SITARA™ ARM® PROCESSORS

BOOt camp

Linux Power Management

This session covers power management within Linux including DVFS with CPUFreq, CPUIdle, and PMICs, with a strong emphasis on Suspend/Resume.

Author: Texas Instruments®, Sitara™ ARM® Processors

May 2014

Creative Commons Attribution-ShareAlike 3.0 (CC BY-SA 3.0)

creative commons

You are free:

to Share – to copy, distribute and transmit the work

to Remix – to adapt the work

to make commercial use of the work

Under the following conditions:

Attribution – You must give the original author(s) credit

Share Alike - If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

With the understanding that:

Waiver — Any of the above conditions can be waived if you get permission from the copyright holder.

Public Domain — Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.

Other Rights — In no way are any of the following rights affected by the license:

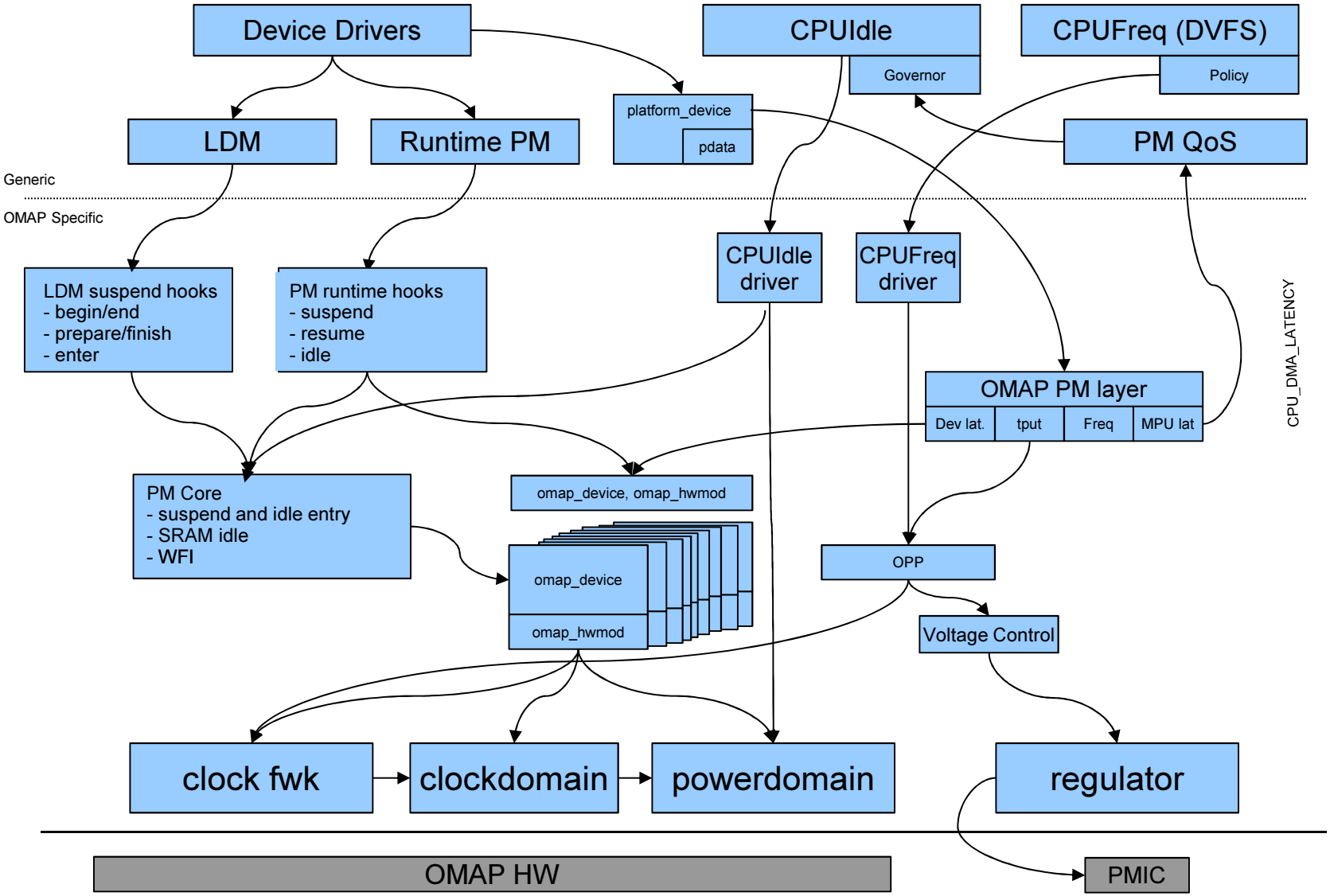
Notice — For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page.

