Test Time Minimization for System-On-Chip with Test Bus Assignment and Sizing

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

***Abstract*— Test access is a major problem in testing embedded cores as it directly impacts testing time and hardware cost. Test access mechanism (TAM) is responsible for test data transport and is characterized by its bandwidth capacity. Efficient TAM design is of critical importance in SOC system integration since a test architecture should reduce test cost by minimizing test application time. In this paper, we propose a genetic algorithm to design test access architectures while investigating test bus sizing concurrently with assigning cores to test buses. We present experimental results that demonstrate the effectiveness of the proposed method.**

I. Introduction

Advances in semiconductor process technologies enable the integration of an entire system on a single 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* embedded cores that are assembled along with their custom user-defined logic (UDL) by the core integrators in order to create the SOC. An important aspect of testing SOC is ensuring the existence of an access path in the *on-chip* hardware through the test access mechanism (TAM) that serves as a "test data highway." The TAM transports test patterns between the *source* and the *core* as well as between the *core* and the *sink* and is characterized by its transport capacity. Typically, 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 [1]. *Genetic algorithms* (GAs) are probabilistic optimization techniques that use a non-deterministic iterative process of *selection* and *combination* of "good" individuals. The process exploits new points in the search space by providing a diversity and avoiding premature convergence to a local optimum.

Various test access strategies were proposed such as core transparency [3], reuse of existing test bus [4], multiplexed access [5], a combination of daisy chain and distribution architecture (TestRail) [7], dedicated test bus [8], and isolated rings [10]. Chakrabarty [2] explored the relationship between test time and test bus widths based on several Integer Linear Programming models (ILP) that were halted in various cases due to the problem complexity. In order to alleviate the problem complexity, genetic algorithms based approaches were proposed [9], [11]. However, these approaches tackled only the simple TAM optimization problem and did not look into test time minimization by exploring tradeoffs among test bus subdivision, core assignment and test buses optimization.

A. Problem Description

This paper proposes a method for the design of test access architectures based on the *TestRail* approach. Given a system with N_c cores, N_B test buses with a maximum width W, the problem is to 1) determine the optimal width that should be distributed among test buses in order to minimize test time; 2) assign the embedded cores to test buses so that the test time is minimized; and 3) explore test time reductions by determining an optimal subdivision of test buses that can test smaller cores in parallel. The problem has been proven to be $\mathcal{NP}$-complete [2]. The proposed method is characterized by the following:

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

We demonstrate the effectiveness of the proposed method by using two indicative and representative benchmarks, the $\mathcal{S}_1$ and $\mathcal{S}_2$ systems [6]. $\mathcal{S}_1$ 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.

B. Test Data De-Serialization Model

Consider an SOC consisting of N_c cores, N_B test buses with widths $w_1, w_2, ..., w_{N_B}$ and let core i, $1 \le i \le N_c$, have n_i inputs and m_i outputs. If the width of the test bus is less than the number of core terminals then test data de-serialization is needed. Chakrabarty [2] 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 is equal to $(p_i+1)\lceil \frac{f_i}{s_i} \rceil + p_i$ where core i contains f_i flip-flops and s_i internal scan chains. The testing time with test data de-serialization for core i assigned to bus j is given by [2]:

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

1-4244-1164-5/07/$25.00 ©2007 IEEE.

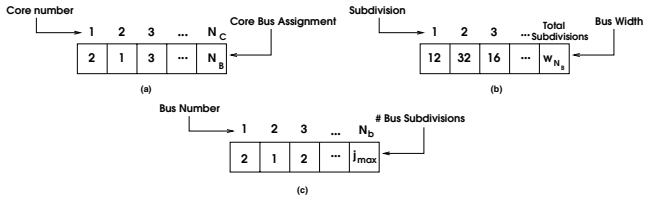

Fig. 1. (a) Chromosome representation, (b) Core bus assignment, (c) Bus subdivisions

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/O [2].

