Save Power Ideas in Android Phone

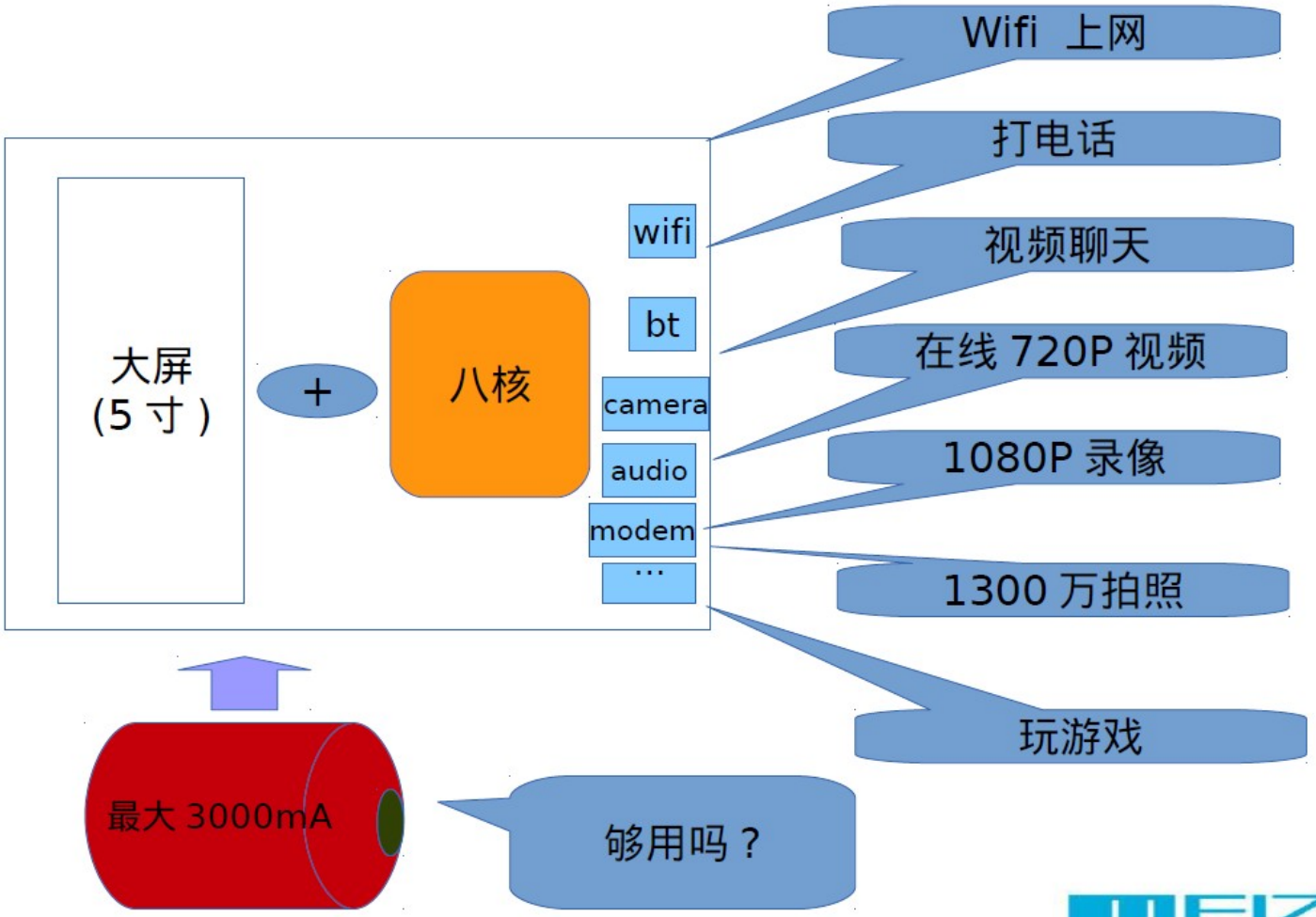
2014-01-15 Chen shaobo

Abstract

- Must be faced with the power problem
- Power Problem analysis
- How to save cpu power
- How to write save power program
- power tools
- Suggustions and need to do

1. Power Problem

Save power importance:



2. Analysis power problem

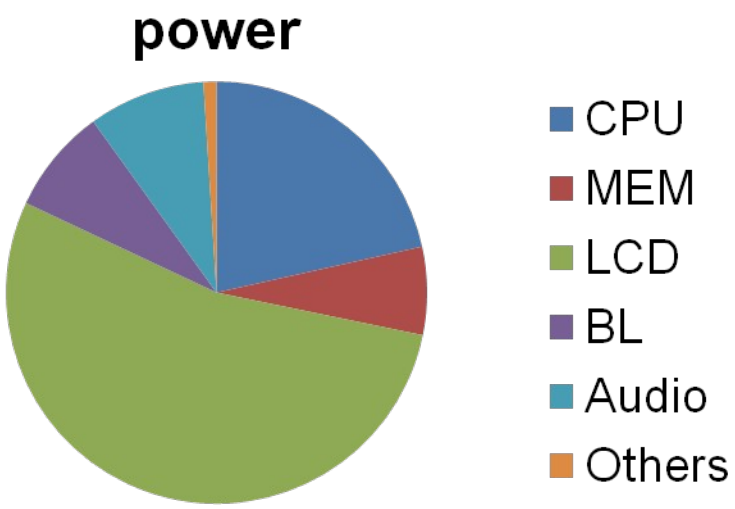
- Standby Power Consumption
- Idle Power Consumption
- Scenario Power Consumption
    1. music <*>
    2. video <*>
    3. camera
    4. telephone
    5. wifi online
    6. game
    7. so on

Power data in M65:

场景	版本	功耗 (mA)	备注
待机	U_12344	4.6	飞行模式
LPA	U_12344	14	
HOME	U_12344	137.8	默认背光 (100)
亮屏播放 mp3	U_12344	222	
灭屏播放 mp3	U_12344	69	
播放 720P	U_12344	252.4	播放测试视频
播放 1080p	U_12344	289	播放测试视频
wifi 上网	U_12344	498	模拟正常上网操作
拍照	U_12344	596.7	
录像 720P	U_12344	726.5	
录像 1080P	U_12344	1078	

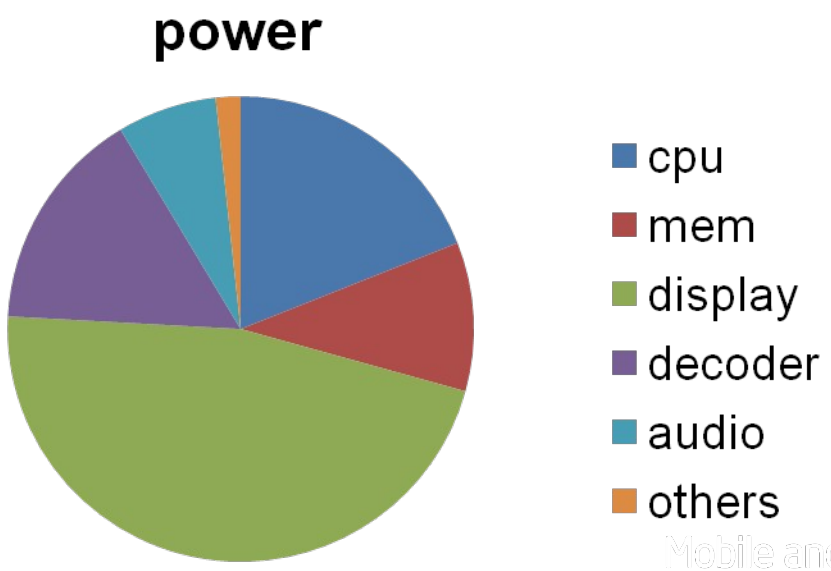
1. MP3 playback devices power comsume ：

Mp3 Playback	CPU	MEM	LCD	Backlight	Audio	Others
Power (mA)	48	15	120	18	20	2
Percentage	21.53%	6.74%	53.8%	8.07%	8.97%	1.00%



2. Mp4 video play devices power comsume:

mp4 1080p playback	cpu	mem	display	decoder	audio	others
Power	55	30	135	45	20	5
Percentage	18.96%	10.34%	46.55%	15.51%	6.89%	1.72%



The main power consumption devices is ?

