Dynamic Thermal Management for Homogeneous Embedded Multicore Processors: Thermal Modeling, System Environment, and Measurement Results

Axel Sikora*, Sebouh Toumassian

Laboratory Embedded Systems and Communication Electronics, Offenburg University of Applied Sciences
D77652 Offenburg, Germany.

* Corresponding author. Tel.: +49-781-205-41; email: axel.sikora@hs-offenburg.de

Manuscript received May 17, 2015; revised August 14, 2015.

doi: 10.17706/ijcee.2015.7.4.223-231

Abstract: The increasing number of transistors being clocked at high frequencies of modern microprocessors lead to an increasing power consumption, which calls for an active dynamic thermal management. In a research project a system environment has been developed, which includes thermal modeling of the microprocessor in the board system, a software environment to control the characteristics of the system's timing behavior, and a modified Linux scheduler, which is enhanced with a prediction controller. Measurement results are shown for this development for a Freescale i.MX6Q quad-core microprocessor.

Key words: Dynamic thermal management (DTM), multi-core processing, thermal modeling.

---

1. Introduction

Today, the domains for embedded systems are uncountable and many applications require modern high performance microprocessors to accomplish increasingly complex tasks. One major problem of high performance microprocessors, especially of multi-core processors, is the high power dissipation [1], [2]. Processors directly convert consumed electrical power into thermal heat [3]. High operation temperatures may have negative influence on the chip reliability, processor performance, storages abilities of memories and leakage power consumption [3]-[6]. Legacy processors come with an emergency shut-off, which reacts at a given temperature threshold.

Active Dynamic Thermal Management (DTM) deals with mechanisms to profile model and regulate operation temperatures of microprocessors. The described project focuses thermal microprocessor modeling, thermal process and thread classification. The closed loop control architecture is based on a predictive controller using Dynamic Voltage and Frequency Scaling (DVFS) as thermal enhancement to the Completely Fair Scheduler (CFS) and the Real-Time Scheduler of Linux. The general goal of the project is to maximize the computational power, especially of real-time processes and threads under all temperature constraints [7].

The remainder of the paper is organized as follows. In Section 2, a short overview of the thermal properties, flow and reduction of the heat in multicore processors is given. The implementation and time consumption are discussed in Section 3. In Section 4, the frequency scaling mechanism in the Linux kernel and the current state of the developed software are explained. Section 5 and Section 6 give an insight into

the control algorithms. Finally a summary of the project with future work suggestion is discussed in Section 7.

2. Thermal Measurements

A precondition for the successful integration of active DTM methods is a well-engineered, configurable thermal model of processor to effectively enable the DTM to apply closed loop control actions. A major challenge in the derivation of a thermal model is the difference of microprocessors with respect to the power dissipation and architectural structure. Every microprocessor type has individual thermal specifications [7]. Currently, the project concentrates on one microprocessor for research and algorithm developments: the i.MX6 Quad Core from Freescale running on a Linux kernel version 3.0.35.

In general, factors which influence the operation temperature of the processor do not only depend on the power dissipation and the architectural structure, but also on the environment. Amongst others, this can be the temperature of the environment, the thermal spreading abilities of the printed circuit board (PCB) or the utilization of active and passive heat sinks [7].

The fundamental thermal model in the project is based on the analogy of temperature to an electrical Resistor-Capacitor (RC) circuit. Within this model, equation

$$T_C = T_S \left(1 - e^{-\frac{t}{\tau}} \right) + T_O \quad (1)$$

Describes the increases of temperature, and equation

$$T_C = T_S \cdot e^{-\frac{t}{\tau}} + T_O \quad (2)$$

Describes temperature reductions, where

T_C defines the actual chip temperature,

T_O the idle temperature,

T_S the difference between the stabilization temperature and the idle temp and

τ the time constant- in analogy to electrical RC circuit.

For example, the curve $x(t)$ in Fig. 1 represents a temperature measurement of the i.MX6Q microprocessor under defined parameters. The $y(t)$ represents a least squares fitting of (1) to the chip temperature $x(t)$. For an ambient temperature of 20°C and the parameter T_S , the least squares method leads to the results in Fig. 2 — nearly linear maximum temperatures with respect to the operation frequency and the number of fully loaded, active cores [7].

Fig. 1. i.MX6Q temperature development under 20°C ambient temperature, three active, fully loaded processor cores each running at 792MHz.

Fig. 2. Results for temperature risings TS at 20°C ambient temperature.

The example above shows that it is possible to derive thermal RC models of microprocessors from detailed temperature measurements with different parameters, which can be applied for real software implementations and for simulation purposes. The results can also be used to predict average or maximum temperature reductions per time. The temperature of a microprocessor is always a combination of heat generation and heat disposal [7], thus, thermal characteristics which define the total heat a microprocessor generates are on focus.