II. Problem Formulation

Test minimization and assignment of individual test buses to cores can be solved while exploring test buses subdivisions. Thus, it is possible to explore test time reductions by taking advantage of the *TestRail* flexibility that allows test buses to fork out and to merge together. 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 be still assigned to the undivided part of the test bus. The problem is formally defined as follows:

> Given N_c cores, 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 test bus widths, and (ii) an optimal assignment of cores to test buses such that the total testing time is minimized.

Another problem that is closely related to the above is the general case where test bus widths must be determined as well. The difference is that there is no restriction on the width of each test bus but rather on the total bus width W. The general case is formally defined as follows:

> 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 (i) an optimal width for each test bus, the optimal subdivision of the width of every bus, and (ii) an assignment of cores to test buses such that the total testing time is minimized.

A. Chromosomal Representation

We solve the core assignment problem using the variable length chromosomal representation shown in Figure 1. The chromosomal representation includes three related parts. Part (a) of the chromosome is a vector of size N_c where a gene j in position i indicates that core i is assigned to either a bus or a subdivision j where $1 \leq i \leq N_c$, $1 \leq j \leq N_B + N_S$, and 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

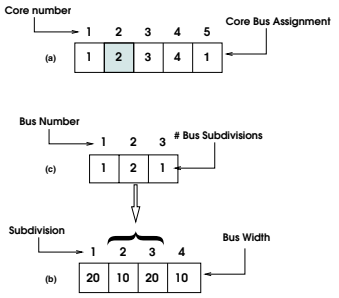

Fig. 2. Sample chromosome representation

indicates that bus or subdivision i is assigned w_j bits. Finally, part (c) of the chromosome allocates the number of bus subdivisions which should be less than or equal to j_{max}. A gene i in the j^{th} position indicates that bus j forks into i subdivisions. Based on this representation, a chromosome grows and shrinks depending on the number of subdivisions in part (c). 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. A sample chromosome is shown in Figure 2 with five cores and three buses with *bus 2* has two subdivisions. For example, *core 2* in Figure 2(a) is assigned to bus 2 while Figure 2(c) indicates that bus 2 has two subdivisions, 2 and 3, that are allocated 10 and 20 bits respectively, as shown in Figure 2(b).

B. Objective Function

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

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

Where T_{ij} is given in equation 1. The general case imposes the following constraint on the total width,

$$W = \sum_{j=1}^{N_B} w_j \tag{3}$$

C. Genetic Operators

In order to explore the design space, we use two genetic operators, *mutation* and *crossover*. The genetic operators are applied iteratively and by taking turns.

1) Mutation: We use a standard random resetting technique 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* (Figure 3). The mutation operator is applied with probability P_m.

2) Crossover: Given two chromosomes, we apply *two-point uniform crossover* where two offspring are created from two parents by exchanging the middle genes of both parents with probability P_c.

```
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)
}
```

Fig. 3. Mutation Operator

```
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 + +;
}
```

Fig. 4. Repair Operator

D. Repair Functions

The variable length nature of the chromosomes may lead to infeasible solutions. For example, 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) become invalid. Thus, if the number of subdivisions in chromosomal part (c) is decreased from 7 to 6 then cores in part (a) that are assigned to subdivision 7 become invalid. We repair these chromosomes by reassigning cores that are assigned to invalid subdivisions to random valid ones. On the other hand, the algorithm may generate infeasible chromosomes due to restrictions on the width of individual buses (equation 3) where the sum of the width in the subdivisions is more than the width of the bus w_j. We repair the second part by randomly reducing the number of bits in the subdivisions a shown in Figure 4. The method iterates over an invalid part (b) and randomly chooses in every iteration a bus. The bus width is decremented and the method repeats until the chromosome is feasible.

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 based on three initial *single* chromosomes. Part (a) of the *single* chromosome is created such that cores are equally assigned to all buses while part (b) is 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) is created based on three categories:

1) The first category includes chromosomes where buses are assigned the maximum number of allowed subdivisions, j_{max}.
2) The second category includes chromosomes with no subdivisions.
3) The third category includes chromosomes where buses are assigned random subdivisions 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% from the last category. The remaining chromosomes are generated such that 40% are generated by mutating part (a) and part (b) while the last 20% are generated by mutating all three parts.

