Optimization Of Test Scheduling Of SOC Using Genetic Algorithm

Thesis submitted in partial fulfillment of requirement for the Degree of

Master of Technology
In
VLSI Design

Submitted By:
Neelam
Roll No. 601261021

Under the Guidance of:
Ms. Harpreet Vohra
Assistant Professor

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
(Established under the section 3 of UGC Act, 1956)
PATIALA-147004(PUNJAB)

June-2014

Certificate

It is hereby certified that the work embodied in this thesis, entitled "**Optimization Of Test Scheduling Of SOC Using Genetic Algorithm**", was carried out in partial fulfilment of M.Tech (VLSI Design) at the Department of Electronics and Communication Engineering, Thapar University, Patiala, carried out under the supervision of Ms. Harpreet Vohra, Assistant Professor, Thapar University. The matter has not been submitted, in part or full, for any other degree of this or any other University.

Date: 18/7/14

Neelam

Neelam

Roll No. 601261021

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

Ms.Harpreet Vohra

Assistant Professor

ECED, Thapar University

Counter Signed By:

Dr. Sanjay Sharma

(Head of the Department)

ECED, Thapar University

Patiala-147004

Dr. S.K.Mohapatra

Dean Academic Affairs

Thapar University

Patiala-147004

Preface

Integration of a complex system, that until recently consisted of multiple Integrated Circuits, onto a single Integrated Circuits, is known as System On a Chip. The shrinking of silicon technology leads to increase in number of faults which in turn leads to the serious increase in test time. Test time reduction is one of the research challenge in the SOC design. The test-application time when testing a system can be minimized by scheduling the execution of the test sets as concurrently as possible. The basic idea in test scheduling is to minimize the test application time. Many system-on-chip (SOC) integrated circuits today contain hierarchical cores that have multiple levels of design hierarchy involving “child cores”. Hierarchy imposes a number of constraints on the manner in which tests must be applied to parent cores and their child cores.

In this thesis we explore and analyze all the previous work on TAM optimization and test scheduling for testing of SOCs taking into consideration test access time. It has been found that all the previous work done in TAM co-optimization and test scheduling for testing has treated all the cores in the SOC as if at same level of hierarchy i.e. flat cores. We perform optimization of TAM width sharing. We explore the impact of different TAM width on test application time.

.

Neelam

Acknowledgement

This thesis would not have been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this study.

Foremost, I would like to express my sincere gratitude to my Supervisor, *Ms. Harpreet Vohra,* for her patience, motivation and continuous support during the course of this thesis. I also thank *Dr. Sanjay Sharma,* Head of the Department and Dr. Kulbir Singh, PG Coordinator for their guidance and support as well as all the faculty members and staff of the Department of Electronics and Communication Engineering for being very supportive to me. I would also like to thank all my colleagues for motivating me all the time whenever I needed them

Last but not the least, I am indebted to my family for their unending love, support and encouragement.

Neelam

601261021

Table Of Contents

List Of Figures

List Of Tables

CHAPTER 1: INTRODUCTION

1.1 Introduction and Motivation

Modern semiconductor process technologies enable the manufacturing of a complete system on a chip. Such system chips typically are very large IC's consisting of millions of transistors, and containing a variety of hardware module. In order to design these in a time, increasingly reusable cores are being utilized. Cores are predesigned and pre-verified design modules, meant to be reused in multiple SOC design. The cores can be designed in-house or bought from core vendors, and it is the task of the system integrator to integrate them into a system. The IC fabrication process may contain defects such as shorts to power or ground, extra materials etc., may appear as faults and cause failures. Therefore, each manufactured IC needs to be tested. The aim of fabrication test is to ensure that the fabricated IC is free from manufacturing defects. The general approach to test is to apply test stimuli and compare the produced responses against the expected ones. As each fabricated IC is tested, it is important to minimize the test application time. For example, if an IC that has a test application time of 10 seconds and is fabricated in million copies. The total test application time for these IC's will be 1157 days. A saving of 1 second per IC leads to a reduction of the total test time with 115 days. The increasing cost for IC testing is in part due to the huge test data-volume test stimuli and expected responses, which can be in the order of tens of gigabits. The huge test-data volume leads to long test application time and requires larger tester memory. The number of transistors in a single IC is growing faster than the number of I/O pins, i.e., the ratio of transistors per I/O pin is growing. This trend leads to increased test application time.

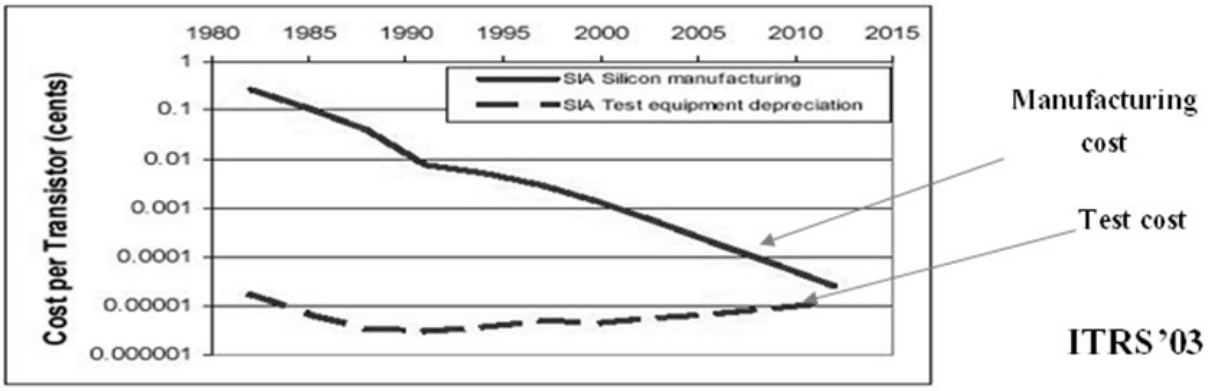


Fig:1.1.1 Trend in test cost versus fabrication cost per transistor[29]

1.2 SOC test process:

As no of transistors increases, on the other hand, the number of input and output pins, so called primary inputs and outputs remain almost same, this led to many test related problems since the amount of test data, which has to be applied to the chip, is becoming very large. Testing is performed by applying test stimuli, consisting of a set of test vectors, to the chip and then comparing the test responses with the expected responses.

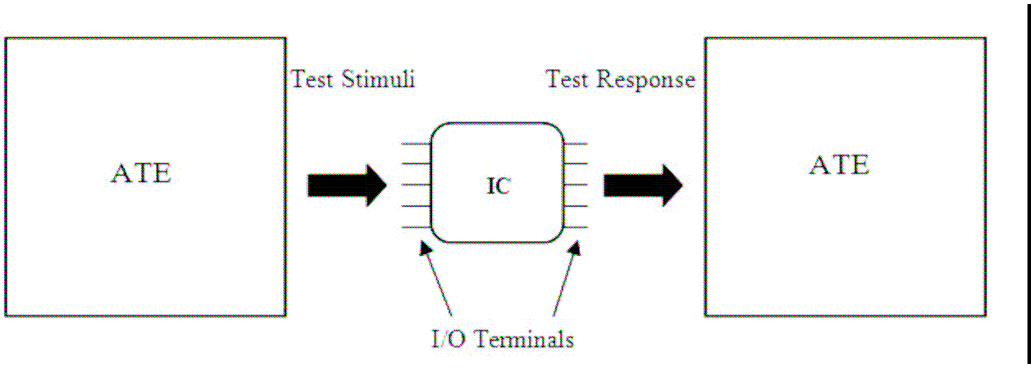


Fig: 1.2.1 SOC Test Process[29]

There are two main approaches for testing embedded cores:

i) The test stimuli are usually applied by using an Automatic Test Equipment (ATE).The ATE is used to compare the test responses with the expected ones. A difference between the two responses indicates that a fault has been detected on the IC.

ii) Built-in self-test (BIST) and applying and observing pre computed test sets at the core terminals. In the case of BIST, an on-chip engine is responsible for generating the test stimuli and for observing the responses. Although BIST obviously simplifies the test task for the system integrator, there are several limitations: Most cores are not suitable for BIST. The BIST is to use pre-computed test sets, provided usually by the core vendors, which are applied and observed via the core's terminals. The core vendor has little or no information about where their cores will be placed on the chip. Therefore they assume that all core terminals are directly accessible. The task of the system integrator is to ensure that the logic surrounding the core allows the tests to be applied, i.e. designing a suitable test access mechanism (TAM) and Test Wrapper[11].

1.3 Core Based SOC Testing:

In recent years, reusable embedded modules have captured the imagination of designers who understand the potential of using such modules in building on-chip systems similar to using integrated circuits on a Printed Circuit Board (PCB). Designers format a rich library of predesigned, pre-verified building blocks, the so-called embedded cores to import technology to a new system and differentiate the corresponding product by investing intellectual property advantages. Most importantly, the use of embedded cores shortened the time to-market for new systems due to design reuse. Cores come in a wide range of hardware description levels, categorized as soft, firm and hard. These three types offer trade-off opportunities.

i)**Soft Cores(Register transfer level)**:

These are given in a hardware description (HDL) language, hence, technology independent, and the soft cores can easily be modified compared to hard cores. The soft core specification has to be synthesized and optimized, and also it is required to perform test generation.

ii)**Hard Core(Technology dependent layout):**

have been optimized for predictable performance, but lack flexibility. Hard cores are given as layout files that cannot be modified. This type of cores are highly optimized for area and performance, and synthesized to a specific technology.

iii)**Firm Cores(Netlist):**

offer a compromise between the two. Firm cores are given as technology-dependent netlists using a library with cells that can be changed according to the core integrator's need[17].

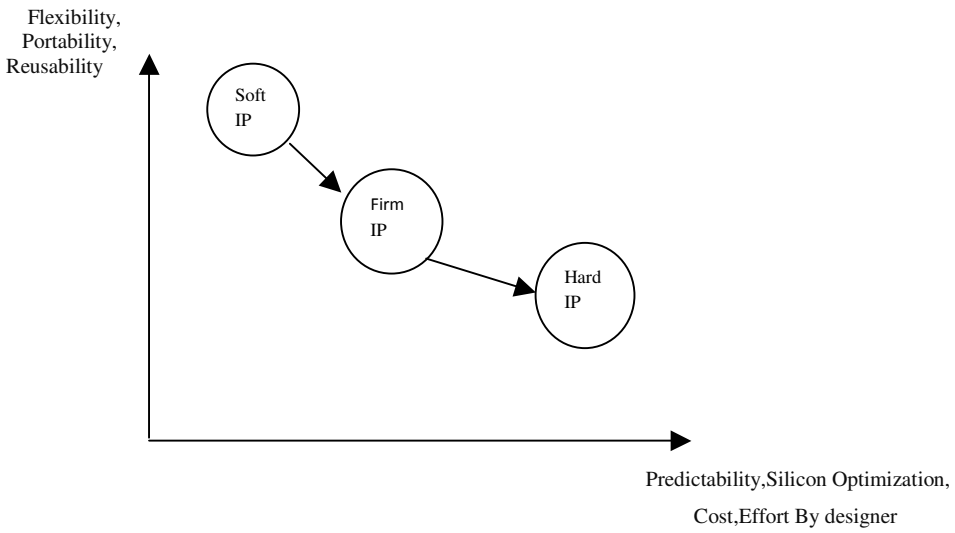


Fig: 1.3.1 comparison for the Hard, Soft and Firm Cores[17].