- Display (LCD + Backlight)
- CPU
- Mem

3.How to save power

- Save Display power
  - -> PSR (Panel Self Refresh)
  - -> Backlight
- Save CPU power
  - -> cpufreq governor
  - -> cpu idle
- Save Bus Power
  - -> Mif bus freq
  - -> Int bus freq
- Reduce Power Leak (omit)

3.How to save display power

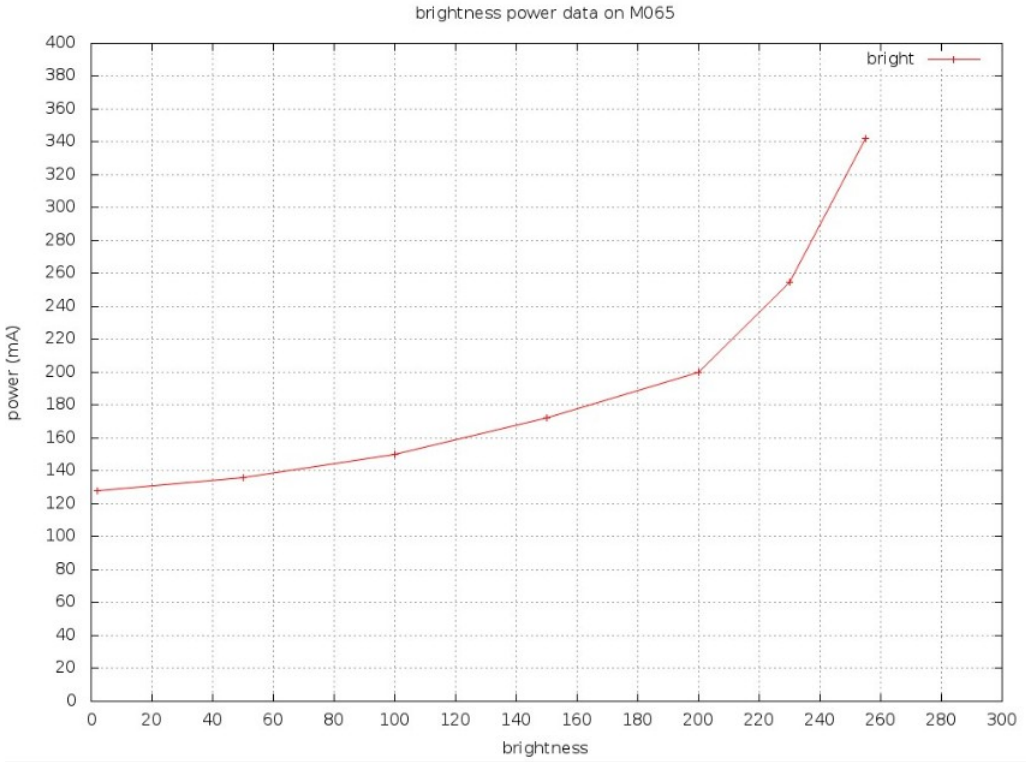
- **PSR (Panel Self Refresh)**

GPU refresh panal 60 per second, it's very important when display motion picture. but display static picture in unnessensary. PSR technology is for sovle this problem.

Display will enter DSR state When the screen is in static picture, the content will be copied to a panel of a roughly 8 MB capacity of small memory chips period. In this state, the 3D engine, video decoder, encoder and display pipes are powered off.

3.How to save display power

- **Brightness**



3.How to save cpu power

- cpu idle

  -> easy into C state and switch.

- cpufreq governor

  -> let cpu run into proper cpufreq level.

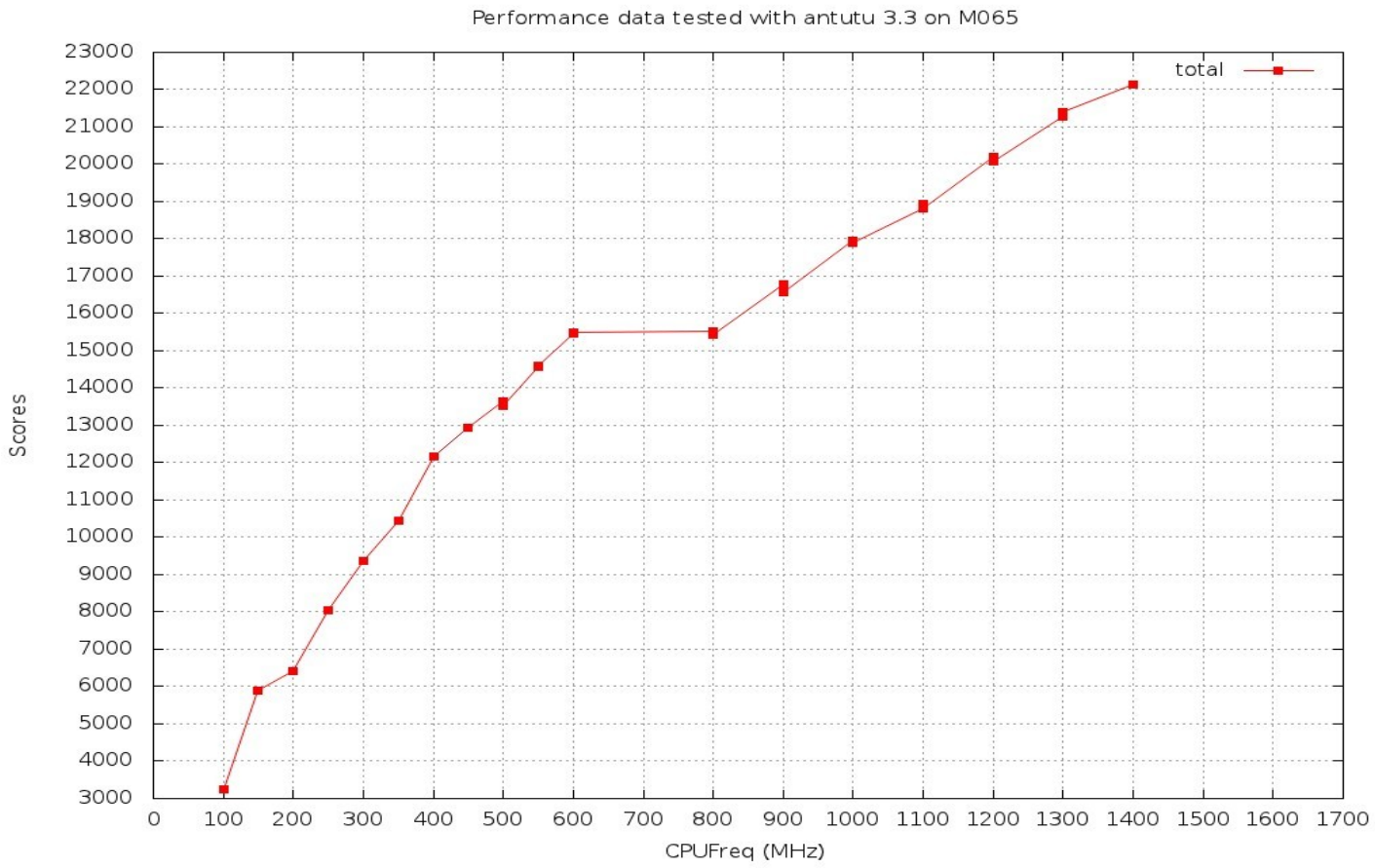
- cpu hotplug

  -> according to cpu load do cpu hotplug

CPU idle

- CPU state
  - -> Active: referred to cpu p-States (C0)
  - -> Idle: referred to cpu c-States
- CPU P states, cpu into C0
  - ->1600M
  - . . .
  - -> 100M
- CPU C states
  - -> C1: core clock off
  - -> C2: reduce voltage
  - -> C3: reduce voltage, flush partial L2 cache
- CPU Power mode
  - -> AFTR: ARM off top running
  - -> LPA: low power audio
  - ->deep-stop
  - -> sleep

P state Perf chart:



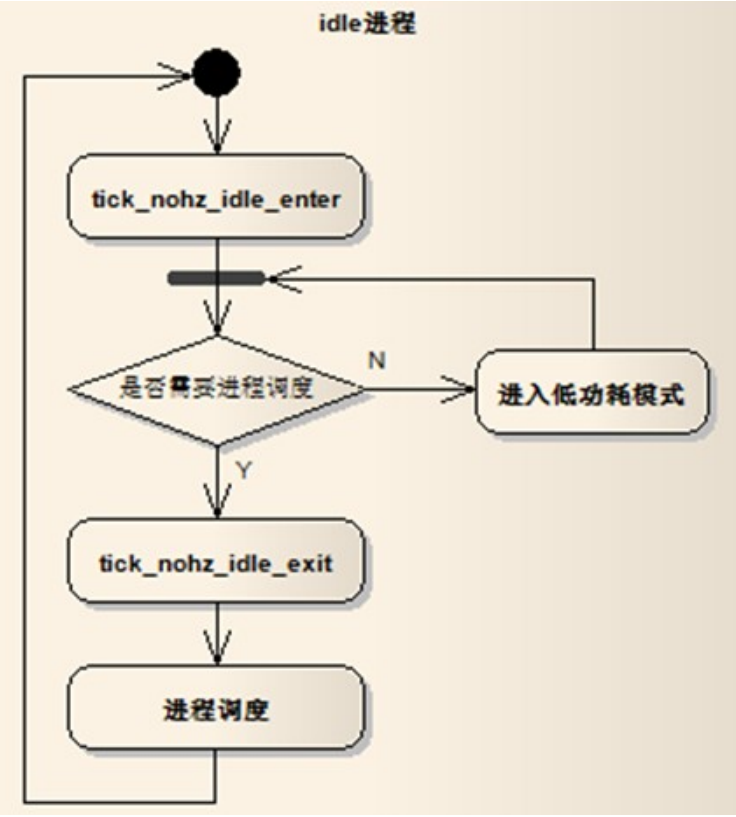
CPU Idle Governor

1. Governors implement the policy side of CPUIdle, More than one governor can be registered at the same time, Two different methods for picking the best state to enter.

