# Test bus assignment, sizing, and partitioning for system-on-chip

# Attribution, dimensionnement et partitionnement de bus de tests pour système sur puce

Haidar M. Harmanani and Rachel Sawan*

The test access mechanism (TAM) is an important element of test architectures for embedded cores and is responsible for on-chip test pattern transport from the source to the core under test to the sink. Efficient TAM design is of critical importance in system-on-chip integration since it directly impacts testing time and hardware cost. In this paper, an efficient genetic algorithm for designing test access architectures while investigating test bus sizing and concurrently assigning cores to test buses is proposed. Experimental results are presented to demonstrate that the proposed TAM optimization methodology provides efficient test bus designs with minimum testing time while outperforming reported techniques.

Le mécanisme de tests d'accès (TAM) est un élément important des architectures d'essais pour les noyaux embarqués. Il est responsable de la circulation des patrons de tests sur puce de la source au noyau sous essais et jusqu'au récepteur. La conception efficace du TAM est d'importance critique dans l'intégration des systèmes sur puce puisqu'elle affecte directement le temps d'essais et le coût du matériel. Dans cet article, un algorithme génétique efficace est présenté. Celui-ci permet de concevoir des architectures d'essais d'accès. On présente aussi une étude des tailles de bus d'essais de même qu'une étude sur l'attribution de noyaux de tests aux bus de tests. Des résultats expérimentaux sont fournis pour démontrer que la méthodologie proposée d'optimisation du TAM fournit des conceptions efficaces de bus d'essais avec un temps d'essais minimum tout en surpassant les techniques connues.

**Keywords:** core-based systems; embedded core testing; test access mechanism

## I. Introduction

Advances in semiconductor process technologies have enabled the integration of an entire system on a single chip, based on a redesign philosophy that divides the CAD community into core providers and core integrators. Core providers create embedded cores, which are predesigned and preverified complex logic blocks that cover a wide range of functions such as CPU, DSPs, media processors, communication modules, memories, and mixed-signal modules. Core integrators, on the other hand, create the system-on-chip (SoC) by assembling the cores within the system and by combining available cores and their custom user-defined logic (UDL).

One of the implications of the core-based design methodology is that test development for large core-based system chips should also be core-based. Test strategies for embedded cores must ensure the existence of an access path in the on-chip hardware. Furthermore, the core tests, as given by the core provider, must be translated from the core terminals to the IC pins. A generic conceptual test access architecture for an embedded core, introduced by [1], consists of a test source and a sink, a test access mechanism (TAM) 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 implemented either off-chip (ATE) or on-chip (BIST). The test access mechanism 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. A TAM is characterized by its transport capacity, also referred to as the TAM bandwidth, which is given by the data rate required by the core's test. Finally, the core is surrounded with test logic, known as the test wrapper, which provides switching functionality between normal access and test access via the TAM [2]. The test wrapper has at least three modes of operation for each input terminal: a normal mode in which the core's terminal is driven by the host chip; an external test mode in which the wrapper element observes the core's input terminal for interconnect test; and an internal test mode in which the wrapper element controls the state of the core's input terminal to test the logic inside the core. On the other hand, for the output terminal of each core, the wrapper provides a normal mode in which the host chip is driven by the core, an external test mode in which the host chip is driven by the wrapper element for interconnect test, and an internal test mode in which the wrapper element observes the core's output for the core test. Typically, suitable modifications are made to customize the test wrapper for cores that have internal scan chains or built-in self-test [2].

Genetic algorithms (GAs) are probabilistic optimization techniques based on the model of natural evolution. They are used to solve problems of high complexity. GAs use a group of randomly initialized points, called a population, in order to nondeterministically search the design space. The population is characterized by the fact that each individual encodes all necessary problem parameters (genes). The quality of an individual is measured by a fitness function. Each offspring undergoes a sequence of probabilistic modifications that changes the genetic material in the population either by inversion, crossover, mutation, or possibly other user-defined operators. The process exploits new points in the search space by providing a diverse population and avoiding premature convergence to a single local optimum. The iterative process of selecting and combining "good" individuals should yield even better ones, until a solution is found or a certain stopping criterion is met.

### A. Related work

Various test access strategies have been proposed in the literature. Reference [3] introduced a test access mechanism that connects the terminals of the embedded cores directly to the IC pins. The approach

*Haidar M. Harmanani and Rachel Sawan are with the Department of Computer Science and Mathematics, Lebanese American University, Byblos, 1401 2010, Lebanon. E-mail: haidar.harmanani@lau.edu.lb

accommodates multiple embedded cores through multiplexing. Reference [4] proposed a test bus architecture known as VisibleCores with two dedicated on-chip variable-width buses, one for transporting test control signals and the other for transporting test data signals. The approach is based on a combination of multiplexing and distribution. Reference [5] proposed a method in which test access to embedded cores is based on transparent paths through other cores. Reference [6] proposed a scalable bus-based architecture called TestRail that provides a flexible and scalable test access mechanism. The approach uses a combination of daisy chain and distribution architecture and wraps cores in a TestShell. An IC may contain one or more TestRails of varying widths, where a single TestRail can provide access to one or more cores. The width of the TestRail is referred to as the test data width, since it determines the overall system testing time. TestRail allows the system designer to trade off testing time for area overhead by varying test data widths and allowing the bus to fork out as well as to merge together; however, the designer must determine the precise relationship between the testing time and the test access mechanism. Other techniques include isolated rings [7] and the reuse of the existing test bus [8].

Reference [9] explored the relationship between testing time and test bus widths based on several integer linear programming (ILP) models with an approach that solved the problem of assigning cores to specific test buses and determining optimal widths of the test buses to minimize testing time. The author later provided an improved deserialization model [10]. Though the ILP models were aimed at optimal results, they were halted in various cases because of the problem complexity. Furthermore, the reported testing times do not conform to the reported assignments. In order to alleviate the problem complexity, heuristic techniques were proposed. For example, [11] and [12] proposed evolutionary approaches in order to solve the TAM optimization problem. The method in [11] was later extended to incorporate place and route constraints [13]. However, the above heuristic approaches tackled only the simple TAM optimization problem that reduces test time by distributing the total test bus width among individual buses. In other words, no attempts were made to find the minimum test data width with a suboptimal core assignment under a maximum test data width constraint. Furthermore, no heuristic approach was attempted to reduce the test time by exploring tradeoffs among test bus subdivision, core assignment, and test bus minimization.

### B. Problem formulation

This work proposes an efficient approach for the design of test access architectures for SoC that minimizes testing time based on the TestRail approach. Given a system with $N_c$ cores and $N_B$ test buses with a maximum width $W$, the problem is to (1) determine the optimal or suboptimal width that should be distributed among the various test buses in order to minimize test time; (2) assign the embedded cores to the test buses so that the test time is minimized; and (3) explore test time reductions by determining an optimal or a suboptimal subdivision of test buses that can test smaller cores in parallel. The problem has been proven to be $\mathcal{NP}$-complete [9]–[10]. We solve the above problem by breaking it into a progression of five incremental problems in order of increasing complexity. The proposed method is characterized by the following contributions:

- a formulation and a solution for several TAM design problems which are $\mathcal{NP}$-complete, using a genetic algorithm;
- a minimum test data width and an assignment of cores to test buses, subject to minimization of testing time;
- a tradeoff mechanism that explores possible test time reduction among core assignments, test data widths, and test time by allowing buses to fork out as well as to merge in.

In order to demonstrate the effectiveness of the proposed method, we use two academic SoC benchmarks from Duke University [10], the $\mathcal{S}_1$ system and the $d695$ system, also known as the $\mathcal{S}_2$ system. Both SoC benchmark examples are hypothetical but nontrivial and contain cores of various sizes and I/O widths. $\mathcal{S}_1$ consists of seven combinational ISCAS 85 and three sequential ISCAS 89 benchmark circuits, while $\mathcal{S}_2$ consists of two combinational ISCAS 85 and eight sequential ISCAS 89 benchmark circuits. The benchmarks are indicative and representative, as the problem's complexity depends on the number of SoC cores rather than on the size of the SoC. We use two and three test buses in order to be able to make comparisons to other works [10]–[12].

## II. Test data deserialization model

Consider an SoC consisting of $N_c$ cores and let core $i$, $1 \leq i \leq N$, have $n_i$ inputs and $m_i$ outputs (including data and scan I/Os). Let the SoC have $N_B$ test buses with widths $w_1, w_2, \ldots, w_{N_B}$. If the width of the test bus is less than the number of core terminals, then test data deserialization is needed. References [9] and [10] noted that if core $i$ is assigned to bus $j$, then the amount of data serialization needed at the I/Os of core $i$ is related to the difference between core $i$'s test width $\phi$ and the width, $w_j$, of bus $j$, where $\phi = \max\{n_i, m_i\}$.

Let $t_i$ be the testing time in cycles required by core $i$ when no deserialization is necessary. For combinational cores, $t_i$ is equal to the number of test patterns $p_i$. However, for cores with internal scan, $t_i = (p_i + 1)\lceil f_i/s_i \rceil + p_i$, where core $i$ contains $f_i$ flip-flops and $s_i$ internal scan chains [14]. The testing time with test data deserialization for core $i$ assigned to bus $j$ is given by [9]

$$T_{ij} = \begin{cases} t_i, & \text{if } \phi_i \leq w_j, \\ (\phi_i - w_j + 1)t_i, & \text{if } \phi_i > w_j. \end{cases} \tag{1}$$

This model is motivated by the need to provide parallel access to core terminals as scan-chain inputs that transport more test data. It assumes a "worst case" deserialization of test data. Thus, the first $(w_j - 1)$ test bus lines are connected to $(w_j - 1)$ core I/Os in parallel, and the last test bus line is serially connected to the remaining $(\phi_i - w_j + 1)$ core I/Os [14]–[15]. Reference [10] noted that a substantial reduction in test time can be obtained if there is a uniform distribution of test bus lines among the core I/Os. Thus, the above test data serialization becomes

$$T_{ij} = \left\lceil \frac{\phi_i}{w_j} \right\rceil t_i. \tag{2}$$

We assume that the test sets for the SoC cores are available in scan format, in which functional input values remain unchanged during successive scan cycles. This implies that the scan input values in the same cycle are specified in multiple cycles instead of being specified as independent values in the same cycle. A similar assumption was implicitly made in [14] and [15].