3. Characteristics of Closed Loop Control

This basic model is used for a PID-Controller to predict and to control the thermal behavior of the microprocessor under load condition. The PID and the Predictive controllers are integrated into the Linux base scheduler and precisely into its “Schedule” function. The entrance point for the adaptation is a context switch of processes, as it is shown in Fig. 3, which also allows the extension with the prediction algorithm. The predictive controller performs speed scheduling and assigns operation frequencies to processes, in order to ensure that a process runs with a specific frequency. This frequency must be set when the process starts the execution. Other implementations cannot ensure that no frequency switch occurs during the process runtime [7].

Fig. 3. The implementation of PID and predictive controllers inside the schedule function.

All implementations of the controllers consume time, which is added to the Linux base scheduler code. For this reason, detailed measurements were executed to determine the runtime for each step of the PID and the predictive controllers in addition to the frequency scaling function. The “schedule” function typically consumes around ~12.250 ns runtime. The controller function of the PID and the prediction add approximately ~33.307 ns on the original runtime of the “schedule” function can be divided into seven sub-functions, where each function’s runtime is listed in Table 1.

It becomes obvious from Table 1 that the “sensors measurements” function consumes most of the runtime. This function contains three sub-functions, where the temperature and the frequency are read. The function “read temperature”, consumes 78.87% of the “sensor measurement” function. This

consumption is due to the timing behavior of the analog-to-digital-converters (ADCs), as they are used by some processors like the i.MX6Q use ADCs to measure and calculate the temperature. Table 2 shows the runtime of the three sub-functions of the “sensor measurement” function.

The “frequency scaling” function is called from inside the “schedule” function in the kernel and is repeated after each successful scheduling process after the PID and the Predictive controllers. The runtime of “frequency scaling” function is ~31.592 ns and is divided into four sub-functions. The runtime of each sub-function is described in Table 3.

Table 1. Time Consumption of the PID and the Prediction Controllers

sensors measurements	~ 15000 ns
profile	~ 1700 ns
controller - error calculations	~ 520 ns
controller – proportional	~ 500 ns
controller – integral	~ 600 ns
controller - derivative	~ 530 ns
controller - prediction	~ 13000 ns

Table 2. Time Consumption of the Three Sub-functions of the “Sensor Measurement” Function

read temperature	~ 13700 ns
read frequency	~ 1700 ns
calculations	~ 1970 ns

Table 3. Time Consumption of the Four Sub-functions of the “Frequency Scaling” Function

get policy	~ 1350 ns
compute relation	~ 500 ns
apply frequency	~ 29262 ns
update frequency	~ 480 ns

4. Software Tools

Fig. 4. Frequency scaling driver in Linux kernel [7].

Fig. 4 illustrates the frequency scaling mechanism in the Linux kernel, the so called CPUfreq driver [8]. It is responsible for providing an interface for processor related hardware drivers and for CPUfreq governors,

policies and notifiers [8]-[10]. Furthermore, the CPUfreq driver exports information of the operation frequency and control interfaces to the sys virtual file system [8].

A CPUfreq hardware driver is a processor specific (or processor family specific) driver to change operating frequencies and corresponding operating voltages of the processor. Once it has registered itself with to the CPUfreq driver, the CPUfreq driver can use it to apply frequency changes to the processor. In case of multi-core processors, the CPUfreq driver holds for every core an instance [11].

The Linux kernel related components which are responsible for managing frequency changes are CPUfreq governors and policies [8]. Basically, the policies consist of a lower and upper frequency. This is important for processors, which contain an inbuilt frequency scaling mechanism to change operating frequencies on the fly [8], which is not further regarded in this work. If a processor does not support such a feature, a CPUfreq governor performs frequency changes. A CPUfreq governor registers itself to the CPUfreq driver — in multicore processors for all cores [8]. Every CPUfreq governor applies an operating frequency for a core according to a strategy at runtime.

Since the CPUfreq driver exports information and control mechanisms to user space and provides an interface for kernel space, it is possible to apply DVFS with a user application, a Loadable Kernel Module (LKM) and from kernel space in general. In order to achieve this, a LKM is developed to make the profiling and apply DVFS. The LKM enables and disables the PID controller, the predictive controllers.

Besides the LKM, an application program was developed. The application gives the user the possibility to register processes and set their parameters such as the name of the process, the priority, the affinity and the execution mode (whether the process can run in periodic mechanism or normal continuous mechanism). Running a process in a periodic mechanism (e.g. 30s running time and 10s sleep time) gives the time to the cores to cool down and hence gain more time to reach the temperature threshold.

