Power-Constrained System-on-a-Chip Test Scheduling Using a Genetic Algorithm

Haidar M. Harmanani and Hassan A. Salamy
Department of Computer Science and Mathematics
Lebanese American University
Byblos, 1401 2010 Lebanon

Abstract— **This paper presents a new and an efficient approach for the test scheduling problem of core-based systems based on a genetic algorithm. The method minimizes the overall test application time of a SoC through efficient and compact test schedules. The problem is solved using a "sessionless" scheme that minimizes the number of idle test slots. The method can handle SoC test scheduling with and without power constraints. We present experimental results for various SoC examples that demonstrate the effectiveness of our method in short CPU time.**

I. Introduction

Traditional systems were designed using printed circuits boards that contain VLSI chips and the wiring among them. However, advances in modern VLSI technology allows to incorporate a complete system including processors, memories, buses, and interfaces on a single chip using the *system-on-a-chip* methodology. The SoC methodology provides high-performance complex digital systems with reliable interconnects and low cost solution using *cores* [1]. Cores are predesigned and preverified intellectual properties (IP) that are embedded within a chip. Cores maybe soft, firm, or hard.

Core-based designs are usually tested after assembly. A major challenge in testing SoC methodologies is test scheduling that determines the order in which various cores are tested. Test scheduling for SoC, even for a simple SoC, is equivalent to the NP-complete *m-processor open shop scheduling* problem [3] and cannot be approximated in bounded limits. One classical approach to solve the test scheduling problem is by organizing tests for the target cores or modules into test sessions. However, recent techniques in test scheduling initiates tests as soon as possible. These techniques have been labeled as "sessionless" schemes [4]. An effective test scheduling approach must minimize test time while addressing resource conflicts among cores arising from the use of shared Test Access Mechanisms (TAMs) and on-chip BIST engines. Furthermore, SOC in test mode can dissipate up to twice the amount of power they do in normal mode, since cores that do not normally operate in parallel may be tested concurrently to minimize testing time [9]. Power constrained test scheduling is therefore essential in order to limit the amount of concurrency during test application to ensure that the maximum power rating of the SOC is not exceeded.

Genetic algorithms are probabilistic optimization techniques based on the model of natural evolution and solves problems of high complexity. Genetic algorithms use a group of randomly initialized points, a population, in order to non-deterministically search the problem space. The population is modified according to the natural evolution process following a parody of Darwinian principle of the survival of the fittest. Individuals are selected according to their quality to produce *offspring* and to propagate their genetic material into the next generation. The quality of an individual is measured by a fitness function. The process exploits new points in the search space by providing a diversity of the population and avoiding premature convergence to a single local optimum.

A. Related Work

There has been various approaches for the test scheduling problem in core based systems. Sugihara et al. [11] formulated the test scheduling problem for core systems as a combinatorial optimization problem which is solved using a heuristic method. Chakrabarty [3] solved the test scheduling problem for core-based systems using an optimal formulation by mapping the problem to the *m-processor open shop scheduling* problem, an NP-complete problem. The problem is solved using a mixed integer linear programming (MILP) approach. For large instances where the MILP model is infeasible, a heuristic algorithm is used. The method was later extended by Iyengar et al. [7] to include TAM optimization with core level wrapper optimizations. Larsson et al. [8] presented an integrated SoC test framework by analyzing the problem of test access mechanism design along with test scheduling. Flottes et al. [4] presented a heuristic approach for test scheduling for SoC with power constraints. Ravikumar et al. [10] proposed a method to solve SoC test scheduling problem under the power constraints. Huang et al. [5] used a *bin packing-based* method to allocate test resources and schedule test sets in order to achieve optimal concurrent SoC test.

B. Problem Description

This paper presents a new approach to SoC test scheduling based on test time and power constraints. Given a set of cores with corresponding test times and test powers, the objective is to minimize the overall test time by optimally determining the start times for the various cores in the test sets such that the peak power during testing does not exceed a specified value, P_{max}. The method is motivated by the following:

- Test scheduling is necessary to reduce test time.

