Spreadtrum Power Management

|  |  |  |  |
| --- | --- | --- | --- |
| **Document Number:** |  | **Document Version:** |  |
| **Owner:** |  | **Date:** |  |
| **Document Type:** |  | | |
| **NOTE:** | ALL MATERIALS INCLUDED HEREIN ARE COPYRIGHTED AND CONFIDENTIAL UNLESS OTHERWISE INDICATED. The information is intended only for the person or entity to which it is addressed and may contain confidential and/or privileged material. Any review, retransmission, dissemination, or other use of or taking of any action in reliance upon this information by persons or entities other than the intended recipient is prohibited.  This document is subject to change without notice. Please verify that your company has the most recent specification.  Copyright © 2013 Spreadtrum Communications Inc. | | |



[www.spreadtrum.com](http://www.spreadtrum.com)

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Date | Author | Description |
| 0.1 | 2013-06-23 | Alex.wang |  |
| 0.2 |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[Spreadtrum Power Management 1](#_Toc391304957)

[Revision History 2](#_Toc391304958)

[Table of Contents 3](#_Toc391304959)

[1. Introduction 4](#_Toc391304960)

[1.1. Request & Purpose 4](#_Toc391304961)

[1.2. Definitions & Abbreviations 4](#_Toc391304962)

[1.3. Reference 4](#_Toc391304963)

[2. Design Overview 5](#_Toc391304964)

[2.1. Concept 5](#_Toc391304965)

[2.2. Function Blocks 5](#_Toc391304966)

[3. Chip deep sleep 7](#_Toc391304967)

[3.1. sleep and wakeup 7](#_Toc391304968)

[3.2. DDR retention 8](#_Toc391304969)

[4. CPU Idle 9](#_Toc391304970)

[4.1. CPU-Idle Flowchart 9](#_Toc391304971)

[4.2. CPU-Idle Data Structure 12](#_Toc391304972)

[4.3. CPU-Idle Interfaces 13](#_Toc391304973)

[5. CPU DVFS 15](#_Toc391304974)

[5.1. CPU-DVFS Flowchart 15](#_Toc391304975)

[5.2. CPU-DVFS Data Structure 16](#_Toc391304976)

[5.3. CPU-DVFS Interfaces 17](#_Toc391304977)

[6. DDR DFS 18](#_Toc391304978)

[6.1. DDR-DFS Flowchart 18](#_Toc391304979)

[6.2. DDR-DFS Data Structure 22](#_Toc391304980)

[6.3. DDR-DFS Interfaces 22](#_Toc391304981)

[7. Thermal Strategy 24](#_Toc391304982)

[7.1. Thermal-Strategy Blocks 24](#_Toc391304983)

[7.2. Thermal-Strategy Data Structure 25](#_Toc391304984)

[7.3. Thermal-Strategy Interfaces 26](#_Toc391304985)

[8. Verification System 27](#_Toc391304986)

[8.1. Function Check List 27](#_Toc391304987)

[8.1.1. Function Completeness 27](#_Toc391304988)

[8.1.2. Performance 27](#_Toc391304989)

[8.1.3. System Stability 27](#_Toc391304990)

[8.1.4. Power Consumption 27](#_Toc391304991)

[8.2. Function Test Items 28](#_Toc391304992)

[8.3. Integration Test Items 29](#_Toc391304993)

# Introduction

This document is to present a chip-specific design for Spreadtrum power management based on the general Linux/Android power management system. It lists the general power features and gives the detailed specific design on the SoC platform.

## Request & Purpose

This design document should aim at the SharkL SoC power management, and give the functionality and features design,

1. CPU should kept in low power mode when it’s idle
2. CPU can be unplugged when it’s nearly used
3. CPU can do dynamical freq/voltage scaling by task demanding
4. DDR can do dynamical freq scaling by bandwidth demanding
5. Individual module should kept in low power mode when it’s idle
6. The typical runtime phone scenarios should get the low power consumption

## Definitions & Abbreviations

DVFS: Dynamical Voltage & Frequency Scaling

DFS: Dynamical Frequency Scaling

## Reference

1. “SharkL Device Spec”
2. “SharkL Low Power Spec”
3. “Spreadtrum\_Power\_Strategy”
4. <http://www.kernel.org/>
5. ...

