A Multiobjective Optimization Method for the SOC Test Time, TAM, and Power Optimization Using a Strength Pareto Evolutionary Algorithm

86

Wissam Marrouche, Rana Farah, and Haidar M. Harmanani

Abstract

System-On-Chip (SOCs) test minimization is an important problem that has been receiving considerable attention. The problem is tightly coupled with the number of TAM bits, power, and wrapper design. This paper presents a multiobjective optimization approach for the SOC test scheduling problem. The method uses a *Strength Pareto Evolutionary Algorithm* that minimizes the overall test application time in addition to power, wrapper design and TAM assignment. We present various *experimental results* that demonstrate the effectiveness of our method.

Keywords

SOC test scheduling • Multiobjective optimization • SPEA2 • Metaheuristics.

86.1 Introduction

System-on-Chip is a design methodology that integrates *predesigned* and *preverified* intellectual property blocks or cores are embedded on a single die [1]. A core maybe designed in a hierarchical fashion and thus may embed other cores. A major challenge in the SOC design paradigm involves integrating the IP blocks and developing a test methodology for post-manufacturing tests since cores can only be tested after integration. A generic conceptual test access architecture for an embedded core, introduced by Zorian et al. [2], consists of a *test source* and a *sink*, a *test access mechanism* and a *core wrapper*. The test source is used for test stimulus generation while the response evaluation is carried out by the test sink. The *source* and the *sink* can be either implemented off-chip or on-chip. Test access mechanism (TAM) serves as a "test data highway" that transports test patterns between the *source* and the *core* as well as between the *core* and the *sink*. Finally, the core is surrounded with test logic, known as the *test wrapper*, that provides switching functionality between *normal* access and *test* access via the TAM [3].

One of the main challenges in core-based designs is test time reduction which aims at maximizing the simultaneous test of all cores. The problem, known as test scheduling, determines the order in which various cores are tested and has been shown to be $\mathcal{NP}$-complete [4]. Hierarchical cores further complicate the SOC test scheduling process especially in the case of hard or legacy cores. Hierarchical cores have multiple levels of test hierarchy with the top level is the SOC itself and consists of several mega-cores that have their own embedded cores. Cores at level n are *parent* cores with respect to the cores at level $n + 1$ [5]. Finally, power dissipation strongly impacts test parallelism since it depends on the switching activity resulting from the application of test vectors to the system [6].

W. Marrouche (✉)
Department of Electrical & Computer Engineering,
American University of Beirut, Beirut, Lebanon
e-mail: wissam.marrouche@gmail.com

R. Farah
Department of Computer & Software Engineering,
École Polytechnique de Montréal, Quebec, Canada
e-mail: rana.farah@polymtl.ca

H.M. Harmanani
Department of Computer Science, Lebanese American University,
Byblos, Lebanon
e-mail: haidar@lau.edu.lb

S. Latifi (ed.), *Information Technology – New Generations*, Advances in Intelligent Systems and Computing 558, DOI 10.1007/978-3-319-54978-1_86

This paper presents a multiobjective approach for test minimization, variable TAM assignment and partitioning, and wrapper design for hierarchical SOCs using a strength Pareto evolutionary algorithm. The method takes into consideration power and precedence constraints. Formally, given a SOC with N_C cores, a total TAM width $\mathcal{W}$, a set of design and test constraints including power and precedence constraints, and a set of parameters for each mega-core, the problem we address in this paper is to minimize the overall test time such that, (i) the test schedule for the entire SOC is efficient, (ii) the TAM wires are optimally partitioned and assigned to cores (iii) the wrapper configuration for each core is determined, (iv) $\mathcal{W}$ is not exceeded, and (v) hierarchical cores receive at least their prespecified TAM widths.

86.1.1 Related Work

Several researchers addressed the test scheduling problem but mostly assumed flat cores with a single level TAM; however, this assumption is only valid if the embedded cores are *mergeable*. Iyengar et al. [7] first formulated the integrated wrapper/TAM optimization problem using ILP and developed later several heuristics for solving the problem. Huang et al. [8] modeled the problem as a restricted 3-D bin packing problem and proposed a heuristic to solve it. Goel et al. [9] proposed an efficient heuristic for fixed-width architecture while Xu et al. [10] proposed a method for multi-level test access mechanism that facilitates test data reuse for hard mega-cores in hierarchical SOCs. Su et al. [11] formulated the problem using graph-based approach and solved it using tabu search. Chakrabarty et al. [12] proposed a combination of integer linear programming, enumeration and efficient heuristics in order to solve the problem for hierarchical cores. Wuu et al. [13] extended the wrapper design method presented by Zou et al. [14] and solved the test scheduling problem by mapping it into a floor planning problem. Wang et al [15] proposed a test scheduling algorithm that is based on the 2-D bin-packing model. Ooi et al. [16] proposed an enhanced rectangle packing test scheduling algorithm efficiently compacts the test scheduling floor plan. SenGupta et al. [17] proposed a new test planning and test access mechanism method for stacked integrated circuits.

