Adaptive Temperature-Aware SoC Test Scheduling Considering Process Variation

Nima Aghaee, Zebo Peng, and Petru Eles
Embedded Systems Laboratory (ESLAB), Linkoping University, Sweden
{nima.aghaee, zebo.peng, petru.eles}@liu.se

Abstract—High temperature and process variation are undesirable effects for modern systems-on-chip. The high temperature is a prominent issue during test and should be taken care of during the test process. Modern SoCs, affected by large process variation, experience rapid and large temperature deviations and, therefore, a traditional static test schedule which is unaware of these deviations will be suboptimal in terms of speed and/or thermal-safety. This paper presents an adaptive test scheduling method which addresses the temperature deviations and acts accordingly in order to improve the test speed and thermal-safety. The proposed method is divided into a computationally intense offline-phase, and a very simple online-phase. In the offline-phase a schedule tree is constructed, and in the online-phase the appropriate path in the schedule tree is traversed, step by step and based on temperature sensor readings. Experiments have demonstrated the efficiency of the proposed method.

Keywords: SoC test scheduling; adaptive test; temperature aware; process variation

I. INTRODUCTION

Deep submicron integration has brought two challenges to the microelectronics world, high power density and process variation [1, 2]. The power density for a System-on-Chip (SoC) in test mode is considerably higher than that in normal mode [3-6] and so is the risk of overheating. Simple introduction of cooling cycles into the test schedule will prevent overheating, but, it will result in excessively long test application time. Therefore, efficient temperature-aware test scheduling techniques have been developed in order to minimize the test application time and avoid overheating [7-10]. These methods neglect, however, the thermal consequences of the process variation and focus only on minimization of the test application time while satisfying a given thermal constraint [7-10].

The negative thermal consequences of process variation include deviations which result in unpredictability of the thermal behavior of the chip. The difference between the expected temperature (estimated by simulation) and the actual temperature (measured by sensors) is called temperature error, which captures all errors generated due to different temperature-related effects. They include ambient temperature fluctuations, voltage variations, and process variation. For traditional technologies, temperature error is small enough to be negligible or to enable design considering only the worst case scenario [7-10]. The general trend of increase in power density and process variation will eventually lead to a situation where thermal effects due to process variation can no longer be ignored. Therefore, they should be taken into account in developing efficient test solutions.

In [11], two process variation aware methods are proposed in order to maximize the test throughput which considers the thermal-safety as a part of the optimization objective. However, one of the proposed methods in [11] does not react to temperature deviations, and the other does not handle intra-chip process variation thermal effects and rapid temperature deviations. In this paper an adaptive test scheduling method is introduced which navigates the tests according to the intra-chip process variation thermal effects and temporal deviations in thermal behavior of the chip. It makes use of multiple on-chip temperature sensors to provide intra-chip temperature information.

Integration of such sensors are already practical, for example Power5 is reported to have 24 sensors in year 2004 [12]. A variety of mechanisms to access the sensors during test are proposed [13, 14], and still one may think of a different one. The proposed approach in this paper assumes an overhead for sensor access and tries to reduce and limit the number of sensor accesses.

A thermal-aware test scheduling technique using on-chip temperature sensors is proposed in [14]. It assumes that each individual test runs to completion, which means that before start of each test, a large cooling time is required to guarantee thermal safety. This leads to excessively long test application times. Moreover, this approach does not handle the long and power intensive tests which will lead to overheating if applied to completion. The thermal simulation technique employed in [14] is based on the superposition principle and only takes the temperature-independent component of the static power (leakage) into account [14].

The method proposed in this paper is based on partitioning and interleaving [10], which reduces and/or utilizes the cooling times in order to decrease the overall test application time. It also handles the long and power intensive tests which are not thermal-safe.

The proposed method generates a near optimal schedule tree at design time (offline-phase). During testing (online-phase), each chip traverses the schedule tree, starting from the tree's root and ending at one of the tree's leaves, depending on the actual temperatures. The schedule indicates when a core is testing and when it is in the cooling state. The order of the test sequences is untouched and the schedule tree occupies a relatively small memory. Traversing the schedule tree requires a very small delay overhead for jumping from one point in the schedule tables to another point. This way, the complexity is moved into the offline-phase and the memory/delay overhead of the online-phase is minor. To our knowledge, this paper is the first work to present an approach which incorporates the on-chip temperature sensors data, repetitively during test, in order to adapt to the temperature deviations caused by process variation and to achieve a superior test performance.

The rest of the paper is organized as follows. A motivational example is given in section II. Section III provides the problem

formulation. Then, the cost function is introduced in section IV. Section V describes the temperature error model. The linear schedule tables are discussed in section VI. The proposed method is presented in section VII. Experimental results are represented in section VIII. Section IX presents the conclusion. A summary of notations is given at the end of the paper.

II. MOTIVATIONAL EXAMPLE

Assume that there are two instances, *o* and *x*, from a set of chips manufactured for a given design. When the temperature error is negligible, the temperatures of *o* and *x* are equal and the same offline test schedule S1 is used for both of them. As illustrated in Fig. 1(a), both *o* and *x* are tested without overheating, since the test schedule includes cooling periods at the time points when the thermal simulator indicates that the chip temperature will exceed the limit.

Under a certain temperature error condition, due to process variation, the thermal responses of the different chips to the same test sequence will, however, be different. Now, assume that chip *x* is warmer than expected, while chip *o* behaves normally. As illustrated in Fig. 1(b), *x* will overheat. To prevent this, a more conservative offline schedule S2 has to be designed based on the thermal profile of *x*, for both chips, as illustrated in Fig. 1(c). This new schedule will avoid overheating, but will lead to longer test application time (TAT2 compared with TAT1). For chip *o*, this test application time is unnecessarily long, since the original schedule, S1, in Fig. 1(a) is a safe schedule for this particular chip. For a set of manufactured chips with large temperature variations, in order to generate a global conservative offline schedule, the hottest chip will be used to determine the test schedule. This test schedule will introduce too long cooling periods for most of the chips, leading to an inefficient test process.

