAM**

After all devices have been put into a lower power state, the only code left running is the kernel. The kernel does its own clean-up and freezes all processors except for the one currently running. After running the kernel-side suspend code, a couple of ACPI methods are executed: PTS (Prepare To Sleep) and GTS (Going To Sleep). These executions tend to poke various things that the kernel knows nothing about, and so a certain amount of "magic" may be involved. At this point, the system should be dormant. Next, the address of the kernel wakeup code is written to a location in the Fixed ACPI Description Table (FADT). Finally, DSDT values are written to registers described in the FADT. This usually causes some sort of system management trap to ensure that the memory starts in self-refresh mode; it also sequences the machine into suspend. For the S3 power state, this involves shutting the machine down completely except for the RAM.

Linux Power Management – **Suspend to RAM**

With the wake vector set, and user processes, devices drivers, and kernel all in a “cleaned-up” state, the BIOS can now put the platform into a lower power state.

Linux Power Management – **Resume/Wake-up**

When the user takes an action to wake the system, for example, a keystroke, mouse movement, or a power button press, the system switches on. It then jumps to the BIOS start address, does some setup such as programming the memory controller, and then looks at the ACPI status register.

The ACPI status register indicates that the machine was previously suspended to RAM, so it then jumps to the wakeup address programmed earlier. The wakeup address leads to real-mode x86 code provided by the kernel, which programs the CPU back into protected mode and restores register state. At this point, the kernel code continues execution.

Linux Power Management – **Resume/Wake-up**

From this point, the process is essentially the reverse of the suspend process. The ACPI WAK method is called, all of the drivers are resumed, and user space is restarted.

If X is running, Linux switches from console to the GUI. In the case of IEGD and X, like the LeaveVT function, there is the analogous EnterVT (“enter Virtual Terminal”), which restores the previously saved graphics mode−the GUI display settings, saved register, and graphics device state information. After EnterVT restores everything fully, it saves the previous mode, i.e., the console mode before reentering the GUI.

With the kernel running again, all devices running, and user space applications, including X, running again the system has successfully resumed/woken-up from S3 Suspend to RAM.

Linux Test Project - POWER MANAGEMENT TESTS AUTOMATION SUITE

The tests requires the Kernel to be compiled with the following config's

for CPU FREQUENCY tests:

CONFIG_CPU_FREQ

CONFIG_CPU_FREQ_TABLE

CONFIG_CPU_FREQ_DEBUG

CONFIG_CPU_FREQ_STAT

CONFIG_CPU_FREQ_STAT_DETAILS

CONFIG_CPU_FREQ_DEFAULT_GOV_*

CONFIG_CPU_FREQ_GOV_*

Linux Test Project - POWER MANAGEMENT TESTS AUTOMATION SUITE

for CPU IDLE tests:

CONFIG_CPU_IDLE

CONFIG_CPU_IDLE_GOV_LADDER

CONFIG_CPU_IDLE_GOV_MENU

for SCHED_MC tests:

CONFIG_SCHED_MC

Linux Test Project - POWER MANAGEMENT TESTS AUTOMATION SUITE

The power management test automation suite helps run the power management functionality

(e.g: cpu frequency, cpu idle etc..) tests and report results.

Test Scripts for CPU FREQUENCY:

change_freq.sh

change_govr.sh

check_cpufreq_sysfs_files.sh

Linux Test Project - POWER MANAGEMENT TESTS AUTOMATION SUITE

Test Scripts for CPU IDLE: will be added soon

Test Scripts for SCHED_MC:

- test_sched_mc.sh

Common functionality:

- pm_include.sh
- check_kv_arch.c
- pwkm_load_unload.sh

Linux Test Project - POWER MANAGEMENT TESTS AUTOMATION SUITE

To run your tests you can execute the **runpwtests.sh**

To run the tests individually :

P.S. As of now the supporting architecture(s) are x86,x86_64

Support of system:

If you see some thing like following,

```
Power Management    1  FAIL  :  Required kernel configuration for SCHED_MC NOT set
```

or

```
Power Management    1  FAIL  :  Required kernel configuration for CPU_FREQ NOT set
```

Then either configuration is not set or the system won't support.