## III. Assignment of cores to test buses

The first problem that we address in this paper is the assignment of cores to test buses so as to minimize the system testing time. The problem is formally defined as follows:

> $\mathcal{P}_1$: Given $N_c$ cores and $N_B$ test buses of widths $w_1, w_2, \ldots, w_{N_B}$, determine an assignment of cores to test buses such that the total testing time is minimized.

Problem $\mathcal{P}_1$ is $\mathcal{NP}$-complete and has been shown to be equivalent to the multi-processor scheduling problem [9].

### A. Chromosomal representation

We solve the core assignment problem using the chromosomal representation shown in Fig. 1(a), where a chromosome represents a possible assignment of cores to test buses. Each chromosome is a vector of

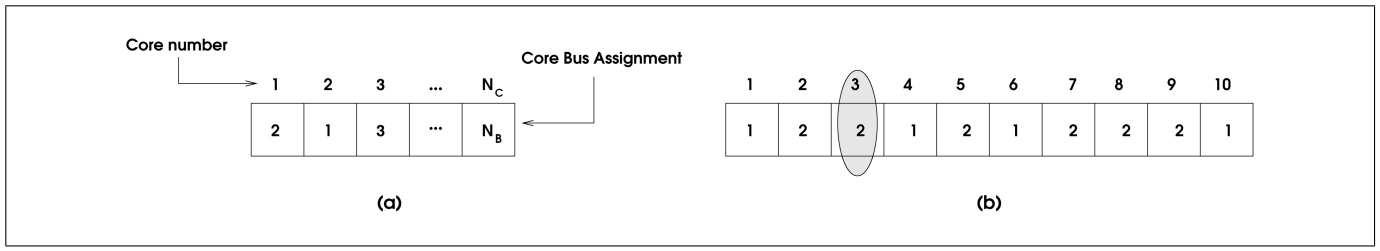


**Figure 1:** *(a) Chromosome representation for core bus assignment. (b) Sample chromosome with* 10 *cores and two buses.*

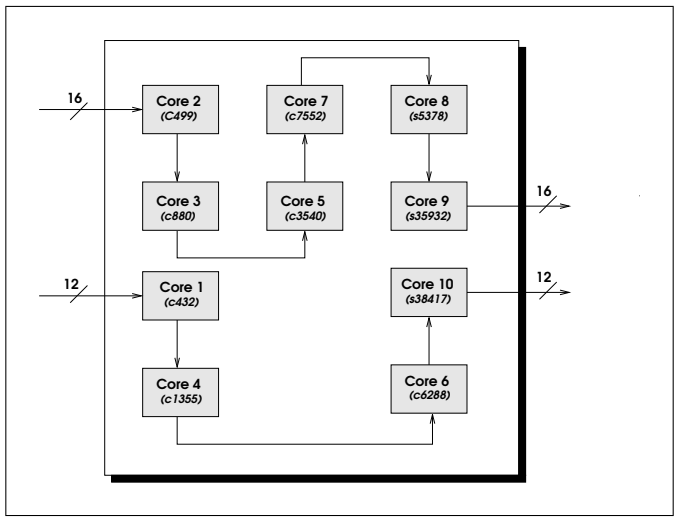


**Figure 2:** *Test bus assignment for system* $\mathcal{S}_1$ *with two test buses of* 12 *and* 16 *bits.*

size $N_c$, where a gene $j$ in position $i$ indicates that core $i$ is assigned to bus $j$, where $1 \leq i \leq N_c$ and $1 \leq j \leq N_B$.

Consider the $\mathcal{S}_1$ SoC example and let $N_B = 2$ with a total test data width of $W = 48$ bits, and let $w_1 = 16$ and $w_2 = 12$. A chromosome for a possible assignment of cores to the two test buses as proposed by our system is shown in Fig. 1(b), where the 2 in position 3 of the chromosome indicates that core 3 is assigned to bus 2. The resulting test time is 452 696 cycles, which is optimal in this case. The proposed test bus assignment is shown in Fig. 2.

### B. Initial population

The initial population is important, as it affects the quality of the final solution in addition to the time needed to converge to such a solution. In order to create the initial population, we create a single chromosome in which cores are equally assigned to all buses. For example, if $N_B = 2$, then half the cores are assigned to bus 1, and the other half are assigned to bus 2. The selection of the cores as well as their assignment is purely random. The initial population is next generated in a fast and simple procedure through $20\%$ mutation and $80\%$ crossover of this chromosome.

### C. Genetic operators

Operators are used in genetic algorithms in order to explore the design space by creating new candidate solutions from old ones. In order to explore the design space in $\mathcal{P}_1$ as well as in the remaining four problems, we use two genetic operators, mutation and crossover. The genetic operators are applied iteratively and by taking turns.

#### *1. Mutation*

Mutation is a genetic operator that is used to explore the design space. We use a standard random resetting technique in which the algorithm chooses a random core and assigns it to a random bus from the set of permissible buses with probability $P_m$.

#### *2. Crossover*

Crossover is a reproduction technique that is considered to be the most important feature in GAs. It combines two parent chromosomes in order to produce two offspring. Crossover has been known to overcome information loss during the genetic evolution process. Given two chromosomes, we apply two-point uniform crossover, in which two offspring are created from two parents by an exchange of the middle genes of both parents with probability $P_c$.

### D. Objective function

The objective function seeks to minimize the time needed to test all cores assigned to the buses; that is, minimize

$$\mathcal{T} = \max_j \sum_{i=1}^{N_c} T_{ij}, \quad 1 \leq j \leq N_B, \tag{3}$$

where $T_{ij}$ is the testing time in cycles with test data deserialization for core $i$ assigned to bus $j$ given by (1).

### E. Algorithm

The genetic algorithm was implemented using Java and tested on both benchmarks, $\mathcal{S}_1$ and $\mathcal{S}_2$. In every generation, $3n/2$ offspring are created from the current $n$ chromosomes through mutation and crossover. The best $n$ chromosomes are kept, and the algorithm repeats for the maximum number of generations.

In order to apply the genetic algorithm successfully, we empirically adjusted the crossover rate, the mutation rate, the population size $N_p$, and the number of generations $N_g$. We performed experiments on the benchmarks with a mutation and a crossover rate of $1, 0.9, \ldots, 0.1$. The population size was varied between 50 and 500. It was determined experimentally that a mutation probability $\mathrm{P}_m = 0.65$ and a crossover probability $\mathrm{P}_c = 0.35$ are effective for obtaining good solutions. The appropriate population size was determined to be 150, and the number of generations was set to 300.

The algorithm generated assignments for both SoCs in less than one second. The detailed results are shown in Tables 1 and 2. As shown, our algorithm obtained either the same test times or outperformed the ILP method in [9] for all attempted cases in a very short time. The tables also show the test improvement that our system was able to attain as a percentage. It should be noted that throughout this paper, we present the results in [9] after correction in order to reflect the reported assignments accurately.

## IV. Test bus sizing

The next problem that we address is the problem of minimizing test time by allocating a given test bus width $W$ among $N_B$ individual test buses while concurrently assigning the cores to these buses. Problem $\mathcal{P}_2$ is a generalization of $\mathcal{P}_1$ and is formally defined as follows:

> $\mathcal{P}_2$: Given $N_c$ cores and $N_B$ test buses of total width $W$, determine the optimal width of each test bus and an assignment of cores to the test buses such that the total testing time is minimized.

