THE UNIVERSITY *of* EDINBURGH

Automatic Test Pattern Generation for Asynchronous Circuits

Dilip P. Vasudevan

Doctor of Philosophy
Institute of Computing Systems Architecture
School of Informatics
University of Edinburgh

2011

Abstract

The testability of integrated circuits becomes worse with transistor dimensions reaching nanometer scales. Testing, the process of ensuring that circuits are fabricated without defects, becomes inevitably part of the design process; a technique called design for test (DFT). Asynchronous circuits have a number of desirable properties making them suitable for the challenges posed by modern technologies, but are severely limited by the unavailability of EDA tools for DFT and automatic test-pattern generation (ATPG).

This thesis is motivated towards developing test generation methodologies for asynchronous circuits. In total four methods were developed which are aimed at two different fault models: stuck-at faults at the basic logic gate level and transistor-level faults. The methods were evaluated using a set of benchmark circuits and compared favorably to previously published work.

First, ABALLAST is a partial-scan DFT method adapting the well-known BALLAST technique for asynchronous circuits where balanced structures are used to guide the selection of the state-holding elements that will be scanned. The test inputs are automatically provided by a novel test pattern generator, which uses time frame unrolling to deal with the remaining, non-scanned sequential C-elements. The second method, called AGLOB, uses algorithms from strongly-connected components in graph graph theory as a method for finding the optimal position of breaking the loops in the asynchronous circuit and adding scan registers. The corresponding ATPG method converts cyclic circuits into acyclic for which standard tools can provide test patterns. These patterns are then automatically converted for use in the original cyclic circuits. The third method, ASCP, employs a new cycle enumeration method to find the loops present in a circuit. Enumerated cycles are then processed using an efficient set covering heuristic to select the scan elements for the circuit to be tested.Applying these methods to the benchmark circuits shows an improvement in fault coverage compared to previous work, which, for some circuits, was substantial. As no single method consistently outperforms the others in all benchmarks, they are all valuable as a designer's suite of tools for testing. Moreover, since they are all scan-based, they are compatible and thus can be simultaneously used in different parts of a larger circuit.

In the final method, ATRANTE, the main motivation of developing ATPG is supplemented by transistor level test generation. It is developed for asynchronous circuits designed using a State Transition Graph (STG) as their specification. The transistor-level circuit faults are efficiently mapped onto faults that modify the original STG. For each potential STG fault, the ATPG tool provides a sequence of test vectors that expose the difference in behavior to the output ports. The fault coverage obtained was 52-72 % higher than the coverage obtained using the gate level tests.

Overall, four different design for test (DFT) methods for automatic test pattern generation (ATPG) for asynchronous circuits at both gate and transistor level were introduced in this thesis. A circuit extraction method for representing the asynchronous circuits at a higher level of abstraction was also implemented.

Developing new methods for the test generation of asynchronous circuits in this thesis facilitates the test generation for asynchronous designs using the CAD tools available for testing the synchronous designs. Lessons learned and the research questions raised due to this work will impact the future work to probe the possibilities of developing robust CAD tools for testing the future asynchronous designs.

Acknowledgements

I would like to extend my sincere gratitude to my supervisor Dr. Aristides Efthymiou for his patience and guidance during my research endeavor. I am very thankful to him for mentoring and supporting during these years. I would like to take this opportunity to thank my second supervisors Prof. Nigel Topham, Dr. Murray Cole and Prof. D.K. Arvind for their continuous guidance.

I would like to thank my postdoctoral mentors Prof. Michel Schellekens and Dr. Emanuel Popovici during my Thesis writing periods. Their support was invaluable and I would like to acknowledge their guidance for my smooth transition in to next stage of my academic career. I would like to thank my external examiner and my internal examiner for providing very helpful and guiding recommendations.

I would also like to thank Prof. Parag K. Lala and Prof. Patrick Parkerson who introduced me to research career during my postgraduate studies.

I would like to acknowledge the support offered by Engineering and Physical Sciences Research Council EPSRC and School of Informatics for my three years of study. I would like to thank UK ASYNC Forum for providing me a platform to network with other Asynchronous Design researchers.

I am grateful to my parents and my brother for their loving support. I would like to extend my sincere thanks to all my friends, relatives and acquaintances who played significant part in keeping me going during my PhD years.

I would like to extend my thanks to the Edinburgh Tango Society, which helped me spend my leisure time embarking something creative.

Finally, I would like to acknowledge my sincere thanks to the comments and reviews of all the blind reviewers of my conference papers, who made me a stronger and stronger academic writer and researcher.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Dilip P. Vasudevan*)

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Synchronous circuit design has been considered the standard for industrial practice due to the availability of advanced CAD tools and testing strategies. At deep sub micron levels, global clock synchronization, power consumption and noise factors are affecting the design performance, as a result asynchronous circuit design is gaining its momentum currently over its synchronous counterpart. On the other hand, asynchronous circuits need thorough research on CAD tool development for the whole design flow with test generation [BE00]Asynchronous designs are classified into speed independent, delay insensitive, and quasi delay insensitive circuits.Thus it has different models and architectures to be designed with and each of them has its own circuit models and delay assumptions. Significant efforts have been taken to develop CAD tools for synthesis of asynchronous circuits which lead to several tools like Petrify [CKK$^+$96b] ,Tangram[KP01], Balsa [BE00] etc.,. Currently,very few tools (commercial tool from [Han])are available for test generation for asynchronous circuits. Testing is essential for the designed systems, as the fabrication and component aging will cause defects in the circuits.

1.1.1 Past work

Several attempts to generate tests for asynchronous circuits have been made in the recent years. Some of the methods involved test generation based on the STG (State Transition Graph) specification of the design. The test methods were introduced mostly by traversing through states of the state transition graph of the circuit. Some attempts have been made to generate test patterns for these circuits at the gate level. Also the test generation was specifically based on DFT methods, which makes the test generation methods dependent on the design methodology of the circuits. Several methods for generating acyclic circuit(circuits without feedback) from cyclic cir-

cuits(circuits with feedback) have been introduced lately [Edw03],[Mal93],[Wei72],[Niv04]. But the methods are restricted for the cyclic circuit without state holding elements and which does not oscillate. But oscillations are predominant in asynchronous cyclic circuits and also presence of state holding elements like c-elements are common in them. Full scan based test generation for the circuits have been proposed in [Bee03]. Partial scan based method for self-timed circuit was proposed in [KB95]. The work in [BCR96] introduces a synchronous test generation to generate test for asynchronous circuits. A STG(State Transition Graph) based approach of test pattern generation was carried out in [RCPP97]. Test pattern generated were applied synchronously to test the target asynchronous circuits. A test generation method for testing redundant circuits in asynchronous designs was introduced in [LKL94] which used a method called "Variable Phase Splitting" to generate test patterns for these NCL circuits which is acyclic. A partial scan based delay fault testing of asynchronous circuit was acclaimed in [KKL$^+$98]. An algorithm similar to the proposed algorithm on this paper was used to test path delay faults. The work in [KSS02] introduced a test method for a subclass of asynchronous circuits called NCL(Null Convention Logic). This method is also based on partial scan test generation by breaking feedback loops. A partial scan test generation method for asynchronous SOC interconnect was presented in [ABE05]. The method focused on generating test for asynchronous interconnect named CHAIN. In [Ron94], a partial scan test generation method for DCC error corrector was provided. A fault simulator called FSIM was used for fault simulation. Micropipelines form the vital components in AMULET processor design and Scan testing for these micropipelines was introduced in [PF95b]. Lately a systematic scan insertion technique was introduced to test Asynchronous interconnects [SO08]. Also a recent work on automating test generation for asynchronous NCL circuits was published in [WA08]. This method promises near 100% test coverage for most of the NCL libraries used to design the NCL circuits. A detailed literature review on related works is given in chapter 3.

1.1.2 Motivation

Most commonly used testing methods for testing digital circuits are structural and functional testing. Both these methods have its own pros and cons.

Structural Vs Functional

Functional testing is the type of testing which is carried out by validating the design under test by its functional specification. This method is more closer to the verification. In other hand, the structural testing is more closer to the implemented circuit structure of the DUT. Asynchronous circuit design does not have a clear standard for its specification. Different research groups have different design methodologies for asynchronous circuit design. Lack of standard design methodology makes the functional testing harder. So taking the route of structural testing will

increase the developed test method more generic for all the design methods of asynchronous circuits. Hence this thesis follows one of the structural testing method called Scan testing for the DFT and preprocessing.

