Methods for System-on-Chip Test Design, Scheduling, and Optimization

By
Rana Farah

Submitted in partial fulfillment of the requirements
for the Degree of Master in Science

Thesis advisor
Dr. Haidar Harmanani

Computer Science and Mathematics Division
Lebanese American University
June 2006

LEBANESE AMERICAN UNIVERSITY

School of *Arts* and Sciences

Thesis Approval

Student Name	**RANA FARAH**	I.D.#:	**199830820**
Thesis Title:	***Methods for System-on-Chip Test Design, Scheduling and Optimization***		

Program:	**Computer Science**
Division /Dept:	**Computer Science and Mathematics**
School:	**School of Arts and Sciences, Byblos**

Approved by:

Thesis Advisor:	***HAIDAR M. HARMANANI***
Member	***DANIELLE AZAR***
Member	***MOUNJED MOUSSALLAM***
Member	

Date: JUNE 2, 2006

Abstract

In this thesis we tackle three problems related to System-on-Chip. The first problem deals with test pattern generation. The purpose is to generate a set of test patterns that catch all the faults in a core. The problem is solved using a dynamic ants based algorithm. The second problem, deals with test scheduling for System-on-chip. The problem was to test the cores of the SOC with minimum time, given a fixed allocated TAM and a given power consumption for every core, and the following constraints, the instantaneous TAM width and power consumption has a total maximum limit, the core are also ruled by precedence and concurrency constraints. In this thesis we adapt a sessionless scheme to minimize idle slots. Finally, the third problem that we solve is defined in a similar way to the second problem, but with the difference that the allocated TAM width for each core is not fixed and should be calculated efficiently. The last two problems are solved using the simulated annealing technique. We present experimental results to compare our work to other previous ones.

To my parents

Acknowledgements

At this stage of my studies, I would like to thank all who contributed to my work in this Thesis. First, I thank my advisor Dr. Haidar Harmanani for his support and guidance throughout my MS studies and my work on this thesis. I would also like to thank Dr. Danielle Azar and Mr. Munjid Musallem for their encouragement and for being on my committee.

I would also like to thank the Lebanese American University, whose financial support all through my studies made it possible

Thanks are also due to my family and friends for their unconditional support.

Table of contents

Title **Page**

List of Tables

Title **Page**

List of figures

Title **Page**

Chapter 1

Introduction

1.1 Embedded Cores:

The design and manufacturing methods of integrated circuits (ICs) have known advances, allowing the creation of complete systems onto a single die. Such systems, until recently, consisted of multiple ICs on a printed board. These large ICs are often called systems on chip (SOCs) or systems ICs. SOCs present several advantages over the traditional multi-chip alternative such as higher performance, lower power consumption, and smaller volume and weight. The reuse concept also reached the SOC technology, and it consists of the reuse of large predesigned and preverified building blocks, and IC-specific modules. The large reusable building blocks are called embedded cores. The advantages that cores offer, is that they shorten the time to market, and they allow the use of external design expertise [1].

In SOC manufacturing, the core provider and the core user are two different entities. They could be two different teams in one company, or even two companies, one that design the core and the other that uses it. Furthermore, A SOC may contain a collection of cores that has different origins.

There are three types of cores, *hard*, *firm*, and *soft* cores. These three types offer the user various design trade-off possibilities. When dealing with soft cores, a big part of the implementation is left to the user: Furthermore soft cores are flexible and process-

independent. Hard cores on the other hand lack the above flexibility, and are optimized for predictable performance and/or power consumption. Finally firm cores are a compromise between hard and soft cores [1].

Hard cores are provided to the core user as black boxes and are provided as an optimized layout due to their high performance and/or design complexity. A hard core includes layout and technology-dependent timing information and is ready to be dropped into a system [3]. Examples of hard cores are microprocessors, phase locked loops and mixed signal blocks [2].

Firm cores on the other hand, are offered without layout information, in fact they come as synthesized netlists after a logic synthesis and technology mapping. Firm cores do not have to be re-synthesized by the user, and are offered in the form of a hardware-description language (HDL) netlist, which can be changed or simulated by the user if it was needed [2]. A firm core contains more structure than a soft core; typically a gate level netlist that is ready for placement and routing [3]. Finally, soft cores are provided in the form of register-transfer level HDL, where the user is responsible for the synthesis and the layout [2]. A soft core can be retargeted to different semiconductor processes. [3].

Compared to the manufacturing of chips, which are tested before being used on a board, a core can only be tested after being integrated in the host SOC. As a matter of fact, internal testing consists of two matters: 1) the internal Design For Testability (DFT) structure, that typically consists of scan chains or test points, and 2) the test patterns associated with the DFT. The test patterns could be generated on chip for the internal

Built-In Self-Test (BIST) design or off chip for the external test design. The test vectors are applied and collected at the peripheries of the core. [3].

Given the fact that the core designer could be other than the core user, the core user is usually unaware of the design of the core, hence can't design tests that are core specific, nor he would know what test vectors to use, especially when dealing with hard cores. Thus, the core designer duty should provide the DFT structure and the test patterns along with the core. On the other hand, the core designer determines the core's internal test requirement, despite the fact that he may be unaware of the target process and application. Many parameters would be unknown to the core builder, as the test method that should be adopted, being BIST, scan, IDDQ, or functional test, the type of faults, being static, dynamic or parametric, and the desired level of fault coverage, which could lead to an inadequate design. If the manufacturing cost and the quality of the chip are determined by a low fault coverage, this would lead to a bad quality chip. On the other hand, if the fault coverage is high, this would be accompanied by a high power consumption, long test time, extra silicon area, and low performance [3].

On the other hand the core user task would be to translate the given pre-computed test, provided by the core designer at the core terminals, to chip level tests, described at the IC pins. This processed is called test expansion in [4]. In addition, the core user has to create a test access pass from the IC pins to the core and vise versa. This test access path should provide enough bandwidth to accommodate the requirement of the core. In fact the core user does not only have to take care of testing the cores of the system, but also to develop

test for the entire system. The test should include the cores as well as the User Defined Logic (UDL) and the interconnect wiring. On top of that, the test should be designed to be at the same time sufficiently effective and efficient and executable on the designated test equipment. While designing the test infrastructure, there are many options for trade offs between the ease of use, the test quality, the test time, and the silicon area. Another trade off is presented while scheduling the test. This trade off takes place among the ability to execute the test without conflicts, power dissipation, overall test time, and the silicon area spent on DFT [1].

1.2 System-On-Chip Test Methodologies

Marinissen and Zorian presented in [1] a testing architecture that consists of three elements, the test pattern source and sink, the test access mechanism (TAM), the core wrapper.

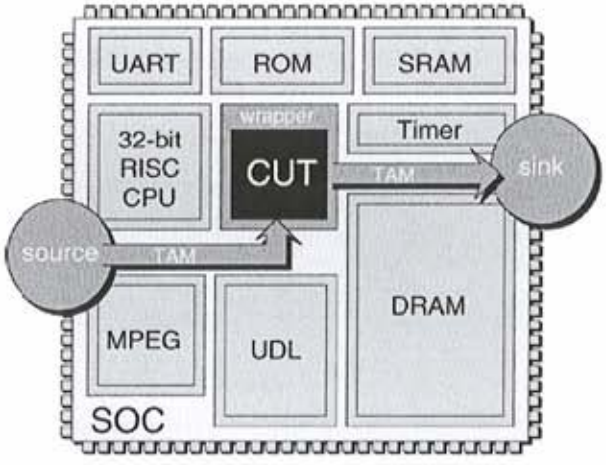

Figure 1: E. J. Marinissen et al. Testing architecture

First, the test pattern sources are the elements that are responsible for generating the test stimuli, while the test pattern sinks are the elements that are responsible for receiving the test responses. The test pattern sources and stimuli come mainly in two forms, on chip and off-chip. The on chip sources and sinks for the data generators and the response evaluators come in the BIST architecture. As for the off-chip sinks and sources, they are provided by the traditional automatic test equipment (ATE). Both forms present advantages and disadvantages. The ATE systems present difficulties in achieving the frequency and timing accuracy that is required to test the SOCs, and to build suitable ATE systems, and require a quite large capital investment. ATEs also require that TAM structure from the IC pins to internal core terminals and vice versa. On the other hand the on chip version of sinks and sources present the advantage of being very close to the cores under test which makes the TAM very short, which means less silicon area allocated for TAM. On the other hand the BIST structure itself occupies some silicon area. In addition, all kinds of test patterns can be generated on-chip, but in practice only algorithmic patterns, such as the regular patterns for memories, functional patterns for analog modules, or pseudo-random patterns for random logic can be generated on-chip without requiring an excessive amount of silicon area. Finally traditional BIST structures produce pass/fail verdicts only [1].

1.2.1 The Test Access Mechanism

Some kind of architecture should take care of test data transfer from the test pattern source to the core under test then to the sink. This is achieved by the use of the TAM. Two parameters affect the TAM design: the width and the length. The TAM width defines the TAM transport capacity, while the TAM length defines the physical distance it has to cross between the source, the core and the sink [1].