**Table 1**
**Assignment of cores to test buses of predetermined widths for $\mathcal{S}_1$ SoC ($\mathcal{P}_1$)**

| $(w_1, w_2)$ | ILP | | GA | | Improvement |
|---|---|---|---|---|---|
| | Test time (cycles) | Bus assignment | Test time (cycles) | Bus assignment | (%) |
| (4, 4) | 497 155 | (2, 2, 2, 1, 2, 1, 2, 2, 2, 1) | 497 155 | (2, 2, 2, 1, 2, 1, 2, 2, 2, 1) | 0 |
| (6, 6) | 487 965 | (2, 1, 2, 1, 1, 1, 1, 1, 1, 2) | 487 965 | (2, 1, 2, 1, 1, 1, 1, 1, 1, 2) | 0 |
| (8, 8) | 478 936 | (2, 2, 2, 2, 2, 2, 2, 2, 2, 1) | 478 936 | (2, 2, 2, 2, 2, 2, 2, 2, 2, 1) | 0 |
| (11, 9) | 470 380 | (2, 1, 1, 2, 2, 2, 2, 2, 2, 1) | 470 378 | (1, 1, 2, 2, 2, 2, 2, 2, 2, 1) | 0 |
| (11, 13) | 461 304 | (2, 1, 1, 1, 1, 1, 1, 1, 1, 2) | 461 304 | (2, 1, 1, 1, 1, 1, 1, 1, 1, 2) | 0 |
| (16, 12) | 452 696 | (1, 2, 2, 1, 2, 1, 2, 2, 2, 1) | 452 696 | (1, 2, 2, 1, 2, 1, 2, 2, 2, 1) | 0 |
| (18, 14) | 443 624 | (2, 1, 2, 2, 2, 2, 2, 2, 2, 1) | 443 624 | (2, 1, 2, 2, 2, 2, 2, 2, 2, 1) | 0 |
| (21, 15) | 434 954 | (1, 1, 2, 1, 2, 1, 2, 2, 2, 1) | 434 954 | (1, 1, 2, 1, 2, 1, 2, 2, 2, 1) | 0 |
| (17, 23) | 439 575 | (2, 2, 2, 1, 1, 1, 2, 1, 1, 2) | 426 006 | (1, 1, 1, 2, 1, 2, 1, 1, 1, 2) | 3.08 |
| (25, 19) | 416 992 | (2, 2, 2, 2, 2, 1, 2, 2, 2, 1) | 416 992 | (2, 2, 2, 2, 2, 1, 2, 2, 2, 1) | 0 |
| (28, 20) | 408 023 | (1, 1, 2, 1, 2, 1, 2, 2, 2, 1) | 408 023 | (1, 1, 2, 1, 2, 1, 2, 2, 2, 1) | 0 |
| (22, 30) | 427 752 | (2, 2, 2, 2, 2, 2, 2, 2, 2, 1) | 399 281 | (1, 2, 1, 1, 1, 2, 1, 1, 1, 2) | 6.66 |
| (32, 16) | 411 884 | (1, 1, 1, 1, 1, 1, 1, 2, 2, 1) | 411 884 | (1, 1, 1, 1, 1, 1, 1, 2, 2, 1) | 0 |
| (32, 24) | 391 192 | (2, 2, 2, 2, 2, 2, 2, 2, 2, 1) | 391 192 | (2, 2, 2, 2, 2, 2, 2, 2, 2, 1) | 0 |
| (32, 28) | 391 192 | (2, 2, 2, 2, 2, 2, 2, 2, 2, 1) | 391 192 | (2, 2, 2, 2, 2, 2, 2, 2, 2, 1) | 0 |
| (32, 32) | 391 192 | (2, 2, 2, 2, 2, 2, 2, 2, 2, 1) | 391 192 | (2, 2, 2, 2, 2, 2, 2, 2, 2, 1) | 0 |

**Table 2**
**Assignment of cores to test buses of predetermined widths for $\mathcal{S}_2$ SoC ($\mathcal{P}_1$)**

| $(w_1, w_2)$ | ILP | | GA | | Improvement |
|---|---|---|---|---|---|
| | Test time (cycles) | Bus assignment | Test time (cycles) | Bus assignment | (%) |
| (15, 1) | 2 520 052 | (2, 2, 2, 1, 2, 1, 2, 1, 1, 1) | 2 470 746 | (2, 1, 1, 1, 2, 1, 1, 2, 1, 2) | 1.95 |
| (1, 19) | 2 437 081 | (2, 2, 1, 2, 1, 2, 1, 2, 2, 2) | 2 423 116 | (1, 1, 1, 2, 1, 2, 1, 2, 2, 2) | 0.57 |
| (23, 1) | 2 374 783 | (2, 1, 1, 1, 2, 1, 2, 1, 1, 1) | 2 361 278 | (2, 2, 1, 1, 2, 1, 2, 1, 1, 1) | 0.56 |
| (3, 29) | 2 275 329 | (2, 2, 2, 2, 1, 2, 2, 2, 2, 1) | 2 255 946 | (2, 2, 2, 2, 1, 2, 1, 2, 2, 2) | 0.85 |
| (4, 32) | 2 258 586 | (2, 2, 2, 2, 1, 2, 2, 1, 1, 2) | 2 195 730 | (1, 2, 2, 2, 1, 2, 2, 2, 2, 1) | 2.78 |
| (9, 31) | 2 222 271 | (2, 2, 2, 2, 1, 2, 2, 2, 2, 1) | 2 205 162 | (2, 2, 2, 2, 1, 2, 1, 2, 2, 2) | 0.77 |
| (12, 32) | 2 195 742 | (2, 2, 2, 2, 1, 2, 2, 2, 2, 1) | 2 179 770 | (2, 2, 2, 2, 1, 2, 1, 2, 2, 2) | 0.72 |
| (32, 16) | 2 195 730 | (2, 1, 1, 1, 2, 1, 1, 1, 1, 2) | 2 145 914 | (1, 1, 1, 1, 2, 1, 2, 1, 1, 1) | 2.27 |
| (32, 20) | 2 182 882 | (2, 2, 1, 1, 2, 1, 1, 1, 1, 2) | 2 125 782 | (1, 2, 1, 1, 2, 1, 2, 1, 1, 1) | 2.62 |
| (25, 31) | 2 160 295 | (2, 2, 2, 2, 1, 2, 1, 2, 2, 2) | 2 132 308 | (1, 1, 1, 2, 1, 2, 1, 2, 2, 2) | 1.30 |
| (28, 32) | 2 133 469 | (2, 2, 2, 2, 1, 2, 1, 2, 2, 2) | 2 100 443 | (2, 2, 2, 2, 1, 2, 2, 1, 2, 1) | 1.55 |
| (32, 32) | 2 114 206 | (2, 2, 1, 2, 2, 1, 1, 1, 1, 2) | 2 072 754 | (1, 2, 2, 1, 2, 1, 1, 2, 1, 2) | 1.96 |

In order to solve $\mathcal{P}_2$, which is an $\mathcal{NP}$-complete problem, the chromosomal representation is extended by the addition of one extra component in the chromosome, as shown in Fig. 3(b). The additional representation in part (b) of the figure is based on a vector, where a gene $w_j$ in the $i$-th position indicates that bus $i$ is assigned $w_j$ bits. The chromosome now has two related parts: the first part assigns cores to test buses, and the second part allocates a certain width to each test bus. The cost function to minimize is

$$\mathcal{T} = \max_j \sum_{i=1}^{N_c} T_{ij}, \quad 1 \leq j \leq N_B, \tag{4}$$

under the following additional constraints:

$$\sum_{j=1}^{N_B} w_j = W, \tag{5}$$

where $w_j \leq \phi_i$ for $1 \leq j \leq N_B$ and $1 \leq i \leq N_C$.

If the equality in (5) is violated, the system iterates over chromosomal part (b), randomly chooses a bus, and reduces its width by one bit. The process continues until the equality is met.

Solving $\mathcal{P}_2$ requires an additional operator, inversion, that randomly selects a gene $w_j$ from chromosomal part (b) and sets its value to $(W - w_j)$. While the crossover operator does not require any modifications in order to handle part (b), we modify the mutation operator such that it can explore the assignment of bus width in part (b) based on the core's test width, $\phi$.

The initial population is generated in a way similar to that for $\mathcal{P}_1$. Thus, we create a single chromosome in which the buses are equally assigned to all cores in part (a). Part (b) is created by assigning a test bus width $w_j$ to each gene in the chromosome, where $w_j = \min_i\{\phi_i\}$ and $\sum_{j=0}^{N_B} w_j \leq W$.

Ten percent of the members in the initial population match the above chromosome. Another $50\%$ of the chromosomes are created through mutation of this single chromosome, and the last $40\%$ are created through inversion.

Another problem that is closely related to $\mathcal{P}_2$ is that of determining the minimum system test width $W$ needed to satisfy testing time constraints while finding a suboptimal distribution of the test bus width among individual test buses, and a suboptimal test bus assignment. Problem $\mathcal{P}_3$ is formally defined as follows:

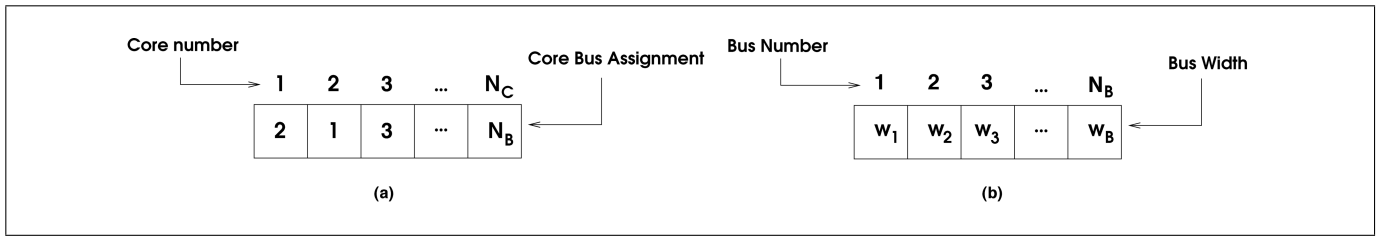


**Figure 3:** *Chromosome representation: (a) core bus assignment; (b) bus width.*

**Table 3**
**Test time and width distribution with two test buses and a given test width for $\mathcal{S}_1$ SoC ($\mathcal{P}_2$)**

