Journal of Circuits, Systems, and Computers
Vol. 15, No. 3 (2006) 331–349
© World Scientific Publishing Company

World Scientific
www.worldscientific.com

POWER-CONSTRAINED SYSTEM-ON-A-CHIP TEST SCHEDULING USING A GENETIC ALGORITHM

HAIDAR M. HARMANANI and HASSAN A. SALAMY
Computer Science and Mathematics Division,
Lebanese American University,
Byblos 1401 2010, Lebanon

Revised 10 January 2006

This paper presents 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 system-on-a-chip 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. The method achieved optimal test schedules in all attempted cases in a short CPU time.

Keywords: Core-based systems; embedded core testing; genetic algorithms.

J CIRCUIT SYST COMP 2006.15:331-349. Downloaded from www.worldscientific.com
by Harman Haman on 11/04/14. For personal use only.

1. Introduction

Traditional systems were designed using printed circuits boards that contain VLSI chips and the wiring among them. However, advances in modern VLSI technology allow to incorporate a complete system including processors, memories, buses, and interfaces on a single chip using the *system-on-chip* (SOC) methodology. The SOC methodology provides high-performance complex digital systems with reliable interconnects and a 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. A *soft* core consists of a synthesizable HDL description that can be retargeted to different semiconductor processes while a *firm* core contains more structure. Finally, a *hard* core includes layout and technology-dependent timing information and is ready to be dropped in a system.

It is widely recognized that testing embedded cores is a major bottleneck. Core-based designs are usually tested after assembly, at the end of the system implementation. Different DFT and test issues exist if the embedded core is soft,

firm, or hard. Vendors supply the core and its tests while the user provides the test access to the core.[1] Typically, the core test problem is solved by surrounding a hard core with test logic, known as a *test-wrapper* or a *test collar*. Each input and output terminal of a core, provides functions for a *normal* mode, *external test* mode, and an *internal test* mode. During external test mode, the wrapper element drives the host chip in order to test the interconnect while during internal test mode the wrapper element tests the core by observing the core output.[1] Usually, a combination of BIST and external testing must be used to achieve a high-fault coverage.

A major challenge in testing embedded cores is test scheduling which 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.[2,3] An effective test scheduling approach must minimize the test time while addressing resource conflicts among cores arising from the use of shared Test Access Mechanisms (TAMs), on-chip BIST engines and power dissipation constraints. Obviously, the minimal test time would be achieved by maximizing the simultaneous test of all individual functions or cores; however, design constraints may prevent this full parallelism. For example, power consumption is an important factor that may impact the test parallelism. Power dissipation during testing is a function of time and depends on the switching activity resulting from the application of test vectors to the system.[4] The 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.[5] 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.

J CIRCUIT SYST COMP 2006.15:331-349. Downloaded from www.worldscientific.com
by Harman Haman on 11/04/14. For personal use only.

1.1. *Related work*

There has been various approaches for the test scheduling problem in core-based systems. Craig *et al.*[6] solved the general test scheduling problem by representing the circuit under test as a directed graph. A node in the graph represents a group of flip-flops that forms a BILBO while the edges represent the combinational logic among each group of flip-flops. This graph is referred to as a register adjacency graph. Graph coloring or clique partitioning algorithms can be used to obtain the minimum number of test sessions. Sugihara *et al.*[7] formulated the test scheduling problem for core systems as a combinatorial optimization problem which is solved using a heuristic method. The authors make two restrictive assumptions: (1) Every core has its own BIST logic and (2) external testing can be carried out for only one core at a time. That is, there is only one test access bus at the system level. 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 finish time of the schedule is the

latest completion time of the individual processor schedule while the length of the job is taken as the amount of time for test to execute. The problem is solved by minimizing the test finish time using a mixed integer linear programming (MILP) approach. For large instances where the MILP model is infeasible, the authors use a heuristic algorithm. The method was later extended by Iyengar *et al.*[8] to include TAM optimization with core level wrapper optimizations. Larsson *et al.*[9] analyzed test scheduling with power and test resource constraints where an integrated SOC test framework is presented by analyzing the problem of test access mechanism design along with test scheduling. Flottes *et al.*[10] presented a heuristic approach for test scheduling for SOC with power constraints. The advantage of the method is that it can handle large size problems in short time. Ravikumar *et al.*[11] proposed a method to solve SOC test scheduling problem under power constraints. Huang *et al.*[12] used a *bin packing-based* method to allocate test resources and to schedule test sets in order to achieve optimal concurrent SOC test. The objective is to minimize test application time for different TAMs under the constraint of peak power consumption.

J CIRCUIT SYST COMP 2006.15:331-349. Downloaded from www.worldscientific.com
by Harman Haman on 11/04/14. For personal use only.

1.2. *Problem description*

This paper presents an efficient approach to SOC test scheduling based on test time and power constraints. Given a set of cores $\{C_1, C_2, \ldots, C_n\}$ with corresponding test times $\{T_1, T_2, \ldots, T_n\}$ and test powers $\{P_1, P_2, \ldots, P_n\}$, the problem we address in this paper 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 based on an efficient genetic algorithm and is motivated by the following:

- Test scheduling is necessary to reduce test time which is important to reduce design and test cost.
- Test scheduling is an *intractable* problem. This work is based on an efficient and fast genetic algorithm that initiates test as soon as resource and power constraints allow it.

We assume that the test access architecture has been determined, and the cores have been assigned to test buses. No restrictions are placed either on the sharing of BIST logic among cores or on the use of multiple test buses for external testing.

The remainder of the paper is organized as follows. Section 2 introduces the *test scheduling problem* while Sec. 3 formulates the *genetic SOC test scheduling problem* and describes the chromosomal representation, the genetic operators, and the cost function. The *genetic test scheduling algorithm* is described in Sec. 4 while *experimental results* are presented in Sec. 5. Finally, we *conclude* with remarks in Sec. 6.

