Thermal management of ARM SoCs using Linux CPUFreq as cooling device

Lei Zhou[1], Shengchao Guo[2]*

[1]Changshu Institute of Technology, Changshu, China

[2]Freescale Semiconductor, Inc., Shanghai, China

Received 13 October 2014, www.cmnt.lv

Abstract

While the performance of modern ARM System on Chips (SoCs) increases significantly than the previous generation, the power dissipation and on-chip temperature becomes dramatically high. The existing thermal management solutions are mostly built on Advanced Configuration and Power Interface (ACPI) and Dynamic Thermal Management (DTM) model for traditional desktop and server machines. They do not directly apply on mobile ARM SoCs as it is. This paper proposes a solution for thermal management on high performance ARM SoCs based on a model which is built on some ACPI thermal concepts and DTM DVFS mechanism. The thermal model is implemented on Freescale i.MX6Q SoC with Linux Kernel 3.6. It uses a lot of help from Linux thermal infrastructural and CPUFreq subsystem, and builds a cooling device backed by CPUFreq driver. A comparison testing on i.MX6Q shows that the cooling device can effectively controls the on-chip temperature around a designed threshold value. The thermal model is built on generic thermal hardware support and common software infrastructural, and therefore should work universally for other ARM SoCs.

Keywords: ARM, Thermal, i.MX6Q

1 Introduction

ARM System on Chips (SoCs) were initially designed for general embedded applications with low power dissipation being the principle, and thermal was never a practical concern in those days. However, the situation gets changed since modern ARM SoCs are widely adopted on mobile devices like smart phones and tablets. To provide a better user experience on these devices, the SoCs are designed with much higher processor frequency and more CPU cores. The consequence of such design is that the power dissipation is much higher, and much more heat gets generated than the prior generation ARM SoCs. The Freescale i.MX28 application processor is one typical early generation ARM SoC. It integrates an ARM9 core, and runs at 454MHz as maximum. In comparison, the i.MX6 Quad (i.MX6Q) produced by Freescale is a typical multi-core ARM SoC for mobile devices, which integrates 4 Cortext-A9 cores and runs up to 996MHz. For given use case, i.MX6Q will obviously provide much higher performance than i.MX28. But as demonstrated in Figure 1, the on-die temperature of i.MX6Q increases dramatically faster and reaches a much higher degree than i.MX28 in a testing with CPU cores fully loaded.

When other on-chip modules run together with the CPU cores under heavy load, modern high performance ARM SoCs like i.MX6Q can easily reach an even higher temperature than what Figure 1 illustrates. It's well known that an excessively high on-chip temperature can result in performance degradation, poor reliability and signal integrity issues [1, 2]. In addition, the leakage power consumption increases exponentially with increasing chip temperature. Higher temperature results in greater power consumption, and this effect in turn increases the on-chip temperature even higher. More importantly, running chip at high temperature will accelerate failure mechanisms, such as dielectric breakdown and electro-migration, which may result in a permanent damage to the chip. Hence there is really a need of thermal management to control temperature on these ARM SoCs.

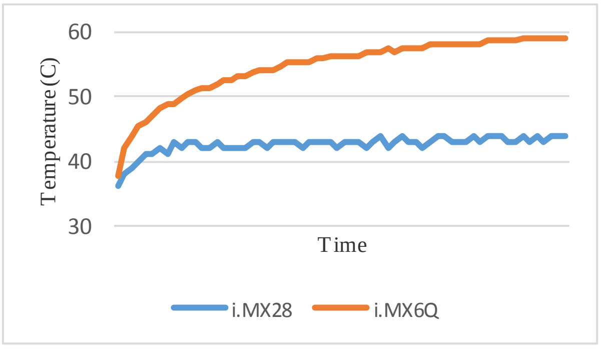


FIGURE 1 Temperature comparison between i.MX28 and i.MX6Q

This paper proposes a solution for thermal management on high performance ARM SoCs based on Linux Kernel thermal framework and CPUFreq support. The paper is organized as follows. Section 2 gives some background of thermal management topic, including ACPI, DTM and related works around thermal management on mobile device. Section 3 introduces the i.MX6Q thermal support at hardware level, and elaborates the design principles and implementation details of the thermal model, which is built on Linux ACPI thermal concepts and CPUFreq subsystem. Section 4 demonstrates the effectiveness of the thermal management model by running a comparison testing on i.MX6Q, and discusses the future works for making the model even more efficient for managing thermal on ARM SoCs. Finally, Section 5 concludes that the thermal model is built on common technology and generic frameworks,

* ***Corresponding author*** e-mail: zhoulei@cslg.cn

therefore can universally work for other ARM SoCs besides Freescale i.MX6Q SoC.

