A Heuristic for Thermal-Safe SoC Test Scheduling

Zhiyuan He, Zebo Peng, and Petru Eles
Embedded Systems Laboratory (ESLAB)
Linköping University, Sweden
{zhihe, zebpe, petel}@ida.liu.se

Abstract [1]

High temperature has become a technological barrier to the testing of high performance systems-on-chip, especially when deep submicron technologies are employed. In order to reduce test time while keeping the temperature of the cores under test within a safe range, thermal-aware test scheduling techniques are required. In this paper, we address the test time minimization problem as how to generate the shortest test schedule such that the temperature limits of individual cores and the limit on the test-bus bandwidth are satisfied. In order to avoid overheating during the test, we partition test sets into shorter test sub-sequences and add cooling periods in between, such that continuously applying a test sub-sequence will not drive the core temperature going beyond the limit. Further more, based on the test partitioning scheme, we interleave the test sub-sequences from different test sets in such a manner that a cooling period reserved for one core is utilized for the test transportation and application of another core. We have proposed a heuristic to minimize the test application time by exploring alternative test partitioning and interleaving schemes with variable length of test sub-sequences and cooling periods. Experimental results have shown the efficiency of the proposed heuristic.

1. Introduction and related work

Production of integrated circuits has moved into the deep submicron technology regime. Scaling of process technology has enabled dramatically increasing the number of transistors, and therefore improving the performance of electronic chips. However, the rapid growth of integration density has posed critical challenges to the design and test of electronic systems, one of which is the power and thermal issue [1], [2], [3], [4]. High temperature can be observed in most high-performance chips due to high power density and high heat dissipation. Overheating can decrease the carrier mobility of electrons and therefore reduces the driving current of CMOS transistors, which consequently degrades the circuit performance. The reliability and lifespan of integrated circuits also decrease at high temperatures. Advanced cooling solutions can partly solve the high temperature problem, however they increase the system cost substantially. The thermal issue becomes even more severe in the case of electronic system testing than in normal functional mode, since testing dissipates more power and heat due to a substantial increase of switching activities [5], [6].

In recent years, a core based system-on-chip (SoC) design methodology has been employed to reduce design complexity by integrating pre-designed and pre-verified intellectual property cores. Although the cost of designing and manufacturing SoCs has been reduced, the testing cost rises because larger quantities of test data are required and longer test times are usually needed. In order to reduce the testing cost, research efforts have been devoted to developing advanced test architectures and approaches to test resource allocation and test scheduling [7], [8], [9], [10], [11], [12], [13], [14], [15]. However, in order to solve the problem related to the testing of new generations of SoCs, novel temperature aware techniques have to be developed.

Recently, thermal-aware testing has attracted many research interests. Liu et al. proposed a technique to evenly distribute the generated heat across the chip during tests, and therefore avoid high temperature [16]. Rosinger et al. proposed an approach to generate thermal-safe test schedules with minimized test time by utilizing the core adjacency information to drive the test scheduling and reduce the temperature stress between cores [17]. In our previous work [18], we proposed a test set partitioning and interleaving technique, and employed constraint logic programming (CLP) to generate thermal-aware test schedules with the minimum test application time (TAT).

In this paper, we assume that, for SoCs composed of cores with moderately large sizes, a continuous test will raise the temperature of a core towards a limit beyond which the core may be damaged. In order to avoid overheating during tests, we partition the entire test set into a number of test sub-sequences and introduce a cooling period between two consecutive test sub-sequences. As the test application time substantially increases when long cooling periods are introduced, we interleaved different partitioned test sets in order to generate a shorter test schedule. An alternative solution to the thermal-related problems during test is to reduce the clock frequency, but it cannot be applied to some types of tests, such as at-speed test, where the clock frequency should not be reduced.

In [18], we restricted the length of test sub-sequences that belong to the same test set to be identical. Moreover, we also restricted the cooling periods between test sub-

[1] This work has been partially supported by the Swedish Foundation for Strategic Research (SSF) under the Strategic Integrated Electronic Systems Research (STRINGENT) program.

sequences from the same test set to have equal length. The main purpose of these restrictions was to keep the size of the design space small and, by this, to reduce the optimization time, so that the CLP-based algorithm will be able to generate the optimal solutions in a reasonable time. However, this restriction has resulted in less efficient test schedules, and, by that, longer test application times. In this paper, we have eliminated this restriction so that both test sub-sequences and cooling periods can have arbitrary lengths. Since breaking the regularity of test sub-sequences and cooling periods dramatically increases the size of exploration space, the CLP-based test scheduling approach proposed in [18] is not feasible any more, especially for practical industrial designs. Therefore, new, low-complexity heuristics are needed which are able to produce efficient test schedules under the less restricted and more realistic assumptions of this paper.

The rest of this paper is organized as follows. The next section presents the assumed basic test architecture. In Section 3, a motivational example is given to illustrate the thermal-safe test scheduling problem. Section 4 gives the problem formulation, and Section 5 demonstrates the overall solution strategy to solve the formulated problem. The proposed heuristic is illustrated in Section 6, and experimental results are presented in Section 7. The paper is concluded in Section 8.

2. Basic test architecture

We have assumed a test architecture using a test bus to transport test data between the tester and the cores under test. A tester can be either an external automatic test equipment (ATE) or an embedded tester integrated on the chip. Each core under test is connected to the test bus with a number of dedicated TAM wires. The test patterns, together with a generated test schedule, are stored in the tester memory. A test controller controls the entire test process according the test schedule, sending test patterns to and receiving test responses from the corresponding cores through the test bus and the TAM wires,.

An example of the assumed test architecture is depicted in Figure 1. In this example, a system of four cores is to be tested. An ATE consisting of a test controller and a local memory serves as an external tester. The generated test patterns and a test schedule are stored in the tester memory. When the test starts, the test patterns are transported to the cores through a test bus. It should be noted that the ATE can be replaced by an embedded tester and the remaining parts of the test architecture are still applicable.

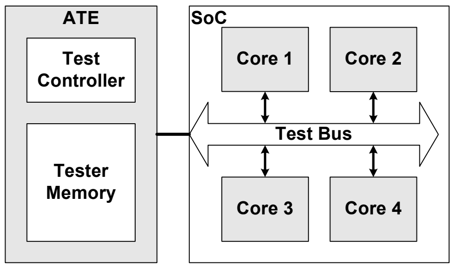


Figure 1. An example of the assumed basic test architecture

3. A motivational example

When a long sequence of test patterns is continuously applied to a core, the temperature of this core may increase towards a certain limit beyond which the core will be damaged. Therefore, a test has to be stopped when the core temperature reaches the limit, and the test can be restarted later when the core has been cooled down. Thus, by partitioning a test set into shorter test sub-sequences and introducing cooling periods between them, we can avoid the overheating during test. Figure 2 illustrates a situation in which the entire test set is partitioned into four test sub-sequences, TS_1, TS_2, TS_3, and TS_4, and cooling periods are introduced between the them. In this way, the temperature of the core remains under the imposed temperature limit.