According to Sandeep and Marinissen [1], given a set of cores and a number of test pins, an access test architecture should be designed with respect to factors as test time, area cost and case of use. To achieve this, four parameters have to be determined for a given set of cores and a given number of test pins:

1. The number of TAMs,
2. The TAM widths,
3. The assignment of cores to TAM
4. The wrapper design.

The total number of cores and some core parameters such as the number of I/O terminals the number and length of the internal scan chains, and the number of test patterns per core, determine the number of TAM and their widths. The connection of the cores to the TAM could be done in three ways depending on the system requirements. First, each core gets its own TAM, which means that each core gets a partial fraction of the total available TAM width. This allows for each core to be tested independently of the other cores, and the cores could be also tested in parallel. Second, all cores could be connected to the same TAM. In this way all cores get access to the full TAM width, but cores are not tested independently. Third, a hybrid architecture could be derived of the two previous

ones. As an example we can mention three main architectures. The first one, the Multiplexing Architecture shown in Figure 2(a). In this architecture, only one wrapper can be accessed at a time making the core testing total time equal to the sum of the time allocated to each core. Due to the fact that the core external testing requires test access to two or more wrapper at the same time, the testing process becomes cumbersome in the least not to say impossible. The second architecture the DaisyChain Architecture, does not have the restriction of the previous architecture. Access to multiple or all core simultaneously is allowed. When testing in the DaisyChain Architecture it is best suited to operate the wrappers in parallel, and when a wrapper test patterns are exhausted the wrapper is switched to bypass mode. Both Daisychain and Multiplexing Architectures are full

TAM access architectures. The third one, the Distribution Architecture, is designed to test the wrapper as much as possible in parallel. The portion of TAM assigned to each core is proportional to the amount of data that should be transported to and from the core. The purpose of this design is to minimize the SOC testing time and to reduce it to the maximum of the individual core testing times. On the other hand the testing time of each individual core is larger than the previous two architectures, because the TAM allocated to each TAM is smaller. Sandeep and Marinissen [5] proposed a new architecture known as the TestRail Architecture, which is a hybrid form of the Daisychain and the Distribution Architecture. The Architecture is formed of one or several TestRail, in parallel on a single SOC and which operate independently. Each TestRail entity resembles the Daisychain Architecture where the core connected to a TestRail could be tested simultaneously or sequentially. This architecture could be used efficiently, when

all the cores are operated simultaneously, and each time a core runs out of test patterns it is switched to bypass mode, which make the total test time of the SOC as being the maximum of it TestRails testing times.

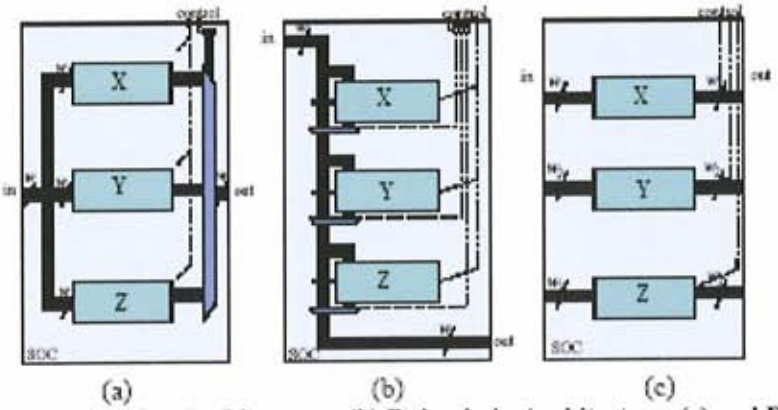

Figure 2: (a) Multiplexing Architecture, (b) Daisychain Architecture, (c) and Distribution Architecture

1.2.2 The Wrapper

Wrappers form an interface between the core and the host SOC and operate in three different modes:

1. Normal operation that represents the normal mode of operation of the core. The core is connected to its host core and the wrapper acts as if it was transparent.
2. Core-internal test mode: This mode is used to test the circuitry of the core itself. The connection of the TAM to the core is done in such a way that the test stimuli are applied at the cores input and the test responses are collected at the cores output.
3. Core-external test mode: In this mode it is the interconnect wiring and logic that are under test. The TAM is connected to these components in such a way that the

test stimuli are applied on the output of the core and the response is collected on the input of the next core.

A wrapper could operate in other modes such as detached which detach the core from it host SOC and the TAM, or the bypass mode for the TAM. The wrapper is designed to connect the core terminals to the TAM. The number of the core terminals is determined by the core functionalities and application, as for the TAM width, it is determined by how much silicon space the SOC builder is willing to allocate to it and to the bandwidth of the source and sink. If the TAM width is smaller than the core terminals, some sort of adaptation is done in the wrapper, through serial-to-parallel conversion at the core inputs and parallel-to-serial conversion at the core outputs [1].

The IEEE P1500 Standard for Embedded Core Test, designed a standard for the core test wrapper. The IEEE P1500, has a scalable width TAM and supports both TestBus and TestRail TAM architecture. Furthermore, the normal mode, InTest mode, ExTest mode, and the ByPass modes are supported by the IEEE 1500 standard.

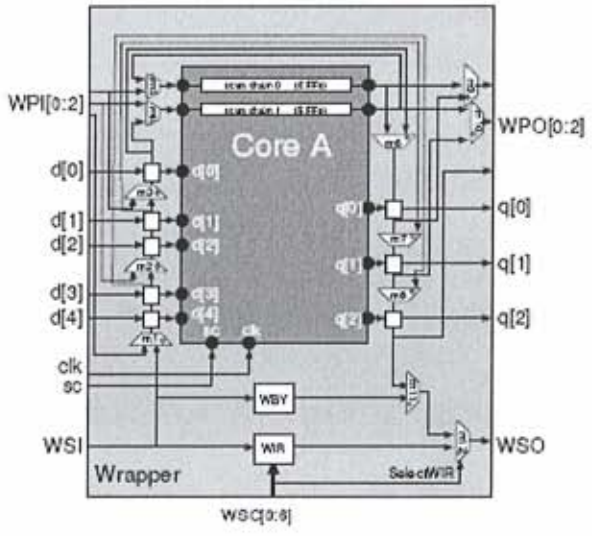

Figure 3: The IEEE P1500 Core Test Wrapper[21]

1.3 Test Pattern Generation:

Testing logic circuits for faults requires a set of specific test patterns that are capable to catch all the detectable faults in the circuit. Several methods were designed to generate test vectors, and they can be categorized as follows:

- Deterministic methods for automatic test pattern generation (ATPG) such as D-Algorithm, PODEM, FAN, Socrates, among others.
- Heuristic methods based on simulated annealing as well as genetic algorithms. This includes examples such as SAARA, GATTO, and Rudnick among others.
- Hybrid methods which consists of the combination of a deterministic algorithm and a heuristic algorithm. Such as Saab.

Test pattern generation includes some common steps that are summarized as follows. Given a logic circuit, inject a fault at a certain site. The test pattern is fed back to the circuit and the output is compared with the expected true value. If they are different, then the test pattern sentisize the fault and is accepted. The following circuit is taken from [20].

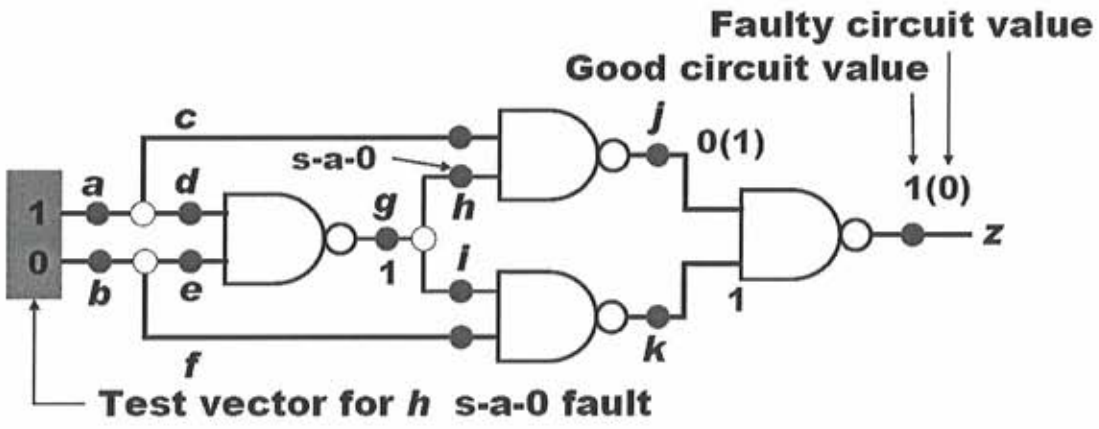

Figure 4: A circuit with a fault [20]

In the figure above, a stuck-at-0 fault is injected at fault site "h." The test pattern "10" is applied to the circuit. As it is shown, 1 is the true output of the non-faulty circuit if "10" was applied at the primary inputs. But given the test pattern "10" and the stuck-at-0 fault at "h", the value of the output differed, that is why we can conclude that the test pattern "10" can catch the fault stuck-at-0 fault at "h".

1.4 The Simulated Annealing Algorithm

Simulated Annealing is an optimization technique applied to combinatorial optimization problems in order to find an approximated solution [13]. Some argue that the simulated annealing algorithm, if given enough time and the correct parameters, it should give the optimum solution.

Simulated annealing operates as follows an initial feasible configuration is first generated. A neighborhood solution is next created by perturbing each time the current configuration. Finally, the cost of the resulting configuration is calculated. If the cost is less than that of the current configuration, the new configuration is accepted and it

replaces the current configuration, otherwise a certain probability is calculated and the new configuration is accepted or rejected according to that probability.