- Ladder : a step-wise approach to selecting an idle state. Although this works fine with periodic tick-based kernels, this step-wise model will not work very well with tickless.
- Menu : looks at the expected sleep time (tickless) and then picks the deepest possible idle state straight away. It aims at getting maximum possible power advantage with little impact on performance.

Dynamic Clock:

内核一直使用周期性的基于 HZ 的 tick, 以产生定期的 tick 事件用于全局的时间管理(jiffies 和时间的更新），或者用于本地 cpu 的进程统计、时间轮定时器框架等 . 周期性时钟虽然简单有效 , 但在系统 idle 的情况下也要产生周期 tick, 会带来功耗问题。

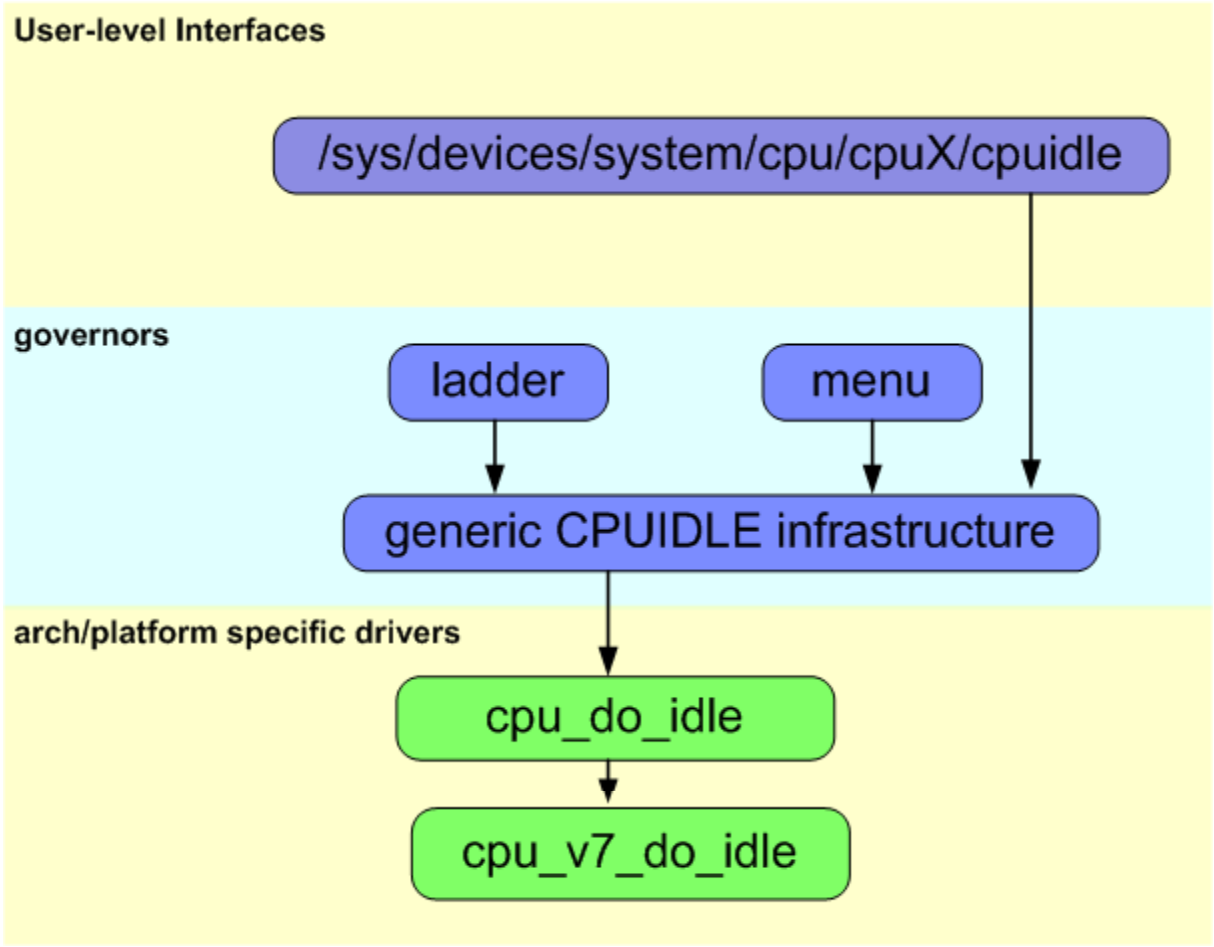


所谓动态时钟，就是当系统空闲时，它不是采用周期性的 tick 超时，而是判断系统的下一个超时处理时间。让时钟设备以单触发的方式，在指定的超时时间触发。

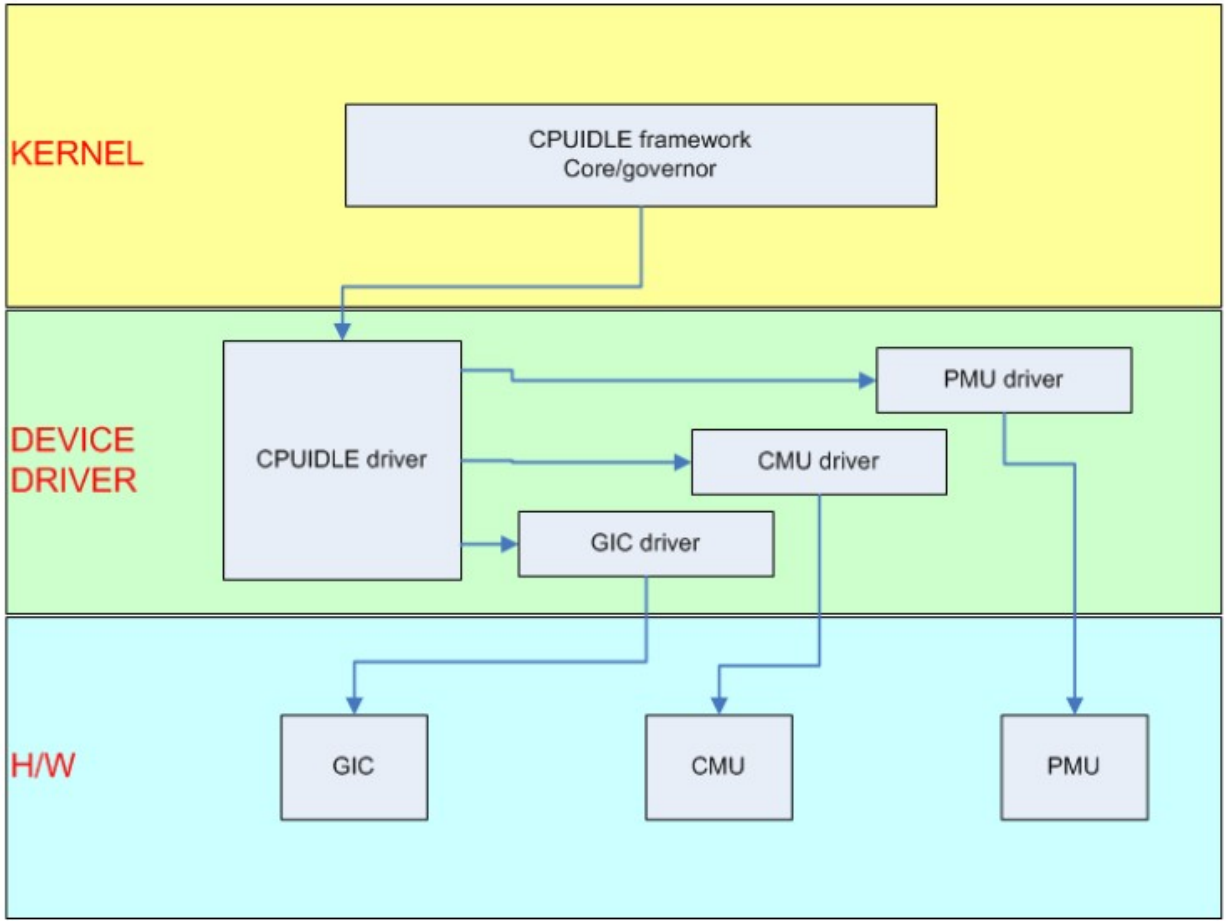
动态时钟开启后， tick_nohz_idle_enter 检测是否关闭周期时钟进入低功耗模式，当有中断事件使得 cpu 退出低功耗 idle 模式后，判断是否有新的进程被激活从而需要重新调度，如果需要则通过 tick_nohz_idle_exit 重新启用周期时钟，然后重新进行进程调度，等待下一次 idle 的发生。

