Thermal–Aware SoC Test Scheduling

Zhiyuan He, Zebo Peng, Petru Eles
Department of Computer and Information Science, Linköping University, Sweden

ABSTRACT

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 chapter, 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 applying a test sub-sequence will not drive the core temperature going beyond the limit. Furthermore, 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 an approach to minimize the test application time by exploring alternative test partitioning and interleaving schemes with variable length of test sub-sequences and cooling periods as well as alternative test schedules. Experimental results have shown the efficiency of the proposed approach.

INTRODUCTION

Nanoscale technology has become the mainstream in the design and production of integrated circuits (ICs). In the latest generation of IC designs, the power density has been substantially increased (Borkar, 1999), (Gunther, Binns, Carmean, & Hall, 2001). As a consequence of the elevated power density, high temperature in the chip becomes a critical challenge (Skadron et al., 2004), (Mahajan, 2002). In particular, compared to the normal functional mode, testing consumes more power (Pouya & Crouch, 2000), (Shi & Kapur, 2004), leading to an even higher temperature on silicon dies. Therefore, temperature control during test is required in order to avoid damages to the circuits under test. Some advanced cooling techniques are proposed to reduce the temperature of ICs, but they substantially increase the overall cost. Other techniques, such as lower frequency and reduced speed, can partly solve the high temperature problem, while making them inapplicable to at-speed test and leading to longer test application time (TAT).

In the case of system-on-chip (SoC) test, the problems of long test time and high temperature become more severe. Due to the high power consumption and high temperature in the latest generation of SoCs, novel techniques are proposed to tackle the problem of long test time. In (Rosinger, Al-Hashimi, & Nicolici, 2004), (Girard, Landrault, Pravossoudovitch, & Severac, 1998), low-power test techniques are proposed to reduce the power consumption during tests. Some other works focus on power-constrained test scheduling (Chou, Saluja, & Agrawal, 1997), (Larsson & Peng, 2006), (Chakrabarty, 2000), (He, Peng, & Eles, 2006), targeting test time minimization restricted in a fixed power envelope. However, only using the power-aware techniques cannot fully avoid the overheating problem because of the complex thermal phenomenon in modern ICs (Rosinger, Al-Hashimi, & Chakrabarty, 2006).

Recently, thermal-aware test techniques have been proposed in order to solve the overheating problem during SoC tests. Liu et al. proposed a technique (Liu, Veeraraghavan, & Iyengar, 2005) to evenly distribute the generated heat across the chip during tests, and as a result, avoid high temperature. Rosinger et al. proposed an approach (Rosinger et al., 2006) to generate thermal-safe test schedules that minimizes the test application time and reduces temperature variation across the silicon die, utilizing the information of core adjacency. In (Yu, Yoneda, Chakrabarty, & Fujiwara, 2007), Yu et al. addressed the thermal-safe TAM/wrapper co-optimization problem and proposed a test scheduling approach to generate efficient test
schedules. Although these proposed approaches generate efficient test schedules, they make strong and simplifying assumption that a core under test is never overheated during the application of a single test set. In this chapter, we address the test time minimization problem for system-on-chip test with thermal awareness. We assume that applying a single test set to a core may raise the temperature of the core under test and exceed a temperature limit beyond which the core may be damaged. In order to generate thermal-safe test schedules and at the same time minimize the test time, we propose a test set partitioning and interleaving (TSPI) technique. Based on the proposed TSPI technique, we propose a heuristic-based approach (He, Peng, & Eles, 2007) which explores alternative test set partitioning and interleaving schemes in which partitions and cooling periods have arbitrary lengths.

BACKGROUND AND MOTIVATION

Thermal-Aware Testing

CMOS technology scaling has enabled the industry to improve the speed and performance of ICs. While all the physical dimensions of a transistor are scaled down, the device area is reduced. At the same time, designers tend to add more functionality and build more complex circuits into chips, leading to increasing die area to accommodate more transistors (Vassighi & Sachdev, 2006). It is shown in (Rabaey, Chandrakasan, & Nikolic, 2003) that the die area sizes of Intel processors increase approximately 7% per year, and the number of transistors are doubled per generation. The latest generation of microprocessors has billions of transistors.