Partial scan based test generation is a promising approach to generate effective test patterns for sequential circuits. Several methods have already been implemented for synchronous sequential circuit. Adapting those methods for testing asynchronous sequential circuit is considered effective. Because the test generation process for asynchronous circuits can be followed in the same manner as that of synchronous circuits for most of the steps except that the feedback loops/cycles have to be appropriately handled. Though it seems easier, handling the cycles and oscillations due to them is a harder task. Thus two different ATPG methodologies for synchronous sequential circuits are studied and the useful aspects of those methods were adapted to develop the algorithms for asynchronous circuit based ATPG methods .

Ballast methodology of generating test for sequential circuit is a promising approach for partial scan based test generation of synchronous sequential circuits. The main technique used in this method involved generating a balanced sub-graph from the circuit topology graph of the sequential circuit which was proved to have equivalent combinational structure when the memory elements in the sub-graph are replaced by a wire. Thus the test patterns for the sequential circuits are generated by treating them as combinational equivalent. This same technique can be applied to the asynchronous sequential circuit to generate test.

As a next step, another partial scan synchronous sequential circuit based test generation method was adapted to define a test methodology for asynchronous sequential circuits. The main technique used in this method involved selecting the memory elements based on finding the strongly connected components (SCC) from the circuit topology graph of the sequential circuit. Thus the test pattern for the sequential circuits is generated by converting the selected memory elements in to the scan elements. This technique can be applied to the asynchronous sequential circuit to generate test. The main challenges faced by applying these technique to the asynchronous circuits are

- Asynchronous circuits have loops which makes them cyclic circuit whereas the synchronous method operates only on acyclic circuits.
- Asynchronous circuits consist of memory element other than latches. c-elements are the frequently appearing memory elements in asynchronous design. These elements constitute the local loop in overall circuit structure.
- The operation of all the c-elements cannot be controlled during their normal operation compared to normal latches controlled by clock.

Until now only the gate level test generation for asynchronous circuits were discussed. The

transistor level test generation is still an active research in synchronous design field also. The reason is that infamous stuck-at fault model cannot model all the defect level faults [FS88b] [Mal87]. Thus the transistor level test generation is one level further down the gate level test generation scenario. This level of test generation provides higher level of fault coverage and a closer realization of the physical defects. But this has to be traded off with the longer test generation time due to the drastic increase in the number of nodes to be tested. Fault simulation of the circuit at transistor level will take relatively longer time compared to the gate level simulation. Thus the test generation at transistor level design poses following drawbacks:

- Increase in number of fault sites to be tested
- Transistor level net list handling
- Longer fault simulation and test generation time

1.1.3 Asynchronous Design and Testing in Industry

Recently, asynchronous circuits based chip designs and their products and applications are introduced in industry. Several companies that design asynchronous circuit based chips are

- ARM,
- Tiempo,
- Elastix Corporation,
- Achronix,
- Handshake Solutions and
- Silistix.

1.1.3.1 Elastix Corporation

The quote from the Elastix Corporation [Ela] on testing asynchronous circuits named "Elastic Circuits" is shown below:

"Elastic circuits are tested in the very same way as synchronous circuits. The fact that the circuit looks like its synchronous counterpart makes it possible to use the same test structures (e.g., scan chains, BIST) and patterns that were initially designed for the synchronous circuit. Additionally, the elastic circuit requires some specific logic to test the Elastic Clocks. This is a negligible extra logic and a small set of extra test patterns."

As stated above, "specific logic" is required to test the elastic circuits. So the test process is more specific to the elastic circuits.

1.1.3.2 Tiempo

In Tiempo [Tie], test process is aided by the type of design method used to design the asynchronous circuits. The design method is based on implementation of group of modules communicating using the handshaking protocols. They use the very well known dual rail logic or other multi-rail encoding for the data detection between the modules. They have implemented this design method by a succession of wavefronts. It has been claimed that the occurrence of the stuck-at-fault in these designs will eventually stop the flow of data in the design as they are implemented as a succession of wave-fronts. Then the fault eventually blocks the handshaking protocol to continue to next stage. The faulty system is detected based on this behaviour.

1.1.3.3 ARM and Handshake Solutions

ARM996HS [BY07], A first licensable clockless processor was designed based on the TiDE design flow Haste. ARM [ARM] released this processor partnering with Handshake solutions. The testing process of this design is based on the full scan method which is still based on synchronous full scan method.

1.1.3.4 Other Companies

Achronix, Silistix , Timeless design automation are the other startups working based on the asynchronous design to extend it to SOC, NOC, FPGA and ethernet based applications. The test method applied by them is still based on synchronous design methodologies.

As discussed above, the test methods applied in industry are more specific to the design method used by them for implementing the asynchronous circuits. A generic or standard asynchronous circuit test methodology is not yet introduced at industry standard.

1.2 Contributions

This thesis is motivated towards developing several automatic test pattern generation (ATPG) methodologies for the asynchronous circuits that can be incorporated in to the currently available industrial synchronous testing tool. Fig.1.1 gives the overview of the contribution of the thesis to the asynchronous circuit test flow.

A high level extraction tool named ACLARION (chapter 5) for extracting the high level circuit structure of the asynchronous circuits was implemented. Always the DFT(Design For Test) methods demanded clear partition and clustering of the registers and combinational gates which will aid the test generation. To fulfill this demand, the tool was implemented. The main contributions based on this tool are

- circuit extraction algorithm for asynchronous circuits
- completely implemented extraction tool based on this algorithm

Next the work on ABALLAST(chapter 6) is motivated towards developing an automatic test pattern generation methodology which uses cyclic to acyclic circuit conversion, partial scan based test generation and Ballast methodology as aids. Thus the contributions of this method are

- Effective handling of the cyclic asynchronous circuits to accommodate them to the usual synchronous test generation flow
- Partial scan element selection based on balanced sequential structures
- Automatic Test pattern generation for the partial scan design generated

The test generation process in AGLOB(chapter 7) also uses cyclic to acyclic circuit conversion, partial scan based test generation and SCC based memory element selection as aids. Thus the contributions of this method are

- Effective handling of the cyclic asynchronous circuits to accommodate them to the usual synchronous test generation flow
- Partial scan element selection based on SCC
- Automatic Test pattern generation for the partial scan design generated

The method ASCP (chapter 8) proposed in this thesis is based on the mapping the partial scan selection problem to the set covering problem.

- A new partial scan selection algorithm based cycle enumeration and set coving problem
- Automatic Test pattern generation for the partial scan design generated

Also, a case study on the proposed test methods is carried out by comparing these three methods based on the figure of merits of each method.

For ATRANTE (chapter 9), the main motivation of developing ATPG is supplemented by transistor level test generation. Here the Petri net based representation of the asynchronous circuits and efficient mapping of transistor level faults to state transition graph (STG) based fault models are used to implement the ATPG methodology. The contribution of this method is:

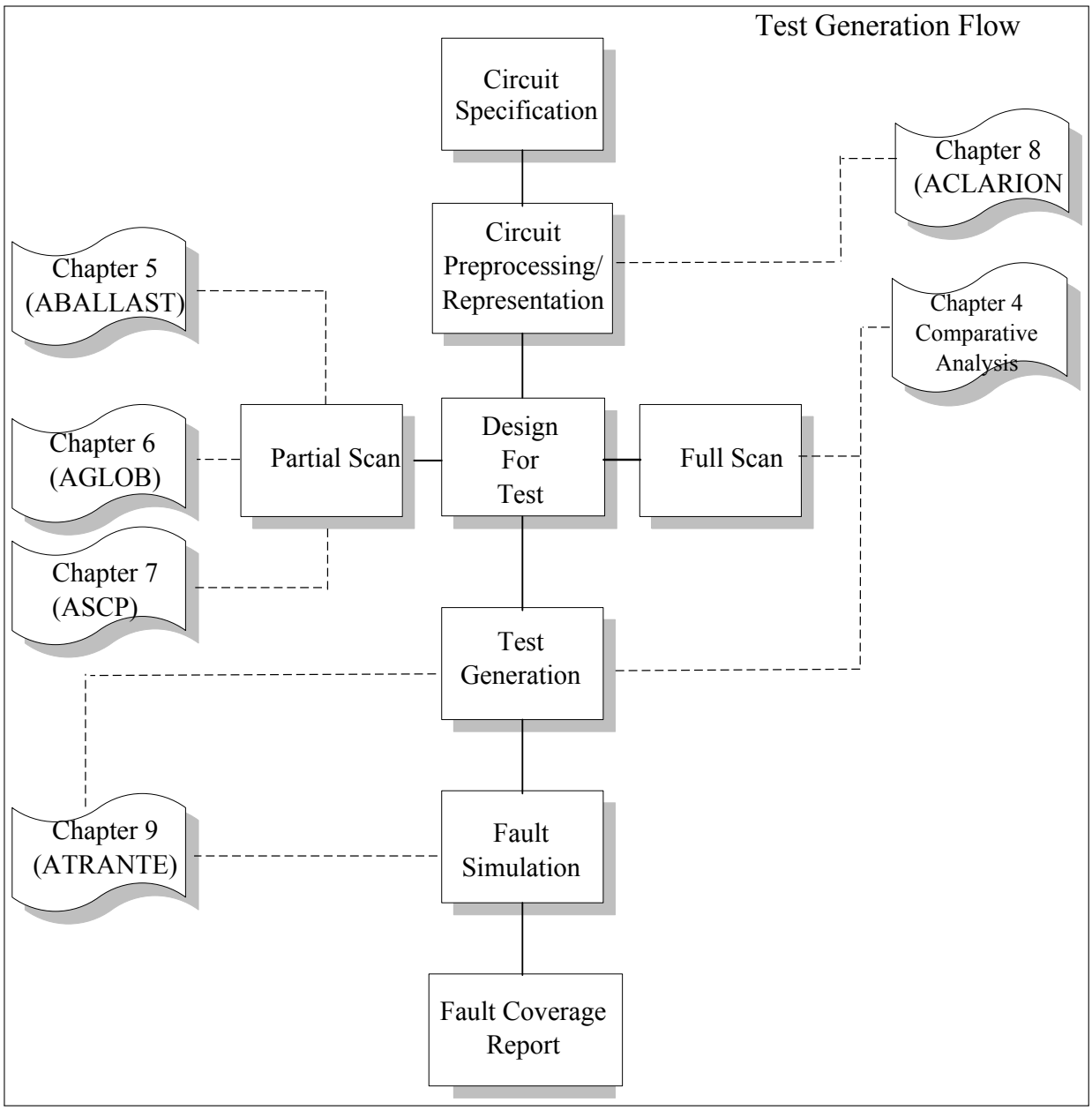


Figure 1.1: Contributation of the Thesis

- Using the transition fault model to generate fault lists
- Mapping of the transistor level faults to transition faults on STG
- Automatic Test Pattern generation method using the transition fault model with STG
- Implemented test pattern generator

1.3 Publications

Parts of this thesis work have been published in the following conferences and workshops.

- D.P.Vasudevan and A.Efthymiou,“ Automatic Test Pattern Generation For Asynchronous Circuits”, SIGDA PhD Forum, 48 th Design Automation Conference (DAC - 2011), San Diego, June 2011.
- D.P.Vasudevan and A.Efthymiou, “A Transistor Level Test Generation for Asynchronous Circuits”, IEEE International Workshop on Design and Test (IDT’09),Riyadh,April 2009. (Accepted)
- D.P.Vasudevan and A.Efthymiou, “Partial Scan Test Generation for Asynchronous Circuits Based on Breaking Global Loops”, 20th UK Asynchronous Forum,Manchester, September 2008.
- D.P.Vasudevan and A.Efthymiou, "A Partial scan based test generation for asynchronous Circuits", 11th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS’08), 2008
- D.P.Vasudevan, "A Novel method of Test generation for Asynchronous Circuits,", 2nd IEEE International Workshop on Design and Test (IDT’07), 2007
- D.P.Vasudevan and A.Efthymiou, " Comparative Analsysis stuck at test generation in asynchronous circuits” 1st IEEE International Workshop on Design and Test (IDT’06), 2006

1.3.1 Thesis Overview

The proposed thesis structure is as presented below. The organization of the thesis is shown in the Fig.1.2

A detailed background on Asynchronous circuit design, testing challenges in the asynchronous paradigm will be given in chapter 2. Then the chapter follows further detailing over the topics on testing (especially scan design) with details on full scan and partial scan design. Then several ATPG methods will be introduced and briefed followed by introducing several fault models.

Chapter 3 briefs the detail literature on the testing asynchronous circuits. The chapter is divided based on the following topics1) Design for test (DFT) for asynchronous circuits, 2) ATPG methods for asynchronous circuits, 3) Self checking designs of asynchronous circuits, and 4)

Testable asynchronous circuit design,5)Test Generation at defect/transistor level.5) Delay fault testing of asynchronous circuits. The related works on each of these topics are reviewed.

Chapter 4 carries out a comparison study on the two automatic test pattern generation methods.Background on the State Transition Graph (STG) based automatic test pattern generation are briefed.The test pattern generation based on the scan insertion technique are introduced.Then a comparison of test generated by these two approaches for a number of small benchmarks are presented. The chapter is concluded by stating the drawbacks and improvements to be incorporated in the proposed test methods.

The ABALLAST method is described in Chapter 5. The chapter gives further detailed background followed by the introduction of overall test methodology. The algorithms involved in this method will be briefed in detail. The following section will be on providing working examples for the test flow and comparison of results. The chapter will be concluded with the results.

The AGLOB method is described in Chapter 6. The chapter follows the same structure as of the chapter 5. Algorithms will be detailed in section 3, followed by the working examples and results in the section 4. The chapter is concluded with the results comparison.

Chapter 7 introduces the method ASCP based on Set Covering Problem. Background on Set Covering Problem and cycle enumeration were provided. Then the algorithms involved in developing the test methodology are briefed. Next section will be describing the overall test methodology. Results are presented and analysis of the experimental results are done. Next, overall case study is carried out as the second part of this chapter. All the three gate level test generation methods of ATPG are compared in terms of fault coverage, test coverage,test patterns, and area overhead.Detailed results of these three methods are analyzed and then the chapter is concluded.

Chapter 8 ACLARION is motivated towards development of an high level extraction tool. It gives the background required for the description of the extraction method. For giving foresight of the ATRANTE method in this chapter , a brief introduction to Petri nets, STG and SG will be provided first.Then it briefs the basic functions required for the implementation of the ACLARION extraction method and the overview of the methodology.Next section describes the proposed heuristics for the Register clustering process. Next section briefs the heuristics for the combination logic unit (CLU) clustering. Fanout clustering heuristics are introduced in detail in section6. Experimental results are analyzed with one working examples and the chapter is concluded.

Chapter 9 introduces the method ATRANTE. This chapter provides further details justifying the need for transistor level test generation in the introduction. Then the test methodology for