86.1.2 Strength Pareto Evolutionary Algorithm

Evolutionary algorithms are effective optimization techniques that are global in scope. Dominance-based evolutionary algorithms have emerged as reliable approaches for generating Pareto optimal solutions to multi-objective optimization problems. SPEA2 [18] is an improved strength Pareto evolutionary algorithm that incorporates a fine-grained fitness assignment strategy, a density estimation technique, and an enhanced archive truncation method. The algorithm uses genetic recombination and mutation in order to locate and maintain a front of non-dominated solutions, ideally a set of Pareto optimal solutions. SPEA2's Pareto measure is the number of solutions that dominates a candidate solution, and its crowding measure is based on the distance to other individuals in the multiobjective space [19]. The algorithm maintains, separate from the population, an *archive* of the non-dominated set which provides a form of elitism.

The remainder of the paper is organized as follows. In Sect. 86.2 we formulate the multiobjective SOC test scheduling problem and describe the genetic encoding, the initial population, the genetic operators, and the fitness function. Section 86.3 presents the annealing test scheduling while Sect. 86.4 presents the hierarchal test scheduling algorithm. We conclude with experimental results in Sect. 86.5.

86.2 Problem Formulation

The test time of a SOC depends on the individual cores test times as well as on the test start times. The cores test times are based on TAM assignments which consequently affect the wrapper design. Test adaptation is performed by serially connecting internal scan-chains, the wrapper input cells and the wrapper output cells in order to form wrapper chains. There is a clear trade-off between test time and TAM capacity. For example, while the length of the wrapper chains directly affects the core's test time, the number of wrapper chains affects the number of TAM bits as each wrapper chain's input and output must be connected to a TAM wire. The test time t_i of core i is determined by the shortest and the longest wrapper chains as follows [20]:

$$t_i = (1 + \max(s_i, s_o)) \times p + \min(s_i, s_o), \qquad (86.1)$$

where, s_i and s_o denote respectively the scan-in and scan-out time for the core and p_i denotes the number of test patterns applied on the core i. Embedding cores adds an additional test conflict problem arising from the fact that it may not be possible to test the parent and the child cores concurrently due, for example, to a conflict in the use of the input wrapper cells. Therefore, an effective test scheduling approach must minimize the test time while addressing test resources allocation and conflicts arising from the use of shared test access mechanism.

In this paper we explore design trade-offs among conflicting objectives by integrating test scheduling, wrapper design and TAM assignment using a multiobjective

evolutionary algorithm. In what follows, we describe the multiobjective evolutionary test scheduling algorithm.

86.2.1 Chromosomal Representation

A chromosome is represented using a vector whose length is equal to the number of cores in the SOC. Each chromosome is represented using a vector whose length is equal to the number of cores in the SOC and corresponds to a candidate solution. Every gene corresponds to a block that includes the assigned *TAM bits*, *core test time*, and *power* as shown in Fig. 86.1. During every generation, chromosomes are selected in order to seed the next generation, yielding sub-optimum solutions.

86.2.2 Initial Population

Increasing the number of TAM bits may not guarantee decreasing the test time which may hit a Pareto optimal point. For all points with the same test time, a Pareto-optimal point is the point where the least number of TAM wires is used [21]. The test time does not decrease even if the number of wrapper chains increases. Figure 86.2a shows that the test time decreases as the number of TAM bits increases for core 5 in the *d695* benchmark, and hits several Pareto optimal points.

At the beginning of each run, the algorithm generates a pool of cores where the test time of each core corresponds to a specific Pareto optimal test point, TAM bits, and power. For example, the pool for the d695 benchmarks includes 82 cores; the pool for core 5 is shown in Fig. 86.2b. The initial population is next generated by randomly selecting cores from the pool, and populating all chromosomes in order to create an initial population of size 70. The wrapper for each core is designed using an improved Best Fit Decreasing (BFD) heuristic [13]. Finally, each individual is test scheduled using a bin packing simulated annealing algorithm.

86.2.3 Archive

Evolutionary algorithms replace individuals with more fit ones using various strategies. However, individuals that are more *fit* than the rest take over the population very quickly leading to premature convergence. This problem is alleviated in *SPEA2* using an archive, A_t, of non-dominated solutions. During each generation, the *best* non-dominated solutions are added to the archive. The archive *size* is set to 70.

86.2.4 Fitness Function

The test scheduling problem has multiple and conflicting objectives that include test time, power, and TAM bits allocation. We optimize all three objectives using a vector-valued fitness function $f = (Test\ Time, Power, TAM\ Width)$, based on the *Pareto dominance concept*. An objective vector f_1 is said to dominate another objective vector f_2 ($f_1 \succ f_2$) if no component of f_1 is smaller than the corresponding component of f_2 and at least one component is greater [18].