J CIRCUIT SYST COMP 2006.15:331-349. Downloaded from www.worldscientific.com
by Harman Haman on 11/04/14. For personal use only.

2. Test Scheduling

The test scheduling problem is a combinatorial optimization problem that has been reduced to *bin-packing*, *m-processor open shop scheduling* and *graph coloring*, which cannot be approximated in bounded limits. One classical approach to solve the test scheduling problem is by organizing tests for the target cores into *test sessions*. A test session brings together the tests of compatible modules. This compatibility is checked with respect to the test resource sharing needs. Individual tests may be conflicting because:

(1) They share common test resources such as a test bus or a test response compactor.
(2) The power consumption during simultaneous testing exceeds the device power allowance.

Sessions-based test scheduling techniques assume either *equal length test sessions* or *unequal length test sessions*. During "equal length test sessions" test scheduling, cores are arranged into sessions where the length of each session is set to the longest test time in all sessions. On the other hand, in "unequal length test sessions" test scheduling, cores are arranged into sessions where the length of a specific session is the time taken to test the core requiring the longest time in that session. Recent techniques in test scheduling arrange cores for testing without sessions where a test is initiated as soon as possible and at any time if the resource sharing and power constraints are not violated. These techniques, labeled as "sessionless", partition testing with run to completion.[9,10]

In order to illustrate the test scheduling problem, we use the example in Fig. 1 that shows four cores $C_i(p_i, t_i)$, C_0 (100, 100), C_1 (75, 75), C_2 (75, 50), and

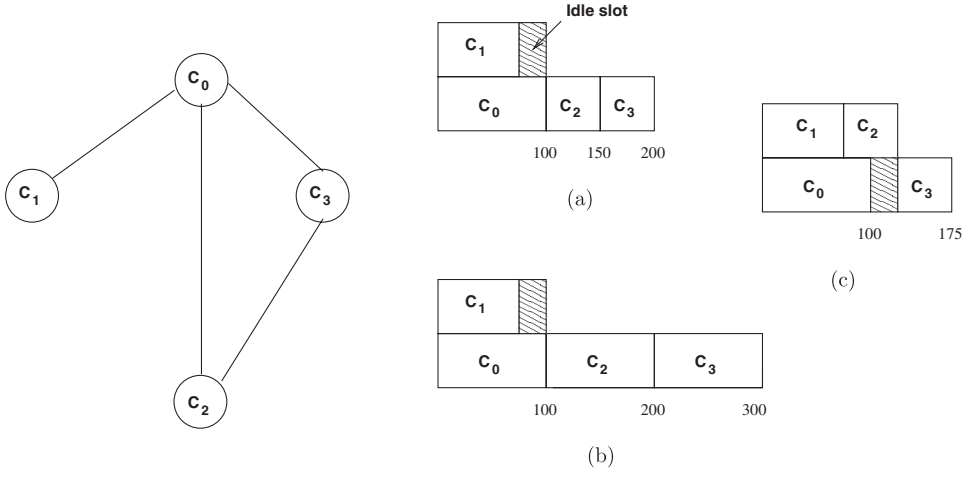

Fig. 1. Compatibility graph and solution representations.

J CIRCUIT SYST COMP 2006.15:331-349. Downloaded from www.worldscientific.com
by Harman Haman on 11/04/14. For personal use only.

C_3 (125, 50) with a peak power constraint of $P_{\max} = 175$. Figure 1 also shows the compatibility relationship among the above four cores where an edge between two cores indicates that the two cores can be tested simultaneously. The "unequal length test session" divides the cores into three sessions as shown in Fig. 1(a) where the total testing time based on this approach is 200. The "equal length test session" schedules the tests of the four cores into three sessions as shown in Fig. 1(b) where the total testing time for a session is equal to the longest test time among all cores in the session and the total testing time of the system is the sum of testing time of all sessions, that is 300. Finally, the "sessionless" scheme schedules the cores with a total test time of 175 as shown in Fig. 1(c).

3. Genetic Core Test Scheduling Formulation

The genetic test scheduling method starts with a compatibility graph of a set of cores and generates through a sequence of evolution steps a set of compact and optimal test schedules. In what follows, we describe our genetic algorithm for core test scheduling in reference to Fig. 1.

3.1. *Chromosomal representation*

In order to solve the SOC test scheduling problem, we propose the chromosomal representation shown in Fig. 2(a). The representation is based on a vector where every gene corresponds to a core with a specific *test start time*, S_i. The core *test finish time*, F_i, is equal to the test start time plus the core test time, T_i, that is, $F_i = T_i + S_i$. Note that the test start time, S_i, is not constant and it changes to

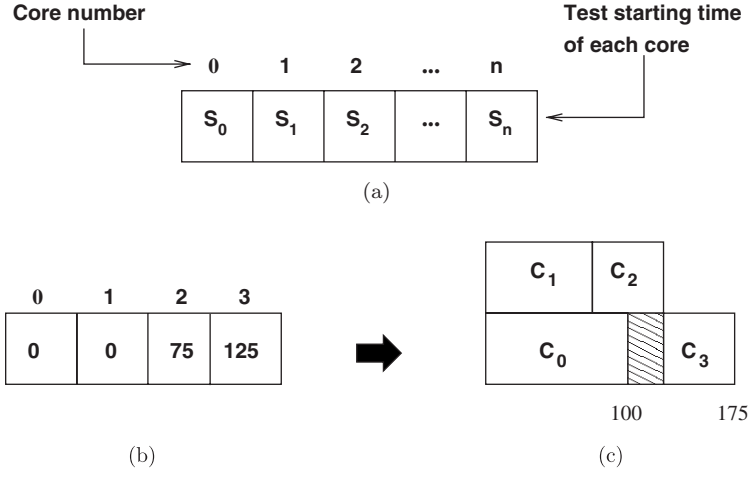

Fig. 2. (a) Chromosome representation, (b) Sample chromosome and (c) Corresponding test schedule.

the end times of other cores as the solution evolves. Figure 2(b) shows a sample chromosome using the compatibility graph shown in Fig. 1. The corresponding test schedule is shown in Fig. 2(c).