With technology scaling, the power consumption of high-performance chips increases exponentially, especially for the chips manufactured with deep-submicron technologies. The main reason is that the scaling of the threshold voltage $V_{TH}$ causes an increase in sub-threshold leakage current (Rabaey et al., 2003). As a consequence of the elevated power consumption, the power density of chips also increases. The power density of a chip is defined as the power dissipated by the chip per unit area under nominal frequency and normal operating conditions. The reason for increasing power density is that the positive supply voltage $V_{DD}$ and the saturated drain current $I_{DSAT}$ are scaling at a lower rate than the device area size (Vassighi & Sachdev, 2006).

The increasing power consumption and power density result in higher junction temperature, especially in high-performance processors and application-specific integrated circuits (ASICs). It is reported in (ITRS, 2009) that the maximum junction temperature of high-performance chips is 90°C, for both near- and long-term years. Junction temperature is one of the key parameters of CMOS devices, as it affects the performance, power consumption, and reliability of the ICs (Segura & Hawkins, 2004), (Vassighi & Sachdev, 2006).

Carrier mobility decreases as temperature increases, because carriers collide with the Si-crystal lattice more frequently at a higher junction temperature. Consequently, the driving currents of transistors decrease with reduced carrier mobility, which causes a degradation of device performance. Similar effects occur in the thin interconnect metal lines using aluminum or copper process. At a higher temperature, the metal resistivity increases, leading to higher interconnect resistance. Thus, circuit performance degradation is often encountered when operating temperature increases. The performance degradation should be avoided for both functional and testing conditions. In the functional mode, the performance of an IC directly affects the system efficiency. In the testing mode, the performance degradation due to high junction temperature may fail the test and cause loss of yield.

The elevation of junction temperature results in an increase in leakage current and higher device power consumption. The elevated power consumption in turn increases the junction temperature (Vassighi & Sachdev, 2006). The positive feedback between the leakage current and junction temperature may lead the chip to thermal runaway in extreme cases. When a chip is in a stress condition, such as a burn-in test where chips are tested with purposely elevated supply voltage and junction temperature, the chance of thermal runaway is much higher. For ICs manufactured with nanometer technology, the situation of positive feedback leading to thermal runaway is more likely to happen.
Another issue closely related to junction temperature is the long-term reliability of ICs. Many failure mechanisms, such as electron migration, gate oxide breakdown, hot electron effects, negative bias temperature instability, etc., are accelerated when junction temperature is elevated (Segura & Hawkins, 2004). In order to maintain the device reliability and the lifetime of ICs, it is very important to efficiently and safely manage the transistor junction temperature and operating temperature of other parts in ICs. It is reported that even a small variation of junction temperature (10–15°C) may result in a factor of two times reduction in device lifetime (Vassighi & Sachdev, 2006).

### SoC Testing

Design and manufacturing of integrated circuits have moved into the deep submicron technology regime. Scaling of process technology has enabled a dramatic increase of the integration density, which enables more and more functionalities to be integrated into a single chip. With the improving system performance, the design complexity has also been increasing steadily. A critical challenge to electronic engineers is that the shorter life cycle of an electronic system has to compete with its longer design cycle. Therefore, more efficient hierarchical design methodologies, such as the core-based SoC design (Murray & Hayes, 1996), (Zorian, Marinissen, & Dey, 1999), have to be deployed in order to reduce the time-to-market.

A common approach to modern core-based SoC design reuses pre-designed and pre-verified intellectual property (IP) cores that are provided by different vendors. It integrates the IP cores into the system and manufactures the system on a single silicon die. The cores are usually processors, memory blocks, bus structure, peripherals interfaces, analog circuits, user defined logic (UDL) etc.

In order to test individual cores in a SoC, a test architecture consisting of certain resources has to be available. The test architecture for SoCs usually includes a test source, a test sink, and a test access mechanism (TAM). A test source is a test-pattern container/generator which can be either external or on-chip. A typical external test source is an automated test equipment (ATE) which stores generated test patterns in its local memory. An on-chip test source can be a linear feedback shift register (LFSR), a counter, or a ROM/RAM which stores already generated test patterns. A test sink is a test response/signature analyzer that detects faults by comparing test responses/signatures with the correct ones. An ATE can be an external test sink that analyzes the test responses/signatures transported from the cores under test (CUTs). The test sink can also be integrated on the chip so that the test responses/signatures can be analyzed on-the-fly. A TAM is an infrastructure designed for test data transportation. It is often used to transport test patterns from the test source to CUTs and to transport test responses/signatures from CUTs to the test sink. The TAM can be a reusable functional bus infrastructure (Harrod, 1999), such as the advanced microprocessor bus architecture (AMBA) (Flynn, 1997), or a dedicated test bus.