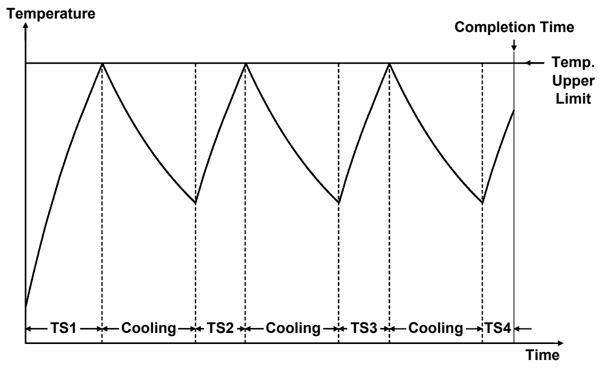


Figure 2. An example of test set partitioning

As we have assumed that a test bus is employed in the test architecture, the limit on the test-bus bandwidth becomes a constraint to the scheduling of the test sub-sequences. It is obvious that introducing long cooling periods between test sub-sequences can substantially increase the test application time. Intuitively, we can reduce the TAT by interleaving the partitioned test sets such that the cooling periods reserved for a core C_i are utilized to transport test data for another core C_j ($j \neq i$), and thereafter to test the core C_j. By interleaving the partitioned test sets belonging to different cores, the test-bus bandwidth is more efficiently utilized. Figure 3 gives an example where two partitioned test sets are interleaved so that the test time is reduced with no need for extra bus bandwidth.

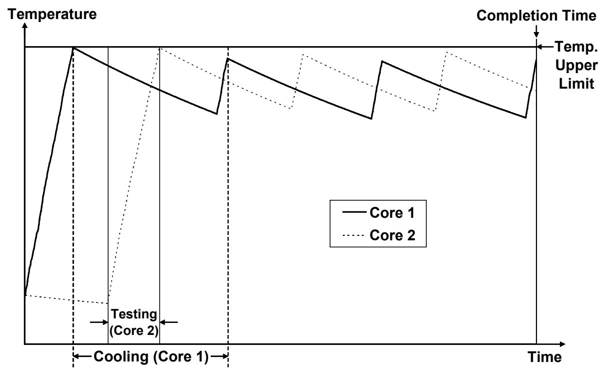
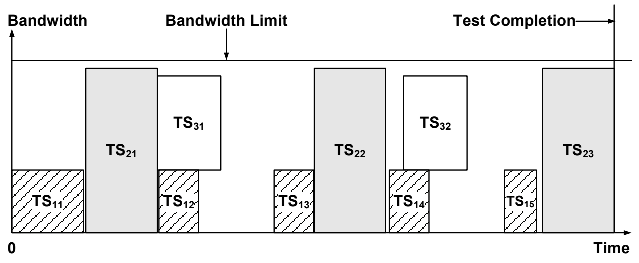


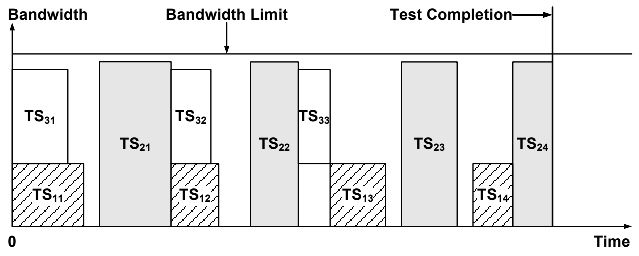
Figure 3. An example of test set interleaving

In this paper, we aim to minimize the test application time by generating an efficient test schedule which avoids violating the temperature limits of individual cores, and, at the same time, satisfies the test-bus bandwidth constraint. Based on the proposed test set partitioning and interleaving technique, we have eliminated the restriction that the lengths of test sub-sequences from the same test set should be identical. We have also eliminated the restriction that the cooling periods between test sub-sequences from the same test set should be identical. Thus the TAT can be further reduced by finding more elaborate test schedules.

In this paper, we consider each test sub-sequence as a rectangle, with its height representing the required test-bus bandwidth and its width representing the test time. Figure 4 gives a motivational example for our test time minimization problem. Suppose that three test sets, TS_1, TS_2, and TS_3, are partitioned into 5, 3, and 2 test sub-sequences, respectively. Note that the partitioning scheme which determines the length of test sub-sequences and cooling periods has ensured that the temperature of each core will not violate the temperature limit, by using a temperature simulation. Figure 4(a) shows a feasible test schedule under the regularity assumption (identical test sub-sequence length an identical cooling periods for each core). In Figure 4(b), an alternative test schedule is depicted, where the test sub-sequence and the cooling periods can have arbitrary lengths. This example shows the possibility to find a shorter test schedule by exploring alternative solutions, where the number and length of test sub-sequences, the length of cooling periods, and the way that the test sub-sequences are interleaved are different from those in Figure 4(a).



(a) A feasible test schedule with regular partitioning scheme



(b) An alternative test schedule with irregular partitioning scheme

Figure 4. A motivational example

4. Problem formulation

Suppose that a system S, consisting of n cores $C_1, C_2, \dots, C_n$, employs the test architecture illustrated in Figure 1. In order to test core C_i, a test set TS_i consisting of l_i generated test patterns is transported through the test bus and the dedicated TAM wires to/from core C_i, utilizing a bus bandwidth W_i. The test bus is designed to allow transporting several test sets in parallel but has a bandwidth limit BL ($BL \geq W_i$, $i = 1, 2, \dots, n$). We assume that continuously applying test patterns belonging to TS_i may cause the temperature of core C_i go beyond a certain limit TL_i so that the core can be damaged. In order to prevent overheating during tests, we allow partitioning a test set into a number of test sub-sequences and introducing a cooling period between two partitioned test sub-sequences, such that no test sub-sequence drives the core temperature higher than the limit and the core temperature is kept within a safe range. The problem that we address in this paper is to generate a test schedule for system S such that the test application time (TAT) is minimized while the bus bandwidth constraint is satisfied and the temperatures of all cores during tests remains below the corresponding temperature limits. The formal problem formulation is given in Figure 5.

