A Method for Optimizing Test Bus Assignment and Sizing for System-on-a-Chip

Haidar M. Harmanani and Rachel Sawan
Department of Computer Science and Mathematics
Lebanese American University
Byblos, 1401 2010, Lebanon

***Abstract*— Test access mechanism (TAM) is an important element of test access architectures for embedded cores and is responsible for on-chip test patterns transport from the *source* to the *core* under test to the *sink*. Efficient TAM design is of critical importance in SOC system integration since it directly impacts testing time and cost. In this paper, we propose an efficient genetic algorithm to design test access architectures while investigating test bus sizing and assignment of cores to test buses in the system. We present experimental results that demonstrate the effectiveness of our method while outperforming reported techniques.**

I. Introduction

Advances in semiconductor process technologies enable the integration of an entire system on a chip (SOC) based on a redesign philosophy that divides the CAD community into core providers and core integrators. Core providers create *pre-designed* and *pre-verified* complex logic blocks that cover a wide range of functions such as CPUs, DSPs, media processors, memories, and mixed-signal modules. Core integrators, on the other hand, create the SOC by assembling and combining available cores and their custom user-defined logic. One of the implications of the core-based design methodology is that test development for core-based systems should also be core-based. Thus, the core user must provide an access path in the *on-chip* hardware while the core tests, as given by the core provider, have to be translated from the core terminals to the IC pins. A generic conceptual test access architecture for an embedded core consists of a test source and a sink, a test access mechanism and a core wrapper [12]. The test source is used for test stimulus generation while the response evaluation is carried out by the test sink. Test access mechanism (TAM) ensures on-chip test patterns transport and is characterized by its capacity. Finally, the core is surrounded with test logic, known as *test-wrapper*, that provides switching functionality between *normal* access and *test* access via the TAM [12].

Genetic algorithms are probabilistic optimization techniques that use a group of random points, a population, in order to non-deterministically search the design space. The population is modified following a parody of Darwinian principle of the survival of the fittest. Individuals encode problem parameters, and are selected according to their quality to produce *offspring* and propagate their genetic material into the next generation. Genetic algorithms use an iterative process of *selection* and *recombination* in order to improve the quality of the population until a solution is found or a certain stopping criterion is met.

A. Related Work

Various test access strategies were proposed in the literature [1]. Immaneni et al. [5] introduced a test access mechanism that accommodates multiple embedded cores through multiplexing. Varma et al. [8] proposed a dedicated bus approach based on a combination of multiplexing and distribution. Ghosh et al. [3] proposed a method in which test access to embedded cores is based on transparent paths through other cores. Marinissen et al. [7] proposed a scalable bus-based architecture based on a combination of daisy chain and distribution architecture. The approach, called TestRail, provides a flexible and scalable test access mechanism. Chakrabarty [2] explored the relationship between testing time and test bus widths based on several ILP models. The method was extended later in order to include power and placement constraints [6]. Though the ILP models were aimed at optimal results, they were halted in various cases in order to obtain nearly-optimal solutions. Other techniques include isolated rings [10] and the reuse of the existing test bus [4]. In order to alleviate the problem complexity, heuristic techniques were proposed. For example, Ebadi et al. [11] and Wang et al. [9] proposed evolutionary approaches in order to solve the TAM optimization problem. 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. Furthermore, no heuristic approach attempted to reduce test time by exploring tradeoffs among test bus subdivision, core assignment and test bus minimization.

B. Problem Formulation

This paper proposes an efficient approach for the design of test access architectures for SOC that minimizes test time based on the TestRail architecture. Given a system with N_c cores, N_B buses with a maximum test data width of W, the problem is to 1) determine the optimal or sub-optimal width that should be distributed among the various test buses in order to minimize test time; 2) assign cores to test buses so that test time is minimized; and 3) determine an optimal subdivision of test buses that can test smaller cores in parallel to save testing time. We solve the above problem by breaking it into a progression of five incremental problems in the order of increasing complexity. We demonstrate the effectiveness of the proposed method using an SOC benchmark, the $\mathcal{S}_2$ system [6]; a non-trivial SOC that contains *ten* cores of various sizes

0840-7789/07/$25.00 ©2007 IEEE

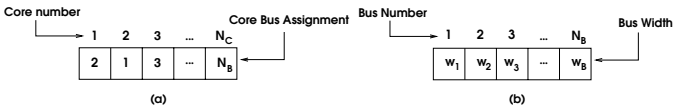

Fig. 1. Chromosome: (a) Core bus assignment, (b) Bus width

and I/O widths. The proposed method is characterized by the following contributions:

- A solution for several TAM design problems that are $\mathcal{NP}$-complete using an efficient genetic algorithm.
- A minimal test data width and an assignment of cores to test buses subject to minimizing testing time.
- A tradeoff mechanism that trades-off test assignment, test data widths and test time by allowing test buses to fork out as well as to merge in.