An example of test architecture for external SoC test 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. When test patterns have been activated, the captured test responses are also transported to the ATE through the test bus.

![Figure 1. Test architecture for external test](image)
As the complexity of systems has been increasing along with the rapid advances of technology, the amount of required test data for SoC testing is growing substantially. This leads to large amount of test data and long test application time, which poses great challenges to SoC test. In order to reduce the testing cost, core-based SoC test has received a wide variety of research interests (Chakrabarty, 2000), (Nicolici & Al-Hashimi, 2000), (Iyengar, Chakrabarty, & Marinissen, 2002), (Larsson, Arvidsson, Fujiwara, & Peng, 2004), (He et al., 2006) concerning advanced test architecture design, test resource allocation, and test scheduling.

In this chapter, we assume that the tester employed for a SoC test is either an automatic test equipment (ATE) or an embedded tester in the chip. The tester consists of two major components, a test controller and a memory. The memory stores a test schedule and the generated test patterns. The test controller reads the test schedule and transports the test data to/from the CUTs accordingly. A test bus is used for the test data transportation between the tester and the cores. Each core is connected to the test bus through dedicated TAM wires. Through the test bus and TAM wires, test patterns are sent to the CUTs and test responses are sent back to the tester.

**Thermal-Aware Test Scheduling**

The state of the art of SoC test has shown that the large test data volume and the long test application time substantially increase the testing cost. When considering SoC test in a thermal-safe context, a long test process applied to a core may lead to a high temperature even before the test is completed. This means that the CUT may be damaged if its temperature goes beyond a certain limit, and the test is not interrupted in time. Thus, in order to prevent overheating, an individual test has to be stopped when the temperature of the core reaches the temperature limit, and a cooling period is needed before the test can be continued. In this chapter, we refer to this cooling as passive cooling, meaning that the core is not activated and does not consume dynamic power. Thus, by partitioning an individual test into a number of test sub-sequences and inserting cooling periods between them, we can avoid overheating during the entire test process. Figure 2(a) gives an example of test set partitioning. It illustrates a scenario in which an entire test set is partitioned into four test sub-sequences, $TS_1$, $TS_2$, $TS_3$, and $TS_4$, with a cooling period introduced between every two consecutive sub-sequences. In this way, the temperature of the core remains under the imposed temperature limit.

When test set partitioning is employed to avoid overheating, the efficiency of the test process and the utilization of the test bus should also be considered for test scheduling. Introducing long cooling periods between test sub-sequences of a core can substantially increase the test application time. On the other hand, during the cooling periods of a core, the bandwidth of the test bus previously allocated to this core is not utilized. Thus we can release the bus bandwidth reserved for a core during its cooling periods, and allocate the released bus bandwidth to other cores for their test-data transportations and test applications. In this way, the test sets of different cores are interleaved and thus the test application time can be reduced. Figure 2(b) shows an example where two partitioned test sets are interleaved so that the test time is reduced with no need for extra bus bandwidth.

In this chapter, we aim to minimize the TAT 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. 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 3 gives a motivational example for our test time minimization problem. In this example, three test sets, namely $TS_1$, $TS_2$, and $TS_3$, are partitioned into test sub-sequences with cooling periods inserted. 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 thermal simulation. Figure 3(a) illustrates a feasible test schedule while Figure 3(b) depicts an alternative test schedule where a different partitioning and interleaving scheme is adopted. 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.
Problem Formulation

Suppose that a system-on-chip, denoted with $S$, consists of $n$ cores, denoted with $C_1, C_2, \ldots, C_n$, respectively, which are placed according to a floorplan, denoted with $F$. In order to test core $C_i$ ($1 \leq i \leq n$), $l_i$ test patterns are generated, and the test set is denoted with $T_{Si}$. The test patterns/responses are transported through the test bus and the dedicated TAM wires to/from core $C_i$, and the amount of required test-bus bandwidth is denoted with $BW_i$. The test bus is designed to transport test data for different cores in parallel and the bandwidth limit is denoted with $BL$ ($BL \geq BW_i$, $i = 1, 2, \ldots, n$). We assume that continuously applying test patterns belonging to $T_{Si}$ increases the temperature of core $C_i$, approaching a temperature limit, denoted with $TL_i$. If the temperature of core $C_i$ goes beyond $TL_i$, the core may be damaged. In order to avoid overheating a CUT, a test set needs to be partitioned into a number of shorter test sub-sequences and a cooling period needs to be inserted between two partitioned test sub-sequences. The problem that we address is how to generate a test schedule for system $S$ such that the TAT is minimized while the test-bus bandwidth limit is satisfied and the temperatures of all cores during tests remains below the corresponding temperature limits. The formal problem formulation is given in Figure 4.
Overall Solution Strategy