cpu idle

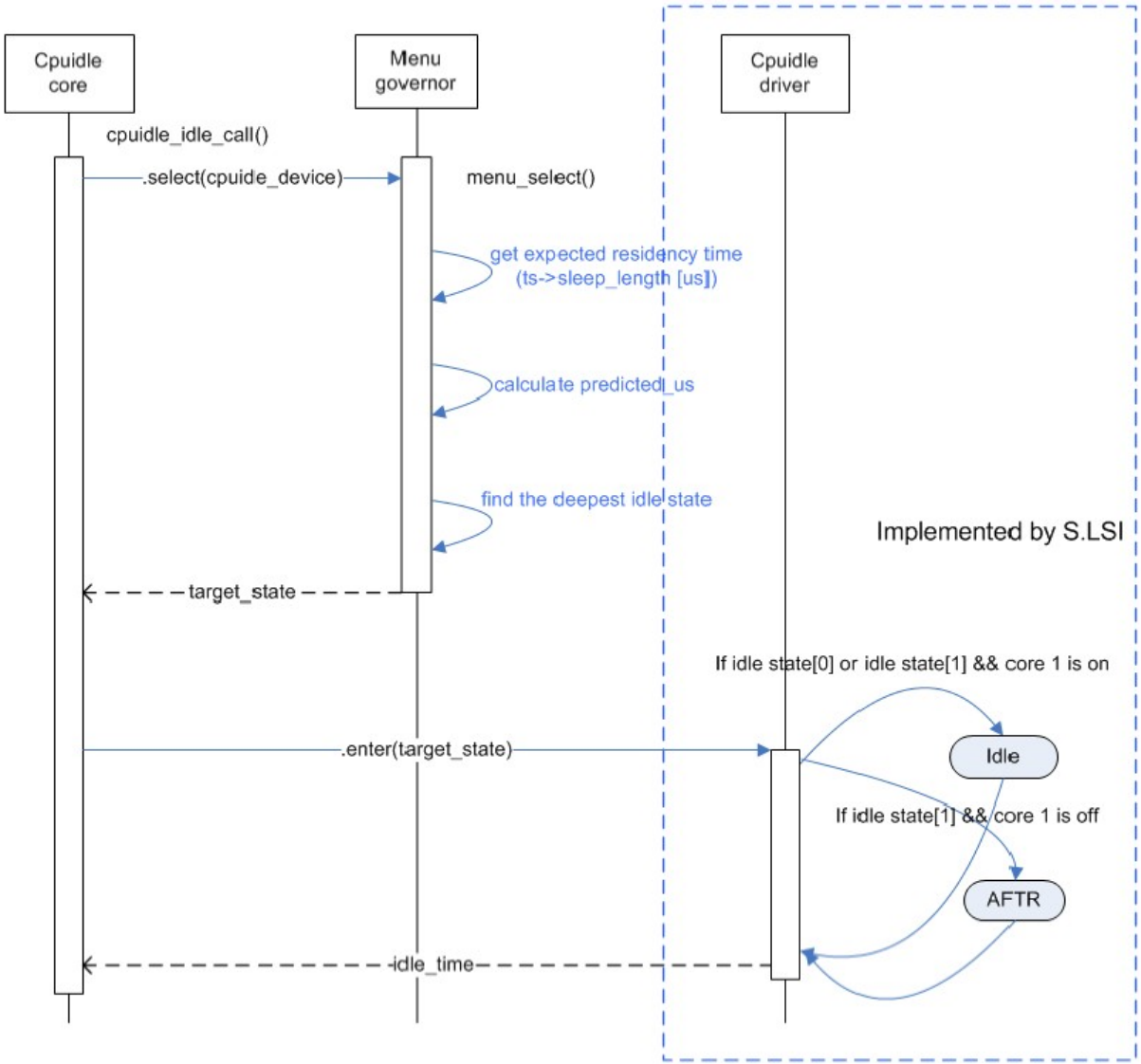
CPU idle power management framework:



cpu idle



GIC(globe interrupt control): Gic is a centralized resource that manages and supports interrupts in a system.