CC BY-SA

Contents

- Linux Power Management Framework (50,000 foot view)
- DVFS / Cpufreq
- DCG / CPU Idle
- DeepSleep0 / Suspend and Resume
  - IO Pad Configuration
  - Debugging steps for low power mode entry

Linux Power Management Framework



Courtesy of Kevin Hilman

Texas Instruments

Dynamic Voltage and Frequency Scaling

- Relies upon the **cpufreq** framework in Linux
- Cpufreq provides support to change frequency and voltage of MPU
  - This can help to save processor power dynamically according to load
  - Remember, processor power is proportional to frequency and square of the voltage
- Cpufreq governor decides frequency at which processor should run
  - **Performance**: CPU statically set to highest possible frequency
  - **Powersave**: CPU statically set to lowest possible frequency
  - **Userspace**: controlled by user via 'scaling_setspeed'
  - **Ondemand**: Sets CPU freq dynamically based on load
  - **Conservative**: Like ondemand, but slower to respond (better for battery powered environment)

CPUFreq User Space Examples

- View all available governors:

```
root@am335x-evm# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors
        ondemand userspace performance conservative powersave
```

- View current governor:

```
root@am335x-evm # cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
        ondemand
```

- View and Change Frequency (OPP):
  - First, set cpufreq governor to userspace
  - View current frequency:

```
root@am335x-evm# cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
720000
```

  - View current voltage:

```
root@am335x-evm# cat /sys/class/regulator/regulator.3/microvolts
1262500
```

  - View Supported OPPs:

```
root@am335x-evm# cat
/sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies
275000 500000 600000 720000
```

  - Change Frequency to 500MHz:

```
root@am335x-evm# echo 500000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
```

CPUFreq: OPP Table

- OPP Table is now defined in the device tree. Can be found in the SoC device tree file:

```
operating-points = <
        /* kHz    uV */
        720000  1285000
        600000  1225000
        500000  1125000
        275000  1125000
>;
```

- All OPPs are defined as frequency/voltage pairs. Only frequency entries must be unique.
- Changing all voltages to the same value (without going lower than default voltage value for that OPP) allows for Dynamic Frequency Scaling.
- You can freely lower the frequency of any OPP without changing the voltage
- In 3.12 Kernel all OPPs must have a corresponding opp-modifier table entry as well.

CPUFreq: OPP Modifier

- OPP Modifier is a new framework introduced in 3.12 kernel to handle dynamically controlled OPPs (eFused OPPs)
- Each entry in the table is four values:
  - Frequency
  - Silicon Revision
  - Register Offset
  - Bit mask
- A value of zero in the mask column will always enable that OPP for the silicon revision
- Otherwise, the bitmask is compared to the register and enabled based on that.
- If a frequency entry does not exist explicitly for the SoC revision in use it will NOT be enabled.

CPUFreq: OPP Modifier

- Example from arch/arm/boot/dts/am33xx.dtsi:

```
mpu_opp_modifier: mpu_opp_modifier {
        opp-modifier = <
        /* kHz    Rev                              offset  mask */
        1000000   OPP_REV(2,1)                     0       AM33XX_EFUSE_SMA_OPP_NITRO_1GHZ_BIT
        800000    OPP_REV(2,1)                     0       AM33XX_EFUSE_SMA_OPP_TURBO_800MHZ_BIT
        720000    OPP_REV(2,1)                     0       AM33XX_EFUSE_SMA_OPP_120_720MHZ_BIT
        600000    OPP_REV(2,1)                     0       AM33XX_EFUSE_SMA_OPP_100_600MHZ_BIT
        300000    (OPP_REV(2,0) | OPP_REV(2,1))    0       0
        1000000   OPP_REV(2,0)                     0       0
        800000    OPP_REV(2,0)                     0       0
        720000    (OPP_REV(1,0) | OPP_REV(2,0))    0       0
        600000    (OPP_REV(1,0) | OPP_REV(2,0))    0       0
        500000    (OPP_REV(1,0))                   0       0
        275000    (OPP_REV(1,0))                   0       0
        >;
};
```