Input:
$\{TS_i \mid i = 1, 2, \dots, n\}$,
$\{W_i \mid i = 1, 2, \dots, n\}$,
Test-bus bandwidth limit BL,
Temperature limit of each individual core $\{TL_i \mid i = 1, 2, \dots, n\}$

Output:
Test schedule with the minimized test application time (TAT)

Subject to the following two constraints:
1. At any time moment t before the test completion, the total amount of utilized test-bus bandwidth $BW(t)$ is less than or equal to the bandwidth limit BL, i.e. $BW(t) := \Sigma BW_i(t) \leq BL$.
2. At any time moment t before the test completion, the temperature $TM_i(t)$ of every individual core C_i is less than or equal to the corresponding temperature limit TL_i, i.e. $\forall i$, $TM_i(t) \leq TL_i$.

Figure 5. Problem formulation

5. Overall solution strategy

We have proposed an overall solution strategy to solve the formulated problem in two major steps, as illustrated in Figure 6. In the first step, we generate an initial partitioning scheme for every test set by using temperature simulation and the given temperature limits. In the second step, we employ the proposed test scheduling algorithm to explore alternative test schedules with respect to different partitioning and interleaving schemes for the test sets. The test sub-sequences are squeezed into a two-dimensional plane constrained by the bandwidth limit of the test bus such that the test application time is minimized.

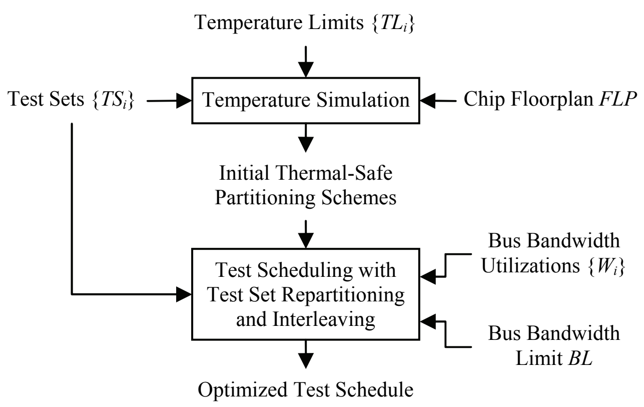

Temperature Limits {TL_i}
Test Sets {TS_i}
Temperature Simulation
Chip Floorplan FLP
Initial Thermal-Safe Partitioning Schemes
Test Scheduling with Test Set Repartitioning and Interleaving
Bus Bandwidth Utilizations {W_i}
Bus Bandwidth Limit BL
Optimized Test Schedule

Figure 6. Illustration of our solution to the formulated problem

In order to generate thermal-safe partitioning schemes, we have used a temperature simulator, HotSpot [4], [22], [23], [24], to simulate instantaneous temperatures of individual cores during tests. HotSpot assumes a circuit packaging configuration widely used in modern IC designs, and it computes a compact thermal model [24] based on the analysis of three major heat flow paths existing in the assumed packaging configuration [23], [24]. Given the floorplan of the chip and the power consumption profiles of the cores, HotSpot calculates the instantaneous temperatures and estimates the steady-state temperatures for each unit. In this paper, we assume that the temperature influences between cores are negligible, since the heat transfer in the vertical direction dominates the transferring of dissipated heat. This has been demonstrated in [18] with the results of temperature simulation using HotSpot.

When generating the initial thermal-safe partitioning scheme, we have assumed that a test set TS_i is started when the core is at the ambient temperature TM_{amb}. Then we start the temperature simulation, and record the time moment t_{h1} when the temperature of core C_i reaches the given temperature limit TL_i. Knowing the latest test pattern that has been applied by the time moment t_{h1}, we can easily obtain the length of the first thermal-safe test sub-sequence TS_{i1} that should be partitioned from the test set TS_i. Then the temperature simulation continues while the test process on core C_i has to be stopped until the temperature goes down to a certain degree. It is obvious that a relatively long time is needed in order to cool down a core to the ambient temperature, as the temperature decreases slowly at a lower temperature level (see the dashed curve in Figure 7). Moreover, from the temperature simulation results, it is observed that the cooling periods are usually much longer than the application times of the test sub-sequences, even if the cooling periods are stopped at the same temperatures that the preceding test sub-sequences are started from. Thus, we let the temperature of core C_i go down only until the slope of the temperature curve reaches a given value k [1], at time moment t_{c1}. At this moment, we have obtained the duration of the first cooling period $d_{i1} = t_{c1} - t_{h1}$. Restarting the test process from time moment t_{c1}, we repeat this heating-and-cooling procedure throughout the temperature simulation until all test patterns belonging to TS_i are applied. Thus we have generated the initial thermal-safe partitioning scheme, where test set TS_i is partitioned into m test sub-sequences $\{TS_{ij} \mid j = 1, 2, \dots, m\}$ and between every two consecutive test sub-sequences, the duration of the cooling period is $\{d_{ij} \mid j = 1, 2, \dots, m\text{-}1\}$, respectively. Figure 7 depicts an example of partitioning a test set into four thermal-safe test sub-sequences with three cooling periods added in between.

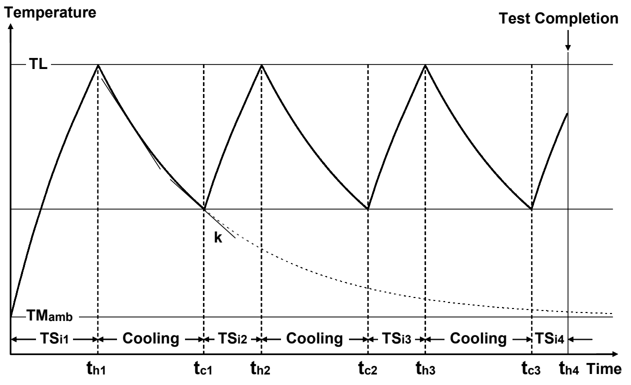


Figure 7. An example of generating initial partitioning scheme

Once the initial thermal-safe partitioning scheme is obtained, the rest of the paper focuses on how to schedule all the test sub-sequences such that the test application time is minimized under the constraint on the test-bus bandwidth. In this paper, since we consider each test sub-sequence as a rectangle, the problem of generating a test schedule with minimized TAT while satisfying the constraint on the test-bus bandwidth can be formulated as a rectangular packing (RP) problem [19], [20], [21]. However, our test scheduling problem is not a classical RP problem, due to the fact that the number of test sub-sequences, the length of the sub-sequences, and the cooling periods are not constant. This makes our problem even more difficult to be solved.