| Width ($W$) | ILP | | | GA | | | Improvement (%) |
|---|---|---|---|---|---|---|---|
| | $(w_1, w_2)$ | Test time (cycles) | Bus assignment | $(w_1, w_2)$ | Test time (cycles) | Bus assignment | |
| 8 | (4, 4) | 497 155 | (2, 2, 2, 1, 2, 1, 2, 2, 2, 1) | (4, 4) | 497 155 | (2, 2, 2, 1, 2, 1, 2, 2, 2, 1) | 0 |
| 12 | (6, 6) | 487 965 | (2, 1, 2, 1, 1, 1, 1, 1, 1, 2) | (6, 6) | 487 965 | (2, 1, 2, 1, 1, 1, 1, 1, 1, 2) | 0 |
| 16 | (8, 8) | 478 936 | (2, 2, 2, 2, 2, 2, 2, 2, 2, 1) | (8, 8) | 478 936 | (2, 2, 2, 2, 2, 2, 2, 2, 2, 1) | 0 |
| 20 | (11, 9) | 470 380 | (2, 1, 1, 2, 2, 2, 2, 2, 2, 1) | (11, 9) | 470 378 | (1, 1, 2, 2, 2, 2, 2, 2, 2, 1) | 0 |
| 24 | (11, 13) | 461 304 | (2, 1, 1, 1, 1, 1, 1, 1, 1, 2) | (11, 13) | 461 304 | (2, 1, 1, 1, 1, 1, 1, 1, 1, 2) | 0 |
| 28 | (16, 12) | 452 696 | (1, 2, 2, 1, 2, 1, 2, 2, 2, 1) | (16, 12) | 452 696 | (1, 2, 2, 1, 2, 1, 2, 2, 2, 1) | 0 |
| 32 | (18, 14) | 443 624 | (2, 1, 2, 2, 2, 2, 2, 2, 2, 1) | (18, 14) | 443 624 | (2, 1, 2, 2, 2, 2, 2, 2, 2, 1) | 0 |
| 36 | (21, 15) | 434 954 | (1, 1, 2, 1, 2, 1, 2, 2, 2, 1) | (21, 15) | 434 954 | (1, 1, 2, 1, 2, 1, 2, 2, 2, 1) | 0 |
| 40 | (17, 23) | 439 975 | (2, 2, 2, 1, 1, 1, 2, 1, 1, 2) | (17, 23) | 426 006 | (1, 1, 1, 2, 1, 2, 1, 1, 1, 2) | 3.17 |
| 44 | (25, 19) | 416 992 | (2, 2, 2, 2, 2, 1, 2, 2, 2, 1) | (1, 43) | 400 553 | (1, 1, 1, 1, 1, 1, 1, 2, 1, 2) | 3.94 |
| 48 | (28, 20) | 408 023 | (1, 1, 2, 1, 2, 1, 2, 2, 2, 1) | (1, 47) | 367 901 | (1, 1, 1, 1, 1, 1, 1, 2, 1, 2) | 9.83 |
| 52 | (22, 30) | 427 752 | (2, 2, 2, 2, 2, 2, 2, 2, 2, 1) | (1, 51) | 335 249 | (1, 1, 1, 1, 1, 1, 1, 2, 1, 2) | 21.63 |
| 56 | (32, 24) | 391 192 | (2, 2, 2, 2, 2, 2, 2, 2, 2, 1) | (55, 1) | 311 611 | (2, 2, 2, 2, 2, 2, 2, 1, 2, 1) | 20.34 |
| 60 | (32, 28) | 391 192 | (2, 2, 2, 2, 2, 2, 2, 2, 2, 1) | (1, 59) | 296 987 | (1, 1, 1, 1, 1, 1, 1, 2, 1, 2) | 24.08 |
| 64 | (32, 32) | 391 192 | (2, 2, 2, 2, 2, 2, 2, 2, 2, 1) | (63, 1) | 282 363 | (2, 2, 2, 2, 2, 2, 2, 1, 2, 1) | 27.82 |

$\mathcal{P}_3$: Given $N_c$ cores, $N_B$ test buses, and a maximum testing time $\mathcal{T}$, determine the minimum total test width, an optimal distribution of the test width among the test buses, and an optimal assignment of cores to test buses.

The problem representation and parameters are the same as in $\mathcal{P}_2$; the difference is that the objective function now aims to minimize

$$W = \sum_{j=1}^{N_B} w_j \tag{6}$$

under time constraint $\mathcal{T}$ and such that

$$\sum_{i=1}^{N_C} T_{ij} \leq \mathcal{T} \quad \text{and} \quad 1 \leq j \leq N_B. \tag{7}$$

We solved problems $\mathcal{P}_2$ and $\mathcal{P}_3$ for several values of $W$ and $\mathcal{T}$. The running time was equal to 4 CPU seconds. Tables 3 and 4 show, respectively, the minimum test time and width distribution for fixed values of $W$ with $N_B = 2$, while Tables 5 and 6 show the same results for $N_B = 3$. Tables 7 and 8 show result comparisons with [9], [11], and [12] for $\mathcal{P}_2$ using two buses, while Tables 9 and 10 compare the proposed method with [9], [11], and [12] for three buses. We note that for smaller values of $W$, our algorithm obtains the same results as the ILP method [9]. However, for larger values of $W$, our algorithm yields considerable improvements that vary between $9.83\%$ for 48 bits and $27.82\%$ for 64 bits. For the other cases, our system showed either slight improvements or the same results as [11] and [12] except for one case in Table 5, where [11] reports improvements in the case of $\mathcal{S}_1$ only.

Table 11 presents the results for test bus width and width distribution for a given maximum test time under time constraint $\mathcal{T}$ for problem $\mathcal{P}_3$. For large $W$ or smaller $\mathcal{T}$, our algorithm obtains substantial improvements. For example, for $\mathcal{T} = 400\,000$, our algorithm saves 7 bits of the TAM, representing a $13.46\%$ improvement in bus width, and for $\mathcal{T} = 410\,000$, there is a 5-bit improvement or a $10.42\%$ decrease in bus width. For $\mathcal{T} = 420\,000$, there is a 1-bit saving. It was not possible to do further comparisons for problem $\mathcal{P}_3$, since this problem was not tackled by researchers other than [9]. .

The problem parameters were determined empirically as well. The population size was chosen to be 150, and the number of generations was set to 300. The probability of crossover was 0.35, and mutation probability was 0.65.

## V. Test bus subdivision

In order to explore further test time improvements, we consider trading off test assignment, test data widths, and test time by taking advantage of the TestRail flexibility that allows test buses to fork out and merge together. Thus, test buses may fork out into a set of smaller test buses that transport, in parallel, test data to smaller cores. Doing so reduces testing time, especially when several small cores with small test widths are assigned to a wide test bus. Larger cores can still be assigned to the undivided part of the test bus as before. Problem $\mathcal{P}_4$ is formally defined as follows:

$\mathcal{P}_4$: Given $N_c$ cores, $N_B$ test buses with known widths $w_1, w_2, \ldots, w_{N_B}$, and an upper limit $j_{\max}$ on the number of subdivisions allowed for test bus $j$, $1 \leq j \leq N_B$, determine (1) an optimal subdivision of test bus widths and (2) an optimal assignment of cores to test buses such that the total testing time is minimized.

**Table 4**
**Test time and width distribution with two test buses and a given test width for $\mathcal{S}_2$ SoC ($\mathcal{P}_2$)**

| Width ($W$) | ILP | | | GA | | | Improvement (%) |
|---|---|---|---|---|---|---|---|
| | $(w_1, w_2)$ | Test time (cycles) | Bus assignment | $(w_1, w_2)$ | Test time (cycles) | Bus assignment | |
| 16 | (15, 1) | 2 520 052 | (2, 2, 2, 1, 2, 1, 2, 1, 1, 1) | (13, 3) | 2 456 793 | (2, 1, 1, 1, 2, 1, 1, 1, 2, 2) | 2.51 |
| 20 | (1, 19) | 2 437 081 | (2, 2, 1, 2, 1, 2, 1, 2, 2, 2) | (1, 19) | 2 423 116 | (1, 1, 1, 2, 1, 2, 1, 2, 2, 2) | 0.57 |
| 24 | (23, 1) | 2 374 783 | (2, 1, 1, 1, 2, 1, 2, 1, 1, 1) | (23, 1) | 2 361 278 | (2, 2, 1, 1, 2, 1, 2, 1, 1, 1) | 0.57 |
| 32 | (3, 29) | 2 275 329 | (2, 2, 2, 2, 1, 2, 2, 2, 2, 1) | (1, 31) | 2 222 247 | (1, 2, 2, 2, 1, 2, 2, 2, 2, 1) | 2.33 |
| 36 | (4, 32) | 2 258 586 | (2, 2, 2, 2, 1, 2, 2, 1, 1, 2) | (4, 32) | 2 195 730 | (1, 2, 2, 2, 1, 2, 2, 2, 2, 1) | 2.78 |
| 40 | (9, 31) | 2 222 271 | (2, 2, 2, 2, 1, 2, 2, 2, 2, 1) | (1, 39) | 2 144 192 | (1, 1, 1, 2, 1, 2, 2, 2, 1, 2) | 3.51 |
| 44 | (12, 32) | 2 195 742 | (2, 2, 2, 2, 1, 2, 2, 2, 2, 1) | (1, 43) | 2 039 183 | (2, 2, 1, 1, 2, 1, 1, 1, 2, 1) | 7.13 |
| 48 | (32, 16) | 2 195 730 | (2, 1, 1, 1, 2, 1, 1, 1, 1, 2) | (1, 47) | 1 966 608 | (2, 2, 2, 2, 1, 2, 2, 2, 1, 2) | 10.43 |
| 48 | (32, 16) | 2 195 730 | (2, 1, 1, 1, 2, 1, 1, 1, 1, 2) | (2, 46) | 1 975 827 | (2, 1, 2, 2, 1, 2, 2, 2, 1, 2) | 10.02 |
| 52 | (32, 20) | 2 182 882 | (2, 2, 1, 1, 2, 1, 1, 1, 1, 2) | (4, 48) | 1 949 151 | (2, 2, 2, 2, 1, 2, 2, 2, 1, 2) | 10.71 |
| 56 | (25, 31) | 2 160 295 | (2, 2, 2, 2, 1, 2, 1, 2, 2, 2) | (7, 49) | 1 931 694 | (2, 2, 2, 2, 1, 2, 2, 2, 1, 2) | 10.58 |
| 60 | (28, 32) | 2 133 469 | (2, 2, 2, 2, 1, 2, 1, 2, 2, 2) | (10, 50) | 1 914 237 | (2, 2, 2, 2, 1, 2, 2, 2, 1, 2) | 10.27 |
| 64 | (32, 32) | 2 114 206 | (2, 2, 1, 2, 2, 1, 1, 1, 1, 2) | (50, 14) | 1 902 625 | (2, 1, 1, 1, 2, 1, 1, 1, 2, 1) | 10.01 |
| 64 | (32, 32) | 2 114 206 | (2, 2, 1, 2, 2, 1, 1, 1, 1, 2) | (12, 48) | 1 933 403 | (1, 1, 2, 2, 1, 2, 2, 2, 1, 2) | 8.55 |
| 64 | (32, 32) | 2 114 206 | (2, 2, 1, 2, 2, 1, 1, 1, 1, 2) | (1, 63) | 1 865 686 | (1, 1, 1, 2, 1, 2, 2, 2, 2, 2) | 11.75 |