CPUFreq and OPP Modifier

- **Relevant Files:**
  - drivers/cpufreq/cpufreq-cpu0.c
    - Generic CPUFreq driver, now used by all TI SoCs supported by 3.12 kernel
  - drivers/power/opp/omap-opp-modifier.c
    - Opp-modifier driver for eFuse controlled opp's
  - Documentation/devicetree/bindings/power/opp-modifier.txt
    - Device tree binding documentation for opp-modifier

Dynamic Clock Gating: CPUIdle

- CPUIdle lets the system enter low power states during periods of no activity.
- Like cpufreq, the cpuidle framework consists of two key components:
  - A governor that decides the target C-state of the system.
  - A driver that causes the transition to the target C-state.
- AM335x supports three different C-states
  - **MPU WFI**
  - **MPU WFI + Bypass MPU PLL**
  - **MPU WFI + Bypass MPU PLL + DDR Self-Refresh (DDR2 Only)**
- Support for AM437x CPUIdle not available in SDK 7.0, but there is a patch available for WFI and Bypass MPU PLL

CPUIdle

- CPUIdle requires the CM3 firmware to be loaded to work, this is done by default
- Cpuidle governors available are 'menu' and 'ladder':
  - 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. This is the default.
- By default, no parameters of CPUidle can be changed at runtime but stats are available under /sys/devices/system/cpu/cpu0/cpuidle/*

CPUIdle

- Relevant Files:
  - arch/arm/mach-omap2/cpuidle33xx.c
    - AM33xx specific cpuidle driver
  - drivers/cpuidle/*
    - All generic cpuidle files including governors with core framework defined in cpuidle.c

DeepSleep0: Suspend/Resume

- Suspend-resume
  - System transitions to a low power state where processor is idle
  - AM33xx has both DeepSleep0 and Standby modes available.
  - AM43xx only officially supports DeepSleep0 in SDK 7.0.
- Suspend process is initiated by the user via the standard kernel interface:

```
echo mem > /sys/power/state        (DS0)
```