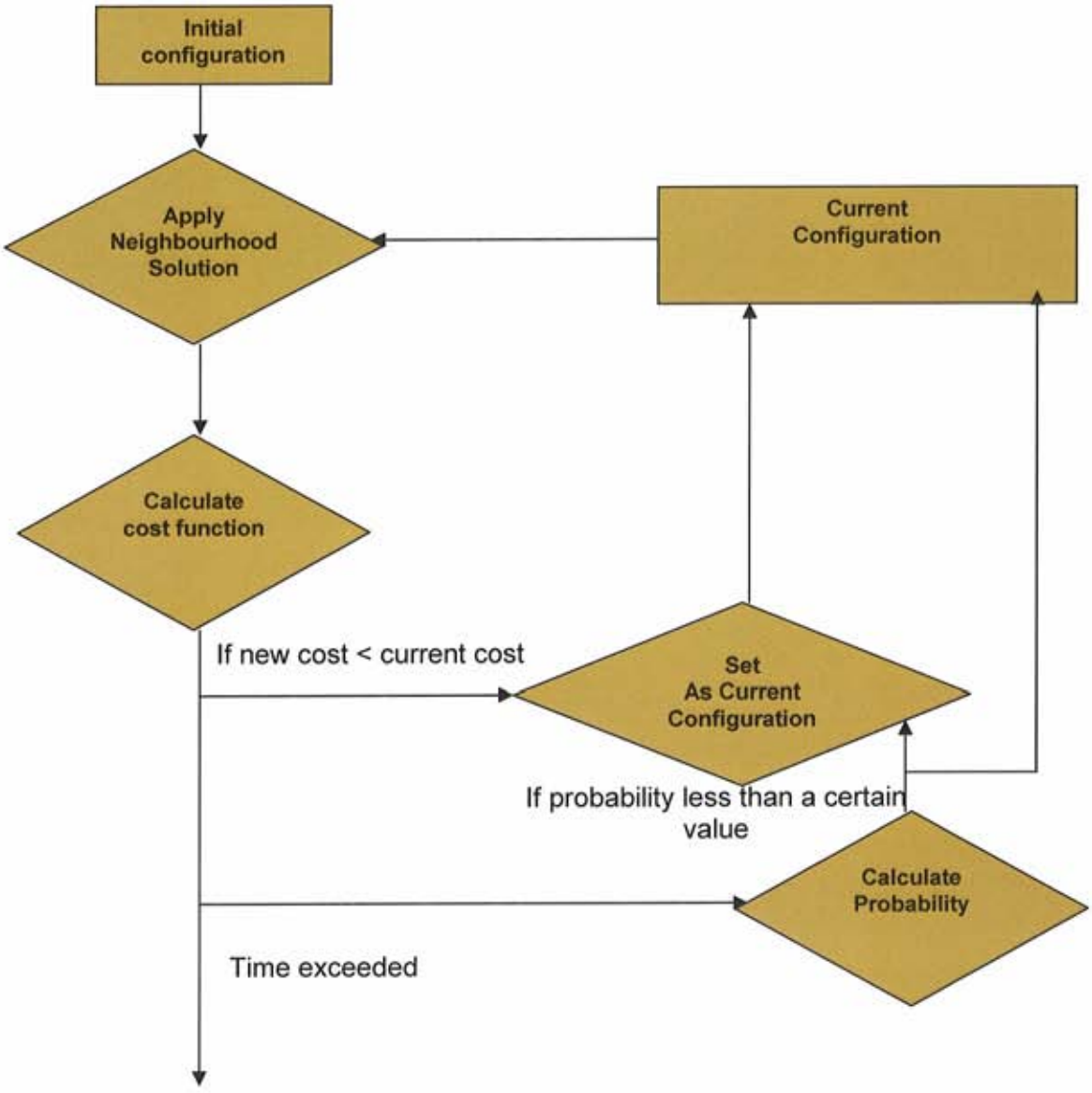

Figure 5: The Simulated Annealing Algorithm

1.5 Problem Description and Thesis Outline

In this thesis we will tackle three related problems that deal with SOC testing and test scheduling. The first problem, that is presented in chapter 3 deals with core's test generation. The second problem, that is presented in chapter 4 deals with test time optimization with concurrency, power and precedence constraints and with fixed TAM partitioning. The third problem is presented in chapter 5 and it tackles test time optimization with concurrency, power and precedence constraints with variable TAM partitioning. Finally, we conclude with remarks and future work in chapter 6.

Chapter 2

Related Work

Pouget et al. [6] dealt with the issue of test optimization by the developing an approach that incorporates a new wrapper design, taking into consideration the P1500 restriction. In addition they proposed a TAM architecture and a test scheduling method. The wrapper was designed using a fast algorithm that attempt to optimize the test time for a given number of connections to the TAM. In addition, the algorithm gives the possible connections to the TAM. Later, one of these possibilities is chosen for TAM design and test scheduling. The team developed a fast heuristic method for TAM design and the test scheduling. The method took into consideration the precedence issue, the power dissipation restriction, and the interconnection test.

Iyengar et al. [7] developed a new technique for the wrapper and TAM co-optimization. Their technique is based on rectangle packaging and used the Pareto-optimal points in order to distribute the TAM width over the cores according to their data needs. Idle time was also minimized by using several heuristics, which lead to a fast and efficient algorithm for TAM width allocation and test scheduling. The down points of this technique is that is doesn't deal with power dissipation, precedence, or interconnection test.

Koranne [8] presented a test plan for the application of tests to the SOC, where a test plan indicates TAM partitioning and test scheduling. Koranne proved that the scheduling

problem can be mapped onto a graph theoretical problem which has a polynomial time optimal solution. Thus, the test planning problem was mapped to the minimum weight perfect bipartite graph matching problem. In his work, also Koranne didn't take into consideration the power dissipation issue or the interconnection test.

J.-K,Wuu et al. [9] approached the test scheduling problem and the core wrapper design by taking into consideration the power dissipation constraints. In their approach they extended the wrapper design method presented by Zou et al. [15] and solved the test scheduling problem by mapping it into a floor planning problem. The height of the floor plan represents the TAM width and the width represents the test time, which should be reduced. The maximum height of the floor plan is fixed since the width of the TAM is given. The floorplan is represented using a binary tree, and a two stage simulated annealing method is applied on it. A penalty value is added to accommodate power constraints. This approach did not take into consideration the precedence problem or the interconnection test.

Larsson and Fujiwara [10] proposed a wrapper that is based on the design of a scan chain that has a flexible length for a hard core that can achieve a flexible TAM bandwidth at each core and also that can control the test power dissipation at each individual core and make it possible to control the clock frequency. They also proposed a preemptive test access mechanism (TAM) scheduling taking into consideration power constraints. The problem was tackled based on a bin packing model. The algorithm didn't take into account the preceding issue or the interconnection test.

Zhao and Upadhyaya [11] considered the problem of optimizing the overall testing time of SOC. Their algorithm takes into consideration the constraints on the total power consumption, and the maximum level available for the TAM. Their algorithm starts by building power constrained concurrent test sets (PCTSs) based on a power-constrained test compatible graph (P-TCG). PCTS also named conflict graphed is used to derive the set seeds to facilitate efficient scheduling. Then the TAM width has been dynamically assigned following the adaptive assignment, and the idle time has been reduced by dynamic partitioning. Once more the algorithm didn't take into consideration the precedence issue or the interconnection test.

Harmanani and Salamy [12] [13], worked on test scheduling using power constraints. In [13] the problem was solved using a genetic algorithm. While the work was extended in [12] to incorporate precedence constraints and was solved using simulated annealing. [12] took into consideration the precedence problem. While both papers, minimized the idle time, neither paper took into consideration the TAM width constraints.

W. Zou et al [15] proposed a SOC test scheduling method based on simulated annealing. The cores were represented as a two dimensional bins, and the problem was reduced to solving a two dimensional bin packing problem. A data structure called sequence pair was used to represent the placement of the bins. The neighborhood solutions that were used consisted of altering an initial sequence pair and changing the width of the core

wrapper. In addition a new method for wrapper design was proposed. The paper ignored precedence and concurrency constraints as well as power constraints.

Chapter 3

Test Generation for Combinational Cores

This Chapter deals with test generation for combinatorial cores. The problem of test generation is not a new one and has been shown to be NP-hard.

3.1 Solution Approach

A variety of methods has been employed for the generation of test vectors. We, however, are going to use a random generation technique. More exactly the dynamic ant algorithm.

3.1.1 The Dynamic Ant Algorithm

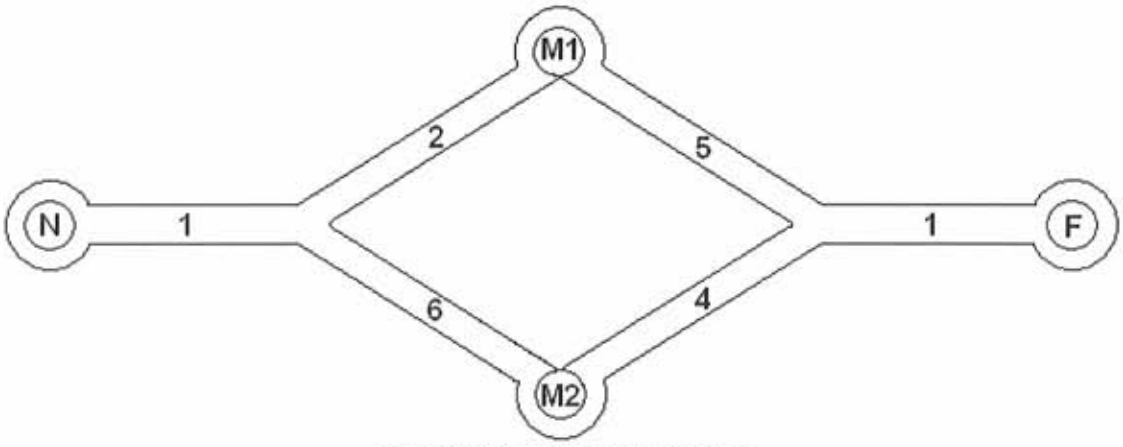

Figure 6: The Dynamic Ant Trail

Suppose we have a colony of ants, and food lying at a distance from their nest. On the graph (Figure 6) the ants' nest is marked by "N" and the food is marked by "F". There are two routes that the ants can take to get to the food one passing through "M1" and the second passing through "M2". While walking from their nest to the food ants deposit on

the ground a substance called pheromone. Ants can smell this substance, and while traversing the route they choose the with a certain probability the route that has the strongest concentration of pheromone on it. Going back to our graph we can see that the route going through M2 has a higher cost than the one going through M1. If we translate this to the ant algorithm, route M2 is longer than route M1. At the start when no ant has traversed either route, there would be no pheromone and an ant would have a probability of 50% to choose either route. But the ants choosing the shortest path would do more traversal than the ants choosing the longer path, thus depositing more pheromone on the route, and the probability of any ant choosing this shortest path will grow with time, and the ants will end up favoring this route always.

This behavior was simulated in the following way, the ant should traverse a graph data structure with a starting node and a destination node. Each edge have a weight, which stands for the cost to cross that edge, and the edge also holds another variable that is the quantity of pheromone deposited on that edge. The quantity of pheromone is increased each time an ant choose that edge to be part of its route. As for the ants each time an ant has to choose a trail it decides which edge to choose by calculating a probability depending on the concentration of pheromone on that trail.