0-7803-8935-2/05/$20.00 ©2005 IEEE.

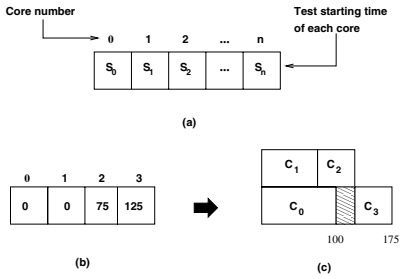

Fig. 1. Chromosome (a) General representation, (b) Example, (c) Corresponding Test schedule

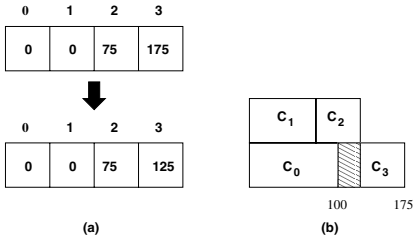

Fig. 2. Chromosome (a) before and after mutation, (b) Solution representation

- Test scheduling is an *intractable* problem. This work proposes an efficient and fast genetic algorithm that initiates test as soon as resource and power constraints allows it.

The remainder of the paper is organized as follows. Section II describes our *genetic test scheduling* formulation along with the chromosomal representation, the genetic operators, and the cost function. The genetic test scheduling algorithm is described in section III while experimental results are presented in section IV.

II. Genetic Core Test Scheduling Formulation

The method starts with a compatibility graph of a set of cores and generates through a sequence of evolution steps a set of compact test schedules. In what follows, we describe our genetic algorithm for core test scheduling.

A. Chromosomal Representation

In order to be able to solve the SoC test scheduling problem, we propose the chromosomal representation that is shown in Figure 1(a). The representation is based on a vector where every gene corresponds to a core with a specific *start time*, S_i. The core *finish time*, F_i, is the start time plus the core test time, T_i; that is, $F_i = T_i + S_i$. The start time changes to the end time of other cores as the solution evolves. Figure 1(b) shows a sample chromosome. The corresponding test schedule is shown in Figure 1(c).

B. Initial Population

The initial population is chosen based on three categories. The first category, which constitutes 30% of the total population, is based on sequential schedules where cores are tested one after the other. The second category, constitutes 40% of the population and is based on a random perturbation of the chromosomes in the first category using the crossover operator. Finally, the last category is generated pseudo-randomly. That is, the algorithm randomly selects a core and then randomly selects a *compatible* core from the compatibility graph in order to guarantee the chromosome feasibility.

C. Selection and Reproduction

Within each generation, individuals in the current population are selected for reproductions using genetic operators and offspring are created. If M is the size of the initial population and $\frac{M}{2}$ is the number of offspring created in each generation, we select M new parents from $M + \frac{M}{2}$ individuals. The algorithm selects during every generation 70% of the best chromosomes while the remaining chromosomes are chosen randomly from the remaining population.

D. Genetic Operators

In order to explore the design space, we use two genetic operators, *mutation* and *crossover* that are applied iteratively with their corresponding probabilities as shown in Figure 3.

1) Mutation: Mutation introduces incremental changes in the offspring by randomly changing allele values of some genes. Thus, we randomly choose a core$_i$ and either change its starting time S_i to the *end time* F_j of a randomly chosen core$_j$ ($i \neq j$) or to 0. The mutation operator is illustrated using the example shown in Figure 2(a). Assume that C_3 is randomly chosen to undergo mutation; then the test start time of C_3 is randomly changed to the test end time of C_1 (C_0 and C_3 are compatible). The resulting chromosome schedules core C_3 at $t = 75$. The test schedule of the resulting chromosome is shown in Figure 2(b).

2) Crossover: Crossover provides a mechanism for the offspring to inherit the characteristics of the parents. We use in this work a *multiple point crossover* that works as follows. We randomly choose two chromosomes from the population and then we choose two random cut points between 0 and $\#Cores - 1$. The crossover operator exchanges next the genes between the two cut points.