2 Background

2.1 ADVANCED CONFIGURATION AND POWER INTERFACE

The Advanced Configuration and Power Interface (ACPI) specification [3] was developed to provide a standardized approach to configuring the hardware, systems, and software necessary for power and thermal management. ACPI defines a set of points on the temperature scale that are referred to as cooling policies: active, passive and critical. These are the trip points where the various cooling methods might be triggered. As the CPU heats up and reaches the active threshold, a fan might be turned on to cool down the processor. If temperature continues to increase and passive threshold is crossed, the system will need to reduce the processor’s clock frequency to decease power dissipation. Finally, if temperature continues to rise to the critical level that might cause damage to the CPU, power should be removed from all or part of the system.

While this whole thermal management model works perfectly fine on desktop and server machines, the active policy doesn’t really fit into mobile devices. Considering the compact form factor of ARM based devices, cooling fan is not really an option there, apart from the facts that cooling fan requires additional power energy and generates noise. Except that, the passive and critical policies should work for ARM SoCs in general.

2.2 DYNAMIC THERMAL MANAGEMENT

Dynamic Thermal Management (DTM) refers to a range of possible software and hardware strategies which work dynamically to control a chip’s operating temperature at runtime [4]. The key goal of DTM is to provide inexpensive software or hardware responses to control the over-heating of the chip and maintain the chip temperature always below a critical temperature Tcritical. A trigger temperature Ttrigger which is generally lower than Tcritical is defined as the trip point, so that when the chip temperature reaches Ttrigger, DTM is invoked to cool down the chip. There are quit some DTM response mechanisms being researched and proposed, but two most typical and practical techniques are task scheduling and Dynamic Voltage and Frequency Scaling (DVFS).

The general idea of task scheduling mechanism is migrating heavy loading task away from an over-heated core to a cooler core [5, 6, 7, 8]. While this technique might work well for other CPU architecture, it doesn’t practically work for ARM SoCs, because the existing ARM processors do not generally have a reliable way to report thermal state of individual core, but only the on-die temperature as a whole. Also, as temperature is a complex function of processor load and cannot really be easily predicted [8], for achieving a good cooling effect, the task migrating algorithms can be very complicated and difficult to implement. On the other hand, DVFS becomes a very effective mechanism for runtime thermal management, due to the quadratic dependence of dynamic power voltage [9, 10, 11]. And more importantly, DVFS technique has been widely adopted by modern ARM SoCs designs to manage power dissipation [12], so it can be naturally employed to implement dynamic thermal management for ARM SoCs.

2.3 RELATED WORKS

There has been some attempt to extend Linux ACPI support designed for desktop and server machines to cover mobile devices [13]. But that work was still around Intel X86 platforms, and relies on ACPI support from firmware to manage processor cooling state. The ACPI is not supported by ARM ecosystem at any level as of today. Also, that work focuses on managing thermal at platform level rather than processor itself, and requires a user application to make decisions on thermal policy control. Such design shifts the implementing burden and policy complexity to user applications, and adds the communication overhead between kernel and user spaces, which results in a less efficient solution than handling cooling policies in kernel space. There is also another work choosing to completely ignore the existing ACPI thermal support in Linux Kernel and manage thermal within CPUFreq subsystem [14]. It implements a new thermally-aware CPUFreq governor, which will directly measures temperature as well as performance status of all CPU cores and then select a CPUFreq level that would not lead to the excessive temperature. While the whole solution might work, it needs to build up every piece of thermal model from the ground and plug the code into CPUFreq subsystem, which makes less sense in terms of software reuse and will likely results in architectural issue if the code gets submitted to upstream kernel for inclusion.