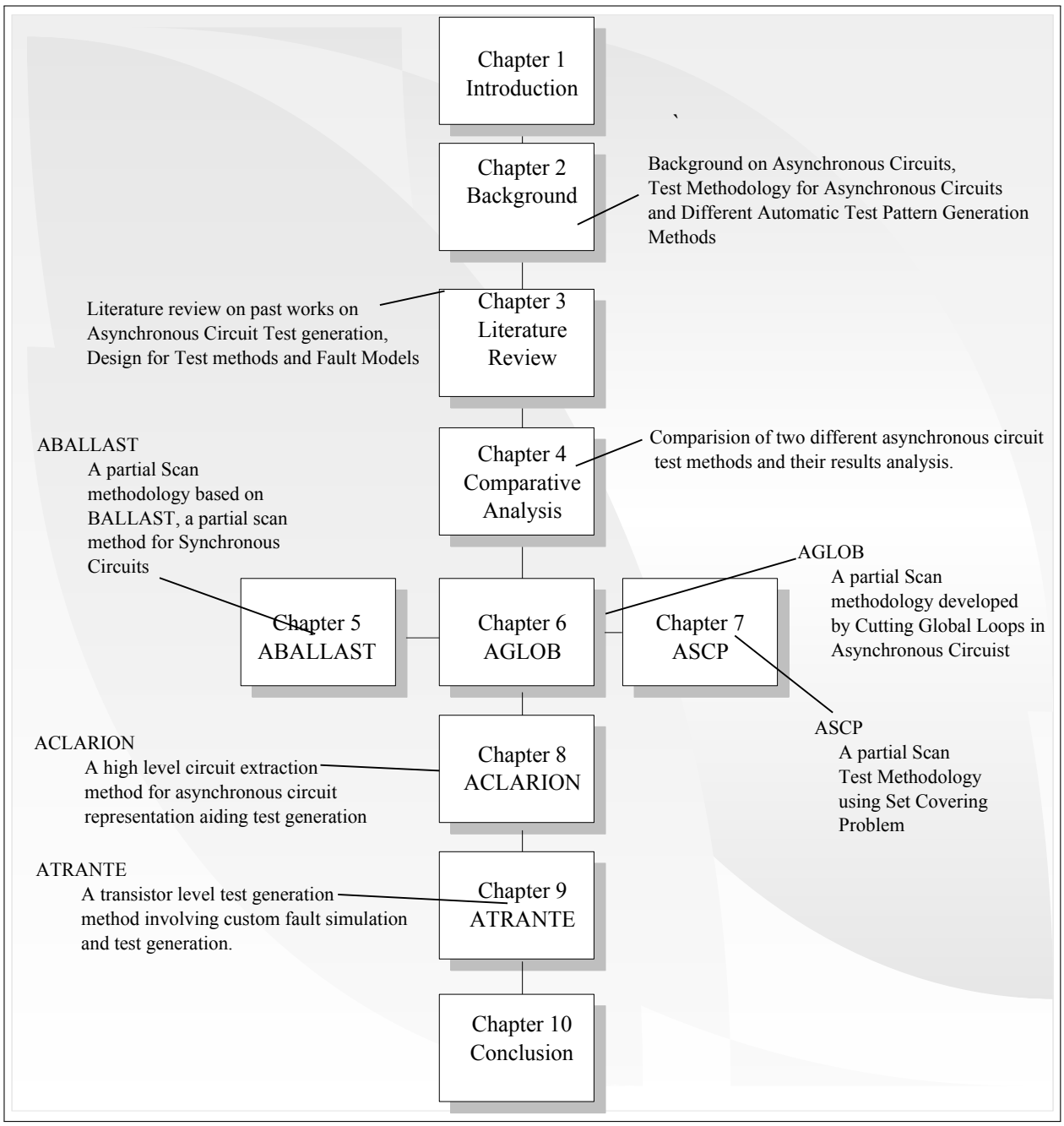


Figure 1.2: Organization of the Thesis

this method is briefed. Working examples and results are detailed in the next section followed by conclusion.

Final chapter 10 is on conclusion and future work.

1.4 Summary

This chapter gave a brief introduction about this thesis. The need for asynchronous design testing based on partial scan testing was discussed. The main motivation towards developing test generation methods for asynchronous circuits was stated briefly. The publications resulted from the several works carried out in this thesis were also listed. The next chapter will be giving a brief introduction to asynchronous design as a first part and the second part of the chapter will provide the introduction to the testing and testable design. This thesis was introduced in this chapter. The main motivation towards developing test generation methods for asynchronous circuits was stated briefly. The publications resulted from the several works carried out in this thesis were also listed. The next chapter will be giving a brief introduction to asynchronous design as a first part and the second part of the chapter will provide the introduction to the testing and testable design.

Chapter 2

Background

2.1 Asynchronous Design

2.1.1 Introduction

Asynchronous Design Methodologies follow the same procedure as synchronous design in most cases, except that the global clocking scheme is not present in it. The clock skew problem is overcome in this design due to the lack of global clock and this problem becomes more local for the circuits with fewer gates. Moreover, asynchronous circuits are considered as circuits modeled by the interconnection of gates and delay models.

Advantages: The main advantages of using asynchronous design are

- Modularity
- Average Case Performance
- Power Management
- Improved Electro-Magnetic Compatibility

Also the disadvantages in asynchronous design based systems are

- Increased Circuit Cost
- Complexity
- Lack of Tools
- Testing is harder

Fig.2.1 overall view of the asynchronous design methodology. The gate and delay models of the asynchronous circuit design is introduced further.

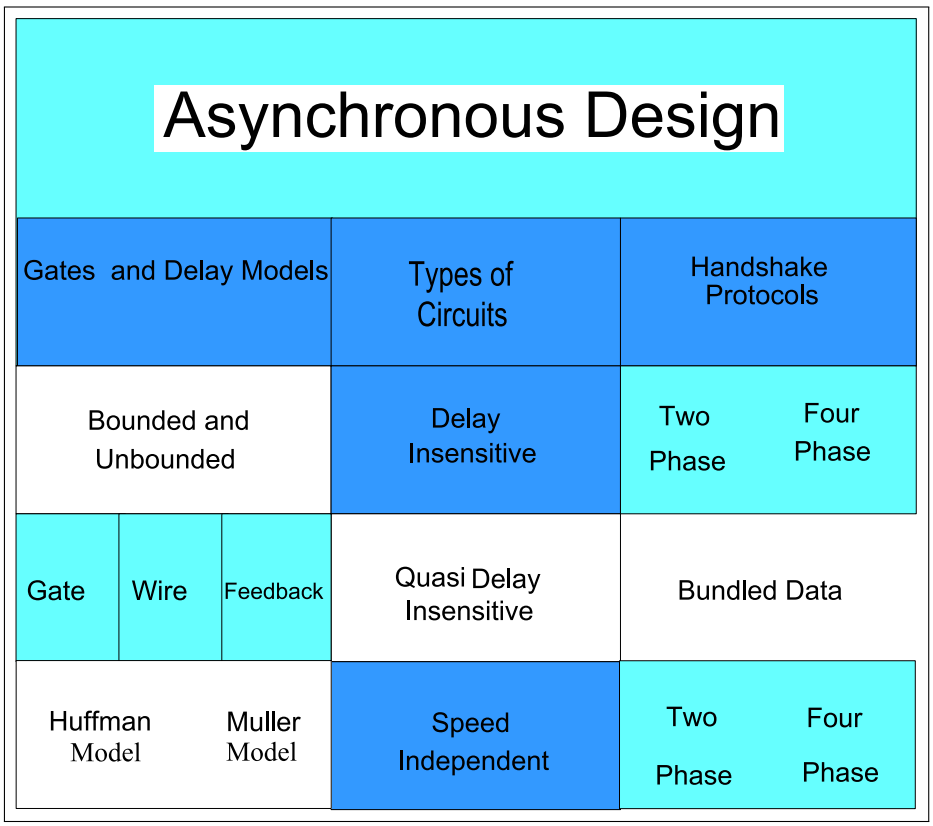


Figure 2.1: Asynchronous Design

2.1.2 Gates and Delay Models

Gates are composed of several inputs and outputs whose value is evaluated by its corresponding logic functions. Delay models are single input-single output elements which, does not evaluate any logic, but reproduce the input after a specified amount of time. Based on the magnitude, delay can further be classified as Bounded and Unbounded.

If the upper and lower bounds of the delay magnitude are known,then it is called bounded. If the bound for the magnitude is not known (but finite) with the only information on whether it is positive or negative is known, then it is unbounded delay. Based on the amount of memory associated with the delay element they are classified further as pure and inertial delays.

If the delay element duplicates the exact wave at its input to the output after the delay magnitude, they are pure If the pulses shorter than the delay magnitudes are filtered out they are called inertial delay

Based on the place where the delays are inserted, delay models are classified as follows

- Feedback Delay Model In this model, every feedback loop present is cut and replaced by at least one delay element.

- Gate Delay Model Here the circuit is modeled with every gate followed by exactly one delay element
- Wire Delay Model In this model, the delay is associated with each wire in the circuit and this can seen as, each input of the gates being associated with a delay.

  Based on different magnitudes and models of delay, some of the commonly known asynchronous models are given below:

- Huffman Model

  Huffman model introduced by Huffman[[Mye01]] is based on representing asynchronous circuits in to two components: the combinational network followed by the bounded inertial wire delay model and feedback lines, modeled by using unbounded inertial feedback delays. This model led to the introduction of several other models with a little variation

- Muller Model

  Muller[[Mye01]] introduced the class of asynchronous circuits in which each gate output is associated with an unbounded inertial delay element with delay of the wires being neglected. This type of circuits are called speed-independent circuits, since they operate correctly even in the presence of delays in their components.

2.1.3 Types of circuits

- Delay Insensitive circuits[[Mye01]] Delay insensitive circuits form the most robust class of asynchronous circuits. The circuits are modeled based on unbounded wire delay model . It is similar to the Muller model in terms of wires connecting a single output to a single input. The delay at the different ends of a fork vary (one output fanned out to more than one input) by placing a delay element at each gate inputs. Thus these forks are not isochronic due to the variation in delay. But only a small family of circuits constitutes this model.
- Quasi delay-insensitive and speed-independent circuits[[Mye01]]

  Quasi delay insensitive circuits[[Mye01]] are the versatile and popular class of asynchronous circuits which is derived from the delay insensitive circuits. In this type of circuits, the forks are considered to be isochronic. For a Delay insensitive circuit, the delays d1, d2 and d3 along with the gate delay dA, dB and dC are arbitrary. To obtain the quasi delay insensitive circuit, the condition d2 = d3 should be satisfied for some forks.