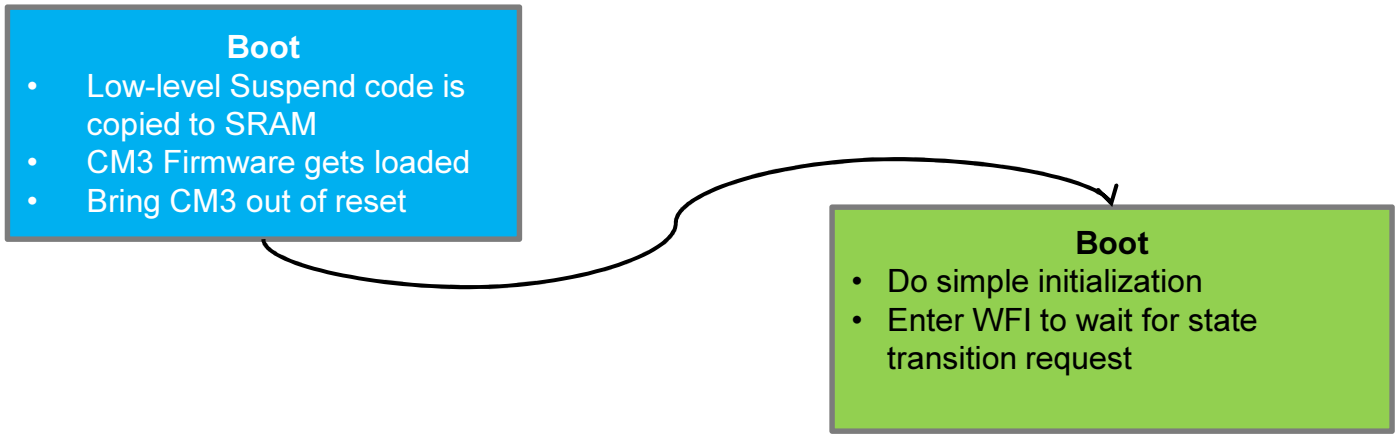
or

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

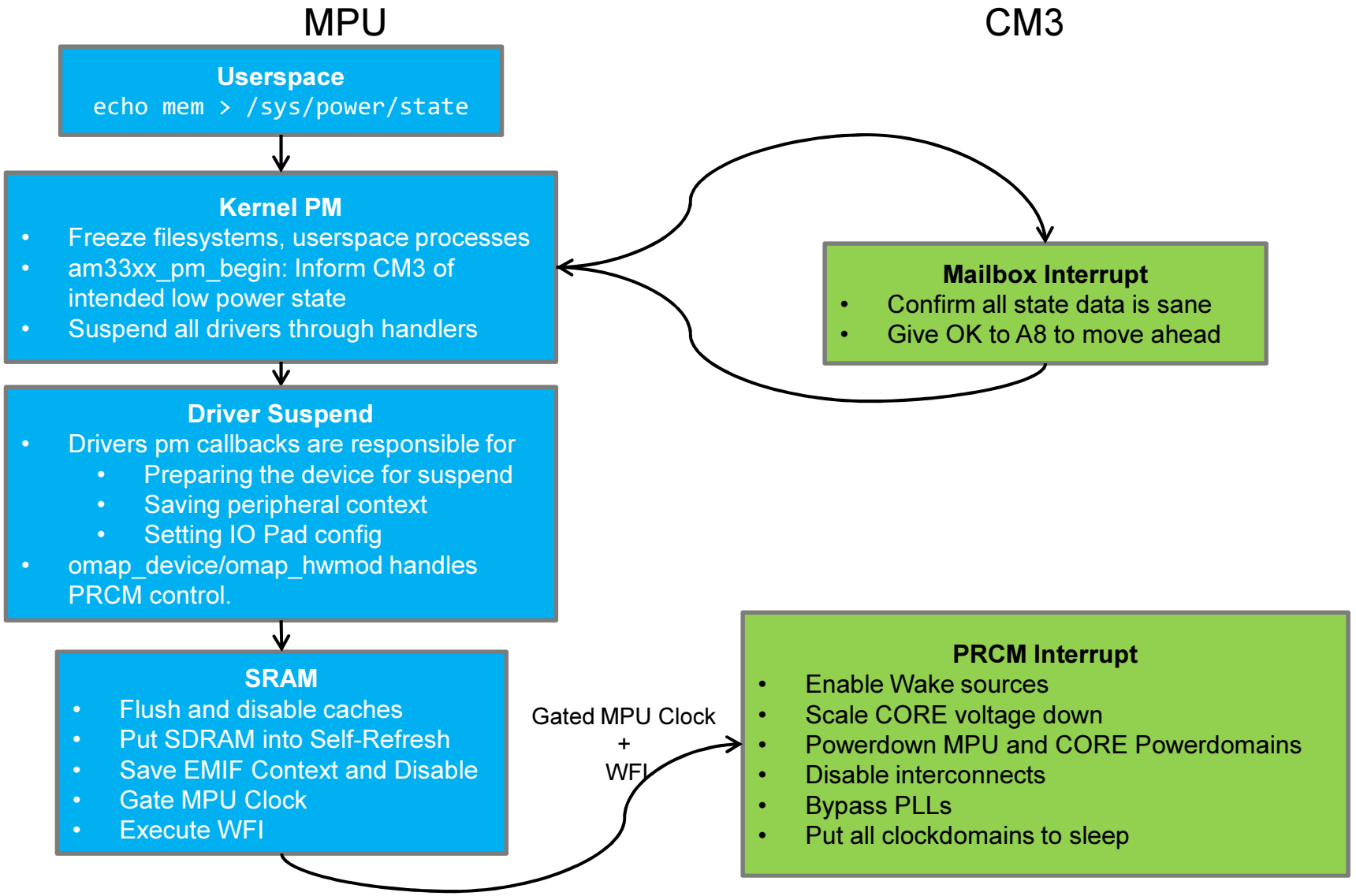
- DeepSleep0 wakeup sources supported:
  - UART0, Touchscreen, GPIO0, RTC, I2C0
- Possible to use other peripheral interrupts as wake sources with Standby mode on AM335x, AM437x standby mode still needs additional validation.
- Both platforms require CM3 firmware!