We have proposed an overall solution strategy to solve the formulated problem in an iterative algorithm containing three steps, as illustrated in Figure 5. In the first step, we generate an initial partitioning scheme for every test set by using thermal 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. In order to ensure the thermal safety, we perform a thermal simulation to check whether the generated test schedule will lead to temperature violations. If temperature violations are detected, we use a new temperature limit to re-partition the test sets and re-generate the test schedule. In each iteration, the new temperature limit equals the current temperature limit minus the maximum amount of the temperature overshoots detected in the thermal simulation. The algorithm stops when no temperature violation is detected.

**Input:**
- SoC floorplan $F$ including physical parameters of die and package;
- The set of tests for each core $\{TS_i | i = 1, 2, \ldots, n\}$;
- The set of required test-bus bandwidth for each test $\{BW_i | i = 1, 2, \ldots, n\}$;
- Test-bus bandwidth limit $BL$;
- Temperature limit of each individual core $\{TL_i | i = 1, 2, \ldots, n\}$.

**Output:**
- A test schedule with the minimized test application time (TAT)

**Subject to the following two constraints:**
1. At any time moment $t$ before the SoC test is accomplished, the total amount of allocated test-bus width $BW(t)$ is less than or equal to the bus bandwidth limit $BL$, i.e. $\forall t, BW(t) \leq BL$, where $BW(t) := \sum BW_i(t)$;
2. At any time moment $u$ when a test is applied to core $C_i$, the instantaneous temperature $T_i(u)$ of core $C_i$ is less than or equal to the corresponding temperature limit $TL_i$, i.e. $T_i(u) \leq TL_i$.

Figure 4. Problem formulation

Figure 5. Overall solution strategy

In order to generate thermal-safe partitioning schemes, we have used a thermal simulator, HotSpot (Huang et al., 2006), to simulate instantaneous temperatures of individual cores during tests. HotSpot
assumes a circuit package configuration widely used in modern IC designs, and it computes a compact thermal model based on the analysis of three major heat flow paths existing in the assumed packaging configuration. 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 work, we assume that the temperature influences between cores are negligible, since the heat transfer in the vertical direction dominates the overall heat transfer.

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 thermal 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 thermal simulation continues while the test process on core $C_i$ has to be stopped until the temperature goes down to a certain degree. It needs a relatively long time 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 6). Moreover, from the thermal 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 $k$ (which can be experimentally set by the designers) at time moment $t_c$. At this moment, we have obtained the duration of the first cooling period $d_{i1} = t_c - t_{h1}$. Restarting the test process from time moment $t_c$, we repeat this heating-and-cooling procedure throughout the thermal 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 $\{TS_{ij} | j = 1, 2, \ldots, m\}$ and between every two consecutive test sub-sequences, the duration of the cooling period is $\{d_{ij} | j = 1, 2, \ldots, m-1\}$, respectively. Figure 6 depicts an example of partitioning a test set into four thermal-safe test sub-sequences with three cooling periods added in between.

![Figure 6. Initial partitioning scheme](image)

Once the initial thermal-safe partitioning scheme is obtained, the rest of the proposed approach 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 chapter, 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 (Baker, Coffman, & Rivest, 1980), (Dyckhoff, 1990), (Lesh, Marks, McMahon, & Mitzenmacher, 2004). 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 (Goel & Marinissen, 2003), (He et al., 2006) 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 may 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.

### 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 7 shows a simple example illustrating the impact of different scheduling consideration order on the test schedule of three test sets, TS₁, TS₂, and TS₃, each of which is partitioned into two test sub-sequences. Figure 7(a) and Figure 7(b) respectively depicts the test schedule when the test sets are considered for scheduling in the order of {TS₁, TS₂, TS₃} and {TS₃, TS₂, TS₁}. 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 once. Instead, we take only 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 {TS₁₁, TS₂₁, TS₃₁, TS₁₂, TS₂₂, TS₃₂} and {TS₃₁, TS₂₁, TS₁₁, TS₃₂, TS₂₂, TS₃₂}, for the case of Figure 7(a) and Figure 7(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.
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 8, 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 9.