Linux Test Project - POWER MANAGEMENT TESTS AUTOMATION SUITE

For CPU consolidation verification ebizzy is included in utils directory of LTP.

To run cpu consolidation test user has to provide -w <workload> -l <sched_mc_level>.

Refer to README in LTPROOT/utils/benchmark/ebizzy-0.2 directory for details of ebizzy.

To test CPU consolidation for sched_mc 2 kernbench has to run. Kernbench needs linux kernel source as input in /root directory . For example download from http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.29.4.tar.bz2. If Linux kernel source not found kernbench wiil not execute.

CPU consolidation testcases will not execute if number of CPU's in package is less then 2. If system is hyper threaded but number of CPU is 1 only sched_smt testcases will be excuted. For better coverage of testcases select a system which is atleast quad core and then hyper threaded so that you will observe 8 CPU's in each package.

Timer migration interface test will execute on kernel versions 2.6.31 and above. Timer migration functionality verification testcases will be executed only on suitable architecture like quad core or the number of CPU's in each package should be atleast 4 and above

Session 5

- Power Management-Hands On

Power Management – Session 5

Zaheer R M

Contents

Powder Management-Hands On Session

- Kernel Configuration
- Dynamic Voltage and Frequency Scaling
- Suspend/Resume
- CPU Idle
- Questions

Kernel Configuration

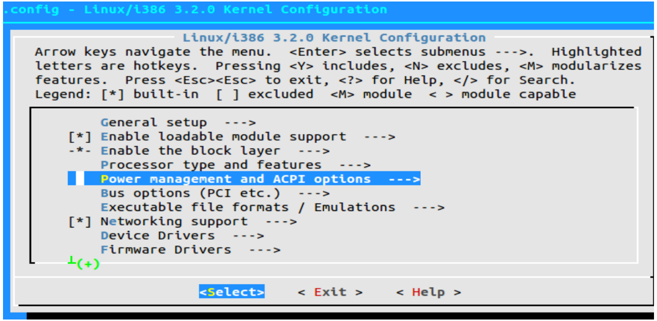
- CPU Idle Kernel Config
- CPU Freq Kernel Config
- PM Firmware Kernel Config (required for suspend/resume)
- Enable/Disable suspend-resume support

Kernel Configuration

CPU Idle Kernel Config

- On host, open terminal and cd to /home/sitara/ti-sdk-am335x-evm-05.04.01.00/board-support/linux-3.2-psp04.06.00.07.sdk
- Start the Linux Kernel Configuration tool
- $ make menuconfig

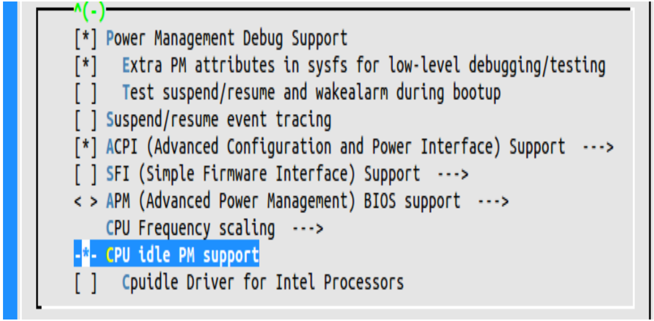
- Select CPU Power Management from the main menu
- Select CPU idle PM support to enable the cpuidle driver.



```
.config - Linux/i386 3.2.0 Kernel Configuration
                    Linux/i386 3.2.0 Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted
  letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.
  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

            General setup  --->
        [*] Enable loadable module support  --->
        -*- Enable the block layer  --->
            Processor type and features  --->
            Power management and ACPI options  --->
            Bus options (PCI etc.)  --->
            Executable file formats / Emulations  --->
        [*] Networking support  --->
            Device Drivers  --->
            Firmware Drivers  --->
        ┴(+)

                    <Select>    < Exit >    < Help >
```



```
    ^(-)
    [*] Power Management Debug Support
    [*]   Extra PM attributes in sysfs for low-level debugging/testing
    [ ]   Test suspend/resume and wakealarm during bootup
    [ ] Suspend/resume event tracing
    [*] ACPI (Advanced Configuration and Power Interface) Support  --->
    [ ] SFI (Simple Firmware Interface) Support  --->
    < > APM (Advanced Power Management) BIOS support  --->
        CPU Frequency scaling  --->
    -*- CPU idle PM support
    [ ]   Cpuidle Driver for Intel Processors
```