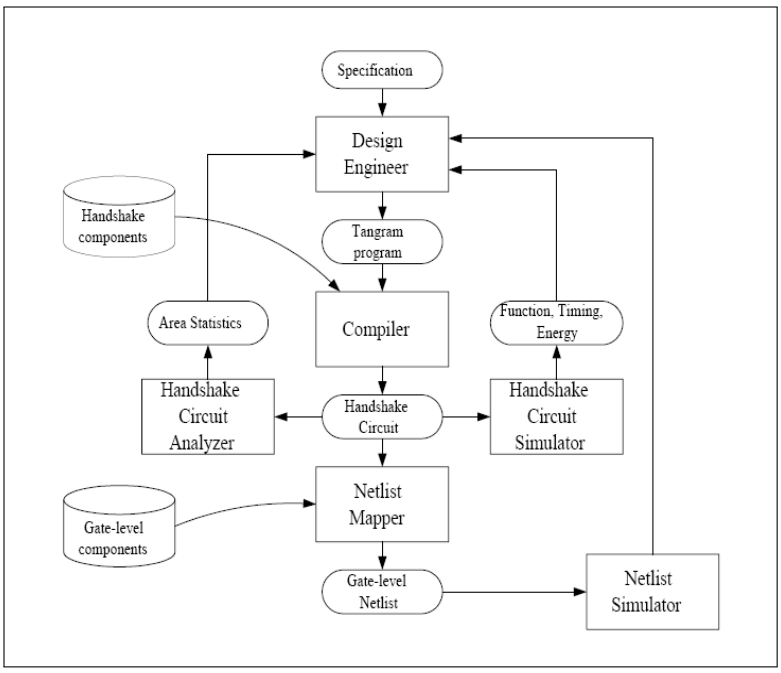


Figure 2.2: Asynchronous Design Flow - language based [KF91]

- Bounded-delay Circuits [[Mye01]] Bounded-delay circuits are the class of asynchronous circuits which use the fundamental-mode assumption that the environment must wait for long enough for the output data to stabilize on the circuit inputs. The fundamental-mode of operation was introduced by Huffman[[Mye01]] and later extended by Unger[Unger1969].

2.1.4 Logic Synthesis and Simulation:

As stated earlier, design flow of the asynchronous circuits can be summarized similar to that of synchronous circuits depending up on the asynchronous design style used. A typical synthesis flow for the CAD tool Tangram developed by Phillips is shown in Fig 2.2. Handshake circuits used in this design flow are the special class of circuits introduced by Phillips, which is implemented based on the handshake protocol. Here the circuit specification is based on the description language.

Several CAD tools for logic synthesis were developed and are further being researched. Some of the CAD tools like Petrify use graph based (petri net) representation of the circuits and the design is synthesized by BDD mapping of the graph specifications. Once the mapping is done, the design flow follows the same steps as used conventionally. A typical Logic synthesis cycle which uses petri net as the input net list specifications is shown in Fig 2.3

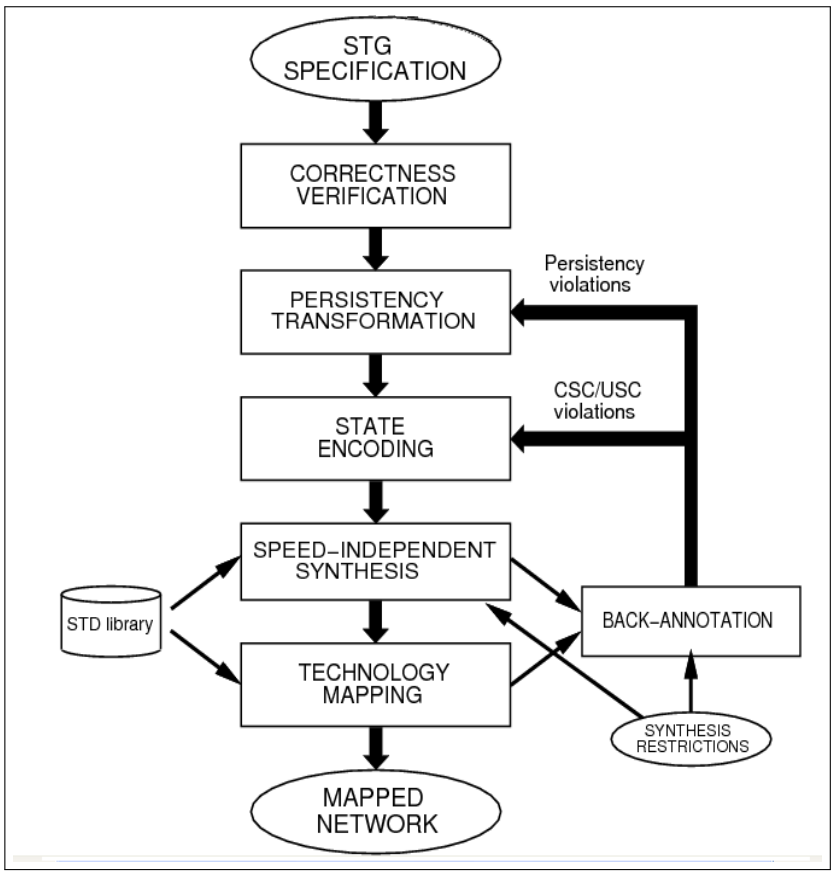


Figure 2.3: Asynchronous Design Flow: Graph based(Derived from petri net) [E.P97]

2.2 Testing

2.2.1 Introduction

Testing is essential for the designed systems, as the fabrication and component aging will cause defects in the designs. The defects in the design can be modeled as faults such as stuck-at, delay, bridging faults, etc. Thus for testing a circuit, fault model plays a major role on simulating the faults. Once the fault model is defined, it is applied to the design under test for generation of the test patterns which are used to validate the design.

2.2.2 Fault Modeling

Before proceeding for testing the circuit, the specific fault models for which the test has to be done should be selected. There are several types of fault models based on the kind of fault which occur during the physical design process like Short, Open circuit etc., some of the commonly used fault models are:

1. Stuck At Fault (Fig.2.4)
   *stuck-at-1:*A Fault at a node is said to be stuck-at-0, if it generates ’0’ output signal for both the value of input signals 0/1 and the node being observed through the primary output.
   Stuck-at-0: A fault at a node is said to be stuck-at-1, if it generates ’1’ output signal for both the value of input signals 0/1 and the node being observed through the primary output.

2. Transistor Level Faults (Fig.2.5)
   Stuck-Open Fault The Stuck-open fault occurs in the transistor when the transistor is always open due to physical defect. If considered as switch, the functionality of the transistor with this fault will always be that of a open (non-conducting) switch.
   Stuck-On Fault The Stuck-On fault occurs in the transistor when the transistor is always conducting due to the physical defect. If considered as switch, the functionality of the transistor with this fault will always be that of a closed/shorted switch.

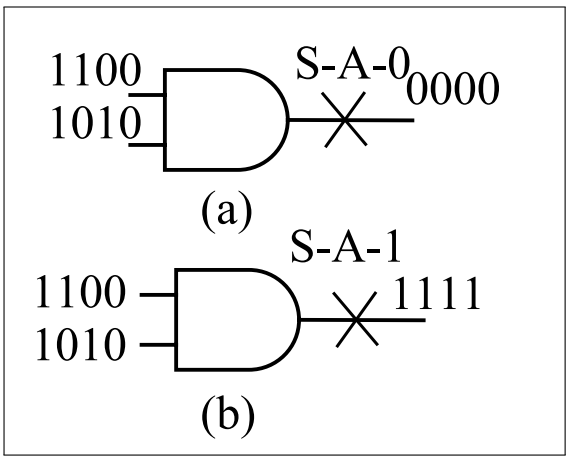


Figure 2.4: Stuck-at Faults

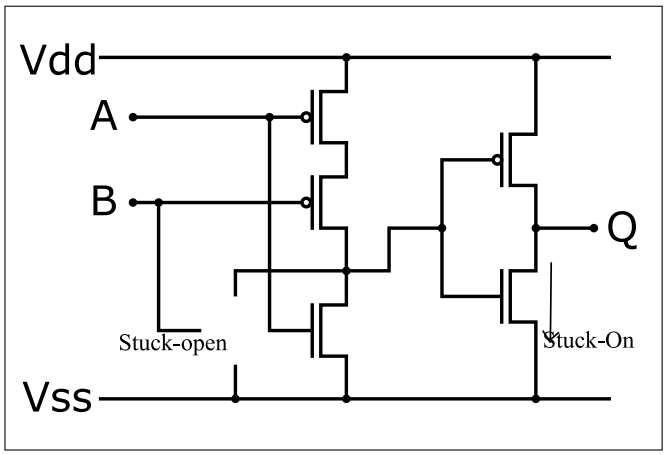


Figure 2.5: Stuck-Open/Close Faults

3. Bridging Fault: Bridging fault occurs when two nodes of the circuit at transistor/gate level were shorted together.

4. Transient Fault: Transient fault occurs at the event level or at the rise or fall transition of the signal either by getting inhibited or by unintended triggering.