From the introductions on ACPI and DTM above, it can be seen that the DVFS based DTM provides a practical and effective response to trigger temperature Ttrigger, which can be well mapped to ACPI passive cooling policy and trip point. While DTM assumes that Tcritical should never be reached on a DTM based system, ACPI recommends removing power from system as the response to critical temperature, which also applies on ARM SoCs. Therefore, this paper proposes a thermal management solution for ARM SoCs, which is built on top of ACPI cooling concepts and DTM DVFS response mechanism. And Linux is chosen as the software environment to implement the proposal, because Linux Kernel has an existing thermal framework designed for ACPI thermal management on X86 CPUs. Though ARM SoCs do not support ACPI specification, quite a lot of thermal concepts and common code can be reused on ARM platforms. Also, the CPUFreq subsystem of Linux Kernel perfectly supports DVFS mechanism for ARM machines [15]. Using these fundamental infrastructural supports from Linux can make the implementation a lot easier and help authors to focus on the innovation part of the whole solution.

3 Thermal Management on i.MX6Q

3.1 I.MX6Q THERMAL SUPPORT

The i.MX6Q application processor integrates a Temperature Monitor (TEMPMON), which implements a temperature senor function based on temperature dependent voltage to time conversion. The module features an alarm function which can raise an interrupt if the temperature goes above a programmed threshold. Software can read the temperature sensor counter (TEMP_CNT) in conjunction with the fused calibration data to determine the on-die operational temperature.

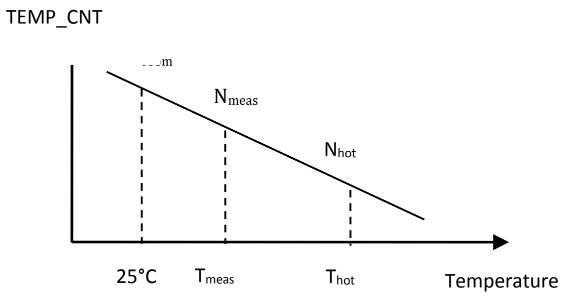


FIGURE 2 Temperature measurement on i.MX6Q

Figure 2 shows the procedure of temperature measurement on i.MX6Q. Nroom, Nhot and Thot are three values that will be read from chip fuse. Nroom and Nhot are TEMP_CNT values at room temperature 25°C and a hot temperature which is specified by Thot. All these three values are written into fuse bits based on the calibration during chip fabrication. With Nmeas read out from TEMP_CNT register, the temperature to be measured Tmeas then can be calculated using (1).

$$T_{meas} = T_{hot} - \left(N_{meas} - N_{hot}\right)\left(\frac{T_{hot} - 25}{N_{room} - N_{hot}}\right). \qquad (1)$$

Besides the temperature sensor, i.MX6Q supports changing core voltage VDDARM and clock frequency at runtime with DVFS technique, so that a Linux CPUFreq driver can be implemented based on that as a cooling device for thermal management. Table 1 lists the recommended DVFS operating points of i.MX6Q as well as the corresponding power dissipation and temperatures measured under the condition that all CPU cores are fully loaded. The data clearly indicates that the i.MX6Q DVFS function can use effectively used to manage no-chip temperature.

TABLE 1 i.MX6Q DVFS characteristics

ARM Freqency (MHz)	**VDDARM (V)**	**Power (mW)**	**Temperature (°C)**
996	1.25	2080.50	59
792	1.15	1647.75	51
396	0.95	598.50	38

3.2 LINUX KERNEL THERMAL MODEL

The Linux thermal management model is built on the concept of thermal zone. As shown in Figure 3, a thermal zone, by definition, not only includes a sensor device reading temperature of the region that the thermal zone manages, but also a list of cooling devices associated with the thermal zone, which are used to bring down the temperature of the thermal zone under control of a governor.