II. Test Data De-Serialization Model

Consider an SOC consisting of N_c cores and let core i, $1 \leq i \leq N_c$, have n_i inputs and m_i outputs (including data and scan I/Os). Let the SOC have N_B test buses with widths $w_1, w_2, ..., w_{N_B}$. Chakrabarty et al. [2], [6] 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 the core i's test width ϕ, 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 de-serialization is necessary. For combinational cores, t_i is equal to the number of test patterns p_i. However, for cores with internal scan $t_i = (p_i + 1)\lfloor f_i/s_i \rfloor + p_i$ where core i contains f_i flip-flops and s_i internal scan chains [2]. If we assume uniform distribution of test bus lines among the core I/Os, then the testing time with test data de-serialization for core i assigned to bus j is given by [6]:

$$T_{ij} = \lceil \frac{\phi_i}{w_j} \rceil t_i \tag{1}$$

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. A similar assumption was implicitly made in [2], [6]

III. Assignment of Cores to Test Buses

The first problem of importance to SOC testing is the assignment of cores to test buses so that to minimize the system test time. The problem is formally defined as follows:

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

A. Chromosomal Representation

In order to solve the core assignment problem, we propose the chromosomal representation shown in Figure 1(a). The representation is based on a vector where a gene j in position i indicates that core i is assigned to bus j where $1 \leq i \leq N_c$ and $1 \leq j \leq N_B$.

TABLE I

Assignment of cores to test buses, $\mathcal{P}_{assign}$

(W_1, W_2)	ILP		GA	
	Test Time	Bus Assignment	Test Time	Bus Assignment
(16,8)	224070	(2,2,2,2,1,1,2,2,1,2)	219196	(2,2,1,1,2,1,1,1,1,1)
(16,16)	167539	(2,2,2,2,2,2,1,1,2,2,1)	163778	(1,1,1,2,1,2,1,1,2,2)
(32,16)	112506	(2,1,2,2,1,1,2,2,1,2)	109596	(2,2,1,1,2,1,1,1,1,1)
(32,32)	84494	(2,1,2,1,1,2,1,2,2,2)	81888	(1,1,1,2,1,2,1,1,2,2)

B. Initial Population

The initial population has an important impact on the quality and the time needed to converge to a final solution. The initial population is generated by creating an initial chromosome where the buses are equally assigned to all cores. The remainder of the population is next generated through 20% mutation and 80% random perturbation of the initial chromosome.

C. Genetic Operators

We explore the design space using two genetic operators, *mutation* and *crossover*. The mutation operator is used to explore the design space while the crossover operator overcomes the information loss during genetic evolution. We use a *non-uniform* mutation operator that chooses a random core and assigns it to a random bus. The crossover operator that we use is a *2-point crossover* where two parents chromosomes are randomly chosen; two random cut points are selected next and the middle genes of both chromosomes are exchanged.

D. Objective Function

The objective function is to minimize the time needed to test all cores assigned to the buses. That is, minimize:

$$\mathcal{T} = \max_j \sum_{i=1}^{N_c} T_{ij} \qquad 1 \leq j \leq N_B \tag{2}$$

Where T_{ij} is the testing time with test data de-serialization for core i assigned to bus j, given by equation 1.

E. Algorithm

The genetic algorithm was implemented using Java and tested on the $\mathcal{S}_2$ benchmark. Every generation, $\frac{3n}{2}$ offspring are created form the current n chromosomes through mutation and crossover. The best n chromosomes are next kept and the algorithm repeats for the maximum number of generations. The algorithm generated in the case of $\mathcal{S}_2$ an optimal assignment in less than *one* second. The detailed results are shown in Table I. As it is shown, our algorithm outperformed the ILP formulation in [6] for all attempted cases in a very short time. The probability for the mutation operator was determined to be P_m = 65% while the crossover probability was set to P_c = 35%. The population size was chosen to be 150 and the number of generations was set to 300.