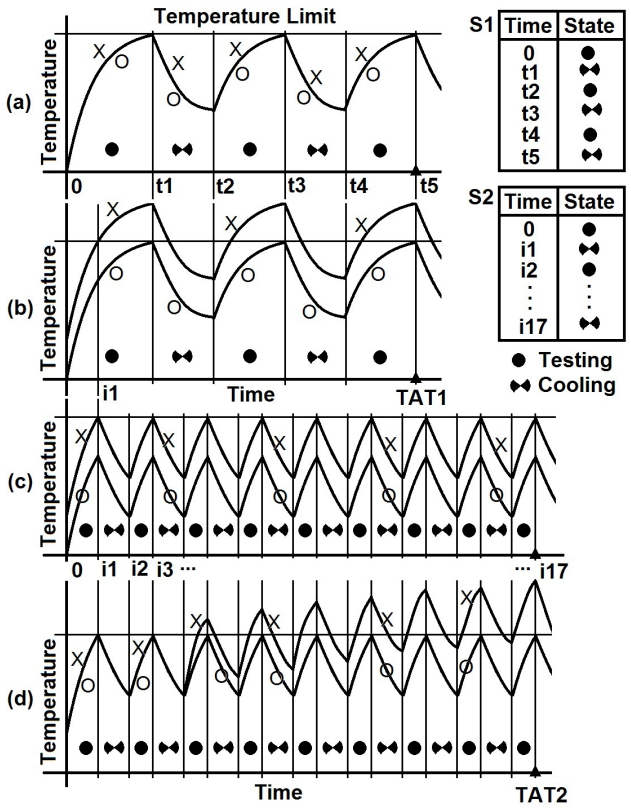

Figure 1. Test schedule examples (curves are only illustrative).

We have proposed a technique, in [11], to address the above problem with the help of a chip classification scheme. This scheme consists of several test schedules for different temperature error ranges. After applying a short test sequence, the actual temperature is read using a sensor and depending on the actual temperature, the proper test schedule is selected. Therefore, the hotter chips will use a test schedule with more cooling, while the cooler chips will have less cooling. The overheating issue is solved and the test application time will not be made unnecessarily long. This approach works fine under the assumption that the thermal behavior of chips is time invariant, as shown in Fig. 1(a-c).

However, in the real world with large process variation, the thermal behavior is time variant and the technique presented in [11] will not be able to achieve high-quality schedules. The variation of thermal response with time is illustrated in Fig. 1(d). In this case, the temperature of chip *x* gradually lifts up compared to chip *o*, and *x* eventually overheats. A scheduling method capable of capturing temporal deviations is therefore required to deal with this new error situation. The temperature behavior given in Fig. 1(d) is captured in Fig. 2(a) with more details. The lift up of the temperatures of chip *x* starts at t3, as shown in Fig. 2(a). Since *x* will only overheat after t4, both chips can be safely tested with schedule S1 up to t4. At t4, the actual temperature of the chip under test can be obtained via sensors. The actual temperature can then be compared to a *Threshold* and two different situations can be identified:

$$\begin{cases} x; & if\ Temperature(t4) > Threshold \\ o; & if\ Temperature(t4) \leq Threshold \end{cases}$$

For the rest of the test, after t4, two dedicated schedules, S2 and S3, are generated in the offline-phase for *o* and *x*, respectively. Therefore, in the online-phase the test of *o* continues using schedule S2, as in Fig. 2(a), and the test of *x* continues using schedule S3, as in Fig. 2(b).

In this illustrative example, at the end of S1, the schedule does a branching to either S2 or S3 based on the actual temperature. This information and the branching condition can be captured in a branching table, B1 in Fig. 2. As shown in Fig. 2(a), *o* is tested initially with S1 and then with S2, while, as shown in Fig. 2(b), *x* is initially tested with S1 and then with a more conservative schedule, S3. The segments of the schedule which are executed sequentially without branching are called linear schedules. An adaptive test schedule consists therefore of a number of branching tables in addition to multiple linear schedule tables. Note that the original test sequences are saved elsewhere in an intact order without being duplicated.

Before going further into details of the proposed method, an example is given to illustrate a realistic test scenario and temperature curves for the cores. Consider an individual chip with four cores. The proposed method will determine the state (testing or cooling) of each core and therefore the resulted temperature curves will look like curves given in Fig. 3, which are taken from [10]. Testing states are in general shorter than cooling states. Testing is interrupted when a core's temperature reaches the *high temperature threshold*. As it is shown in Fig. 3, more than one core may be tested at the same time, e.g. core 1 and core 3 at 20 us. Also, cores will utilize the cooling times of each other in order to run the test, e.g. at time 80 us core 1 and core 2 are utilizing the cooling time of core 3 and core 4.

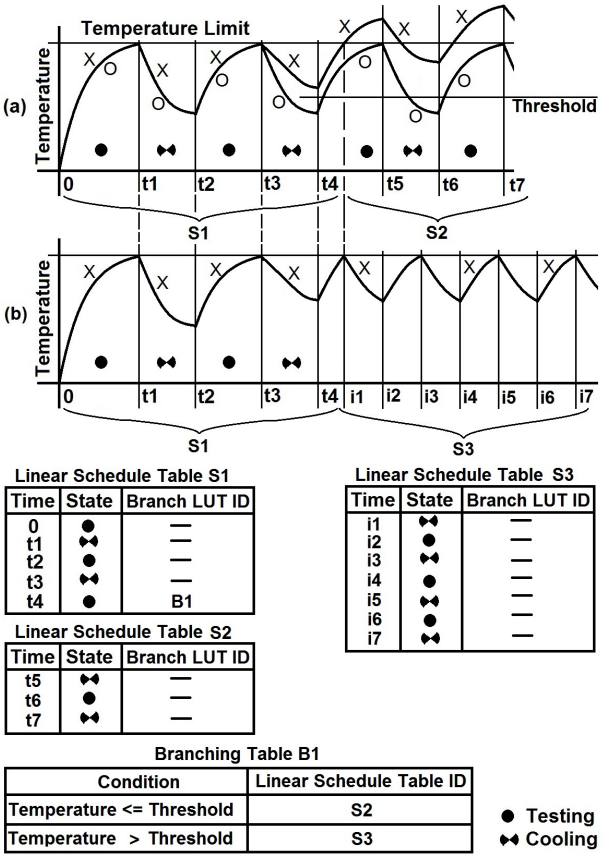

Figure 2. Schedule and branching tables (curves are only illustrative).