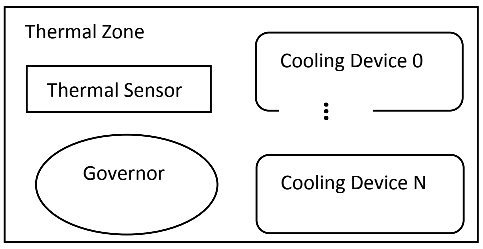


FIGURE 3 Thermal zone device model

In a computer system with particular complexity, there might be multiple thermal sensors for different regions, like processor and motherboard. Multiple thermal zone devices will be created for such systems. In Linux Kernel thermal infrastructural, each thermal zone is represented by struct thermal_zone_device and registered by calling function thermal_zone_device_register(). And all registered thermal zone device is maintained on thermal_tz_list. Each thermal zone can have more than one cooling device to control temperature of the thermal zone. Cooling devices can be categorized into two types as per the cooling mode, active cooling and passive cooling. Active cooling increases the power consumption of the system, e.g. turning on a fan, to reduce the temperature. Passive cooling reduces the power consumption at the cost of system performance, e.g. throttling processor clock, to lower down the processor temperature. Each cooling device is registered by calling function thermal_cooling_device_register() and identified as a struct thermal_cooling_device in Linux thermal framework. And all registered cooling devices are collected on another list thermal_cdev_list. Function thermal_zone_bind_cooling_device() searches given thermal zone and cooling devices on thermal_tz_list and thermal_cdev_list, and associates these cooling devices with the thermal zone. Then under the control of a thermal governor, cooling devices will perform particular operations to react to the temperature change of the thermal zone.

3.3 MANAGE I.MX6Q TEMPERATURE WITH LINUX THERMAL MODEL

Same as the most ARM SoCs with thermal support today, i.MX6Q integrates one temperature sensor reporting on-die temperature of the SoC. So one thermal zone device is enough for the thermal model implementation on i.MX6Q. Also, unlike traditional X86 platforms that generally have multiple cooling devices for a single thermal zone, e.g. fan as an active cooling device and processor throttling as a passive cooling device, active cooling is not applicable for i.MX6Q. Based on these facts, a simplified thermal model as shows in Figure 4 is implemented on Linux for i.MX6Q system.

There is one thermal zone driver which works as a sensor driver in this model. It can report the on-chip temperature by polling i.MX6Q TEMPMON block and also program TEMPMON to send interrupt when temperature reaches a threshold value. The thermal zone has only one cooling device bound to it. And the cooling device is backed by i.MX6Q CPUFreq driver which can scale both the clock frequency and voltage of CPU at different level to control the thermal zone temperature. As CPUFreq lowers down the temperature by reducing the system power consumption at the cost of performance, it's clearly a passive cooling device. Therefore, i.MX6Q thermal driver defines a passive trip point at 85 Celsius degree as per the recommendation from Freescale. The thermal zone governor in this model makes decision on when and how to invoke cooling device for temperature control. The sensor drive will report the on-chip temperature back to the thermal zone governor in every 2 seconds by polling TEMPMON. Once the temperature crosses the passive trip point, the governor will trigger the cooling procedure by calling into cooling device. Besides of the passive trip point, a critical trip point has to be defined, so that the thermal core can react to it by powering off the system immediately when the critical temperature is reached. The critical trip point is defined at 105 Celsius degree according to i.MX6Q data sheet [16]. As running a chip at the critical temperature for even 2 seconds, i.e. the polling period in this model, might be a dangerous thing, the thermal driver chooses to implement the critical trip point with interrupt. Thus, the cooling reaction to critical temperature can be taken as soon as possible to avoid the chip being damaged.