E. Objective Function

Given a set of cores $\{C_1, C_2, ..., C_n\}$ with corresponding test times $\{T_1, T_2, ..., T_n\}$ and test powers $\{P_1, P_2, ..., P_n\}$. If the peak power dissipation is estimated as the $\sum_{C_i} P_i$, then the objective function is to minimize the overall test time by optimally determining the start times for the cores in the test sets such that the peak power during testing is not exceeded.

III. Genetic Test Scheduling Algorithm

Each chromosome represents an intermediate test core schedule that has a different cost. During every generation, chromosomes are selected for reproduction, resulting in new test schedules. The algorithm must ensure that 1) Selected cores in a schedule that are tested concurrently are compatible; and 2) The tests of concurrent cores does not exceed the allowed power. Two test sets are conflicting if (i) they share resources such as an external bus for example; (ii) they share

```
Genetic_TestScheduling()
{
   M = Population size.
   N0 = Population size / 2
   Ng = Number of generations
   Read the SoC blocks to be test scheduled
   Read the system's power and test compatibility constraints
   Get the population size and the nb. of generations (Ng)
   Generate an initial population, current_pop
   evaluate(current_pop)
   for i = 0 to Ng do
   {
     for j = 0 to N0 do
         Select two chromosomes from current_pop for mating
         apply crossover with probability Pxover
     for k = 0 to N0 do
         Select a chromosome from current_pop
         apply mutation with probability Pm
     Evaluate the population fitness.
     new_pop = select(current_pop, offspring)
     current_pop ← new_pop
   }
}
```

Fig. 3. Genetic SoC test scheduling algorithm

BIST test set for cores that share a BIST resource or they are the external and BIST components of a core's test set. Cores compatibility is solved using a compatibility graph where nodes represent tests while edges indicate compatibility among nodes. The algorithm, shown in Figure 3, starts by randomly selecting an initial population of chromosomes. During every generation, test start times within genes are evolved through a sequence of genetic operators. The genetic operators randomly set the test start time of a selected core to the finish time of a randomly chosen core. The reproduction process replaces half of the population and the best chromosomes are maintained for the next generation. The algorithm repeats the above process for the maximum set number of generations.

IV. Experimental Results

We have implemented the proposed algorithm using the Java language on a Pentium III with 450 Mhz clock and 32 MB RAM. We tested our method using various examples from the literature and compare our work to Flottes et al. [4], Chakrabarty [3], and Muresan et al. [9].

A. Parameters

Various GA parameters are important in achieving good results. We have experimentally determined that for the attempted problems, a population size of 200 and a generation number of 250 were sufficient to achieve good solutions. We have also experimentally determined the crossover probability, P_{xover}, to be 0.35 and the mutation probability, P_m, to be 0.65.

B. Test Scheduling Without Power Consideration

The first example that we scheduled is the *System S* example that was initially reported by Chakrabarty [2] and whose data are shown in Table I. Our system found the optimal test schedule for this example, that is a total testing time of $1,152,180$ cycles compared to $1,204,630$ in [2]. We show our optimal schedule for this example in Figure 4. We assume in this example that each core has its own dedicated BIST.

Core/Number	External Test Time	BIST Test Time
C880/1	377	4096
C2670/2	15958	64000
C7552/3	8448	64000
S953/4	28959	217140
S53785	60698	389214
S1196/6	778	135200