Interleaving test sub-sequences belonging to different test sets can introduce time overheads [25], [15], when the test controller stops one test and switches to another. Therefore, partitioning a test set into more test sub-sequences may lead to a longer test application time, since more time overheads and more cooling periods are introduced into the test schedule. On the other hand, partitioning a test set into more test sub-sequences results in a shorter average length of the individual test sub-sequences, which in principle can be packed in a more compact way and thus lead to shorter test application times. Thus, we need a global optimization algorithm, in which different numbers and lengths of test sub-sequences as well as variant cooling periods are explored. We have proposed a heuristic to generate optimized test schedules by scheduling test sub-sequences with test set repartitioning and interleaving.

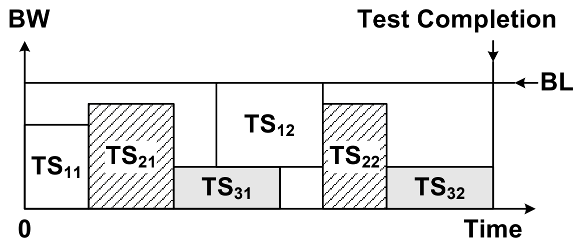
[1] The value of k can be experimentally set by the designers.

6. Heuristic for test scheduling

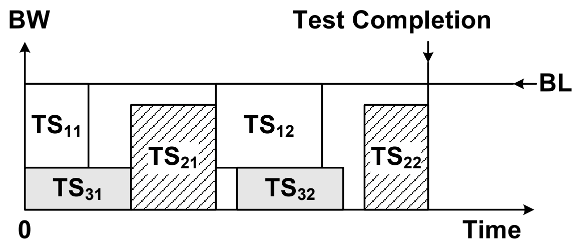
We have proposed a heuristic to do the test scheduling with test set repartitioning and interleaving. Since the order in which the test sets are considered for test scheduling has a large impact on the final test schedule, we construct an iterative algorithm to obtain a good scheduling consideration order (SCO) for all partitioned test sets, and thereafter schedule the test sub-sequences according to the obtained SCO.

Figure 8 shows a simple example illustrating the impact of different scheduling consideration order on the test schedule of three test sets, TS_1, TS_2, and TS_3, each of which is partitioned into two test sub-sequences. Figure 8(a) and Figure 8(b) respectively depicts the test schedule when the test sets are considered for scheduling in the order of $\{TS_1, TS_2, TS_3\}$ and $\{TS_3, TS_2, TS_1\}$. It is obvious that using the second SCO results in a shorter test schedule. Note that in this example the test sets are scheduled to the earliest available time moments.

It should also be noted that the scheduling consideration order refers to the precedence of partitioned test sets to be considered for scheduling. However, when a test set is taken into account for scheduling, we do not schedule all the test sub-sequences of this test set at one time. Instead, we always take the first unscheduled test sub-sequence of the currently considered test set for scheduling, and thereafter take the first unscheduled test sub-sequence of the next test set into account. Thus, in this example, the overall scheduling consideration order (OSCO) for all test sub-sequences of all test sets is $\{TS_{11}, TS_{21}, TS_{31}, TS_{12}, TS_{22}, TS_{32}\}$ and $\{TS_{31}, TS_{21}, TS_{11}, TS_{32}, TS_{22}, TS_{12}\}$, for the case of Figure 8(a) and Figure 8(b) respectively. The main concern of not scheduling all test sub-sequences of one test set at one time is to avoid generating low efficient test schedule due to unnecessarily long cooling periods, inappropriate partition length, and inefficient test-set interleaving.



(a) Test schedule with the SCO $\{TS_1, TS_2, TS_3\}$



(b) Test schedule with the SCO $\{TS_3, TS_2, TS_1\}$

Figure 8. Illustration of how SCO affects test schedule length

The basic idea of the proposed heuristic is to iteratively construct a queue that finally consists of all partitioned test sets in a particular order. The pseudo-code of the proposed heuristic is depicted in Figure 9, denoted with ALG. 1. Note that, inside the heuristic, a scheduling algorithm (denoted with ALG. 2) is invoked, and its pseudo-code is given in Figure 10.

```
ALG. 1. HEURISTIC for test scheduling
01 Set of test sets :: $U := \{TS_i \mid i = 1, 2, \dots, n\}$;
02 Queue of test sets :: $Q := \emptyset$;
03 Queue of test sets :: $Q_{best} := \emptyset$;
04 for ($\forall TS \in U$) loop   /* outer loop */
05    $\eta_{max} := 0$;
06    $Q := Q_{best}$;
07    for ($\forall POS$ in $Q$) loop   /* inner loop */
08       Insert($TS$ , $Q$ , $POS$);
09       SCHEDULE($Q$);
10       Calculate the efficiency $\eta$ of the current partial test schedule;
11       if ($\eta > \eta_{max}$) then
12          $\eta_{max} := \eta$;
13          $TS_{best} := TS$;
14          $Q_{best} := Q$;
15       end if
16       Remove($TS$ , $Q$);
17    end for
18    Remove($TS_{best}$ , $U$);
19 end for
20 SCHEDULE($Q_{best}$);
```

Figure 9. Pseudo-code of the heuristic for test scheduling

```
ALG. 2. SCHEDULE(Queue of test sets :: $Q$)
21 for ($j$ = 1 to $max\{GetNumOfPar(\forall TS \in Q)\}$) loop /* outer loop */
22    for ($q$ = 1 to $|Q|$) loop   /* inner loop */
23       Choose the $q$-th test set $TS_q$ in $Q$ for scheduling;
24       if ($TS_q = \emptyset$) then
25          Skip $TS_q$ and continue with the next test set;
26       else
27          Schedule the first unscheduled test sub-sequence $TS_{q,j}$
               to the earliest available time moment
                    $t_{q,j} := GetFinishingTime(TS_{q,j-1}) + d_{q,j}$
                    where $d_q := InitialCoolingSpan(TS_q)$;
28          if ($FAILED$ to schedule $TS_{q,j}$ to $t_{q,j}$) then
29             Estimate the completion time $t_e$ of the entire test set $TS_q$
                  by either postponing $TS_{q,j}$ or repartitioning all the
                  unscheduled test sub-sequences in $TS_q$;
30             Choose the solution that has a smaller $t_e$ and
                  schedule the first unscheduled test sub-sequence;
31          end if
32       end if-then-else
33    end for
34 end for
```

Figure 10. Pseudo-code of the scheduling algorithm

Given a set of all test sets $U = \{TS_i \mid i = 1, 2, ..., n\}$ (line 1), the heuristic iteratively selects test sets and inserts them into a queue Q (line 2 to 19). The positions of the test sets in Q represents the order in which the test sets are considered for test scheduling (SCO), the closer to the queue head, the earlier to be considered.