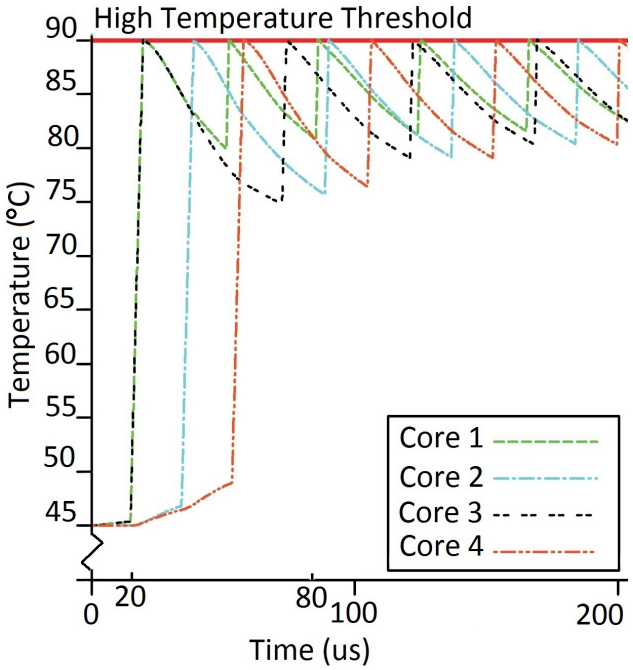

Figure 3. Temperature curves for a four core chip under test.

III. Problem Formulation

Our goal is to generate an efficient adaptive test schedule, as discussed in the previous section, offline. This is formulated as an optimization problem. The input consists of a SoC design with a set of cores and their corresponding test sequences. The floor plan, the thermal parameters, the static power parameters, and the dynamic power parameters for the chip are given. The statistical data that models the deviations are also given. The adaptive test schedule should be generated to minimize the test application time and overheating. These objectives are captured by a cost function which expresses the overall efficiency of the generated test schedule, and will be discussed in the next section.

The test schedule should be generated under two constraints. The first one is the available test bus width. The test bus width limits the number of cores tested in parallel. The second constraint is the available ATE memory which limits the linear schedule tables and branching tables.

IV. Cost Function

In order to prevent overheating of a chip which responses to the test schedules with higher temperature, more cooling cycles are required. These extra cooling cycles increase the test application time as shown in Fig. 1, which leads to low utilization of the test equipment. On the other hand, if not enough cooling is introduced some chips will be overheated and damaged during test. In this paper a comprehensive *cost function* is defined by combining the cost of the overheated chips and the cost of the test facility operation, as follows:

$$cost\ function = CTF \cdot \frac{1}{TT} + POC \cdot \frac{TOP}{1-TOP}. \quad (1)$$

The *cost function* consists of two terms. The first one is related to the test environment cost, which is defined as the Cost of the Test Facility per time unit (CTF) divided by the Test Throughput (TT). The *cost of the test facility per time unit* depends on the cost of the ATE machines, their maintenance costs, and other similar factors. The *test throughput* is the *applied test size* per time unit.

The second term of the cost function is related to the price of the overheated chips, which is the product of the Price of One Chip (POC) and the number of overheated chips. The number of overheated chips is calculated based on the Test Overheating Probability (TOP), which represents the number of overheated chips per number of chips entering the test facility.

In Equation (1), *test overheating probability* (TOP) is divided by (1-TOP) in order to give the number of overheated chips per number of non-overheated chips (at the test facility exit). The *cost of the test facility per time unit* (CTF) and the *price of one chip* (POC) depend on the particular manufacturing and test facility and on the particular SoC. To have a simple model for the *test throughput* (TT), assume that the given test facility is characterized by its overall Affordable Test Time Per Second (ATTPS) and Test Handling Time (THT). *Affordable test time per second* and *test handling time* are specific to the ATE machines. The *test handling time* represents the wasted times that chips are not actually under test, e.g. placing, connecting, and detaching the chips. The *test throughput* (TT) which depends on the Applied Test Size (ATS) and Test Application Time (TAT) is expressed as:

$$TT = \frac{ATTPS \cdot \mathrm{ATS}}{TAT+THT} \cdot (1 - TOP). \quad (2)$$

In order to simplify the *cost function*, the Normalized Test Throughput (NTT) is defined by normalizing the *test throughput*

(TT) to the *affordable test time per second* (ATTPS) and assuming that the *test handling time* (THT) is negligible. The *normalized test throughput* (NTT) is expressed as:

$$NTT = \frac{\text{ATS}}{TAT} \cdot (1 - TOP). \quad (3)$$

Normalizing the *cost function* to the *cost of the test facility per time unit* (CTF) and using the *normalized test throughput* (NTT) instead of the *test throughput* (TT) will result in the Normalized Cost Function (NCF) as expressed in (4). The ratio of the *price of one chip* (POC) to the *cost of the test facility per time unit* (CTF) is then called the Balancing Coefficient (BC), which is used to balance the cost of the overheated chips against the cost of the test facility.

$$NCF = \frac{\text{TAT}}{ATS \cdot (1 - TOP)} + BC \cdot \frac{TOP}{1 - TOP} \quad (4)$$

V. Temperature Error Model

The temperature error has various sources including process variation, ambient temperature fluctuations, voltage variations, simulator errors, and the temperature dependent errors, e.g. static power (leakage). Temperature error is broadly categorized into spatial temperature error and temporal temperature error. For example, spatial temperature error shows that two chips have different temperature error values; and temporal temperature error, for example, shows that a certain chip has a lower temperature error at the end of the test, compared to its temperature error at the middle of the test.

A temperature error model gives the probabilities of the temperature error values for each core and for each test cycle. The spatial temperature error model gives the initial error distribution and the temporal temperature error model is used to recursively estimate the error distribution for the next test cycle. The spatial temperature error is the statistical distribution of errors within a certain error range and resolution for a certain core. For example, the range and the resolution for a SoC design can be -20℃ to $+20$℃ and 0.1℃, respectively. In this example, the size of the data set, M, is 400 for each core and $M \times C$ for a SoC having C cores. As an example, the spatial temperature error model will indicate that just after the warm up, the probability of an error equal to -2.3℃ for core 1 is 0.001, while for core 2 is 0.02.