# Design Overview

## Concept

In SharkL system, some power control features are presented in chip level,

1. Each CPU core has an independent power domain and can be shut down individually.
2. All CPU cores share a clock source and can do DVFS synchronously.
3. The DDR can do DFS and get auto gate below 200Mhz
4. The system can enter light-sleep mode by gating DDR PHY when no access to DDR.

The basic Power-Saving scheduler is to keep the active process in fewer CPU cores. Then the idle CPU can be unplugged or enter low power mode. To get low power run-time power, CPU and DDR should run with a frequency that’s as low as possible if the MIPS and bandwidth is acceptable.

## Function Blocks

When the system is really idle, it enters a deepsleep state that shuts down most the chip power and keeps it in an extreme low power state. However, most the consumption is leaked in the run time that requires the dynamical power control. CPU and DDR are common resource for the system and usually consume big power in the runtime. So that we tune the runtime power with the following sub-modules,

1. CPU Idle

When the next scheduler timer is got, the CPU can be set to a low power idle mode.

1. CPU DVFS

When CPU load is changed, the CPU frequency and voltage can be updated to keep lower power.

1. DDR DFS

When DDR bandwidth is changed, the DDR frequency can be updated to keep lower power.

1. Thermal Strategy

The thermal monitor reports the chip temperature and guides the power strategy.



# Chip deep sleep

## sleep and wakeup

Sleep:

Like Tshark, AP and CP can enter deep sleep mode individually, and the sleep process is the same. The chip can enter deep sleep mode in every DDR frequency, it does not keep DDR in 200MHz before enter deep sleep.

AP deep sleep:

The flow chart of AP deep sleep is described in figure3.1.



Wakeup:



1. Sleep/wakeup process document should be applied by ASIC designer.
2. DDR retention is the most important function in wakeup process.

## DDR retention

To be update

1. The hardware register status for checking whether the pub and phy is power down.
2. DDR retention document from ASIC
3. The hardware register or software status indicates DDR can be accessed safely.

# CPU Idle

Cpuidle can be implemented base on cpuidle framework in linux kernel, there already are two cpuidle governors, menu and ladder. In idle process, if no reschedule is required, cpuidle entry is called. The CPU idle interfaces are defined in <linux/cpuidle.h>. Users can check the driver and change the governor in /sys/devices/system/cpu/cpuidle/. Each SoC can register the cpuidle entry to handle the multi-level CPU idle state. For NO\_HZ system, the SoC idle driver can enter different states of power due to the idle time. tick\_nohz\_get\_sleep\_length() is used to get the idle length.

In SharkL platform, CA7 cores can enter 3 levels of CPU idle,

|  |  |  |  |
| --- | --- | --- | --- |
| C-State | Sub-State | Transition Latency(μs) | Description |
| C0  idle process | Idle-0 | ? | CPU Standby only |
| Idle-1 | ? | DDR Light Sleep |
| Idle-2 | ? | CA7 Core Power Down |

Idle-0:

Each core can enter standby and do Wait-For-Interrrupt.

Idle-1:

Each core stops to access DDR and set light sleep flag, then it enter standby and do WFI.

Idle-2:

The CA7 cores can be shut down automatically in idle time.

Basically, we select the **menu** governor as default. The menu governor looks at different parameters like what the expected sleep time is (as seen by dyntick), latency requirements, previous C-state residency, max c-state requirement etc, and then picks the deepest possible idle state straight away. This governor aims at getting maximum possible power advantage with little impact on performance.

## CPU-Idle Flowchart

In the CPU Idle driver, the governor will judge the idle state according to the idle time. Each CPU core can enter the different idle levels. The flowchart of the idle level and transition is showed below.

In SharkL, every CA7 core can be waken up by the IRQ directly. We can let each cores enter auto shutdown mode.