Given a set of all test sets $U = \{TS_i | i = 1, 2, \ldots, n\}$ (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 $Q$.

**Figure 7. Impact of SCO on TAT**

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

(b) Test schedule with the SCO $\{TS_3, TS_2, TS_1\}$
The example depicted in Figure 10 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 ALG2).

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 ALG2. 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 $\eta$, 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 $\eta$ represents the better test schedule.

Figure 11 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 \( \eta = \frac{x}{y} \).

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 \( \eta_{\text{max}}, TS_{\text{best}}, Q_{\text{best}} \), respectively (line 12 to 14). The iteration terminates when all test sets in \( U \) have been moved into \( Q \). The obtained \( Q_{\text{best}} \) consists of all test sets in the best SCO, in which the test sets will be considered for scheduling (line 20).

The algorithm (ALG2) that schedules a queue of test sets is depicted in Figure 9. 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 12 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.

In the given pseudo-code depicted in Figure 9, 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 \( \text{GetNumOfPar}(TS) \) 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 13, 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 $t_p$.

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 14 illustrates both solutions for the same example given in Figure 13, where the entire test sub-sequence $TS_{q,j}$ cannot be scheduled at time moment $t_{q,j}$. In Figure 14(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 14(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.

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 14(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 14(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.

---

**Figure 12. Illustration of test scheduling algorithm ALG2**

**Figure 13. An example of scheduling constraints**
(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 14, 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).

![Figure 14. Two solutions to schedule a test sub-sequence](image)

**Heuristic to Ensure Thermal-Safety**

The proposed test scheduling approach can generate efficient test schedules, but may not ensure that the generated test schedule is thermal-safe. This is because the original thermal-safe partitioning schemes may have been changed due to regrouping test sub-sequences. Therefore, we propose a heuristic to ensure the thermal-safety of generated test schedules. The pseudo-code of the heuristic, denoted with $ALG3$ is given in Figure 15. It takes the original temperature limits, the generated test schedule and the test sets as inputs and generates a thermal-safe test schedule as the output. The algorithm perform a thermal simulation to check whether the generated test schedule is thermal-safe or not, with respect to the original temperature limits (line 37). In case the generated test schedule is not thermal-safe, the maximum amount of temperature overshoot, denoted with $D$, is obtained by scanning the thermal simulation result (line 39) and new temperature limits are calculated by subtracting $D$ from the old temperature limits (line 40). Thereafter, a new thermal-safe partitioning scheme is generated using the new temperature limits (line 41) and a new test schedule is generated (line 42) using the test scheduling approach presented in previous sections. A new thermal simulation is then performed with the newly generated test schedule against the original temperature limits (line 43). The procedure is repeated until a thermal-safe test schedule is generated and then algorithm output the test schedule (line 45). Using this heuristic, we can ensure the thermal safety of the generated test schedule while keeping the goal of minimizing the test schedule length.
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. For example, a SoC design of 12 cores can consist of the following ISCAS’89 cores: s1423mg, s1488mg, s1494mg, s3271mg, s3330mg, s3384mg, s4863mg, s5378mg, s6669mg, s9234mg, s13207mg, and s15850mg. We have used the approach proposed in (Samii, Larsson, Chakrabarty, & Peng, 2006) to obtain the power consumption values, taking the amounts of switching activity as inputs. HotSpot has been used for the thermal simulation and the imposed temperature limit for each core is set to 90°C.

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 6). 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

```
ALG3. GenThermalSafeSchedule(Original temperature limits :: TL_orig, Generated test schedule :: GTS, Test sets :: TS, Floorplan :: F)

35 TLnew := TL_orig;
36 GTSnew := GTS;
37 THERMALSAFE := ThermalSafetyCheck(GTS, TL_orig); /* Perform a thermal simulation to check thermal safety */
38 while (NOT THERMALSAFE) loop
39  D := maximum amount of temperature overshoot; /* Obtain the amount of temperature overshoot */
40  TLnew := TLnew − D; /* Reduce the temperature limit */
41  PSnew := ThermalSafePartitioning(TS, F, TLnew); /* Generate a new partitioning scheme */
42  GTSnew := TestScheduling(TS, PSnew); /* Generate a new test schedule */
43  THERMALSAFE := ThermalSafetyCheck(GTSnew, TL_orig); /* Perform a thermal simulation to check thermal safety */
44  end while
45  Output GTSnew as a thermal-safe test schedule;
```

Figure 15. Pseudo-code of heuristic to generate a thermal-safe test schedule
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.

