On Chip Communication-Architecture Based Thermal Management for SoCs

Aseem Gupta†‡, Sudeep Pasricha±, Nikil Dutt†, Fadi Kurdahi†, Kamal Khouri‡, Magdy Abadir‡

† University of California, Irvine
Irvine, CA 92697 USA
{aseemg, dutt, kurdahi}@uci.edu

± Colorado State University
Fort Collins, CO 80523 USA
sudeep@engr.colostate.edu

‡Freescale Semiconductor Inc.
Austin, TX 78729 USA
{kamal.khouri,m.abadir}@freescale.com

Abstract— In current Systems-on-Chip (SoC) designs, managing peak temperature is critical to ensure operation without failure. Our novel Communication Architecture Based Thermal Management (CBTM) scheme manages thermal behavior of components by delaying the execution of chosen IP-blocks or components by regulating the flow of data over the on-chip communication bus. This temperature aware traffic flow over the bus is achieved by dynamically changing the communication priority table in response to thermal readings from sensors. With CBTM, the temperatures of individual components can be controlled selectively. In this paper we demonstrate the effectiveness of CBTM on four industrial size SoC designs and also evaluate its performance impact. We observe that CBTM maintained thermal thresholds and reduced the peak temperature of an SoC by as much as 29°C.

I. INTRODUCTION

Process scaling and high levels of integration in SoC designs have led to higher power densities (Watt per mm^2) which in turn lead to higher operating temperatures. High operating temperatures have many unfavorable consequences for SoC designs: (i) Increased probability of timing violations because of higher signal propagation delay and switching time, (ii) Reduced lifetime because of phenomena such as electro-migration, (iii) Lower rated frequencies of designs because of higher delays, (iv) Increased leakage power due to super-linear relationship with temperature, (v) Expensive cooling mechanisms are required. (vi) Increase in error rate for memory (SRAM) accesses.

While static (offline) temperature management schemes can be useful, they typically are unable to account for dynamic operating conditions. Efficient run-time temperature management strategies are needed to ensure that the on-chip temperature remains below an acceptable threshold temperature. The on-chip communication architecture in a typical SoC has a *global view* of all the IP-blocks or components in the system. Our Communication Architecture Based Thermal Management (**CBTM**) technique uses temperature aware arbitration to reshape the data traffic over the bus and modulate the execution of components for which the temperatures are critically high. This is achieved by dynamically altering the communication priorities of the system components when their temperatures need to be managed. To the best of our knowledge, this is the first contribution in the direction of thermal management using the on-chip communication architecture.

An SoC can use CBTM in combination with other dynamic thermal management techniques such as voltage frequency and clock frequency scaling. If designers aim to implement selective voltage and frequency scaling on components, the design complexity increases by many folds. With our proposed CBTM, there is a minor overhead in controlling the temperatures of *each individual component* in the SoC.

II. RELATED WORK

[1] provides a comprehensive overview and comparison of different Dynamic Thermal Management (DTM) techniques. The closest related work to our proposed approach is [2], which proposes using communication architectures for power management. Their system level communication based power management (CBPM) approach tailors the run time system power profile to ensure efficient battery discharge for maximized battery life. This is done by exercising regulatory control over the system components, by proactively blocking transfer of data to these components over the communication bus and thus also blocking the execution of these components. In our work, we focus on regulating the temperature of each individual IP-block or core in the SoC such that the temperature of any IP-block does not exceed a threshold temperature. Even though temperature and power density are closely related, it is not necessary that higher power densities will lead to higher temperature because of effects of thermal diffusion among neighboring blocks. This has also been shown in [3]. Another difference is that in [2], the arbiter needs to have on-board power models of the different IP-blocks in order to estimate the total power dissipation of the chip. In our work, we use thermal sensors which are increasingly present in multiple locations in modern VLSI designs such as PowerPC [4].

Our technique uses the on-chip communication architecture that can exploit the system wide knowledge of the complete SoC (since all the IP-blocks are tied to the communication architecture). Our technique has low implementation overhead, does not require complex level shifters which are hard to verify and thus unlike other techniques it does not require special tools for verification and testing. Another advantage is that our technique can easily manage temperatures of select individual IP-blocks.