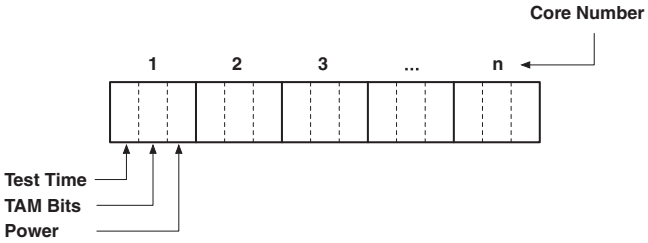
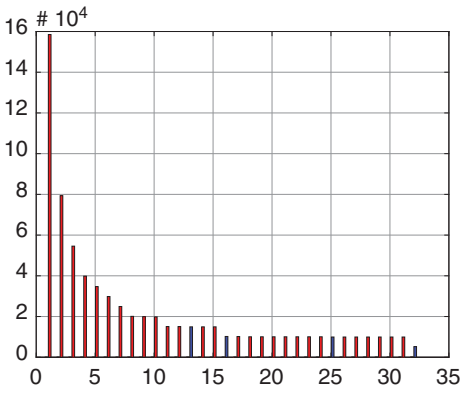


Fig. 86.1 Chromosome representation

Fig. 86.2 (**a**) Pareto test points for core 5, d695, (**b**) Cores pool



<TAM Bits, Test Time, Power>

<1, 158507, 690>
<2, 79364, 690>
<3, 54576, 690>
<4, 39847, 690>
<5, 34699, 690>
<6, 29814, 690>
<7, 24930, 690>
<8, 20089, 690>
<9, 19935, 690>
<10, 19825, 690>
<11, 15072, 690>
<12, 15051, 690>
<13, 14940, 690>
<16, 10210, 690>
<17, 10166, 690>
<18, 10056, 690>
<25, 9946, 690>
<26, 9945, 690>
<32, 5215, 690>

(b)

Each individual i in the archive A_t as well as in the population P_t is assigned a strength value, $Strength(i)$, representing the number of solutions it dominates:

$$Strength(i) = \sum_{i, j \in P_t \cup A_t,} [i \succ j] \tag{86.2}$$

The bracketed notation [S] is 1 if S is true and 0 otherwise. The raw fitness $f(i)$ of an individual i is calculated as follows:

$$f(i) = \sum_{j \in P_t \cup A_t, j \succ i} Strength(j) \tag{86.3}$$

The raw fitness is determined by the strengths of its dominators in the *archive* and in the *population*. Thus, the fitness to be minimized, $f(i) = 0$, corresponds to a non-dominated individual, while a high $f(i)$ value means that i is dominated by many individuals. Ties with the same fitness are broken by a nearest neighbor density estimation function $D(i)$:

$$D(i) = \frac{1}{\sigma_i^k + 2} \tag{86.4}$$

Where σ_i^k is the Euclidean distance of the objective values between a given solution i and the k^{th} nearest neighbor, $k = \sqrt{(N_P + N_A)}$.

86.2.5 Selection and Reproduction

We use a multi-objective tournament selection of size 2. Thus, during each generation, two individuals are selected randomly and the fittest is preserved for the next generation.

86.2.6 Genetic Operators

We explore the design space using two genetic operators, *mutation* and *crossover*, that are applied iteratively with their corresponding probabilities. Both operators ensure that the generated solutions are feasible, and are followed by the annealing bin packing test scheduling algorithm.

86.2.6.1 Mutation

Mutation is used to explore the search space while inhibiting premature convergence. The operator selects a random chromosome. A random core is next selected and replaced with another random core from the cores pool using a probability $P_m = 80\%$. The operator results with an increase or decrease in the number of TAM bits.

86.2.6.2 Crossover

We use multiple point uniform crossover which adds more variability than fixed-point crossover with probability $P_c = 40\%$. The crossover operator takes two randomly selected chromosomes, and creates two offspring by selecting each gene from either parent with probability P_c. The uniform crossover generates a diverse new offspring than the traditional one-point or two-point crossover.

86.3 Annealing Bin Packing Algorithm

Once the individuals have been created using the evolutionary operators, the algorithm uses an annealing bin-packing strategy in order to compute the test time for the new population. The annealing algorithm uses a floorplanning sequence-pair representation where each configuration is represented using an ordered pair (S_+, S_-) of block permutations. Together, the two permutations represent geometric relations between every pair of blocks.

Each annealing configuration represents a test schedule but with a different cost. The search space is explored using the following neighborhood functions:

(1) *Test Schedule Exploration:* the algorithm randomly selects a random core i from the current configuration and changes its starting time S_i to the *end time* F_j of a randomly chosen core j by adjusting the corresponding elements in the sequence-pair.
(2) *Test Schedule Swap:* the algorithm randomly selects and swaps two random elements in the sequence-pair.

The neighborhood functions use a constructive approach that leads to incremental feasible test schedules. The variation in the cost functions, Δ_C, is computed and if negative then the transition from C_i to C_{i+1} is accepted. If the cost function increases, the transition is accepted with a probability based on the *Boltzmann* distribution. The temperature is gradually decreased from a high starting value, $T_0 = 1000$, where almost every proposed transition, positive or negative, is accepted to a freezing temperature, $T_f = 0.\ 1$, where no further changes occur. As for the number of iterations, M, and the iteration multiplier, β, they were respectively set to 5 and 1.05. The algorithm stops after a Max_{Time} of 6000 ms.