3.1.2 Dynamic Ant and Pattern Generation

We apply the dynamic ant algorithm to our problem as follows. Each primary input of the SOC is modeled by two edges holding the number of the input and one of them holding the binary value 0 and the other one holding the binary value 1. Thus, we represent the two possibilities for the binary value that each primary input could hold. Each edge is connected to the two edges that represent the pin with the higher number. The connections are shown in Figure 7. The vertices that connect the edges are weights that represent the quantity of pheromone deposited on this trail by the ants that chose it. This resulting data structure will be the graph that the ants should cross. Each crossing by, one ant, on the graph would result in a candidate test vector. This candidate test vector would be given to the circuit and a fault simulation on the circuit would result in a number of detected faults. The weights of the crossed edges would be updated accordingly.

At each level of the graph an ant has to choose between two vertices, that represent the same pin, but each one holds a possible value for the input, 0 or 1. An ant would choose one of the two pins based on the trail's probability, which is calculated as follows:

$$p = \frac{T_1}{T_2 + T_1},$$

where T_1 is the weight of the candidate trail, and T_2 is the weight of the adjacent trail.

The trails are updated as follows, $T_{ij}(t+n) = p.T_{ij}(t) + \Delta T_{ij}$,

where $\Delta T^k{}_{ij} = \begin{cases} Q \times L & \text{if the k}^{\text{th}}\text{ ant uses edge(i, j) in its tour between time t and t + n} \\ 0 & \text{otherwise} \end{cases}$

where Q is a constant chosen to be **0.06** in our case and L being the number of faults detected by the ant in this pass through the graph. The value of Q was determined experimentally.

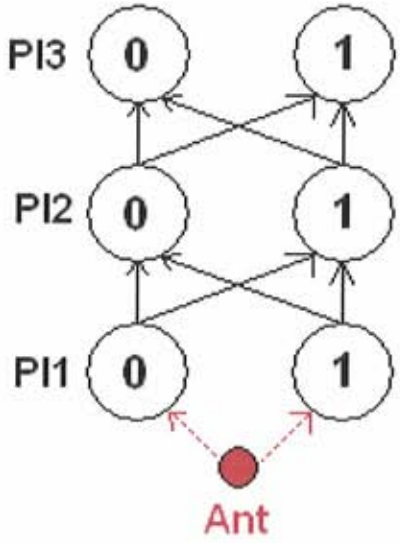

Figure 7:The graph that represents the pins of a SOC

To illustrate the ant algorithm, the following example is applied on the circuit shown in **Figure 8**.

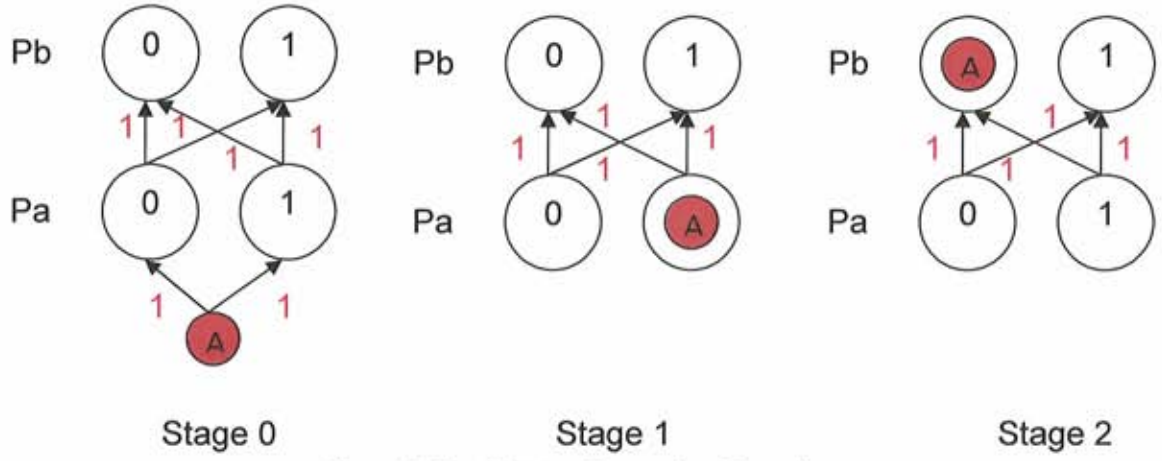

Figure 8: Test Pattern Generation Example

In Stage 0 the "Ant" has the choice to take the route to Pa-1 or Pa-0. For the first iteration the weights of both of the routes are set to 1, which means that the ant has an equal

change to take any of the two routes. In our case the "ant" chooses Pa-1, which takes us to Stage 1. The same case is presented for Stage 1, but this time the "ant" chooses Pb-0, which takes us to Stage 2. At stage 2, we have a completed test pattern, which is fed to the circuit.

When the test pattern is fed into the circuit, a concurrent fault simulation takes place. At first the circuit is marked with the true values (non faulty) at each fault site, the circuit is shown in Figure 9.

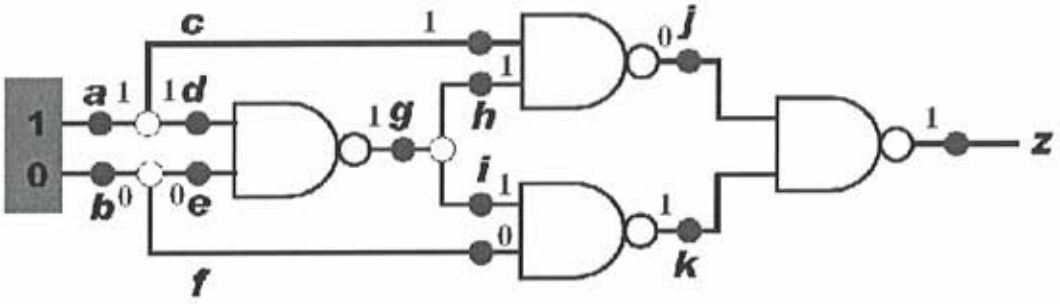

Figure 9: Test Pattern Generation Example (stage4)

Then for each gate in the circuit, we compute the list of faults that is propagated until this point by the given test pattern.

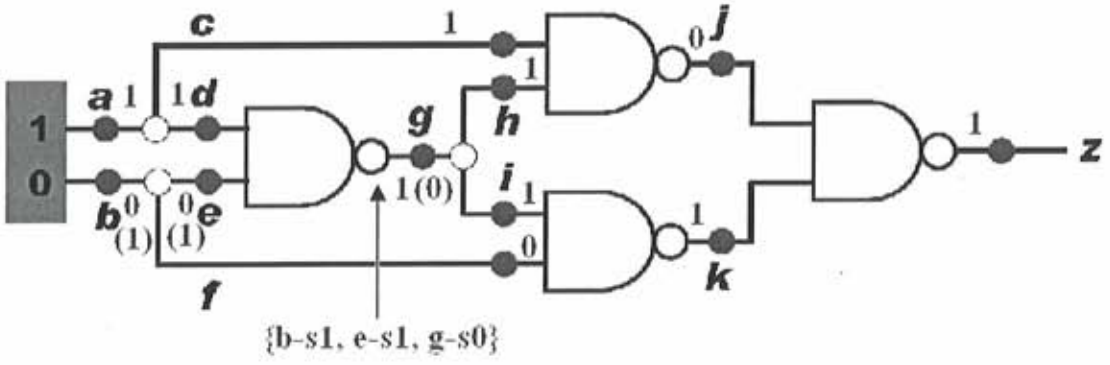

Figure 10: Test Pattern Generation Example (stage5)

Figure 10 shows the list of faults that the test pattern "1 0" has propagated till the first NAND gate. In fact if point "b" has a stuck-at-1 fault, the value of point "b" would be "1" instead of "0" which makes the value of point "g" to be equal to "0" instead of "1". The same applies for point "e". And a stuck-at-0 fault could be detected at point "g" if we can also test at that point. In this way, we would have computed the list faults detected at the output "g" of the first NAND gate.

The same procedure is applied to the other gates.

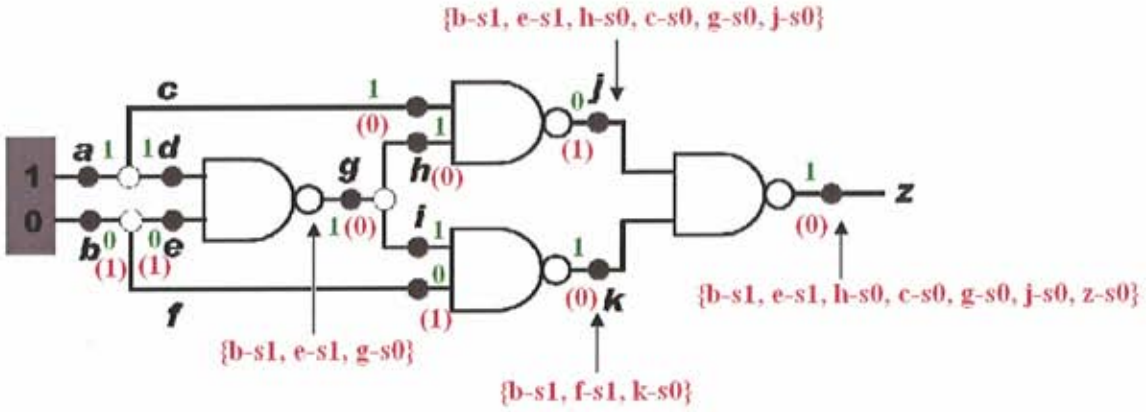

Figure 11: Test Pattern Generation Example (stage6)

The list of faults that could be detected at the primary output of the circuit is computed in the same way, and would be the list of faults that the given test pattern could catch.

Figure11 shows, at point "z" the list of faults that the test pattern "1 0" has caught. We can count 7 faults.