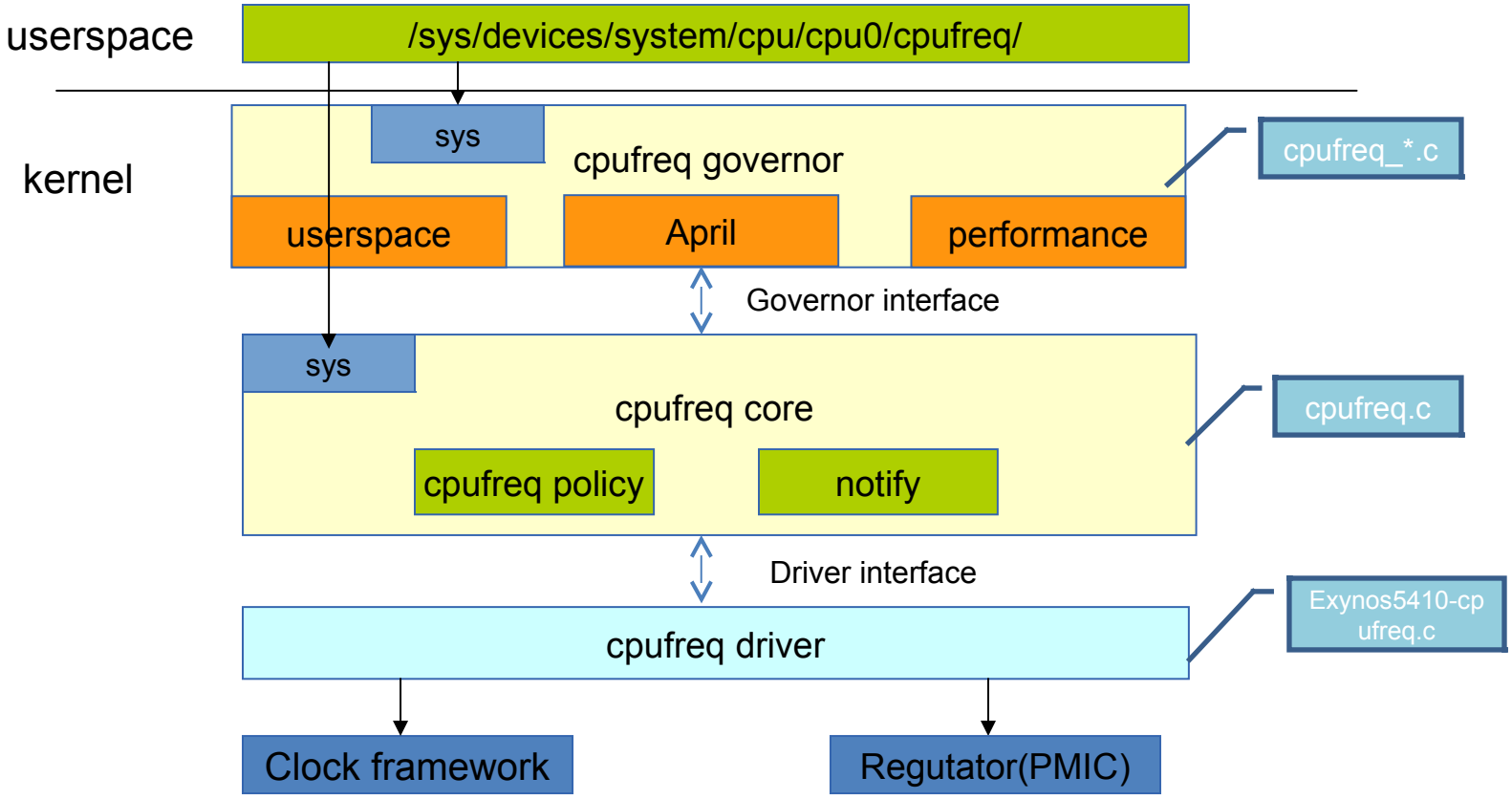


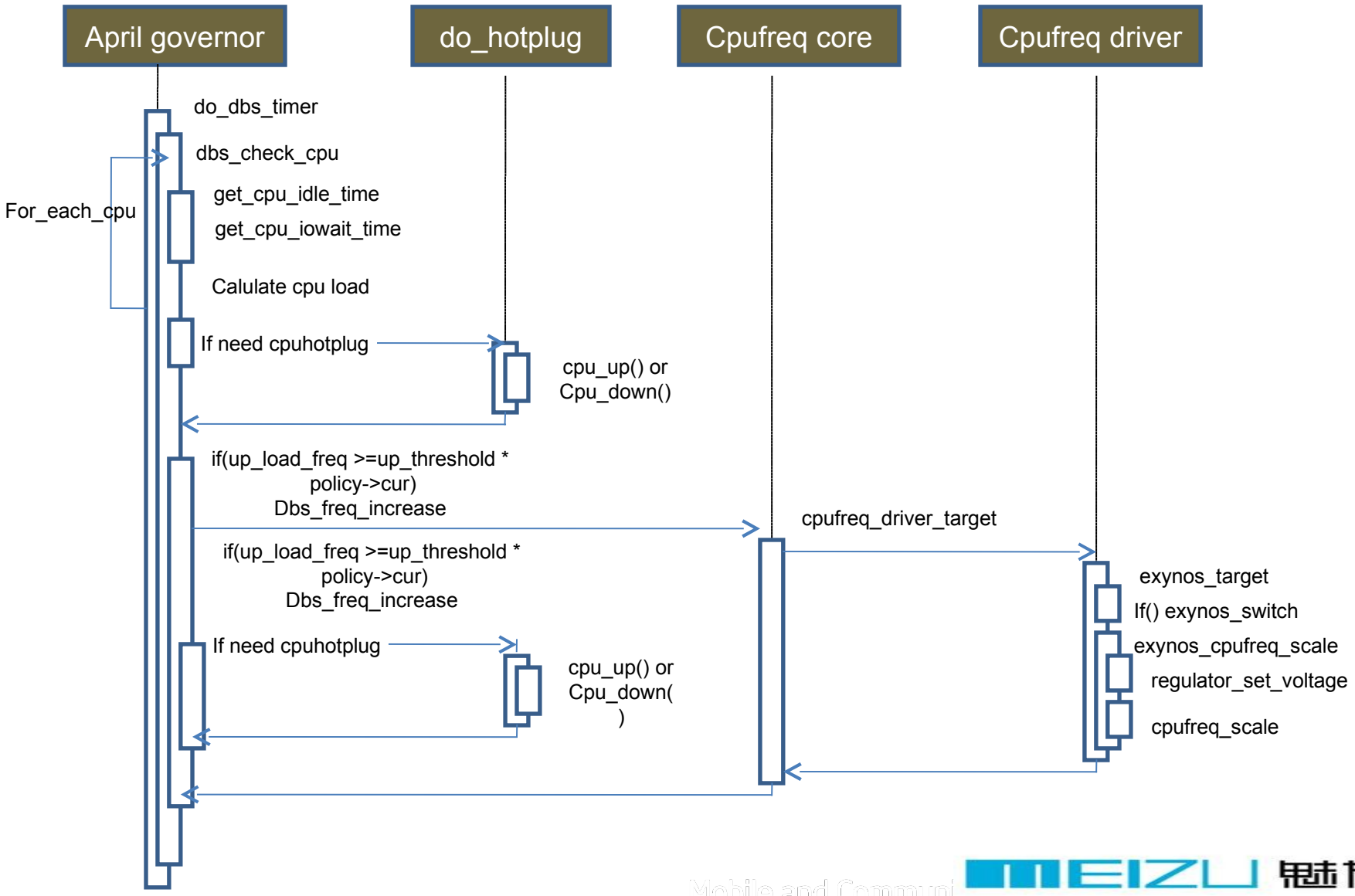
Cpufreq governor

1. Cpufreq Governors implement the policy side of dynamic adjustment cpufreq. More than one governor can be registered at the same time, provide /sys interface for switch governor ,defaut governor is april in M65.

NO	Gorvernor	Feature	Note
1	performance	Highest frequency for performance	
2	powersave	Lowest frequency for power	
3	userspace	User Manual set frequencies	
4	april	Samsung provide for exynos5 cpu	default

Cpufreq framework:





Reference governor perameter.

```
1. static unsigned int default_trans_costs[] = {
    40,    30,
    250000, 80,    40,
    400000, 85,    25,
    600000, 90,    20,
    800000, 80,    20,
    1000000, 95,   25,
    1300000, 100, 5,
    };
2. static unsigned int default_trans_steps[] = {
    200000,        SCALE_BOTH,
    600000,        SCALE_UP,
    800000,        SCALE_DOWN,
    1200000,        SCALE_UP,
};
3. static unsigned int default_trans_delays[] = {
    1, 4,
    600000, 1, 2,
    800000, 1, 3,
    1300000, 3, 1,
};
```

How to use cpufreq interface?

1. Change governor

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

2. Lock cpufreq

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

   echo 800000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed

3. Set min cpufreq

   echo 800000 >/sys/power/cpufreq_min_limit

4. Set max cpufreq

   echo 800000 >/sys/power/cpufreq_max_limit

5. Debug governor

   echo 1 >/sys/devices//system/cpu/cpufreq/april/april_debug

cpu hotplug

1.After the Linux 2.6.14, add the CPU hotplug support. CONFIG_HOTPLUG_CPU enables logical online/offline capability in the kernel,but physical addition/removal are need platform and CPU architectures permit partitioning support.

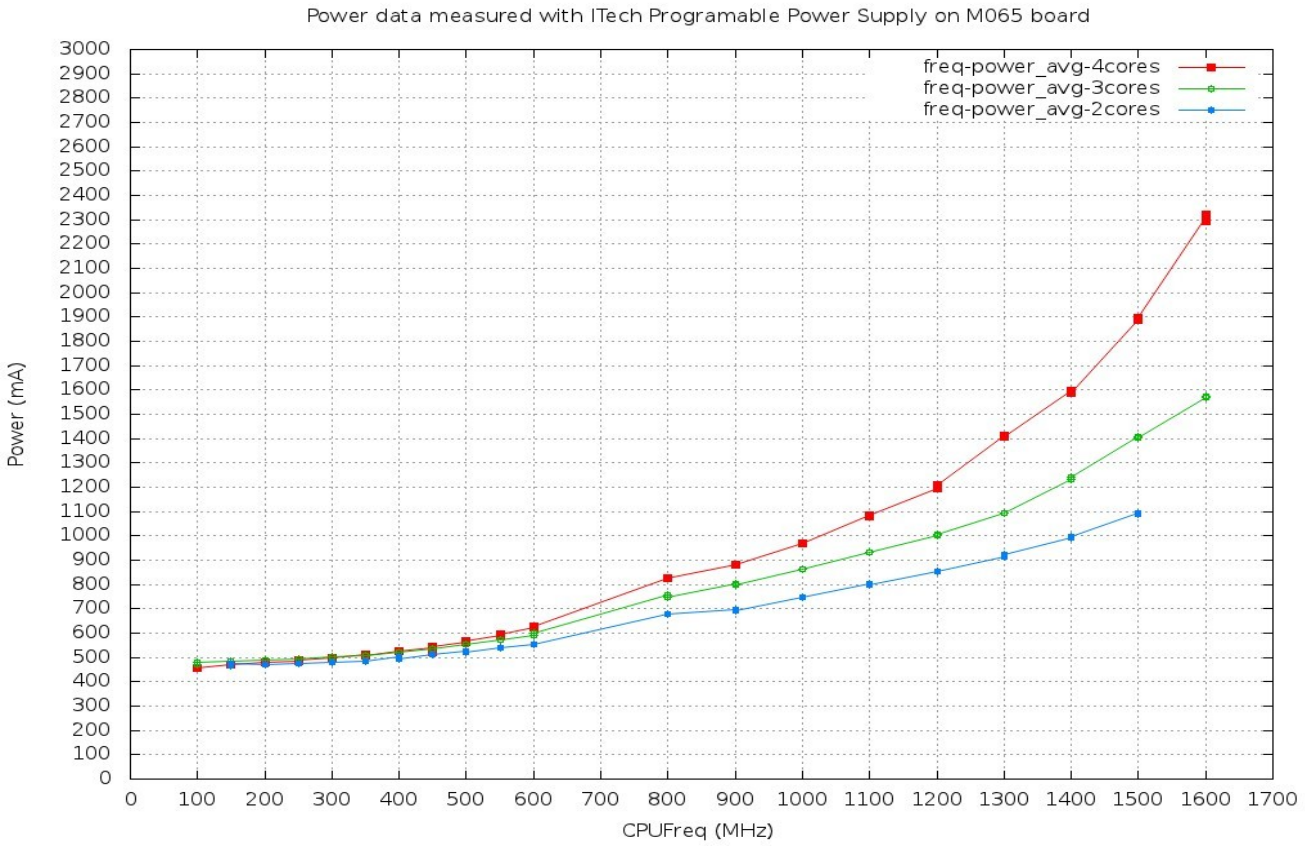
2.The CPU hotplug function changes the number of used processors dynamically. When a cpu hotplug out ,all All processes are migrated away from this CPU to new CPUs. All interrupts targeted to this CPU is migrated to a new CPU timers/bottom half/task lets are also migrated to a new CPU Send cpu_dead event for successful cleanup.