The architecture consists of three structural elements:

i)Test pattern source and sink
ii)Test access mechanism
iii)Core test wrapper

i)**Test pattern and sink**:

The test pattern source generates the test stimuli for the embedded core, and the test pattern sink compares the response to the expected response.

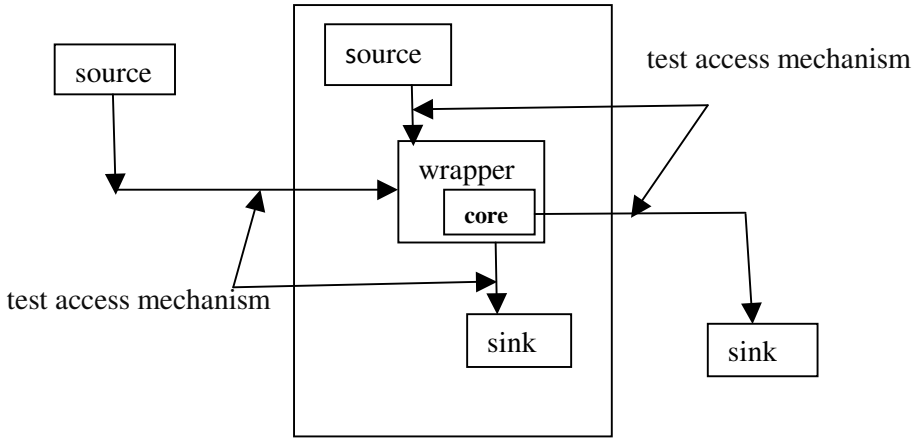


Fig 1.3.2 Overview of three elements in an embedded-core test approach[17]

ii)**Test access mechanism**:

The test access mechanism transports test patterns. It can be used for on-chip transport of test stimuli from a test pattern source to the core-under-test, and for transport of test responses from the core-under-test to a test pattern sink.

iii)**Core test wrapper**:

The core test wrapper forms the interface between the embedded core and its environment. It connects the terminals of the embedded core to the rest of the IC and to the test access mechanism[16].

1.4 Constraints In SOC Test Process:

i)**Interconnect(Cross Core):**

The interconnections between wrapped cores must be tested. The problem with interconnect testing is that the interconnections and logic between wrapped cores do not have their own dedicated wrapper and hence no direct connection to the TAM. As wrappers are the required interface to the TAM, test stimuli and test responses for interconnection tests have to be transported to and from the TAM via wrappers at wrapper cores[20].

ii)**Hierarchy Conflict**:

In a core-based environment, cores are embedded in the system. However, a core can be embedded in a core. One or more child cores can be embedded in a parent core. The embedding of cores in cores results in conflicts at testing. The testing of a parent core can usually not be performed at the same time as the testing of a child core.

iii)**Power Constraint**:

When cores are tested concurrently, then, it may dissipate more power than the maximum power. Placing more and more functions on a silicon die has resulted in higher power consumption. There are two major sources of power consumption in CMOS VLSI circuits: dynamic power dissipation, due to capacitive switching, and static power dissipation, due to leakage and sub-threshold currents[19].

iv) **Precedence Constraint:**

In some cases, the order in which the tests are executed is important. Such an order imposes precedence constraints, which means that some tests must be executed prior to others[20].

v) **Multiple Test Sets:**

A core, for instance, can be tested using one test set generated by an LFSR and another test set stored in the ATE. The selection of the test sets for the cores in the system affects the total test application time and the ATE usage.

vi) **TAM Wire Assignment**:

Each test must be assigned a start time and an end time as well as TAM wires in the case of tests stored in the ATE. However, TAM wire are limited due limited number of input, output and bidirectional pins of SOC and the cores within the SOC have higher number of input, output and bidirectional pins which puts a serious constraint over the TAM bandwidth utilization and amount to test data that can be transferred to the core.

vii) **Hetergenous IP Cores**:

Many SOC designs incorporate cores that use different technologies, such as random logic, memory blocks, and analog circuits. For systems assembled on printed circuit boards each of these components was tested using different types of dedicated ATEs (e.g., digital, memory or analog testers). For SOC testing one can use generic high-performance mixed-signal ATEs, however their high production cost brings limited benefits to complex designs, since cores using heterogeneous technologies still need to be tested sequentially, thus lengthening the testing time and ultimately raising the manufacturing test cost[16].

1.5 Problem Formulation:

In this thesis the increasing testing time for core-based SOCs is targeted by changing the TAM width and for different TAM partitions. The increased usage of embedded pre-designed reusable cores necessitates a core-based test strategy in which cores are tested as separate entities. Test application time is a major issue in System-on-Chip Testing (SOC). As the complexity of system increases, the test application time also significantly increases. The test application time must be minimized to transport test data to and from the cores. In this thesis, a Genetic Algorithm (GA)-based approach to optimize the test time for different SOC Benchmark Circuits. This provides optimal results comparable to other methods of similar problems. Genetic Algorithm is used Different ITC'02 benchmarks are used to show the testing time. These benchmarks circuits consisting of a number of cores where each core is delivered with test data volume. The SOC test planning problem is solved such that the given cost function is minimized. The main contributions of this thesis are as follows:

The analysis of the test application time for different TAM width shows that the test application time is minimum as the TAM partition increasing.

1.6 Thesis Outline:

This thesis is divided into six chapters. Chapter 1 gives an introduction and background for the entire thesis.

Chapter 2 presents some preliminary issues related to the SOC testing. These preliminaries are specific for this work, such as SOC model, test access architecture and scheduling.

Chapter 3 focus on the system on chip architecture: Wrapper, Test Access Mechanism and Test scheduling. And also briefly explain some of the scheduling algorithm.

Chapter 4 presents the basics of genetic algorithm including its introduction, selection, crossover and mutation.

Chapter 5 presents the results for the six ITC'o2 Benchmark circuits and shows the testing time for the different TAM width and different TAM partitions.

Chapter 6, presents conclusion and proposals for the future work

CHAPTER 2: PRELIMINARIES

2.1 System On Chip Model:

With technological advances that allow us to integrate complete multi-processor systems on a single die, Systems-on-Chip (SOCs) are at the core of most embedded computing and consumer devices, such as cell phones, media players and automotive, aerospace or medical electronics. Nowadays, large system-on-a-chip (SOC) designs commonly use IP cores that are deeply embedded in the system chip and direct access is often impossible. Individual cores have to be tested on a system level after manufacturing and therefore special test access mechanisms (TAMs) are required. Choosing and scheduling test solutions for SOC embedded IP cores is a very complex problem. In order to facilitate reuse of test vectors provided by the core vendor, an embedded core must be isolated from the surrounding logic, and test access must be provided from the I/O pins of the SOC. Test wrappers form the interface between TAM and core, while TAMs transport test data between SOC pins and wrappers.

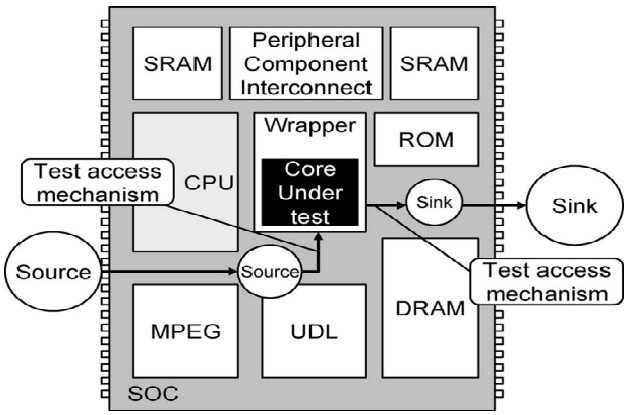


Fig.2.1.1 SOC Model[15]

The major components of test access architecture are:

a) Test source and sink

b) TAMs

c) Test wrapper

The general problem of SOC test integration includes the design of Test Architectures, optimization of core wrappers, test scheduling wrapper pin assignments.

2.2 Wrapper Design:

A core test wrapper named TestShell consists of following components:

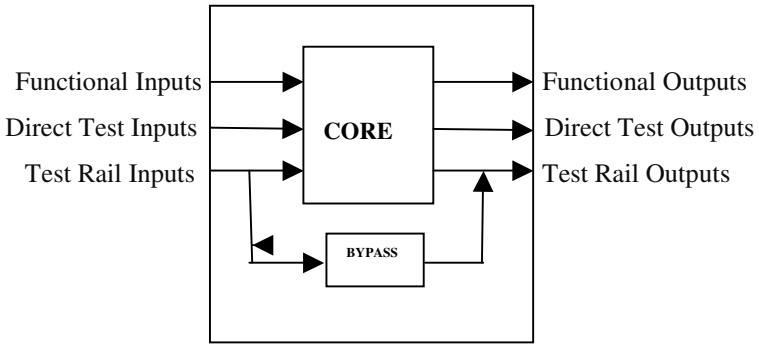


Fig. 2.2.1 Testshell/Wrapper[7]

a) A **Test Cell** for every core terminal. The test cell provides controllability as well as observability.
b) **An Bypass Register** allows a TAM to bypass core and wrapper, in order to test another core that is connected to the same TAM.
c) A **Test Control Block** (TCB) The TCB has a bit-slice nature and consists of a shift and an update register. The TCB is primarily meant to control the operation of the Testshell, through several mandatory bit slices[17].

For SOC designs where modules are not tested prior to mounting leads to an increasing amount of test data volume to be transported in a system. For that reason, core wrappers such as TestShell, TestCollar and P1500 have to be developed.

The TestShell consists of three layers of hierarchy. The TestShell consists of three layers hierarchy:

- The core or the IP module,
- The TestShell, and
- The host.

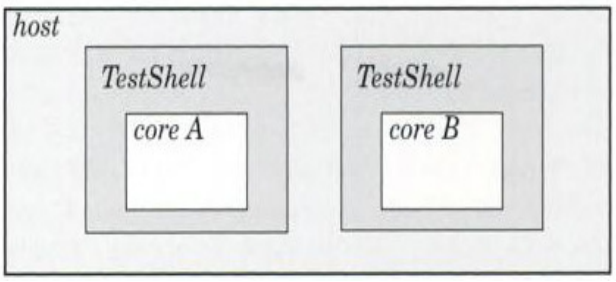


Fig: 2.2.2 Three Hierarchy Layers Core[27]

The core is the object to be tested. The host is the environment where the core is embedded. It can be a complete IC, or a design module which will be an IP module itself. Finally, the TestShell is the interface between the core and the host and it contains three types of input/output terminals:

- Function input/output corresponds one-to-one to the normal inputs and outputs of the core.
- Testrail input and output are the test access mechanism for the TestShell with an optional bypass.
- Direct test input/output are used for signals which cannot be produced through the TestRail

2.3 **Test Access Architecture (TAM):**

There are three main kinds of test access architectures:

a)Multiplexing architecture

b)Distribution architecture

c)Daisy-chain architecture

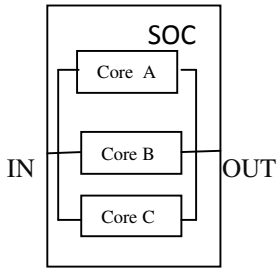
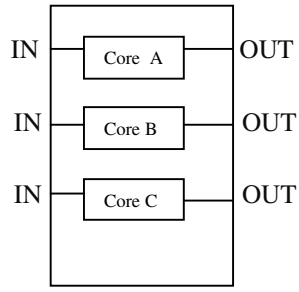


Fig:2.3.1(b) Multiplexing architecture[18]



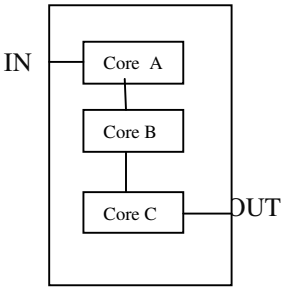


Fig:2.3.1(c)Distribution architecture[18]

Fig:2.3.1(d) Daisy-chain architecture[18]

TAMs were modeled as fixed-width test buses. In a fixed-width test bus architecture, the total TAM width is partitioned among several test buses.