The heuristic starts with an empty queue $Q = \emptyset$ (line 2). At each iteration step (line 5 to 18), the objective is to select one test set TS_k from U, and insert it into Q at a certain position POS, such that the $|Q| + 1$ test sets are put in a good order while the precedence between test sets excluding the newly inserted one remains unchanged. The algorithm terminates when all test sets in U have been moved into Q, and thereafter it schedules the partitioned test sets according to the SCO obtained in Q_{best} (line 20).

For each iteration step, there are $|U|$ alternative test sets for selection, where $|U|$ is the current number of test sets remaining in U. For each selected test set, there are $|Q| + 1$ alternative positions which the selected test set can be inserted to, where $|Q|$ is the current number of test sets that have already been inserted into Q throughout previous iteration steps. Thus, at one iteration step, there are $|U| \times (|Q| + 1)$ alternative solutions, in which a selected test set is associated with an insertion position in Q.

The example depicted in Figure 11 illustrates a situation that 3 test sets have been inserted in Q (TS_3, TS_8, and TS_6) and 5 test sets remain in U (TS_1, TS_2, TS_4, TS_5, and TS_7). For each test set in U, there are 4 positions for insertion, which the arrows point to. In this example, there are 20 alternative solutions for consideration. Note that each test set in the example has already been partitioned into a number of test sub-sequences, and the scheduling algorithm takes every individual test sub-sequence for scheduling (see ALG. 2).

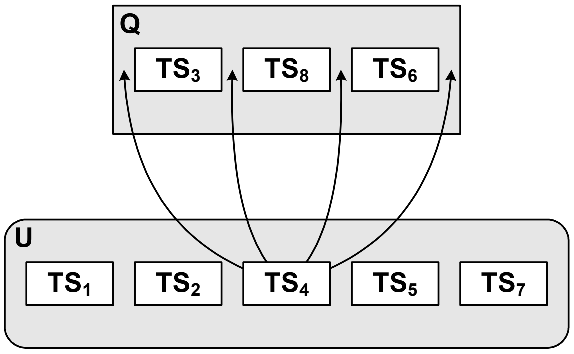


Figure 11. An example illustrating alternative solutions

We evaluate the obtained scheduling consideration order by the efficiency of the generated partial test schedule, the higher efficiency, the better the SCO. The partial test schedule is generated (line 9) by the scheduling algorithm ALG. 2. Based on the test-schedule efficiency defined below, we explore different solutions and make decisions according to the efficiency of the generated partial test schedules.

We define the efficiency of a test schedule, denoted with η, as follows. Suppose x is the size of the area covered by all scheduled test sub-sequences, and y is the total area size constrained by the bus bandwidth limit and the completion time moment of the test schedule. The efficiency of the test schedule is the value of x / y. The larger value of η represents the better test schedule.

Figure 12 illustrates how the efficiency of a test schedule is calculated. In the example, a test schedule is given as the area covered by slashed lines. By calculating x as the size of the area covered by the actual test schedule, and y as the size of the area covered by the large rectangle surrounded by thick lines, we get $\eta = x / y$.

By calculating and comparing the efficiencies of the alternative partial test schedules (line 10), the best solution that obtains the maximum efficiency is chosen. The maximum efficiency, the chosen test set, and the entire queue, are recorded in η_{max}, TS_{best}, Q_{best}, respectively (line 12 to 14). The iteration terminates when all test sets in U have been moved into Q. The obtained Q_{best} consists of all test sets in the best SCO, in which the test sets will be considered for scheduling (line 20).

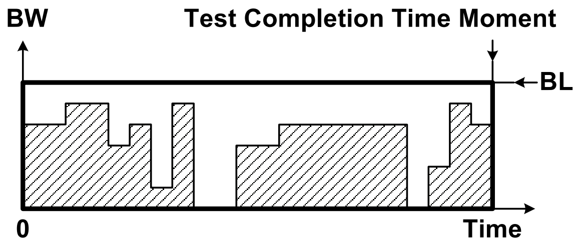


Figure 12. Illustration of calculating test schedule efficiency

The algorithm (ALG. 2) that schedules a queue of test sets is depicted in Figure 10, from line 21 to 34. Given a queue Q of test sets, the scheduling algorithm takes the first unscheduled test sub-sequence from every test set for scheduling, in a round-robin fashion. More concretely, the strategy of the scheduling algorithm is explained as follows. According to the SCO given in Q, the scheduler considers one test set at a time for scheduling. When considering each test set, the scheduler only schedules the first unscheduled test sub-sequence, and thereafter turns to consider the next test set. When one round is finished for all the test sets in Q, the scheduler takes the next round for consideration of scheduling test sub-sequences of all the test sets, in the same SCO. This procedure repeats until all test sub-sequences are scheduled.

Figure 13 illustrates how the scheduling algorithm works with an example of three test sets, TS_2, TS_1, and TS_3, sorted with the SCO of $\{TS_2, TS_1, TS_3\}$ in Q. The test set TS_2 has been initially partitioned into three test sub-sequences, TS_{21}, TS_{22}, and TS_{23}. The rest two test sets, TS_1 and TS_3, are both partitioned into four test sub-sequences. The OSCO of all test sub-sequences is $\{TS_{21}, TS_{11}, TS_{31}, TS_{22}, TS_{12}, TS_{32}, TS_{23}, TS_{13}, TS_{33}, TS_{14}, TS_{34}\}$, which is given by the dashed arrows.

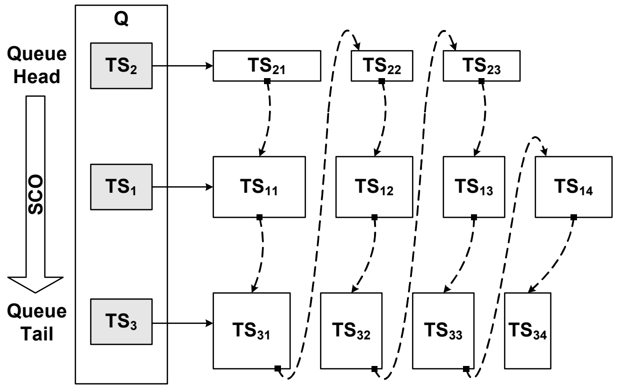


Figure 13. Illustration of the scheduling algorithm ALG. 2

In the given pseudo-code depicted in Figure 10, the scheduling algorithm is constructed with two nested loops. The outer loop (line 21 to 34) selects the first unscheduled test sub-sequence for the current test set, while the inner loop (line 22 to 33) selects a test set for scheduling according to its position in Q. The algorithm terminates when all the test sub-sequences have been scheduled. Note that the function *GetNumOfPar*(*TS*) in line 21 takes a test set *TS* as an input, and returns the number of test sub-sequences that the test set has been partitioned into.