3.2. *Initial population*

The quality of the final solution is affected by how the initial population has been constructed. The initial population is chosen based on three categories. The first category, which constitutes 30% of the total population, is based on the worst possible schedules; that is serial schedules where cores' tests are serialized. The order of the cores in the serial schedules is random. 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. Thus, the algorithm randomly selects a core and then randomly picks a *compatible* core from the compatibility graph in order to guarantee the chromosome feasibility.

J CIRCUIT SYST COMP 2006.15:331-349. Downloaded from www.worldscientific.com
by Harman Haman on 11/04/14. For personal use only.

3.3. *Selection and reproduction*

Within each generation, individuals are selected for reproductions using the genetic operators. In order to keep the population size fixed, we select a constant number of individuals from the current population and from the generated offspring. Thus, if M is the size of the initial population and $M/2$ is the number of offspring created in each generation, we select M new parents from $M + (M/2)$ individuals. The new generation is selected using a simple selection procedure that ensures a mix of "good" and "bad" chromosomes. Thus, during every generation the algorithm selects 70% of the best chromosomes while the remaining chromosomes are chosen randomly from the remaining population.

3.4. *Genetic operators*

In order to explore the decision space, we use two genetic operators, *mutation* and *crossover* that are applied iteratively with their corresponding probabilities. The genetic operators are followed with a deterministic compaction operation, *Fill Gap*, that compacts the chromosomes by filling the gaps or the idle slots among the genes. The algorithm for the *Fill Gap* operation is shown in Fig. 3.

3.4.1. *Mutation*

Mutation is an important operator that introduces incremental changes in the offspring by randomly changing allele values of some genes. The algorithm randomly chooses a core_i and changes its test starting time S_i to the *test end time* F_j of another randomly chosen core_j $(i \neq j)$ or to 0.

```
Fill_Gap(Chromosome V_old, Chromosome V_new)
{
  ∀ cores in V_old
    i ← 0
    core(t_i, f_i) ← V[i]
    List_1 = All cores in V_new and V_old whose start_time > t_i in increasing order
    List_2 = All cores in V_new and V_old such that: start_time ≤ t_i < end_time, in
             increasing order
    if List_2 is empty and List_1 is not empty
      subtract f_i − t_i from all nodes in List_1
    else (if List_2 ≠ φ and List_1 ≠ φ) {
      max_1 = Largest end_time in List_2
      for(all nodes u ∈ List_1)
        max_2 = Largest end_times in List_2 ≤ start_time (u)
        if (max_2 == 0)
          max_2 = t_i
        if start-time(u) < max_1)
          if (f_i − max_2 > 0)
            start_time(u) = start_time(u) − (f_i − max_2)
          else
            if start_time(u) > max_1)
              if (f_i − max_1 > 0)
                start_time(u) = start_time(u) − (f_i − max_1)
              else
                max_3 = Largest end time in List_1 and List_2 <= start_time(u)
                start_time(u) = start_time(u)−max_3
  V_new ← core(t_i, f_i); i ← i + 1
}
```

Fig. 3. Fill gap operation pseudo-code.

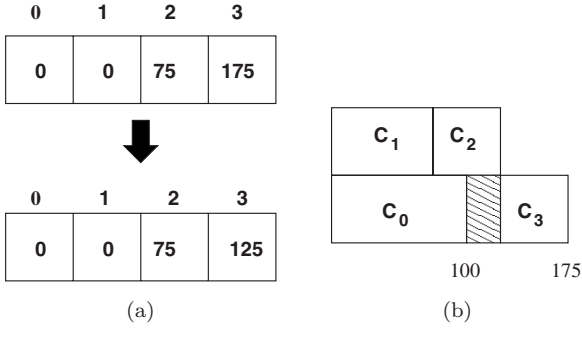

Fig. 4. Chromosome (a) Before and after mutation and (b) Solution graphical representation.

In order to illustrate the mutation operator, consider the example shown in Fig. 4(a). Assume that C_3 is randomly chosen to undergo mutation and assume that the test start time of C_3 is randomly changed to the test end time of C_2. The chromosome after mutation schedules core C_3 at $t = 125$. The graphical representation of the schedule after mutation is shown in Fig. 4(b).

J CIRCUIT SYST COMP 2006.15:331-349. Downloaded from www.worldscientific.com
by Harman Haman on 11/04/14. For personal use only.

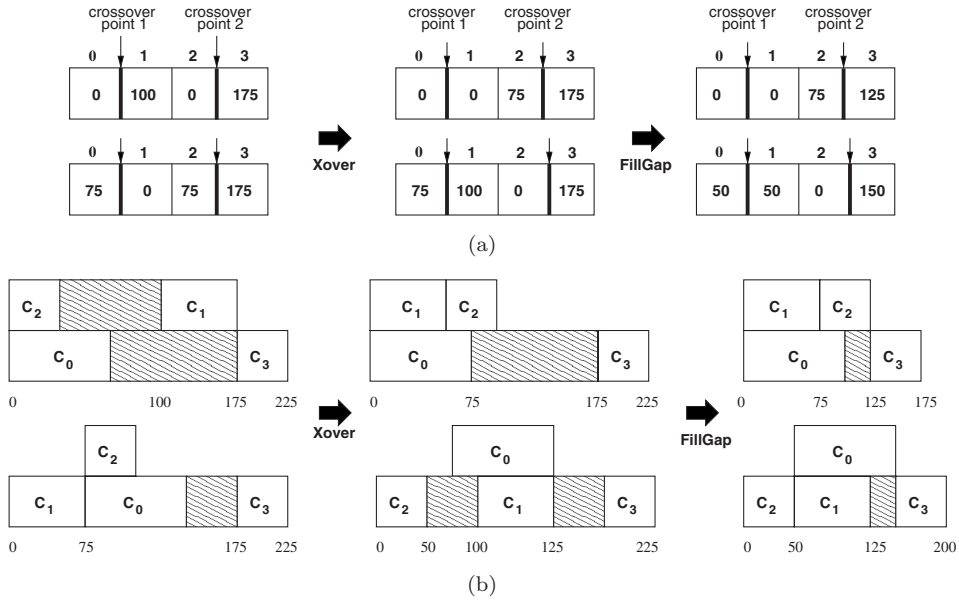

Fig. 5. (a) Parents chromosomes before and after crossover and (b) Corresponding test schedules.