The temporal temperature error model is a discrete-time model, i.e., the temperature error value is fixed during a period and then it changes discretely from one period to the next. Therefore, the temporal temperature error model has two pieces of information, the period which is called *temporal error period* and a table of error change probabilities. The temporal temperature error table gives the probability of a certain error change and it has a certain range and resolution. For example, the probability of an error change equal to $+0.6$℃ happening for core 2, is shown to be 0.015 by its temporal temperature error table. Assume that the *temporal error period* is 1ms, and that the error value for core 2 is measured to be -5.3℃ at time 0, as shown in Fig. 4. The error will remain equal to -5.3℃ up to time 1ms (0 + *temporal error period*). Then, after the time 1ms the exact error is not known any more. However the probability of a certain error can be estimated using the temporal temperature error model. In this example, the probability of a temperature error equal to $(-5.3℃ + 0.6℃) = -4.7℃$, between time 1 and 2ms is 0.015. Without a measurement at 2ms, the only available information is that the probability of a temperature error equal to $(-4.7℃ + 0.6℃) = -3.1℃$ is 0.015×0.015, between 2 and 3ms. In Fig. 4, a new measurement is done at time 3 and the actual error is -4.7℃.

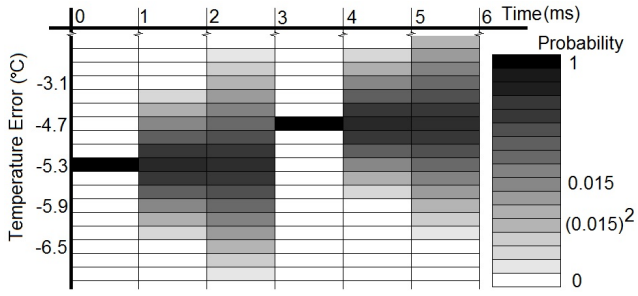

Figure 4. Example for temporal temperature error probabilities.

The spatial temperature error resolution is categorized into inter-chip, inter-core, and intra-core resolutions. The temporal resolution (*temporal error period*) is a multiple of test cycles. In this paper, the spatial resolution is inter-core and each core has a temperature sensor (However, it is possible to have inter-core resolution with only some of the cores having a sensor.). The sensor reading moments are multiples of *temporal error periods*. The overheads associated to sensors and branching tables are taken into account for scheduling.

In order to reduce the size of the temperature error data set, which originally is equal to $M \times C$, in accordance with the accuracy and optimization speed requirements, a simplification scheme is employed to cluster the errors into error clusters. The error clusters are characterized by temperature Error cluster Borders (EB). The temperature error range, resolution, and error clustering are assumed to be identical for all cores. The Temperature Error Values (TEV) and the Spatial Temperature Error Probabilities (STEP) are given for M points, as shown below.

$$TEV = \{EV(m);\ m = 0, 1, \dots, M-1\} \quad (5)$$

$$STEP \triangleq \{STEP(c,m);\ c = 0, 1 \dots, C-1;\ m = 0, 1, \dots, M-1\} \quad (6)$$

The Temporal Temperature Error Probability (TTEP) gives the probability of a certain Temperature Error Value Change (TEVC), as shown below.

$$TTEP = \{TTEP(TEVC)\} \quad (7)$$

The error clustering is uniform and *error borders* are identical for all cores and all branching tables (schedule tree nodes). Assuming L error clusters, the original data set size reduces to $L \times C$. The *error border* (EB) is expressed as

$$EB = \{EB(l); l = 0, 1, \dots, L\}. \quad (8)$$

The error clustering will divide the C-dimensional error space into *error cluster cells* indexed using Cartesian system, i.e. $(l_0, l_1, \dots, l_{C-1})$. For example assume that in a 2-core SoC, each core has 2 error clusters. The 2-dimensional error space is divided into 4 *error cluster cells*, indexed with (0, 0), (0, 1), (1, 0), and (1, 1). While the original size of error space is M^C, the number of *error cluster cells* is L^C. Assuming $M = 400$ and $C = 2$, the original size is $400^2 = 160000$ while the size of the clustered error space, with $L = 3$, is $3^2 = 9$.

VI. Linear Schedule Tables

A linear schedule table, as discussed in section II, captures a schedule without branching (offline). The linear schedule table entries (time) should be optimized in the offline-phase. In order

to simplify the search space, the possible times are assumed to be multiples of a constant, denoted by *linear scheduling period*. The *states* in the linear schedule tables are generated using the heuristic proposed in [10]. According to this heuristic, the test of the cores with lower temperature and higher remaining test size is started or resumed earlier. Activating the cores with lower temperature provides longer testing states and this way it reduces the number of test partitions, which is desirable (less overhead). Also by choosing the colder cores and knowing that the thermal simulator has taken the effect of adjacent cores into account, in fact the algorithm activates the cores which are far from the currently active cores. This will save the newly activated cores from the accumulated heat in their neighbors. It is still helpful in another way; by not activating the adjacent cores, the newly deactivated cores will experience a faster cooling. The heuristic gives advantage to the cores with longer remaining tests, thus maximizing the interleaving opportunities.

The *temperature error value* which is used to estimate the temperature is called *representative temperature error*. The estimated temperature is updated periodically with *linear scheduling period* by correcting the cores' simulated temperatures with the *representative temperature error* value. The estimated temperature is then used to compute the static power and to determine the "state" of the cores for the next *linear scheduling period*. The *representative temperature error* is updated periodically with *temporal error period* while the estimated temperature, static power, and state of the cores are updated periodically with *linear scheduling period*. After updating the state of cores, the dynamic cycle-accurate power sequence for the next *linear scheduling period* is computed. The initial temperatures are available as the results of the previous *linear scheduling period*'s thermal simulation. Having dynamic and static power sequences in addition to the initial temperatures, the next *linear scheduling period*'s thermal simulation is performed.

The optimization problem for a linear schedule table is to minimize the partial NCF by finding the proper *representative temperature error*. The utilized test bus width is the sum of the Test Access Mechanism (TAM) widths of the active cores. The schedule size is the product of the number of the linear schedule table entries and the record size. The schedule tree is equivalent to a number of linear schedule tables (edges) in addition to a number of branching tables (nodes), as shown in Fig. 5(a).

VII. Adaptive Test Scheduling

The adaptive method is based on the on-chip temperature sensors, implemented on each core. During test, the actual temperatures (of carefully selected cores) are read (at carefully selected moments) and the gap between sensor readouts is filled with thermal simulation. A group of chips with similar thermal behavior which are tested with a certain schedule is called a *chip cluster*. During the test, chips are dynamically classified into one of the *chip clusters* and are tested using the corresponding schedule. The *chip clusters* vary during the test, and at each adaptation moment the *chip clusters* change into a new scheme which is optimum for the new situation.