TABLE I
Test Data for the cores in System S

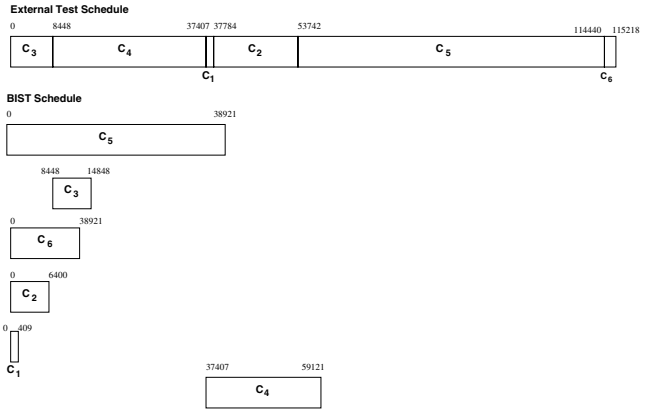

Fig. 4. Test schedule for *System S* where each core has a dedicated BIST

The *System S* example is next modified by adding an additional core, C_7. Core C_7 is tested entirely using BIST while cores C_3, C_4, C_5, C_7 share BIST resources. The optimal test schedule that was reached by our system is shown in Figure 5 where the total testing time is equal to $1,182,350$ cycles and that was generated in 7.21 CPU seconds. On the other hand, the shortest task algorithm in [3] was not able to reach the optimal solution and produced a schedule of a total testing time equals to $1,213,330$ cycles.

The next example we attempted is the *d5018* which consists of eight ISCAS benchmark cores. The example was solved in [3] using the shortest-task-first resulting in $7,851$ cycles. The total testing time obtained using our algorithm is $6,809$

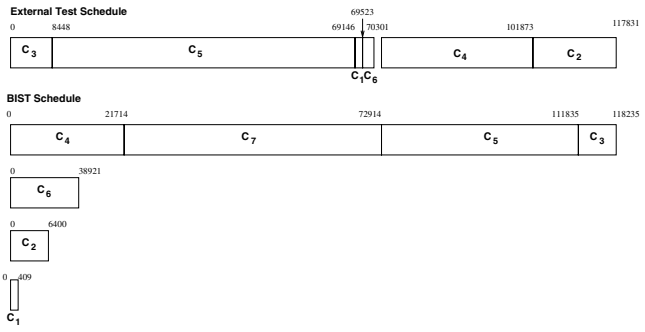

Fig. 5. Test schedule for *System* S with seven cores

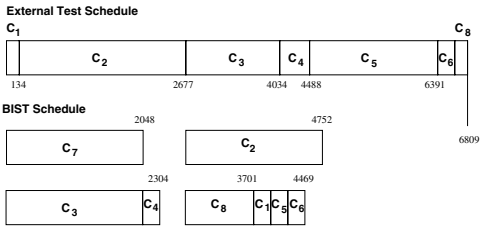

Fig. 6. Test schedule for *d5018*

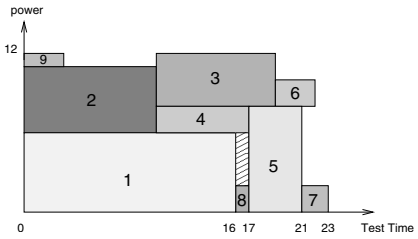

Fig. 7. Power constrained test schedule for the SoC example in [4]

cycles with the schedule shown in Figure 6. The example was scheduled in 8.11 CPU seconds.

C. Power-Constrained Test Scheduling

We next run the algorithm by considering power constraints and test schedule the example reported by Muresan et al. [9] who improved the "unequal length session approach" by allowing several cores to be sequentially tested within a session. The example was later attempted by [4] with a power constraints of $P_{max} = 12$ units. Muresan et al. [9] test scheduled the example in $31,000,000$ clock cycles. We did run this example using our genetic algorithm based on a "sessionless" test schedule resulting with a test schedule of $23,000,000$ clock cycles, which is the optimal test schedule. The optimal test schedule based on our method is shown in Figure 7 and was scheduled in 3.8 CPU seconds. We also attempted the simple example reported in [9]. For a power limit of $P_{max} = 12$ units the test schedule proposed by [9] leads to a total testing time of 29 cycles, while in our "sessionless" approach we reach a schedule of total test time of 25 cycles in 2.273 CPU seconds and is shown in Figure 8.

We next consider the SoC example presented in [4] that includes $fourteen$ cores with a power limit of $P_{max} = 30$. The total testing time obtained in [4] for the above example is $52,000$. Solving this example using our genetic algorithm leads to the schedule in Figure 9 in 32.5 CPU seconds for a total testing time of $46,000$.