Linux Boot Sequence for CM3

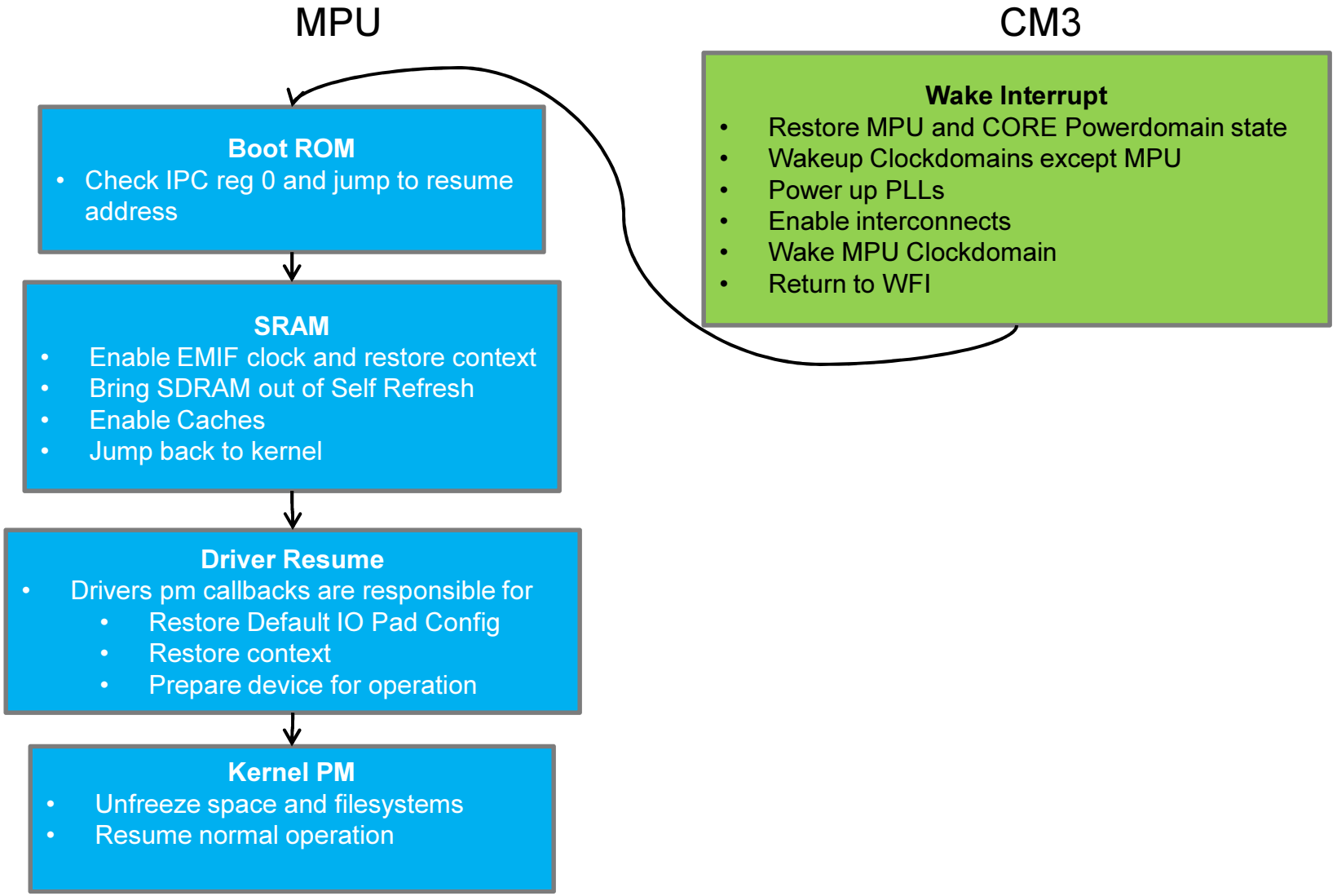
MPU CM3



Linux Suspend Sequence



Linux Resume Sequence



Suspend/Resume Latency

- Suspend/Resume Latencies (DeepSleep0 on AM335x EVM)
  - Unoptimized suspend latency is ~150ms
  - Unoptimized resume latency is ~300ms
- Most latency comes from drivers, if speed is needed remove what you don't need
- Standby mode (PER domain ON) can allow for better latency because context is not lost.
- Software instrumentation can be added in drivers/base/power/main.c, in the dpm_suspend() and dpm_resume() calls to see how many devices there are, and the contribution of each to the overall latency.
  - Practice usually shows that most of the delay is caused by a small number of devices, so optimization is possible depending on the use case.

Suspend/Resume

- Relevant Files
  - `arch/arm/mach-omap2/pm33xx.c`
    - AM33XX and AM43XX pm init code, most suspend/resume functionality
  - `arch/arm/mach-omap2/sleep33xx.S` and `arch/arm/mach-omap2/sleep43xx.S`
    - Assembly code that performs final steps just prior to WFI instruction
  - `kernel/power/suspend.c`
    - Core kernel suspend/resume code. Starts with pm_suspend function.
  - `drivers/base/power/main.c`
    - Core driver suspend logic is here `dpm_suspend` and `dpm_resume`
  - `drivers/*`
    - Every driver with suspend/resume handlers!
  - `arch/arm/mach-omap2/omap_hwmod.c` and `arch/arm/mach-omap2/omap_device.c`