If auto shutdown mode can not work normally, we need to keep the implementation like shark and Tshark in figure below.



In Core Down mode, because the secondary CPUs can’t waken up by the IRQ directly, we make core0 to help power down/up the secondary CPUs. A secondary call remote func on CPU0 to shut down itself and register broadcast timer to ask CPU0 to wake up itself.



## CPU-Idle Data Structure

The cpuidle state structure is initialized as the following,

static struct cpuidle\_state sc9630\_cpuidle\_set[] \_\_initdata = {

[0] = {

.enter = sc9630\_enter\_idle,

.exit\_latency = 1,

.target\_residency = 1,

.flags = CPUIDLE\_FLAG\_TIME\_VALID,

.name = "C0",

.desc = "ARM clock gating(WFI)",

},

[1] = {

.enter = sc9630\_enter\_lightsleep,

.exit\_latency = 50,

.target\_residency = 100,

.flags = CPUIDLE\_FLAG\_TIME\_VALID,

.name = "C1",

.desc = "DDR Light Sleep",

},

[2] = {

.enter = sc9630\_enter\_coredown,

.exit\_latency = 2000,

.target\_residency = 5000,

.flags = CPUIDLE\_FLAG\_TIME\_VALID,

.name = "C2",

.desc = "CA7 Core power down",

},

};

## CPU-Idle Interfaces

CPU0 has a function “set\_cpu\_pd” to be called by the secondary CPUs to do the power off.

We do need to keep this function to avoid auto shutdown work abnormally.

/\*

\* IPI handler to shut down cpu power

\* @\*data, pointer of cpu id

\*/

static void set\_cpu\_pd(void \*data)

/\*\*

\* sc\_enter\_idle - Programs arm core to enter the specified state

\* @dev: cpuidle device

\* @drv: cpuidle driver

\* @index: the index of state to be entered

\*

\* Called from the CPUidle framework to program the device to the

\* specified low power state selected by the governor.

\* Returns the index of power state.

\*/

static int sc\_enter\_idle(struct cpuidle\_device \*dev,

struct cpuidle\_driver \*drv,

int index)

/\*

\* sc\_fill\_cstate: initialize cpu idle stata, and parameters

\* @drv: cpuidle\_driver

\* @idx: cpuidle status index

\*/

static inline void sc\_fill\_cstate(struct cpuidle\_driver \*drv,

struct cpuidle\_device \*dev, int idx)

/\*

\* register cpuidle driver to cpuidle framework

\* @drv: pointer of cpuidle driver

\* @cpu: cpu id

\* @return: 0 success, -1 fail.

\*/

static int sc\_cpuidle\_register\_device(struct cpuidle\_driver \*drv, unsigned int cpu)

/\*

\* sc9630 cpuidle initialization

\*/

int \_\_init sc\_cpuidle\_init(void)

# CPU DVFS

The CPUFreq provides the framework of SoC DVFS. It offers a standardized interface for the CPUFreq architecture drivers (those pieces of code that do actual frequency transitions), as well as to "notifiers".

A new governor “interactive” is added by Google in android kernel 3.4. The CPUfreq governor "interactive" is designed for latency-sensitive, interactive workloads. This governor sets the CPU speed depending on usage, similar to "ondemand" and "conservative" governors. However, the governor is more aggressive about scaling the CPU speed up in response to CPU-intensive activity.

In sharkL system, we implement the usual SoC-specific driver to adapt the DVFS driver. Because all CPU cores share the same clock source, the clock rate is set to the maximal one. To get minimal power consumption of CPU, we add dynamical hotplug strategy in the DVFS driver.

|  |  |  |  |
| --- | --- | --- | --- |
| P-State | Voltage (V) | Frequency (MHz) | Transition Latency(μs) |
| P0 | ? | ? | ? |
| P1 | ? | ? | ? |
| P2 | 0.9 | 900 | ? |
| P3 | 1.0 | 1350 | ? |

This table will be updated according to the latest chip update.

The flowchart is to be updated.

## CPU-DVFS Flowchart

The flow of DVFS is shown in below chart.