TestRail:

The TestRail[30] architecture is a combination of Daisychain and Distribution architecture. The advantage of this is that it allows concurrent as well as sequential testing. An example of the Test Bus Architecture where the N TAM wires are partitioned into three Test buses each of Width w_1 , w_2 and w_3.

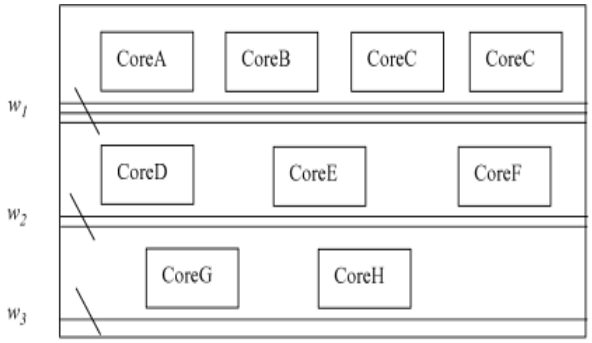


Fig:2.3.2 Test Rail Architecture[30]

Testbus:

It is a combination of the multiplexing and distribution architectures. The Testbus[30] allows architecture allows for multiple Test buses on one SOC that operate independently, as in the Distribution architecture cores connected to the same test bus suffer from the same drawbacks as in the multiplexing i.e. their wrappers cannot be accessed simultaneously, hence making core-external testing. The TAM is responsible for the transportation of test data (test stimuli and test response). The Two common classes of test bus architectures are fixed-width test bus and flexible width test bus architecture.

Fixed-width test bus:

In fixed width TAM, the total tam width is explicitly partitioned among a finite number of TAMs and each core is assigned to a TAM. Therefore,a large number of cores of varying TAM width requirements are often assigned to a small number of TAM. Since TAM width can't be explicitly tailored to each core's requirement, wasteful TAM wires are often assigned to certain cores. In a fixed-width test bus architecture, the total TAM width is partitioned among several test buses.

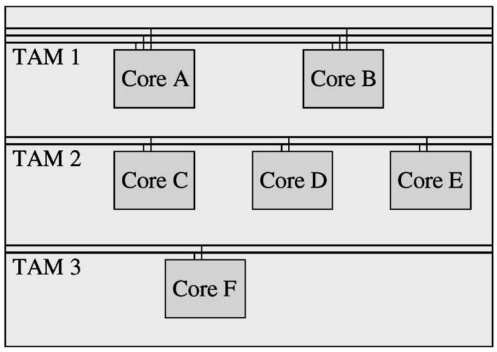


Fig:2.3.3 Fixed Width TAM[3]

There are three main drawbacks of fixed-width architectures that limit the choice of available TAM widths for cores.

i) each core does not receive a TAM width explicitly tailored to its own requirements.

ii) the maximum TAM width available to any core equals the width of the widest test bus in the SOC. Therefore, to increase TAM width per core, the number of TAMs is reduced.

Flexible-width architecture:

The TAM width supplied to each core is based explicitly on the core's TAM width needs, each core can be assigned a unique number of TAM wires corresponding to its testing time v/s TAM width. This leads to the more effective TAM design[3][4].

For making an easy access to cores, cores can be placed in wrapper such as TestShell, Boundary Scan technique. The Boundary scan technique suffers from long testing time due to shifting

process. So it become worse for SOC designs because it increase the amount of test data to be transported.

2.4 Test Scheduling:

After TAM, the main problem is the order of testing for the cores. The general problem of test scheduling for SOCs is related to the NP-Hard. The test scheduling is described for the bus-based TAM-architecture and for Flexible-width architecture. Using bus-based TAM architectures, for transporting test-data usually entails a sequential schedule, and hence, only one core is tested at a time, as shown in Figure 2.4.1. The transportation of tests on the functional bus *bf*1 is shown in Figure 2.4.1 (a). The example shows that the bus is fully occupied all the time. Still, the cores are only activated one after the other (Figure 2.4.1 (b)). This makes the scheduling very simple but the drawback is the long test application time obtained because the cores are not tested in parallel.

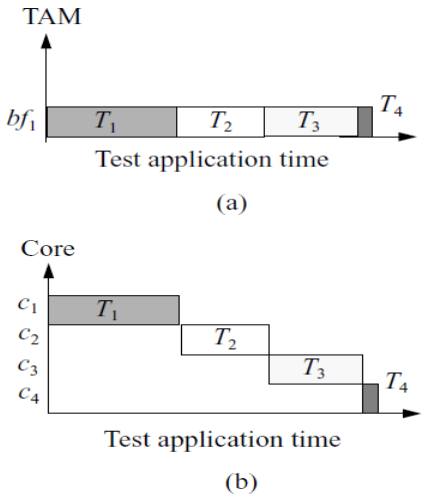


Fig:2.4.1 (a) sequential test scheduling and (b) test application using one functional bus *bf*1[36].

When a bus-based TAM is used, concurrent test scheduling where multiple tests are scheduled in parallel is only possible by adding multiple TAMs. Such a test-architecture is shown in figure 2.4.2 where two Test buses, bt1 and bt2, are used. In the example, c1 and c2 have been assigned to bt1 and c3 and c4 have been assigned to bt2. An example of a concurrent test schedule using

bt1 and bt2 is shown in figure 2.4.2. The transportation of tests on the Test buses bt1 and bt2 is shown in figure 2.4.2 (a). The corresponding test application, where c1 is tested at the same time as c3 and c4, is shown in Figure 2.4.2 (b).

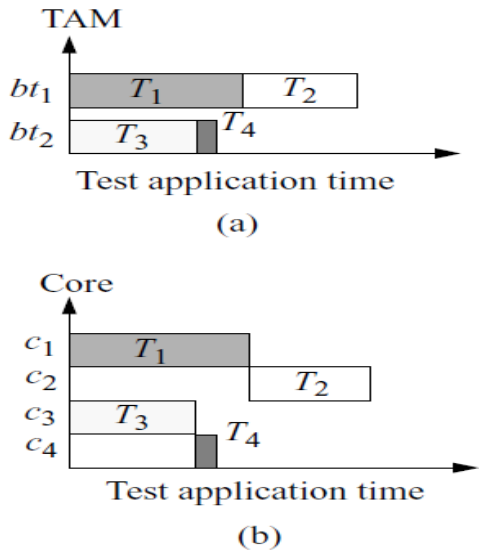


Fig: 2.4.2 (a) concurrent test scheduling and (b) test application using two Test buses *bt*1 and *bt*2[36].

As opposed to the bus-based TAM architectures, the Flexible-width architecture allows concurrent test transportation and application even if only one TAM is used. An example of concurrent test scheduling and application using the Flexible-width architecture is shown in Figure 2.4.3. The transportation of tests on the TAM wires is shown in Figure 2.4.3(a). The corresponding test application, where *c*1 is tested at the same time as *c*2 and *c*3, is shown in Figure 2.4.3(b).

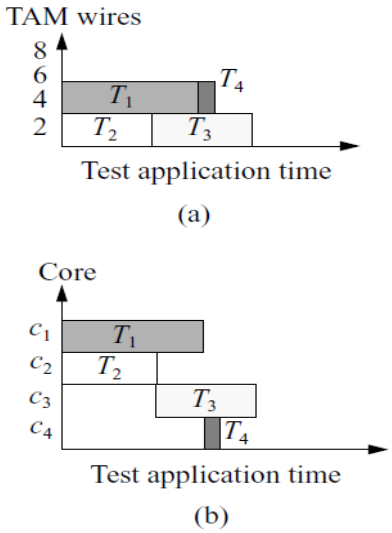


Fig: 2.4.3 (a) concurrent test scheduling and (b) test application using a Flexible-width architecture[36].

Test scheduling techniques can be divided into the following three categories:

a) Non-partitioned testing
b) Partitioned testing with run to complete
c) Partitioned testing

The three test scheduling techniques using the four tests. In the example, it is assumed that the cost function is the test application time, which will be minimized without violating the hardware constraint given by the maximum number of TAM wires. Figure 2.4.3 shows an example of a test schedule using a non-partitioned technique. In non-partitioned test scheduling no new test is allowed to start until all tests in a session are completed. This method produces long test application times due to long periods of time when no core in the system is tested, so-called idle times.

The test schedule can be improved by using a partitioned (sessionless) techniques. In the partitioned technique, tests are allowed to be scheduled as soon as possible, which can decrease the test application time. However, a more advanced test controller is required for the invocation of tests since more possible start times of tests can be used. In order to further optimize the schedule, a pre-emptive test scheduling technique can be used. [18]

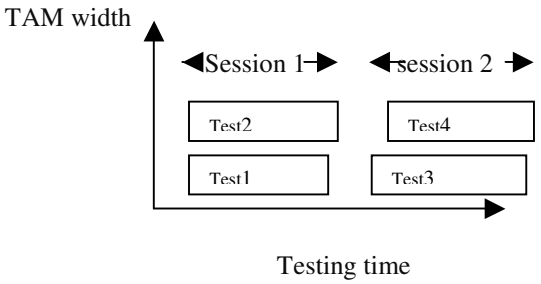


Fig:2.4.3(a) Non-Partitioning Testing[18]

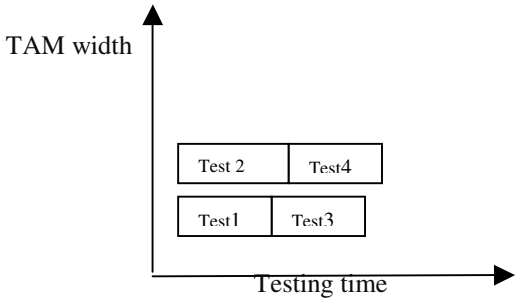


Fig:2.4.3(b) Partitioned testing with run to completion[18]

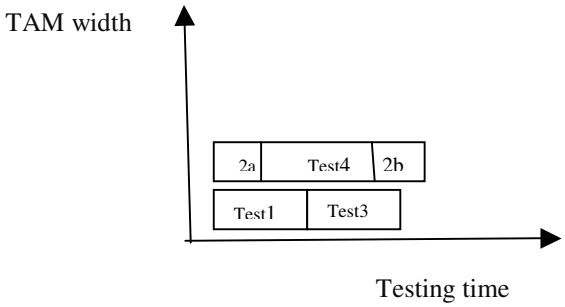


Fig:2.4.3(c) Partitioned testing[18]

