DVFS

# 目录

/arch/arm64/boot/dts/rockship/rk3399.dtsi

包含关系

<https://blog.csdn.net/congyingleng3402/article/details/100496002>

# 基本介绍

## 名词解释

DVFS: DVFS 即动态电压频率调整，Dynamic voltage and frequency scaling,动态技术则是根据芯片所运行的应用程序对计算能力的不同需要，动态调节芯片的运行频率和电压(对于同一芯片，频率越高，需要的电压也越高)，从而达到节能的目的。

[Mali (GPU)](https://zh.wikipedia.org/wiki/Mali_(GPU))：Mali 是一款由ARM Holdings（ARM，安谋科技）研发设计的移动显示芯片组（GPUs）系列。该显示芯片组的电路设计和架构研发完全由ARM自主设计，ARM特别设立了ARM Norway（ARM挪威）显示处理事业部来负责研发设计ARM Mali显示芯片系列。

RK3288: 芯片内嵌的最新一代和最强大的GPU（Mali-T764）能顺利支持高分辨率（3840X2160）显示和主流游戏。支持OpenVG1.1，OpenGL的ES1.1/2.0/3.0，OpenCL1.1，RenderScript以及DirectX11等，在3D效果方面相对同类产品有较大的提升。

## 基本原理

降低频率可以降低功率，但是单纯地降低频率并不能节省能量。因为对于一个给定的任务，F\*t是一个常量，只有在降低频率的同时降低电压，才能真正地降低能量的消耗。 目前许多芯片支持DVFS，比如InteI公司的芯片支持SpeedStep，ARM的支持IEM(Intelligent Energy Man-ager)和AVS(Adaptive Voltage Scaling)等。但是要让DVFS发挥作用，真正地实现节能，只有芯片的支持还是不够的，还需要软件与硬件的综合设计。目前S3C6410开发板中，重庆海特克的开发板有DVFS设计，说明了他们对算法预测这方面有很大的突破。S3C6410芯片本身针对的是手持设备，如果有DVFS这项技术的辅助，功耗这方面将会降的更低，对手持设备研发帮助将会更大。

DVFS系统流程：

1. 采集与系统负载有关的信号，计算当前的系统负载。

2. 根据系统的当前负载，预测系统在下一时间段需要的性能。

3. 将预测的性能转换成需要的频率，从而调整芯片的时钟设置。

4. 根据新的频率计算相应的电压。通知电源管理模块调整给CPU的电压。另外，在调整频率和电压时，要特别注意调整的顺序。当频率由高到低调整时，应该先降频率，再降电压；相反，当升高频率时，应该先升电压，再升频率。虽然现在做DVFS的不是很多，是因为很多都被预测算法给难住，但是作者相信，随着预测算法的进步，DVFS技术必将得到广泛的应用，因为它能够节省很多能量。而节能对许多便携式设备来说，常常是第一要求。

## Android dvfs

dvfs一共有3处  
cpu，gpu，ddr

# RK

cpu和gpu都是默认开启的  
ddr dvfs代码还没有进行合并,现在是一直工作在533mhz

## CPU

## GPU

Fpv锁频

#cat /sys/devices/ffa30000.gpu/dvfs

mali\_dvfs is OFF

current\_gpu\_clk\_freq : 99 MHz

#cat /sys/devices/ffa30000.gpu/dvfs

mali\_dvfs is OFF

current\_gpu\_clk\_freq : 594 MHz

## 操作

查看cpu可用频点

cat /sys/devices/system/cpu/cpu0/cpufreq/scaling\_available\_frequencies

126000 216000 312000 408000 600000 696000 816000 1008000 1200000 1416000 1512000 1608000

设置cpu最高频率

echo 1608000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling\_max\_freq  
设置cpu最低频率

echo 216000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling\_min\_freq

cat /sys/kernel/debug/clk/clk\_summary

# 实现原理

## GPU

Platform: Rockchip

mali\_device\_driver 分为两个部分 : platform\_dependent\_part 和 common\_parts, 参见 mali\_kbase\_config\_rk.c 开头部分的注释.

gpu dvfs核心控制在mali\_kbase\_dvfs.c中.