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 straightforward 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 (ALG2). The only difference between them is how they generate the SCO for all test sets. The straightforward 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 ALG2 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 straightforward 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 straightforward algorithm and the simulated annealing algorithm, respectively.

### Table 1. Our heuristic vs. equal-length scheduling algorithm vs. two-phase scheduling algorithm

<table>
<thead>
<tr>
<th># of Cores</th>
<th>Equal-length</th>
<th>Two-phase</th>
<th>Our heuristic</th>
<th>TAT gain (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TAT (s)</td>
<td>CPU Times (s)</td>
<td>TAT (s)</td>
<td>CPU Times (s)</td>
</tr>
<tr>
<td>12</td>
<td>1502</td>
<td>0.01</td>
<td>1390</td>
<td>0.01</td>
</tr>
<tr>
<td>18</td>
<td>2761</td>
<td>0.02</td>
<td>2029</td>
<td>0.01</td>
</tr>
<tr>
<td>24</td>
<td>3975</td>
<td>0.05</td>
<td>3571</td>
<td>0.02</td>
</tr>
<tr>
<td>30</td>
<td>2831</td>
<td>0.01</td>
<td>2510</td>
<td>0.02</td>
</tr>
<tr>
<td>36</td>
<td>3587</td>
<td>0.08</td>
<td>3368</td>
<td>0.08</td>
</tr>
<tr>
<td>42</td>
<td>4845</td>
<td>0.03</td>
<td>4012</td>
<td>0.03</td>
</tr>
<tr>
<td>48</td>
<td>4878</td>
<td>0.06</td>
<td>4513</td>
<td>0.06</td>
</tr>
<tr>
<td>54</td>
<td>5696</td>
<td>0.06</td>
<td>5024</td>
<td>0.08</td>
</tr>
<tr>
<td>60</td>
<td>6303</td>
<td>0.19</td>
<td>5504</td>
<td>0.13</td>
</tr>
<tr>
<td>66</td>
<td>6868</td>
<td>0.34</td>
<td>5889</td>
<td>0.41</td>
</tr>
<tr>
<td>72</td>
<td>7903</td>
<td>0.17</td>
<td>6923</td>
<td>0.22</td>
</tr>
<tr>
<td>78</td>
<td>7900</td>
<td>0.72</td>
<td>6803</td>
<td>0.77</td>
</tr>
<tr>
<td>AVG</td>
<td>4920.75</td>
<td>0.15</td>
<td>4294.67</td>
<td>0.15</td>
</tr>
</tbody>
</table>
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.

**CONCLUSION**

In this chapter, we propose a technique 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 thermal simulation guided procedure, the scheduling algorithm 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 proposed technique.
REFERENCES


KEY TERMS & DEFINITIONS

System-on-Chip Testing: Manufacturing test techniques for systems-on-chip which integrate pre-designed and pre-verified cores into a single chip.

Test Scheduling: Scheduling techniques for manufacturing tests. Usually, test scheduling aims to reduce the test application time through efficiently planning the starting and finishing times of the tests.

Thermal-Aware Testing: Manufacturing test techniques that consider the temperature of the devices under test in order to avoid possible damages.

Test Set Partitioning: A technique that partitions test sets into multiple sub-sequences of test patterns such that the individual test sub-sequences can be separately applied to the core under test. The technique provides possibility to interrupt a test before its completion and can potentially reduce the test application time as it improves the efficiency of test schedule.

Test Set Interleaving: A technique that applies tests for different cores in a manner that the test for a core can be stopped and continued later, and, during the period when the test is stopped, tests for other cores can be applied.

Thermal Modeling: A technique that provides mathematical models to predict the temperature of objects. The thermal model usually considers thermal resistance and thermal capacitance of the object to its surroundings, as well as the heat generated in and removed from the object.

Optimization Heuristics: Optimization algorithms that seek near-optimal solutions at a reasonable computational cost without being able to guarantee either feasibility or optimality.