Kernel Configuration

CPU Freq Kernel Config

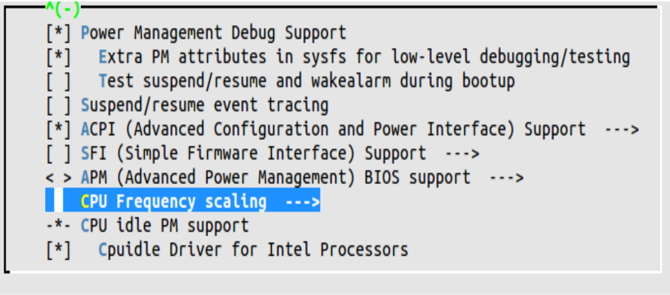
Select CPU Power Management from the main menu.

Select CPU Frequency scaling and required governors.

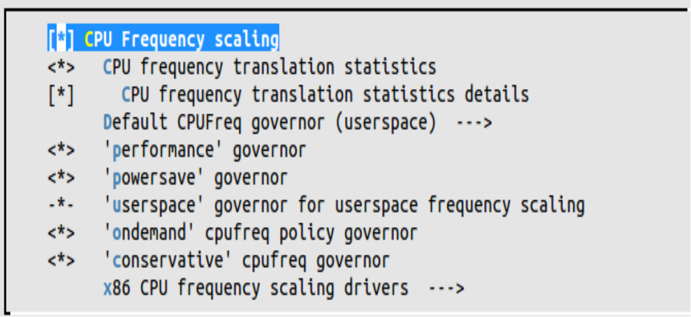
Not all governors may be enabled by default.

If your application requires a specific governor,

check here to ensure it has been enabled.



```
^(-)
[*] Power Management Debug Support
[*]   Extra PM attributes in sysfs for low-level debugging/testing
[ ]   Test suspend/resume and wakealarm during bootup
[ ] Suspend/resume event tracing
[*] ACPI (Advanced Configuration and Power Interface) Support  --->
[ ] SFI (Simple Firmware Interface) Support  --->
< > APM (Advanced Power Management) BIOS support  --->
    CPU Frequency scaling  --->
-*- CPU idle PM support
[*]   Cpuidle Driver for Intel Processors
```



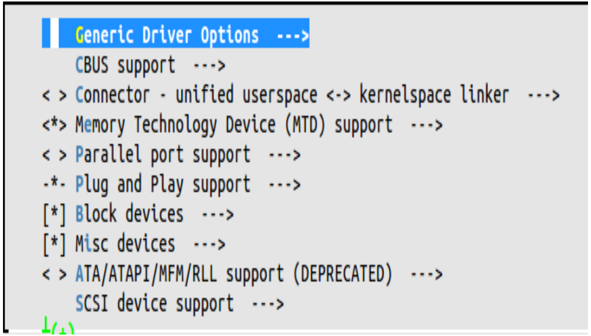
```
[*] CPU Frequency scaling
<*>   CPU frequency translation statistics
[*]     CPU frequency translation statistics details
      Default CPUFreq governor (userspace)  --->
<*>   'performance' governor
<*>   'powersave' governor
-*-   'userspace' governor for userspace frequency scaling
<*>   'ondemand' cpufreq policy governor
<*>   'conservative' cpufreq governor
      x86 CPU frequency scaling drivers  --->
```