**[cpp]** [view plain](https://blog.csdn.net/qq1028850792/article/details/79346153) [copy](https://blog.csdn.net/qq1028850792/article/details/79346153)

1. /\*\*
2. \* .DP : mali\_dvfs\_level\_table.
3. \* 其中的 level\_items 的 gpu\_clk\_freq 从低到高.
4. \*
5. \* 运行时初始化阶段, 将从 'mali\_freq\_table' 进行运行时初始化,
6. \* 若获取 'mali\_freq\_table' 失败, 则使用这里的 缺省配置.
7. \* 参见 kbase\_platform\_dvfs\_init.
8. \*/
9. **static** mali\_dvfs\_info mali\_dvfs\_infotbl[] = {
10. {925000, 100000, 0, 70, 0},
11. {925000, 160000, 50, 65, 0},
12. {1025000, 266000, 60, 78, 0},
13. {1075000, 350000, 65, 75, 0},
14. {1125000, 400000, 70, 75, 0},
15. {1200000, 500000, 90, 100, 0},
16. };

mali\_dvfs\_infotbl定义gpu所有可变的freq level以及对应的min utilisation和max utilisation.

不过它会被dts中的operating-points所覆盖

Rk3288.dts

clk\_gpu\_dvfs\_table: clk\_gpu {

operating-points = <

/\* KHz uV \*/

200000 1200000

300000 1200000

400000 1200000

>;

};

Kpad.dts实现自定义

gpu dvfs运行时, common parts (metrics system)会周期地(20ms)通知回调 platform\_dependent\_part 中的函数 kbase\_platform\_dvfs\_event(), 并传入当前的 gpu utilization.

dvfs\_callback -> kbase\_pm\_get\_dvfs\_action -> kbase\_platform\_dvfs\_event

**[cpp]** [view plain](https://blog.csdn.net/qq1028850792/article/details/79346153) [copy](https://blog.csdn.net/qq1028850792/article/details/79346153)

1. mali\_kbase\_dvfs.c
2. // callback\_interface\_to\_common\_parts\_in\_mdd
4. /\*\*
5. \* 由 common\_parts\_in\_mdd 调用的, 将 dvfs\_event (utilisation\_report\_event) 通知回调到 platform\_dependent\_part\_in\_mdd.
6. \*/
7. **int** kbase\_platform\_dvfs\_event(**struct** kbase\_device \*kbdev,
8. u32 utilisation,          // current\_calculated\_utilisation
9. u32 util\_gl\_share\_no\_use,
10. u32 util\_cl\_share\_no\_use[2] )
11. {
12. unsigned **long** flags;
13. **struct** rk\_context \*platform;
15. BUG\_ON(!kbdev);
16. platform = (**struct** rk\_context \*)kbdev->platform\_context;
18. spin\_lock\_irqsave(&mali\_dvfs\_spinlock, flags);
19. **if** (platform->time\_tick < MALI\_DVFS\_UP\_TIME\_INTERVAL) {
20. platform->time\_tick++;
21. platform->time\_busy += kbdev->pm.backend.metrics.time\_busy;
23. platform->time\_idle += kbdev->pm.backend.metrics.time\_idle;
24. } **else** {
25. platform->time\_busy = kbdev->pm.backend.metrics.time\_busy;
26. platform->time\_idle = kbdev->pm.backend.metrics.time\_idle;
27. platform->time\_tick = 0;
28. }
30. **if** ((platform->time\_tick == MALI\_DVFS\_UP\_TIME\_INTERVAL) &&
31. (platform->time\_idle + platform->time\_busy > 0))
32. platform->utilisation = (100 \* platform->time\_busy) /
33. (platform->time\_idle + platform->time\_busy);
35. /\* 记录 current\_calculated\_utilisation. \*/
36. mali\_dvfs\_status\_current.utilisation = utilisation;
37. spin\_unlock\_irqrestore(&mali\_dvfs\_spinlock, flags);
39. /\* 要求在 cpu\_0 上, 使用 workqueue mali\_dvfs\_wq, 执行 mali\_dvfs\_work. \*/
40. queue\_work\_on(0, mali\_dvfs\_wq, &mali\_dvfs\_work);
41. /\*add error handle here \*/
42. **return** MALI\_TRUE;
43. }