**Table 5**
**Test time and width distribution with three test buses and a given test width for $\mathcal{S}_1$ SoC ($\mathcal{P}_2$)**

| Width ($W$) | Bus assignment | Bus distribution | Test time (cycles) |
|---|---|---|---|
| 16 | (2, 2, 2, 2, 2, 2, 3, 3, 2, 1) | (14, 1, 1) | 457 000 |
| 20 | (3, 2, 2, 2, 2, 3, 2, 2, 3, 1) | (18, 1, 1) | 442 376 |
| 24 | (2, 3, 3, 2, 2, 3, 2, 3, 2, 1) | (22, 1, 1) | 427 752 |
| 28 | (2, 2, 2, 2, 2, 2, 2, 3, 2, 1) | (26, 1, 1) | 413 128 |
| 32 | (3, 3, 3, 3, 2, 2, 3, 2, 3, 1) | (30, 1, 1) | 398 504 |
| 36 | (2, 3, 3, 2, 2, 2, 3, 3, 2, 1) | (34, 1, 1) | 383 880 |
| 40 | (3, 2, 2, 2, 2, 3, 2, 2, 3, 1) | (38, 1, 1) | 369 256 |
| 44 | (1, 1, 3, 3, 1, 3, 3, 3, 1, 2) | (1, 42, 1) | 354 632 |
| 48 | (3, 2, 2, 3, 3, 2, 3, 2, 3, 1) | (46, 1, 1) | 340 008 |
| 52 | (3, 3, 2, 2, 2, 2, 2, 3, 2, 1) | (50, 1, 1) | 325 384 |
| 56 | (2, 3, 3, 3, 3, 2, 2, 2, 3, 1) | (54, 1, 1) | 310 760 |
| 60 | (1, 1, 1, 2, 2, 2, 2, 1, 2, 3) | (1, 1, 58) | 296 136 |
| 64 | (1, 1, 1, 1, 2, 2, 1, 1, 2, 3) | (1, 1, 62) | 281 512 |

**Table 6**
**Test time and width distribution with three test buses and a given test width for $\mathcal{S}_2$ SoC ($\mathcal{P}_2$)**

| Width ($W$) | Bus assignment | Bus distribution | Test time (cycles) |
|---|---|---|---|
| 16 | (2, 1, 2, 1, 3, 2, 1, 1, 1, 1) | (8, 3, 5) | 1 694 860 |
| 20 | (2, 2, 3, 3, 1, 2, 3, 3, 3, 3) | (8, 1, 11) | 1 684 002 |
| 24 | (3, 2, 3, 3, 1, 2, 3, 3, 3, 3) | (11, 1, 12) | 1 664 230 |
| 28 | (3, 2, 3, 3, 1, 2, 3, 3, 3, 3) | (14, 1, 13) | 1 648 915 |
| 32 | (3, 2, 3, 3, 1, 2, 3, 3, 3, 3) | (17, 1, 14) | 1 633 623 |
| 36 | (3, 3, 2, 2, 2, 1, 3, 3, 3, 3) | (1, 34, 1) | 1 618 512 |
| 40 | (3, 3, 1, 1, 1, 2, 3, 3, 3, 3) | (35, 4, 1) | 1 598 231 |
| 44 | (3, 3, 2, 2, 2, 1, 3, 3, 3, 3) | (6, 37, 1) | 1 574 068 |
| 48 | (1, 3, 2, 2, 2, 1, 3, 3, 3, 3) | (7, 38, 2) | 1 563 240 |
| 52 | (2, 3, 1, 1, 1, 2, 3, 3, 3, 3) | (40, 9, 3) | 1 542 814 |
| 56 | (1, 1, 2, 2, 2, 1, 3, 3, 3, 3) | (12, 41, 3) | 1 530 756 |
| 60 | (3, 3, 2, 2, 2, 1, 3, 3, 3, 3) | (14, 41, 5) | 1 530 756 |
| 64 | (1, 3, 2, 2, 2, 3, 1, 1, 1, 1) | (5, 45, 14) | 1 507 432 |

Finally, we formulate the general case of the test bus subdivision problem, where the test bus widths must be determined as well. Problem $\mathcal{P}_5$ is formally defined as follows:

> $\mathcal{P}_5$: Given $N_c$ cores, $N_B$ test buses of total width $W$, and an upper limit $j_{\max}$ on the number of subdivisions allowed for test bus $j$, $1 \leq j \leq N_B$, determine (1) an optimal width for each test bus, the optimal subdivision of the width of every bus, and (2) an assignment of cores to test buses such that the total testing time is minimized.

The main difference between $\mathcal{P}_4$ and $\mathcal{P}_5$ is that in $\mathcal{P}_5$ the restriction is on the total width of $W$ rather than on the width of each test bus; this gives the problem a higher degree of freedom.

### A. Chromosomal representation

In order to solve $\mathcal{P}_4$, we need to extend the chromosomal representation to incorporate bus subdivisions. The extended chromosomal representation, shown in Fig. 4, is of variable length and includes three related parts. Part (a) of the chromosome represents a possible assignment of cores to either test buses or to a test subdivision. Thus, a gene $j$ in position $i$ indicates that core $i$ is assigned to either a bus or a subdivision $j$, where $1 \leq i \leq N_c$ and $1 \leq j \leq N_B + N_S$, where $N_S$ is the total number of subdivisions. Part (b) of the chromosome allocates a width to each bus or subdivision and is based on a vector, where a gene $w_j$ in the $i$-th position indicates that bus $i$ or subdivision $i$ is assigned $w_j$ bits. Finally, part (c) of the chromosome assigns the number of bus subdivisions, which should be less than or equal to $j_{\max}$. A gene $i$ in the $j$-th position in part (c) indicates that bus $j$ forks into $i$ subdivisions.

A sample chromosome is shown in Fig. 5 with five cores and three buses, with bus 2 having two subdivisions. For example, core 2 in Fig. 5(a) is assigned to bus 2, while Fig. 5(c) indicates that bus 2 has two subdivisions, 2 and 3, that are allocated 10 and 20 bits respectively, as shown in Fig. 5(b).

Based on this representation, a chromosome grows and shrinks according to the number of subdivisions in part (c), with the result that the chromosome has variable length. The algorithm invokes the

**Table 7**
**Test time comparison of proposed algorithm with other works for $\mathcal{S}_1$ SoC with two test buses ($\mathcal{P}_2$)**

| Width ($W$) | Test time (cycles) | | | |
|---|---|---|---|---|
| | ILP [9] | Ebadi [11] | Wang [12] | GA |
| 20 | 470 380 | n/a | 470 378 | 470 378 |
| 24 | 461 304 | n/a | 461 304 | 461 304 |
| 28 | 452 696 | 452 696 | 452 696 | 452 696 |
| 32 | 443 624 | 443 624 | 443 624 | 443 624 |
| 36 | 434 954 | 434 954 | 434 954 | 434 954 |
| 40 | 439 975 | 426 006 | 426 006 | 426 006 |
| 44 | 416 992 | 400 553 | 400 553 | 400 553 |
| 48 | 408 023 | 367 901 | 367 901 | 367 901 |
| 52 | 427 752 | 335 249 | 335 249 | 335 249 |
| 56 | 391 192 | 311 611 | 311 611 | 311 611 |
| 60 | 391 192 | 296 987 | n/a | 296 987 |
| 64 | 391 192 | 282 363 | n/a | 282 363 |

**Table 8**
**Test time comparison of proposed algorithm with other works for $\mathcal{S}_2$ SoC with two test buses ($\mathcal{P}_2$)**

| Width ($W$) | Test time (cycles) | | | |
|---|---|---|---|---|
| | ILP [9] | Ebadi [11] | Wang [12] | GA |
| 16 | 2 520 052 | 2 477 722 | n/a | 2 456 793 |
| 20 | 2 437 081 | 2 423 284 | 2 323 116 | 2 423 116 |
| 24 | 2 374 783 | 2 361 278 | 2 301 278 | 2 361 278 |
| 32 | 2 275 329 | 2 222 247 | 2 202 247 | 2 222 247 |
| 36 | 2 258 586 | 2 195 730 | 2 174 501 | 2 195 730 |
| 40 | 2 222 271 | 2 144 192 | 2 144 192 | 2 144 192 |
| 44 | 2 195 742 | 2 039 183 | 2 039 183 | 2 039 183 |
| 48 | 2 195 730 | 1 966 608 | 1 966 608 | 1 966 608 |
| 52 | 2 182 882 | 1 949 151 | 1 949 151 | 1 949 151 |
| 56 | 2 160 295 | 1 931 694 | 1 931 694 | 1 931 694 |
| 60 | 2 133 469 | n/a | 1 914 237 | 1 914 237 |

*reduceSlots* operation in order to decrease the size of part (b). At the same time, if the number of subdivisions increases in part (c), then the algorithm increases the size of part (b) using the *addSlots* operation.

### B. Objective function

The objective function in $\mathcal{P}_4$ and $\mathcal{P}_5$ aims to minimize the time needed to test all cores assigned to the test buses; that is, minimize

$$\mathcal{T} = \max_j \sum_{i=1}^{N_c} T_{ij}, \quad 1 \le j \le N_B, \tag{8}$$

where $T_{ij}$ is given by (1) and $T_{ij}$ is the testing time with test data deserialization for core $i$ assigned to bus $j$. Problem $\mathcal{P}_5$ has constraints on the total width only:

$$W = \sum_{j=1}^{N_B} w_j. \tag{9}$$

### C. Genetic operator

To explore the design space for problems $\mathcal{P}_4$ and $\mathcal{P}_5$, we use the same genetic operators as in the previous three problems. However, we extend the mutation operator in order to accommodate part (c) as shown in Fig. 6. To achieve a good hill-climbing effect, we define two types of mutations. The first mutates part (a) only, while fixing parts (b) and (c), and the second mutates all three parts concurrently.