Kernel Configuration

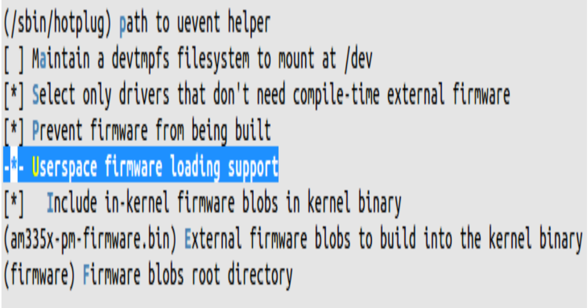
PM Firmware Kernel Config

- On AM335x, suspend-resume involves loading a binary to the Cortex-M3 core during the boot process. For this, the binary named am335x-pm-firmware.bin needs to be kept in the firmware/ folder on the kernel sources before the build process is started.
- For more information on how to obtain the above mentioned binary, refer to the accompanying Release Notes for the PSP package.
- The pre-built kernel image in the PSP package has the binary blob compiled into the kernel image.
- To manually compile a kernel image with the PM firmware start the Linux Kernel Configuration tool.

- Start the Linux Kernel Configuration tool.
- Select Device Drivers from the main menu.
- Select Generic Driver Options
- Configure the name of the PM firmware and the location



```
    Generic Driver Options  --->
    CBUS support  --->
< > Connector - unified userspace <-> kernelspace linker  --->
<*> Memory Technology Device (MTD) support  --->
< > Parallel port support  --->
-*- Plug and Play support  --->
[*] Block devices  --->
[*] Misc devices  --->
< > ATA/ATAPI/MFM/RLL support (DEPRECATED)  --->
    SCSI device support  --->
↓(+)
```

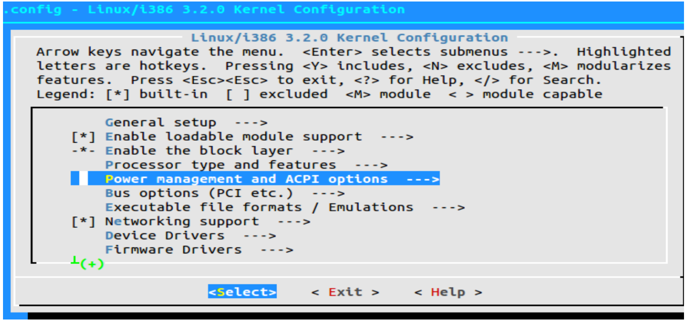


```
(/sbin/hotplug) path to uevent helper
[ ] Maintain a devtmpfs filesystem to mount at /dev
[*] Select only drivers that don't need compile-time external firmware
[*] Prevent firmware from being built
-*- Userspace firmware loading support
[*]   Include in-kernel firmware blobs in kernel binary
(am335x-pm-firmware.bin) External firmware blobs to build into the kernel binary
(firmware) Firmware blobs root directory
```

Kernel Configuration

Enable/Disable suspend-resume support

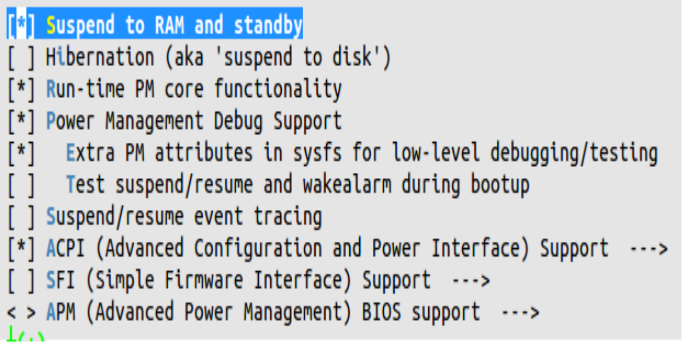
1. To enable/ disable suspend-resume support start the Linux Kernel Configuration tool.
2. Select Power management options from the main menu
3. Select Suspend to RAM and standby to toggle the power management support.



```
.config - Linux/i386 3.2.0 Kernel Configuration
                    Linux/i386 3.2.0 Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted
  letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.
  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

            General setup  --->
        [*] Enable loadable module support  --->
        -*- Enable the block layer  --->
            Processor type and features  --->
            Power management and ACPI options  --->
            Bus options (PCI etc.)  --->
            Executable file formats / Emulations  --->
        [*] Networking support  --->
            Device Drivers  --->
            Firmware Drivers  --->
        ↓(+)

                  <Select>    < Exit >    < Help >
```