5. Delay Fault: Delay faults are modeled based on the timing assumption of the circuits. Two types of delay faults are gate delay fault and path delay faults. The occurrence of this fault will cause the circuit to produce delayed response in the output for the specific input stimuli to the circuit.

2.2.3 Terminologies

1. Controllability:
   It is a testability measure, which defines whether the logic value at given node is controllable by effectively applying the vectors through the primary inputs.
2. Observability:
   It is a testability measure, which defines whether a fault excited at a given node is observable at the primary output of the circuit.
3. Fault Equivalence:
   If every test in the test set of one fault A, also detects the fault B, then the two faults are equivalent. This is used to reduce the number of faults that need to be tested.
4. Fault Dominance:
   If for two faults A and B, the test set of B is a subset of the test for A, then the fault A is said to be dominating the fault B.
5. D-Frontier:
   The D-frontier is composed of all the gates in the circuit being tested, whose output value is x(don't care), but one or more of their input has been set to either D or D'(where D and D' are the logic values used in D-algorithm to differentiate the good and faulty circuit logic values of a node). The D-Frontier is used in error propagation process.
6. J-Frontier:
   J-Frontier is composed of the set of all the gates in the circuit, whose output value is known,but are not implied(assumed based on the gate's functionality and net list connection) by their input values. This happens during the justification process(process of setting logic values on each node of the circuit during fault simulation), when a particular node is assigned a value to imply the value at the target fault node.

2.2.4 Automatic Test Pattern Generation

Test generation involves following basic steps to generate test vectors

1. Fault List Generation
2. Test Vector Generation
3. Fault Simulation
4. Test compaction

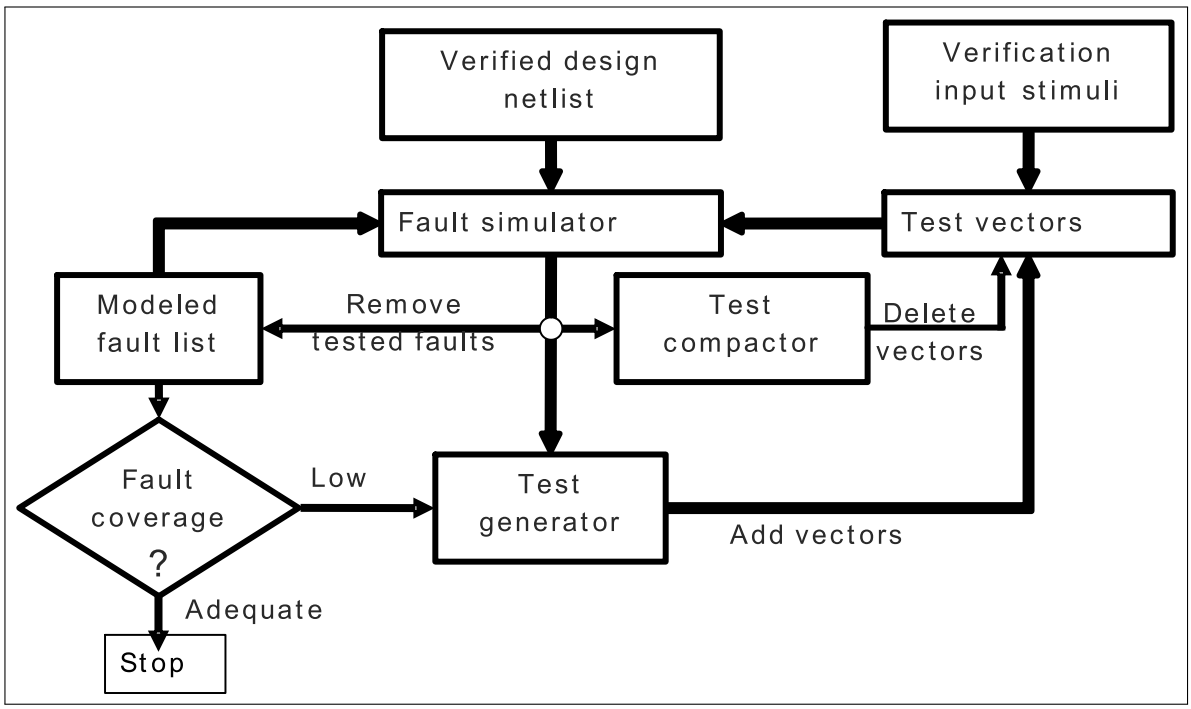


Figure 2.6: An ATPG System in a VLSI Design Process [MV00]

These steps can be automated to generate a automatic test generation system. A general ATPG system flow is shown in Fig.2.6.[MV00]

Verified net list is fed to the fault simulator, where the modeled faults are simulated over the input design net list. Once the fault is detected, the fault is removed from the fault list. The test generator generates the test vectors to be used to test the modeled fault list over the design net list using the fault simulator. Test compactor is used to generate optimal number of vectors to test the design by using the fault dominance and equivalence properties. Once all the faults are simulated and the test vectors are compact, the design is checked for the fault coverage. If the coverage is satisfactory, the system exists, otherwise the steps are repeated to get the desired fault coverage. At the event of not finding the optimal test coverage, the system exists with the low coverage test vectors or with a report on untestable faults and redundancies

2.2.5 ATPG Algorithms

Some of the classic ATPG algorithms are reviewed in this section. The algorithms reviewed are

- D-algorithm
- FAN

- PODEM

2.2.5.1 D algorithm

The pseudo code for the D-algorithm[[JPRS67]] is shown in Fig.2.7[[MAF90]]. This is the first algorithm proposed for the automatic test pattern generation for synchronous circuits. The algorithm is based on a newly introduced concept called D-Algebra. The logic values used in this algebra are 0,1,X,D,D'. The values D and D' are the newly introduced logic values to implement the proposed D-algorithm. The value of D(D') will be represent the value of the node being testing in the circuit. It will be 0(1) for good circuit and 1(0) for bad circuit. Thus a value D (D') placed on a node a, during test generation will place 0(1) on the node for good circuit simulation and 1(0) for the bad circuit simulation. The crux of this algorithm is of setting this value on the testable nodes and propagating this to the output.

A terminology named singular cube was also introduced in this method. The singular cube of the Boolean function is defined as an assignment $(x_1...x_n, y_1...y_m) = (l_1, l_2, ..., l_{m+n})$. where x_i are inputs, y_i are outputs and $l_i \in$ { 0,1,X }. Also the fault model for the D-algorithm is called the Primitive D-Cube of Failure (PDCF). PDCF is defined as the set of logic values on the input and output of a gate that will prove the fault on its output. The steps involved in the D-algorithm are defined below. First the fault for which the test has to be generated is selected from the fault list. Then the PDCF for the fault is generated. Then it is checked whether there is D or D' on the primary output after applying the PDCF.

a. If there is a D or D' in the primary output: If there is a D or D' then it is checked whether there are more lines to justify. If there are no lines to justify, then the pattern is stored as the test for the fault. If there are more lines to justify, a line should be selected to justify all other lines. If there is no inconsistency, then further it is checked for any other lines to justify. If there is an inconsistency, availability of an alternative path is searched for justification. If an alternative path is found then, it is checked whether there are more lines to justify otherwise backtrack one level and select another path. While backtracking it is checked whether the node is revisited or not. If the node is already visited, then it is reported that no pattern exits. If it is not a revisited node then the same steps of finding more lines to justify are carried out. This forms one branch of the whole D-algorithm process, when the D or D' is found on the primary output after applying the PDCF.

```
D-alg()
begin
    if Imply_and_check() = FAILURE then return FAILURE
    if (error not at PO) then
      begin
        if D-frontier = ∅ then return FAILURE
        repeat
          begin
            select an untried gate (G) from D-frontier
            c = controlling value of G
            assign c̄ to every input of G with value x
            if D-alg() = SUCCESS then return SUCCESS
          end
        until all gates from D-frontier have been tried
        return FAILURE
      end
    /* error propagated to a PO */
    if J-frontier = ∅ then return SUCCESS
    select a gate (G) from the J-frontier
    c = controlling value of G
    repeat
      begin
        select an input (j) of G with value x
        assign c to j
        if D-alg() = SUCCESS then return SUCCESS
        assign c̄ to j   /* reverse decision */
      end
    until all inputs of G are specified
    return FAILURE
end
```

Figure 2.7: D-Algorithm [[MAF90]]

b. If there is no D or D' in the primary output:

If there is no D or D', the D cube is propagated and intersected. If there is any inconsistency, check for an alternative gate for propagation. If the alternative gate is found, then follow the same steps of propagating the D-cube. If the alternative gate is not found, then backtrack one level and select another path. If the PDCF is reached then it is reported that the pattern does not

exist otherwise the process of propagating the D-cube is continued. But during the first step of propagation the D-cube, if inconsistency is not found then the lines to be justified are marked and go back to the step of finding whether the D or D' is present in the primary output.