TABLE II

MINIMUM TEST TIME AND WIDTH DISTR. FOR A FIXED W, $\mathcal{P}_{mintime}, N_B = 2$

W	ILP		GA		
	(W_1, W_2)	Test Time	(W_1, W_2)	Test Time	Bus Assignment
32	(16,16)	166604	(7,25)	163690	(1,2,1,1,2,2,1,2,1,2)
36	(20,16)	147627	(27,9)	145528	(2,1,1,2,1,1,2,1,1,2)
40	(24,16)	134060	(30,10)	130951	(2,2,1,2,1,1,2,2,2,1)
44	(28,16)	120542	(5,39)	119042	(1,2,1,2,2,2,2,2,2,1)
48	(28,20)	111400	(21,27)	109148	(1,2,1,1,1,2,2,1,2,2)
52	(24,28)	103458	(27,25)	100734	(2,1,2,1,1,2,2,1,2,1)
56	(28,28)	96544	(22,34)	93547	(1,2,1,2,1,2,2,2,1,2)
60	(28,32)	91287	(41,19)	87313	(2,2,2,1,1,1,2,2,2,2)
64	(32,32)	88332	(55,9)	81844	(1,1,1,2,1,1,1,2,2,1)

IV. TEST BUS SIZING

The next problem that we address is the problem of determining optimal widths of test buses concurrently with the assignment of cores to test buses. The problem is a generalization of $\mathcal{P}_{assign}$ and is formally defined as follows:

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

In order to solve $\mathcal{P}_{mintime}$, the chromosomal representation is extended by adding an additional component as shown in Figure 1(b). The additional representation in part (b) 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 has now two related parts; the first part assigns cores to test buses while the second part allocates the widths of each test bus. The cost function to minimize is the same as in $\mathcal{P}_{assign}$; however, under the additional width constraint $\sum_{j=1}^{N_B} w_j = W$ and $w_j \leq \phi_i$.

Solving $\mathcal{P}_{mintime}$ required an additional operator, *inversion*, that randomly selects a gene and inverts its value. The inversion operator 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 ϕ (Figure 3).

The initial population was generated in a similar way to $\mathcal{P}_{mintime}$. Thus, we create a single chromosome where buses are equally assigned to all cores in part (a); part (b), was created by assigning each gene in the chromosome a test bus width w_j where $w_j = \min_i\{\phi_i\}$. The initial population includes 10% of the above chromosome. The next 50% chromosomes are created through mutation of this single chromosome and the last 40% are created through inversion.

Another problem that is closely related to $\mathcal{P}_{mintime}$ is the problem of determining the minimum system test width W needed to satisfy testing time constraints in addition to finding an optimal distribution of the test bus width among individual test buses, and an optimal test bus assignment. Problem $\mathcal{P}_{minwidth}$ is formally defined as follows:

> $\mathcal{P}_{minwidth}$: Given N_c cores, N_B test buses, and a maximum test time specification $\mathcal{T}$, determine the optimal width of the test buses and an assignment of cores to test buses such that the testing time is less than $\mathcal{T}$.

TABLE III

MINIMUM TEST TIME AND WIDTH DISTR. FOR A FIXED W, $\mathcal{P}_{mintime}, N_B = 3$

Width	Bus Assignment	Bus Width	Test Time
16	(1, 2, 2, 1, 3, 2, 1, 2, 3, 1)	(4, 6, 6)	327768
20	(3, 3, 1, 3, 1, 1, 2, 2, 2, 1)	(15, 4, 1)	261947
24	(1, 2, 2, 1, 3, 2, 1, 2, 3, 1)	(6, 9, 9)	218511
28	(2, 3, 3, 1, 3, 2, 1, 3, 2, 1)	(7, 10, 11)	187226
28	(3, 2, 2, 1, 2, 1, 3, 2, 1, 1)	(14, 11, 3)	187226
32	(3, 2, 1, 2, 1, 1, 2, 3, 3, 1)	(24, 5, 3)	163757
36	(1, 3, 1, 2, 3, 3, 2, 1, 1, 2)	(4, 9, 23)	145604
40	(1, 1, 2, 1, 2, 2, 3, 3, 3, 2)	(2, 30, 8)	130973
44	(2, 2, 2, 3, 1, 1, 3, 3, 2, 3)	(28, 3, 13)	119064
48	(1, 3, 3, 1, 2, 1, 2, 2, 1, 1)	(24, 23, 1)	109205
52	(1, 1, 1, 1, 2, 3, 3, 3, 1, 3)	(6, 17, 29)	100898
56	(3, 3, 1, 1, 3, 2, 3, 3, 2, 1)	(9, 20, 27)	93599
60	(1, 1, 1, 2, 2, 2, 3, 3, 3, 1)	(7, 41, 12)	87315
64	(2, 1, 2, 1, 2, 3, 1, 2, 3, 3)	(10, 25, 29)	81878