    - Low level handling of omap modules, where CLKCTRLs are controlled from.

IO Pad Configuration

- Allows for different IO Pad configuration (Pad control registers in control module) during runtime and suspend
- In general, significant power can be saved by performing for a board, but the configuration is specific to that board.
- Handled in the board device tree now with `pinctrl` driver.
- All devices are responsible for their own pins and have both a default and sleep state.
- If a sleep state is defined a default state must also be defined so runtime state can be restored during resume.
- Driver is responsible for switching state in its suspend and resume handlers using `pinctrl_pm_select_sleep_state` and `pinctrl_pm_select_default_state`

IO Pad Configuration

- Example (am33xx.dtsi):

```
davinci_mdio_default: davinci_mdio_default {
        pinctrl-single,pins = <
                /* MDIO */
                0x148 (PIN_INPUT_PULLUP | SLEWCTRL_FAST | MUX_MODE0)
                0x14c (PIN_OUTPUT_PULLUP | MUX_MODE0)
        >;
};

davinci_mdio_sleep: davinci_mdio_sleep {
        pinctrl-single,pins = <
                /* MDIO reset value */
                0x148 (PIN_INPUT_PULLDOWN | MUX_MODE7)
                0x14c (PIN_INPUT_PULLDOWN | MUX_MODE7)
        >;
};
…
…
&davinci_mdio {
        pinctrl-names = "default", "sleep";
        pinctrl-0 = <&davinci_mdio_default>;
        pinctrl-1 = <&davinci_mdio_sleep>;
};
```