Finally, we consider again the *d5018* SoC example. Iyengar et al. [6] solved this problem with power constraint using an MILP formulation. They report a total test time of $7,985$ cycles for a power constraint of $P_{max} = 950$. We solve the same problem assuming the same resource sharing and power constraints. The power constrained test schedule for *d5018* using our genetic approach is shown in Figure 10 where the total testing time is $6,809$ cycles in 14.28 CPU seconds compared to $7,985$ cycles obtained in [6].

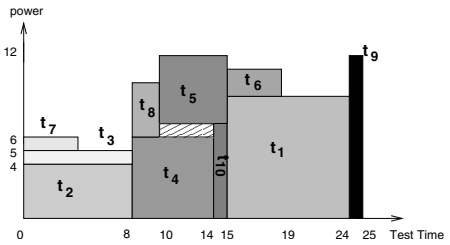

Fig. 8. Power constrained sessionlesss test schedule for SoC example in [9]

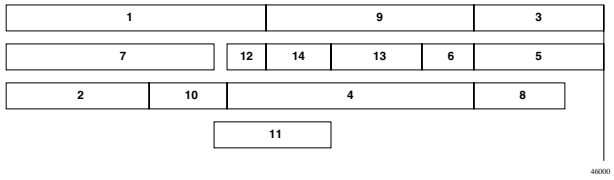

Fig. 9. Power constrained test schedule for the SoC example in [4]

REFERENCES

[1] M. Bushnell and V. Agrawal. *Essentials of Electronic Testing for Digital, Memory & Mixed-Signal VLSI Circuits*. Kluwer-Academic Publishers, 2000.

[2] K. Chakrabarty. Test Scheduling for Core-Based Systems. In *In Proc. ICCAD*, pages 391–394, San Jose, CA, 1999.

[3] K. Chakrabarty. Test Scheduling for Core-Based Systems Using Mixed-Integer Linear Programming. *IEEE Trans. on CAD*, 19(10):1163–1174, 2000.

[4] M-L. Flottes, J. Pouget, and B. Rouzeyre. Power-Constrained Test Scheduling for SoCs Under a no Session. In *Proc. SOC*, pages 401–412, 2001.

[5] Y. Huang, W.-T. Cheng, C.-C Tsai, N. Mukherjee, O. Samman, Y. Zaidan, and S. Reddy. Resource Allocation and Test Scheduling for Concurrent Test of Core-Based SoC Design. In *Proc. ATS*, pages 265–270, 2001.

[6] V. Iyengar and K. Chakrabarty. System-on-a-Chip Test Scheduling Precedence Relationships, Preemptive, and Power-Constraints. *IEEE Trans. on CAD*, 21(9):1088–1094, September 2002.

[7] V. Iyengar, K. Chakrabarty, and E. Marinissen. Test Wrapper and Test Access Mechanism Co-Optimization for System-on-a-Chip. In *Proc. ITC*, pages 1023–1032, Baltimore, 2001.

[8] E. Larsson and Z. Peng. An integrated system-on-chip test framework. In *Proc. DATE*, pages 138–144, March 2001.

[9] V. Muresan, X. Wang, V. Muresan, and M. Vladutiu. A Comparison of Classical Scheduling Approaches in Power-Constrained Block-Test Scheduling. In *Proc. ITC*, pages 882–891, 2000.

[10] C. Ravikumar, G. Chandra, and A. Verma. Simultaneous Module Selection and Scheduling for Power Constrained Testing of Core Based Systems. In *Proc. VLSI Design*. IEEE, 2000.

[11] M. Sugihara, H. Date, and H. Yasuura. A Novel Test Methodology for Core-Based System LSIs and a Testing Time Minimization Problem. In *Proc. ITC*, pages 465–472, Oct. 1998.

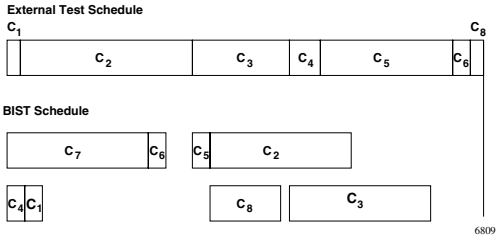

Fig. 10. Power-Constraint Test Schedule for d5018