该 event 被转发到 mali\_dvfs\_event\_proc() 中处理,

static DECLARE\_WORK(mali\_dvfs\_work, mali\_dvfs\_event\_proc);

**[cpp]** [view plain](https://blog.csdn.net/qq1028850792/article/details/79346153) [copy](https://blog.csdn.net/qq1028850792/article/details/79346153)

1. /\*---------------------------------------------------------------------------\*/
3. /\*\*
4. \* mali\_dvfs\_work 的实现主体, 即对 dvfs\_event 的处理流程的主体函数.
5. \*/
6. **static** **void** mali\_dvfs\_event\_proc(**struct** work\_struct \*w)
7. {
8. unsigned **long** flags;
9. mali\_dvfs\_status \*dvfs\_status;
11. **static** **int** level\_down\_time = 0; // counter\_of\_requests\_to\_jump\_down\_in\_dvfs\_level\_table :
12. //      对 mali\_dvfs\_level 下跳 请求 发生的次数的静态计数.
13. **static** **int** level\_up\_time = 0;   // counter\_of\_requests\_to\_jump\_up\_in\_dvfs\_level\_table :
14. //      对 mali\_dvfs\_level 上跳 请求发生的次数的静态计数.
15. **static** u32 temp\_tmp;
16. **struct** rk\_context \*platform;
17. u32 fps = 0;            // real\_fps.
18. u32 fps\_limit;
19. u32 policy;
20. mutex\_lock(&mali\_enable\_clock\_lock);
21. dvfs\_status = &mali\_dvfs\_status\_current;
23. **if** (!kbase\_platform\_dvfs\_get\_enable\_status()) {
24. mutex\_unlock(&mali\_enable\_clock\_lock);
25. **return**;
26. }
27. platform = (**struct** rk\_context \*)dvfs\_status->kbdev->platform\_context;
29. fps = rk\_get\_real\_fps(0);
31. dvfs\_status->temperature\_time++;
33. temp\_tmp += rockchip\_tsadc\_get\_temp(1);         // .Q : 获取当前温度? "1" : 意义? 指定特定的测试通道?
35. **if**(dvfs\_status->temperature\_time >= gpu\_temp\_statis\_time) {
36. dvfs\_status->temperature\_time = 0;
37. dvfs\_status->temperature = temp\_tmp / gpu\_temp\_statis\_time;
38. temp\_tmp = 0;
39. }
41. spin\_lock\_irqsave(&mali\_dvfs\_spinlock, flags);
42. /\*
43. policy = rockchip\_pm\_get\_policy();
44. \*/
45. policy = ROCKCHIP\_PM\_POLICY\_NORMAL;
47. **if** (ROCKCHIP\_PM\_POLICY\_PERFORMANCE == policy) {
48. dvfs\_status->step = MALI\_DVFS\_STEP - 1;
49. } **else** {
50. fps\_limit = (ROCKCHIP\_PM\_POLICY\_NORMAL == policy)?LIMIT\_FPS : LIMIT\_FPS\_POWER\_SAVE;
51. V("policy : %d , fps\_limit = %d", policy, fps\_limit);
53. /\*give priority to temperature unless in performance mode \*/
54. **if** (dvfs\_status->temperature > gpu\_temp\_limit)  // 若记录的 gpu 温度 超过了 上限, 则 ...
55. {
56. **if**(dvfs\_status->step > 0)
57. dvfs\_status->step--;
59. **if**(gpu\_temp\_statis\_time > 1)
60. dvfs\_status->temperature = 0;
61. /\*
62. pr\_info("decrease step for temperature over %d,next clock = %d\n",
63. gpu\_temp\_limit, mali\_dvfs\_infotbl[dvfs\_status->step].clock);
64. \*/
65. V("jump down in dvfs\_level\_table to level '%d', for temperature over %d, next clock = %d",
66. dvfs\_status->step,
67. gpu\_temp\_limit,
68. mali\_dvfs\_infotbl[dvfs\_status->step].clock);
69. }
70. // 若 current\_calculated\_utilisation 要求 上调 mali\_dvfs\_level,
71. //      且 current\_dvfs\_level 还可能被上调,
72. //      且 real\_fps "小于" fps\_limit,