The parameters that affect the efficiency of the adaptive method are the moments when branching/adaptation happens, the number of branches (linear schedule tables), and the branching condition (*chip clustering*). For the example in Fig. 2, the adaptation is happening at t4, the number of branches is 2 (two linear schedule tables), and the branching condition is a comparison with the *Threshold*. As mentioned before, the branching moments are selected out of possible adaptation points (set of all multiples of *temporal error period*). Therefore, the first design parameter simplifies to whether branch or not at a possible adaptation point. The first design parameter, whether to branch or not, is then merged with the second design parameter, the number of branches (*chip clusters*). The third design parameter is the *chip clustering* for each adaptation point. The problem is summarized into the two following sub-problems.

1. How many *chip clusters* (branches or linear schedule table) at each possible adaptation point (node) are needed? One branch means no branching, no sensor reading, and no extra effort.

2. What is the proper *chip clustering* into the given number of *chip clusters*? The number of *chip clusters* is known from question 1. Depending on the *chip clustering* some cores may do not need sensor readout.

When the answer to question 1 is one, question 2 is skipped. These two questions are then formulated into two different forms: the first question is described as a tree topology and the second question is to find the optimum *chip clustering* for the nodes of the specific tree topology.

A candidate schedule tree is generated by putting a possible tree topology together with a possible corresponding *clustering*. The number of candidate tree topologies and the number of candidate *chip clustering* alternatives grows very fast with a small increase in the resolutions of the parameters, e.g. higher temporal error resolution. The complexity growth is also fast with increase in the number of cores. Since the number of candidate trees is the product of the tree topology alternatives and the *chip clustering* alternatives, the search space is unacceptably large. In order to reduce the search space, a constructive method is used.

The schedule tree is constructed by adding small partial trees to its leaves. These small partial trees which are the building blocks of the schedule tree are called sub-trees. A sub-tree consists of a small number of linear schedule and branching tables which makes it possible to be clustered and optimized (scheduled) at once. For example, assume that there is a reproducing tree, *Tree 1*, as shown in Fig. 5(a). The linear

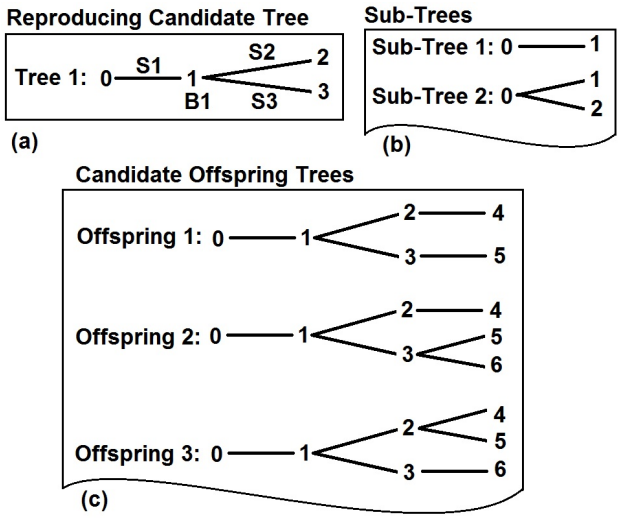

Figure 5. Reproducing tree, sub trees, and candidate offspring trees. For S1, S2, S3, and B1 in (a) refer to Fig. 2.

schedule tables of Fig. 2 correspond to the edges of *Tree 1* while the branching table corresponds to node 1, as shown in Fig. 5(a). Two sub-trees with 1 and with 2 edges are shown in Fig. 5(b). *Tree 1* has two leaves, which combinations of sub-trees are added to them in order to generate the offspring as shown in Fig. 5(c). *Offspring 2* is generated by attaching the *Sub-tree 1* to node 2 of *Tree 1* and attaching the *Sub-tree 2* to node 3 of *Tree 1*. The sub-tree scheduling is explained in section A. The construction approach (how to construct the schedule tree out of the sub-trees) is presented in section B.

A. Sub-tree Scheduling

To find the near optimal sub-tree, the partial cost function must be evaluated for different sub-tree *clustering* alternatives. When the schedule is a tree, the expected values of *test application time* (TAT), *applied test size* (ATS), and *test overheating probability* (TOP) which are denoted by ETAT, EATS, and ETOP should be used in the NCF, Eq. (4), computation in order to utilize the temperature error statistics. The expected values are computed as each edge is scheduled. An edge is represented by its destination node. Assuming that the number of nodes is N, the Nodes' Probabilities (NP), the Nodes' Applied Test Sizes (NATS), and the Nodes' Test Application Times (NTAT) are expressed as follows.

$$NP = \{NP(n); n = 0, 1, \dots, N-1\} \tag{9}$$

$$NATS = \{NATS(n)\} \tag{10}$$

$$NTAT = \{NTAT(n)\} \tag{11}$$

The corresponding expected values are computed as follows.

$$EATS = \sum_n NATS(n) \cdot NP(\mathrm{n}) \tag{12}$$

$$ETAT = \sum_n NTAT(n) \cdot \mathrm{NP(n)} \tag{13}$$

$$ETOP = \sum_n NOP(n) \cdot NP(n) \Big/ \sum_n NP(n) \tag{14}$$

While the *expected applied test size* (EATS) and *expected test application time* (ETAT) are accumulated, the *expected test overheating probability* (ETOP) is averaged since it is a probability. The Nodes' Overheating Probabilities (NOP) will be explained later.

The *chip clustering* at each node is done in a C-dimensional space (C is the number of cores). Each *chip cluster* consists of a certain combination of error values for cores. A candidate sub-tree *clustering* is a set of node *clustering* alternatives. For each candidate sub-tree topology there are a number of candidate *clustering* alternatives, which labels the nodes' *error cluster cells* with their corresponding *chip clusters*. Each *chip cluster* for a node corresponds to an edge branching out of that node (equivalent to a linear schedule table). Each node has its dedicated Error Cluster Cell Labeling (ECCL) as follows.

$$ECCL = \{ECCL(n, (l_0, l_1, \dots, l_{C-1})); n = 0, 1, \dots, N-1\} \tag{15}$$

In order to indicate which of the parent's clusters will lead to a certain node, Node's Cluster Label (NCL) is defined as

$$NCL = \{NCL(n); n = 1, \dots, N-1\}. \tag{16}$$

Having the *error cluster cell* labeling corresponding to an edge, the edge is scheduled (optimum *representative temperature error* is found and the linear schedule table is determined). The candidate sub-tree *clustering* is evaluated based on the optimized linear schedule tables and optimized branching conditions. The candidate *clustering* alternatives are explored to find the optimum *clustering*. The *error cluster cell* probabilities of nodes and the nodes probabilities are computed based on the temperature error model and based on the *clustering* of the ancestor nodes. In order to ease the computation of the Error Cluster Cells Probabilities (ECCP) for different nodes, initially, the Error Cluster Cell Change Probabilities (ECCCP) are computed as shown below. In order to ease the representation, the equations are further broken down by introducing intermediate variables *Ia* and *Ib*. The origin error cluster cells are indexed with *o* and the destination error cluster cells are indexed with *d*.