When scheduling a test sub-sequence $TS_{q,j}$ (the j-th test sub-sequence of the q-th test set in Q, see line 23 to 27), the scheduler tries to schedule it to the earliest available time moment $t_{q,j}$ (line 27). The earliest time moment that a test sub-sequence can be scheduled to is the time moment when the required minimum cooling span succeeding the precedent test sub-sequence has finished. The minimum cooling span $d_{q,j}$ is given by the initial partitioning scheme for the test set TS_q (line 27).

Although we would like to schedule a test sub-sequence to the earliest available time moment, there can be constraints that make this impossible. Such a constraint is the availability of test-bus bandwidth to be allocated for the required time duration in order to complete the entire test sub-sequence. In Figure 14, for example, it is impossible to schedule the test sub-sequence $TS_{q,j}$ at time moment $t_{q,j}$, due to the insufficient space between the bandwidth limit *BL* and the area occupied by scheduled test sub-sequences (depicted with slashed lines). Actually, in this example, the earliest available time moment that $TS_{q,j}$ can be scheduled at is t_p.

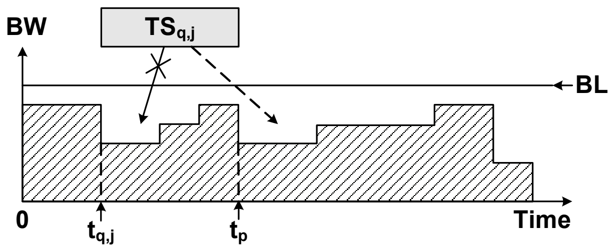
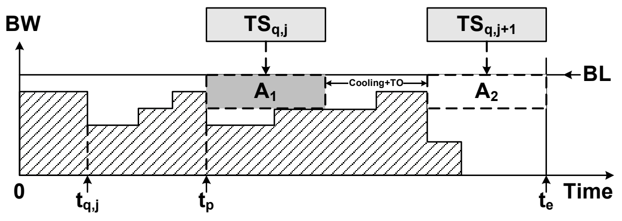
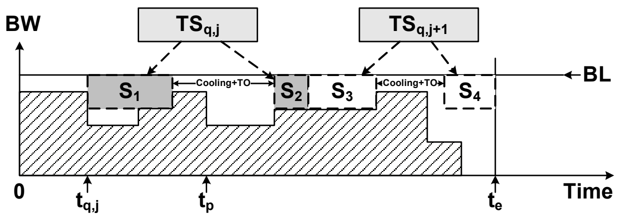


Figure 14. An example of scheduling constraints

When encountering such scheduling constraints, two alternatives can be considered. One is to postpone the entire test sub-sequence to a time moment that it can be successfully scheduled to. The other alternative is to split the test sub-sequence into smaller pieces such that the first piece can be squeezed into the available area. Figure 15 illustrates both solutions for the same example given in Figure 14, where the entire test sub-sequence $TS_{q,j}$ cannot be scheduled at time moment $t_{q,j}$. In Figure 15(a), the solution is to postpone the entire test sub-sequence $TS_{q,j}$ to time moment t_p, which means squeezing $TS_{q,j}$ into the dark grey rectangular area A_1 that the dashed arrow points to. Figure 15(b) illustrates the alternative solution, where $TS_{q,j}$ is split into two pieces which can fit the dark grey rectangular areas S_1 and S_2, respectively.



(a) Postponing the entire test sub-sequence



(b) Splitting the test sub-sequence into smaller pieces

Figure 15. Two solutions to schedule a test sub-sequence

Both solutions can result in long test schedules. The first solution, which postpones the entire test sub-sequence, also delays the succeeding test sub-sequences. This can result in delaying the completion of the entire test set. As illustrated in Figure 15(a), the succeeding test sub-sequence $TS_{q,j+1}$ is delayed and finishes at time moment t_e. The second solution, which splits the test sub-sequence into smaller pieces, also generates more partitions and introduces more time overheads (TO). In order to avoid these drawbacks, we repartition all the unscheduled test sub-sequences from the same test set, such that the total number of test sub-sequences will not increase dramatically due to the splitting. This is explained in Figure 15(b). After splitting $TS_{q,j}$ into two pieces which fits in S_1 and S_2 respectively, we also repartition the succeeding test sub-sequence $TS_{q,j+1}$ such that its two pieces fits into S_3 and S_4. Note that due to the splitting of $TS_{q,j}$ and $TS_{q,j+1}$, time overheads (denoted with *TO*) are added between the repartitioned test sub-sequences.

As demonstrated above, both solutions can be adopted when scheduling a test sub-sequence. In order to decide which solution should be employed, we estimate the completion time t_e for the entire test set (line 29), by assuming that all the unscheduled test sub-sequences of this test set can be scheduled to their earliest available time moments. The solution that results in an earlier estimated completion time is chosen (line 30). In the example given in Figure 15, the second solution should be chosen, since it leads to a smaller t_e. The scheduling algorithm terminates when all test sub-sequences of all test sets in Q have been scheduled (line 34).

It should be noted that by scheduling test sub-sequences in the demonstrated manner, the test sets have been interleaved and the temperatures of cores under test will not be higher than the temperature limit. This is because that the test sub-sequences are not longer than those in the initial partitioning schemes, and the cooling periods are not shorter than those in the initial partitioning schemes.

7. Experimental results

We have done experiments using SoC designs with randomly selected cores in the ISCAS'89 benchmarks. The designs for our experiments have 12 to 78 cores. We have used the approach proposed in [26] to obtain the power consumption values, taking the amounts of switching activity as inputs. HotSpot has been used for the temperature simulation and the imposed temperature limit for each core is set to 90°C. Temperature simulations have been done for the generated test schedules and the simulation results have confirmed that the temperatures of cores during the tests are below the imposed temperature limits.

With the first group of experiments, we demonstrate the impact on test application time due to the different flexibility of test set partitioning schemes.

We compare our heuristic with two other scheduling algorithms. The first algorithm employs a fixed order in which all the test sets are sorted decreasingly according to the length of test sets in their initial partitioning schemes. Then it schedules the entire test sets to the earliest available time moment, according to the obtained SCO. When scheduling the test sub-sequences of a test set, it keeps the regularity of the partitions and cooling periods given by the initial partitioning scheme. For the sake of convenience, we call the first algorithm "equal-length scheduling algorithm".