III. DYNAMIC THERMAL MANAGEMENT

At any given time, different components in a SoC have different temperatures because of varying activity levels and heat diffusion from neighboring blocks. The solid curve in Fig. 1 shows how the maximum temperature inside an SoC changes over a period of time when no DTM is done. Thus the SoC needs to be designed at a higher temperature with a safety margin. The dashed curve shows the maximum temperature when a DTM technique is used. When the SoC temperature exceeds a DTM trigger level, the DTM technique kicks in at point Ⓐ. Typically, there is a *Response Delay* before the temperatures actually start showing the effect of DTM. When the temperatures fall below the trigger level at point Ⓑ, DTM must be deactivated. DTM should be deactivated only after a *Deactivation Delay* so that the peak temperature does not fluctuate around the trigger causing DTM to be activated frequently. Thus, DTM can lower the design temperature of the SoC.

IV. COMMUNICATION BASED THERMAL MANAGEMENT

A. Background

In a bus based on-chip communication architecture, such as one in Fig. 2, a communication transaction is initiated by *masters* (e.g., cores & DSPs), and this data request is serviced by *slaves* (e.g., memories). Since multiple masters share the bus, an *arbiter* ensures that

978-1-4244-2782-6/09/$25.00 ©2009 IEEE

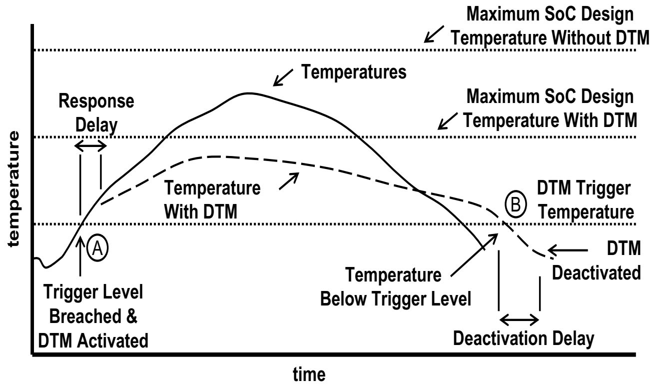

Fig. 1. Principles of Dynamic Thermal Management (DTM)

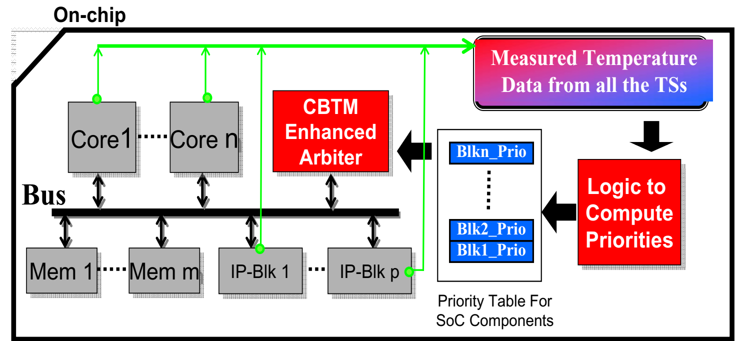

Fig. 2. Thermal Management Using Bus Based Communication Architecture

only one master has the control of the bus at any given time. Different arbitration mechanisms such as round robin scheme, priority based scheme etc. can be used. In a priority based scheme, the arbiter allocates the control of the bus to the request originating from the master with the highest priority. Priorities of system components can be either statically assigned by the designer or can be dynamically changing at run time.

B. CBTM Approach

The basic concept behind Communication Based Thermal Management (CBTM) is that it regulates the execution of different components in the SoC in order to limit their temperatures. CBTM achieves this by regulating the flow of data over the bus. The arbiter has a global view of the system because all the components are attached to the bus. This is utilized by CBTM in extending the role of the arbiter beyond simply deciding which master gains control of the bus. The arbiter manages the execution of select components. The temperature readings for system components can be obtained by using Digital Thermal Sensors (DTS) [6] and can be reported to the CBTM logic as shown in Fig. 2. Details on the implementation are in Section *E*. When the temperature of a component exceeds a designer specified trigger temperature, the CBTM logic reduces the communication priority of that component. When the priority of a master is reduced, it gets lesser control of the bus. This leads to a 'blocking' of the data requests serviced from the master. A 'blocked' master does not execute until its data request has been serviced. We assume that internal circuitry exists within components to ensure that they dissipate significantly less power when blocked and not executing. Thus, when the priority of a master is decreased, its execution gets blocked more often and the power dissipation of this component in a given time interval is decreased. This causes a negative performance impact on the data throughputs of the components over the bus which leads to reduction in the overall SoC performance. It should be noted that a reduction in the communication priority of a component does not mean that the component's data requests over the bus will not be serviced at all, but rather that the number of granted service requests from this component in a time interval are decreased. In this interval more requests from other components will be granted control over the bus by the arbiter. The reduced power dissipation leads to an arrest of the rising temperatures. The penalty on the communication priority of a master component is imposed by CBTM when its temperature exceeds the trigger level. However, this penalty is not permanent and the priorities do not stay reduced for rest of the execution. Section *D* discusses policies for penalty computation. The CBTM logic continues to monitor the temperatures of the components and when the temperatures fall below the trigger level, the priorities are restored to their original values after *Deactivation Delay*. Thus:

if $(Temp_X >= Temp_{Trigger})$
$\quad New\ Priority_X \leftarrow Original\ Priority_X - Penalty$
if $(Temp_X < Temp_{Trigger})$
$\quad New\ Priority_X \leftarrow Original\ Priority_X$

C. Discussion on CBTM

(i) Can there be a scenario where no component gets access to the bus and the bus remains unused while data requests are pending? The arbiter grants control of the bus to the request originating from the master with the highest priority. Even when the priorities are penalized by CBTM, requests originating from these components still remain in the fray for arbitration. If there is at least one data request pending with the arbiter it will grant control of the bus to this request.

(ii) There could be a scenario where temperatures of many components exceed trigger level and CBTM penalizes their communication priorities. But if many components have their priorities reduced, the components will not remain execution-blocked for sufficient time because low priority components will continue to get bus-control more often and CBTM will be ineffective. Though unlikely, to avoid this the CBTM enhanced arbiter has an emergency turn off mechanism. If reduction in priorities does not arrest the increasing temperatures and if temperatures reach hazardously close to the threshold temperature, the arbiter can prevent any master from accessing the bus. The execution of the complete SoC is stalled until it is safe for resumption. Other DTM techniques also have such shut off mechanisms [1].

(iii) Use of the above described emergency turn off mechanism will also be needed if a master is heavily compute intensive and hardly needs to communicate over the shared bus. Typical SoC behavior has a very low probability for occurrence of cases (ii) and (iii).

(iv) Can there be a scenario where a component is starved all throughout the course of execution? Even without CBTM, typical arbiters also grant low priority requests which have been pending for a long period of time. This is done by keeping a record of the wait time of a request and after a designer specified period, a low priority request can be granted access to the bus.

D. CBTM Policies

When a component's temperature exceeds the trigger level, the CBTM must penalize or reduce its priority. If the *penalty is too high* then the bus accesses from this component are reduced by a large amount. A quick fall in the temperature of this component is accompanied by a higher performance impact. If the *penalty is too low* then there will be a sluggish fall in the temperature but the performance impact will be marginal. User has the option to use any heuristic and we propose two CBTM policies for penalty computation:

Fixed Penalty Policy (FPP): In FPP, the penalty is a constant number. The choice of this penalty constant is up to the designer.

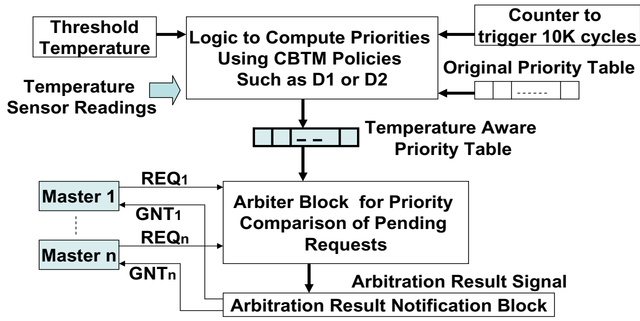

Fig. 3. CBTM Implementation

Adaptive Penalty Policy (APP): In APP, the penalty is a sum of: (i) a fixed constant term, and (ii) a term that is directly proportional to the rate at which the temperature is increasing. The higher the rate of increase in temperature the more aggressive is the penalty. This penalty computation policy is very effective in controlling fast rising temperatures when they are above the trigger temperature level. The fixed contributor ensures that there is a non-zero penalty even when the temperatures are not increasing but are above the trigger level.