Thus by looping through the above two decision branches, either the test will be generated if it exists or else it will be reported that there is no pattern for the fault. Once it reaches this decision, then the algorithm again loops through this process for the next fault. Thus the test for all the faults in the circuit will be generated using the D-algorithm.

2.2.5.2 PODEM

The pseudo code for the PODEM(Path-Oriented-Decision-Making)[[Goe81]] algorithm is shown in Fig.2.8[[MAF90]]. This algorithm is straight forward compared to the D-algorithm. This algorithm generates test pattern for the target fault in the circuit by implicit enumeration of all possible input vectors to the primary inputs of the circuit. The assignment of the input values is carried out by constructing the search tree for each input line of the circuit by setting values of either 0 or 1 and checking the implication of setting them. The detailed steps in this algorithm are briefed below.

First step involves selecting fault from the fault list for which the test has to be generated. Initially, the value X is assigned to all the inputs. Select a primary input from the list of primary inputs of the circuit. Assign a binary value to that input and determine implications of all other inputs and other nodes due to this assignment. Check whether there is a D or D' found in the primary outputs. If there is a D or D' then, store the input pattern value as the test for the target fault. If there is no D or D', then check whether the test is possible by assigning values for more inputs. If possible, then start the step of assigning the binary value to the new input from the list of primary inputs that are not assigned values. Continue this process until all the primary inputs are exhausted. If still the test is not found, check whether there is any unassigned input pattern combination. If so, then go to the step of determining the implications of that pattern over the other nodes. In other case if all the combinations of the patterns are tried, then report that there exists no test pattern for this fault.

The main advantage of this method is that, the number of backtracking taking place in D-algorithm is considerably reduced and thus it speeds up the test pattern search. The method for assigning the binary values to the primary inputs is carried out by constructing the search tree over the list of inputs along with the process of checking the implication.

```
PODEM()
begin
    if (error at PO) then return SUCCESS
    if (test not possible) then return FAILURE
    (k,v_k) = Objective()
    (j,v_j) = Backtrace(k,v_k)  /* j is a PI */
    Imply (j,v_j)
    if PODEM() = SUCCESS then return SUCCESS
    /* reverse decision */
    Imply (j,v̄_j)
    if PODEM() = SUCCESS then return SUCCESS
    Imply (j,x)
    return FAILURE
end
```

Figure 2.8: PODEM Algorithm [[MAF90]]

2.2.5.3 FAN

The pseudo code for the FAN [[Fuj85]](FAN-out-oriented) algorithm is shown in Fig.2.9 [[MAF90]]. The exhaustive searching of all the input combination patterns in PODEM algorithm is avoided in this algorithm. This speeds up the search for the test pattern considerably. The strategies used in the FAN algorithm are

- At each step of the enumeration, as many signal values that are uniquely implied are determined.
- assign the value D or D' that is uniquely determined or implied by the target fault
- When the D-frontier has a single gate,apply a unique sensitization
- stop the backtracking at a headline, and postpone the line justification for the headline later
- multiple backtracking is more efficient that single path backtracking
- in the multiple backtrack, if an objective at the fanout point has a contradictory requirement, then stop at the backtrack so as to assign a binary value to the fanout point.

Applying these strategies, the test search time of the FAN algorithm was considerably reduced. The multiple backtracking and the justification and implication on either direction enhances the test pattern finding capability of the algorithm.

```
FAN()
begin
    if Imply_and_check() = FAILURE then return FAILURE
    if (error at PO and all bound lines are justified) then
      begin
          justify all unjustified head lines
          return SUCCESS
      end
    if (error not at PO and D-frontier = ∅) then return FAILURE
    /* initialize objectives */
    add every unjustified bound line to Current_objectives
    select one gate (G) from the D-frontier
    c = controlling value of G
    for every input (j) of G with value x
       add (j,c̄) to Current_objectives
    /* multiple backtrace */
    (i,v_i) = Mbacktrace(Current_objectives)
    Assign(i,v_i)
    if FAN() = SUCCESS then return SUCCESS
    Assign(i,v̄_i)  /* reverse decision */
    if FAN() = SUCCESS then return SUCCESS
    Assign(i,x)
    return FAILURE
end
```

Figure 2.9: FAN Algorithm[[MAF90]]

2.2.6 Scan Design

The full scan architecture with test control and scan in/out pins are shown in fig.2.10. The flip-flops SFF1, SFF2 and SFF3 form the scan flip-flop group and they form the scan chain. The circuit is designed using the predefined design rules. A test control input pin is added to the design to control the scan flip-flops operation in normal and test mode. Scan chain originates from the scan-in pin and terminates at the scan out pin. The DUT (combinational block) will be operated in its usual mode through the primary inputs (PI) and primary outputs (PO).

Fig.2.11 gives the design flow for the automated scan design. Behaviour, RTL and logic design is synthesized to gate level net list. Design and Test data generation for manufacturing involves two parallel flows, where, the scan insertion is carried out at one branch and combinational ATPG is carried out at the other. In the scan insertion flow, the scan net list is inserted to the

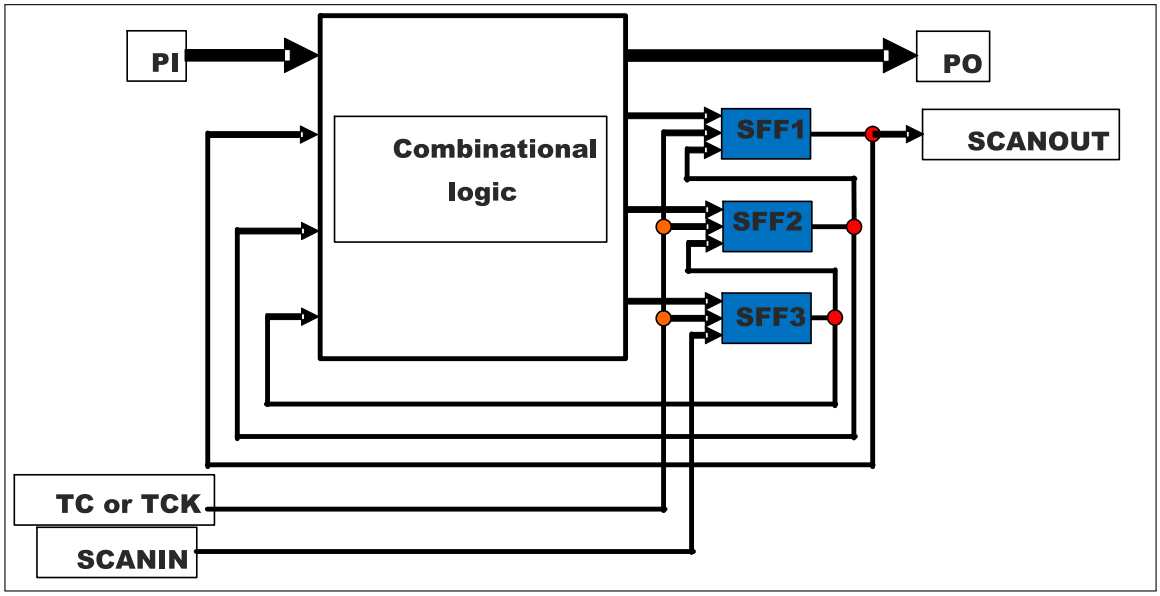


Figure 2.10: Full Scan Architecture [MV00]

gate-level net list. Scan chain optimization and timing verification are carried out in the next level.

Then the mask data is produced which will be tested with the test program to generate the test data for manufacture. In the ATPG flow, the gate level net list without the scan designs are evaluated with combinational ATPG for the combinational test vectors. From these vectors and the scan chain order obtained from the scan chain optimization, the test program with scan sequences is generated. Finally, test data along with the design data will be generated from the test program and the mask data which is available for manufacturing.

2.2.7 Partial Scan Design

To minimize the overhead caused by the full scan design, partial scan design was introduced. In partial scan design, only minimal set of flip-flops are selected for scan to eliminate all cycles. Sometimes, to keep the overhead low, only long cycles may be eliminated. In cycles with self-loops, all cycles other than self loops may be eliminated.