73. // 则 ....
74. **else** **if** ( (dvfs\_status->utilisation > mali\_dvfs\_infotbl[dvfs\_status->step].max\_threshold)
75. && (dvfs\_status->step < MALI\_DVFS\_STEP - 1)
76. && fps < fps\_limit )
77. {
78. // 至此, 可认为一次请求 mali\_dvfs\_level 上跳 发生.
80. level\_up\_time++;
82. /\* 若 上跳请求的次数 达到 执行具体上跳 要求, 则... \*/
83. **if** (level\_up\_time == MALI\_DVFS\_UP\_TIME\_INTERVAL)
84. {
85. V("to jump up in dvfs\_level\_table, utilisation=%d, current clock=%d, fps = %d, temperature = %d",
86. dvfs\_status->utilisation,
87. mali\_dvfs\_infotbl[dvfs\_status->step].clock,
88. fps,
89. dvfs\_status->temperature);
90. /\* 预置 current\_dvfs\_level 上跳. \*/      // 具体生效将在最后.
91. dvfs\_status->step++;
92. /\* 清 上跳请求计数. \*/
93. level\_up\_time = 0;
95. V(" next clock=%d.", mali\_dvfs\_infotbl[dvfs\_status->step].clock);
96. BUG\_ON(dvfs\_status->step >= MALI\_DVFS\_STEP);    // 数组中元素的 index 总是比 size 小.
97. }
99. /\* 清 下跳请求计数. \*/
100. level\_down\_time = 0;
101. }
102. /\* 否则, 若 current\_calculated\_utilisation 要求 current\_dvfs\_level 下跳, 且 还可以下跳, 则... \*/
103. **else** **if** ((dvfs\_status->step > 0)
104. && (dvfs\_status->utilisation < mali\_dvfs\_infotbl[dvfs\_status->step].min\_threshold))
105. {
106. level\_down\_time++;
108. **if** (level\_down\_time==MALI\_DVFS\_DOWN\_TIME\_INTERVAL)
109. {
110. V("to jump down in dvfs\_level\_table ,utilisation=%d, current clock=%d, fps = %d, temperature = %d",
111. dvfs\_status->utilisation,
112. mali\_dvfs\_infotbl[dvfs\_status->step].clock,
113. fps,
114. dvfs\_status->temperature);
116. BUG\_ON(dvfs\_status->step <= 0);
117. dvfs\_status->step--;
118. level\_down\_time = 0;
120. V(" next clock=%d",mali\_dvfs\_infotbl[dvfs\_status->step].clock);
121. }
123. level\_up\_time = 0;
124. }
125. /\* 否则, ... \*/
126. **else**
127. {
128. level\_down\_time = 0;
129. level\_up\_time = 0;
131. V("keep current\_dvfs\_level, utilisation=%d,current clock=%d,fps = %d,temperature = %d\n",
132. dvfs\_status->utilisation,
133. mali\_dvfs\_infotbl[dvfs\_status->step].clock,
134. fps,
135. dvfs\_status->temperature);
136. }
137. }
138. #ifdef CONFIG\_MALI\_MIDGARD\_FREQ\_LOCK
139. // #error               // 目前配置下, 本段代码有效.
141. // 若 指定了 dvfs\_level\_upper\_limit,
142. //      且 预置的 current\_dvfs\_level "大于" dvfs\_level\_upper\_limit,
143. // 则...
144. **if** ((dvfs\_status->upper\_lock >= 0)
145. && (dvfs\_status->step > dvfs\_status->upper\_lock))
146. {
147. /\* 将 预置的 current\_dvfs\_level 调整到 dvfs\_level\_upper\_limit. \*/
148. dvfs\_status->step = dvfs\_status->upper\_lock;
149. }
151. **if** (dvfs\_status->under\_lock > 0) {
152. **if** (dvfs\_status->step < dvfs\_status->under\_lock)
153. dvfs\_status->step = dvfs\_status->under\_lock;
154. }
155. #endif
156. spin\_unlock\_irqrestore(&mali\_dvfs\_spinlock, flags);
157. /\* 将命令 dvfs\_module 让 current\_dvfs\_level 具体生效. \*/
158. kbase\_platform\_dvfs\_set\_level(dvfs\_status->kbdev, dvfs\_status->step);
160. mutex\_unlock(&mali\_enable\_clock\_lock);
161. }