At the end by using this application, the user can set manually the maximum temperature threshold, the desired governor (Ondemand, Interactive, etc.), the operation frequency and the process's run duration.

5. PID Controller

This chapter deals with the implementation of a PID controller to regulate the operation frequency of the i.MX6Q for DTM. In addition, the upcoming subchapters define the implementation in the Linux kernel.

There is one instance of the PID controller, which is independently executed by every core. Fig. 5 shows the architecture of the PID controller, where $w(t)$ denotes the reference value at time t . It relates to the critical temperature threshold. The function $e(t)$ denotes the control deviation, $f(t)$ the control value, $f_R(t)$ the value of the adjusted operation frequency, $d_A(t)$ the disturbance variable, which relates to the ambient temperature and $x(t)$ the operation temperature of the i.MX6Q at time t .

Fig. 5. Architecture of the PID controller.

Fig. 6 illustrates how the temperature of the processor increases up to 82°C after $t = 23$ s. The red line

denotes the critical threshold of 80°C. After this time, the temperature is kept constant and oscillates with a quantization error of $\pm 1\text{K}$ around the critical threshold of 80°C.

Fig. 7 shows the values of $f(t)$ for the same measurement. The chart has a linear scale. Therefore, the red lines in the plot denote the discrete operation frequencies of the CPU [7].

The values of $f(t)$ are correlated with those in $x(t)$. Before time $t = 23\text{s}$, the values of $f(t)$ are high – in particular, the values are above the maximum operation frequency of 996 MHz. Therefore, the operation frequency $f_R(t)$ is at 996 MHz in the time interval of $t \in [0; 23]$ in Fig. 8.

Fig. 6. Functioning of the PID controller: $x(t)$ step-up function [7].

Fig. 7. Functioning of the PID controller: $f(t)$ frequency control value at step-up function [7].

Fig. 8. Functioning of the PID controller: $f_R(t)$ frequency scaling between the available operation frequencies.

When the temperature exceeds the critical temperature level, the values of $f(t)$ decrease. When the value of $f(t)$ undershoots 996MHz, the operation frequency switches to 792MHz in Fig. 8. The longer the operation temperature stays over the critical threshold, the lower the value of $f(t)$ gets and thus, the

operation frequency $f_R(t)$. If the temperature undershoots the critical threshold again, the frequency $f(t)$ rises again.

When the temperature oscillates around the critical threshold, the values of $f(t)$ and $f_R(t)$ oscillate, too. The PID controller is responsible for the oscillation, since the reference value is at 80°C — it regulates the temperature with the help of the operation frequency as control value with the goal to set $x(t) \equiv w(t)$.

6. Predictive Controller

The intention of this work is to give priority to real time processes over conventional processes with respect to the operation frequency, while the thermal constraints and the maximum performance are still observed. The proposed method reduces the operation frequency when conventional processes run on a core to obtain a temperature budget, which is big enough to execute an upcoming real-time task at maximum operation frequency. To accomplish this task, the PID controller must be extended with a mechanism to predict future temperatures (cf. Fig. 9). Thermal prediction methods use the thermal model of the processor and the related thermal characteristics for temperature developments. Important requirements for a thermal model are computational effectivity, high accuracy and simplicity [3]. The requirements for thermal models also apply for thermal predictions, since prediction methods base on thermal models. However, there are conflicts in the design goals. Simple thermal models require less computational effort, but lead to inaccurate calculations. In contrast, complex models are usually computational expensive, but lead to accurate results [7].

Fig. 9. Architecture of the predictive PID controller.

7. Summary and Outlook

This work characterizes the thermal characteristics of a high-performance microprocessor (i.MX6Q quad-core) with regard to total generated heat in dependence of operation frequencies and computational load. It also presents thermal characteristics for cooling mechanisms, which depend on the ambient temperature. In addition, it implements DTM technique into the Linux Operating System for the given microprocessor. The predictive controller is able to prioritize one real-time process on every core.

The predictive controller supports DVFS for one periodical RT process on each core in a multi-core environment, where the time period is known. Future work will extend the support for multiple periodical RT processes with variable period. In such an implementation, the predictive controller must know the upcoming RT process. Thus, a mechanism is necessary, which tracks RT processes on a core, and a data structure which permits the controller to store process references, and to select or search for the next RT process. The Linux scheduler uses RBTs as data structures to handle processes in the kernel. RBTs are also a possibility to hold references to all available RT processes on a core [7].

Acknowledgement

This work was executed in ZIM project KF2786003PR29, which was supported by the Federal Ministry for Economic Affairs and Energy on the basis of a decision by the German Bundestag. The authors are grateful for this support.