CHAPTER 3:LITERATURE REVIEW

3.1 Test Architecture Of SOC :

In order to shorten the time to market, core based SOC design has become a widely used technique of integrated circuit design, where an entire system is built on a single chip using pre-designed, pre-verified complex logic blocks. The system designer or integrators may use the cores which cover a wide range. However, testing such a core-based SOC poses a major challenge for system integrators, as they may have limited knowledge of the cores due to intellectual property (IP) protection. On the other hand, various testing methods such as built-in self test (BIST)[39], scan, functional are provided by different core vendors. As a result, it increases system complexity and test cost in terms of test application time. One major objective of SOC testing is to reduce the test application time. The SOC testing problem usually consists of three parts[3][11][17]:

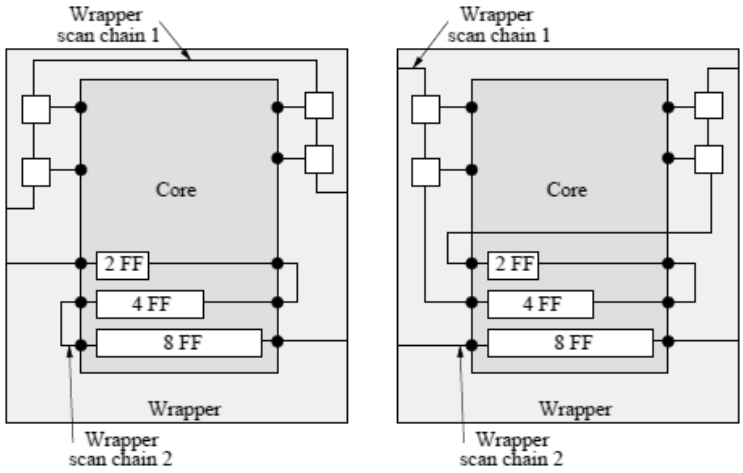
(1) the core wrapper design,

(2) test access architecture (TAM) design and

(3) test scheduling

3.2 SOC Core Wrapper Design:

The SOC core wrapper is an interface between TAM and the core. Since large cores usually have more pins than the number of TAM wires to perform core testing, the core wrapper may often need to perform width adoption while the TAM width is not equal to the number of core pins. Also, while different wrapper designs[32] greatly affect the test time of the core, optimizing wrapper design can improve the TAM efficiently and reduce test time.



(a) (b)

Fig:3.2.1 Wrapper chains: (a) unbalanced, (b) balanced[31]

Recent research work on wrapper design has stressed the need for Balanced wrapper scan chains [32]. Balanced wrapper scan chains are those that are as equal in length to each other as possible. Balanced wrapper scan chains are important because the number of clock cycles to scan in (out) a test pattern to (from) a core is a function of the length of the longest wrapper scan-in (scan-out) chain. Let S_i(S_o) be the length of the longest wrapper scan-in (scan-out) chain for a core. The time required to apply the entire test set to the core is then given by T[32][31]:

$T = \{1+\max(S_i , S_o)\}.P +\min(S_i, S_o)$

where T is the time,

S_i is the maximal length of the wrapper's input scan chain,

S_o is the maximal length of the wrapper's output scan chain, and

P is the number of test patters.

Obviously, the test time T decreases as both S_i and S_o are reduced, i.e., as the wrapper scan-in (and scan-out) chains become more equal in length[31].

The TestShell[32] is proposed to reduce the test access and test isolation problems for SOC. The TestShell consists of three layers of hierarchy. The TestShell consists of three layers hierarchy:

the TestShell is the interface between the core and the host and it contains three types of input/output terminals:

- Function input/output corresponds one-to-one to the normal inputs and outputs of the core.
- Testrail input and output are the test access mechanism for the TestShell with an optional bypass.
- Direct test input/output are used for signals which cannot be produced through the TestRail

Another wrapper approach, called **Testcollar**[32], this is similar to the TestShell: however, the approach does not have the bypass feature which reduces the flexibility by allowing only one core to be served at a time.

P1500: The P1500 consists of a Core Test Wrapper and a Core Test Language.The wrapper uses wrapper boundry cells with functionality similar to the Test cell in TestShell. Instructions are loaded to the Wrapper Instruction Register (WIR).

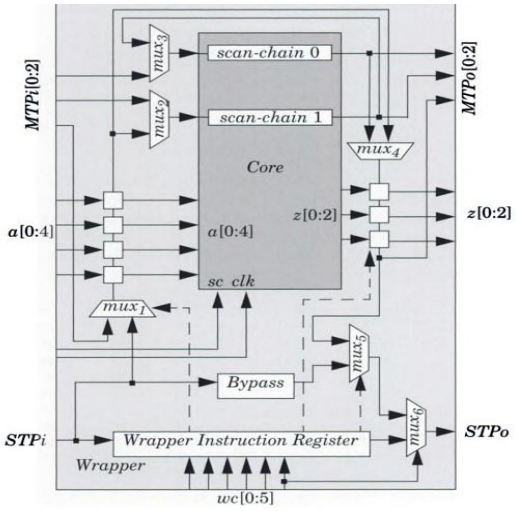


Fig: 3.2.2 The P1500 Approach[29]

3.3 TAM :

The test Access Mechanism[4] takes care of chip test pattern transport. It can be used to transport test stimuli from the test pattern source to the core under test and to transport test responses from the core under test[7][9].

The following are the problems structured in this thesis:

- Reducing the testing time,
- Partition of a TAM, such that the core testing time is minimized.

3.4 SOC Test Scheduling:

Given a SOC with N cores, and W_{max} as the maximal TAM width, each core C_i ($1 \leq i \leq N$) having a set of U_i wrapper configurations. Each wrapper configuration is represented by a pair (W_{ij}, $T(W_{ij})$), where W_{ij} stands for the wrapper width of the j-th wrapper configuration for core C_i and $T(W_{ij})$ stands for the test time of core C_i with the wrapper width W_{ij}. The objective of SOC test scheduling is to determine wrappers for every core and set the start time and finish time for every core so as to minimize the overall test time. This problem can be transformed into a floor planning problem, where the height of the floorplan represents the fixed number of TAM wires W_{max} and the width of the floorplan represents the SOC test time[4][9]

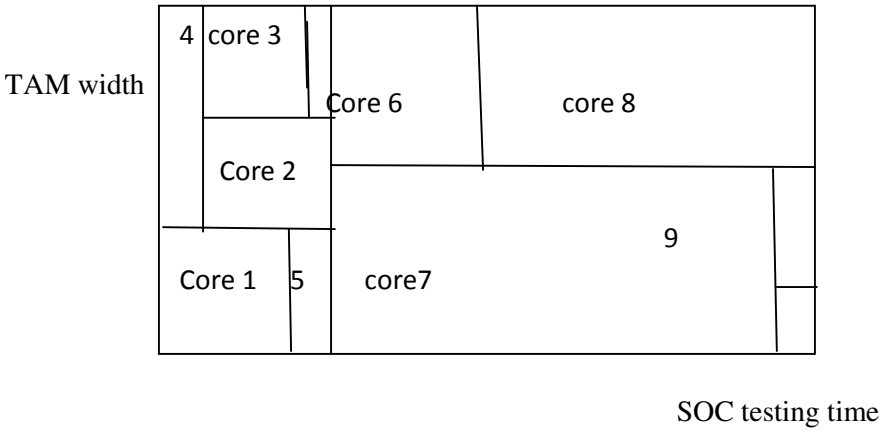


Fig:3.3.1 an example of SOC test Scheduling[2]

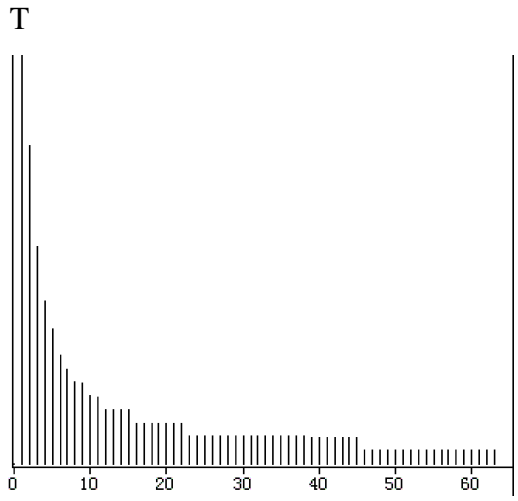


Fig:3.3.2relationship between test time *T* and TAM width[2]

Each core test is represented by a rectangle with height equal to the TAM width W_{ij} and width equal to the test time $T(W_{ij})$. The objective is to choose a wrapper rectangle for every core Ci and place all the rectangles in the floorplan, such that the width of the floorplan is minimized and the height is no larger than W_{max}. It should be noted that the rectangles in the floorplan cannot overlap, because at any time during testing any TAM wires can only be assigned to one core[14].

There are various scheduling algorithms:

FCFS, SJF, priority RR, all these are non-pre-emptive and testing time by using all these scheduling algorithm is more as compare to other and utilization of CPU is not optimum.

A new approach is proposed, improved RR. This combines round robin with shortest job first schedule, hence utilization and testing time both improved[14].

a) <u>Earliest Deadline First:</u>

Earliest-Deadline-First(EDF)[13] is also called Deadline Driven-Scheduling. In EDF, the periodic task which has the earliest deadline will be assigned the highest priority. The task which has the highest priority, namely the task which has the earliest deadline is scheduled firstly. Different from RMS, tasks' priorities are changed over time in EDF. At each schedulable moment, tasks' priorities will be changed dynamically according to their earliest deadlines[13][14].

b) Open Shop Scheduling:

In the open shop scheduling problem, given a shop consisting of m processors, a set of jobs j each job consisting of m tasks .a schedule for m –shop is a set of m processor schedules, one for each processor in the shop. The m processor schedule must be such that no job is processed simultaneously on more than one processor. The objective in open shop scheduling is to minimize the testing time. However, the order in which the processing steps happen can vary freely. The goal is to assign a time for each job to be processed by each processor, so that no two jobs are assigned to the same processor at the same time, no job is assigned to two processors at the same time, and every job is assigned to each processor for the desired amount of time. The usual measure of quality of a solution is its makespan, the amount of time from the start of the schedule (the first assignment of a job to a processor) to its end (the finishing time of the last job at the processor). The m processor schedule must be such that no job is processed simultaneously on more than one processor. The objective in open shop scheduling is to minimize the testing time[1][29].

c) Integer Linear Programming:

In linear programming problems, variables may take non integral values, whereas the quantities to be modeled are often intrinsically integer. In these cases there is no way out: integrality has to be included in the model. This leads to integer (linear) programming problems, that is, problems that ask to optimize a linear function cx where x is a vector with integral coefficients that satisfies a system $a_1x \leq b_1 \ldots, a_mx \leq b_m$ of linear inequalities. Integer programming is so general that so far no coefficient method has been developed that solves all its instances[29][35][37].

d) Mixed Integer Linear Programming:

given a set of tasks (test sets for the cores), a set of test resources (e.g. test buses, BIST hardware) and a test access architecture, determine start times for the tasks such that the total test application time is minimized. Two test sets are conflicting if they cannot be applied to the system at the same time. Test sets can be conflicting if 1) they share an external test bus, or 2) they are BIST test sets for cores that share a BIST resource[1][39].

e) Simulated Annealing:

Simulated Annealing(SA)[38] algorithm is used to solve the 2-D bin packing problem that occurs when optimizing core wrapper design /TAM [2] for test scheduling to achieve optimal SOC test application time. During the process of 2-D packing, the best configuration for every core is selected by SA algorithm to minimize the test application time. To represent test schedules, a data structure is used called a sequence pair that is used for floor planning. The algorithm begins with an initial solution, then a neighboring solution is created by a perturbing the current solution. If the cost of the neighboring solution is less than that of the current solution, the neighboring solution is accepted; else it is accepted or rejected with some probability. The probability of accepting an inferior solution is a function of a parameter called the temperature. The probability function is defined as follows[7]:

$$p=e^{(-\Delta c/T)}$$

where

Δc is the change in cost between the neighboring solution and current solution.