Accordingly the weight of the route that the ant has crossed is updated.

$$T_{ij}(t) = 1$$

$$p = 0.5$$

$$\Delta T_{ij} = Q \times L = 0.66 \times 7 = 4.62$$

$$T_{ij}(t+n) = p.T_{ij}(t) + \Delta T_{ij}$$

$$T_{ij}(t+n) = 0.5 \times 1 + 4.62 = 5.12$$

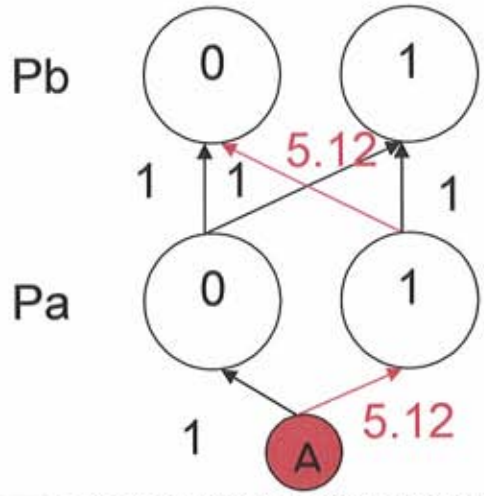

Figure 12: Test Pattern Generation Example (stage7)

```
Ant_Algorithm_Test_Pattern_Generation
{
  MaxIterations ← Maximum number of time that the Ants will cross the graph
  iterations ← 0
  BuildGraph
  while (iterations < MaxIterations)
  {
    while (there are more levels)
    {
      if(random < probability(edge0))
      {
        Ant choose edge0
        add 0 to test pattern
      }
      else
      {
        Ant choose edge1
        add 1 to test pattern
      }
  }
    send the test pattern to the test simulator
    update the test pattern set
    update the weight on the edges of the graph
  }
}
```

Figure 13: Test Pattern Generation Pseudo Code

In order to verify our algorithm, we run various benchmark circuits, which results are shown in Table 1. We compare with results from [14], [22], [23], [24], and [25] were different strategies were used to solve this problem.

Table 1: The Results for the Ant Algorithm

		# detected Faults						
Circuit	**# of detectable faults**	**Ant**	**[23]**	**Random[23]**	**[22]**	**[24]**	**[25]**	**[14]**
c17	22	22	-	-	-	-	-	-
c432	524	514	-	-	-	-	-	-
c880	942	905	942	942	942	942	937	942
c1355	1566	1496	1566	1566	1566	1566	1536	1566
c1908	1870	1734	1870	1870	1870	1868	1852	1870
c2670	2630	2300	2630	2357	2630	2526	2290	2630
c3540	3291	3170	3291	3291	3291	3288	3277	3291
c5315	5291	5242	5291	5291	5287	5291	5258	5291
c7552	7419	6865	7419	7270	7362	7356	7120	7419

Chapter 4

Test Time optimization with concurrency, power and precedence constraints with fixed TAM partitioning

Test Scheduling for SOC's falls under the category of NP hard problems. In fact, given a set of cores and a set of resources, such as TAM wires, and a set of test patterns, a set of test should be scheduled taking into consideration four main issues. First no two tests for the same core should be scheduled concurrently. Second, avoid test resources conflict. Third, take into consideration precedence and concurrency constraints. Fourth, the peak power and the overall TAM width should not exceed a given value. In this section of the thesis we assume that the TAM architecture and the wrapper design has already been done, and the remaining task is to schedule the tests for these core optimizing the time.

4.1 Problem Formulation

The problem that we solved in this part of the thesis is to optimize the total test time of a SOC, taking into consideration the limits on the maximum TAM width, the maximum power consumption, the concurrency constraints, and the precedence constraints. By precedence constraints we mean that for certain reasons certain cores should not be tested before some other cores. As for concurrency constraints, certain cores could not be tested concurrently. In fact our target is to schedule as many core to be tested in parallel, which will reduce the test time, but it would be governed by the previously mentioned constraints.

Of course in this part we assume that the wrapper as been designed and the TAM bits allocated.

4.2 Solution Approach

Our core will be represented by 3-D pins shown in Figure 14, where one dimension represents the time that it would take one core to finish its testing period. The second dimension represents the number of TAM bits allocate to this core. Finally, the third dimension represents the maximum amount of power consumed by the core. Our task is to arrange these cores (3-D bins) in a 3-D architecture in a manner that the dimension allocated to the time would be the least possible, while the other two dimensions won't exceed certain prefixed values, while once more taking into consideration the precedence and the concurrency constraints. In addition a Core has an assigned start time and a resulting finish time.

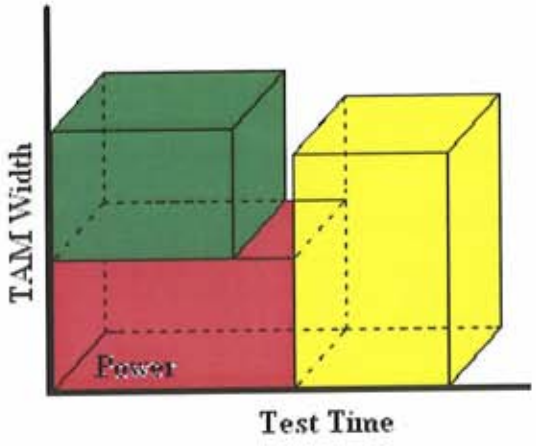

Figure 14: 3-D representation of a core

4.3 The Simulated Annealing Applied to Test Scheduling

A core is characterized by the following: a start time, a finish time, a TAM width that is allocated to it, its individual test time, and the power that it emits.

4.3.1 The initial configuration

The initial configuration is achieved by scheduling the cores in a serial fashion, which means that all the cores are tested subsequently.

4.3.2 The neighborhood solutions

Two neighborhood solutions are used in our algorithm.

The first neighborhood solution consists of choosing a core at random and setting its start time to be the finish time of another randomly chosen core, taking into consideration the constraints cited above.

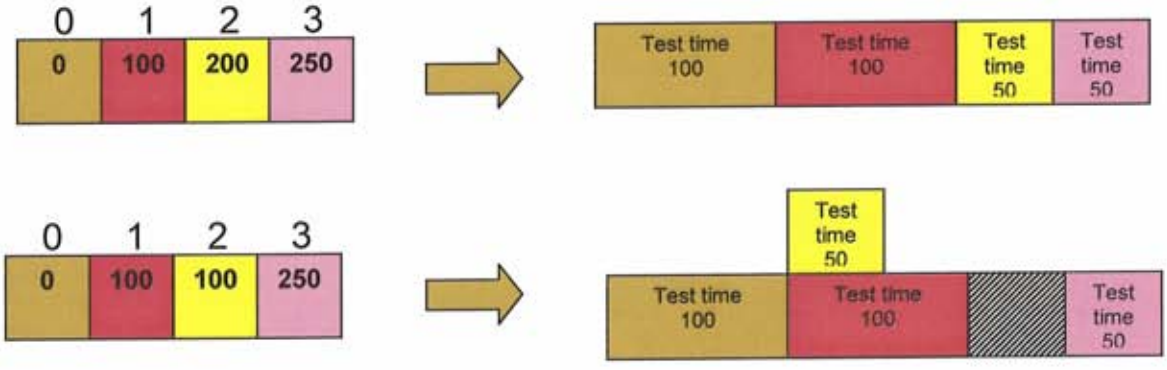

Figure 15: Example of neighborhood solution 1

Figure 15 illustrate the first neighborhood solution. Each of the configuration that are displayed corresponds to a vector were each cell corresponds to the start time of a core. Core 2 was chosen at random, and its start time was set to be the finish time of Core 0, which was also chosen at random.

The second neighborhood solution consists of randomly choosing two cores and switching their start time, while taking into consideration the precedence and concurrency constraints.

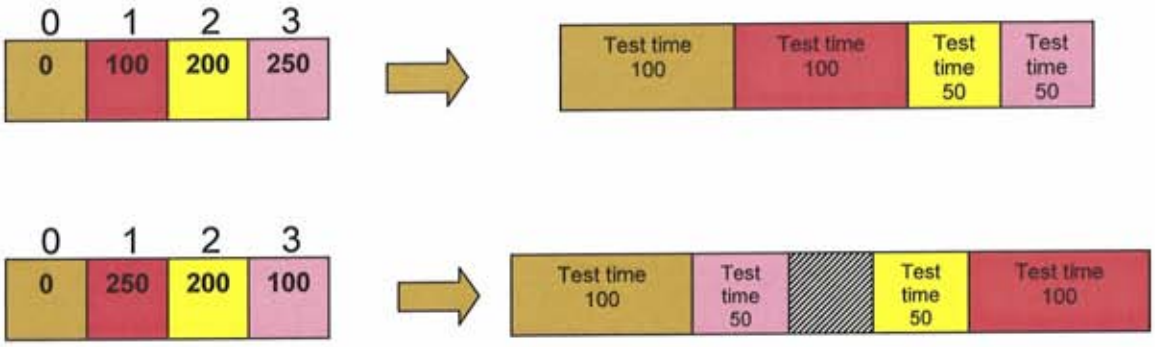

Figure 16: Example for neighbourhood solution 2

Figure 16 illustrate the second neighborhood solution, where Core 3 and Core 1 were chosen at random, and their start times were flipped.

Cores that are governed by precedence constrains, are restricted from having their start time before the cores that should precede them. While cores that are governed by concurrency constrains, they are prevented from having interleaving test times.

4.3.3 The cost function

The cost function is designed to minimize the overall test time of the SOC. The cost function is chosen to be the finish time of the latest scheduled core.

```
Annealing_Test_Scheduling
{
  β = iteration multiplier
  S0 = initial Solution
  maxTime ← Total allowed time for the annealing process
  α ← Cooling rate
  M ← time until next parameter update
  T0 = initial temperature
  currentS ← S0
  currentCost ← cost(currentS)
  bestCost ← ∞
  T ← T0
  time ← 0

  while (time < maxTime)
  {
    while ( M > 1)
    {
      change ← false
      while (change = false)
      {
        newS = neighborhood (currentS)
        check for power. TAM. precedence and concurrency constrains
        if (no violation)
        {
          change = true
        }
      }
      newCost ← cost(newS)
      deltaCost ← newCost – currentCost
      if(deltaCost < 0)
      {
        currentS ← newS
        currentCost ← newCost
        if(new_Cost < best_Cost )
        {
          bestS ← newS
          bestCost ← newCost
        }
      }
      else
      {
                          delta_cost
        if (random < e      T       ))
        {
          currentS ← newS
          currentCost ← newCost
        }
      }
    }
    M ← M-1
  }
  time ← time + M
  T ← α *T
  M ← β*M
}
```

Figure 17: Pseudo Code for Problem 2

4.4 Results

The algorithm was tested on two SOC's chosen from the literature. In both cases the individual test time of the cores, the TAM width allocated to each core, and the power emitted by each core is supplied by the authors of the papers.