J CIRCUIT SYST COMP 2006.15:331-349. Downloaded from www.worldscientific.com
by Harman Haman on 11/04/14. For personal use only.

3.4.2. *Crossover*

Crossover is the main genetic operator as it provides a mechanism for the offspring to inherit the characteristics of the parents. We use a *two-point crossover* that randomly chooses two chromosomes that are split into three segments of contiguous genes. The offspring are created by taking alternative segments from the two parents. To illustrate how the crossover operator works, consider the chromosomes in Fig. 5(a). The algorithm selects two random cut points as shown and the genes between the cut points are exchanged. The test schedules corresponding to the offspring are shown in Fig. 5(b).

3.5. *Objective function*

The fitness of an individual is crucial for the transmission of its gene information to the next generation. Given a set of cores $\{C_1, C_2, \ldots, C_n\}$ with corresponding test times $\{T_1, T_2, \ldots, T_n\}$ and test powers $\{P_1, P_2, \ldots, P_n\}$. 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 various cores in the test sets and such that the peak power is not exceeded during testing.

4. 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 the following:

(1) Selected cores that are tested concurrently are compatible.
(2) The peak power of cores that are tested concurrently does not exceed the peak power.

Two test sets are conflicting if (i) they share resources such as an external bus for example; (ii) they share 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 Fig. 6, starts by randomly selecting an initial population of chromosomes. During every generation, tests 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. In order to obtain meaningful and feasible schedules, the mutation operator uses a constructive approach that minimizes the generation of unfeasible test schedules. However, unfeasible solutions that result from the crossover operator are punished with a prohibitive cost. The reproduction process replaces half of the population

J CIRCUIT SYST COMP 2006.15:331-349. Downloaded from www.worldscientific.com
by Harman Haman on 11/04/14. For personal use only.

```
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 number of generations (Ng)
   Generate an initial population, current_pop
   for i = 1 to M
     evaluate(current_pop)
   Keep_the_best()
   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
     Apply Fill_Gap() to offspring
     Evaluate the population fitness.
     new_pop = select(current_pop, offspring)
     current_pop ← new_pop
   }
}
```

Fig. 6. Genetic SOC test scheduling algorithm.

and the best chromosomes are maintained for the next generation. The algorithm repeats the above process for the maximum number of generations.

5. Experimental Results

5.1. *Parameters*

Various GA parameters are important in achieving good results. Given a sufficient population size and number of generations, a good and a suboptimal test schedule can be found, however execution time is directly proportional to both parameters. We have experimentally determined that for the problems we attempted, a population size of 150 and a generation number of 200 were sufficient to achieve a good solution. We have also determined experimentally the crossover probability, P_{xover}, to be 0.35 and the mutation probability, P_m, to be 0.65.

5.2. *Benchmark results*

The proposed algorithm was implemented using the Java language on a Pentium Centrino with 2.13 GHz clock and 1 GB of RAM. The method was tested on various benchmark examples from the literature and we compare our work to Refs. 3, 5, 10 and 13. Detailed results comparisons are shown in Tables 7 and 8.

5.2.1. *System* $\mathcal{S}$

The *System* $\mathcal{S}$ SOC example was initially reported by Chakrabarty[4] and used later by others for comparison purposes. The data for this example are shown in Table 1. Our system found the optimal test schedule, shown in Fig. 7, with a total testing time of 1 152 180 cycles compared to 1 204 630 in Ref. 4. We assume in this example that each core has its own dedicated BIST.

The *System* $\mathcal{S}$ SOC 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, and C_7 share BIST resources. The optimal test schedule that was achieved by our system is shown in Fig. 8 with a total testing time of 1 182 350 cycles. On the other hand, the shortest task algorithm in Ref. 3 was not able to reach the optimal solution and it results in a schedule with a total testing time of 1 213 330 cycles. Figure 9 shows the speed of

Table 1. Test data for the cores in system $\mathcal{S}$.

Core/number	External test time	BIST test time
C880/1	377	4096
C2670/2	15 958	64 000
C7552/3	8448	64 000
S953/4	28 959	217 140
S53785	60 698	389 214
S1196/6	778	135 200

J CIRCUIT SYST COMP 2006.15:331-349. Downloaded from www.worldscientific.com
by Harman Haman on 11/04/14. For personal use only.