Working Suspend/Resume Cycle

- Results in the following prints on the console.

```
root@am335x-evm:~# echo mem > /sys/power/state
[37.219927] PM: Syncing filesystems ... done.
[37.229145] Freezing user space processes ... (elapsed 0.001 seconds) done.
[37.238027] Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
[37.247270] Suspending console(s) (use no_console_suspend to debug)
[37.276634] PM: suspend of devices complete after 21.721 msecs
[37.277679] PM: late suspend of devices complete after 1.013 msecs
[37.278959] PM: noirq suspend of devices complete after 1.248 msecs
[37.279101] PM: Successfully put all powerdomains to target state
[37.279101] PM: Wakeup source UART
[37.295638] PM: noirq resume of devices complete after 16.451 msecs
[37.296527] PM: early resume of devices complete after 0.689 msecs
[37.297477] net eth0: initializing cpsw version 1.12 (0)
[37.300505] net eth0: phy found : id is : 0x7c0f1
[37.465140] PM: resume of devices complete after 168.566 msecs
[37.526079] Restarting tasks ... done.
[39.298392] libphy: 4a101000.mdio:00 - Link is Up - 100/Full
```

Bad Suspend/Resume Cycle

- Suspend can be intentionally broken by activating a clock in PER domain

```
root@am335x-evm:~# devmem2 0x44e00044 w 0x2
```

- Resulting cycle looks like:

```
root@am335x-evm:~# echo mem > /sys/power/state
[132.618493] PM: Syncing filesystems ... done.
[132.628861] Freezing user space processes ... (elapsed 0.001 seconds) done.
[132.637859] Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
[132.647116] Suspending console(s) (use no_console_suspend to debug)
[132.669458] PM: suspend of devices complete after 14.680 msecs
[132.670411] PM: late suspend of devices complete after 0.923 msecs
[132.671670] PM: noirq suspend of devices complete after 1.228 msecs
[132.671808] PM: Could not transition all powerdomains to target state
[132.671808] PM: Wakeup source UART
[132.688225] PM: noirq resume of devices complete after 16.331 msecs
[132.689110] PM: early resume of devices complete after 0.687 msecs
[132.689938] net eth0: initializing cpsw version 1.12 (0)
[132.692895] net eth0: phy found : id is : 0x7c0f1
[132.855116] PM: resume of devices complete after 165.962 msecs
[132.916424] Restarting tasks ... done.
-sh: echo: write error: Operation not permitted
```

Suspend Failures

- Suspend is a system wide action: Each component is responsible for suspending itself. Changes in many different places can break suspend or resume.
- Test suspend ALL THE TIME during development. Develop it along with everything else.
- Most customer failures will come from different or new drivers and configuration that is removed.
- The board will typically fail in one of three ways:
  1. Suspend pass through with reported failure (as in previous example)
  2. Hang with a kernel panic
  3. Complete hang with no messages and no recovery

Debugging Suspend Resume

First Steps:

- Use a minimal rootfs and a minimal config, disable every possible driver (including mmc, use a ramdisk), then slowly add things back in.
- `echo no > /sys/module/printk/parameters/console_suspend` allows for additional console output during the suspend and resume sequences, may catch errors reported while UART is off.
  - TRADEOFF: No more UART wake while this is active, must use other sources
- pr_info is your friend, use it everywhere in the suspend/resume sequence and in driver pm handlers, even if just to place markers to indicate where in the suspend sequence the system is.

Debugging Suspend Resume

- JTAG can be very helpful and will sometimes be the only option.
- Being connected to Cortex A during WFI instruction will cause problems with suspend.
- Solution: Debuggers cause fewer problems when connecting to CM3. A break-point during ds0 entry function can allow for some exploration.
- CM3 Memory Map can access L4_WKUP peripherals, including PRCM and Control Module
- However, remember some hwmods are turned off by CM3, so place breakpoint after disable_x_hwmod functions or make sure to check CM3 code for which modules are turned off!

Debugging Suspend Resume

- In the case of reported power domain transition failure, it's most likely that all clocks are not off in the peripheral power domain.
- General things to check:
  1. Check the IDLEST bits in CM_PER_<MODULE>_CLKCTRL registers. Modules are idled when IDLEST bits transition to 0x3.
  2. CM_PER_<MODULE>_CLKCTRL of some of the modules also has a STANDBY bit. Modules enter STANDBY when this bit transitions to 0x1.
  3. For modules stuck in transition (IDLEST = 0x1), check the SYSCONFIG register for that module. Ensure IDLEMODE is set to "force idle", and STANDBYMODE is "forced standby", for the following modules: TPTC0/1/2, USB, CPSW, GPMC.
     - Note: SYSCONFIG registers are only set to "forced idle" or "forced standby" when transitioning to DeepSleep states. When active, they will read "no idle" or "smart idle"
  4. Other modules IDLEMODE should be set to "smart idle".
- CLKCTRL = 0x00030000 (no standby) or 0x00070000 (standby)
- CLKSTCTRL = 0x00000001
- Good idea to compare to known working setup.