a. 如果当前温度过高,那么执行jump\_down\_actually()下调频率.

b. 如果当前utilization 大于 max\_utilization\_of\_current\_dvfs\_level, 则直接上跳到最高频率(jump\_up\_to\_highest\_actually() ).不过要满足NUM\_OF\_REQUESTS\_TO\_PERFORM\_ACTUAL\_JUMP\_UP的次数才上跳.

c. 如果当前utilization 小于max\_utilization\_of\_current\_dvfs\_level,则一级级下跳(jump\_down\_actually()).不过要满足NUM\_OF\_REQUESTS\_TO\_PERFORM\_ACTUAL\_JUMP\_DOWN的次数才下跳.

如果觉得目前上跳策略太激进, 则可以在 mali\_dvfs\_event\_proc() 进一步修改, 比如每次只上跳一级, 而不是直接到 highest.

### 开关DVFS

GPU的DVFS不是在dts而是在驱动中直接控制并且默认打开的,如果要关闭，注释queue\_work\_on(即可

是否关闭

C:\Users\key.guan>adb shell

root@gl300e:/ # cat /sys/devices/ffa30000.gpu/dvfs

mali\_dvfs is OFF

current\_gpu\_clk\_freq : 594 MHz

root@gl300e:/ # cat /sys/devices/ffa30000.gpu/dvfs

mali\_dvfs is OFF

current\_gpu\_clk\_freq : 99 MHz

root@gl300e:/ # echo on > /sys/devices/ffa30000.gpu/dvfs

root@gl300e:/ # cat /sys/devices/ffa30000.gpu/dvfs

mali\_dvfs is ON

gpu\_utilisation : 60

current\_gpu\_clk\_freq : 99 MHz

/\*\*

\* mali\_dvfs\_work : 处理来自 kbase\_platform\_dvfs\_event 的 dvfs\_event 的 work.

\*/

**[cpp]** [view plain](https://blog.csdn.net/qq1028850792/article/details/79346153) [copy](https://blog.csdn.net/qq1028850792/article/details/79346153)

1. **static** ssize\_t set\_dvfs(**struct** device \*dev, **struct** device\_attribute \*attr, **const** **char** \*buf, **size\_t** count)
2. {
3. **struct** kbase\_device \*kbdev = dev\_get\_drvdata(dev);
4. #ifdef CONFIG\_MALI\_MIDGARD\_DVFS
5. **struct** rk\_context \*platform;
6. #endif
8. **if** (!kbdev)
9. **return** -ENODEV;
11. #ifdef CONFIG\_MALI\_MIDGARD\_DVFS
12. platform = (**struct** rk\_context \*)kbdev->platform\_context;
13. **if** (sysfs\_streq("off", buf)) {
14. /\*kbase\_platform\_dvfs\_enable(false, MALI\_DVFS\_BL\_CONFIG\_FREQ);\*/
15. D("to disable mali\_dvfs, and set current\_dvfs\_level to the highest one.");
16. kbase\_platform\_dvfs\_enable(**false**, p\_mali\_dvfs\_infotbl[MALI\_DVFS\_STEP-1].clock);
17. platform->dvfs\_enabled = **false**;
18. } **else** **if** (sysfs\_streq("on", buf)) {
19. /\*kbase\_platform\_dvfs\_enable(true, MALI\_DVFS\_START\_FREQ);\*/
20. D("to disable mali\_dvfs, and set current\_dvfs\_level to the lowest one.");
21. kbase\_platform\_dvfs\_enable(**true**, p\_mali\_dvfs\_infotbl[0].clock);
22. platform->dvfs\_enabled = **true**;
23. } **else** {
24. printk(KERN\_DEBUG "invalid val -only [on, off] is accepted\n");
25. }
26. #else
27. printk(KERN\_DEBUG "mali  DVFS is disabled\n");
28. #endif
29. **return** count;
30. }