$$Ia\Big(\big(l_{0,\mathrm{o}}, l_{1,\mathrm{o}}, \dots, l_{C-1,\mathrm{o}}\big), \big(l_{0,\mathrm{d}}, l_{1,\mathrm{d}}, \dots, l_{C-1,\mathrm{d}}\big)\Big) = \prod_{\mathrm{c}} \sum_{mo} \sum_{md} TTEP\big(TEV(md) - TEV(mo)\big) \tag{17}$$

where the range of mo is $EB\big(l_{c,o} - 1\big) < mo \le EB\big(l_{c,o}\big)$ *and* the range of md is $EB\big(l_{c,d} - 1\big) < md \le EB\big(l_{c,d}\big)$

$$ECCCP\Big(\big(l_{0,\mathrm{o}}, l_{1,\mathrm{o}}, \dots, l_{C-1,\mathrm{o}}\big), \big(l_{0,\mathrm{d}}, l_{1,\mathrm{d}}, \dots, l_{C-1,\mathrm{d}}\big)\Big) = \frac{Ia\Big(\big(l_{0,\mathrm{o}}, l_{1,\mathrm{o}}, \dots, l_{C-1,\mathrm{o}}\big), \big(l_{0,\mathrm{d}}, l_{1,\mathrm{d}}, \dots, l_{C-1,\mathrm{d}}\big)\Big)}{\sum_{\left(l_{0,\mathrm{p}}, l_{1,\mathrm{p}}, \dots, l_{C-1,\mathrm{p}}\right)} Ia\Big(\big(l_{0,\mathrm{p}}, l_{1,\mathrm{p}}, \dots, l_{C-1,\mathrm{p}}\big), \big(l_{0,\mathrm{d}}, l_{1,\mathrm{d}}, \dots, l_{C-1,\mathrm{d}}\big)\Big)} \tag{18}$$

For the root node (n=0), just after the warm up:

$$ECCP(n = 0, (l_0, l_1, \dots, l_{C-1})) = \prod_{\mathrm{c}} \sum_{m} STEP(c, m) \tag{19}$$

where the range of m is $EB(l_c - 1) < m \le EB(l_c)$

For other non-root nodes, the child's node index is cn ($cn > 0$) and the parent's node index is pn:

$$H(cn, pn, (l_0, l_1, \dots, l_{C-1})) = \begin{cases} ECCP(pn, (l_{0,\mathrm{o}}, l_{1,\mathrm{o}}, \dots, l_{C-1,\mathrm{o}})) & ; if \\ \qquad ECCL\big(pn, (l_0, l_1, \dots, l_{C-1})\big) == NCL(cn) & \\ 0 & ; Otherwise \end{cases} \tag{20}$$

$$Ib\Big(cn, \big(l_{0,\mathrm{d}}, l_{1,\mathrm{d}}, \dots, l_{C-1,\mathrm{d}}\big)\Big) = \sum_{\left(l_{0,\mathrm{o}}, l_{1,\mathrm{o}}, \dots, l_{C-1,\mathrm{o}}\right)} H\Big(cn, pn, \big(l_{0,\mathrm{o}}, l_{1,\mathrm{o}}, \dots, l_{C-1,\mathrm{o}}\big)\Big) \times ECCCP\Big(\big(l_{0,\mathrm{o}}, l_{1,\mathrm{o}}, \dots, l_{C-1,\mathrm{o}}\big), \big(l_{0,\mathrm{d}}, l_{1,\mathrm{d}}, \dots, l_{C-1,\mathrm{d}}\big)\Big) \tag{21}$$

$$ECCP\Big(cn, \big(l_{0,\mathrm{d}}, l_{1,\mathrm{d}}, \dots, l_{C-1,\mathrm{d}}\big)\Big) = \frac{Ib\Big(cn, \big(l_{0,\mathrm{d}}, l_{1,\mathrm{d}}, \dots, l_{C-1,\mathrm{d}}\big)\Big) \times \sum_{\left(l_{0,\mathrm{o}}, l_{1,\mathrm{o}}, \dots, l_{C-1,\mathrm{o}}\right)} H\Big(cn, pn, \big(l_{0,\mathrm{o}}, l_{1,\mathrm{o}}, \dots, l_{C-1,\mathrm{o}}\big)\Big)}{\sum_{\left(l_{0,\mathrm{d}}, l_{1,\mathrm{d}}, \dots, l_{C-1,\mathrm{d}}\right)} Ib\Big(cn, \big(l_{0,\mathrm{d}}, l_{1,\mathrm{d}}, \dots, l_{C-1,\mathrm{d}}\big)\Big)} \tag{22}$$

TTEP is temporal temperature error probability as introduced in (7), TEV is temperature error value as introduced in (5), EB is error border as introduced in (8), STEP is spatial temperature error probability as introduced in (6), H is an intermediate variable, and NCL is node cluster label as introduced in (16).

The applied test size of an edge is the sum of the difference between the test sequence pointers in the destination node and origin node, scaled by the TAM width of that core. The test application time of an edge is either equal to the difference between the test cycles of the destination and origin nodes (when

the test is still running to the destination node) or is smaller (when the test is over just before the destination node). Overheating is counted, for each *error cluster cell*, when the Simulated Temperature (ST) plus Temperature Error Cluster Cell Center (TECCC), exceeds the High Temperature Threshold (HTT). Assuming that the edge duration is τ cycles, the *node overheating probability* (NOP) is computed as follows.

$$TECCC(l) = (EB(l) + EB(l+1))/2\,;\, l = 0, 1, \dots, L-1 \tag{23}$$

$$NOP(n) = \frac{\sum_{(l_0,l_1,\dots,l_{C-1}),t} S(n,(l_0,l_1,\dots,l_{C-1}),t)}{\tau \times \sum_{(l_0,l_1,\dots,l_{C-1})} ECCP\big(n,(l_0,l_1,\dots,l_{C-1})\big)} \tag{24}$$

$$\mathrm{S}(n,(l_0,l_1,\dots,l_{C-1}),t) = \begin{cases} ECCP\big(n,(l_0,l_1,\dots,l_{C-1})\big) & ;if \\ & \exists c \mid \big(ST(n,c,t) + TECCC(l_c)\big) > HTT \\ 0 & ;Otherwise \end{cases} \tag{25}$$

EB is error border as introduced in (8), S is an intermediate variable, and ECCP is error cluster cell probability as introduced in (19) and (22). The *clustering* alternatives are explored by a heuristic, and finally the best is selected.