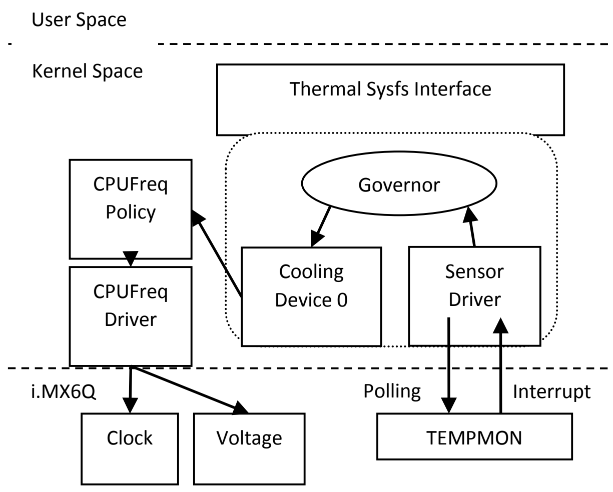


FIGURE 4 i.MX6Q thermal management model

As listed in Table 1, there are three CPU operating points which have different level of thermal performance. Based on that, the passive cooling device is implemented to provide three different cooling states for thermal governor to manipulate. When cooling device performs at the lowest cooling state, CPU runs at the fastest operating point. When cooling device performs at the highest cooling state, CPU runs at the slowest operating point. The cooling device driver maintains such mappings between cooling states and CPU operating points by setting CPUFreq policy->max to limit the maximum frequency that CPU can run at. Figure 5 illustrates a simplified flow chart of cooling procedure. In each temperature measurement, the thermal governor compares the current temperature Tcur with the passive trip point temperature 85 Celsius degree. The cooling procedure will be activated. It in turn determines whether the temperate is rising or dropping by comparing Tcur and the last measured temperature Tlast. If the temperature is rising, the governor will apply the next higher cooling state, which maps to the next slower CPU operating point. On the contrary, if the temperature is dropping, the governor will move the cooling state to the next lower level, which maps to the next faster CPU operating point.

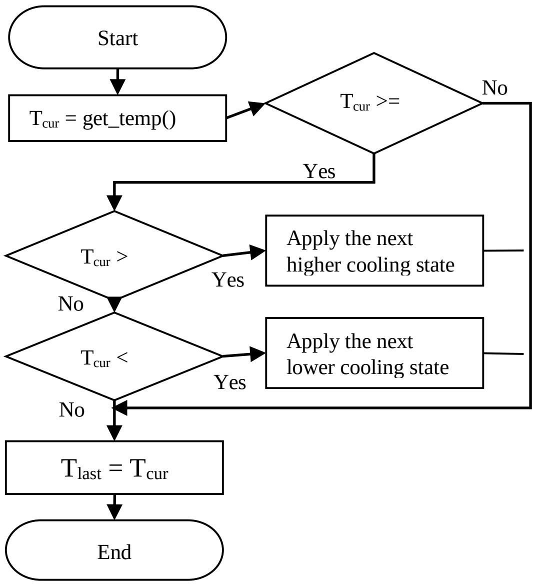


FIGURE 5 Cooling procedure flow chart

Although the policy of thermal management is implemented in kernel space as a thermal zone governor, the sysfs interface provided by Linux thermal infrastructural is still quite useful for user space applications or tools to query and configure the system thermal parameters. With this i.MX6Q thermal implementation, after the thermal driver successfully registers the thermal zone device and cooling device to Linux thermal framework, folders cooling_device0 and thermal_zone0 will be seen under sysfs entry /sys/class/thermal. The information of the cooling device like name and cooling state can be found in folder cooling_device0, and thermal_zone0 folder contains various configurations about the thermal zone device. The most interesting entries in thermal_zone0 would be temp, trip_point_0_temp and mode. The temp is a read-only entry which reports the current temperature of the thermal zone, while trip_point_0_temp can be read or written to get or set the threshold temperature of the passive trip point. The mode entry can be used to enable or disable the cooling model at runtime. While changing the thermal configurations by writing these sysfs entries is not suggested to normal users, it's quite useful for developers to debug and test the thermal model.

4 Measurement and discussions

To verify whether the thermal model can work practically to have the on-chip temperature under control, a comparison test is created on i.MX6Q. The test is performed in the following steps.

Power off the system if it's running and wait for around 5 minutes to let i.MX6Q completely cool down.

Boot up the system and login the console.

Run command echo disabled > /sys/class/thermal/thermal_zone0/mode to disable the cooling model.

Launch a bash script which reports the temperature of i.MX6Q by looking at sysfs entry /sys/class/thermal/thermal_zone0/temp in every 2 seconds with an indefinite loop.

Wait a few minutes for the temperature to be stable.

Launch 4 instances of a simple console program which does nothing but a busy loop to get all 4 CPU cores of i.MX6Q fully loaded.

Monitor the reported temperatures and stop the test script after the temperature reaches a relatively stable value.