T is the current temperature.

At the beginning of the algorithm, the temperature T is large and an inferior solution has a high probability of being accepted. During this period, the algorithm acts as a random search to find a promising region in the solution space. The algorithm finds the local optimum of the current region[7][38].

f) Genetic Algorithm:

GA are perhaps the most widely known types of evolutionary computation method. GA of evolutionary computation is inspired by Darwin's theory of evolution[21]. A GA generates a population of possible solutions encoded as chromosomes[24], evaluates their fitness and creates a new population by applying genetic operators which are crossover and mutation. Gas uses fitness values of individual chromosomes to carry out reproduction. As reproduction takes place, the crossover[10] operator changes the gene values in some randomly chosen location of the chromosome. After a number of successive reproductions, the less fit chromosomes become extinct, while those best fit gradually come to dominate the

population. Solving a problem using GAs involves defining constraints and optimum criteria, encoding the problem solutions as chromosomes, defining a fitness function to evaluate a chromosome's performance and creating appropriate crossover and mutation operators[10][21][25].

g) <u>RAIN(RAndom INsertion):</u>

When a core is represented by a rectangle, the height of the rectangle is the TAM width assigned to the core, and the width of the rectangle is the test application time for that TAM width. To schedule the cores represented by rectangles, this algorithm used sequence pair representation. In SOC test scheduling, [7] also used sequence pair representation. The sequence pair representation uses two permutations ((Γ+ , Γ-) to describe the placement of rectangles. Since the insertion into a random position of the sequence pair is major operation of this algorithm, hence its name is algorithm RAIN (RAndom INsertion). To derive the placement of rectangles from a sequence pair, one can construct a 45-degree oblique grid as shown in [7][33].

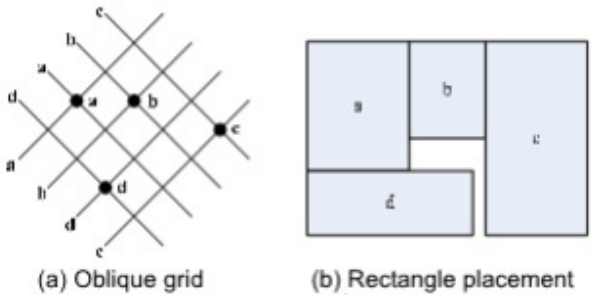


Fig:3.3.3 Sequence pair (Γ+,Γ-) = (abdc, dabc)[33]

h) <u>ACO (Ant Colony Optimization):</u>

ACO is a heuristic based technique which is motivated by intelligent behaviors of ants.. ACO system consists of several agents. Ant behavior represents the behavior of an agent in the colony. Ants deposit pheromone (a chemical matter) on the paths they traversed which helps them perform indirection communication. The routes marked in this way help them find shortest path even if multiple paths are present in the search environment. To solve scheduling problem, there are N processes and all processes have static burst time. And goal

is to find out the optimum processes sequence in which the processes can be scheduled for the CPU allocation in a method that total turnaround time, response time and waiting time should be minimum[34].

CHAPTER4: TEST SCHEDULING USING GENETIC ALGORITHM

4.1 Test Scheduling Using Genetic Algorithm:

Genetic algorithms (GA) [23] are stochastic optimization search algorithms based on the mechanics of natural selection and natural genetics. The genetic formulation of our problem involves the careful and efficient choice of the following.

a. A proper encoding of the solutions to form chromosomes.

b. To decide upon a crossover operator.

c. To identify a proper mutation operator.

Cost function for measuring the fitness of the chromosome in a population.

4.1.1 Solution representation:

SoC test scheduling can be divided into two problems: (1) Optimal assignment of cores to buses, and (2) Optimal distribution of test width. (1): Minimizing the system testing time for a given SoC with NC cores and NB test buses with width w_1 , w_2,..., w_{NB}, respectively, by the optimally assigning the cores to the test buses.

Problem 2: Minimize the system testing time for a given SOC with N_C cores and N_B test buses with total width W, by optimally allocating the total width among the N_B buses and assigning cores to test buses.

4.1.2 Selection:

Its purpose is to choose the fitter individuals in the population that will create offsprings for next generation,

---

Algorithm: Roulette Wheel Selection()

1. Set l=1, j=1,i= ng
2. While l<= pool

   Begin

a) While j<-N

Begin

Compute FRWij

End

b) set j=1, S=0

c) while j<=N

begin

compute S=S + FRWij

end

d) generate random number r from interval(0,S)

e) set j=1,S=0

f) while j<=N

begin

calculate cj =cj-1 + FRWij

if r<=cj , select the individual j

end

g) l=l+1

end

Where

ng= current no of generation

N=total population size

FRWij=Fitness of jth individual in ith generation

pool= number of chromosomes in mating pool

ci is variable storing cumulative fitness and

r is random number generated between given interval

4.1.3 Chromosome Representation:

Population of individuals is maintained within search space for a GA, each representing a possible solution to a given problem. Each individual is coded as a finite length vector of

components, or variables. To continue the genetic analogy these individuals are likened to chromosomes and the variables are analogous to genes. Thus a chromosome (solution) is composed of several genes (variables).

W_1	W_2	….	W_{NB}	N_1	N_2	…..	N_B

<…….Assignment Part……> <……..…..Partition Part……>

Fig:4.1.3.1 Chromosome Structure[5]

First part is the assignment part and second part is partition. In this, first of all TAM is Partition into N_B parts with widths $W_1, W_2 \ldots W_{NB}$. And then these widths will be assigned to cores in such a way to reduced testing time.

4.1.4 Fitness measure:

Fitness of the chromosome is measured in terms of the cost of a solution, which is the total time required to test all cores in the system. If SOC design consist of N_C cores, then the core test width is defined as follows:

$\Phi_i = \max\{n_i, m_i\}$.

A combinational core has no scan inputs and outputs. The required test cycles t_i for testing core i is defined as follows:

$t_i = p_i$ combinational cores

$= (p_i + 1)\,[\, f_i / N_i \,] + p_i$ sequential cores

If core i is assigned to bus j, then the testing time T_{ij} for core i can be calculated as follows:

$T_{ij} = t_i$ if $\Phi i \leq w_i$

$= (\Phi_i - w_i + 1)t_i$ if $\Phi i \geq w_i$

$x_{ij} = 1$ if the core i is assigned to the bus j

$=0$ otherwise

N_c

Fitness(x) =$\max_j(\sum_{i=1} T_{ij}.x_{ij})$

where $1 \leq i \leq N_c$

$1 \leq j \leq N_B$

p_i is the number of test patterns;

n_i inputs including data inputs and scan inputs;

m_i outputs including data outputs and scan outputs;

f_i is the number of flip-flops;

N_i is the internal scan chains;

4.1.5 Crossover:

Crossover is a genetic operator that combines (mates) two chromosomes (parents) to produce a new chromosome (offspring). The idea behind crossover is that the new chromosome may be better than both of the parents if it takes the best characteristics from each of the parents.

Algorithm for single point crossover:

```
for i = 1:N
        CP = round(rand(1)*(N-1)+1); %point of crossover
        for j=1:G
                if j<CP
                    idx = (i-1)*2 + 1;
                else
                    idx = (i-1)*2 + 2;
                end
         Pop2(i,j)= Pop(W(i),j);
        end
end
```

Where

N is the population size

CP is the point of crossover

Pop is randomly generated population

4.2 Introduction to Genetic Algorithm:

Genetic Algorithm is an adaptive heuristic search algorithm based on the evolutionary ideas of natural selection. The basic techniques of GA are designed to simulate process in natural systems necessary for evolution, especially those following the principles first laid down by Charles Darwin of "survival of the fittest". GA is based on an analogy with the genetic structure and behavior of chromosomes within a population of individuals. The GA maintains a population of chromosomes associated with fitness values. Parents are selected to mate on the basis of their fitness, producing offspring via a reproductive plan.

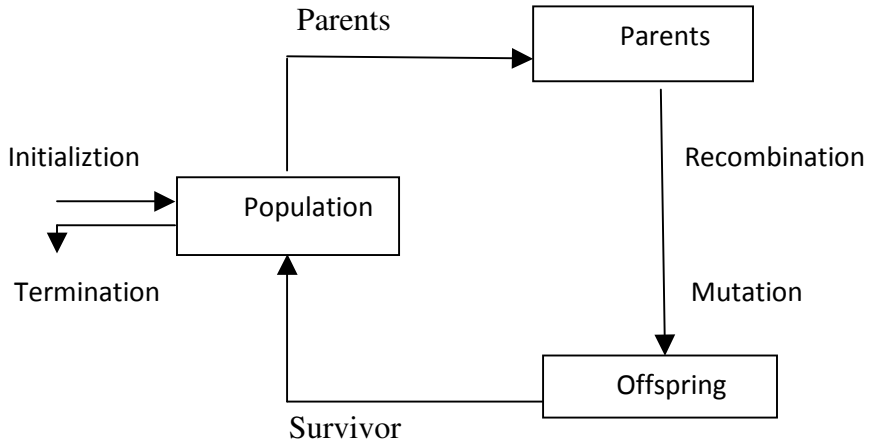


Fig. 4.2.1 General Scheme of Evolutionary Process[23]

It was in the 1950/60s that several independent researchers were studying the idea that evolution could be used as an optimization tool for engineering problems. The idea behind it all was to

evolve solutions to problems by using natural means based on "**survival of the fittest".** Evolutionary strategies were introduced in the mid 60s as a method he used to optimize real-valued parameters for hardware devices. Genetic algorithms were invented by Holland in the 60's and developed later in the 70's. This method was defined as a way to move from one population of chromosomes to another by utilizing natural selection and the operator of crossover, mutation, and inversion. In recent years, there was been widespread interaction among researchers from varied evolution-based studies and as a result, we find some breakdown in the boundaries that define and separate the fields of genetic algorithms, evolution strategies, and evolutionary programming. Often, the term - genetic algorithm - is used to describe something very different from what was originally defined. The three most important aspects of using genetic algorithms are: (1) definition of the objective function, (2) definition and implementation of the genetic representation, and (3) definition and implementation of the genetic operators. Once these three have been defined, the generic genetic algorithm should work fairly well.

4.3 A Simple Genetic Algorithm:

Given a clearly defined problem to be solved and a bit string representation for chromosomes solutions. A typical genetic algorithm requires[23]:

2. a genetic representation of the solution domain,
3. a fitness function to evaluate the solution domain.