B. Tree Construction

The construction starts with a root node and in each iteration the reproducing candidate tree extends and multiplies by adding possible combinations of sub-trees to its active leaf nodes, as shown in Fig. 5. Then, the candidate offspring are pruned and a small number of promising under-construction trees are selected as reproducing candidates for the next iteration. For example, a reproducing list (similar to Fig. 5(a)) will be obtained by pruning the offspring list (partially shown in Fig. 5(c)). The algorithm ends when all the reproducing trees have completed the test.

The pruning process must guarantee the ATE memory constraint while providing the freedom to put more clusters in the more beneficial regions. To provide such a freedom, the pruning is done based on the Scaled Cost Function (SCF), as defined in the following.

$$SCF = NCF \cdot (adjusting\_offset + number\_of\_nodes) \tag{26}$$

The *normalized cost function* (NCF) is scaled by the tree's number of nodes plus *adjusting offset*. Now, adding nodes to the tree is only beneficial if it gives a reasonable cost reduction otherwise a smaller tree may get a lower *scaled cost function* and manage to survive to the next iteration, while bigger trees are discarded. The effect of the number of nodes is adjusted by *adjusting offset*. A small *adjusting offset* promotes having fewer branches compared to a large *adjusting offset* which promotes having more branches.

To guarantee the memory constraint, when a reproducing candidate is selected based on its *scaled cost function*, it is scheduled for the rest of test by just using the linear schedule tables (no further branching). During this scheduling, the linear scheduling aborts as soon as the memory limit is violated. If the linear scheduling succeeded in respecting the memory limit, the iterations continue. Otherwise, the currently chosen reproducing candidate is dropped and the next candidate (with larger or equal *scaled cost function*) is tested for its compliance with memory limit. The scheduling will fail if no candidate could meet the memory limit, meaning that the limit is too tight even for an offline method.

The number of the sub-tree topologies is controlled with the sub-tree length and the maximum allowed number of branches per node. Increase in the sub-tree length will improve the global optimality and increase in the allowed number of branches per node improves the *chip clustering* resolution, but both will increase the CPU time.

VIII. Experimental Results

In our experiments, the temperature simulation is done using HotSpot [15]. The test sequences for cores are from [16]. The dynamic power sequences for different TAM widths are computed using a cycle-accurate method introduced in [17]. The wrapper chain configurations for cores are determined by the "Design_Wrapper" algorithm [18]. The static power is computed using the method given in [19].

The spatial temperature error and temporal temperature error are assumed to be normal distributions $\aleph(0, 36)$ with range $-20℃$ to $+20℃$ and a resolution equal to $0.1℃$. The balancing coefficient (BC in Eq. 4) is 10. Experiments are performed with three SoC examples, SoC1, SoC2, and SoC3, with 2, 3, and 4 cores, respectively. The cores are instances of ITC'02 benchmark cores [20]. The memory size is a normalized value which must be scaled according to the implementation specification to give the size in Bytes.

The traditional and the proposed methods are compared, and the comparison results for SoC1 are reported in Table I. Column 2 refers to normalized cost function which is defined in section IV. Column 3 refers to the ATE memory size which is required to store the schedule (the test sequences are stored elsewhere and are not affected by the schedule). The corresponding percentages of these values relative to the Offline method are given in columns 4 and 5, respectively. The traditional methods include the Offline method (only one linear schedule is used) and the Hybrid method (similar to [11]). Our proposed adaptive method has reduced the cost to 80 percent relative to the offline method, while the cost achieved by the hybrid method is 87 percent. This difference demonstrates the advantage of the proposed adaptive method. The cost reduction comes with increase in the schedule size because of increased number of linear schedule and branching tables, which consume ATE memory space.

The reduction of the cost with the increase of the memory limit is shown in Table II. The memory limit is increased with small steps. It is expected that the increase in the memory limit improves the cost before it reaches the saturation limit. The saturation limit for this set of experiments is equal to 1320. The CPU time increases in general with the increase of memory limit. This trend continues even if the cost is not improved (after

TABLE I. Comparison of Traditional and Proposed Method

Methods	**Results**			
	NCF	***Size***	***NCF***	***Size***
Offline	3.3875	460	***Relative to the Offline***	
Hybrid	2.9389	920	86.76%	200.00%
Adaptive	2.7170	1320	80.21%	286.96%

TABLE II. Experiments with Different Memory Limits

Memory Limit	**Results**				
	NCF	***Size***	***NCF***	***size***	***CPU Time (H:M:S)***
			Relative to Limit = 500		
300	Aborted, Mem. Limit is too tight				1:03:42
500	3.3875	460	100.00%	100.00%	3:15:21
750	3.3875	460	100.00%	100.00%	3:34:20
1000	2.9389	920	86.76%	200.00%	3:41:03
1250	2.9389	920	86.76%	200.00%	3:48:47
1500	2.7170	1320	80.21%	286.96%	3:53:52
1750	2.7170	1320	80.21%	286.96%	3:59:12
2000	2.7170	1320	80.21%	286.96%	4:04:16

the saturation) since the algorithm has more space to search and thus it takes more time.

The scheduling results for the different SoCs are given in Table III. The effect of the increase in the number of cores depends on the interaction of the new thermal behavior, and the new available test bus widths and memory. As a result of lower ratio of the available to requested test bus width and memory, and as a result of slightly higher working temperature, the cost (NCF) may increase with an increase in the number of cores. Note that the growth of the CPU time with the number of cores is due to very slow HotSpot thermal simulation and could be reduced by using a faster simulator.