E. Implementation of CBTM and its Overhead

The implementation of CBTM is very simple, has low overhead, needs no alteration of communication protocols, and has a much lower design complexity than other DTM techniques. As shown in Fig. 3 the temperatures of the system components are measured using Digital Thermal Sensors (DTS) [6]. The underlying communication bus itself can be used to transfer the DTS readings to the arbiter over the bus. Since the temperature information needs to be polled once every about 10K cycles, the overhead of this extra communication of temperature reading from the components to the arbiter over the bus is negligible. Because of thermal RC time constant, time intervals in the order of few hundred microseconds are needed for any considerable change in temperature [5].

Let us now assess the overhead of implementing CBTM. A 'CBTM logic' must poll the temperatures of system components every 10K cycles. The temperature values can be transferred over the bus. This requires a counter which can initiate the polling after every 10K cycles. The logic needs extra registers to store the temperature values. Another set of registers are needed to store the original priority values of the components in the SoC. The number of additional registers required are equal to twice the number of system components connected to the bus. CBTM also requires one Digital Thermal Sensor (DTS) per system component. A single DTS consists of a few PNP transistors and one 8-bit analog-to-digital converter. The CBTM logic checks the temperature registers to see if any of them have exceeded the thermal trigger level and computes penalties based on policies in Section *D*. The logic mainly comprises of a 4-bit (for up to 15 components) subtractor. The arbiter reads the priorities of the master components as shown in Fig. 3.

The logic which computes the priorities does not slow the arbitration because the CBTM logic is not in serial execution path of the arbiter and *its execution delay has no effect on the arbiter*, as shown in Fig. 3. Priority calculation by the CBTM logic is done once every 10K cycles. The total overhead of CBTM in terms of area is very small and will be less than 0.1% for an average sized industrial SoC. With our proposed CBTM, we are able to perform fine-grained thermal management in a SoC with minimum overhead by controlling the temperatures of each individual component in the SoC. It is also important to note that *CBTM is a system level approach and can be used alongside in combination with other DTM techniques*.

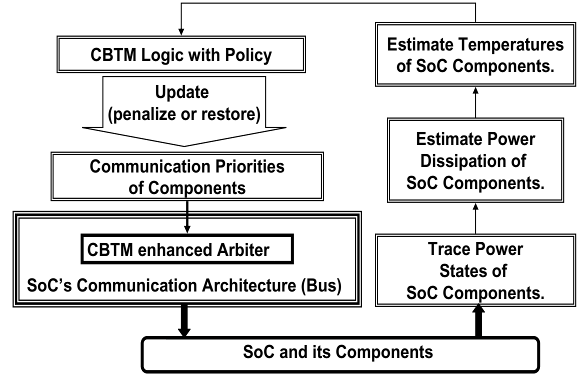

Fig. 4. CBTM Simulation Setup

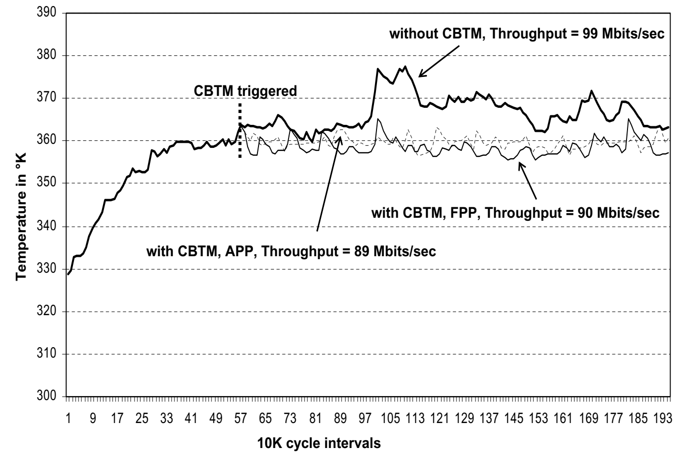

Fig. 5. Maximum Temperature Profiles for SoC-1 Without CBTM and With CBTM for Different Policies

V. Simulation and Results

A. Simulation Setup for CBTM

Since communication behavior is unpredictable due to dynamic access requests and arbitration, a simulation based approach was used and is shown in Fig. 4. We have used the ARM AMBA AHB [7] bus. We used SystemC based behavioral models of components, bus architecture, and arbiter. We used relatively simple component power estimators similar to [8]. For each component we estimate the 'busy' and 'idle' state power by averaging gate-level cycle by cycle power. The two state power estimator model is more than 95% accurate over a set of benchmarks [8]. However, *the actual implementation of CBTM does not need any power estimators*. We used HotSpot [9] for estimating the thermal behavior of the components. The inputs to HotSpot are: the power profile of components, the floorplan of the SoC, and a library of physical properties of silicon. However, the *actual implementation of CBTM does not require any thermal simulator because of use of thermal sensors*.