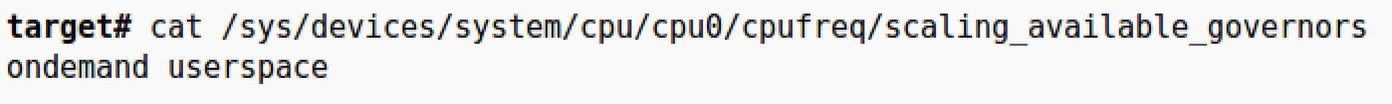


```
[*] Suspend to RAM and standby
[ ] Hibernation (aka 'suspend to disk')
[*] Run-time PM core functionality
[*] Power Management Debug Support
[*]   Extra PM attributes in sysfs for low-level debugging/testing
[ ]   Test suspend/resume and wakealarm during bootup
[ ] Suspend/resume event tracing
[*] ACPI (Advanced Configuration and Power Interface) Support  --->
[ ] SFI (Simple Firmware Interface) Support  --->
< > APM (Advanced Power Management) BIOS support  --->
↓(+)
```

Dynamic Voltage and Frequency Scaling

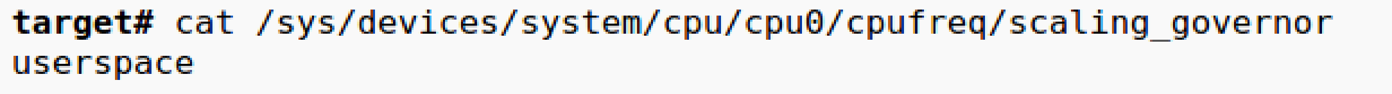
Exploring CPUFreq Governors

- View all available governors:



```
target# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors
ondemand userspace
```

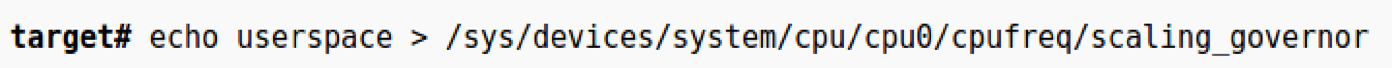
- View current governor:



```
target# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
userspace
```

View and Change Frequency (OPP)

- Set the cpufreq governor back to 'userspace', which is required in order to change the frequency manually.

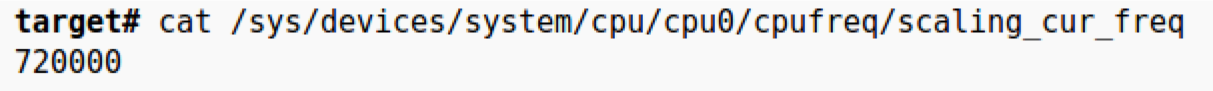


```
target# echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

Dynamic Voltage and Frequency Scaling

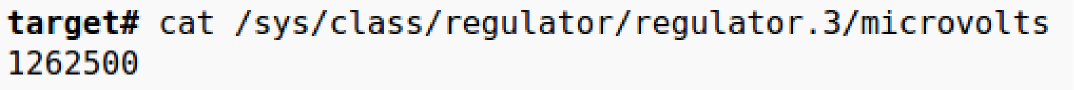
View and Change Frequency (OPP)

- View the current frequency



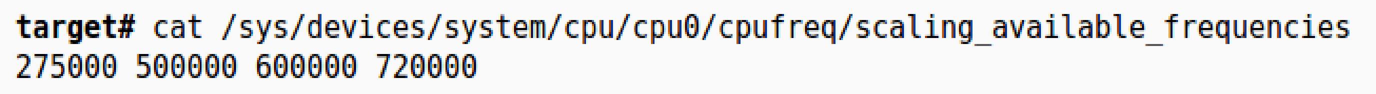
```
target# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
720000
```

- View the current voltage



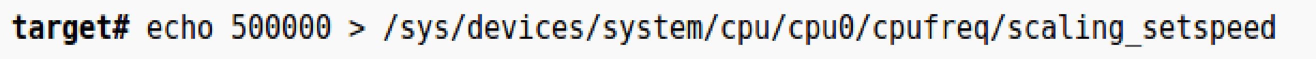
```
target# cat /sys/class/regulator/regulator.3/microvolts
1262500
```

- View the supported OPP's (frequencies)