3. Cpu hotplug user interface

```
cat  /sys/devices/system/cpu/online       //online 的 cpu 号
cat  /sys/devices/system/cpu/cpu1/online      // 1 是 online
```

4.2 cpu hotplug

why hotplug?



cpu hotplug rules

- Hotplug in
  1. screen off && 1400M < current cpufreq <250M
  2. 800M < current cpufreq <250M
  3. current cpufreq > 800M, freq_next < 1400M

- Hotplug out
  1. current cpufreq>=1400M && cpu[i] load < 20%
  2. screen off && current cpufreq<=150M && total_load_freq< ((dbs_tuners_ins.up_threshold – dbs_tuners_ins.down_differential) * policy->cur) &&15 consecutive times.

3. How to save Mem power

User interface:
/sys/devices/platform/exynos5-busfreq-int/devfreq/exynos5-busfreq-int.
/sys/devices/platform/exynos5-busfreq-mif/devfreq/exynos5-busfreq-mif
Governor: simple_usage

```
struct mif_bus_opp_table {
        unsigned int idx;
        unsigned long clk;
        unsigned long volt;
        cputime64_t time_in_state;
};

struct mif_bus_opp_table mif_bus_opp_list[] = {
        {LV_0, 800000, 1062500, 0},
        {LV_1, 667000, 1062500, 0},
        {LV_2, 533000, 1062500, 0},
        {LV_3, 400000, 1062500, 0},
        {LV_4, 267000, 1062500, 0},
        {LV_5, 200000, 1062500, 0},
        {LV_6, 160000, 1062500, 0},
        {LV_7, 100000, 1062500, 0},
};

static struct devfreq_dev_profile exynos5_mif_devfreq_profile = {
        .initial_freq   = 800000,
        .polling_ms     = 100,
        .target         = exynos5_mif_busfreq_target,

.get_dev_status=exynos5_mif_bus_get_dev_status
};
```

```
struct int_bus_opp_table {
        unsigned int idx;
        unsigned long clk;
        unsigned long volt;
        cputime64_t time_in_state;
};

struct int_bus_opp_table int_bus_opp_list[] = {
        {LV_0, 800000, 1137500, 0},          /* ISP
Special Level */
        {LV_1, 700000, 1137500, 0},          /* ISP
Special Level */
        {LV_2, 400000, 1137500, 0},
        {LV_3, 267000, 1137500, 0},
        {LV_4, 200000, 1137500, 0},
        {LV_5, 160000, 1137500, 0},
        {LV_6, 100000, 1137500, 0},
        {LV_7,  50000, 1137500, 0},
};
```

3. How to save bus power

User interface:
/sys/devices/platform/exynos5-busfreq-int/devfreq/exynos5-busfreq-int.
/sys/devices/platform/exynos5-busfreq-mif/devfreq/exynos5-busfreq-mif
Governor: simple_usage

```
struct mif_bus_opp_table {
        unsigned int idx;
        unsigned long clk;
        unsigned long volt;
        cputime64_t time_in_state;
};

struct mif_bus_opp_table mif_bus_opp_list[] = {
        {LV_0, 800000, 1062500, 0},
        {LV_1, 667000, 1062500, 0},
        {LV_2, 533000, 1062500, 0},
        {LV_3, 400000, 1062500, 0},
        {LV_4, 267000, 1062500, 0},
        {LV_5, 200000, 1062500, 0},
        {LV_6, 160000, 1062500, 0},
        {LV_7, 100000, 1062500, 0},
};

static struct devfreq_dev_profile exynos5_mif_devfreq_profile = {
        .initial_freq   = 800000,
        .polling_ms     = 100,
        .target         = exynos5_mif_busfreq_target,

.get_dev_status=exynos5_mif_bus_get_dev_status
};
```

```
struct int_bus_opp_table {
        unsigned int idx;
        unsigned long clk;
        unsigned long volt;
        cputime64_t time_in_state;
};

struct int_bus_opp_table int_bus_opp_list[] = {
        {LV_0, 800000, 1137500, 0},          /* ISP
Special Level */
        {LV_1, 700000, 1137500, 0},          /* ISP
Special Level */
        {LV_2, 400000, 1137500, 0},
        {LV_3, 267000, 1137500, 0},
        {LV_4, 200000, 1137500, 0},
        {LV_5, 160000, 1137500, 0},
        {LV_6, 100000, 1137500, 0},
        {LV_7,  50000, 1137500, 0},
};
```

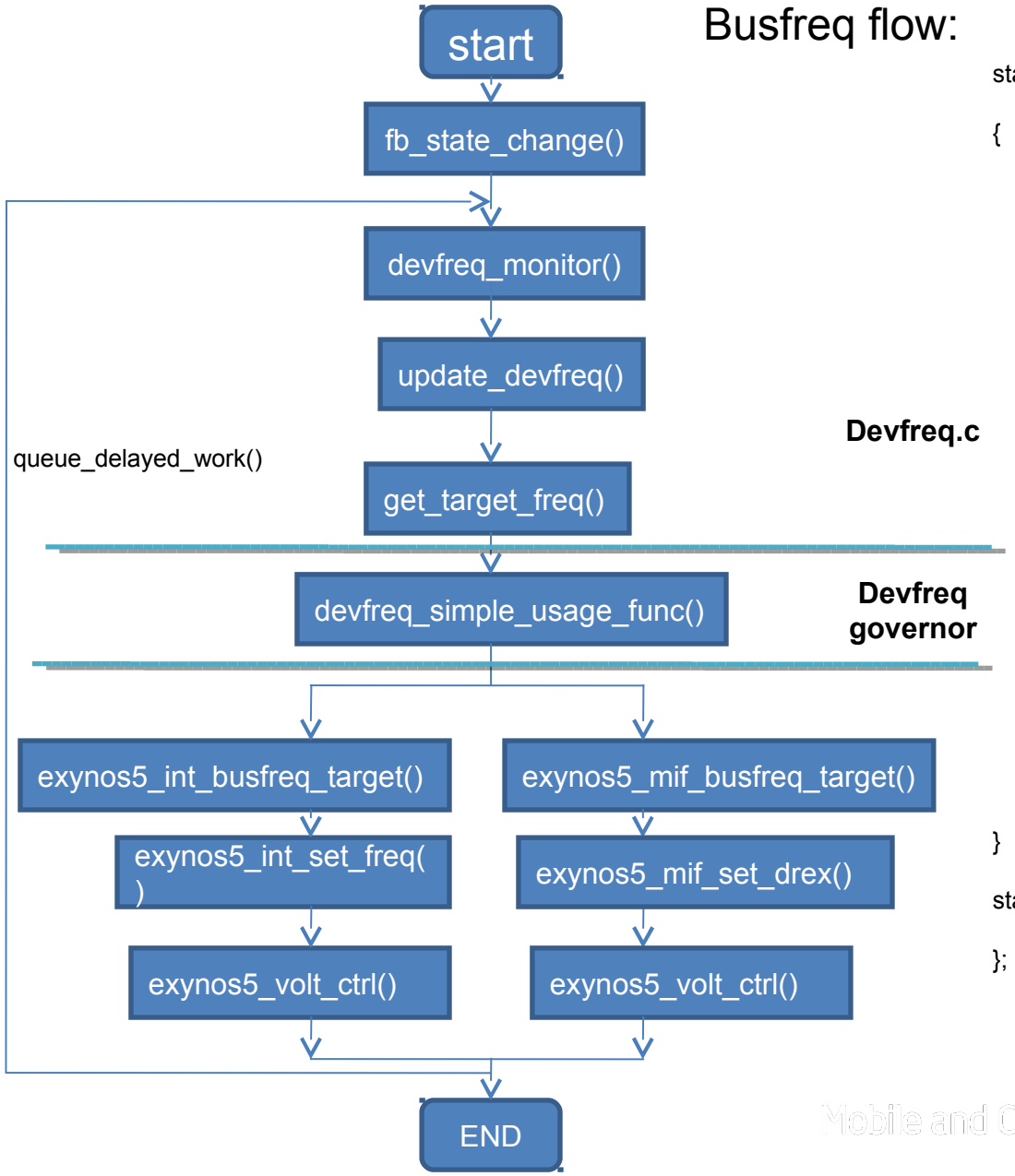
Busfreq-Int 实现：

User interface:/sys/devices/platform/exynos5-busfreq-int/devfreq/exynos5-busfreq-int.
Governor: simple_usage

```
struct int_bus_opp_table {
        unsigned int idx;
        unsigned long clk;
        unsigned long volt;
        cputime64_t time_in_state;
};

struct int_bus_opp_table int_bus_opp_list[] = {
        {LV_0, 800000, 1137500, 0},
                /* ISP Special Level */
        {LV_1, 700000, 1137500, 0},
                /* ISP Special Level */
        {LV_2, 400000, 1137500, 0},
        {LV_3, 267000, 1137500, 0},
        {LV_4, 200000, 1137500, 0},
        {LV_5, 160000, 1137500, 0},
        {LV_6, 100000, 1137500, 0},
        {LV_7,  50000, 1137500, 0},
};

static struct devfreq_dev_profile exynos5_mif_devfreq_profile = {
        .initial_freq   = 800000,
        .polling_ms     = 100,
        .target         = exynos5_mif_busfreq_target,

.get_dev_status=exynos5_mif_bus_get_dev_status
};
```