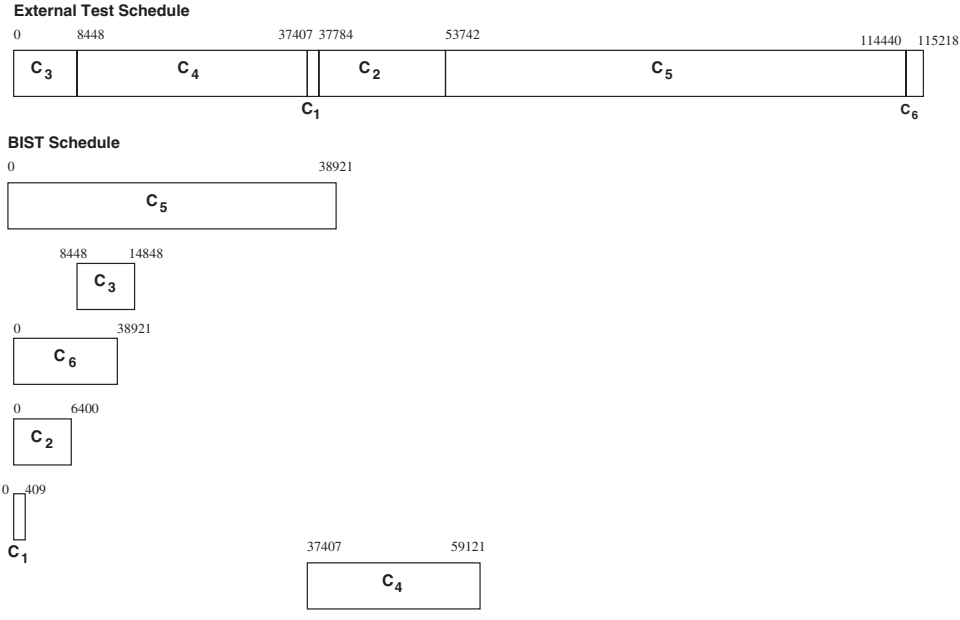

Fig. 7. Test schedule for *System* $\mathcal{S}$ where each core has a dedicated BIST.

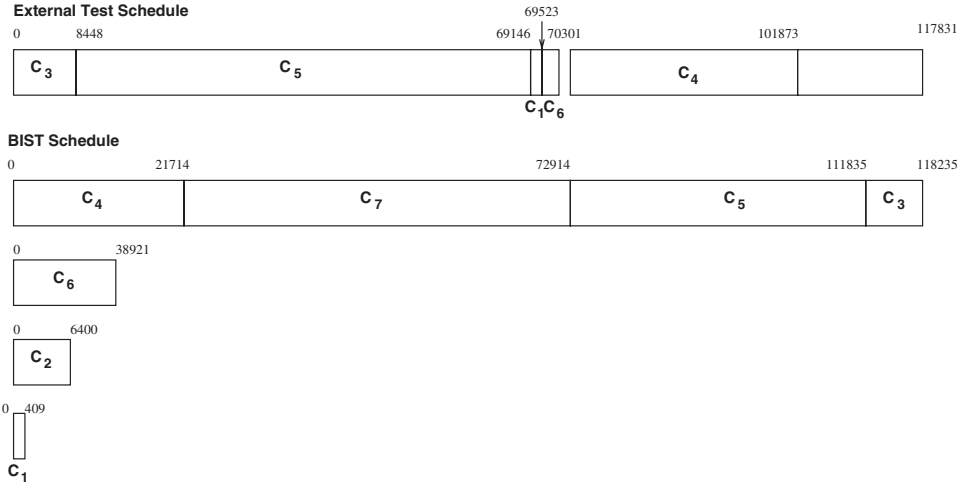

Fig. 8. Test schedule for *System* S with seven cores.

J CIRCUIT SYST COMP 2006.15:331-349. Downloaded from www.worldscientific.com
by Harman Haman on 11/04/14. For personal use only.

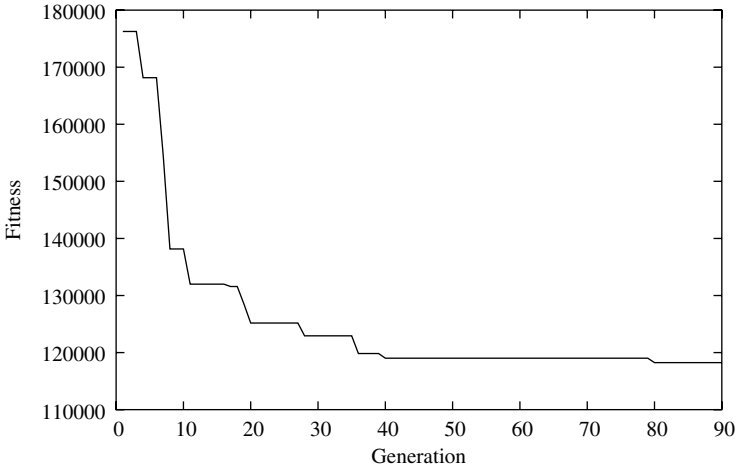

Fig. 9. Curve depicts the fitness of the best chromosomes in a population of 150 for the SOC example used in *System* $\mathcal{S}$.

J CIRCUIT SYST COMP 2006.15:331-349. Downloaded from www.worldscientific.com
by Harman Haman on 11/04/14. For personal use only.

convergence for our algorithm in the case of the *System* $\mathcal{S}$ example. The example was scheduled in 0.901 CPU seconds.

5.2.2. *d5018 system-on-chip*

The d5018 system, whose details are shown in Table 2, is an example SOC that consists of eight ISCAS benchmark cores. The example was solved in Ref. 3 using the shortest-task-first. While the total testing time obtained using the shortest-task-first is 7851 cycles, the total testing time obtained using our approach is 6809 cycles. The schedule for the *d5018* example obtained by our approach is shown in Fig. 10 and was test scheduled in 1.114 seconds.

The *d5018* SOC example is next test scheduled by considering power constraints. Iyengar *et al.*[14] test scheduled this example using an MILP formulation in 7985 cycles with a power constraint of $P_{\max} = 950$. The power-constrained test schedule

Table 2. Test data for the d5018 system.