TABLE III. EXPERIMENTS WITH DIFFERENT SOCS

Settings		Results		
SoC	***Memory Limit***	***NCF***	***Size***	***CPU Time (H:M:S)***
SoC1	1500	2.7170	1320	4:04:16
SoC2	2250	5.8597	1950	5:37:12
SoC3	3000	6.2336	2880	10:26:23

NOTATIONS

Notation	Description	Equation
ATS	Applied Test Size	
ATTPS	Affordable Test Time Per Second	
BC	Balancing Coefficient	
C	Number of cores	
CTF	Cost of the Test Facility per time unit	
EATS	Expected Applied Test Size	(12)
EB	Temperature Error cluster Borders	(8)
ECCCP	Error Cluster Cell Change Probabilities	(18)
ECCL	Error Cluster Cell Labeling	(15)
ECCP	Error Cluster Cells Probabilities	(19), (22)
ETAT	Expected Test Application Time	(13)
ETOP	Expected Test Overheating Probability	(14)
HTT	High Temperature Threshold	
L	Number of temperature error clusters	
M	Size of the temperature error (value/probabilities) data set	
N	Number of nodes in a tree/sub-tree	
NATS	Nodes' Applied Test Sizes	(10)
NCF	Normalized Cost Function	(4)
NCL	Node's Cluster Label	(16)
NOP	Nodes' Overheating Probabilities	(24)
NP	Nodes' Probabilities	(9)
NTAT	Nodes' Test Application Times	(11)
NTT	Normalized Test Throughput	(3)
POC	Price of One Chip	
SCF	Scaled Cost Function (to select reproducing candidate trees)	(26)
ST	Simulated Temperature	
STEP	Spatial Temperature Error Probabilities	(6)
TAM	Test Access Mechanism	
TAT	Test Application Time	
TECCC	Temperature Error Cluster Cell Center	(23)
TEV	Temperature Error Values	(5)
TEVC	Temperature Error Value Change	
THT	Test Handling Time	
TOP	Test Overheating Probability	
TT	Test Throughput	(2)
TTEP	Temporal Temperature Error Probability	(7)
τ	Edge duration cycles	

IX. CONCLUSION AND REMARKS

This paper presents an adaptive SoC test scheduling technique to deal with spatial and temporal temperature deviations, caused by process variations in deep submicron technologies. The main contribution is an algorithm to generate a set of efficient test schedules, each corresponding to a different temperature behavior of different cores during test. During the test, on-chip temperature sensors are used to monitor the actual temperatures of the different cores and to guide the selection of the corresponding test schedules accordingly. In this way, the overall test efficiency will be maximized. The adaptive test schedule generation algorithm has been applied to several SoC designs and the results demonstrate that the generated test schedules are much more efficient than the traditional off-line and hybrid methods.

The proposed technique uses two important and standalone software modules, the *thermal-aware test scheduler* and the *thermal simulator*. The *thermal-aware test scheduler* must keep the test sequences intact and the *thermal simulator* should be fast. However the *thermal simulator* and sensor errors are not addressed explicitly here, in practice they are partially handled using temperature error model. There still might be small residual errors which are not captured by temperature error model. These small residual errors are addressed by introducing a small safety margin on top of the schedule. The effect of this small safety margin on cost is negligible as shown in [11], but it will cover the residual errors.

REFERENCES

[1] W. Nebel and J. Mermet (Editors). Low power design in deep submicron electronics. Kluwer, 1997.

[2] K.-T. Cheng, S. Dey, M. Rodgers, and K. Roy. "Test challenges for deep sub-micron technologies." pp. 142–149, DAC 2000.

[3] Y. Bonhomme, P. Girard, C. Landrault, S. Pravossoudovitch. "Test power: a big issue in large SOC designs." pp. 447-449, DELTA 2002.

[4] Y. A. Zorian. "Distributed BIST control scheme for complex VLSI devices." pp. 4-9, VTS 1993.

[5] T. E. Yu, T. Yoneda, K. Chakrabarty, and H. Fujiwara. "Test infrastructure design for core-based system-on-chip under cycle-accurate thermal constraints." pp. 793-798, ASP-DAC 2009.

[6] U. Chandran and D. Zhao. "Thermal driven test access routing in hyper-interconnected three-dimensional System-on-Chip." pp. 410 – 418, IEEE DFT 2009.

[7] Z. He, Z. Peng, P. Eles, P. Rosinger and B. M. Al-Hashimi. "Thermal-aware SoC test scheduling with test set partitioning and interleaving." J. Electronic Testing, pp. 247-257, Vol. 24, No.1-3, 2008.

[8] Z. He, Z. Peng, P. Eles. "Thermal-aware test scheduling for core-based SoC in an Abort-on-First-Fail test environment." Euromicro DSD 2009.

[9] C. Yao, K. K. Saluja, P. Ramanathan. "Partition based SoC test scheduling with thermal and power constraints under deep submicron technologies." pp. 281-286, ATS 2009.

[10] Z. He, Z. Peng, P. Eles. "Simulation-driven thermal-safe test time minimization for System-on-Chip." pp. 283-288, ATS 2008.

[11] N. Aghaee, Z. He, Z. Peng, and P. Eles. "Temperature-aware SoC test scheduling considering inter-chip process variation." ATS 2010.

[12] J. Clabes et al. "Design and implementation of the POWER5 microprocessor." DAC 2004.

[13] IEEE P1687 (IJTAG) http://grouper.ieee.org/groups/1687/.

[14] C. Yao, K. K. Saluja, P. Ramanathan. "Thermal-aware test scheduling using on-chip temperature sensors." VLSI Design 2011.

[15] W. Huang, M. R. Stan, K. Skadron, K. Sankaranarayanan, S. Ghosh, and S. Velusamy. "Compact thermal modeling for temperature-aware design." DAC 2004.

[16] K. Miyase and S. Kajihara. "XID: don't care identification of test patterns for combinational circuits." IEEE Trans. CAD, vol. 23, no. 2, pp. 321-326, 2004.

[17] S. Samii, M. Selkala, E. Larsson, K. Chakrabarty, Z. Peng. "Cycle-accurate test power modeling and its application to SoC test architecture design and scheduling." IEEE Trans. CAD, Vol. 27, No. 5, pp. 973-977, 2008.

[18] V. Iyengar, K. Chakrabarty, and E. J. Marinissen. "Test wrapper and test access mechanism co-optimization for System-on-Chip." ITC 2001.

[19] W. P. Liao, L. He, K. M. Lepak. "Temperature and supply voltage aware performance and power modeling at microarchitecture level." IEEE Trans. CAD, Vol.24, No.7, pp. 1042- 1053, 2005.

[20] E. J. Marinissen, V. Iyengar, K. Chakrabarty. "A Set of Benchmarks for Modular Testing of SOCs." ITC 2002.