F. Experimental Results

We have implemented the proposed genetic algorithm using the Java language. The results, shown in Tables I, II, III, and IV, were generated in less than 20 seconds and provide various TAM architectures for $\mathcal{S}_1$ and $\mathcal{S}_2$ based on a two test buses architecture. For all cases, our algorithm generated a test bus assignment in addition to subdividing one bus into w_1 and w_2 bits. As it can be observed, our algorithm either obtained the same results as in [2] or with the shown improvement. We believe this is due to the fact that the ILP formulation [2] performs better when the search space is constrained. On the other hand, our algorithm outperformed the ILP formulation in the general case 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%. It should be noted that it was not possible to further compare both problems as these

(w_1, w_2)	ILP		GA		% Imp
	Distribution	Test Time	Distribution	Test Time	
(19,1)	(1,18, 1)	442889	(18,1,1)	442376	0.12
(23,1)	(1,22,1)	664491	(22,1,1)	427752	35.63
(27,1)	(26,1,1)	413425	(26,1,1)	413128	0.072
(19,13)	(1,18,13)	444974	(18,1,13)	442376	0.58
(32,4)	(31,1,4)	624918	(31, 1, 4)	394848	36.82
(32,12)	(31,1,12)	395010	(31, 1, 12)	394848	0.041
(22,20)	(1,21,20)	437668	(21, 1, 20)	427752	2.27

TABLE I
ASSIGNMENT OF CORES TO TEST BUSES OF PREDETERMINED WIDTHS AND TEST BUSES SUBDIVISION FOR $\mathcal{S}_1$

(w_1, w_2)	ILP			GA			% Imp
	Distribution	Test Time	Assignment	Distribution	Test Time	Assignment	
(23,1)	(12,11,1)	1772909	(1a,1a,1a,1a,1b,1a,2,2,2,2)	(11,12,1)	1664230	(1b,2,1b,1b,1a,2,1b,1b,1b,1b)	6.13
(23,1)	(12,11,1)	1772909	(1a,1a,1a,1a,1b,1a,2,2,2,2)	(11,12,1)	1664230	(2,2,1b,1b,1a,2,1b,1b,1b,1b)	6.13
(28,8)	(16,12,8)	1700237	(1a,1,1a,1a,1b,1a,2,2,2,2)	(17,11,8)	1633600	(2,1b,2,1b,1a,2,1b,1b,1b,1b)	3.92
(28,8)	(16,12,8)	1700237	(1a,1,1a,1a,1b,1a,2,2,2,2)	(17,11,8)	1633600	(1b,1b,2,1b,1a,2,1b,1b,1b,1b)	3.92
(30,10)	(18,12,10)	1672119	(1a,1,1a,1a,1b,1a,2,2,2,2)	(19,11,10)	1623390	(1b,2,2,1b,1a,2,1b,1b,1b,1b)	2.91
(30,10)	(18,12,10)	1672119	(1a,1,1a,1a,1b,1a,2,2,2,2)	(19,11,10)	1623390	(2,2,2,1b,1a,2,1b,1b,1b,1b)	2.91
(32,12)	(17,15,12)	1682069	(1a,1,1a,1a,1b,1a,2,2,2,2)	(20,12,12)	1618285	(1b,1b,2,1b,1a,2,1b,1b,1b,1b)	3.79
(32,12)	(17,15,12)	1682069	(1a,1,1a,1a,1b,1a,2,2,2,2)	(20,12,12)	1618285	(1b,1b,2,1b,1a,2,1b,1b,1b,1b)	3.79