Core	BIST test time	External test time	Power in BIST mode (mW)
1	256	134	54
2	2048	2543	159
3	2048	1357	453
4	256	454	57
5	256	1903	324
6	256	242	72
7	2048	—	792
8	1024	176	75

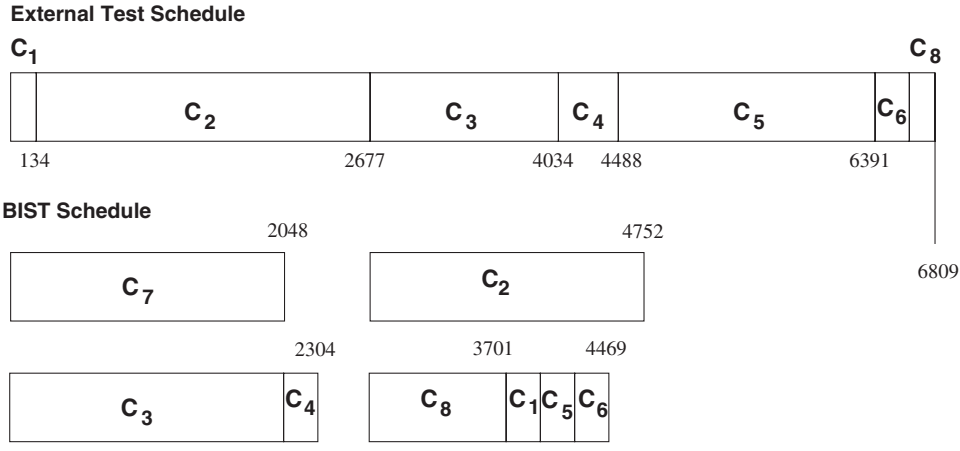

Fig. 10. Test schedule for d5018.

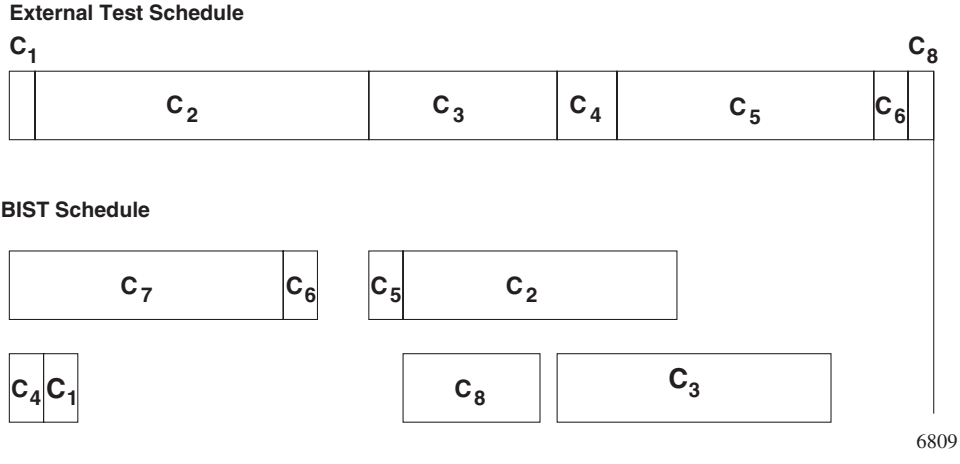

Fig. 11. Power-constrained test Schedule for d5018.

J CIRCUIT SYST COMP 2006.15:331-349. Downloaded from www.worldscientific.com
by Harman Haman on 11/04/14. For personal use only.

for *d5018* using our approach is shown in Fig. 11 where the total testing time is 6809 cycles. The example was test scheduled in 1.785 CPU seconds.

5.2.3. *Muresan system-on-chips*

The Muresan simple example, whose characteristics are shown in Table 3, was first presented by Muresan *et al.*[5] and was test scheduled in 31 000 000 clock cycles. The authors improved the "unequal length session approach" by allowing several cores to be tested sequentially within a session. The example was test scheduled based on our genetic approach using a "sessionless" scheme in 23 000 000 clock cycles, which

Table 3. Test data for SOC example used in Muresan SOC.[5]

Core	P_i	D_i	Share test with
1	6	16 000	6 7 8
2	5	10 000	4 5 6 7
3	4	9000	6 7 9
4	2	7000	5
5	8	4000	2 4 7
6	2	3000	1 2 3
7	2	2000	1 2 3 5
8	2	1000	1
9	1	3000	3

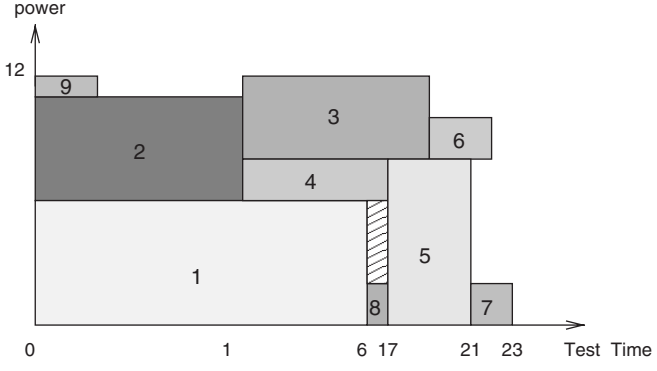

Fig. 12. Power constrained test schedule for the Muresan SOC example.[5]

is the optimal test schedule (Fig. 12). The example was test scheduled in 0.475 CPU seconds.

5.2.4. *Muresan II*

The *Muresan II* SOC example, whose characteristics are shown in Table 4, was early reported in Refs. 5, 13, and 15. We compare our approach with the improved approach used in Ref. 5 for the same example. For a power constraint of $P_{\max} = 12$ units, the test schedule proposed by Muresan *et al.*[5] leads to a total testing time of 29 cycles, while in our "sessionless" approach we reach the schedule shown in Fig. 13 with a total testing time of 25 cycles. The example was test scheduled in 0.284 CPU seconds.

5.2.5. *Flottes system-on-chip*

The *Flottes SOC* example, whose details are shown in Table 5, was first reported in Ref. 10. The example includes *fourteen* cores with a power constraint of $P_{\max} = 30$. The testing time obtained in Ref. 10 for this example is 52 000 while our method lead

J CIRCUIT SYST COMP 2006.15:331-349. Downloaded from www.worldscientific.com
by Harman Haman on 11/04/14. For personal use only.