The problem representation and parameters are the same as in $\mathcal{P}_{mintime}$; however, the difference is the objective function which is now to minimize:

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

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

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

We solve problems $\mathcal{P}_{mintime}$ and $\mathcal{P}_{minwidth}$ for several values of W and $\mathcal{T}$. The run time was equal to 4 CPU seconds. Table II shows the minimum test time and width distribution for a fixed W and $N_B = 2$ while Table III shows the same results for $N_B = 3$. We note that for smaller values of W, our algorithm generates slight improvements over the ILP method [6]. However, for larger values of W, our algorithm gives considerable improvements up till 7.34%. Table IV shows the results for test bus width and width distribution for a given maximum test time under time constraint $\mathcal{T}$ for problem $\mathcal{P}_{minwidth}$. Again, for smaller $\mathcal{T}$, our algorithm obtains substantial improvements. For example, for $\mathcal{T} = 100,000$, our algorithm saves 3 bits of the TAM or 5.36% decrease in bus width. The population size for this part was 150 and the number of generations was chosen to be 300. The probability of crossover was 35% and mutation was 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. Thus, test buses may fork out into a set of smaller test buses that transport, in parallel, test data to smaller cores. This reduces testing time especially when several small cores with small test widths are assigned to a wide test bus. Larger cores can

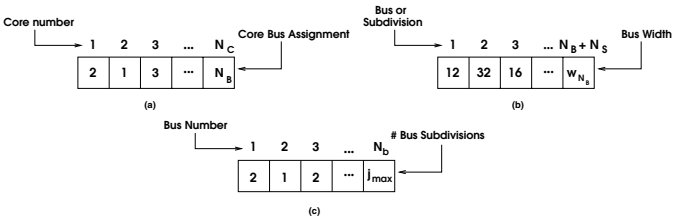

Fig. 2. (a) Chromosome representation, (b) Core bus assignment, (d) Bus subdivisions

be still assigned to the undivided part of the test bus as before. The problem, $\mathcal{P}_{fork}$, is defined as follows:

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

Finally, we formulate the general case of the test bus subdivision problem where the optimal test bus widths must be determined as well. The main difference is that there is now no restriction on the width of each test bus but rather on the total width W which gives the problem a higher degree of freedom. The problem, $\mathcal{P}_{general}$, is now formally defined as follows:

> $\mathcal{P}_{general}$**:** Given N_c cores and 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 (i) optimal width for each test bus, (ii) optimal subdivision of the test buses, and (iii) an assignment of cores to test buses such that the total testing time is minimized.

In order to solve $\mathcal{P}_{fork}$ and $\mathcal{P}_{general}$, we extend the chromosomal representation so that to incorporate bus subdivisions. The extended chromosomal representation, shown in Figure 2, is of variable length and includes three related parts. Part (a) 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 the width of 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 explores 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. Thus, a chromosome grows and shrinks depending on the number of subdivisions in part (c) resulting with a variable length chromosome. The algorithm invokes the *reduceSlots* operation in order to decrease the size of part (b) while if the number of subdivisions increases in part (c) then the algorithm increases the size of part (b) using the *addSlots* operation.

TABLE IV

WIDTH AND WIDTH DISTRIBUTION FOR A MAX TEST TIME, $\mathcal{P}_{minwidth}$

Max $\mathcal{T}$	ILP		GA		
	W	(W_1, W_2)	W	(W_1, W_2)	Bus Assignment
200000	27	(4,23)	27	(14,13)	(2,2,1,2,2,1,1,1,2)
190000	28	(5,23)	28	(18,10)	(2,2,1,1,1,2,1,2,1,1)
180000	30	(2,28)	30	(15,15)	(1,2,2,1,2,1,2,1,2,1)
170000	32	(6,26)	31	(16,15)	(1,2,1,2,2,1,2,1,1,1)
160000	34	(6,28)	33	(19,14)	(2,2,1,2,2,1,1,2,1,1)
150000	36	(8,28)	35	(24,11)	(2,1,2,1,1,1,2,2,2,2)
140000	39	(22,17)	38	(14,24)	(1,2,2,2,1,2,2,1,2,2)
130000	41	(18,23)	41	(22,19)	(1,1,2,1,1,2,1,2,1,2)
120000	46	(18,28)	44	(30,14)	(2,2,2,1,1,1,2,2,2,2)
110000	49	(28,21)	48	(22,26)	(1,1,2,1,2,1,2,2,2)
100000	56	(29,28)	53	(4,49)	(2,1,1,1,2,2,2,2,2,2)