The first one was retrieved from [11], it consists of seven cores in Table 2, and a concurrency graph is also shown in Figure 18. Each node of that graph represents a core and each edge between the two cores indicates that these cores could not be tested at the same time. The number that is recorded next to each node of the graph is the power emitted by each core. For this SOC, the precedence constraints were not applied.

Table 2: Information related to the Core extracted from [11]

Core number	Assigned TAM width
c0	5
c1	18
c2	5
c3	7
c4	19
c5	30
c6	20

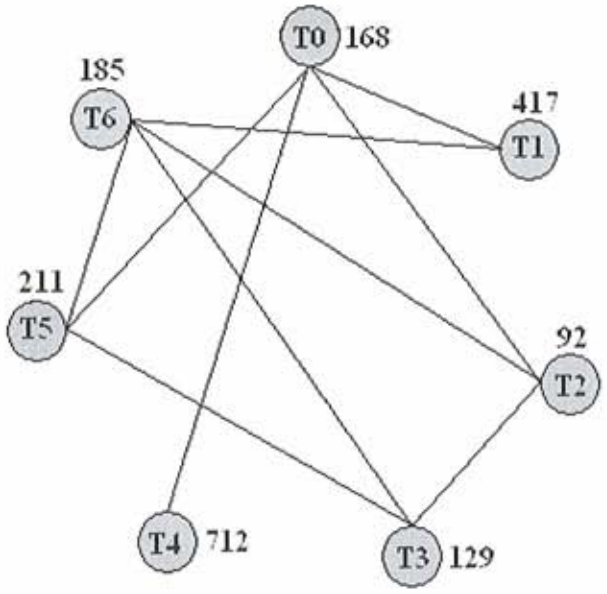

Figure 18: The concurrency graph related to the core extracted from [11]

After running our algorithm we got the optimum result, that is 10436, reported in [11], in 200 milli CPU seconds. The overall test time was calculated to be 10436 for a maximum power 880mW, while the maximum TAM width of 32.

The simulated annealing cooling schedule that was used for this SOC consists of the following: $\alpha = 0.99$, $\beta = 100$, $T_0 = 4000$, $M = 5$, which were determined experimentally.

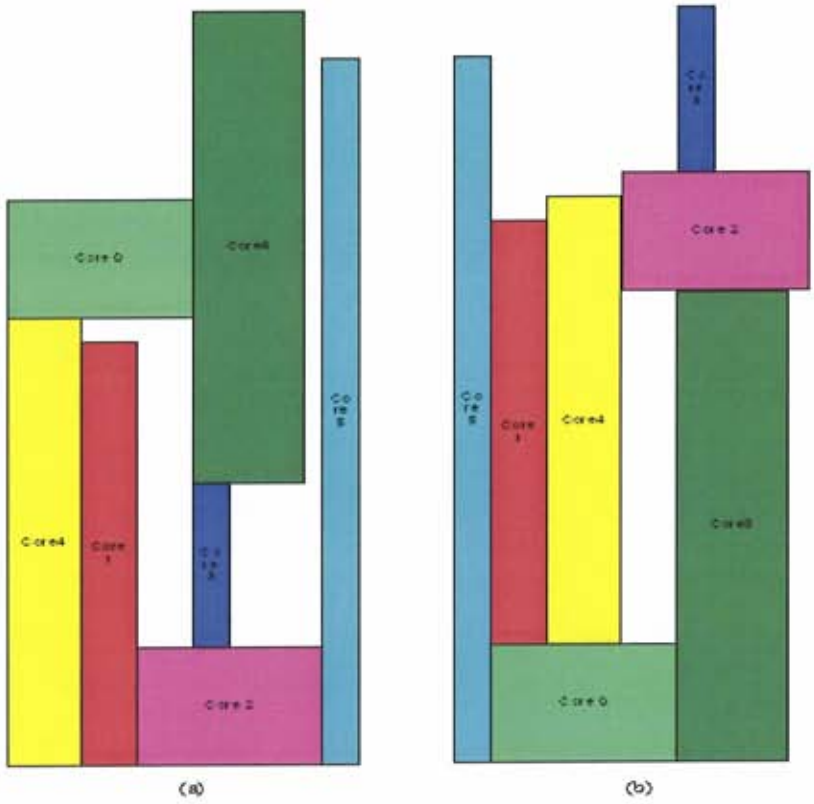

Figure 19: Results for the SOC described in [11] – (a) D. Zhao and S. Upadhyaya's Configuration – (b) our configuration

Our second example is the d695 SOC witch was used in [6]. In that paper the SOC was presented with precedence constraints but no concurrency constraints. The information related to the core are shown in Table 3. Last column, indicates that corresponding cores should not be preceded by the core of that row.

The simulated annealing parameters that were used for the SOC are the following: β = 1000, $T_0 = 4000$, M = 2, $\alpha_0 = 10$. α was decreased with every iteration by a factor of 0.5. These values were determined experimentally.

Table 3: The d695 Characteristics used in [6]

Core number	Test Time	TAM Width	Power (mW)	Precedence
1	416	2	30	-
2	7992	3	150	-
3	5167	2	250	-
4	11129	6	100	-
5	10100	19	400	-
6	9869	19	950	7,8
7	12959	10	700	-
8	4605	11	450	-
9	2820	19	350	-
10	7106	17	550	7,5

Our design was able to achieve better results than those reported in [6]. In fact for a maximum TAM of 32, and a maximum power of 1680w, the maximum test time was 56834. In our case for a maximum TAM width of 32 and a maximum power consumption of 1350 we got a test time of 29934.

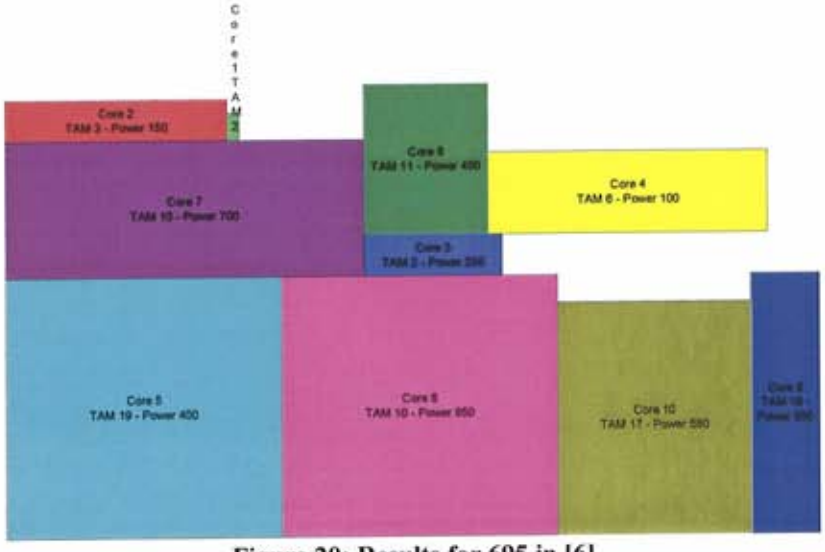

Figure 20: Results for 695 in [6]

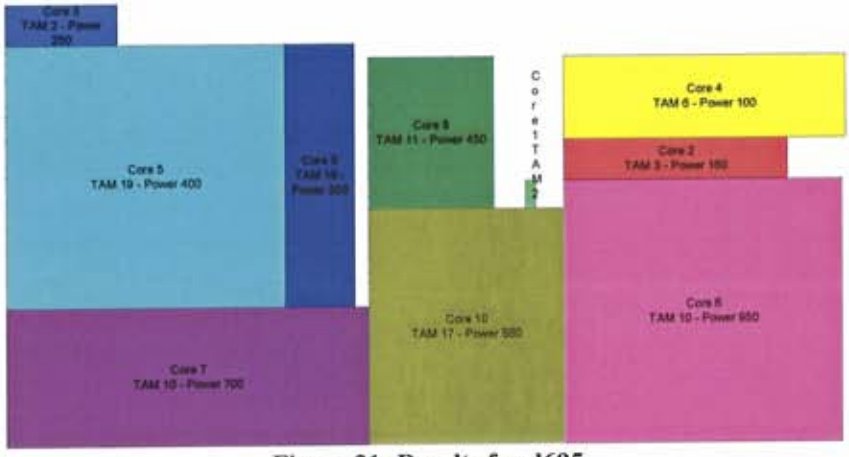

Figure 21: Results for d695

Chapter 5

Test Time optimization with concurrency, power and precedence constraints with variable TAM partitioning

5.1 Problem Formulation:

This chapter deals with test time optimization with concurrency, power, precedence constraints and variable TAM partitioning. In other words given the maximum TAM width allocated for a SOC, the TAM width should be partitioned among all cores, a test scheduling should be performed in such a way that the overall test time is reduced taking into consideration all the other constraints.

5.2 Solution Approach:

A core is represented with a 3-D bin. Its dimensions representing the individual test time, the individual power consumption and the TAM width. The way TAM width is allocated to each core is done randomly. In fact a random numbers that are less than the overall TAM width allocated to the whole SOC, are generated and assigned as TAM width to each of the cores.

5.3 Wrapper Design:

After being allocated with a certain TAM width w_i, we should create a wrapper design for the core in order to be able to calculate its individual test time.

A core has a certain number of inputs pins "pi", a certain number of outputs pins "po", a certain number of bidirectional pins "pb", and a certain number of scan-chains each one having a given length. The number of pins or scan chains would be, in general, greater than the number of TAM bits allocated for the core, which would require a wrapper to accommodate the difference. A wrapper consists of a set of scan chains whose task is to serialize the input of the core. Each bit of the TAM will be connected to a series of pins and internal scan-chains that form larger scan-chains. The test time of the core is calculated as follows:

$$t_i = (1 + \max[s_i, s_o])p + \min[s_i, s_o]$$

Where, "s_i" is the longest input scan-chain, "s_o" is the longest output scan-chain and "p" the number of test patterns applied on the core. This means, that to minimize the test time it would be better to have the most balanced combination of scan-chains possible.