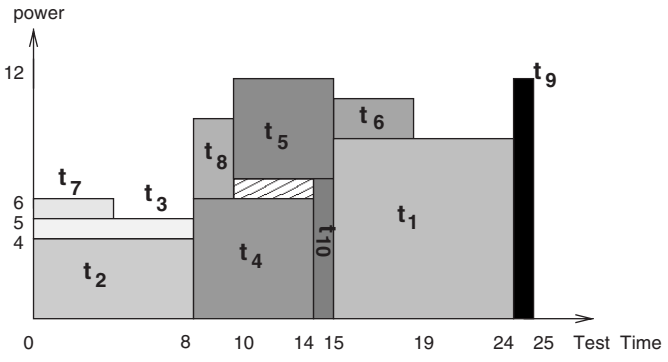

Fig. 13. Test schedule using a sessionless scheme for the Muresan II SOC example.[5]

Table 4. Test data for the Muresan II SOC.[5]

$t_1(9, 9, t_2, t_3, t_5, t_6, t_8, t_9)$	$t_6(2, 4, t_1, t_7, t_8, t_9)$
$t_2(4, 8, t_1, t_3, t_7, t_8)$	$t_7(1, 3, t_2, t_3, t_4, t_6, t_8, t_9)$
$t_3(1, 8, t_1, t_2, t_4, t_7, t_9, t_{10})$	$t_8(4, 2, t_1, t_2, t_4, t_6, t_7, t_9, t_{10})$
$t_4, (6, 6, t_3, t_5, t_7, t_8)$	$t_9(12, 1, t_1, t_3, t_5, t_6, t_7, t_8, t_{10})$
$t_5(5, 5, t_1, t_4, t_9, t_{10})$	$t_{10}(7, 1, t_3, t_5, t_8, t_9)$

Table 5. Test data for Flottes SOC example.[10]

Core	Test length	Power consumption	Share test with
1	20 000	11	3 6 14
2	11 000	9	9 12 13
3	10 000	11	1 9 12
4	19 000	6	7
5	10 000	16	—
6	4000	15	1
7	16 000	10	4 12
8	7000	2	—
9	16 000	9	2 3
10	6000	6	—
11	9000	7	—
12	3000	3	2 3 7
13	7000	15	2
14	5000	8	1

to the the schedule shown in Fig. 14 with a total testing time of 46 000. Figure 15 shows the speed of convergence for our algorithm in this case which was test scheduled in 4.063 CPU seconds.

5.2.6. *ASIC Z*

The *ASIC Z* example, whose details are shown in Table 6, was reported by Zorian.[13] The example consists of four RAM's, two ROM's and three random logic blocks.

J CIRCUIT SYST COMP 2006.15:331-349. Downloaded from www.worldscientific.com
by Harman Haman on 11/04/14. For personal use only.

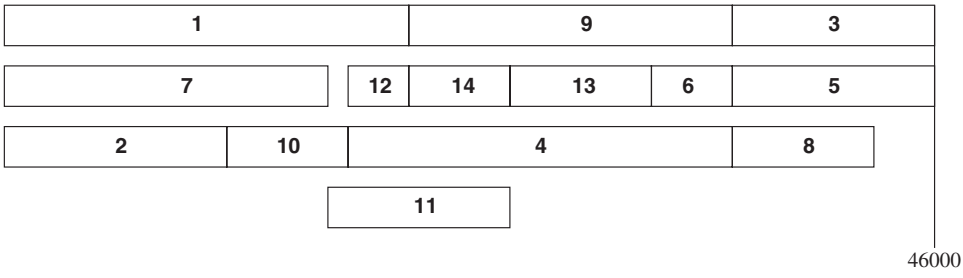

Fig. 14. Test schedule for Flottes SOC example.[10]

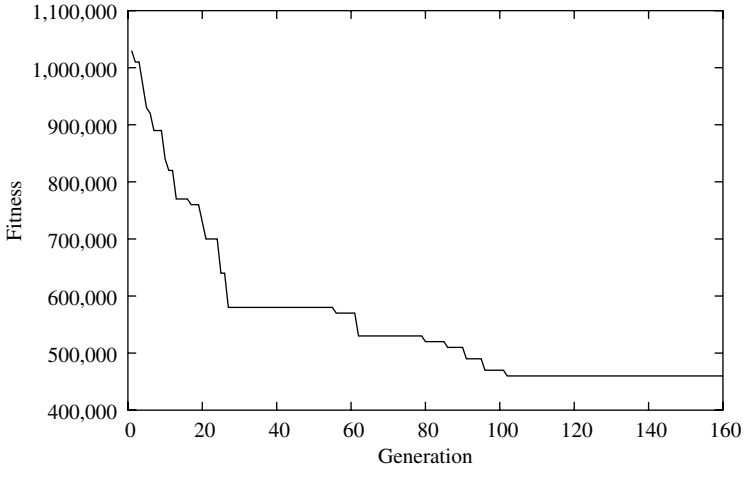

Fig. 15. Curve depicts the fitness of the best chromosomes in a population of 150 for the Flottes SOC example.[10]

Table 6. Test length and power data for ASIC Z system.

Block	Power (mW)	Test length
RL1	295	134
RL2	352	160
RF	95	10
RAM1	282	69
RAM2	241	61
RAM3	213	38
RAM4	96	23
ROM1	279	102
ROM2	279	102

J CIRCUIT SYST COMP 2006.15:331-349. Downloaded from www.worldscientific.com
by Harman Haman on 11/04/14. For personal use only.

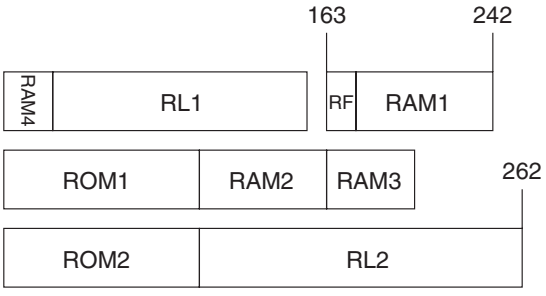

Fig. 16. Power-constrained test schedule for ASIC Z.