The variable length nature of the chromosome is due to part (c) that affects 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) becomes invalid. For example, if the number of subdivisions in part (c) of the chromosome is decreased from 7 to 6 then cores in part (a) that are assigned to subdivision 7 become invalid. The resulting chromosome is infeasible. In addition, the crossover operator may generate infeasible chromosomes in $\mathcal{P}_{fork}$ only due to restrictions on the width of individual buses where the sum of the width in the subdivisions is more than the width of the bus w_j.

We repair the first part in the chromosome by reassigning cores that are assigned to invalid subdivisions to random valid subdivisions. We repair the second part by randomly reducing the number of bits in the subdivisions. The method iterates over an invalid part (b) and chooses in every iteration a random bus whose width is decremented. The function repeats until the chromosome is feasible.

In order to explore the design space for problems $\mathcal{P}_{fork}$ and $\mathcal{P}_{general}$, 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 Figure 3. In order to have a good hill climbing effect, we define two types of mutations. The first mutates part (a) *only* while fixing part (b) and (c) and the second mutates all three parts *concurrently*.

The initial population was generated based on three initial *single* chromosomes, in a similar way to the previous three problems. Part (a) of the *single* chromosome is created such that cores are equally assigned to all buses while part (b) was created by assigning 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) was created based on three categories. The first category includes a chromosome such that each bus has the maximum number of subdivisions allowed, j_{max} while the second category includes a chromosome such that buses are not subdivided. 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 based on 10% from the first category, 10% of the second category, and 20% of the last category. The next 40%

TABLE V

CORE ASSIGNMENT TO TEST BUSES OF PREDETERMINED WIDTHS AND TEST BUSES SUBDIVISION, $\mathcal{P}_{fork}$

(W_1, W_2)	ILP			GA		
	Distribution	Test Time	Bus Assg.	Distribution	Test Time	Bus Assg.
(23,1)	N/A	N/A	N/A	(23,1)	223137	(2,2,2,1,1,1,1,1,1,1)
(23,1)	N/A	N/A	N/A	(7,16,1)	223291	(1a,2,2,1a,1b,1b,1a,1b,1a,1a)
(24,16)	(4,20,16)	132895	(2,2,2,2,2,1b,1b,1a,1a,1b)	(4,20,16)	131794	(1a,1a,1b,1b,1b,2,1b,2,2,1a)
(24,16)	(4,20,16)	132895	(2,2,2,2,2,1b,1b,1a,1a,1b)	(1,23,16)	131794	(1a,1a,1a,1b,1b,2,1b,2,2,1b)
(24,16)	(4,20,16)	132895	(2,2,2,2,2,1b,1b,1a,1a,1b)	(24,16)	131466	(1,1,1,2,1,2,1,2,1,1)
(28,8)	N/A	N/A	N/A	(28,8)	145613	(2,2,2,2,1,1,2,1,2,1)
(30,10)	N/A	N/A	N/A	(30,10)	130951	(2,2,1,2,1,1,2,2,2,1)
(32,12)	N/A	N/A	N/A	(32,12)	119334	(2,2,2,1,1,1,2,1,2,2)

TABLE VI

MINIMUM TEST TIME AND WIDTH DISTR. FOR TWO TEST BUSES, $\mathcal{P}_{general}$

W	GA		
	Distribution	Test Time	Bus Assg.
24	(18,6)	218295	(2,1,1,2,1,1,2,1,1,2)
36	(27,9)	145500	(2,2,1,2,1,1,2,2,2,1)
40	(15,25)	130991	(1,1,1,2,2,1,2,1,2,2)
44	(16,28)	119064	(1,1,1,1,2,2,1,1,1,1)
44	(13,31)	119064	(1,2,2,1,2,2,1,1,2,1)

are generated through mutation of part (a) and part (b) while the last 20% are generated though mutations of all three parts.