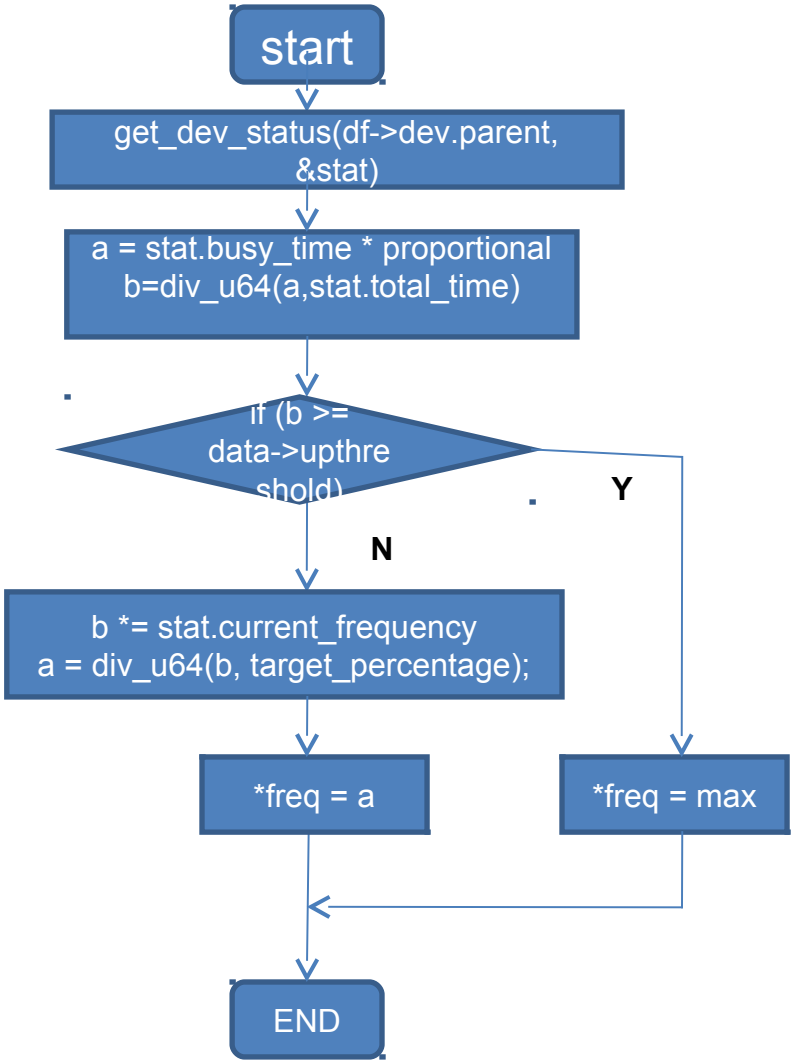
```
static struct devfreq_simple_usage_data
exynos5_int_governor_data = {
        .upthreshold            = 80,
        .target_percentage      = 70,
        .proportional           = 100,
        .cal_qos_max            = 400000,
        .pm_qos_class           =

PM_QOS_DEVICE_THROUGHPUT,
        .en_monitoring          = true,
};
        struct nocp_info *exynos5_int_nocp_list[] = {
                &nocp_mfc0,
                &nocp_mfc1,
                &nocp_gsc23,
                &nocp_isp0,
                &nocp_isp1,
                &nocp_gen,
                &nocp_gsc0,
                &nocp_gsc1,
                &nocp_disp0,
                &nocp_disp1,
                &nocp_fsys,
        };
```

Busfreq flow:




```
static int fb_state_change(struct notifier_block *nb,
                                unsigned long val, void *data)
{
        struct fb_event *evdata = data;
        unsigned int blank;

        if (val != FB_EVENT_BLANK)
                return 0;

        blank = *(int *)evdata->data;

        switch (blank) {
        case FB_BLANK_POWERDOWN:
                lcd_is_on = false;
                resched_devfreq_monitor();
                break;

        case FB_BLANK_UNBLANK:
                lcd_is_on = true;
                resched_devfreq_monitor();
                break;
        default:
                break;
        }

        return NOTIFY_OK;
}

static struct notifier_block fb_block = {
        .notifier_call = fb_state_change,
};
```

simple_usage flow:




```
const struct devfreq_governor
devfreq_simple_usage = {
                .name = "simple_usage",
                .get_target_freq =
                devfreq_simple_usage_func,
                .init = devfreq_simple_usage_init,
                .exit = devfreq_simple_usage_exit,
};

 static struct devfreq_simple_usage_data
 exynos5_mif_governor_data = {
                .upthreshold            = 85,
                .target_percentage      = 80,
                .proportional           = 100,
                .cal_qos_max            = 800000,
                .pm_qos_class           =

 PM_QOS_BUS_THROUGHPUT,
                .en_monitoring          = true,
 };
   static struct devfreq_simple_usage_data
   exynos5_int_governor_data = {
                .upthreshold            = 80,
                .target_percentage      = 70,
                .proportional           = 100,
                .cal_qos_max            = 400000,
                .pm_qos_class           =

   PM_QOS_DEVICE_THROUGHPUT,
                .en_monitoring          = true,
   };
```

4. writing power efficient program

1. Timers & Interrupts
   - Thou shall not Poll
   - Thou shall not “busy wait”
   - Thou shall reduce Interrupts
   - Thou shall use timers effectively
   -
2. Threading Model
   - Thou shall use multithreading efficiently

3. Memory Management
   - Thou shall improve cache locality

4. Data Efficiency
   Thou shall program “power-smart”

5. Power Awareness
   - Thou shall be “power-aware”

Timers & interrupts

1. Thou shall NOT Poll

Avoid Polling
Subscribe to events and wait for events to happen
Bad examples:
-checking if the mouse moved, key pressed .. once per second
-checking if the sound volume changed .. 10 times per second
-checking to smartcard reader got inserted on USB ... 10 times per second

2. Thou shall NOT “busy wait”

Minimize the use of ‘tight loops’
Eliminate spinning loops, reduce spin lock.
Convert polling loops to be ‘event driven’

3. Thou shall reduce Interrupts

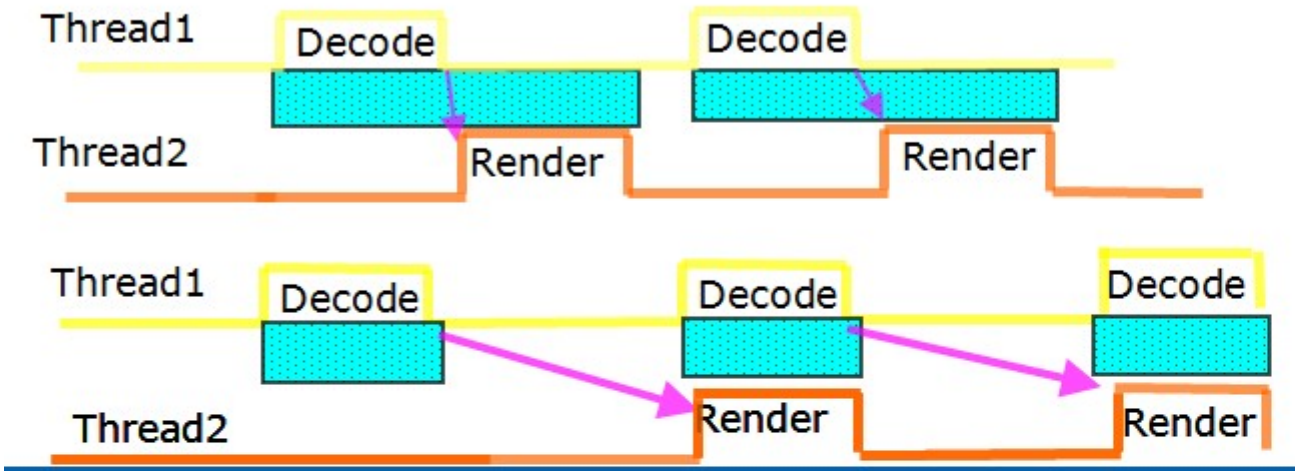
Avoid waking up the CPU

-An interrupt will wake the CPU if it was sleeping

Defer interrupts to the maximum delay tolerable

-Synchronize multiple threads to interrupt at the same time

Example: Align the two threads so that on Cpu wakeup both threads, reduce cpu wakeup.