Algorithm 1 Hierarchical test scheduling algorithm

```
1: function HIERARCHICAL_TEST_SCHEDULE()
2:     for all levels starting from level n − 1 to level 0 do
3:         find all mega cores
4:         for each mega core do
5:             assign TAM bits for the children cores
6:             calculate the test time for each child core
7:             Schedule all children cores using simulated annealing
8:             calculate overall test time
9:         end for
10:        if (level == 0) then
11:            calculate the overall time of the SOC
12:        end if
13:    end for
14:    for (i = 0; i < n; i++) do
15:        Select a random core i from any level
16:        Move to the next/previous Pareto-optimal point
17:        if the transformation is accepted then
18:            Annealing_Bin_Packing_Algorithm()
19:        end if
20:        Update T_i and all parents cores that contain this core.
21:    end for
22: end function
```

86.4 Hierarchical Test Scheduling Algorithm

The hierarchical core model is a recursive model where a wrapped parent core has an external TAM that connects externally to the parent core and an internal TAM that consists of the TAM architecture of the children cores. We start with the *mega-cores* that are at level n where each mega-core is considered as a separate SOC. The embedded child cores are initially assigned TAMs based on Algorithm 1 where we define a dominant core as a core such that when allocated the same TAM width as all its peers it returns a larger test time than its peers. For cores that are at level n and that have more than one peer, the algorithm allocates half the TAM bits of the parents to the child cores. During the TAM assignment process, the algorithm ensures that no mega-core is assigned more than the specified TAM bits and that no core is assigned more than the TAM bits of its parent. Once the TAM assignment is performed, the wrapper design is determined using the BFD algorithm. The algorithm next iterates over each mega-core using our simulated annealing in order to determine the *test schedule* as well as the wrapper design for the mega-core itself. The test wrapper is next designed by configuring the scan elements into wrapper chains using the BFD algorithm. On the other hand, test data serialization maybe necessary since a mega-core width maybe assigned less than the TAM width it requires. Thus, a mega-core i with top-level TAM width w_i maybe provided with ${w_i}^*$ TAM bits such that ${w_i}^* \leq w_i$. The test time for mega-core i then reduces to $\mathcal{T}_i^* = \left\lceil \frac{w_i}{w_i^*} \right\rceil * \mathcal{T}_i$ where $\mathcal{T}_i$ is the total testing time for the embedded cores on the internal TAM partition for mega-core i. Finally, the testing time for mega-core i must include the test time for the top-level test time and this reduces to: $\mathcal{T}_i^{W_i^*} = \mathcal{T}_i^* + \mathcal{T}_i^{Mega}$ where $\mathcal{T}_i^{W_i^*}$ is the testing time for mega-core i with an external TAM of width ${w_i}^*$ and $\mathcal{T}_i^{Mega}$ is the testing time at the system level TAM architecture. Once all cores at level i have been processed, the algorithm recurses back to level $i - 1$ and considers the remaining mega-cores.

Algorithm 2 Multi-Objective Test Scheduling Algorithm()

```
Input: A set of Cores
Output: Minimum test Schedule
1: N_P = N_A = 70
2: N_t ← 20
3: P_0 ← Initial(population)
4: A_0 ← ∅
5: t ← 1
6: while t < N_t do
7:     Design the cores wrappers using the modified BFD algorithm
8:     Schedule all Si ∈ P_t ∪ A_t using simulated annealing
9:     for all S_i ∈ P_t do
10:        f_{S_i} ← fitness values of individuals in P_t and A_t
11:        Rank all S_i by their fitness value and the k-nearest neighbor
12:    end for
13:    A_{t+1} ← All non-dominated individuals in P_t ∪ A_t
14:    if size(A_{t+1}) > N_A then
15:        Truncate (A_{t+1})
16:    else if size(A_{t+1}) < N_A then
17:        A_{t+1} ← best non-dominated individuals in P_t ∪ A_t
18:    end if
19:    Create N_P offspring using Mutation and crossover
20:    for (i = 0; i < N_P; i++) do
21:        Selects two random individuals from P_t
22:        Keep the best
23:    end for
24:    t ← t + 1
25: end while
```

86.5 Experimental Results

The proposed algorithm was implemented using `Python`, and tested on the ITC'02 benchmarks. The algorithm starts by creating an initial population of size 70 where each chromosome represents a candidate solution that has a different test time, power, and wrapper design. The algorithm also creates an archive of the same size. The archive is initially empty.

During every generation, chromosomes are selected for reproduction using the *mutation* and *crossover* operators, resulting in a new population that is equal in size to the initial population. The algorithm schedules the newly generated individuals using simulated annealing in order to minimize the test scheduling time and power. Next, all individuals are assigned a fitness function, and the new population is next selected using a tournament selection of size 2. All the best solutions are maintained in the archive using elitism. Figure 86.3 shows the archive cost fitness function quality which monotonically decreases with each evolutionary generation.