J CIRCUIT SYST COMP 2006.15:331-349. Downloaded from www.worldscientific.com
by Harman Haman on 11/04/14. For personal use only.

The maximum power dissipation limit, $P_{\max}$, is 900 mW. The test schedule obtained using our method is shown in Fig. 16 with an optimal test time of 262 cycles. The example was test scheduled in 0.6 CPU seconds.

6. Conclusion

We have presented a fast and efficient method for the test scheduling problem of core-based systems using a "sessionless" scheme. The method is based on a genetic algorithm that explores the decision space in a very short time. The method minimizes the overall test application time of a SOC through efficient and compact test schedules and handles SOC test scheduling with and without power constraints (Tables 7 and 8). We presented experimental results for various SOC examples that demonstrate the effectiveness of our method. The method achieved optimal test schedules in all attempted cases.

Table 7. Test scheduling results without power considerations.

Design	Number of cores	Approach	Test time	Difference to optimum (%)
System $\mathcal{S}$	6	Optimal	1 152 180	—
		Chakrabarty	1 204 630	4.5
		Larsson	1 152 180	0
		Ours	1 152 180	0
System $\mathcal{S}$	7	Optimal	1 182 350	—
		Chakrabarty	1 213 330	2.62
		Ours	1 182 350	0
d5018	8	Optimal	6809	—
		Chakrabarty	7851	15.30
		Ours	6809	0

Table 8. Test scheduling results with power considerations.

Design	Number of cores	Approach	Test time	P_{max}	Difference to optimum (%)
Muresan I	9	Optimal	23 000 000	12	—
		Muresan	31 000 000		34.78
		Flottes *et al.*	23 000 000		0
		Ours	23 000 000		0
Muresan II	10	Optimal	25	12	—
		Muresan	29		16
		Larsson	26–28		4–12
		Flottes *et al.*	25		0
		Ours	25		0
Flottes	14	Optimal	46 000	30	—
		Flottes *et al.*	52 000		13
		Ours	46 000		0
d5018	8	Optimal	6809	950	—
		Iyengar	7985		17.27
		Flottes *et al.*	6809		0
		Ours	6809		0
ASIC Z (1)	9	Optimal	262	900	—
		Larsson	262		0
		Ours	262		0
ASIC Z (2)	9	Optimal	300	900	—
		Zorian	392		23
		Chou *et al.*	331		9
		Larsson	300		0
		Ours	300		0

J CIRCUIT SYST COMP 2006.15:331-349. Downloaded from www.worldscientific.com
by Harman Haman on 11/04/14. For personal use only.

References

1. M. Bushnell and V. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits* (Kluwer-Academic Publishers, Dordrecht, 2000).
2. T. Gonzales and S. Sahni, Open shop scheduling to minimize finish time, *J. ACM* **23** (1976) 665–679.
3. K. Chakrabarty, Test scheduling for core-based systems using mixed-integer linear programming, *IEEE Trans. Computer-Aided Design* **19** (2000) 1163–1174.
4. K. Chakrabarty, Test scheduling for core-based systems, *Proc. Int. Conf. Computer-Aided Design (ICCAD)* (1999), pp. 391–394.
5. V. Muresan, X. Wang, V. Muresan and M. Vladutiu, A comparison of classical scheduling approaches in power-constrained block-test scheduling, *Proc. Int. Test Conf.* (2000), pp. 882–891.
6. G. L. Craig, C. R. Kime and K. K. Saluja, Test scheduling and control for VLSI built-in self-test, *IEEE Trans. Comput.* **37** (1988) 1099–1109.
7. M. Sugihara, H. Date and H. Yasuura, A novel test methodology for core-based system LSIs and a testing time minimization problem, *Proc. Int. Test Conf.* (1998), pp. 465–472.
8. V. Iyengar, K. Chakrabarty and E. Marinissen, Test wrapper and test access mechanism co-optimization for system-on-a-chip, *Proc. Int. Test Conf.* (2001), pp. 1023–1032.

9. E. Larsson and Z. Peng, An integrated system-on-chip test framework, *Proc. Design Automation Test Europe* (2001), pp. 138–144.
10. M.-L. Flottes, J. Pouget and B. Rouzeyre, Power-constrained test scheduling for SoCs under a no session, *SoC Design Methodologies*, eds. M. Robert, B. Rouzeyre, C. Piguet and M.-L. Flottes (Springer, 2002), pp. 401–412.
11. C. Ravikumar, G. Chandra and A. Verma, Simultaneous module selection and scheduling for power constrained testing of core-based systems, *Proc. VLSI Design* (2000).
12. 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, *Proc. ATS* (2001), pp. 265–270.
13. Y. Zorian, A distributed BIST control scheme for complex VLSI devices, *Proc. 11th IEEE VLSI Test Symp.* (1993), pp. 4–9.
14. V. Iyengar and K. Chakrabarty, System-on-a-chip test scheduling precedence relationships, preemptive, and power-constraints, *IEEE Trans. Computer-Aided Design* **21** (2002) 1088–1094.
15. R. Chou, K. Saluja and V. Agrawal, Scheduling tests for VLSI systems under power constraints, *IEEE Trans. VLSI Syst.* **5** (1997) 175–185.

J CIRCUIT SYST COMP 2006.15:331-349. Downloaded from www.worldscientific.com
by Harman Haman on 11/04/14. For personal use only.

This article has been cited by:

1. HAIDAR M. HARMANANI, HASSAN A. SALAMY. 2006. A SIMULATED ANNEALING ALGORITHM FOR SYSTEM-ON-CHIP TEST SCHEDULING WITH, POWER AND PRECEDENCE CONSTRAINTS. *International Journal of Computational Intelligence and Applications* **06**:04, 511-530. [Abstract] [References] [PDF] [PDF Plus]

J CIRCUIT SYST COMP 2006.15:331-349. Downloaded from www.worldscientific.com
by Harman Haman on 11/04/14. For personal use only.