The objective function in $\mathcal{P}_{fork}$ and $\mathcal{P}_{general}$ is to minimize the test time needed to test all cores assigned to the test buses. That is, minimize:

$$\mathcal{T} = \max_{j} \sum_{i=1}^{N_c} T_{ij} \qquad 1 \leq j \leq N_B \tag{3}$$

Where T_{ij} is the testing time with test data de-serialization for core i assigned to bus j given in equation 1 with problem $\mathcal{P}_{general}$ having constraints on the total width, $W = \sum_{j=1}^{N_B} w_j$.

We solve problems $\mathcal{P}_{fork}$ and $\mathcal{P}_{general}$ using our genetic algorithm. The results, shown in Tables V and VI, were generated in less than 20 seconds and provide various TAM architectures for the $\mathcal{S}_2$ SOC based on two test buses 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 $\mathcal{P}_{general}$. As it can be observed, our algorithm outperformed the ILP method in [6] while generating more design alternatives. It should be noted that it was not possible to compare $\mathcal{P}_{general}$ as no results were reported in [6]. The population size for this part was 150 and the number of generations was chosen to be 300. The probability of crossover was 35% and mutation was 65%.

Mutation(Chromosome C)
{
  // Mutate Part (a)
  $Core_i \leftarrow$ RandomCore($1 \ldots N_C$);
  $Bus_j \leftarrow$ RandomBus($1 \ldots N_S + N_B$);
  Assign $Core_i$ to Bus_j

  // Mutate Part (b)
  $Core_i \leftarrow$ RandomCore($1 \ldots N_C$);
  if $\frac{\phi_i}{2} \geq W$
    w_i = RandomWidth($0 \ldots W$);
  else if $\phi_i < W$
    w_i = RandomWidth($\frac{\phi_i}{2} \ldots \phi_i$);
  else if $\phi_i > W$
    w_i = RandomWidth($\frac{\phi_i}{3} \ldots W$);

  // Mutate Part (c)
  $Bus_k \leftarrow$ RandomCore($1 \ldots N_B$);
  $m \leftarrow$ Original number of bus subdivisions;
  $l \leftarrow$ RandomDivision($1 \ldots j_{max}$);
  Split Bus_k into l subdivisions
  gap = $m - l$
  if (gap < 0)
    AddSlot in Part (C)
  else
    ReduceSlot in Part (c);
    Repair Part (b)
}

Fig. 3. Mutation Operator

ACKNOWLEDGMENT

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

REFERENCES

[1] J. Aerts and E. J. Marinissen. Scan Chain design for test Time reduction in Core-Based ICs. In *Proc. ITC*, pages 448–457, 1998.

[2] K. Chakrabarty. Optimal Test Access Architectures for System-on-a-chip. *ACM TODAES*, 6:26–49, 2001.

[3] I. Ghosh, N. K. Jha, and S. Dey. A Fast and Low Cost Testing Technique for Core-Based System-on-Chip. In *Proc. DAC*, pages 542–547, 1998.

[4] P. Harrod. Testing Re-Usable IP: A Case Study. In *Proc. ITC*, pages 493–498, 1999.

[5] V. Immaneni and S. Raman. Direct Access Test Scheme-Design of Block and Core Cells for embedded ASICs. In *Proc. ITC*, pages 488–492, 1990.

[6] V. Iyengar and K. Chakrabarty. Test Bus Sizing for System-on-a-chip. *IEEE Trans. on Computers*, 51:449–459, 2002.

[7] 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. ITC*, pages 284–293, 1998.

[8] P. Varma and S. Bhatia. A Structured Test Re-Use Methodology for Core-Based System Chips. In *Proc. ITC*, pages 294–302, 1998.

[9] Y. Wang and W. Huang. Optimizing Test Access Mechanism Under Constraints by Genetic Local Search Algorithm. In *Proc. ATS*, pages 428–431, 2003.

[10] L. Whetsel. A IEEE 1149.1 Base Test Access Architecture For ICs With Embedded Cores. In *Proc. ITC*, pages 69–78, 1997.

[11] Z. S. Ebadi and A. Ivanov. Design of An Optimal Test Access Architecture Using a Genetic Algorithm. In *Proc. ATS*, pages 205–210, 2001.

[12] Y. Zorian, E.J. Marinissen, and S. Dey. Testing Embedded Core-Based System Chips. In *Proc. ITC*, pages 130–143, 1998.