Debugging Suspend Resume

- What to do for stuck in transition errors?
  - Remove the driver, does it suspend now?
    - If yes, you probably have an issue directly with the driver.
  - Check for pending interrupts or error conditions INSIDE the peripheral
    - Bus errors (L3 interconnect in particular) can occur for many reasons, check error registers within IP as a pending error will prevent transition. Either clear the error or prevent it from occurring in the first place. AM43xx has omap_l3_noc for this.
  - Check PLL registers in the PRCM
    - On AM43xx, some have a *_GATE_CTRL bit that can force a PLL active which prevents transition
  - Make sure the suspend handler is actually being called

Debugging Suspend Resume

- In the case of hangs with no messages at all:
  - Immediately after suspending, measure power. Then try to resume and measure power again. If it increased, suspend likely worked and the hang happened during resume.
  - Try a loop in arch/arm/mach-omap2/sleepxxxx.S, both pre and post WFI instruction

```
hang:
    b hang
```

  - Connect with JTAG, manually set PC to PC + 4, and then step through.
  - Make sure to disconnect before executing WFI on A8!
  - Certain modules must have clocks controlled by CM3, make sure kernel is not touching those (CM3 Source: `src/pm_services/hwmod_xxxx.c` for a list).
  - Also, try loops in CM3 code (CM3 Source: `src/pm_services/pm_handlers.c`, function `a8_lp_ds0_handler`), best way is to do:

```
volatile int stop = 0x12341234;
while (stop);
```

  - Then use JTAG and find the register = 0x12341234 and set to 0 while single stepping instructions to get past loop and continue execution.
  - Also basic printf support in CM3 but currently only readable by looking at memory address 0x81000 (from CM3).

Debugging Suspend Resume

- Try intentionally breaking suspend as in previous example
  - This will prevent PER context loss, which will indicate something is being saved or restored incorrectly.
- If the code does not make it to a loop before WFI, you are almost definitely stuck in a driver failure.
  - Again, pr_info everywhere!
  - Turn on `Device Drivers > Generic Driver Options > Driver Core verbose debug messages` under menuconfig
  - Many drivers have a debug configuration built-in. For example : `Device Drivers > MMC/SD/SDIO card support > MMC debugging`.
- If you can determine that the hang happened during resume:
  - Put a loop after am*3xx_resume_from_deep_sleep in sleepxxxx.S code and connect with JTAG
  - When stepping through code, watch where code jumps back to DDR. If the instructions are wrong/change from suspend to suspend, the DDR is likely being corrupted.
  - Check DDR signals, make sure EMIF context is restored properly, and if using VTT regulator, make sure it is being controlled properly. Both the AM335x-EVMSK and the AM437x-GPEVM have examples of controlling a VTT DDR3 regulator during suspend.

Other References

- Linux Core Power Management User's Guide
  - http://processors.wiki.ti.com/index.php/Linux_Core_Power_Management_User%27s_Guide
- AM335x CM3 Firmware Code
  - http://arago-project.org/git/projects/?p=am33x-cm3.git;a=summary
- AM335x TRM
  - http://www.ti.com/litv/pdf/spruh73e
- AM335x Datasheet
  - www.ti.com/am335x
- AM335x Sitara Power Management Training (Slides and Labs)
  - http://processors.wiki.ti.com/index.php/Sitara_Linux_Training:_Power_Management
- AM335x Linux Release
  - http://www.ti.com/tool/linuxezsdk-sitara
- AM335x Hardware Design Guide
  - http://processors.wiki.ti.com/index.php/AM335x_Hardware_Design_Guide