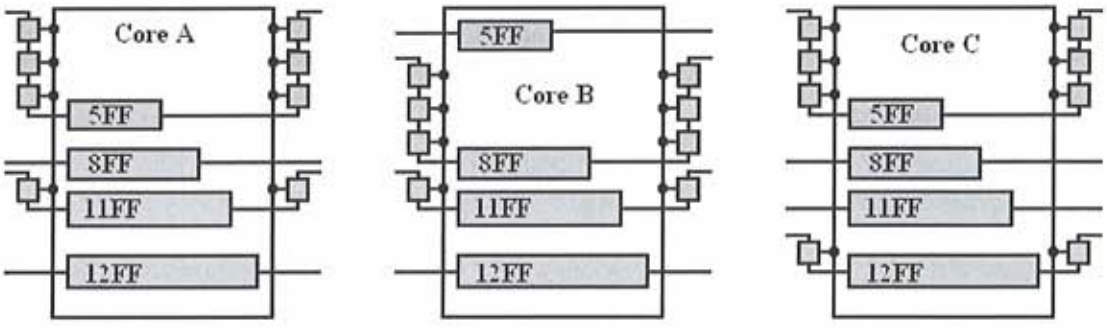

Figure 22: Wrapper Design Example

Figure 22 shows three similar cores. These cores are supplied by a 4-bit TAM. The core has 4 input pins 4 output pins, 4 scan chains of lengths, respectively, 5FF, 8FF, 11FF, and 12 FF. The wrapper design of the three cores differs. Core A and Core B, although they have different wrapper design, they will end up with the same $s_i = 12$ and $s_o = 12$, and consequently the same test time if they had the same number of test patterns. Core C has greater $s_i = 13$ and $s_o = 13$, and consequently a greater test time, if it had the same number of test patterns applied to the previous two cores.

The method that we propose to design our wrapper is as follows. Given that the lengths of the scan chains are sorted, distribute the scan-chains among the bits of TAM going from the first to the last then from the last to the first until all the scan-chains are assigned to the bits of TAM. This will result in a number of scan-chains that are equal to the number of bits of TAM assigned to the Core. These scan-chains are not necessarily equal in length. Then, when distributing the terminal pins among these scan-chains we first allocate these pins to the scan-chains that do not have the maximum length, to make their equal to the maximum length. The remaining pins are distributed equally, if possible, among the bits of TAM. This method will result in balanced length scan-chains, thus minimizing the test time, given the number of TAM bits. The following example illustrates the method that we used for the wrapper design.

A SOC is allocated with two bits of TAM. The SOC has 4 scan chains of length respectively, 12, 11, 8 and 5. In addition the SOC has four primary inputs and four primary outputs. Finally, the number of test patterns used to test this SOC is 100.

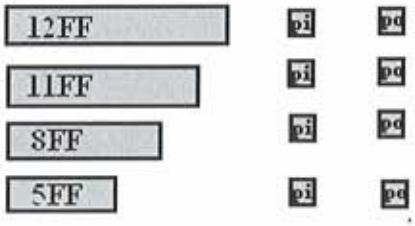

Figure 23: Wrapper Design Example (Stage 1)

The scan chains are first sorted by decreasing length. Then we allocate the scan chain of length 12 to the first TAM bit (TAM bit 1). The scan chain of length 11 is allocated to the second TAM bit (TAM bit 2).

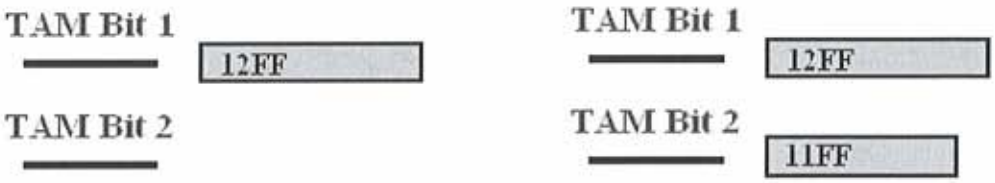

Figure 24: Wrapper Design Example (Stage 2)

The scan chain of length 8 is then allocated to TAM bit 2, and the scan chain of length 5 is allocated to TAM bit 1.

Figure 25: Wrapper Design Example (Stage 3)

Now our wrapper consists of two scan chain one of length 19 at TAM bit 2 and the other of length 17 at TAM bit 1. Next we add two primary inputs to TAM bit 1, to balance the scan-in-chains.

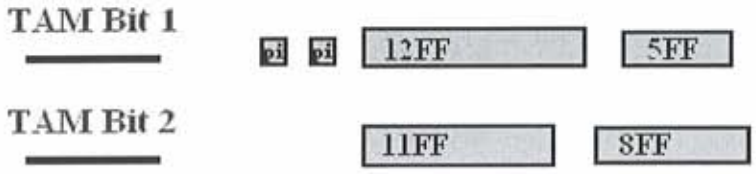

Figure 26: Wrapper Design Example (Stage 4)

Our wrapper, now, has two scan-in-chains of equal length of 19, we distribute the remaining two primary inputs equally among the two TAM bits, and the length of the two scan-in-chains will be 20.

Figure 27: Wrapper Design Example (Stage 5)

The same procedure is applied to the primary outputs. And the length of each scan-out-chain will, also, be 20.

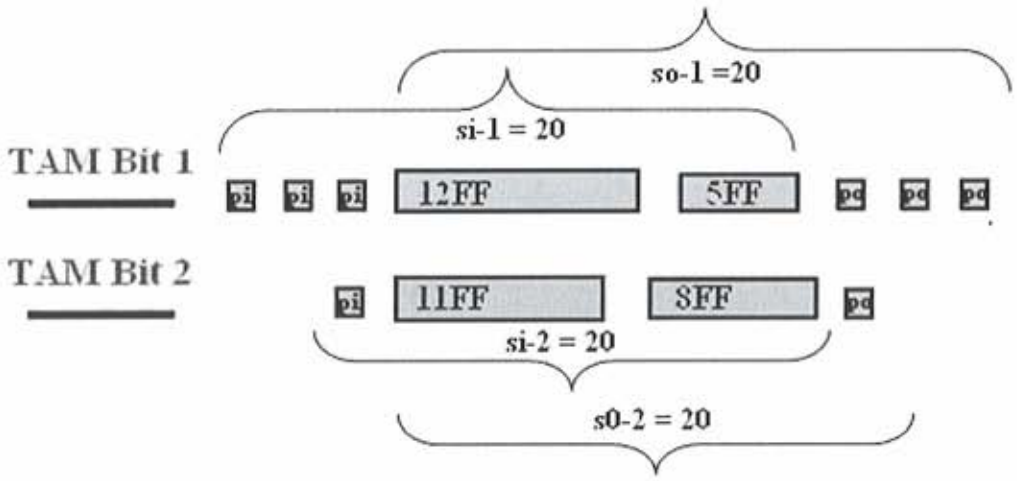

Figure 28: Wrapper Design Example (Stage 6)

si = the maximum length of scan-in-chains = 20 in our case.

s0 = the maximum length of scan-out-chains = 20 in our case.

p = number of test patterns = 100 in our case.

Test time ti = (1 + max[si, so])p + min[si, so]

ti = (1 + 20)x100 + 20 = 2120 clock cycles.

```
Wrapper_Design (TAMwidth)
{
   sort(scan-chain by length)
   distribute scan-chains among TAM bits going from
   the first to the last, then from the last to the
   first utill there is no more scan-chains

   The input pins are then distribured among the TAM
   pins, trying to make all scan-chains on each bit of
   the TAM equal to the longest one

   the remaining pins are distibuted evenly among the
   bits of TAM

   the same is done for the output pins

   si ←longest scan-in chain
   so ← longest scon-out chain

   t ← (1 + max(si,so))*NumberOfTestPatterns + min(si,so)
}
```

Figure 29: Wrapper Design Pseudo Code

5.4 Test Scheduling

After computing the test time for each core, and given the number of TAM bits allocated to it, the power consumption, and the precedence and the concurrency constraints, the same procedure that was applied for the fixed TAM problem is applied here to compute an optimal overall test time.

The procedure of assigning TAM width, wrapper design, and test scheduling is repeated several times, each time recording the best answer so far.

```
Variable_Tam_Test_Scheduling
{
  maxIterations ← maximum number of iterations allowed
  iterations ← 0
  while (iterations < maxIterations)
  {
    randomly assign TAM width to each core
    Wrapper_Design(tamAssigned)
    Annealing_Test_Svheduling
    iterations ← iterations + 1
  }
}
```

Figure 30: Pseudo Code for variable TAM test scheduling

5.5 Results

The algorithm was ran on SOC design in the ITC'02 benchmark suite, shown in Table 4. The results are compared to other results for the same benchmarks published in [15]. It can be shown that from the table our algorithm outperformed the algorithms in all attempted cases. The improvement reached a value of 46%.

The simulated annealing cooling schedule that was used for this SOC consists of the following: $\alpha = 0.99$, $\beta = 100$, $T_0 = 4000$, $M = 5$, which were determined experimentally. Automatic tuning for the parameters was attempted in this instance, but it didn't improve on the results, which resulted in not applying it.