Table 86.1 shows the results of our algorithm with power constraints. We compare our hierarchical test scheduling algorithm with Chakrabarty et al. [12], Wang et al. [15],

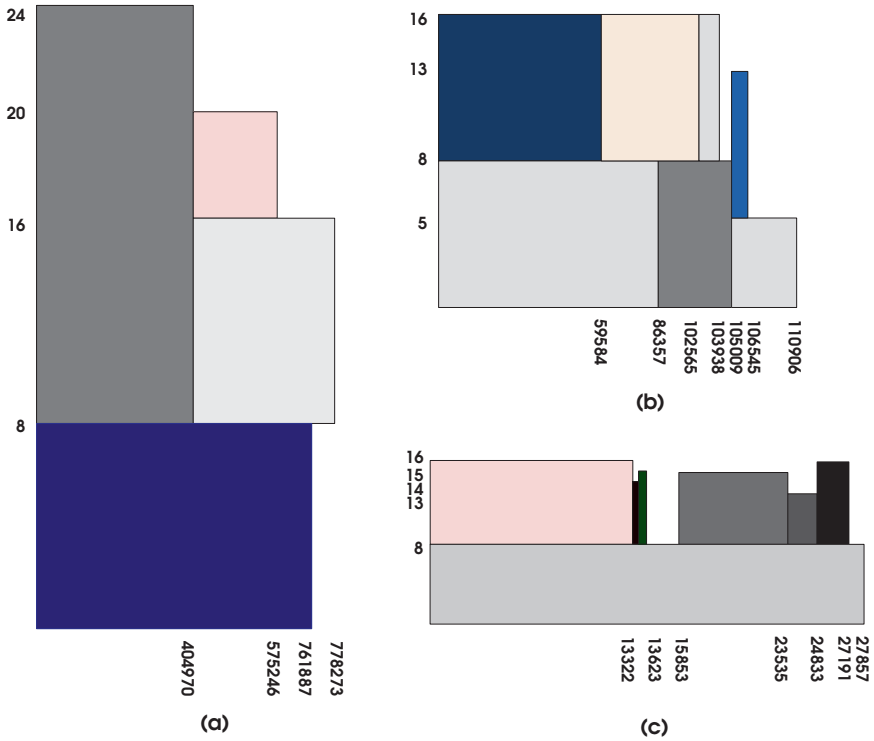


Fig. 86.3 Archive quality in the case of the D965 benchmark. (**a**) TAM Width. (**b**) Test Time. (**c**) TAM, Test Time, and Power

Table 86.1 ITC'02 benchmark test times with power dissipation (in mW)

	TAM bits							
	16	24	32	40	48	56	64	80
d695	412224 (1177)	27822 (2463)	20766 (2364)	17657 (2321)	16612 (1960)	15111 (2587)	3392 (3746)	10634 (3746)
p22810	412224 (2947)	312910 (3265)	247737 (2992)	199110 (4184)	176758 (3920)	159160 (5426)	143760 (3735)	110137 (4727)
p93791	1803764 (9886)	1157365 (15891)	1068967 (10784)	914832 (21440)	772649 (16415)	630287 (19026)	612505 (20720)	502997 (23429)