B. Peak Temperature Reduction

We evaluated CBTM on four industrial size SoC designs. For each of the four SoCs we show the peak temperature of the SoC under three cases: (i) Without CBTM, (ii) With CBTM running FPP, and (iii) With CBTM running APP. Fig. 5 shows results of SoC-1 which has a total of 25 system components, out of which 9 are master IP-blocks and others are slave IP-blocks such as memories. The trigger temperature was 360K. The three curves follow the same path until the trigger temperature is breached and CBTM is triggered. Without CBTM, the maximum temperature reached is about 378K, which fell to 365K for FPP-CBTM and 363Kfor APP-CBTM. This leads to a direct reduction in: design temperature, cooling cost, package cost,

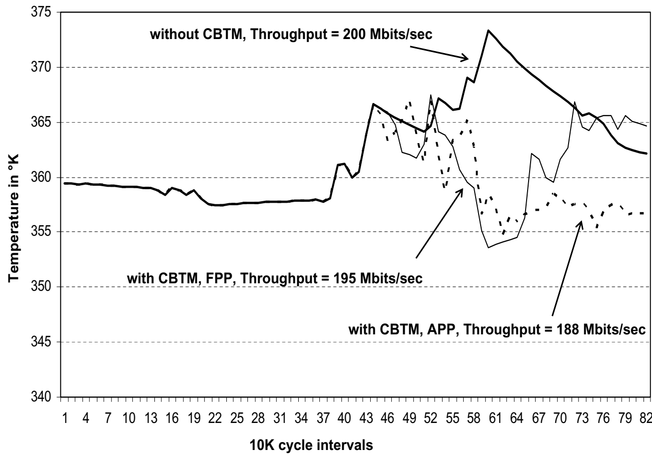

Fig. 6. Maximum Temperature Profiles for SoC-2 Without CBTM and With CBTM for Different Policies

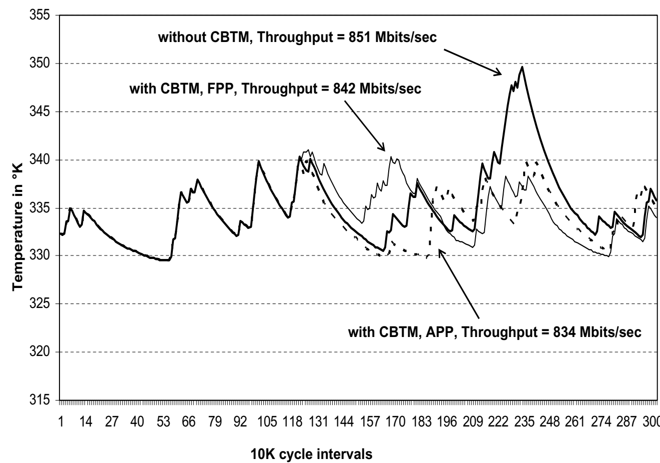

Fig. 7. Maximum Temperature Profiles for SoC-3 Without CBTM and With CBTM for Different Policies

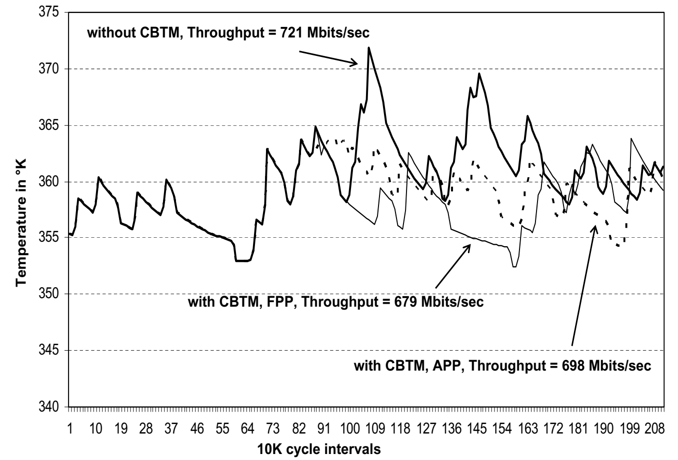

Fig. 8. Maximum Temperature Profiles for SoC-4 Without CBTM and With CBTM for Different Policies