Table 4 : Results for the ITC'02 benchmarks

SOC	Method	TAM width						
		32	48	64	80	96	112	128
D695	SA	26617	23107	17091	15148	14643	13942	14180
	[9]	39489	26203	19773	16149	13649	11285	9885
	[15]	41604	28064	21161	16993	14182	12085	10723
	[15]	41899	28165	21258	17101	14310	12134	10760
	[20]	41949	28372	21423	17210	16403	13023	12327
	[7]	43723	30317	23021	18459	15698	13415	11604
	[16]	42716	28639	21389	17366	15142	13208	11279
	[17]	44307	28576	21518	17617	14608	12462	11033
	[18]	46152	30777	22669	19551	16388	13877	11893
P22810	SA	275885	223784	190028	207669	178443	161840	161840
	[9]	438619	287999	216747	167792	149592	129624	115406
	[15]	438619	289287	218855	175946	147944	126947	109591
	[15]	438619	293019	219923	180004	151886	132812	112515
	[7]	452639	307780	246150	197293	167256	145417	136941
	[16]	446684	300723	223462	184951	167858	145087	128512
	[17]	458068	299718	222471	190995	160221	145417	133405
	[18]	541402	356039	294788	220674	203257	175946	157527
P34392	SA	702360	584874	553904	545604	544579	544579	544579
	[9]	935649	635237	544579	544579	544579	544579	544579
	[15]	944768	628602	544579	544579	544579	544579	544579
	[15]	965252	657561	544579	544579	544579	544579	544579
	[7]	1023820	759427	544579	544579	544579	544579	544579
	[16]	1016640	681745	553713	544579	544579	544579	544579
	[17]	1010821	680411	551778	544579	544579	544579	544579
	[18]	1191289	838643	568133	544579	544579	544579	544579
P93793	SA	1112419	786997	587793	467959	416464	385408	32869
	[9]	1782067	1190565	890092	707664	609580	517017	452245
	[15]	1757452	1169945	878493	718005	594575	509041	447974
	[15]	1765779	1178397	893892	718005	597182	510516	451472
	[20]	1775099	1192980	899807	705164	602613	521806	463707
	[7]	1851135	1248795	975076	794020	627934	568456	511268
	[16]	1791860	1200157	900798	719880	607955	521168	459233
	[17]	1791638	1185434	912233	718005	601450	528925	455738
	[18]	2404321	1598829	1179795	1060369	717602	625506	491495
G1023	SA	25283	17507	14794	14953	14794	14794	14794
	[9]	29765	20032	15302	14794	14794	14794	14794
	[15]	31139	21024	15890	14794	14794	14794	14794
	[15]	31398	21365	16067	14794	14794	14794	14794
	[16]	31444	21409	16489	14794	14794	14794	14794
	[17]	34459	22821	16855	14794	14794	14794	14794
U226	SA	10664	8013	8013	5347	5332	5332	5332
	[9]	13333	10666	8084	8000	5333	5333	5333
	[15]	13333	8084	6746	5332	5332	4080	4080
	[15]	13416	10750	6746	5332	5332	4080	4080
	[16]	13416	10750	6746	5332	5332	4080	4080
	[17]	18663	13331	10665	8084	7999	7999	7999
F2126	SA	335334	335334	335334	335334	335334	335334	335334
	[9]	350030	335334	335334	335334	335334	335334	335334
	[15]	357088	335334	335334	335334	335334	335334	335334
	[15]	357088	335334	335334	335334	335334	335334	335334
	[16]	357109	335334	335334	335334	335334	335334	335334
	[17]	372125	335334	335334	335334	335334	335334	335334
T512505	SA	11901919	6843446	6507480	681728	6669326	66833118	6537797
	[9]	10504020	10453470	5255380	5228420	5228420	5228420	5228420
	[15]	10530995	10453470	5268868	5228420	5228420	5228420	5228420

	[15]	10530995	10453470	5268868	5228420	5228420	5228420	5228420
	[16]	10531003	10453470	5268872	5228420	5228420	5228420	5228420
	[17]	10530995	10453470	5268868	5228420	5228420	5228420	5228420
D281	SA	4709	4163	4163	4163	3926	3926	3926
	[9]	7102	4870	3926	3926	3926	3926	3926
	[11]	7432	5003	3926	3926	3926	3926	3926
	[11]	7946	5485	4070	3926	3926	3926	3926
	[16]	7948	5486	4070	3926	3926	3926	3926
	[17]	8444	6408	5084	3964	3926	3926	3926

Table 5: The run time recorded for some of the benchmarks

	Run Time (milli seconds)						
SOC	**32**	**48**	**64**	**80**	**96**	**112**	**128**
D695				901915	923791	875026	879993
P22810		1874988	1962335	207669	2033837	173705	2097107
P34392		996486	964599	926492	544579	969986	926492
P93793		4562201	4468763	4664482	4689696	4408625	4717429
G1023		691709	646408	704531	655017	719821	695453
U226	5428521	331147	329116	133735	137873	135326	
F2126				210566	106087	209248	208251
T512505	1309381	1295540	489646	1232054	722034	471144	488303

Chapter 6

Conclusion

This Thesis tackled three problems that are related to system-on-chip technology. The first problem that was solved is the test generation problem using the dynamic ant algorithm. The results that were generated were very promising. If combined with a deterministic way for generation, such as ATPG, should give optimum results. The second problem that we solved is SOC test scheduling using simulated annealing. This method was applied to two SOCs that were presented in papers [6] and [11]. For the first SOC, our method resulted in better time, as for the second one we got the same time. The third problem dealt with test scheduling and wrapper design. All reported results achieved the optimum answers.

Some improvement could be considered in the future. For instance, in the test generation problem, heuristics methods, as dynamic ants, are known not to perform well. We recommend that either a hybrid method to be used or to parallelize the current Ants method in order to improve speed up.Another improvement could be applied to the dynamic TAM test scheduling. In fact one may assign the TAM using Taby search, Simulated Annealing, or Genetic Algorithms. This wouls guide the random search process in a hill-climbing fashion.

Bibliography

[1] Marinissen, E.A., & Zorian, Y. (1999). Challenges in testing core-base system ICs. I*EEE Communication Magazine*, *37*(6), 104 – 109.

[2] Bergamashi, R.A., & Cohn, J.(2002, November 10-14). The a to z of SoCs. *Proceedings of the ICCAD 2002,* 791 - 798

[3] Gupta, R., & Zorian, Y. (1997). Introducing core-based system design. IEEE *Design & Test of Computers, 14*(4), 15 - 25.

[4] Marinissen, E.J., Arendsen, R., & Bos, G. (1998, Octobre 18-23). A structured and scalable mechanism for test access to embedded reusable cores. *Proceedings of the International Test Conference*, 284 - 293.

[5] Goel, S.K., & Marinissen, E.J. (2002). Clustered-based test architecture design for system-on-chip. *Proceedings of the 20th IEEE VLSI Test Symposium.*

[6] Pouget, J., Larson, E., Peng, Zebo, Flottes, M., & Rouzeyre, B. (2003).An efficient approach to soc wrapper design, tam configuration and test scheduling. *Proceedings of the Eigth IEEE European Test Workshop.*

[7] Iyengar, V., Chakrabarty, K., & Marinissen, E.J. (2002). On using rectangle packing for SOC wrapper/tam co-optimization. *VLSI Test Symposium* .

[8] Koranne, S. On test scheduling for core-based SOCs.(2002, January 7-11). *Proceedings of ASP-DAC 2002, 7th Asia and South Pacific and the 15th International Conference on VLSI Design,* 505 – 510.

[9] Wuu, J.-Y., Chen, T.-C., & Chang, Y.-W. (2005 January) SoC test scheduling using the B*-Tree based floorplanning technique. *Proceedings of ACM/IEEE Asia South Pacific Design Automation Conference*, Shanghai, China, 1188-1191.

[10] Larsson, E., & Fujiwara, H. (2002).Power constrained preemptive TAM scheduling. *Proceedings of the Seventh IEEE European Test Workshop, IEEE Computer society*.

[11] Zhao, D., & Upadhyaya, S. (2005). Dynamically partitioned test scheduling with adaptive tam configuration for power-constrained SOC testing. *IEEE Transaction on Computer aided design of integrated circuits and systems, 24*(6).

[12] Harmanani, H. M., & Salamy, H. A. (2005, June). Power-constrained system-on-a-chip test scheduling using a genetic algorithm. *Third IEEE Northeast Workshop on Circuits and Systems*, Quebec City, Canada, 203-206.

[13] Harmanani, H. M., & Salamy, H. A. (2006). On power-constrained system-on-chip test scheduling with precedence relationships. *Third IEEE Northeast Workshop on Circuits and Systems,* Quebec City, Canada.

[14] Harmanani, H., & Karablieh, B. (2005). A hybrid distributed test generation method using deterministic and genetic algorithms. Fifth International Workshop on System-on-Chio for Real Time Applications, 317-322.

[15] Zou, Wei, Reddy, Sudhakar M, Pomeranz, Irith, & Huang, Yu. (2003). SOC testing using simulated annealing. *Proceedings of the 21st IEEE VLSI Test Symposium*.

[16] Huang, Y., et al. Optimal core wrapper width selection and soc test scheduling on 3-D bin packing algorithm. *ITC*, 74-82.

[17] Goel, S.K, & Marinissen, E. J. (2002). Effective and efficient test architecture design for SOCs. *ITC*, 529-538.

[18] Koranne, S., & Iyengar, V. (2002). On the use of k-tuples for SoC test schedule representation. *ITC*, 539-548.

[19] Iyengar, V., Chakrabarty, K., & Marinsen, E. J. (2001). Test wrapper and test access mechanism co-optimization for system-on-chip. *ITC* ,1023-1032.

[20] Bushnell, M., & Agrawal, V. (2000). *Essentials of electronic testing for digital, memory and mixed-signal vlsi circuits.* Boston: Kluwer Academic Publishers.

[21] Sehgal, A., Goel, S. K., Marinissen, E. J., & Chakrabarty, K. (2004). IEEE P1500-compliant test wrapper design for hierarchical cores. *ITC International Test Conference,* 1203-1212.

[22] Lee, H.K., & Ha, D.S. (1991). An efficient forward fault fault simulation algorithm based on the parallel pattern single fault propagation. *Proceedings of the ITC*, 946-955.

[23] Pomeranz, I., & Reddy, S. M. (1997). On improving genetic optimization based test generation. *Proceedings of the European Design and Test Conf*, 506-511.

[24] Rudnick, E., & Patel, J. (1997). A genetic algorithm framework for test generation. *IEEE Trans. on CAD, 16, (9).*

[25] Saab, D. G., Saab, Y., & Abraham, J. (1992).CRIS: a test cultivation program for sequential VLSI circuits. *Proceedings of the ICCAD*, 216-219.