**Table 9**
**Test time comparison of proposed algorithm with other works for $\mathcal{S}_1$ SoC with three test buses ($\mathcal{P}_2$)**

| Width ($W$) | Test time (cycles) | | |
|---|---|---|---|
| | Ebadi [11] | Wang [12] | GA |
| 16 | n/a | n/a | 457 000 |
| 20 | n/a | n/a | 442 376 |
| 24 | n/a | 422 752 | 427 752 |
| 28 | 409 742 | n/a | 413 128 |
| 32 | 394 848 | n/a | 398 504 |
| 36 | 380 224 | 398 504 | 383 880 |
| 40 | 365 600 | n/a | 369 256 |
| 44 | 350 976 | 354 632 | 354 632 |
| 48 | 336 352 | n/a | 340 008 |
| 52 | 321 728 | 325 384 | 325 384 |
| 56 | 307 104 | n/a | 310 760 |
| 60 | 292 480 | n/a | 296 136 |
| 64 | 277 856 | n/a | 281 512 |

**Table 10**
**Test time comparison of proposed algorithm with other works for $\mathcal{S}_2$ SoC with three test buses ($\mathcal{P}_2$)**

| Width ($W$) | Test time | | |
|---|---|---|---|
| | Ebadi [11] | Wang [12] | GA |
| 16 | n/a | n/a | 1 694 860 |
| 20 | n/a | n/a | 1 684 002 |
| 24 | n/a | 1 669 335 | 1 664 230 |
| 28 | 409 742 | n/a | 1 648 915 |
| 32 | 394 848 | n/a | 1 633 623 |
| 36 | 380 224 | 1 618 512 | 1 618 512 |
| 40 | 365 600 | 1 595 724 | 1 598 231 |
| 44 | 350 976 | n/a | 1 574 068 |
| 48 | 336 352 | 1 565 027 | 1 563 240 |
| 52 | 321 728 | n/a | 1 542 814 |
| 56 | 307 104 | n/a | 1 530 756 |
| 60 | 292 480 | n/a | 1 530 756 |
| 64 | 277 856 | n/a | 1 507 432 |

### D. Repair functions

Our genetic algorithm generates infeasible chromosomes for the following reasons:

1. The variable-length nature of the chromosome causes part (c) to change in order to explore the number of subdivisions, thus changing the number of genes in part (b). Furthermore, if the number of subdivisions is reduced in part (b), then some of the subdivisions will cease to exist, and some of the assignments in part (a) will become invalid. For example, if the number of subdivisions in chromosomal part (c) is decreased from seven to six, then cores in part (a) that are assigned to subdivision 7 will become invalid.
2. The crossover operator may generate infeasible chromosomes in $\mathcal{P}_4$ only because of restrictions on the width of individual buses (9), where the sum of the width in the subdivisions is more than the width of bus $w_j$.

Typically, there are two strategies to deal with infeasible chromosomes: the first repairs them, and the second imposes a penalty function. We choose the first approach. Thus, the first part is repaired by reassigning cores that have been assigned to invalid subdivisions to random valid ones. We repair the second part by randomly reducing

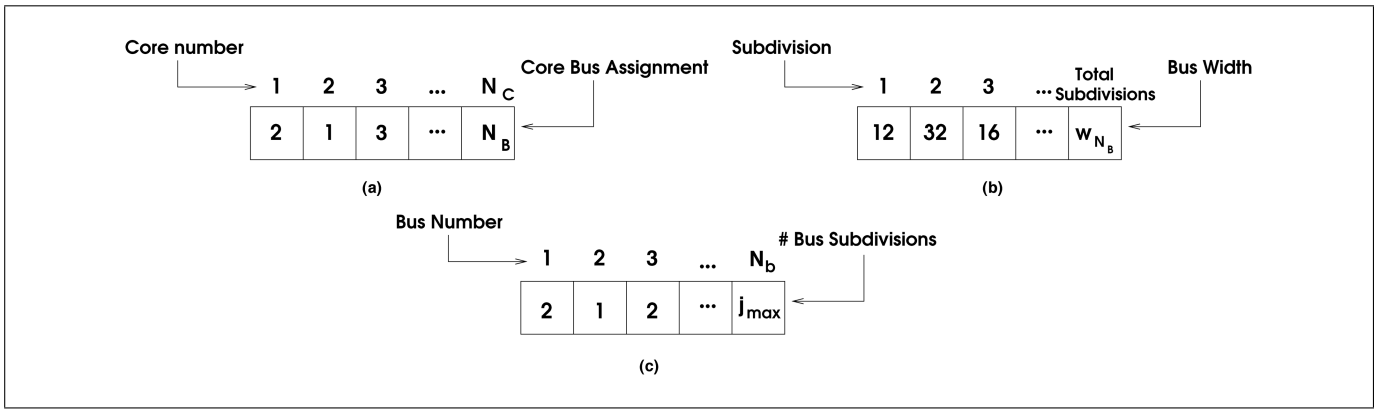


**Figure 4:** *(a) Chromosome representation. (b) Core bus assignment. (c) Bus subdivisions.*

**Table 11**
**Width distribution for $\mathcal{S}_1$ SoC with two test buses and a given maximum time ($\mathcal{P}_3$)**

| Max $\mathcal{T}$ | ILP | | | GA | | | Improvement |
|---|---|---|---|---|---|---|---|
| | Width ($W$) | $(w_1, w_2)$ | Bus assignment | Width ($W$) | $(w_1, w_2)$ | Bus assignment | (%) |
| 400 000 | 52 | (21, 31) | (2, 2, 2, 2, 2, 2, 1, 1, 1, 2) | 45 | (44, 1) | (1, 1, 1, 2, 2, 2, 2, 1, 2, 1) | 13.46 |
| 410 000 | 48 | (27, 21) | (2, 2, 2, 2, 2, 2, 2, 2, 2, 1) | 43 | (42, 1) | (2, 1, 2, 2, 1, 2, 2, 1, 2, 1) | 10.42 |
| 420 000 | 43 | (25, 18) | (2, 2, 2, 2, 1, 1, 2, 2, 2, 1) | 42 | (41, 1) | (1, 1, 1, 2, 1, 1, 2, 1, 2, 1) | 2.32 |
| 430 000 | 39 | (22, 17) | (2, 2, 2, 2, 2, 2, 2, 2, 2, 1) | 39 | (22, 17) | (1, 2, 1, 2, 2, 2, 2, 2, 2, 1) | 0 |
| 440 000 | 34 | (19, 15) | (2, 2, 2, 2, 2, 2, 2, 2, 2, 1) | 34 | (20, 14) | (2, 1, 1, 2, 1, 2, 2, 2, 2, 1) | 0 |
| 450 000 | 30 | (16, 14) | (2, 2, 2, 2, 2, 2, 2, 2, 2, 1) | 30 | (18, 12) | (1, 1, 1, 1, 1, 1, 2, 2, 2, 1) | 0 |
| 460 000 | 25 | (14, 11) | (2, 2, 2, 1, 2, 2, 2, 2, 2, 1) | 25 | (11, 14) | (1, 1, 2, 2, 2, 2, 2, 2, 2, 1) | 0 |
| 470 000 | 21 | (11, 10) | (2, 2, 2, 2, 2, 2, 2, 2, 2, 1) | 21 | (13, 8) | (1, 1, 2, 1, 1, 1, 2, 2, 2, 1) | 0 |
| 480 000 | 16 | (8, 8) | (2, 2, 2, 2, 2, 2, 2, 2, 2, 1) | 16 | (8, 8) | (2, 2, 2, 2, 2, 1, 2, 2, 2, 1) | 0 |
| 480 000 | 16 | (8, 8) | (2, 2, 2, 2, 2, 2, 2, 2, 2, 1) | 16 | (7, 9) | (1, 1, 1, 1, 2, 2, 1, 1, 1, 2) | 0 |

the number of bits in the subdivisions as shown in Fig. 7. The method iterates over an invalid part (b) and randomly chooses a bus in every iteration. The bus width is decremented, and the method repeats the process until the chromosome is fully repaired and (5) is satisfied.

### E. Initial population

The initial population is important, as it affects the quality of the final solution in addition to the time needed to converge to such a solution. The initial population is generated on the basis of three initial single chromosomes, in a way similar to that for the previous three problems. Part (a) of the single chromosome is created such that cores are assigned equally to all buses, and part (b) is created by assigning to each gene in the chromosome a test bus width $w_j$, where $w_j = \min_i\{\phi_i\}$ and $\sum_{j=0}^{N_B} w_j \leq W$. The selection of the cores, test bus widths, and assignment is purely random. Part (c) is created on the basis of three categories:

1. The first category includes a chromosome such that each bus has the maximum number of subdivisions allowed, $j_{\max}$.
2. The second category includes a chromosome such that buses are not subdivided.
3. The third category includes a chromosome such that the number of subdivisions for each bus is randomly chosen between 1 and $j_{\max}$.

The initial population is next generated with $10\%$ of its members from the first category, $10\%$ from the second category, and $20\%$ from the last category. The remaining chromosomes are generated through mutations, of which $40\%$ are mutations of parts (a) and (b), while the remaining $20\%$ are mutations of all three parts.