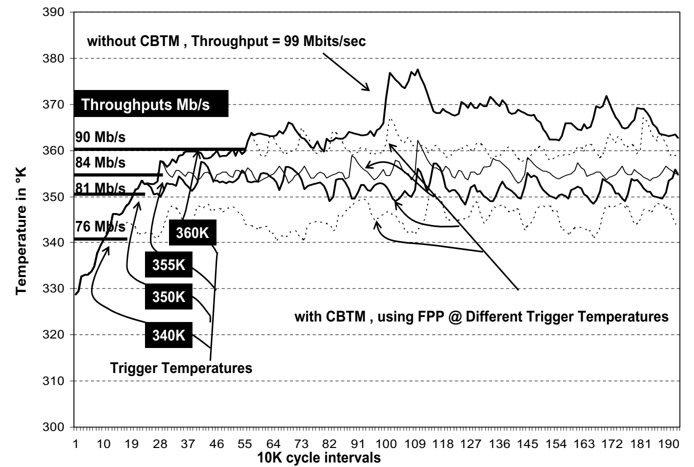

Fig. 9. Maximum Temperature Profiles for SoC-1 Without CBTM and With CBTM at Different Trigger Temperatures

leakage and fan power. To evaluate the impact on performance due to CBTM, we report the data throughputs of critical components over the bus. The data throughput penalty is about 10% for the two CBTM techniques. Fig. 6 shows the results for SoC-2 with a total of 34 system components, out of which 12 are master components and others are slave components. The reduction in the peak temperature comes with a performance impact of 6%. Figs. 7 and 8 show the results for SoC-3 and SoC-4 respectively. SoC-3 has a total of 29 components with 12 master components and SoC-4 has 8 master IP-blocks and 14 slave IP-blocks. In all these results we observe that the performance impact is very minimal.

C. Choice of Trigger Level Temperature

The choice of trigger temperature for CBTM ultimately affects the maximum design temperature of the SoC. For SoC-1, Fig. 9 shows peak temperatures without CBTM and with FPP-CBTM at four different trigger temperatures. Low trigger temperature leads to lower peak temperatures and higher performance impact. The peak temperature dropped from 378K without CBTM to 349K with FPP-CBTM at 340K trigger temperature, a reduction of 29K ($\equiv 29^o$Celsius). Designer must do a careful selection of the trigger temperature based on tradeoff between peak temperature and performance impact.

VI. Conclusion

This paper took the first step in the direction of thermal management using the system level on-chip communication architecture with CBTM. CBTM relies on the arbiter to delay the execution of components for which the temperatures are near a threshold temperature. This is achieved by dynamically reducing the communication priorities of the system components when their temperatures exceed the thermal trigger. These priorities are restored when the thermal emergency is averted. CBTM allows temperature control of individual components. We demonstrated the effectiveness of CBTM on several industrial size SoC designs and also evaluated the impact on the performance of the SoC. We noted that CBTM implementation is simple with a low overhead.

Acknowledgement: This work was partially supported by SRC contract 1617. The authors would like to thank Kanishka Lahiri of AMD.

References

[1] M. Martonosi et al., "Dynamic Thermal Management for High-Performance Microprocessors," *Proceedings HPCA*, 2001.
[2] K. Lahiri,A. Raghunathan, S. Dey "Communication Architecture Based Power Management for Battery Efficient System Design," *Proceedings of Design Automation Conference*, June 2002.
[3] Y. Han et al., "Temperature Aware Floorplanning,"*Workshop on Temperature Aware Computer Systems,* 2005.
[4] K. Skadron et al.,"Monitoring Temperature in FPGA based SoCs,"*Proceedings of International Conference of Computer Design*, 2005.
[5] K. Skadron et al.,"Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management,"*Proceedings of International Symposium on High-Performance Computer Architecture*, 2002.
[6] K. Roy et al.,"A CMOS Thermal Sensor and Its Applications in Temperature,"*International Symposium on Quality Electronic Design*, 2006.
[7] http://www.arm.com/products/solutions/AMBAHomePage.html
[8] I. Lee, et al.,"PowerViP: SoC Power Estimation Framework at Transaction Level,"*Proceedings of Asia South Pacific Design Automation Conference*, 2006.
[9] W. Huang et al.,"HotSpot: Thermal Modeling for CMOS VLSI Systems,"*IEEE Transactions on Component Packaging and Manufacturing Technology*, 2005.