In this test case, it typically takes about 2 minutes to see the temperature rise to a relatively stable value. Authors choose to record 64 samples which are collected in 128 seconds, and compose the trend line marked as Non-cooling in Figure 6. To demonstrate the effectiveness of CPUFreq cooling technology, other heat contributors of i.MX6Q are not enabled in this test. That's why the high temperature with all CPU cores loaded only reaches around 60 Celsius degree in this test. Since the passive trip point of the thermal model is defined at 85 Celsius, the cooling device is not trigger to work anyway. That said, the step 3 which disables the cooling model is not really necessary for this test. It also means that the trip point temperature needs to be lowered to trigger the cooling device in this testing circumstance. So in a comparison test to the first one, step 3 is replaced by the command echo 50000 > trip_point_0_temp to set the passive trip point temperature to 50 Celsius degree. (The unit used in Linux thermal infrastructural is milli-celsius.) The result is shown as the trend line marked as Cooling in Figure 6. It's quite clear that under the control of cooling device, the temperature stops rising around 50 Celsius degree and stays at that level all the way forward.

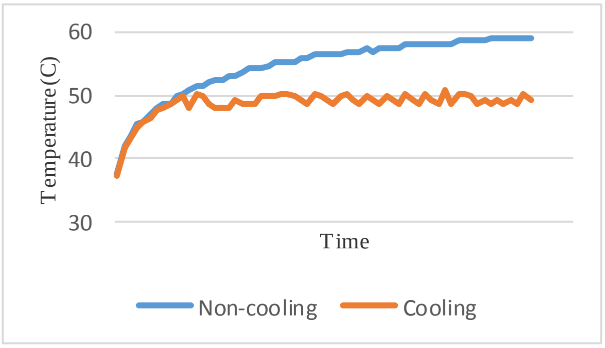


FIGURE 6 i.MX6Q temperature between non-cooling and cooling

Though the above comparison test proves that the cooling device backed by CPUFreq can effectively manage the thermal of i.MX6Q, there are a few future works to be done in order to make the thermal model even better for controlling the on-chip temperature of ARM SoCs. First of all, the comparison test needs to be set up in a laboratory which can provide a high temperature environment to ensure the thermal model works as good as under room temperature. In that case, the 85 Celsius trip point can be verified with a real world temperature. Secondly, similar to what's been done on Intel Pentium platform in article [10], some experiments should be set up on ARM SoCs to identify the response timescale and influence of factors beyond voltage and frequency on the chip temperature. Also, for multi-core ARM SoCs, CPU hot-plug is another possible cooling method used to implement CPU cooling device. It would be interesting to benchmark the cooling efficiency between CPUFreq and CPU hot-plug, so that a most efficient thermal solution can be modelled. Last but not least, besides CPU, modern ARM SoCs for mobile devices typically integrate graphic processing unit and video processing unit, which are both big heat contributors to the on-chip temperature. In some particular use case, cooling CPU only might be unable to keep SoC from reaching an excessive temperature. So some additional cooling devices may be needed to manage graphic and video processing units together with CPU to keep the temperature of the whole SoC within a safe range.

5 Conclusions

The thermal model proposed by the paper is implemented and verified on Freescale i.MX6Q. The testing result demonstrates it's an effective dynamic thermal management solution using DVFS technology. As thermal becomes a major concern of high performance ARM SoCs today, thermal sensor and DVFS support are universally available on all the popular ARM chips like Samsung EXYNOS and STE UX500. On the other hand, the thermal infrastructural and CPUFreq subsystem in Linux Kernel provide a quite generic interface to SoC specific thermal sensor and DVFS drivers. Therefore, with some limited consolidation effort, this i.MX6Q thermal model proposed by the paper can be easily ported to those ARM SoCs, and therefore becomes a generic thermal solution for ARM mobile application processors.

References

[1] Hertl M, Weidmann D and Ngai A 2009 An advanced reliability improvement and failure analysis approach to thermal stress issues in IC packages 35th International Symposium for Testing and Failure Analysis pp. 28–32

[2] Wang W, Reddy V, Vattikonda R, Krishnan S and Cao Y 2007 Compact modelling and simulation of circuit reliability for 65-nm CMOS technology IEEE Transactions on Device and Materials Reliability 7, 509–517

[3] Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba 2013 Advanced Configuration and Power Interface Specification. http://www.acpi.info/ 10 Sept 2014

[4] Brooks D and Martonosi M 2001 Dynamic thermal management for high-performance microprocessors Proceedings of the 7th International Symposium on High-Performance Computer Architecture pp. 171 –182.