```
target# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies
275000 500000 600000 720000
```

- Change the frequency (This can be done only for userspace governor)



```
target# echo 500000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
```

Suspend / Resume

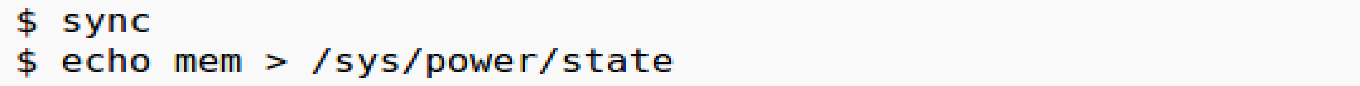
- The suspend operation results in the system transitioning to the lowest possible power state. On AM335x, this maps to DeepSleep0 state.

- 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 quiesce the peripherals and release the clocks to reach the desired low power state. The actual transition to suspend is implemented in the function am33xx_pm_suspend()

- Wakeup from suspend is supported using the following interfaces:
  - UART0 (console)
  - GPIO0
  - Touchscreen

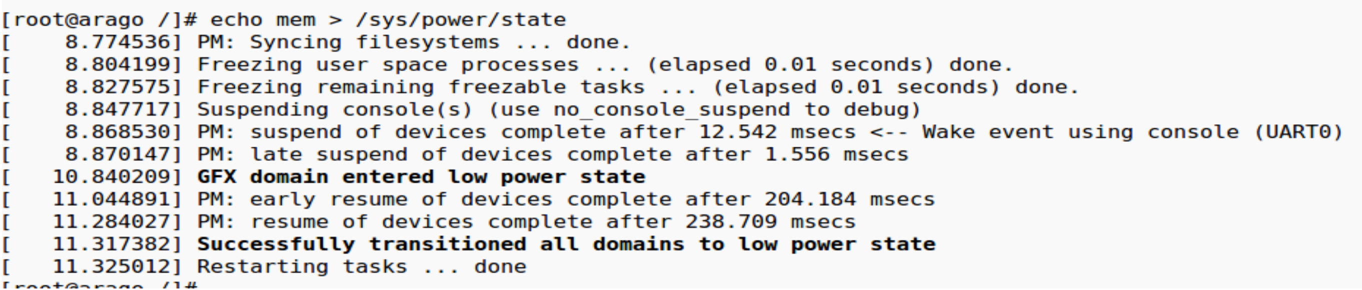
Suspend / Resume

- From Matrix home screen, touch '3D', then touch 'Chameleon' to see 3D graphics in action.
- The suspend for device can now be asserted as follows (note that it is recommended to issue a 'sync' first to avoid data loss):



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

- To wakeup, send a character over the UART0 terminal, or touch the touchscreen.
- The following output is expected for a successful suspend/resume cycle:



```
[root@arago /]# echo mem > /sys/power/state
[    8.774536] PM: Syncing filesystems ... done.
[    8.804199] Freezing user space processes ... (elapsed 0.01 seconds) done.
[    8.827575] Freezing remaining freezable tasks ... (elapsed 0.01 seconds) done.
[    8.847717] Suspending console(s) (use no_console_suspend to debug)
[    8.868530] PM: suspend of devices complete after 12.542 msecs <-- Wake event using console (UART0)
[    8.870147] PM: late suspend of devices complete after 1.556 msecs
[   10.840209] GFX domain entered low power state
[   11.044891] PM: early resume of devices complete after 204.184 msecs
[   11.284027] PM: resume of devices complete after 238.709 msecs
[   11.317382] Successfully transitioned all domains to low power state
[   11.325012] Restarting tasks ... done
[root@arago /]#
```

CPU Idle

The cpuidle framework consists of two key components:

1. A governor that decides the target C-state of the system.
2. A driver that causes the transition to the target C-state.

- AM335x contains only two C-states. State C1 is described as 'MPU WFI', where WFI means 'wait for interrupt'. Essentially, the MPU is notified that there is no work to do, and it enters a low-power state. State C2 is deeper, and in addition to MPU WFI, it takes the DDR into self-refresh mode, saving more power.

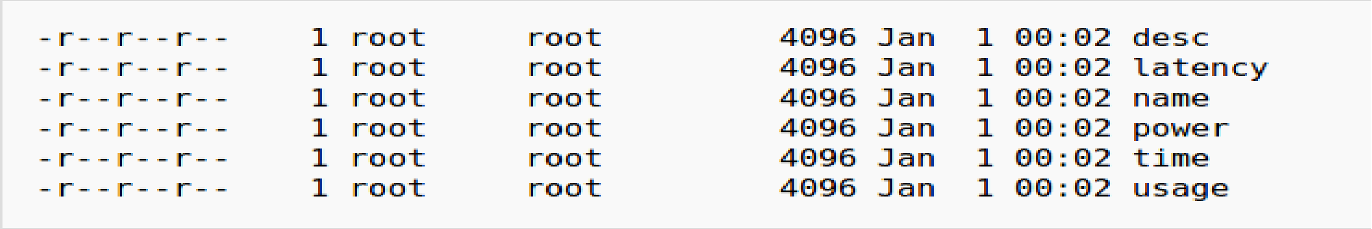
Cpuidle governors available are :

- **ladder -** steps down or up sleep states one at a time depending on the time spent in the last idle idle period. It works well with a regular timer tick, but not with dynamic tick.
- **menu -** selects sleep state based on expected idle time. Works well with dynamic tick systems.

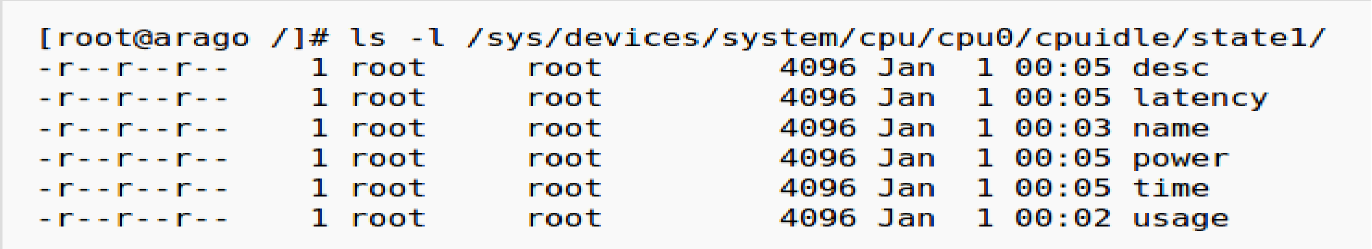
CPU Idle

- Use sysfs interface to view detailed information about the C-states (state0 and state1). Sysfs contains useful entries which have detailed information about the C-states (state0 and state1). Most importantly, 'time' allows you to see how much time has been spent in a given state.

```
[root@arago /]# ls -l /sys/devices/system/cpu/cpu0/cpuidle/state0/
```



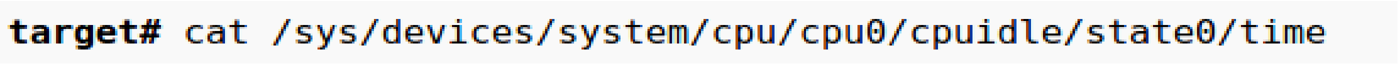
```
-r--r--r--    1 root     root          4096 Jan  1 00:02 desc
-r--r--r--    1 root     root          4096 Jan  1 00:02 latency
-r--r--r--    1 root     root          4096 Jan  1 00:02 name
-r--r--r--    1 root     root          4096 Jan  1 00:02 power
-r--r--r--    1 root     root          4096 Jan  1 00:02 time
-r--r--r--    1 root     root          4096 Jan  1 00:02 usage
```