4.3.1 Outline of the basic genetic algorithm:

1) Start with a randomly generated population of n-bit chromosomes
2) Calculate the fitness f(x) of each chromosome x in the population
3) Repeat the following steps until n offspring have been created:
    a) Select a pair of parent chromosomes from the current population, the probability of selection being an increasing function of fitness.
    b) With probability pc (the "crossover probability" or "crossover rate"), cross over the pair at a randomly chosen point (chosen with uniform probability) to form two

offspring. If no crossover takes place, form two offspring that are exact copies of their respective parents.

c) Mutate the two offspring at each locus with probability pm (the mutation probability and place the resulting chromosomes in the new population.

4) Replace the current population with the new population.
5) Go to step 2

A genetic algorithm is a fairly general formulation, accommodating many different forms of selection, crossover and mutation are performed.

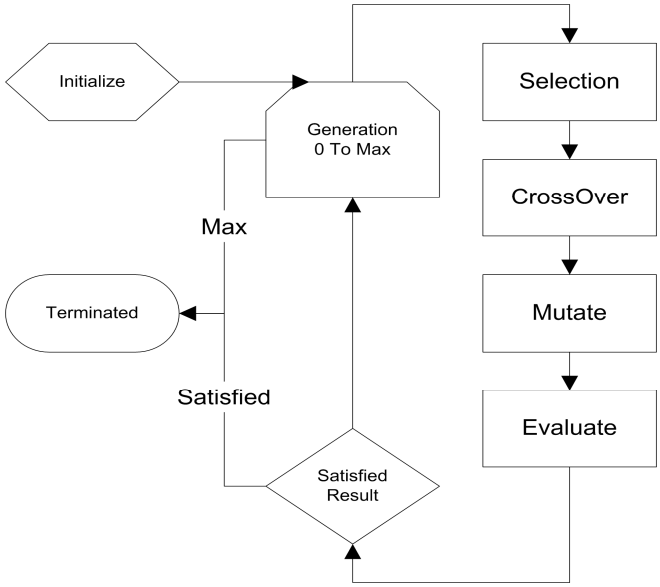


Fig: 4.3.1 A basic flowchart of a genetic algorithm[23].

4.3.2 Psuedo Code

Choose an initial population of chromosomes;
while termination condition not satisfied **do**
repeat
if crossover condition satisfied **then**
{select parent chromosomes;
choose crossover parameters;

perform crossover};

if mutation condition satisfied **then**

{choose mutation points;

perform mutation};

evaluate fitness of offspring

until sufficient offspring created;

select new population;

endwhile

4.4 Operators Of Genetic Algorithm:

Genetic operators used in genetic algorithms maintain genetic variation. Genetic variation is a necessity for the process of evolution. Genetic operators are analogues to those which occur in the natural world:

-Reproduction (or Selection);

-Crossover (or Recombination); and

-Mutation

4.4.1 Selection:

Selection is the first genetic operation in the reproductive phase of genetic algorithm. Its purpose is to choose the fitter individuals in the population that will create offsprings for next generation, commonly known as mating pool. The mating pool thus selected takes part in further genetic operations, advancing the population to the next generation and hopefully close to the optimal solution. Selection of individuals in the population is fitness dependent and is done using different algorithms. Selection chooses more fit individuals in analogy to Darwin's theory of evolution – survival of fittest. There are several schemes for the selection process: roulette wheel selection , scaling techniques, tournament, elitist models, and ranking methods.

4.4.1.1 Roulette Wheel Selection

The basic part of the selection process is to stochastically select from one generation to create the basis of the next generation. The requirement is that the fittest individuals have a greater

chance of survival than weaker ones. This replicates nature in that fitter individuals will tend to have a better probability of survival and will go forward to form the mating pool for the next generation. Weaker individuals are not without a chance. In nature such individuals may have genetic coding that may prove useful to future generations[24].

A common selection approach assigns a probability of selection Pj to each individual j based on its fitness value. A series of N random numbers is generated and compared against the cumulative probability Ci= □Pi of the population. The appropriate individual i is selected and copied into the new population if Ci-1 < U(0, 1) ≤ Ci. Various methods exist to assign probabilities to individuals: roulette wheel, linear ranking and geometric ranking. Roulette wheel is the first selection method. The probability Pi for each individual is defined by:

P [Individual i is chosen] = Fi/ □Fj

j=1 to pop size

where Fi equals the fitness of individual i. The use of roulette wheel selection limits the genetic algorithm to maximization. Every sector represents the probability of selecting the chromosomes for mating[24].

4.4.2 Crossover:

Crossover is a genetic operator that combines (mates) two chromosomes (parents) to produce a new chromosome (offspring). The idea behind crossover is that the new chromosome may be better than both of the parents if it takes the best characteristics from each of the parents. Crossover occurs during evolution according to a user-definable crossover probability. In crossover, portions of two parents from the current generation are combined to create two offspring: a random subpart of the one's bit string is swapped with a random subpart of the other's bit string[25].

a)single-point crossover:

When performing crossover, both parental chromosomes are split at a randomly determined crossover point. Subsequently, a new child genotype is created by appending the first part of the first parent with the second part of the second parent. A single crossover point on both parents' organism strings is selected. All data beyond that point in either organism string is swapped between the two parent organisms[25].

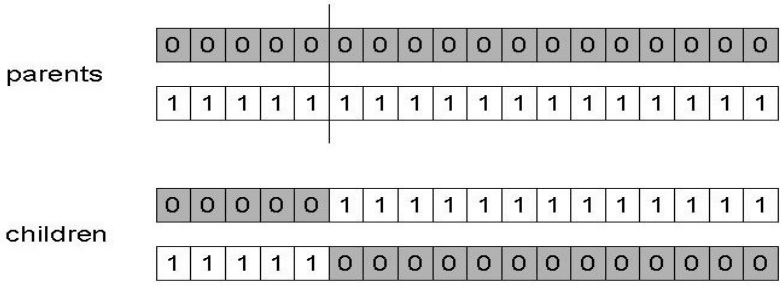


Fig:4.4.2.1 Single Point Crossover[25]

b) **n- Point Crossover:**

Apart from SPC, many different crossover algorithms have been devised, often involving more than one cut point. It should be noted that adding further crossover points reduces the performance of the GA. The problem with adding additional crossover points is that building blocks are more likely to be disrupted. However, an advantage of having more crossover points is that the problem space may be searched more thoroughly. In n-point crossover (NPC), n crossover points are chosen and the contents between these points are exchanged between two mated parents[25].

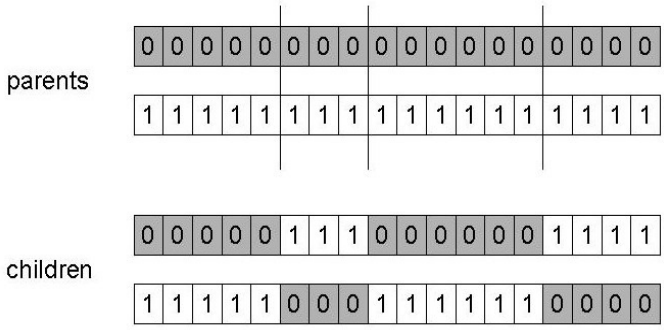


Fig:4.4.2.2 n-point Crossover[25]

c) **Uniform Crossover:**

In arithmetic crossover (AC), arithmetic creates children that are the weighted arithmetic mean of two parents. Children are feasible with respect to linear constraints and bounds[27].

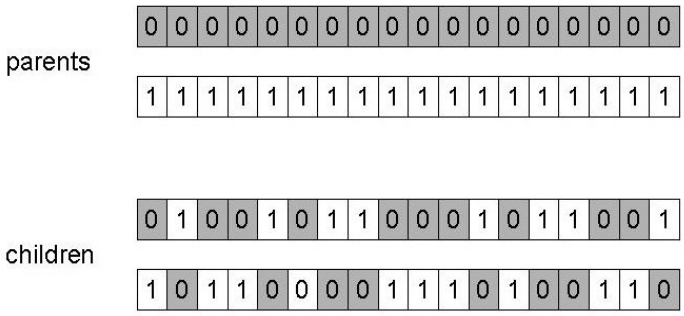


Fig:4.4.2.3 Uniform Crossover[25]

4.4.3 Mutation:

Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next. It is analogous to biological mutation. Mutation alters one or more gene values in a chromosome from its initial state. In mutation, the solution may change entirely from the previous solution. Hence GA can come to better solution by using mutation. Mutation occurs during evolution according to a user-definable mutation probability. This probability should be set low. If it is set too high, the search will turn into a primitive random search. Mutation is a genetic operator that alters one or more gene values in a chromosome from its initial state. This can result in entirely new gene values being added to the gene pool. With these new gene values, the genetic algorithm may be able to arrive at better solution than was previously not possible.

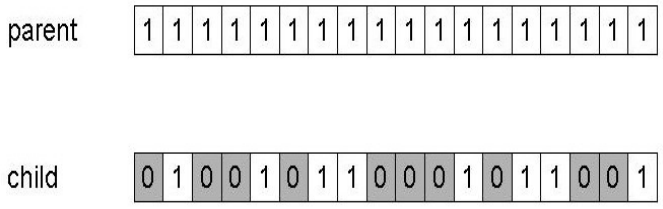


Fig: 4.4.3.1 Mutation[27]

CHAPTER 5:RESULTS

In this section, experimental results on test scheduling for the three largest SOCs (in terms of the number of cores) from the ITC '02 SOC Test Benchmarks are presented [23]. The proposed work is implemented in Matlab. Following table presents the testing time for various values of TAM widths w.

5.1 SOC a586710

The a586710 SOC benchmark consists of 7 modules, out of which 5 are combinational and 2 are sequential circuits. The Table-1 gives the number of test cycles and Table-2 gives the results with two and three partitions.

Table 1: Results for Soc a586710 for tesing time:

	Φ_i =max(i/p, o/p)	p_i	N_i	f_i	f_i/N_i	t_i
SMod1	437	2945	8	17225	2153	6345683
CMod2	275	2679692	-	-	-	2679692
CMod3	407	6029308	-	-	-	6029308
CMod4	324	181140	-	-	-	181140
SMod5	343	2945	8	20430	2553	7524083
CMod6	35	40431	-	-	-	40431
CMod7	226	1914433	-	-	-	1914433

Table 2: Results for Soc a586710 with N_b=2:

w	TAM Partition	Assignment vector	Testing time
24	11,13	1,1,1,2,1,2,1	43728931
36	15,21	2,2,1,2,1,2,1	47685935
48	1,47	1,1,1,1,2,1,2	49879824

Table 3: Results for Soc a586710 with N_b=3:

w	TAM Partition	Assignment vector	Testing time
24	6,810	2,2,3,3,3,1,1	42170725
36	8,12,16	3,3,2,1,1,2,1	**40155351**
48	12,16,20	1,1,2,2,3,3,1	45276984

5.2 SOC d281:

The d281 SOC benchmark consists of 8 modules, out of which 3 are combinational and 5 are sequential circuits. The Table-4 give the number of test cycles and Table-5 and Table-6 gives the results with two and three partitions.