## CPU-DVFS Data Structure

struct cpufreq\_conf {

struct clk \*clk;

struct regulator \*regulator;

unsigned int orignal\_freq;

struct cpufreq\_frequency\_table freq\_tbl[FREQ\_TABLE\_SIZE];

unsigned int vddarm\_mv[FREQ\_TABLE\_SIZE];

};

struct cpufreq\_status {

unsigned int    global\_target;

unsigned int    percpu\_target[CONFIG\_NR\_CPUS];

int             is\_suspend;

};

static int dhp\_enable;

Except for the CPU freq/voltage table and percpu target, we also enable to support Dynamical-CPU-Hotplug in DVFS by a user interface.

## CPU-DVFS Interfaces

static int sprd\_cpufreq\_target(struct cpufreq\_policy \*policy,

unsigned int target\_freq,

unsigned int relation);

static void sprd\_plug\_new\_cpu(void);

# DDR DFS

In shark system, the DDR ram and controller consumes more power. To minimize the power consumption of DDR, the Dynamical Frequency Scaling and auto gating is supported.

Usually the DDR bandwidth can be monitored by bus monitor. And some HW/SW modules may request a certain bandwidth. We’ll launch a DDR DFS management system with the following features,

1. DDR DFS can be triggered by the DDR bandwidth monitor and increase/decrease the DDR frequency by the governor.
2. DDR bandwidth can be set by the requirement of kernel modules or user services.
3. DDR frequency can be set and fixed by the requirement of kernel modules or user services.
4. DDR DFS can be stopped and enter a low power mode if the system suspends.
5. DDR DFS can be executed without breaking display controller. It requires that dispc has a dedicated ram buffer to hold some display line buffer(that contains 40us or more content). DDR dfs set freq function should hold bus for less than 10us.
6. Some modem services can request dedicated frequency in advance, such as edge.

DFS flowchart is to be updated

## DDR-DFS Flowchart

In linux kernel, DDR DFS is implemented in the devfreq framework, which is similar to the DVFS framework that has device governor and profile driver.



In a delayed work of the devfreq framework, the update\_devfreq is called periodically. In this function, the device load is calculated and updated to the new frequency. I.e the ddr bus bandwidth is calculated and a new DDR freq is updated. Besides, it accepts the kernel or user request to update a DDR bandwidth requirement. Besides, DDR DFS may have request from modem side, we use IPI and shared memory as the request interface.

See the whole DDR DFS module flowchart below,



The loop of devfreq monitor -- governor -- dev profile is to update the DDR frequency. The logic of the delayed work and ddr request is showed below.

In modem\_irq\_handler(), when AP get the IPI, it polls all the ddr request status from the SIPC shared memory to get the exact DFS request and add this request to devfreq. In the memory layout of SIPC shared memory, every 128B after SIPC TX/RX BUF is customized for usage. The second one is used for DDR DFS request.



In the AP DFS handler, to keep CPU safe MMU is disabled in the DFS call and the operation code is put in the iram. Because DDR DFS operations may hold the bus access, no memory access can be done in the DFS period. The SMP CortexA7 may hang if multiple CPUs are racing on the bus. So that the secondary CPUs should be offline before DFS is executed.

Besides, SW & CHIP should guarantee the following pointes,

1. the time of disabling irq should be limited (<600us)
2. the period of bus hold is less than 10us
3. only one CPU is alive in DFS processing
4. displc flush rate is able to be controlled
5. modem should request to ap if it disables ddr dfs
6. no stack or memory is accessed in dfs operation iram.