```
[root@arago /]# ls -l /sys/devices/system/cpu/cpu0/cpuidle/state1/
-r--r--r--    1 root     root          4096 Jan  1 00:05 desc
-r--r--r--    1 root     root          4096 Jan  1 00:05 latency
-r--r--r--    1 root     root          4096 Jan  1 00:03 name
-r--r--r--    1 root     root          4096 Jan  1 00:05 power
-r--r--r--    1 root     root          4096 Jan  1 00:05 time
-r--r--r--    1 root     root          4096 Jan  1 00:02 usage
```

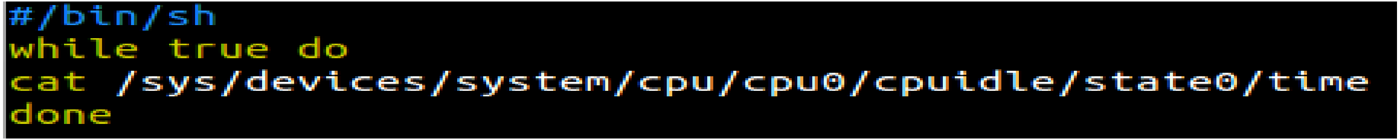
CPU Idle

- Print out the time spent in state0



```
target# cat /sys/devices/system/cpu/cpu0/cpuidle/state0/time
```

- Run the above command a few times to observe that the timestamp is increasing. This is because the MPU is actually able to re-enter the idle state between your commands!
- Now, let's write a simple shell script to repeatedly dump the timestamp.



```
#/bin/sh
while true do
cat /sys/devices/system/cpu/cpu0/cpuidle/state0/time
done
```

- Make script executable using chmod

CPU Idle

- Execute the script! You will notice that , state0 timestamp does not advance because we are keeping the CPU busy
- Kill the script with Ctrl-Z at anytime.
- You may notice on the AM335x EVM that state1 is never entered. This is due to the LCD module being enabled and driving constant interrupts. If the LCD module was disabled, CPUIdle state1 is possible and represents a lower power state of idle.

Q&A – Any suggestions

1. What is meant by policy in PM definition ?
2. What is hardware timing closure results in PM principles ?
3. What is meant by all clocks are running in PM state diagram ?
4. Each and every device run with different clock speed. how these clocks will be handled ?
5. What is the active state of a peripheral (all other on-chip peripherals are active) ?
6. How do you define inactivity of a peripheral. Based on interrupt or instruction ?
7. Who processes the interrupt when CPU is sleeping in idle mode ?
8. How the state changes will happen between two states ?
9. When the system is in sleep state, where the data will be stored ?
10. What is the load. How its defined ?
11. After display off what state CPU is going? is it due to interrupt inactivity or no instructions to execute ?

Q&A – Any suggestions

12. PM events in APM model are constant or configurable (add or remove events) ?
13. How APM BIOS will be interacted with the driver ?
14. In standby mode, only the CPU is suspended or any other peripherals ?
15. In suspend mode what is the speed of CPU ?
16. For some systems, after suspend Power LED blinks. Who will take care of this operation ?
17. Is there any call flow model between device states D1 & D2 in APM model ?
18. Can we make single core sleep in multi core environment ?
19. How to define system performance requirements in DVFS ?
20. Who will control DVFS ?
21. How the interrupts are handled in sleep mode ?
22. Is there an order for suspending the device states in APM model ?
23. ACPI having the backward compatibility to support the APM standard ?

Large enough to Deliver, Small enough to Care

Global Village
IT SEZ
Bangalore

South Main Street
Milpitas
California

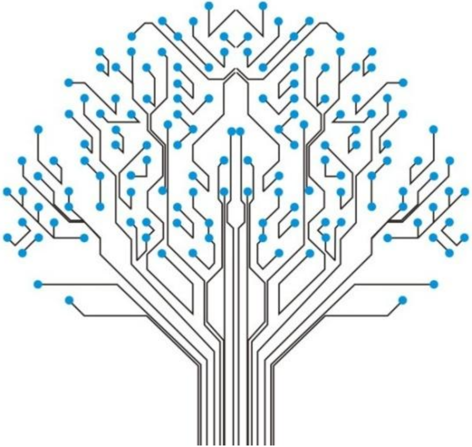
Raheja Mindspace
IT Park
Hyderabad

www.globaledgesoft.com

GLOBALEDGE

Intelligence Of Things

Thank you



Fairness

Learning

Responsibility

Innovation

Respect