The second algorithm also employs the fixed order according to the lengths of partitioned test sets (longest first). However, different from the equal-length scheduling algorithm, it schedules a test set in two phases. In the first phase, it schedules only the first partition of all test sets, according to the obtained SCO. This is due to the fact that the first test sub-sequence is usually much longer than the other ones of the same test set in the initial partitioning scheme (see Figure 7). Then, in the second phase, it schedules all the remaining test sub-sequences of every test set, according to the same SCO. Similar to the first algorithm, it schedules test sets to the earliest available time moment. When scheduling the test sub-sequences in the second phase, it keeps the regularity of all test partitions and cooling periods given by the initial partitioning scheme, and the first cooling period after the first test sub-sequence may not be shorter than that in the initial partitioning scheme. It can be seen that by separating the scheduling of a test set into two phases, the restriction on partitioning regularity is slightly relaxed, thus this algorithm has higher flexibility on test set partitioning schemes than the equal-length partitioning algorithm. We call the second scheduling algorithm "two-phase scheduling algorithm".

Compared to the equal-length scheduling and two-phase scheduling algorithm, our heuristic has the highest flexibility on test set partitioning schemes, since it allows repartitioning test sets and allows arbitrarily increasing cooling periods during the scheduling.

Experimental results regarding the first group of experiments are shown in Table 1. The first column in the table lists the number of cores used in the designs. Columns 2, 4, and 6 show the test application times of the generated test schedules for the corresponding designs, by using the equal-length scheduling algorithm, the two-phase scheduling algorithm, and our heuristic, respectively. Columns 3, 5, and 7 list the CPU times for executing the corresponding algorithms. Columns 8 and 9 show the percentage of TAT reduction by using our heuristic, against using the equal-length scheduling algorithm and the two-phase scheduling algorithm, respectively. It can be seen that by eliminating restrictions on the regularity of partitioning schemes, the TAT is in average 30.6% and 20.5% shorter than that of the equal-length scheduling algorithm and the two-phase scheduling algorithm, respectively.

Table 1. Our heuristic vs. equal-length scheduling algorithm vs. two-phase scheduling algorithm (to demonstrate the impact of relaxing regularity restriction on test partitioning schemes)

#cores	Equal-length		Two-phase		Our heuristic		TAT gain (%)	
	TAT	CPU Times (s)	TAT	CPU Times (s)	TAT	CPU Times (s)	From Equal-length	From Two-phase
12	1502	0.01	1390	0.01	1048	2.74	30.2%	24.6%
18	2761	0.02	2029	0.01	1535	5.41	44.4%	24.3%
24	3975	0.05	3571	0.02	2318	21.88	41.7%	35.1%
30	2831	0.01	2510	0.02	1915	32.41	32.4%	23.7%
36	3587	0.08	3368	0.08	2539	67.52	29.2%	24.6%
42	4845	0.03	4012	0.03	3334	101.39	31.2%	16.9%
48	4878	0.06	4513	0.06	3509	151.33	28.1%	22.2%
54	5696	0.06	5024	0.08	4290	244.36	24.7%	14.6%
60	6303	0.19	5504	0.13	4692	371.73	25.6%	14.8%
66	6868	0.34	5889	0.41	5069	511.88	26.2%	13.9%
72	7903	0.17	6923	0.22	5822	720.53	26.3%	15.9%
78	7900	0.72	6803	0.77	5769	987.75	27.0%	15.2%
AVG	N/A	N/A	N/A	N/A	N/A	N/A	30.6%	20.5%

The second group of experiments has been set up in order to see how efficient the test schedules are, which are generated by our heuristic. We compare our heuristic with other two algorithms, a straight forward algorithm (SF) and the simulated annealing algorithm (SA). In this group of experiments, we assume the same flexibility for all the three algorithms, i.e. all of them employ flexible partitioning of test sets and arbitrary length of cooling periods.

All the three algorithms employ the same scheduling algorithm (ALG. 2). The only difference between them is how they generate the SCO for all test sets. The straight forward algorithm sorts all test sets decreasingly by the lengths of the entire test sets with the initial partitioning schemes. According to the obtained SCO, the scheduler chooses each test set and schedules the first unscheduled test sub-sequences to the earliest available time moment, until all test sub-sequences of every test set are scheduled.

The simulated annealing algorithm employs the same scheduling algorithm ALG. 2 to schedule the test sub-sequences, while the SCO of test sets is generated based on a simulated annealing strategy. When a randomly generated SCO is obtained, the scheduler is invoked to schedule the test sub-sequences according to the current SCO. During iterations, the best SCO that leads to the shortest test schedule is recorded and the algorithm returns this recorded solution when the stopping criterion is met.

The experimental results are listed in Table 2. Column 1 lists the number of cores used in the designs for experiments. Column 2 shows the test application time of the generated test schedule when the straight forward algorithm is employed, and column 3 lists the corresponding CPU times to obtain the test schedules. Similarly, columns 4 and 5 are the TAT and CPU times for our heuristic, respectively (which are the same as the columns 6 and 7 in Table 1). Columns 6 and 7 list the TAT and execution times for the simulated annealing algorithm. In columns 7 and 8, the percentage of reduced TAT of the test schedules generated by our heuristic are listed, compared to those generated by the straight forward algorithm and the simulated annealing algorithm, respectively.

Table 2. Our heuristic vs. straight forward algorithm vs. simulated annealing algorithm (to show the efficiency of our heuristic in terms of generating efficient test schedules)

#cores	SF		Our heuristic		SA		TAT gain (%)	
	TAT	CPU Times (s)	TAT	CPU Times (s)	TAT	CPU Times (s)	From SF	From SA
12	1213	0.01	1048	2.74	992	148.31	13.6%	-5.6%
18	1716	0.01	1535	5.41	1513	208.06	10.5%	-1.5%
24	2632	0.01	2318	21.88	2234	229.94	11.9%	-3.8%
30	2274	0.01	1915	32.41	1869	417.08	15.8%	-2.5%
36	3161	0.01	2539	67.52	2494	540.48	19.7%	-1.8%
42	3846	0.01	3334	101.39	3292	631.00	13.3%	-1.3%
48	4328	0.01	3509	151.33	3485	898.77	18.9%	-0.7%
54	4877	0.01	4290	244.36	4051	675.44	12.0%	-5.9%
60	5274	0.01	4692	371.73	4457	2171.73	11.0%	-5.3%
66	5725	0.01	5069	511.88	4917	2321.39	11.5%	-3.1%
72	6538	0.01	5822	720.53	5689	1994.56	11.0%	-2.3%
78	6492	0.01	5769	987.75	5702	3301.45	11.1%	-1.2%
AVG	N/A	N/A	N/A	N/A	N/A	N/A	13.4%	-2.9%

The comparison between our heuristic and the straight forward algorithm aims to show how much TAT can be reduced by a more advanced test scheduling technique. On the other hand, the comparison between our heuristic and the simulated annealing algorithm is to find out how close the generated test schedule is to a solution which is assumed to be close to the optimal one. In order to generate a close-to-optimal solution, the SA algorithm has been run for long optimization times.