#### kbase\_platform\_dvfs\_enable

Sdgh

**[cpp]** [view plain](https://blog.csdn.net/qq1028850792/article/details/79346153) [copy](https://blog.csdn.net/qq1028850792/article/details/79346153)

1. **int** kbase\_platform\_dvfs\_enable(**bool** enable, **int** freq)
2. {
3. mali\_dvfs\_status \*dvfs\_status;
4. **struct** kbase\_device \*kbdev;
5. unsigned **long** flags;
6. **struct** rk\_context \*platform;
8. dvfs\_status = &mali\_dvfs\_status\_current;
9. kbdev = mali\_dvfs\_status\_current.kbdev;
11. BUG\_ON(kbdev == NULL);
12. platform = (**struct** rk\_context \*)kbdev->platform\_context;
14. mutex\_lock(&mali\_enable\_clock\_lock);
16. **if** (enable != kbdev->pm.backend.metrics.timer\_active) {
17. /\* 若要 使能 dvfs, 则... \*/
18. **if** (enable) {
19. spin\_lock\_irqsave(&kbdev->pm.backend.metrics.lock, flags);
20. kbdev->pm.backend.metrics.timer\_active = MALI\_TRUE;
21. spin\_unlock\_irqrestore(&kbdev->pm.backend.metrics.lock, flags);
22. hrtimer\_start(&kbdev->pm.backend.metrics.timer,
23. HR\_TIMER\_DELAY\_MSEC(KBASE\_PM\_DVFS\_FREQUENCY),
24. HRTIMER\_MODE\_REL);
25. }
26. /\* 否则, 即要 disable dvfs, 则 ... \*/
27. **else** {
28. spin\_lock\_irqsave(&kbdev->pm.backend.metrics.lock, flags);
29. kbdev->pm.backend.metrics.timer\_active = MALI\_FALSE;
30. spin\_unlock\_irqrestore(&kbdev->pm.backend.metrics.lock, flags);
31. hrtimer\_cancel(&kbdev->pm.backend.metrics.timer);
32. }
33. }
35. **if** (freq != MALI\_DVFS\_CURRENT\_FREQ) {
36. spin\_lock\_irqsave(&mali\_dvfs\_spinlock, flags);
37. platform->time\_tick = 0;
38. platform->time\_busy = 0;
39. platform->time\_idle = 0;
40. platform->utilisation = 0;
41. dvfs\_status->step = kbase\_platform\_dvfs\_get\_level(freq);
42. spin\_unlock\_irqrestore(&mali\_dvfs\_spinlock, flags);
43. kbase\_platform\_dvfs\_set\_level(dvfs\_status->kbdev, dvfs\_status->step);
44. }
46. mutex\_unlock(&mali\_enable\_clock\_lock);
48. **return** MALI\_TRUE;
49. }

Asdfg

GPU DVFS控制策略小结

# Io口配置

# 实战

降低功耗，不需要单独编译

降低CPU主频，降低GPU主频，搞一个app来交互嘛

# REF

Keywords：DVFS 3288，

[DVFS](https://baike.baidu.com/item/DVFS)

[Rockchip Wiki](http://rockchip.wikidot.com/dvfs)

<http://blog.csdn.net/kris_fei/article/details/78059392>

TASK

[RK3288 硬件设计指南](https://wenku.baidu.com/view/86ebb8cfbe23482fb4da4ccd.html)