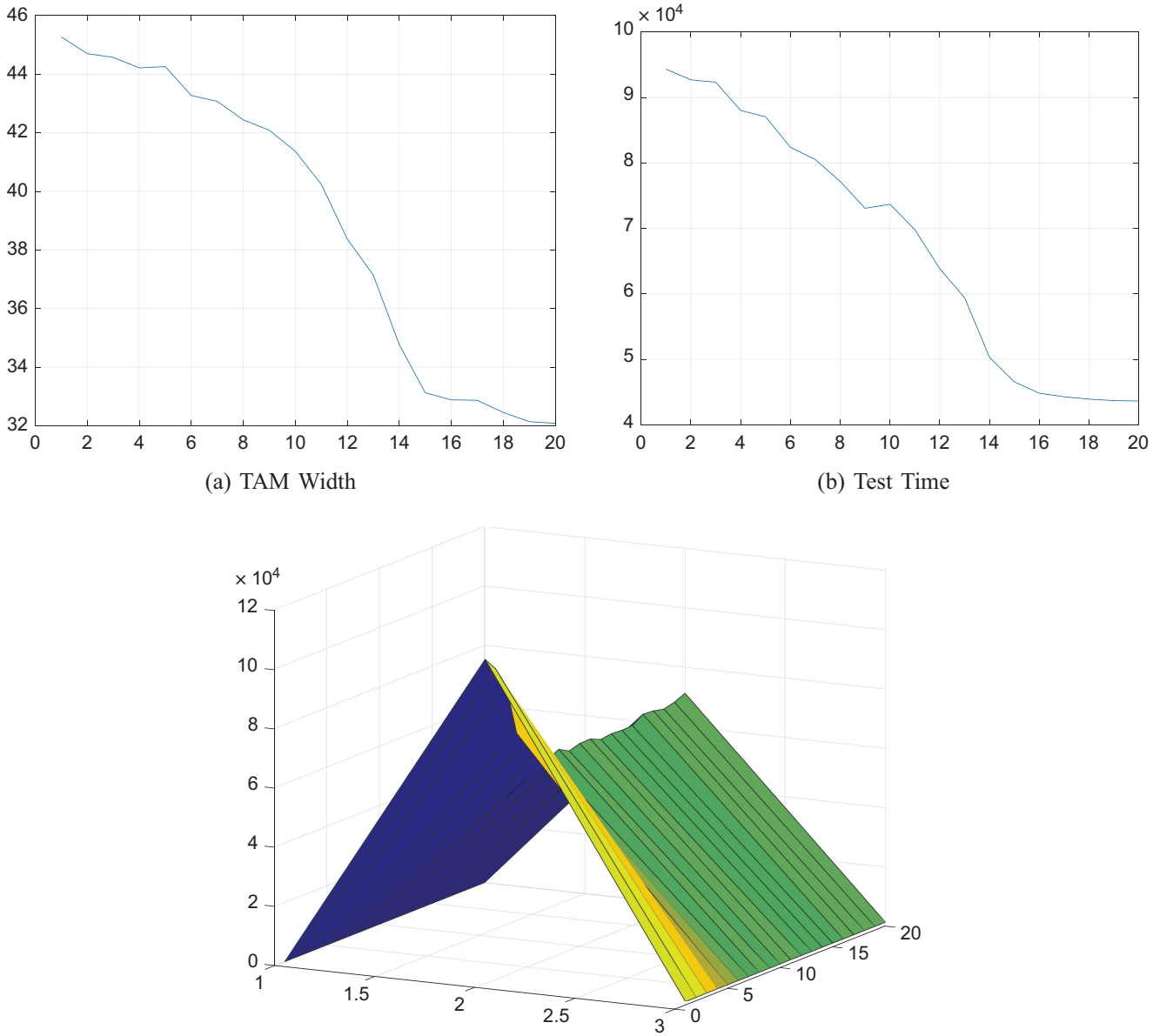
and Xu et al. [10]. Detailed results comparisons are shown in Table 86.2. The TAM widths supplied to the mega-cores were determined to be 8 bits for SOCs *p22810* and *a586710*, and 16 bits for SOCs *p34392* and *p93791*. Figure 86.4 illustrates the hierarchical test schedule for the *p34392* SOC benchmark using a TAM width of 16-bits. The resulting schedule takes into consideration two megacores, and the system was able to test it in 778,273 cycles. We also compare our method to various researchers [7–9, 11, 14, 14, 21–23] for flat designs based on fixed-width as well as based on flexible width TAM. Detailed results comparisons are shown in Table 86.3.

86.6 Conclusion

We presented a multiobjective approach for test minimization, variable TAM assignment and partitioning, and wrapper design for hierarchical SOCs using a strength Pareto evolutionary algorithm. The algorithm was able to achieve near optimal solutions in a reasonable amount of time for big systems.

Table 86.2 Hierarchical ITC'02 SOC test times comparisons

SOC	$\mathcal{W}$	[10]	[15]	[12]	Ours
a586710	16	5.27×10^7	–	4.44×10^7	**42117546**
	24	3.06×10^7	–	3.06×10^7	**22973206**
	32	2.19×10^7	–	2.28×10^7	**21058772**
	40	1.91×10^7	–	2.50×10^7	**17229905**
	48	1.53×10^7	–	2.14×10^7	**13401037**
	56	–		2.14×10^7	**13031723**
	64	1.41×10^7		2.14×10^7	**13031723**
p93791	16	**1865140**	–	–	1953223
	24	1486628	–	1650880	**1377332**
	32	1486628	1937130	1021320	**991915**
	40	**801271**	–	916852	845391
	48	801271	1272314	681816	**670660**
	56	–	–	632125	**629510**
	64	553775	947554	**521064**	545583
p22810	16	496804	–	505858	**421422**
	24	353619	–	412682	**305936**
	32	**280634**	466044	396473	**232531**
	40	**280634**	–	366260	**280634**
	48	**280634**	338374	366260	**280634**
	56	–	–	–	**280634**
	64	**280634**	285231	366260	**280634**
p34392	16	1467705	–	–	**1188916**
	24	1467705	–	1347023	**778273**
	32	776537	–	788873	**713071**
	40	776537	–	728426	**606261**
	48	**60261**	–	618597	**606261**
	56	–		618597	**606261**
	64	**60261**	–	618597	**606261**



(a) TAM Width

(b) Test Time

(c) TAM, Test Time, and Power

Fig. 86.4 (**a**) Hierarchical test schedule for p34392; (**b**) Mega-core 2 test schedule; (**c**) Mega-core 1 test schedule