4. Thou shall use timers effectively

Timers are needed for house-keeping

Reduce usage of high-resolution periodic timers

- Do you really need a periodic timer event every 10 ms?
- Disable periodic timers when not in use

Group timers effectively

-In kernel space : Use timer APIs in application

```
Deferrable timers & Init_timer_deferrable
round jiffies & schedule_hrtimeout_range &
usleep_range
```

Deferrable timers :

In the kernel to use timer for precise timeout value is not sensitive, sooner or later execution not very important. So we can using deferrable timers, it can think of these is not sensitive to time timeout at the same time gather together, can further reduce the number of CPU wakeups, achieve energy saving purpose.

Threading Model

1. Thou shall use multithreading efficiently

Use multiple threads to speed up execution
Balance the threads
-Synchronize threads to work on different cores simultaneously and idle simultaneously
-Shut down threads if they are not doing anything
-Idle threads should be interrupt driven
-Block a thread on read instead of polling select/poll

Memory Managerment

1. Thou shall improve cache locality

Operate on small data at one time so data stays in caches
- Improves performance
- Reduces power
- Less bus activity
- Less memory traffic

2. Thou shall manage memory efficiently

Get more memory than you need
- Saves time searching for memory at run time

Pre-Allocate in larger chunks for creating big buffers
- Avoids data movement between disk/memory and cache

IF memory is needed at run time: allocate in smaller chunks:
1 page vs. multiple contiguous pages

Data Efficency

1. Thou shall program “power-smart”

Fast execution

- Handle CPU sleep transitions effectively
  - Be effective in handling suspend-resume
- Use high performance algorithms and data structures
  - Use performance libraries
  - Avoid recursion: may be power inefficient
- Compiler optimized for the most common execution path

Power awareness

1. Don’t let application do actions when in background
2. Update graphics less when running on battery
3. Disable animations when running on battery
4. Update window content only when necessary
5. When running on battery use “low-power” algorithms
6. Reduce read/write to storage in battery mode
7. Let you code run in low cpufreq

5. Power tools

1.Measure Tools

-> IT6322B Programmable Power

http://redmine.meizu.com/issues/16222

2. Test Evernorment

-> Atest test framework (git@172.16.10.242:users/falcon/atest.git)

-> event_record, event_replay

3. Analysis Tools

-> powertop

-> ftrace

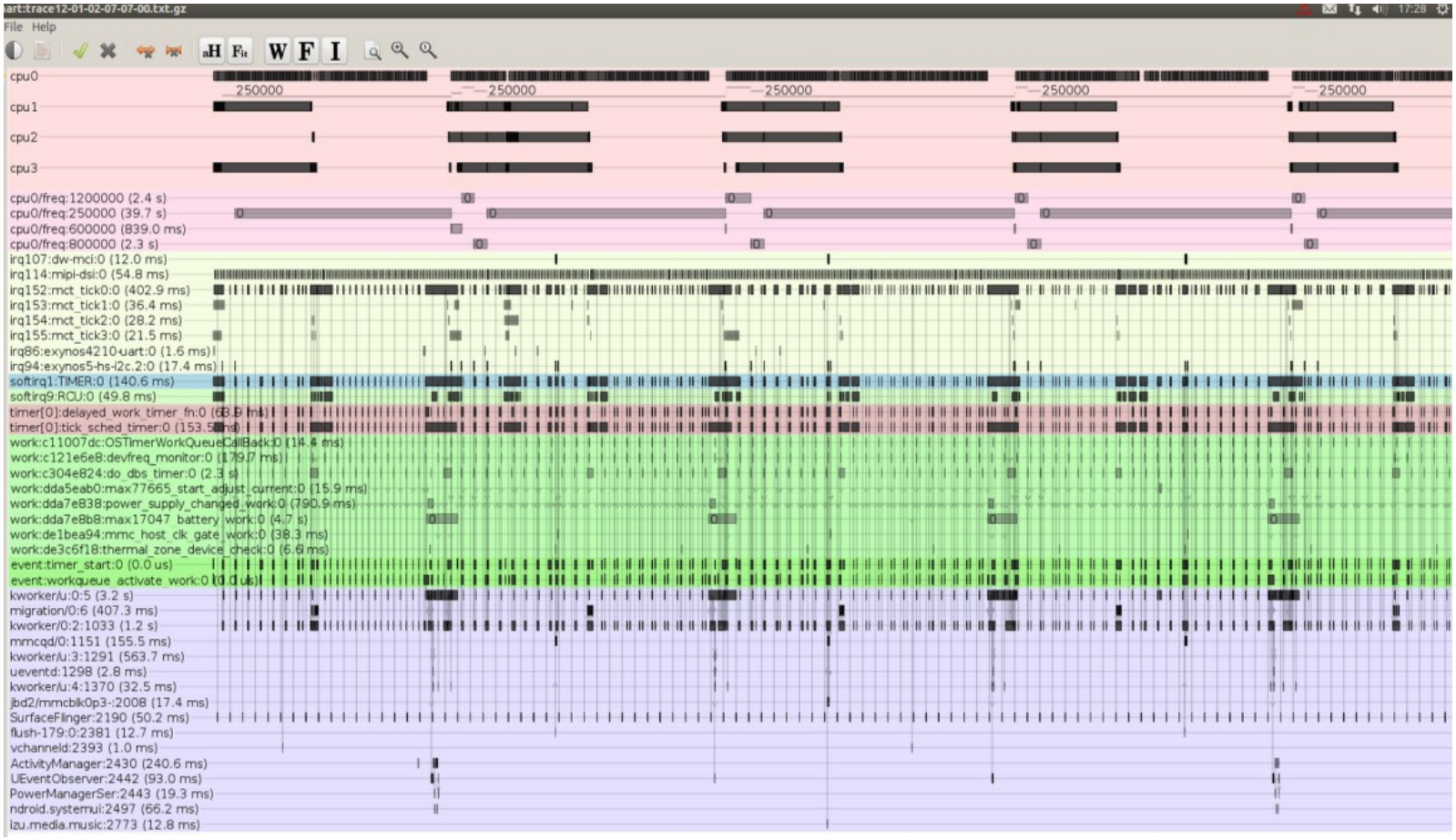
-> vmstate

-> sysrq-trigger

-> pytimechart

http://redmine.meizu.com/projects/2013-m65-bsp/wiki

pytimechart 分析 idle 下功耗：



pytimechart 分析打开应用：

4.4 应用的优化

1. 播放 mp4 的状态 bar

在进行 mp4 播放时 , 使用 overlay 来显示会节省 power, 但使用 overlay 有一些条件限制 , 比如合并的图层不能超过二个 . 早期的版本播放 mp4 会比三星 S4 高出许多 , 最近发现 S4 只有二个图层使用了 overlay,flyme 播放器多了一个 status Bar 图层 , 通过 dumpsys SurfaceFlinger 命令发现三个图层使用的 hwcomposer, 而且没有用到 overlay, 造成 power 的增加 .

2. idle 状态的 system UI

在 home 界面下 , 发现 power 波形跳动很大 , 通过 pytimechart 抓出 trace 数据 , 发现 laucher 进程经常引起 cpu 调频到 1200M, 造成了功耗的突出跳动 , 和 shell team 沟通 , 换了一个 launch 的实现方式 , 就没有 laucher 进程经常引起功耗跳动情况 .

3. 对一些拉圾应用的处理

在网上下载的有些应用 , 做得很拉圾 , 会引起 cpu 占用率 100% 的情况和定时唤醒 , 更新 , 自动下载等操作 , 引起了 cpu 的唤醒或调频 , 引起不必要的 power
消耗 .

5. Power tools

1.Measure Tools

-> IT6322B Programmable Power

http://redmine.meizu.com/issues/16222

2. Test Evernorment

-> Atest test framework (git@172.16.10.242:users/falcon/atest.git)

-> event_record, event_replay

3. Analysis Tools

-> powertop

-> ftrace

-> vmstate

-> sysrq-trigger

-> pytimechart

http://redmine.meizu.com/projects/2013-m65-bsp/wiki

6. Suggestions and need to do

- Suggestions
  -> Must highlight power, Enhance power awareness.
  -> Opt code reduce cpu utilization rate, interrupt,wakeup.
  -> reduce unnessensary power leak.
  -> Add save power setting.
  -> Add cpufreq manager servers, App ask for cpufreq.
- Need to do
  -> power auto test and weekly power test report.
  -> set Scenario Power test rules.
  -> make a power data target and acceptable ranges.
  -> App funtion power data detailed,opt high power funtions
  -> Alarm wakeup in sleep status.
  -> Garbage APK update, broadcast, wakeup processing.

Thanks !