## DDR-DFS Data Structure

struct ddr\_freq\_bandwidth\_talble {

int ddr\_freq, /\* in KHz. \*/

int bandwidth, /\* in MegaByte. \*/

}

static struct ddr\_freq\_bandwidth\_talble sc8830\_ddr\_table[] {

{.ddr\_freq = 192000, bandwidth = 768}

{.ddr\_freq = 384000, bandwidth = 1536}

{.ddr\_freq = 533000, bandwidth = 2204}

{.ddr\_freq = -1, bandwidth = -1}

}

static struct ddr\_request\_state

{

u32 req\_sum, /\* in MB \*/

u32 ddr\_freq\_after\_req, /\* in KHz \*/

}

static u32 ddr\_curr\_freq;

static u32 dfs\_down\_threshold = 90;

#define DDR\_DFS\_CPT\_REQUEST\_BASE \

(CPT\_START\_ADDR + SMSG\_TXBUF\_SIZE + SMSG\_RXBUF\_SIZE + 128)

#define DDR\_DFS\_CPW\_REQUEST\_BASE \

(CPW\_START\_ADDR + SMSG\_TXBUF\_SIZE + SMSG\_RXBUF\_SIZE + 128)

#define DDR\_DFS\_REQUEST\_TEDGE (DDR\_DFS\_CPT\_REQUEST\_BASE + 0)

#define DDR\_DFS\_REQUEST\_WEDGE (DDR\_DFS\_CPW\_REQUEST\_BASE + 0)

#define DDR\_DFS\_REQUEST\_AAAA (DDR\_DFS\_CPW\_REQUEST\_BASE + 4)

static addr\_t ddr\_dfs\_cp\_request[] = {

DDR\_DFS\_REQUEST\_TEDGE,

DDR\_DFS\_REQUEST\_WEDGE,

DDR\_DFS\_REQUEST\_AAAA,

};

dfs\_down\_threshold is only for optional component design.

## DDR-DFS Interfaces

Kernel public interfaces,

/\*

\* add/remove a new ddr bandwidth request.

\* @req\_bw: ddr bandwidth

\* @add: add(true) or remove(false)

\*/

void dfs\_request\_bw(u32 req\_bw, boolean add)

/\*

\* set current ddr frequency and hold auto dfs

\* @freq: ddr frequency in Hz

\* @hold: true or false to hold auto dfs, the second entry dfs is not active

\*/

u32 dfs\_set\_freq(int freq, boolean hold)

User space interfaces,

/\*

\* convert DDR bandwidth to frequency

\* @bw: DDR bandwidth in byte

\* @return: DDR frequency

\*/

static u32 dfs\_bandwidth\_to\_freq(u32 bw)

/\*

\* set DDR frequency, update ddr\_current\_freq.

\*if two scaling occur in a very short time, omit the later one.

\* @freq: MHz

\* @return: 0 scecuss, -1 fail

\*/

static int ddr\_freq\_set(u32 freq)

/\*

\* get the DDR bandwidth in the passed certain time

\* @return: bandwidth in byte

\*/

inline u32 ddr\_monitor\_bandwidth\_get( void );

/\*

\* start DDR bus monitor count bandwidth

\*/

inline void ddr\_monitor\_count\_start( void );

/\*

\* stop DDR bus monitor count bandwidth

\*/

inline void ddr\_monitor\_count\_stop( void );

# Thermal Strategy

In shark system, to prevent from over-temperature and keep the chip in a stable working status, some thermal sensors are added in the chip to monitor the system.



Thermal 0 monitors A-die, and Thermal 1 monitors the Coretex A7 components in D-Die. If thermal 0 warns, that means the power supply is over-heat, we have to shut down the whole system to cool down the chip. Thermal 1 monitors 4 levels of temperature:

1. Lower Level: (Cold)

System can free run regardless of low power strategy

1. Warning Level: (High offset -> Low offset)

System should enable power-prior strategy to prevent from quick heat increasing

1. Critical Level: (Hot -> Normal)

System should keeps in a low power running mode to prevent from heat burst

1. Cut-Off Level: (Overheat)

System should be shut down immediately to protect the chip

## Thermal-Strategy Blocks

To manage the shark power and temperature management, a linux general thermal zone device and cooling device drivers are implemented. Two thermal zone devices are presented and the corresponding cooling devices are created.



For the A-Die thermal, there’s only one level that’s to protect the chip from overheat.