TABLE II

ASSIGNMENT OF CORES TO TEST BUSES OF PREDETERMINED WIDTHS AND TEST BUSES SUBDIVISION FOR $\mathcal{S}_2$

W	ILP			GA			% Imp
	Distribution	Test Time	Bus Assg.	Distribution	Test Time	Bus Assg.	
20	(1,18, 1)	4442889	(1b,1a,1a,1a,1a,1a,2,2,1a,1b)	(18,1,1)	442376	(2,1b,2,1b,1b,1b,1b,2,1b,1a)	0.02
24	(1,22,1)	664491	(1b,1a,1a,1a,1a,1a,2,2,1b,1b)	(22,1,1)	427752	(2,1b,2,1b,1b,1b,2,1b,2,1a)	35.63
28	(26,1,1)	413425	(1a,1b,1b,1b,1b,1b,2,2,1b,1a)	(26,1,1)	413128	(2,2,1b,1b,1b,1b,2,2,1b,1a)	0.07
32	(1,18,13)	444974	(1b,1a,1a,1a,2,1a,2,2,2,1b)	(30,1,1)	398504	(2,2,1b,2,1b,1b,2,1b,2,1a)	10.44
36	(31,1,4)	624918	(1a,1b,1b,1b,1b,1b,2,2,1a,1a)	(34,1,1)	383880	(2,2,1b,1b,1b,1b,2,2,1b,1a)	38.57
44	(31,1,12)	395010	(1a,1b,1b,1b,1b,1b,2,2,1b,1a)	(1,1,42)	354632	(1a,1a,1a,1a,1b,1a,1b,1b,1a,1b,2)	10.28
52	(1,21,20)	437668	(1b,1a,1a,1a,2,1a,1,1,1,a2)	(1,1,50)	325384	(1a,1b,1b,1b,1a,1b,1b,1b,1a,2)	25.55

TABLE III

TESTING TIME AND WIDTH DISTRIBUTION OBTAINED FOR TWO TEST BUSES, ONE OF WHICH IS ALLOWED TO FORM INTO TWO BRANCHES FOR $\mathcal{S}_1$

W	ILP			GA			% Imp
	Distribution	Test Time	Bus Assg.	Distribution	Test Time	Bus Assg.	
24	(12,11,1)	1772909	(1a,1a,1a,1a,1b,1a,2,2,2,2)	(1,11,12)	1664230	(1a,1a,2,2,1b,1a,2,2,2,2)	6.13
36	(16,12,8)	1700237	(1a,1,1a,1a,1b,1a,2,2,2,2)	(20,15,1)	1618512	(1a,1a,1b,1b,1b,1a,2,1b,1b,1b)	4.81
40	(18,12,10)	1672119	(1a,1,1a,1a,1b,1a,2,2,2,2)	(4,35,1)	1598231	(2,2,1b,1b,1b,1a,2,2,2,2)	4.42
44	(17,15,12)	1682069	(1a,1,1a,1a,1b,1a,2,2,2,2)	(7,36,1)	1584896	(1a,1a,1b,1b,1b,1a,2,2,2,2)	5.78

TABLE IV

TESTING TIME AND WIDTH DISTRIBUTION OBTAINED FOR TWO TEST BUSES, ONE OF WHICH IS ALLOWED TO FORM INTO TWO BRANCHES FOR $\mathcal{S}_2$

problems were not attempted by other researchers. Figure 5 shows the architecture generated by our system for $\mathcal{S}_1$ with two test buses for $W = 44$ and bus 1 is allowed to have one subdivision. The 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 was 150 while the number of generations was chosen to be 300. The crossover and mutation probabilities were set to 0.35 and 0.65 respectively.

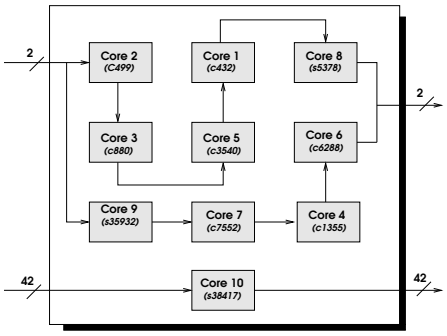

Fig. 5. Test bus architecture for System $\mathcal{S}_1$ with $W = 44$.

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

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

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

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

[5] V. Immaneni and S. Raman. Direct Access Test Scheme-Design of Block and Core Cells for embedded ASICs. In *Proc. ITC*, pp. 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. *Proc. ITC*, pp. 284–293, 1998.

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

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

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

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