### F. Experimental results

We solved problems $\mathcal{P}_4$ and $\mathcal{P}_5$ using our genetic algorithm. The algorithm has a running-time order of $O(N_g \times N_B \times j_{\max})$, where $N_g$ is the number of generations, $N_B$ is the number of test buses, and $j_{\max}$ is the number of bus subdivisions. The results, shown in Tables 12 to 15, were generated in less than 20 seconds and provide various TAM architectures for the $\mathcal{S}_1$ and $\mathcal{S}_2$ SoCs based on a two-test-bus architecture. For all attempted cases, our algorithm generated a test bus assignment vector in addition to subdividing one bus into $w_1$ and $w_2$ bits in problem $\mathcal{P}_5$. As can be observed, our algorithm either obtained the same results as [9] or showed improvement. We believe that this is due to the fact that the ILP formulation [9] performs better when the search space is constrained. On the other hand, our algorithm outperforms the ILP formulation [9] in $\mathcal{P}_5$ for larger $W$. For example, in the case of $W = 32$, there is a decrease of 56 470 cycles or an improvement of $10.44\%$, while for $W = 44$, there is a decrease of 40 378 cycles or an improvement of $10.28\%$. For $W = 52$ bits, there is a decrease of 112 284 cycles or $25.5\%$. Furthermore, our system generated more design alternatives than [9] while reducing the test time. It should be noted that it was not possible to further compare $\mathcal{P}_4$ and $\mathcal{P}_5$, as these two problems were not attempted by other researchers. Fig. 8 shows the architecture generated by our system for $\mathcal{S}_1$ with two test buses of total width $W = 44$, where bus 1 is allowed to have one subdivision. The resulting chromosome generated by our algorithm is (1a, 1a, 1a, 1b, 1a, 1b, 1b, 1a, 1b, 2) with $w_{1a} = 1$, $w_{1b} = 1$, and $w_3 = 42$. The resulting test time is 354 632 cycles. The population size for this part was 150, and the number of generations was chosen to be 300. The probability of crossover was 0.35, and mutation probability was 0.65.

## VI. Conclusion

We presented a genetic formulation for the design of SoC test architectures that explores test bus sizing while concurrently assigning cores to test buses. Our system generated test architectures with minimum

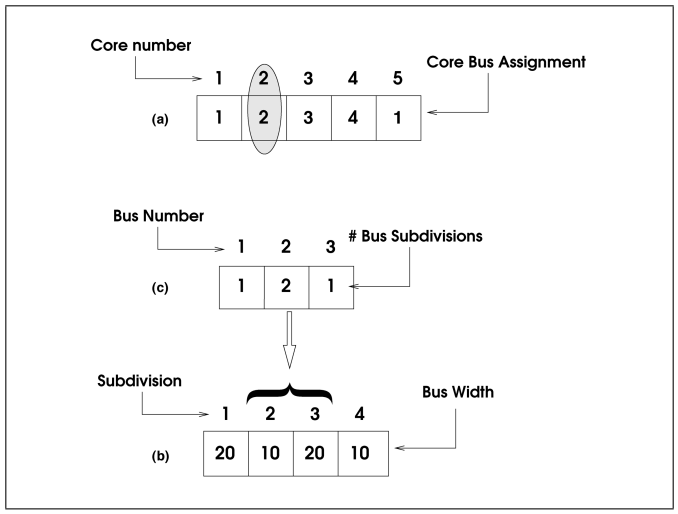


**Figure 5:** *Sample chromosome representation with five cores and three buses, where two buses are subdivided into two subdivisions.*

```
Mutation(Chromosome C)
{
   // Mutate Part (a)
   Core_i ← RandomCore(1..N_C);
   Bus_j ← RandomBus(1..N_S + N_B);
   Assign Core_i to Bus_j

   // Mutate Part (b)
   Core_i ← RandomCore(1..N_C);
   if φ_i/2 ≥ W
      w_i = RandomWidth(0, W);
   else if φ_i < W
      w_i = RandomWidth(φ_i/2, φ_i);
   else if φ_i > W
      w_i = RandomWidth(φ_i/3, W);

   // Mutate Part (c)
   Bus_k ← RandomCore(1..N_B);
   m ← Original number of bus subdivisions;
   l ← RandomDivision(1..j_max);
   Split Bus_k into l subdivisions
   gap = m − l
   if (gap < 0)
      AddSlot in Part (C)
   else
      ReduceSlot in Part (c);
      Repair Part (b)
}
```

**Figure 6:** *Mutation operator.*

testing time while minimizing test data width and performing assignment of cores to test buses. The system includes a tradeoff mechanism that explores possible test time reduction among core assignments, test data widths, and test time by allowing buses to fork out as well as to merge in. We presented experimental results demonstrating that the proposed TAM optimization methodology provides efficient test bus designs with minimum testing time while outperforming reported techniques.

## Acknowledgements

This work was supported in part by a grant from the Lebanese National Council for Scientific Research (CNRS).

```
Repair (Chromosome C)
{
   while i < N_B + N_S
         j ← Number of subdivisions in bus i;
         k ← Total Number of bus widths, w_j;
         gap ← (j-k)
         while (gap < 0)
            s_l = randomWidth(0, subdivision);
            s_i − −;
            gap + +;
         i + +;
}
```

**Figure 7:** *Repair operator.*

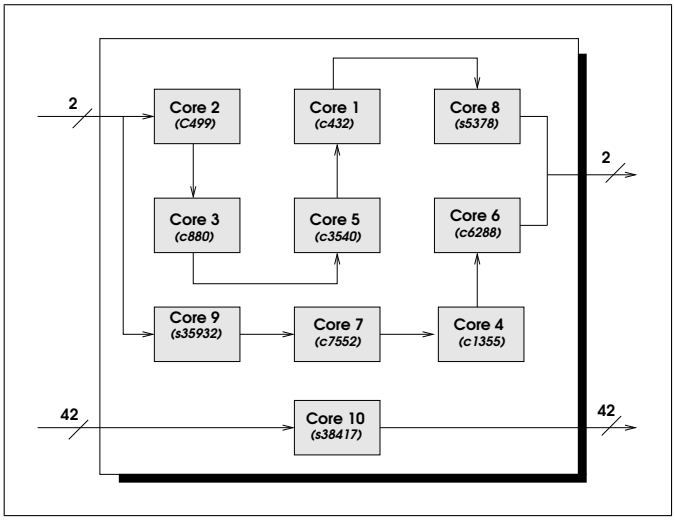


**Figure 8:** *Test bus architecture for system $S_1$ with two test buses and total width $W = 44$. Bus 1 is allowed to have only one subdivision.*

## References

[1] Y. Zorian, E.J. Marinissen, and S. Dey, "Testing embedded core-based system chips," in *Proc. Int. Test Conf.*, 1998, pp. 130–143.

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

[3] V. Immaneni and S. Raman, "Direct access test scheme: Design of block and core cells for embedded ASICs," in *Proc. Int. Test Conf.*, 1990, pp. 488–492.

[4] P. Varma and S. Bhatia, "A structured test re-use methodology for core-based system chips," in *Proc. Int. Test Conf.*, 1998, pp. 294–302.

[5] I. Ghosh, N.K. Jha, and S. Dey, "A fast and low cost testing technique for core-based system-on-chip," in *Proc. Design Automat. Conf.*, 1998, pp. 542–547.

[6] E.J. Marinissen, R. Arendsen, G. Bos, H. Dingemanse, M. Lousberg, and C. Wouters, "A structured and scalable mechanism for test access to embedded reusable cores," in *Proc. Int. Test Conf.*, Oct. 1998, pp. 284–293.

[7] L. Whetsel, "An IEEE 1149.1 base test access architecture for ICs with embedded cores," in *Proc. Int. Test Conf.*, 1997, pp. 69–78.

[8] P. Harrod, "Testing re-usable IP: A case study," in *Proc. Int. Test Conf.*, 1999, pp. 493–498.

[9] K. Chakrabarty, "Optimal test access architectures for system-on-a-chip," *ACM Trans. Design Automat. Electron. Syst.*, vol. 6, Jan. 2001, pp. 26–49.

[10] V. Iyengar and K. Chakrabarty, "Test bus sizing for system-on-a-chip," *IEEE Trans. Comput.*, vol. 51, May 2002, pp. 449–459.

[11] Z.S. Ebadi and A. Ivanov, "Design of an optimal test access architecture using a genetic algorithm," in *Proc. 10th Asian Test Symp.*, 2001, pp. 205–210.

[12] Y. Wang and W. Huang, "Optimizing test access mechanism under constraints by genetic local search algorithm," in *Proc. 12th Asian Test Symp.*, 2003, pp. 428–431.

[13] Z.S. Ebadi and A. Ivanov, "Design of an optimal test access architecture under power and place-and-route constraints using GA," in *Proc. IEEE Latin-American Test Workshop*, 2002, pp. 154–159.

[14] K. Chakrabarty, "Design of system-on-a-chip test access architectures using integer linear programming," in *Proc. VLSI Test Symp.*, 2000, pp. 127–134.

[15] ———, "Design of system-on-a-chip test access architectures under place-and-route constraints," in *Proc. Design Automat. Conf.*, 2000, pp. 432–437.