The D-Die thermal is implemented as a thermal zone and bound with 2 cooling devices: CPU and DDR. In cool mode, cooling devices can be stopped and keep system performance-prior. In warning mode, cooling devices is enabled to do power-prior strategy, such as limited CPU and DDR Freq and Voltage, enable hotplug, etc. In Critical mode, only limited CPU/DDR resource can be enabled to quickly cool down the chip. If cut-off mode is reported, the system should be shut down very soon to protect the chip from overheat.

## Thermal-Strategy Data Structure

enum sprd\_thm\_sensor\_id {

SPRD\_ARM\_SENSOR = 0,

SPRD\_PMIC\_SENSOR = 1,

};

struct sprd\_trip\_point {

unsigned long temp;

enum thermal\_trip\_type type;

char cdev\_name[COOLING\_DEV\_MAX][THERMAL\_NAME\_LENGTH];

};

struct sprd\_thm\_platform\_data {

struct sprd\_trip\_point trip\_points[THERMAL\_MAX\_TRIPS];

int num\_trips;

};

extern int sprd\_thm\_temp\_read(u32 sensor);

The thermal devices and the trip points will be defined in the arch devices and emulated.

## Thermal-Strategy Interfaces

Please refer to {kernel}/Documentation/thermal/sysfs-api.txt for detailed guide.

# Verification System

## Function Check List

*TODO ... will merge from Jiajun’s LTP testsuite document*

### Function Completeness

*模块功能的完备性*

### Performance

AP sleep/wakeup:

1. suspend time consumption
2. resume time consumption

### System Stability

*模块稳定性和对系统影响*

### Power Consumption

*对系统功耗影响*

## Function Test Items

*必须，模块的单元测试定义，以内存测试为例*

* Dependency

*依赖于其他那些模块*

DDR/MMU/Cache

* Interface

*基于实现的接口形态*

Posix Libc

* Command

*命令格式*

Usage:

**utest\_mem** bandwidth [–s size] [-n count] [-p processes]

**utest\_mem** verify [-s size] [-n count]

bandwidth

test the bandwidth of the memory copy and output the result as MB/s. When the

size is below cache size, the result is the bandwidth of the cache.

verify

verify the write/read coherence of memory access and output “Success” or

“Failure”.

-s size

specify the memory block size. It’s 16MB by default.

-n count

specify the count to repeat the testing. It’s 16 by default.

-p processes

specify the threads number to do the test. It’s 1 by default.

* Interaction

*测试过程是否需要人为交互，以及如何交互*

NO.

* Auto Test

*机器自动测试和输出支持*

All the test items can be automatic. The final results are in the command output.

* Cases

*标准单元测试样例*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Prio | Function | Input | Output | Comments |
| 0001 | H | Mem Check | utest\_mem verify -s 16777216 -n 16 | Success/Failure |  |
| 0002 | H | L1 Perf | utest\_mem bandwidth -s 4096 -n 262144 | nnnn.xxx MB/s | size is less than half of L1 data cache, size\*count > 1GB. get the average by multiple test |
| 0003 | H | L2 Perf | utest\_mem bandwidth -s 65536 -n 16384 | nnnn.xxx MB/s | size is more than L1 data cache, but less than half of L2 cache, size\*count > 1GB. get the average by multiple test |
| 0004 | H | Mem Perf | utest\_mem bandwidth -s 16777216 -n 64 | nnnn.xxx MB/s | size is much more than L1+L2, size\*count > 1GB. get the average by multiple test |
| 0005 | H | L1 Perf | utest\_mem bandwidth -s 4096 -n 262144 -p 2 | nnnn.xxx MB/s | same with 0002. nproc is the CPU core number or twice |
| 0006 | H | L2 Perf | utest\_mem bandwidth -s 65536 -n 16384 -p 2 | nnnn.xxx MB/s | same with 0003. size\*nproc is less than half of L2 cache |
| 0007 | H | Mem Perf | utest\_mem bandwidth -s 16777216 -n 64 -p 2 | nnnn.xxx MB/s | same with 0004 |
|  |  |  |  |  |  |

## Integration Test Items

*集成测试环境和耦合验证描述*