Table 86.3 ITC'02 benchmark results

	Fixed-width				Flexible-width						
SOC	$\mathcal{W}_{max}$	[7]	[22]	[9]	[14]	[21]	[8]	[11]	[14]	[23]	Ours
d695	16	42568	42644	44307	–	44545	42716	41905	41604	41847	**36937**
	24	28292	30032	28576	–	31569	28639	28231	28064	29106	**26482**
	32	21566	22268	21518	–	23306	21389	21467	21161	20512	**20089**
	40	17901	18448	17617	–	18837	17366	17308	16993	18691	**16284**
	48	16975	15300	14608	–	16984	15142	14643	14182	17257	**14093**
	56	13207	12491	12462	–	14974	13208	12493	12085	–	**11519**
	64	12941	12941	11033	–	11984	11279	11036	10723	13348	**10643**
g1023	16	–	–	34459	–	–	31444	32602	31139	–	**15063**
	24	–	–	22821	–	–	21409	22005	21024	–	**15063**
	32	–	–	16855	–	–	16489	17422	15890	–	**15063**
	40	–	–	14794	–	–	14794	14794	14794	–	**14794**
	48	–	–	14794	–	–	14794	14794	14794	–	**14794**
	56	–	–	14794	–	–	14794	14794	14794	–	**14794**
	64	–	–	14794	–	–	14794	14794	14794	–	**14794**
p22810	16	462210	468011	458068	434922	489192	446684	465162	438619	473418	**421422**
	24	361571	313607	299718	313607	330016	300723	317761	289287	352834	297596
	32	312659	246332	222471	245622	245718	223462	236796	218855	236186	227938
	40	278359	232049	190995	194193	199558	184951	193696	175946	195733	186788
	48	278359	232049	160221	164755	173705	167858	174491	147944	159994	166169
	56	268472	153990	145417	145417	157159	145087	155730	126947	–	144374
	64	260638	153990	133405	133628	142342	128512	145417	109591	128332	121961
p34392	16	998733	1033210	1010821	1021510	1053491	1016640	995739	944768	–	**873120**
	24	720858	882182	680411	729864	759427	681745	690425	628602	–	**587034**
	32	591027	663193	551778	630934	544579	553713	544579	544579	–	**544579**
	40	544579	544579	544579	544579	544579	544579	544579	544579	–	**544579**
	48	544579	544579	544579	544579	544579	544579	544579	544579	–	**544579**
	56	544579	544579	544579	544579	544579	544579	544579	544579	–	**544579**
	64	544579	544579	544579	544579	544579	544579	544579	544579	–	**544579**
p93791	16	1771720	1786200	1791638	1775586	1932331	1791860	1767248	1757452	1827819	1777840
	24	1187990	1209420	1185434	1198110	1310841	1200157	1178776	1169945	1220469	1204168
	32	887751	894342	912233	936081	988039	900798	906153	878493	945425	943087
	40	698583	741965	718005	734085	794027	719880	737624	718005	787588	760868
	48	599373	599373	601450	599373	669196	607955	608285	594575	639217	638993
	56	514688	514688	528925	514688	568436	521168	539800	509041	–	557890
	64	460328	473997	455738	472388	517958	549233	485031	447974	457862	489591
u226	16	–	–	18663	–	–	13416	18663	13333	–	**8109**
	24	–	–	13331	–	–	10750	14745	8084	–	**4157**
	32	–	–	10665	–	–	6746	10665	6746	–	**4157**

(continued)

Table 86.3 (continued)

	Fixed-width				Flexible-width						
SOC	$\mathcal{W}_{max}$	[7]	[22]	[9]	[14]	[21]	[8]	[11]	[14]	[23]	Ours
	40	–	–	8084	–	–	5332	8084	5332	–	**4157**
	48	–	–	7999	–	–	5332	7999	5332	–	**4157**
	56	–	–	7999	–	–	4080	7999	4080	–	**4157**
	64	–	–	7999	–	–	4080	7999	4080	–	**4157**
f2126	16	–	–	372125	–	–	357109	357089	357088	–	**335666**
	24	–	–	335334	–	–	335334	335334	335334	–	**335334**
	32	–	–	335334	–	–	335334	335334	335334	–	**335334**
	40	–	–	335334	–	–	335334	335334	335334	–	**335334**
	48	–	–	335334	–	–	335334	335334	335334	–	**335334**
	56	–	–	335334	–	–	335334	335334	335334	–	**335334**
	64	–	–	335334	–	–	335334	335334	335334	–	**335334**
t512505	16	–	–	10530995	–	–	10531003	11210100	10530995	–	**10530995**
	24	–	–	10453470	–	–	10453470	10525823	10453470	–	**10453470**
	32	–	–	5268868	–	–	5268872	6370809	5268868	–	**5268868**
	40	–	–	5228420	–	–	5228420	5240493	5228420	–	**5228420**
	48	–	–	5228420	–	–	5228420	5239111	5228420	–	**5228420**
	56	–	–	5228420	–	–	5228420	5228474	5228420	–	**5228420**
	64	–	–	5228420	–	–	5228420	5228489	5228420	–	**5228420**
a586710	16	–	–	41523868	–	–	42198943	42,067,708	32,626,782	–	**32417445**
	24	–	–	28,716,501	–	–	27,785,885	27,907,180	23,413,604	–	**22973206**
	32	–	–	22,475,033	–	–	21,735,586	22,70,4821	18,838,663	–	**17195389**
	40	–	–	19,048,835	–	–	19,041,307	19,041,307	14,260,216	–	**14249791**
	48	–	–	15,315,476	–	–	15,071,730	15,212,440	12,811,087	–	**12811087**
	56	–	–	13,415,476	–	–	14,945,057	13,401,034	12,573,448	–	**11486603**
	64	–	–	12,700,205	–	–	12,754,584	11,567,464	10,65,9014	–	**9572169**
d281	16	–	–	8444	–	–	7948	8156	7946	–	**7347**
	24	–	–	6408	–	–	5486	5830	5485	–	**4992**
	32	–	–	5084	–	–	4070	4640	4070	–	**3926**
	40	–	–	3964	–	–	3926	3926	3926	–	**3926**
	48	–	–	3926	–	–	3926	3926	3926	–	**3926**
	56	–	–	3926	–	–	3926	3926	3926	–	**3926**
	64	–	–	3926	–	–	3926	3926	3926	–	**3926**