It can be seen that, when using our heuristic, the TAT is in average 13.4% shorter than those using the straight forward algorithm. The TAT is in average 2.9% longer than those using the simulated annealing algorithm which however needs much longer execution times.

8. Conclusions

In this paper, we have proposed a heuristic to generate thermal-safe test schedules for systems-on-chip and minimize the test application time. Based on the initial partitioning scheme generated by a temperature simulation guided procedure, the heuristic utilizes the flexibility of changing the length of test sub-sequences and the cooling periods between test sub-sequences, and interleaves them to generate efficient test schedules. Experimental results have shown the efficiency of our heuristic.

References

[1] S. Borkar. “Design challenges of technology scaling”. *IEEE Micro*, Vol. 19, Iss. 4, pp. 23-29, 1999.

[2] S. Gunther, F. Binns, D. M. Carmen, and J. C. Hall. “Managing the impact of increasing microprocessor power consumption”. *Intel Technology Journal*. 2001.

[3] R. Mahajan. “Thermal management of CPUs: A perspective on trends, needs and opportunities”. Keynote presentation at the *8th Int’l Workshop on THERMal INvestigations of ICs and Systems (THERMINIC)*. 2002.

[4] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan. “Temperature-aware microarchitecture: Modeling and implementation”. *ACM Transactions on Architecture and Code Optimization (TACO)*. Vol. 1, No. 1. pp. 94-125, Mar. 2004.

[5] B. Pouya and A. Crouch. “Optimization trade-offs for vector volume and test power”. *International Test Conference (ITC)*, 2000, pp. 873-881.

[6] C. Shi and R. Kapur. “How power-aware test improves reliability and yield”. *EE Times*, September 15, 2004. http://www.eetimes.com/showArticle.jhtml?articleID=47208594.

[7] B. T. Murray, and J. P. Hayes. “Testing ICs: Getting to the core of the problem”. *IEEE Transactions on Computer*, Vol. 29, pp. 32-39, Nov. 1996.

[8] Y. Zorian, E. J. Marinissen, and S. Dey. “Testing embedded core-based system chips”. *IEEE International Test Conference (ITC)*, 1998, pp. 130-143.

[9] Y. Huang, W.-T. Cheng, C.-C. Tsai, N. Mukherjee, O. Samman, Y. Zaidan, and S. M. Reddy. “Resource allocation and test scheduling for concurrent test of core-based SoC design”. *IEEE Asian Test Symposium (ATS)*, 2001, pp. 265-270.

[10] E. Larsson, and Z. Peng. “An integrated framework for the design and optimization of SoC test solutions”. *Journal of Electronic Testing; Theory and Applications (JETTA)*, Vol. 18, No. 4/5, pp. 385-400, 2002.

[11] J. Aerts, and E. J. Marinissen, “Scan chain design for test time reduction in core-based ICs”, *International Test Conference (ITC)*, 1998, pp. 448-457.

[12] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, “Test access mechanism optimization, test scheduling, and test data volume reduction for System-on-Chip”, *IEEE Transactions on Computer*, Vol. 52, No. 12, Dec. 2003.

[13] P. Varma, and B. Bhatia, “A structured test re-use methodology for core-based system chips”, *International Test Conference (ITC)*, 1998, pp. 294-302.

[14] R. Chou, K. Saluja, and V. Agrawal. “Scheduling tests for VLSI systems under power constraints”. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 5(2):175-184, June 1997.

[15] Z. He, Z. Peng, and P. Eles. “Power constrained and defect-probability driven SoC test scheduling with test set partitioning”. *Design Automation and Test in Europe Conference (DATE)*, 2006, pp. 291-296.

[16] C. Liu, K. Veeraraghavant, and V. Iyengar. “Thermal-aware test scheduling and hot spot temperature minimization for core-based systems”. *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, 2005, pp. 552-560.

[17] P. Rosinger, B. M. Al-Hashimi, and K. Chakrabarty. “Thermal-safe test scheduling for core-based System-on-Chip integrated circuits”. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. Vol. 25, No. 11, pp. 2502-2512, Nov. 2006.

[18] Z. He, Z. Peng, P. Eles, P. Rosinger, and B. M. Al-Hashimi. “Thermal-aware SoC test scheduling with test set partitioning and interleaving”. *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, 2006, pp. 477-485

[19] B. S. Baker, E. G. Coffman Jr., and R. L. Rivest. “Orthogonal packings in two dimensions”. *SIAM Journal on Computing*, Vol. 9, No. 4, pp. 846-855, Nov. 1980.

[20] H. Dyckhoff, “A typology of cutting and packing problems”. *European Journal of Operational Research*, Vol. 44, No. 2, pp. 145-159. 1990

[21] N. Lesh, J. Marks, A. McMahon, and M. Mitzenmacher. “Exhaustive approaches to 2D rectangular perfect packings”, *Elsevier Information Processing Letters*, Vol. 90, No. 1, pp. 7-14, Apr. 2004.

[22] W. Huang, S. Ghosh, K. Sankaranarayanan, K. Skadron, and M. R. Stan. “HotSpot: Thermal modeling for CMOS VLSI systems.” *IEEE Transactions on Component Packaging and Manufacturing Technology*. 2005. (to appear).

[23] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. “Temperature-aware microarchitecture.” *International Symposium on Computer Architecture*, 2003, pp. 2-13.

[24] W. Huang, M. R. Stan, K. Skadron, K. Sankaranarayanan, S. Ghosh, and S. Velusamy. “Compact thermal modeling for temperature-aware design”. *Design Automation Conference (DAC)*, 2004. pp. 878-883.

[25] S. K. Goel, and E. J. Marinissen. “Control-aware test architecture design for modular SoC testing”. *European Test Workshop (ETW)*, 2003. pp. 57-62.

[26] S. Samii, E. Larsson, K. Chakrabarty, and Z. Peng. “Cycle-accurate test power modeling and its application to SoC test scheduling”. *IEEE International Test Conference (ITC)*, 2006, pp. 1-10.