[5] Yang J, Zhou X, Chrobak M, Zhang Y and Jin L 2008 Dynamic thermal management through task scheduling IEEE ISPASS pp. 191–201.

[6] Yeo I, Liu C C and Kim E J 2008 Predictive dynamic thermal management for multicore systems Proc. Design Automation Conf. (DAC), ACM, pp. 734–739.

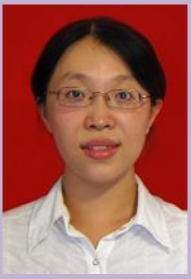
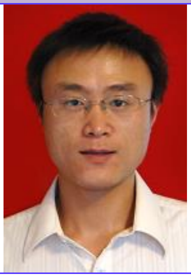
[7] Xia L, Zhu Y, Yang J, Ye J and Gu Z 2010 Implementing a thermal-aware scheduler in linux kernel on a multi-core processor The Computer Journal 53, 895–903.

[8] Liu G, Fan M and Quan G 2012 Neigbor-aware dynamic thermal management for multi-core platform Proc. European Design and Test Conf. (DATE), pp. 187–192.

[9] Huang W, Allen-Ware M, Carter J, Cheng E, Skadron K and Stan M

2011 Temperature-aware architecture: lessons and opportunities Micro, IEEE 31, 82–86.
[10] Hanson H, Keckler S, Ghiasi S, Rajamani K, Rawson F and Rubio J 2007 Thermal response to DVFS: analysis with an Intel Pentium M ACM/IEEE ISLPED, pp. 219–224.
[11] Mukherjee R and Memik O S 2006 Physical aware frequency selection for dynamic thermal management in multi-core systems Proc. Int. Conf. on Computer Aided Design (ICCAD), pp. 547–552.
[12] Herbert S and Marculescu D 2007 Analysis of dynamic voltage/frequency scaling in chip-multiprocessors Proc. Int. Symp. on Low Power Electronics and Design (ISLPED), pp. 38–43.
[13] Sujith T and Zhang R 2008 Thermal management in user space Proceedings of the Linux Symposium, pp. 227–233.
[14] Wojciechowski B and Bawiec M A 2012 Practical dynamic thermal management of multi-core microprocessors Thermal Investigations of ICs and Systems (THERMINIC), pp. 1–4.
[15] Zhou L, Lv Q and Guo S 2013 A generic linux cpufreq driver for ARM SoCs International Journal of Online Engineering (iJOE) 9, 29–32.
[16] Freescale Semiconductor 2014 i.MX 6Dual/6Quad Automotive and Infotainment Applications Processors, Rev.3

Authors

	**Lei Zhou, Mar. 4, 1980, Changshu, Jiangsu, China** **Current position, grades:** Lecturer **University studies:** Computer Science **Scientific interest:** Power management on mobile devices **Publications:** 3 **Experience:** Lei Zhou graduated from Soochow University with a master degree of Computer Science at 2005, and she has been staff of Changshu Institute of Technology since then. The primary courses she teaches are C language programming and database technology. She is proficient in software engineering and algorithm design, experienced in embedded system development and database design with Microsoft Access and Oracle. Her main research areas include: embedded system, power management, and information processing. There were one article about embedded system power management published on EI journals, and two about information processing on Chinese core journals.
	**Shengchao Guo, Aug. 15, 1978, Zhenjiang, Jiangsu, China** **Current position, grades:** Senior software engineer **University studies:** Computer Science **Scientific interest:** Embedded system and device drivers **Publications:** 1 **Experience:** Shengchao Guo graduated from Soochow University with a master of Computer Science at 2004. He had one year working experience in Macronix (Suzhou) developing multimedia system on MIPS core. And he has been an employee of Freescale Semiconductor since year 2005, developing board support package for i.MX application processors. His expertise include: ARM core architecture, Linux device driver development, power management, and Freescale i.MX SoCs.