References

1. Saleh, R., Wilton, S., Mirabbasi, S., Hu, A., Greenstreet, M., lemieux, G., Pande, P., Grecu, C., & Ivanov, A. (2006). System-on-chip: Reuse and integration. In *Proceedings of the IEEE*.
2. Zorian, Y., Marinissen, E., & Dey, S. (1998). Testing embedded core-based system chips. In *Proceedings of ITC*.
3. Bushnell, M., & Agrawal, V. (2000). *Essentials of electronic testing for digital, memory & mixed-signal VLSI circuits*. New York: Kluwer-Academic Publishers.
4. Chakrabarty, K. (2000). Test scheduling for core-based systems using mixed-integer linear programming. *IEEE Transactions on CAD*.
5. Sehgal, A., Goel, S., Marinissen, E., & Chakrabarty, K. (2004). IEEE P1500-Compliant test wrapper design for hierarchical cores. In *Proceedings of ITC*.
6. Chakrabarty, K. (1999). Test scheduling for core-based systems. In *Proceedings of ITC*.
7. Iyengar, V., Chakrabarty, K., & Marinissen, E. (2002). Test wrapper and test access mechanism co-optimization for system-on-a-chip. *JETTA*.
8. Huang, Y., Reddy, S., Cheng W.-T., Reuter, P., Mukherjee, N., Tsai, C., Samman, O., & Zaidan, Y. (2002). Optimal core wrapper width selection and SOC test scheduling on 3-D bin packing algorithm. In *Proceedings of ITC*.
9. Goel, S., & Marinissen, E. (2003). SOC test scheduling design for efficient utilization of bandwidth. In *ACM TODAES*.
10. Xu, Q., & Nicolici, N. (2004). Time/area tardeoffs in testing hierarchical SOCs with hard mega-cores. In *Proceedings of ITC*.
11. Su, C., & Wu, C. (2004). A graph-based approach to power-constrained test scheduling. *JETTA, 20*.
12. Chakrabarty, K., Iyengar, V., & Krasniewski, M. (2005). Test planning for modular testing of hierarchical SOCs. In *IEEE Transactions of CAD*.
13. Wuu, J.-Y., Chen, T.-C., & Chang, Y.-W. (2005). SOC test scheduling using the B*-tree based floorplanning technique. In *Proceedings of ASP-DAC*.
14. Zou, W., Reddy, S. R., Pomeranz, I., & Huang, Y. (2003). SOC test scheduling using simulated annealing. In *Proceedings of VTS*.
15. Wang, T.-P., Tsai, C.-Y., Shieh, M.-D., & Lee, K.-J. (2005). Efficient test scheduling for hierarchical core based designs. In *Proceedings of TSA-DAT*.
16. Ooi, C. Y., Sua, J. P., & Lee, S. C. (2012). Power-aware system-on-chip test scheduling using enhanced rectangle packing algorithm. *CEE*.
17. SenGupta, B., & Larsson, E. (2014). Test planning and test access mechanism design for stacked chips using ILP. In *Proceedings of VTS*.
18. Zitzler, E., Laumanns, M., & Bleuler, S. (2004). A tutorial on evolutionary multiobjective optimization. In X. Gandibleux (Ed.), *Metaheuristics for multiobjective optimisation*. Berlin/Heidelberg/New York: Springer.
19. Luke, S. (2015). *Essentials of metaheuristics*. Available online on http://cs.gmu.edu/∼sean/book/metaheuristics/. Lulu.
20. Marinissen, E., Goel, S., & Lousberg, M. (2000). Wrapper design for embedded core test. In *Proceedings of ITC*.
21. Iyengar, V., Chakrabarty, K., & Marinissen, E. J. (2002). On using rectangle packing for SOC wrapper/TAM co-optimization. In *Proceedings of VTS*.
22. Iyengar, V., Chakrabarty, K., & Marinissen, E. (2002). Efficient wrapper/TAM co-optimization for large SOCs. In *Proceedings of DATE*.
23. Pouget, J., Larsson, E., & Peng, Z. (2005). Multiple-constraint driven system-on-chip time optimization. *JETTA*.