**Table 12**
**Assignment of cores to two test buses of predetermined widths and test bus subdivisions, where only one bus is allowed to fork into two branches, for $\mathcal{S}_1$ SoC ($\mathcal{P}_4$)**

| $(w_1, w_2)$ | ILP | | GA | | Improvement |
|---|---|---|---|---|---|
| | Distribution | Test time (cycles) | Distribution | Test time (cycles) | (%) |
| (19, 1) | (1, 18, 1) | 442 889 | (18, 1, 1) | 442 376 | 0.12 |
| (23, 1) | (1, 22, 1) | 664 491 | (22, 1, 1) | 427 752 | 35.63 |
| (27, 1) | (26, 1, 1) | 413 425 | (26, 1, 1) | 413 128 | 0.072 |
| (19, 13) | (1, 18, 13) | 444 974 | (18, 1, 13) | 442 376 | 0.58 |
| (32, 4) | (31, 1, 4) | 624 918 | (31, 1, 4) | 394 848 | 36.82 |
| (32, 12) | (31, 1, 12) | 395 010 | (31, 1, 12) | 394 848 | 0.041 |
| (22, 20) | (1, 21, 20) | 437 668 | (21, 1, 20) | 427 752 | 2.27 |

**Table 13**
**Assignment of cores to two test buses of predetermined widths and test bus subdivisions, where only one bus is allowed to fork into two branches, for $\mathcal{S}_2$ SoC ($\mathcal{P}_4$)**

| $(w_1, w_2)$ | ILP | | | GA | | | Improv. |
|---|---|---|---|---|---|---|---|
| | Distribution | Test time (cycles) | Bus assignment | Distribution | Test time (cycles) | Bus assignment | (%) |
| (23, 1) | (12, 11, 1) | 1 772 909 | (1a, 1a, 1a, 1a, 1b, 1a, 2, 2, 2, 2) | (11, 12, 1) | 1 664 230 | (1b, 2, 1b, 1b, 1a, 2, 1b, 1b, 1b, 1b) | 6.13 |
| (23, 1) | (12, 11, 1) | 1 772 909 | (1a, 1a, 1a, 1a, 1b, 1a, 2, 2, 2, 2) | (11, 12, 1) | 1 664 230 | (2, 2, 1b, 1b, 1a, 2, 1b, 1b, 1b, 1b) | 6.13 |
| (28, 8) | (16, 12, 8) | 1 700 237 | (1a, 1, 1a, 1a, 1b, 1a, 2, 2, 2, 2) | (17, 11, 8) | 1 633 600 | (2, 1b, 2, 1b, 1a, 2, 1b, 1b, 1b, 1b) | 3.92 |
| (28, 8) | (16, 12, 8) | 1 700 237 | (1a, 1, 1a, 1a, 1b, 1a, 2, 2, 2, 2) | (17, 11, 8) | 1 633 600 | (1b, 1b, 2, 1b, 1a, 2, 1b, 1b, 1b, 1b) | 3.92 |
| (30, 10) | (18, 12, 10) | 1 672 119 | (1a, 1, 1a, 1a, 1b, 1a, 2, 2, 2, 2) | (19, 11, 10) | 1 623 390 | (1b, 2, 2, 1b, 1a, 2, 1b, 1b, 1b, 1b) | 2.91 |
| (30, 10) | (18, 12, 10) | 1 672 119 | (1a, 1, 1a, 1a, 1b, 1a, 2, 2, 2, 2) | (19, 11, 10) | 1 623 390 | (2, 2, 2, 1b, 1a, 2, 1b, 1b, 1b, 1b) | 2.91 |
| (32, 12) | (17, 15, 12) | 1 682 069 | (1a, 1, 1a, 1a, 1b, 1a, 2, 2, 2, 2) | (20, 12, 12) | 1 618 285 | (1b, 1b, 2, 1b, 1a, 2, 1b, 1b, 1b, 1b) | 3.79 |
| (32, 12) | (17, 15, 12) | 1 682 069 | (1a, 1, 1a, 1a, 1b, 1a, 2, 2, 2, 2) | (20, 12, 12) | 1 618 285 | (1b, 1b, 2, 1b, 1a, 2, 1b, 1b, 1b, 1b) | 3.79 |

**Table 14**
**Test time and width distribution for two test buses, where only one bus is allowed to fork into two branches, for $\mathcal{S}_1$ SoC ($\mathcal{P}_5$)**

| Width ($W$) | ILP | | | GA | | | Improv. |
|---|---|---|---|---|---|---|---|
| | Distribution | Test time (cycles) | Bus assignment | Distribution | Test time (cycles) | Bus assignment | (%) |
| 20 | (1, 18, 1) | 442 889 | (1b, 1a, 1a, 1a, 1a, 1a, 2, 2, 1a, 1b) | (18, 1, 1) | 442 376 | (2, 1b, 2, 1b, 1b, 1b, 1b, 2, 1b, 1a) | 0.02 |
| 24 | (1, 22, 1) | 664 491 | (1b, 1a, 1a, 1a, 1a, 1a, 2, 2, 1b, 1b) | (22, 1, 1) | 427 752 | (2, 1b, 2, 1b, 1b, 1b, 2, 1b, 2, 1a) | 35.63 |
| 28 | (26, 1, 1) | 413 425 | (1a, 1b, 1b, 1b, 1b, 1b, 2, 2, 1b, 1a) | (26, 1, 1) | 413 128 | (2, 2, 1b, 1b, 1b, 1b, 2, 2, 1b, 1a) | 0.07 |
| 32 | (1, 18, 13) | 444 974 | (1b, 1a, 1a, 1a, 2, 1a, 2, 2, 2, 1b) | (30, 1, 1) | 398 504 | (2, 2, 1b, 2, 1b, 1b, 2, 1b, 2, 1a) | 10.44 |
| 36 | (31, 1, 4) | 624 918 | (1a, 1b, 1b, 1b, 1b, 1b, 2, 2, 1a, 1a) | (34, 1, 1) | 383 880 | (2, 2, 1b, 1b, 1b, 1b, 2, 2, 1b, 1a) | 38.57 |
| 44 | (31, 1, 12) | 395 010 | (1a, 1b, 1b, 1b, 1b, 1b, 2, 2, 1b, 1a) | (1, 1, 42) | 354 632 | (1a, 1a, 1a, 1b, 1a, 1b, 1b, 1a, 1b, 2) | 10.28 |
| 52 | (1, 21, 20) | 437 668 | (1b, 1a, 1a, 1a, 2, 1a, 1, 1, 1, a2) | (1, 1, 50) | 325 384 | (1a, 1b, 1b, 1b, 1a, 1b, 1b, 1b, 1a, 2) | 25.55 |

**Table 15**
**Test time and width distribution for two test buses, where only one bus is allowed to fork into two branches, for $\mathcal{S}_2$ SoC ($\mathcal{P}_5$)**

| Width ($W$) | ILP | | | GA | | | Improv. |
|---|---|---|---|---|---|---|---|
| | Distribution | Test time (cycles) | Bus assignment | Distribution | Test time (cycles) | Bus assignment | (%) |
| 24 | (12, 11, 1) | 1 772 909 | (1a, 1a, 1a, 1a, 1b, 1a, 2, 2, 2, 2) | (11, 12, 1) | 1 664 230 | (1b, 2, 1b, 1b, 1a, 2, 1b, 1b, 1b, 1b) | 6.13 |
| 24 | (12, 11, 1) | 1 772 909 | (1a, 1a, 1a, 1a, 1b, 1a, 2, 2, 2, 2) | (1, 11, 12) | 1 664 230 | (1a, 1a, 2, 2, 1b, 1a, 2, 2, 2, 2) | 6.13 |
| 36 | (16, 12, 8) | 1 700 237 | (1a, 1, 1a, 1a, 1b, 1a, 2, 2, 2, 2) | (1, 15, 20) | 1 618 512 | (1b, 1b, 1b, 1b, 2, 1a, 1b, 1b, 1b, 1b) | 4.81 |
| 36 | (16, 12, 8) | 1 700 237 | (1a, 1, 1a, 1a, 1b, 1a, 2, 2, 2, 2) | (20, 15, 1) | 1 618 512 | (1a, 1a, 1b, 1b, 1b, 1a, 2, 1b, 1b, 1b, 1b) | 4.81 |
| 40 | (18, 12, 10) | 1 672 119 | (1a, 1, 1a, 1a, 1b, 1a, 2, 2, 2, 2) | (35, 1, 4) | 1 598 231 | (2, 1b, 1a, 1a, 1a, 2, 1b, 1b, 1b, 1b) | 4.42 |
| 40 | (18, 12, 10) | 1 672 119 | (1a, 1, 1a, 1a, 1b, 1a, 2, 2, 2, 2) | (4, 35, 1) | 1 598 231 | (2, 2, 1b, 1b, 1b, 1a, 2, 2, 2, 2) | 4.42 |
| 44 | (17, 15, 12) | 1 682 069 | (1a, 1, 1a, 1a, 1b, 1a, 2, 2, 2, 2) | (7, 36, 1) | 1 584 896 | (2, 1a, 1b, 1b, 1b, 1a, 2, 2, 2, 2) | 5.78 |
| 44 | (17, 15, 12) | 1 682 069 | (1a, 1, 1a, 1a, 1b, 1a, 2, 2, 2, 2) | (7, 36, 1) | 1 584 896 | (1a, 1a, 1b, 1b, 1b, 1a, 2, 2, 2, 2) | 5.78 |

**Haidar M. Harmanani** received the B.S., M.S., and Ph.D. degrees, all in computer engineering, from Case Western Reserve University, Cleveland, Ohio, U.S.A., in 1989, 1991, and 1994, respectively. He joined the Lebanese American University (LAU), Lebanon, in 1994 as an assistant professor of computer science. Currently, he is an associate professor of computer science and the chairperson of the Computer Science and Mathematics Division at LAU, Byblos Campus. Prof. Harmanani has been on the program committee of various international conferences, including the IEEE NEWCAS Conference (NEWCAS 2006 and 2007), the IEEE International Conference on Electronics, Circuits, and Systems (ICECS 2000, 2006, and 2007), and the 14th IEEE International Conference on Microelectronics, 2002. His research interests include electronic design automation, high-level synthesis, system-on-chip testing, design for testability, and cluster parallel programming. He is a senior member of IEEE and ACM.

**Rachel Sawan** received the Bachelor of Engineering degree in computer engineering from the Lebanese American University, Byblos, Lebanon, in 2005. Currently, she is a system engineer at Consolidated Contractors Company (CCC) International, Beirut, Lebanon. Her research interests include evolutionary programming and VLSI testing.