Table 4: Result for the SOC d281 for testing time :

	Φ_i =max(i/p, o/p)	p_i	N_i	f_i	f_i/N_i	t_i
CMod1	60	26	-	-	-	26
CMod2	233	158	-	-	-	158
CMod3	207	96	-	-	-	96
SMod4	52	90	4	29	7	727
SMod5	228	118	6	179	29	3569
SMod6	32	80	2	18	9	809
SMod7	790	2048	20	638	31	65567
SMod8	32	58	58	2	29	589

Table 5:Results for SOC d281 with N_b=2:

w	TAM Partition	Assignment vector	Testing time
24	11,13	2,1,1,1,1,1,1,2	52022780
36	4,32	2,2,1,2,2,2,2,1	51517192
48	1,47	1,1,1,1,1,2,1,2	52707712

Table 6:Results for SOC d281 with N_b=3:

w	TAM Partition	Assignment vector	Testing time
24	6,8,10	2,2,3,3,3,1,1,2	**51087210**
36	8,12,16	3,3,2,1,1,2,1,3	51849513
48	12,16,20	1,1,2,2,3,3,1,2	51113043

5.3 Result For SOC d695:

The d695 SOC benchmark consists of 10 modules, out of which 2 are combinational and 8 are sequential circuits. The Table-7 give the number of test cycles and Table-8 and Table-9 gives the results with two and three partitions.

Table 7: Result for the SOC d695 for testing time:

	Φi =max(i/p, o/p)	pi	Ni	fi	fi/Ni	ti
CMod1	32	12	-	-	-	12
CMod2	207	73	-	-	-	73
SMod3	34	75	1	32	32	2507
SMod4	39	105	4	211	52	5617
SMod5	304	110	32	1426	44	4994
SMod6	152	234	16	638	39	9399
SMod7	150	95	16	534	33	3263
SMod8	49	97	4	179	44	4409
SMod9	320	12	32	1728	54	714
SMod10	106	68	32	1636	51	3587

Table 8: Result for the SOC d695 with N_b=2:

w	TAM Partition	Assignment vector	Testing time
24	11,13	2, 1, 1, 1, 1, 1, 1,1, 1,2	3669107
36	15,21	2, 2,1, 2,1, 2, 1, 1, 1, 2	2096483
48	1,47	1, 1,1,1, 1, 1, 1, 2, 1, 2	3756070

Table 9: Result for the SOC d695 with N_b=3:

w	TAM Partition	Assignment vector	Testing time
24	6,8,10	2,2,3,3,3,1,1,2,2,1	2015693
36	8,12,16	3,3,2,1,1,2,1,3,2,1	2323724
48	12,16,20	1,1,2,2,3,3,1,2,1,3	**2006516**

5.4 Result For SOC f2126

The f2126 SOC benchmark consists of 5 modules, out of which 1 are combinational and 4 are sequential circuits. The Table-10 give the number of test cycles and Table-11 and Table-12 gives the results with two and three partitions.

Table 10: Result for the SOC f2126:

	Φi =max(i/p, o/p)	Pi	Ni	fi	fi/Ni	ti
CMod1	140	196	-	-	-	196
SMod2	529	334	8	7990	9987	334664
SMod3	30	103	1	452	452	47111

SMod4	139	422	16	5102	318	134936
SMod5	30	103	1	452	452	47111

Table 11: Result for the SOC f2126 with N_b=2:

w	TAM Partition	Assignment vector	Testing time
24	11,13	1,1,1,1,2	192065060
36	15,21	2,1,1,1,2	190467366
48	1,47	1,1,2,1,2	178478026

Table 12: Table 9: Result for the SOC f2126 with N_b=3:

w	TAM Partition	Assignment vector	Testing time
24	6,8,10	2,2,1,3,1	19259199
36	8,12,16	3,3,2,1,1	**17271808**
48	12,16,20	1,1,2,2,3	17338123

5.5 Result For SOC q12710

The q12710 SOC benchmark consists of 4 modules, out of which 0 are combinational and 4 are sequential circuits. The Table-13 give the number of test cycles and Table-14 and Table-15 gives the results with two and three partitions.

Table 13: Result for the SOC q12710:

	Φi =max(i/p, o/p)	pi	Ni	fi	fi/Ni	ti
SMod1	777	852	3	2085	695	593687
SMod2	3784	1314	4	4808	1202	1581944
SMod3	1316	1223	3	3049	1016	1244807
SMod4	1316	1223	3	3049	1016	1244807

Table 14: Result for the SOC q12710with N_b=2:

w	TAM Partition	Assignment vector	Testing time
24	11,13	1,2,2,1	191097360
36	15,21	2,2,1,1	173716096
48	1,47	1,2,1,2	174632836

Table 15: Result for the SOC q12710with N_b=3:

w	TAM Partition	Assignment vector	Testing time
24	6,8,10	1,1,3,2	175416580
36	8,12,16	3,1,2,1	172042580
48	12,16,20	1,2,3,1	**172017296**

5.6 Result for SOC h953

The h953 SOC benchmark consists of 8 modules, out of which 1 are combinational and 7 are sequential circuits. The Table-16 give the number of test cycles and Table-17 and Table-18 gives the results with two and three partitions.

Table 16: Result for the SOC h953:

	Φi =max(i/p, o/p)	pi	Ni	fi	fi/Ni	ti
SMod1	152	341	4	1129	282	6785
SMod2	89	9	2	654	327	3279
SMod3	17	39	2	63	31	1279
SMod4	88	49	4	84	21	1099
SMod5	19	10	4	483	120	13430
SMod6	15	182	4	737	184	33854
CMod7	80	65	-	-	-	65
SMod8	69	305	8	1505	188	57833

Table 17: Result for the SOC h953 with N_b=2:

W	TAM Partition	Assignment vector	Testing time
24	11,13	1,2,1,2,1,2,2,2	34530866
36	15,21	2,1,2,1,2,1,1,2	49509613

48	1,47	1,1,1,1,1,2,1,2	34618453

Table 18: Result for the SOC h953 with N_b=3:

w	TAM Partition	Assignment vector	Testing time
24	6,8,10	2,2,3,3,3,1,1,2	39566982
36	8,12,16	3,3,2,1,1,2,1,3	**38326759**
48	12,16,20	1,1,2,2,3,3,1,2	38682661

5.7 Result For SOC g1023

The g1023 SOC benchmark consists of 14 modules, out of which 2 are combinational and 12 are sequential circuits. The Table-19 give the number of test cycles and Table-20 and Table-21 gives the results with two and three partitions.

Table 19: Result for the SOC g1023:

	Φi =max(i/p, o/p)	Pi	Ni	fi	fi/Ni	ti
SMod1	273	134	14	592	42	5804
SMod2	221	74	2	167	83	6299
SMod3	192	57	1	53	53	3131
SMod4	155	268	4	216	54	14794
SMod5	32	51	4	127	31	1683
SMod6	20	36	2	94	47	1775
SMod7	20	34	2	94	47	1679
SMod8	80	31	2	104	52	1695
SMod9	56	68	1	64	64	4484
SMod10	377	29	1	13	13	419
SMod11	191	15	1	9	9	159
SMod12	161	16	1	13	13	237
CMod13	64	512	-	-	-	512
CMod14	140	1024	-	-	-	1024

Table 20: Result for the SOC g1023 with N_b=2:

w	TAM Partition	Assignment vector	Testing time
24	11,13	1,2,1,1,2,2,1,2,1,2,1,2,2,1	6023204
36	15,21	2,2,1,1,2,1,2,2,2,2,1,1,1,2	4780124
48	1,47	1,1,2,2,1,2,1,1,1,1,2,2,2,1	4888048

Table 21: Result for the SOC g1023 with N_b=3:

w	TAM Partition	Assignment vector	Testing time
24	6,8,10	3,3,3,1,1,1,2,2,2,3,1,2,2,1	5068311
36	8,12,16	2,2,2,3,3,3,1,1,1,2,3,1,1,3	**4186968**
48	12,16,20	1,1,1,2,2,2,3,3,3,12,3,3,2	4874120

5.8 Result For SOC p34392

The p34392 SOC benchmark consists of 20 modules, out of which 15 are combinational and 5 are sequential circuits. The Table-22 give the number of test cycles and Table-23 and Table-24 gives the results with two and three partitions.

Table 22: Result for the SOC p34392:

	Φi =max(i/p, o/p)	pi	Ni	fi	fi/Ni	ti
CMod1	32	11	-	-	-	11
SMod2	94	210	1	806	806	170276
SMod3	263	514	29	7548	260	134414
CMod4	37	3108	-	-	-	3108
CMod5	38	6180	-	-	-	6180
CMod6	62	12336	-	-	-	12336
CMod7	11	1965	-	-	-	1965
CMod8	9	512	-	-	-	512
CMod9	46	9930	-	-	-	9930
CMod10	41	228	-	-	-	228
SMod11	207	454	19	4731	249	113749
CMod12	23	9285	-	-	-	9285
CMod13	7	173	-	-	-	173
CMod14	16	2560	-	-	-	2560
CMod15	11	432	-	-	-	432
CMod16	22	4440	-	-	-	4440
CMod17	7	128	-	-	-	128
CMod18	15	786	-	-	-	786
SMod19	212	745	14	6555	468	349873
CMod20	62	12336	-	-	-	12336

Table 23: Result for the SOC p34392 with N_b=2:

W	TAM Partition	Assignment vector	Testing time
24	11,13	1,2,1,2,1,1,1,2,2,2,2,1,1,2,1,2,1,2,2,1	500167896
36	15,21	1,2,1,2,1,1,1,2,2,2,2,1,1,2,1,2,1,2,2,1	499458900
48	1,47	1,1,2,2,2,1,1,1,2,2,1,1,2,1,2,1,1,1,2,1	508989893

Table 24: Result for the SOC p34392 with N_b=3:

w	TAM Partition	Assignment vector	Testing time
24	6,8,10	1,2,3,3,2,1,1,2,3,3,2,1,1,1,3,3,2,2,3,3	248701346
36	8,12,16	1,2,3,3,2,1,1,2,3,3,2,1,1,1,2,3,2,2,3,3	**248648686**
48	12,16,20	1,2,3,3,2,1,1,2,3,3,2,1,1,1,2,3,2,2,3,3	508989893