References

- [1] Brooks, D., & Martonosi, M. (2001). Dynamic thermal management for high-performance microprocessors. *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture* (pp. 171-182).
- [2] Powell, M. D., Goma, M., & Vijaykumar, T. N. (2004). Heat-and-run: Leveraging SMT and CMP to manage power density through the operation system. *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems* (pp. 260-270).
- [3] Skadron, K., Stan, M. R., Huang, W., Velusamy, S., Sankaranarayanan, K., & Tarjan, D. (2003). Temperature-aware microarchitecture. *Proceedings of the 30th Annual International Symposium on Computer Architecture* (pp. 2-13).
- [4] Wang, S., & Chen, J.-J. (2010). Thermal-aware lifetime reliability in multicore systems. *Proceedings of 11th International Symposium on Quality Electronic Design* (pp. 399-405).
- [5] Kumar, A., Shang, L., Peh, L.-S., & Jha, N. K. (2006). HybDTM: A coordinated hardware-software approach for dynamic thermal management. *Proceedings of 43rd ACM/IEEE Design Automation Conference* (pp. 548-553).
- [6] Freescale Semiconductor. (July 2013). i.MX 6Dual/6Quad Application Processors for Industrial Products (rev 2.3).
- [7] Jäckle, D. (2014). *Dynamic Voltage and Frequency Scaling under Temperature and Soft Real-Time Constraints*. Master Thesis, Albert-Ludwigs-Universität Freiburg im Breisgau.
- [8] Linux Cross Reference. From <http://lxr.free-electrons.com/source/Documentation/cpu-freq/user-guide.txt?v=3.11>
- [9] Linux governors. From <http://lxr.free-electrons.com/source/Documentation/cpu-freq/governors.txt?v=3.11>
- [10] Linux core. From <http://lxr.free-electrons.com/source/Documentation/cpu-freq/core.txt?v=3.11>
- [11] Linux cpu-drivers. From <http://lxr.free-electrons.com/source/Documentation/cpu-freq/cpu-drivers.txt>

Sebouh Toumassian holds a B.Eng degree in 2011 from University of Kalamoon, Deirattiah, Syria and a M.Sc. degree in biomedical engineering from Universität zu Lübeck and Fachhochschule Lübeck in 2013. Since 2014, he is working as a research engineer in the Laboratory of Embedded Systems and Communication Electronics at the University of Applied Sciences.

Axel Sikora received his Dipl.-Ing. degree of electrical engineering (corresponding to M.Sc. degree) in 1993 and a Dipl. Wirt.-Ing. degree of business administration (equivalent to M.B.A.) in 1995, both at Aachen Technical University, Rheinisch-Westfälische Technische Hochschule, RWTH Aachen. From 1994 to 1996, he did a Ph.D. degree in electrical engineering at the Fraunhofer Institute of Microelectronics Circuits and Systems, Duisburg, with a thesis on the optimization of digital device and circuit design for

high-temperature SOI-technologies.

Afterwards, he has been working in the Telecommunications and Semiconductor Industry from 1996 to 1999. He worked as a project manager for data services business at O.Tel.O Communications GmbH & Co KG in Cologne from 1996 to 1997, and joined NEC Electronics (Europe) GmbH in Düsseldorf, where his last position was the group leader of system LSI products.

In 1999, he joined at the Baden-Wuerttemberg Cooperative State University Loerrach as a professor. There he helped to found the Department of Information Technology that he headed from 2001 onwards. In 2011, he joined the Faculty of Electrical and Information Technology of Offenburg University of Applied Sciences, where he now heads the Laboratory of Embedded Systems and Communication Electronics.

Since his early day, the research activities of Dr. Sikora were around safe and secure, wired and wireless, efficient and modular network developments using embedded systems. These developments are now popular as Internet of Things (IoT) or Cyber Physical Systems (CPS). As these technologies are quite application-agnostic, projects are in various fields. Major projects were executed in traffic automation, i.e. Car-to-Car-communication, industrial automation, i.e. Industry 4.0, healthcare, i.e. telehealth and telecare, home and building automation, i.e. smart home and smart building, and energy generation and distribution, i.e. smart metering and smart grid.

He founded the Steinbeis Transfer Center Embedded Design and Networking (STZEDN) in 2002, which helps companies in research and development projects around the design of wireless and wired embedded networking solution. Until today, he is leading the Transfer Center, which was founded into STACKFORCE GmbH, a private company, in 2014.

Dr. Sikora is the author, co-author, editor and co-editor of several textbooks and numerous scientific papers. He is a reviewer, member of program committees, he is also a invited speaker to many national and international conferences.