5.9 Comparison of SOCs with respect to Testing time

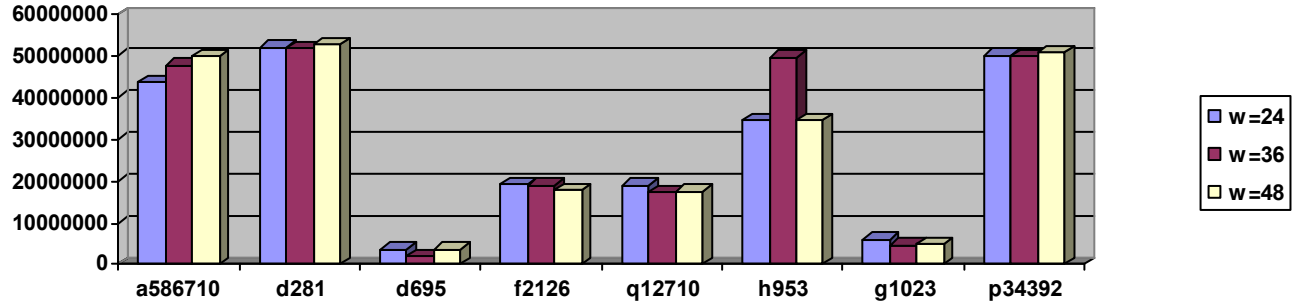


Fig:5.9.1 Comparison of various SOC with respect to testing time for N_b=2

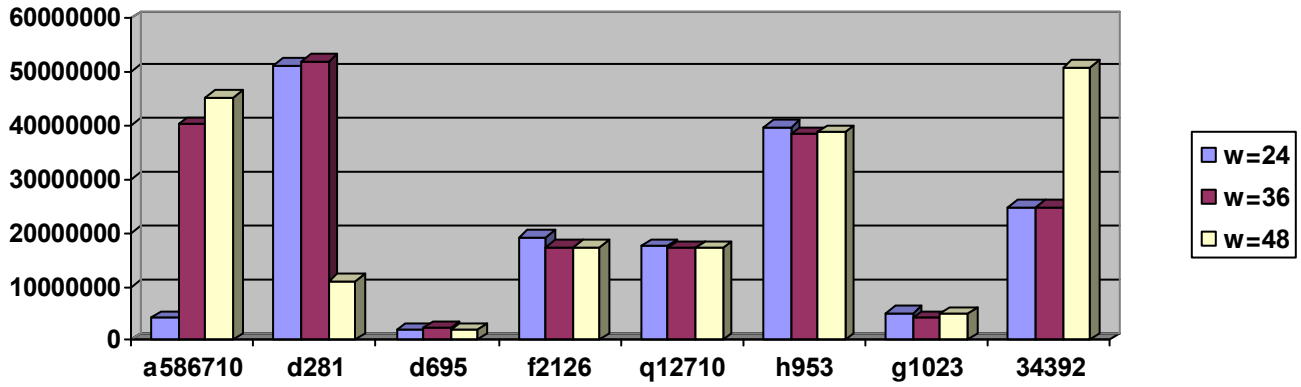


Fig:5.9.2 Comparison of various SOC with respect to testing time for N_b=3

CHAPTER 6:CONCLUSION AND FUTURE WORK

6.1 Conclusion

This thesis has presented contributions in the area of testing. Modern semiconductor design methods and manufacturing technologies enable the creation of a complete system on a single chip, the so called System-On-Chip or SOC. To reduce time-to-market for large SOCs, reuse of predesigned and preverified blocks called cores is employed. The objective of the work described in this thesis was to reduce the testing time. This approach provides optimal results comparable to other methods of similar problems. Based on our experiments, the test results for different ITC-02 SOC Test Benchmark circuits are presented. The results of GA-based approach are shown to be superior to the heuristic approaches proposed in the literature. The experimental results are shown for different ITC-02 SOC benchmark circuits with two and three partitions. The result gives good approximation compared to two partitions.

6.2 Future Work

There are several possible extensions to the work presented in this thesis. In this section possible approaches, directly related to the work are presented:

- In this approach, there will be different levels for testing parent and child cores. So that both can be testing concurrently and will achieve minimum possible test application time.
- In this approach, further partitioning of TAM will be done to reduce the testing time.
- Other scheduling algorithms will be applied for the same partitioning and compare the results.

References

1) Krishnendu Chakrabarty, "Test Scheduling for Core-Based Systems Using Mixed-Integer Linear Programming", IEEE Transactions on Computer-Aided Design Of Integrated circuits and systems, vol.19, no.10, pp.1163-1174, 2000.
2) Vikram Iynegar and Krishnendu Chakrabarty, "System-on-Chip Scheduling with Precedence Relationships, Preemption, and Power Constraints", IEEE Transactions on Computer-Aided Design Of Integrated circuits and systems, vol.21, no.9, pp.1088-1094,2002.
3) Vikram Iynegar, Krishnendu Chakrabarty and Erik Jan Marinissen, "On using Rectangle Packing for SOC Wrapper/Tam Co-OPtimization", Proceedings 20th IEEE VLSI Symposium, pp.253-258, 2002.
4) Vikram Iynegar, Krishnendu Chakrabarty and Erik Jan Marinissen, "Test Access Mechanism Optimization, Test Scheduling, and Tester Data Volume Reduction for System-on-Chip" , IEEE Transcations on computers, Vol. 52, No. 12, pp.1619-1632, 2003.
5) Santanu Chattopadhyay and K.Sudarsana, "Genetic Algorithm based Test Scheduling and test mechanism for System on chips", Proceedings 16th IEEE International Conference on VLSI Design, pp.341-346, 2003.
6) Giri,Sarkar, and Chattopadhyay, "A Genetic Algorithm based Heuristic Technique for Test Scheduling in Core-based SOCS", IEEE IFIP International Conference on VLSI, pp.302-306, 2005.
7) Farah, and Harmanani, "Integrating Wrapper Design, TAM assignment, and Test scheduling for SOC Test optimization", IEEE Journal, pp.149-152, 2008.
8) Xing Gu, Qun Yang, De-chang Pi and He-yang Ke, "A Global Scheduling Algorithm Based on Dynamic Critical Path", 10th International Joint Conference of IEEE on TrustCom, pp.1335-1338, 2011.
9) Jinyi Zhang,Feng Lin, Yanhui Jiang, and Jia Wang, "TAM Optimization and Test Scheduling for SOC Based on Zigzag Design Flow", IET Conference on Wireless, mobile and Sensor Networks, pp.928-931, 2007.

10) Shiwang Hou, Yongjiang, Liu, Haijun, and Yuepeng Chen, "A Self-Crossover Genetic Algorithm for Job Shop Scheduling Problem", IEEE International Conference on Industrial Engineering and Engineering Management, pp.549-553, 2011.

11) Chih-Sheng Hou,Jin-Fu Li, and Che-Wei Chou, "Test and Repair Scheduling for BIST RAMs in SOC", IEEE International Symposium on Electronic Design, 2010.

12) Hou,Li, and Chou, "Test and Repair for BISR RAMs in SOCs", IEEE 31st International VLSI Test, pp.3-7, 2010.

13) Li Jie,Guo Ruifeng, and Shao Zhixiang, "The Research of Scheduling Algorithms in Real time system", IEEE International Conference on Computer and Communication Technologies in Engineering, pp.333-336, 2010.

14) Yadav,Mishra, Prakash, and Sharma, "An improved Round Robin Scheduling Algorithm for CPu scheduling", IEEE Journal of Global Science,Vol.02, No.04, pp.1064-1066, 2010.

15) Zorian, Marinissen and Dey, "Testing Embedded-Core Based System Chips", Proceedings IEEE International Test Conference, pp.52-60, 1998.

16) Pounget and Erik, "Multiple-Constrained Driven System-on-chip Test Time Optimization", Springer Journal of Electronic Testing, pp.599-611, 2005.

17) Erik, Peng, "An Integrated Framework for the Design and Optimization of SOC Test Solutions", Springer Journal of electronic testing, pp.385–400, 2002.

18) Yervant,Erik and Dey, "Testing Embedded-core-based System Chips", Proceedings IEEE Test conference on Computer, Vol. 32, No. 6, pp.52-60,1998.

19) Chandan Giri, S. Sarkar and Santanu Chattopadhyay, "A Genetic Algorithm Based Heuristic Technique for Power Constrained Test Scheduling in Core-based SOCs", IFIP International Conference on VLSI, pp.320-323, 2007.

20) K. Charkrabarty, V. Iyengar and M. D. Kkrasniewski, "Test Planning for Modular Testing of Hierarchical SOCs," IEEE Trans. on computer aided designs of integrated circuits and systems, vol. 24, 2005.

21) Malanie Mitchell, “An Introduction to Genetic Algorithm” , MIT Press,1996.

22) Z. Jinyu, Li Xunsheng,Guo Bing, Xiong Guangze and Sang Nan, “A Test Scheduling Scheme for Core-Based SOCs Using Genetic Algorithm”, IEEE International Conference on Embedded Softwrae and Systems Symposia, pp.38-43, 2008.

23) E.J.Marinissen, V.Iyengar, and K.Chakrabarty, “A set of benchmarks for modular testing of SOCs”, IEEE International Proceedings, Conference, pp.519-528, 2002.

24) Jyotishree and Rakesh Kumar, “Blending Roulette Wheel Selection and Rank Selection in Genetic Algorithm”, IEEE International Journal of Machine Learning and Computing, Vol.2, N0.4, pp.365-369, 2012.

25) Yilmaz Kaya, Murat Uyar and Ramazan Tekin, “A Novel Crossover Operator for Genetic Algorithm” ,IEEE International Journal of Soft Computing and Engineering,Vol.2, No. 3, pp.56-60, 2011.

26) Y Yu, X Y Peng and Y Peng, “A test Scheduling Algorithm Based on Two-Stage GA”, IEEE International Symposium on Instrumentation Science and Technology, pp.658-662, 2006.

27) S. Goel, E. J. Marinissen, A. Sehgal and K. Chakrabarty, “Testing of SOCs with Hierarchical Cores: Common Fallacies, Test Access Optimization and Test Scheduling,” Proc. IEEE Tran. on computers, vol. 58, 2009.

28) P. Sakthivel and P.Narayanasamy, “An Approach to the Design of Optimal Test Scheduling for System-On–Chip based on Genetic Algorithm”, IAACISE, Springer, pp.25-28, 2007.

29) E. Larsson, “Introduction to Advanced System on Chip Test Design and Optimization,” Springer, 2005.

30) Vikram Iyengar, Krishnendu Chakrabarty and Erik Jan Marinissenz, “Recent Advances in Test Planning for Modular Testing of Core-Based SOCs”, IEEE, 11th Asian Test Symposium, , 2002.

31) Vikram Iyengar, Krishnendu Chakrabarty and Erik Jan Marinissenz, "Test Wrapper and Test Access Mechanism Co-Optimization for System-on-Chip", Proceedings IEEE International Test ConFerence, pp.301-308, 2011.

32) E.J. Marinissen, S.K. Goel and M. Lousberg, "Wrapper design for embedded core test", Proceedings of International Test Conference(ITC), pp. 911–920, 2000.

33) Jung-Beem Im, S.Chun and Geunbae,Jin-Ho An and Sungho Kang, "RAIN(Random Insertion) scheduling algorithm for SOC test", 13th Asian IEEE Test Symposium,pp.242-247, 2004.

34) Fariha Nosheen, Sadia Bibi and Salabat Khan, "Ant Colony Optimization based Scheduling Algorithm", IEEE International Conference on Open Source Systems and Technologies (ICOSST), pp.18-22, 2103.

35) T.H. Cormen, et al., Introduction to algorithms, 2001.

36) E. J. Marinissen, R. Arendsen, G. Bos, H. Dingemanse, H.Lousberg, and C. Wouters, "A Structured and Scalable Mechanism for Test Access to Embedded Reusable Cores," In Proceedings of IEEE International Test Conference (ITC), pp.284–293, 1998.

37) James Chin, Mehrdad Nourani, "FITS: An Integrated ILP-Based Test Scheduling Environment", IEEE transactions on computers, vol. 54, no. 12, December 2005.

38) W. Zou, S. R. Reddy, I. Pomerance, Y. Huang, "SOC Test Scheduling Using Simulated Annealing", Proceedings of VLSI Test Symposium (VTS), Napa Valley, CA, USA, pp. 325-330, April 2003.

39) L.Li and K. Chakrabarty, Test set embedding for deterministic BIST using a reconfigurable interconnection network, IEEE Trans. on Computer-Aided Design, 23(9), pp. 1289–1305, September 2004.