Fig.2.12 shows typical partial scan architecture. The flip-flops F1 and F2 forms the non- scan flip-flop group. The flip-flops SF1 and SF2 form the scan flip-flop group.

Test Generation: For a partial scan design, separate clocks are used for scan flip-flops and non-scan flip-flops. Alternatively, separate design can be used for scan flip flops, which will require only one clock signal.

Cyclic to acyclic conversion of the circuit should be preformed for the effective test generation

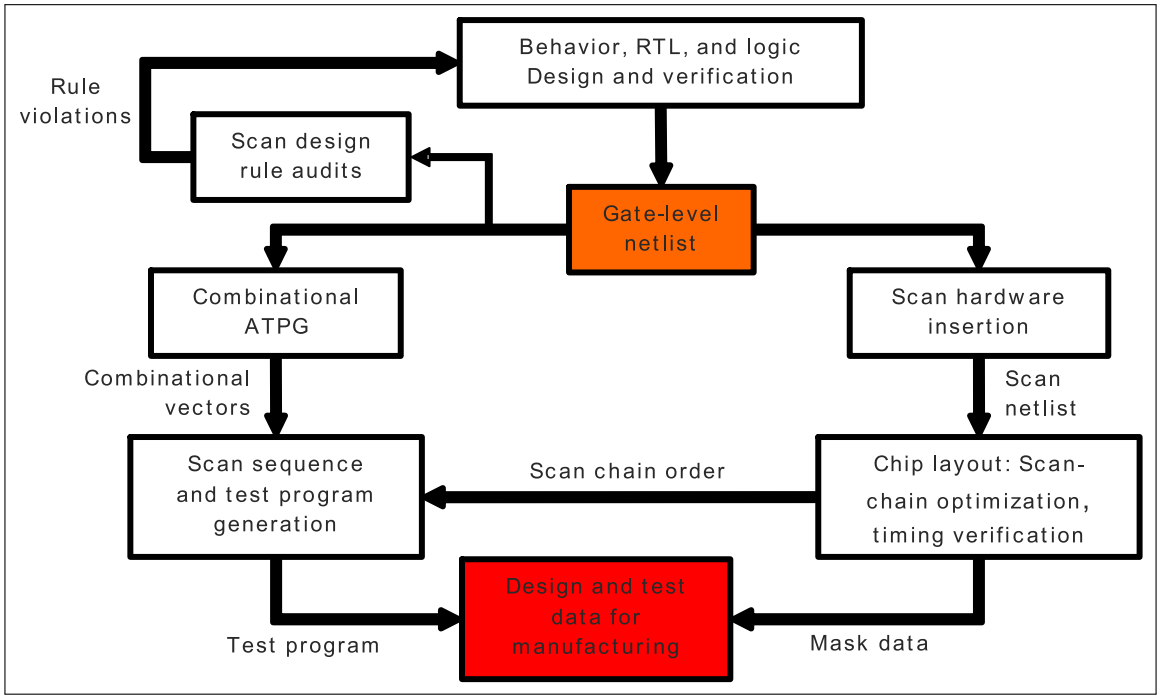


Figure 2.11: Automated Scan Design [MV00]

of asynchronous circuits using a synchronous sequential test generation CAD tool. The conversion removes all the feedback loops formed in the cyclic circuit which eases the test generation capability of the CAD tool. For instance, the original or actual cyclic asynchronous circuit fed to tool will result in low fault coverage as the tool discards most of the fault sites present in the path of the feedback loop. So the cyclic to acyclic conversion will increase the visibility of the fault sites to the tool to generate test patterns.

2.3 Summary

A brief introduction to asynchronous design was given in this chapter. Different gate and delay models of asynchronous circuits were briefed. Signaling protocols of these designs were introduced. Logic synthesis design flows for asynchronous circuit design were also described. Next a brief introduction to the testing and test generation principles were introduced. Several basic terminologies involved in testing were listed. Several Fault models used for testing circuits were also discussed. The topic of automatic test pattern generation and scan design techniques were briefed in detail. Several ATPG algorithms were described by showing the pseudo code of the algorithms. Two types of DFT scan architectures namely Full scan and Partial Scan design were introduced and briefed in detail, which will be used extensively in this thesis. The next chapter provides a comprehensive literature review over works related to the testing of

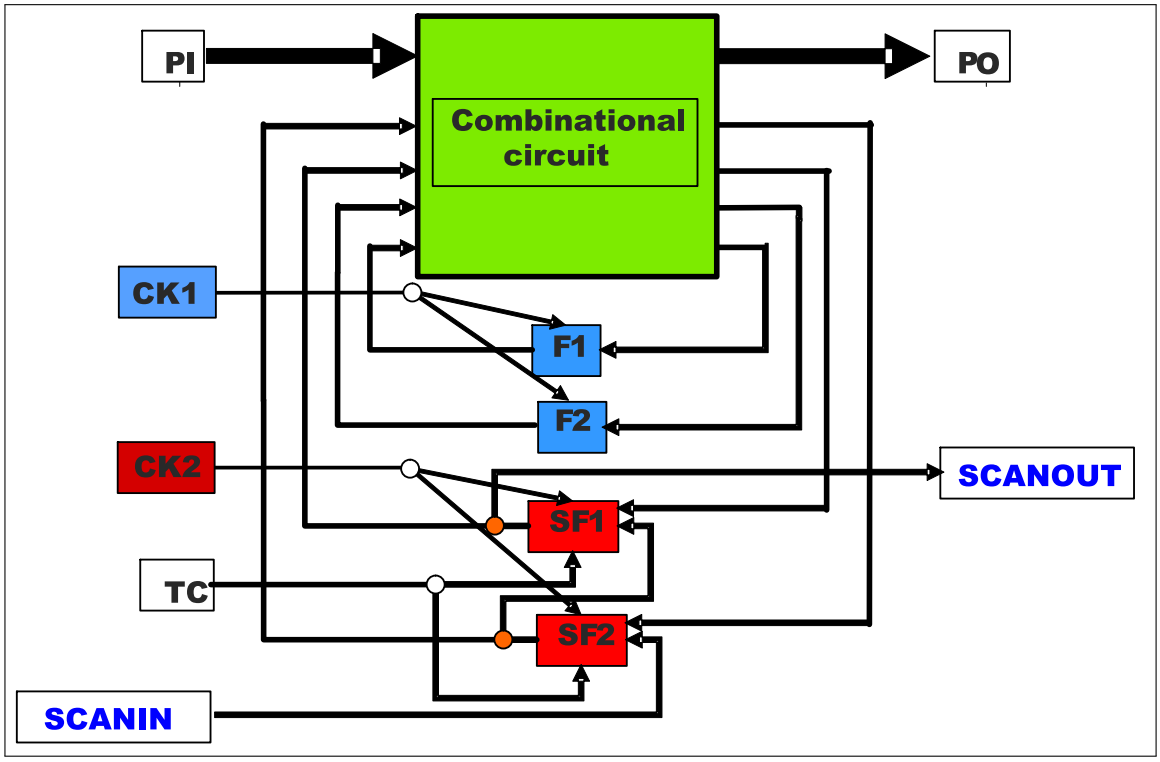


Figure 2.12: Partial Scan Architecture [MV00]

asynchronous designs.

Chapter 3

Related Work

3.1 Introduction

This chapter briefs the literature review over the related works involving the design for testability(DFT) and test generation of asynchronous circuits. The literature review in this chapter includes the topics 1) Design for test (DFT) for asynchronous circuits, 2) ATPG methods for asynchronous circuits, 3) Self checking designs of asynchronous circuits, 4) Testable asynchronous circuit design,5)Test Generation at defect/transistor level of asynchronous circuits. A detailed survey on testing asynchronous circuit was elaborated in [HBB94]. An introduction to defects in the circuit and the terminologies and method for testing are studied. A detailed description of self checking circuits was also given in a complete section. Self checking property for the delay-insensitive asynchronous circuits and speed independent circuits were briefed. Several conventional test generation methods were listed and the algorithm behind the methods were analyzed. Topics on Automatic test pattern generation and fault simulation were described using a specific example. Topics on design for test (DFT) were also briefed in detail. The topics on testability, controllability, observability were tutored. The conventional full scan path design techniques were detailed using a specific example. An example based on a n-bit asynchronous counter design was demonstrated. Finally delay fault testing method was described. Delay model used here was path delay fault model. The delay fault test procedure was demonstrated using the circuit equivalent to a majority gate circuit with inversion on each of its AND gates. Thus a broad coverage of all the topics in testing was provided with respect to the testing of asynchronous circuits.

3.2 Related Work

The detailed flow of the asynchronous circuit testing topics covered in this review is shown in the Fig. 3.1.

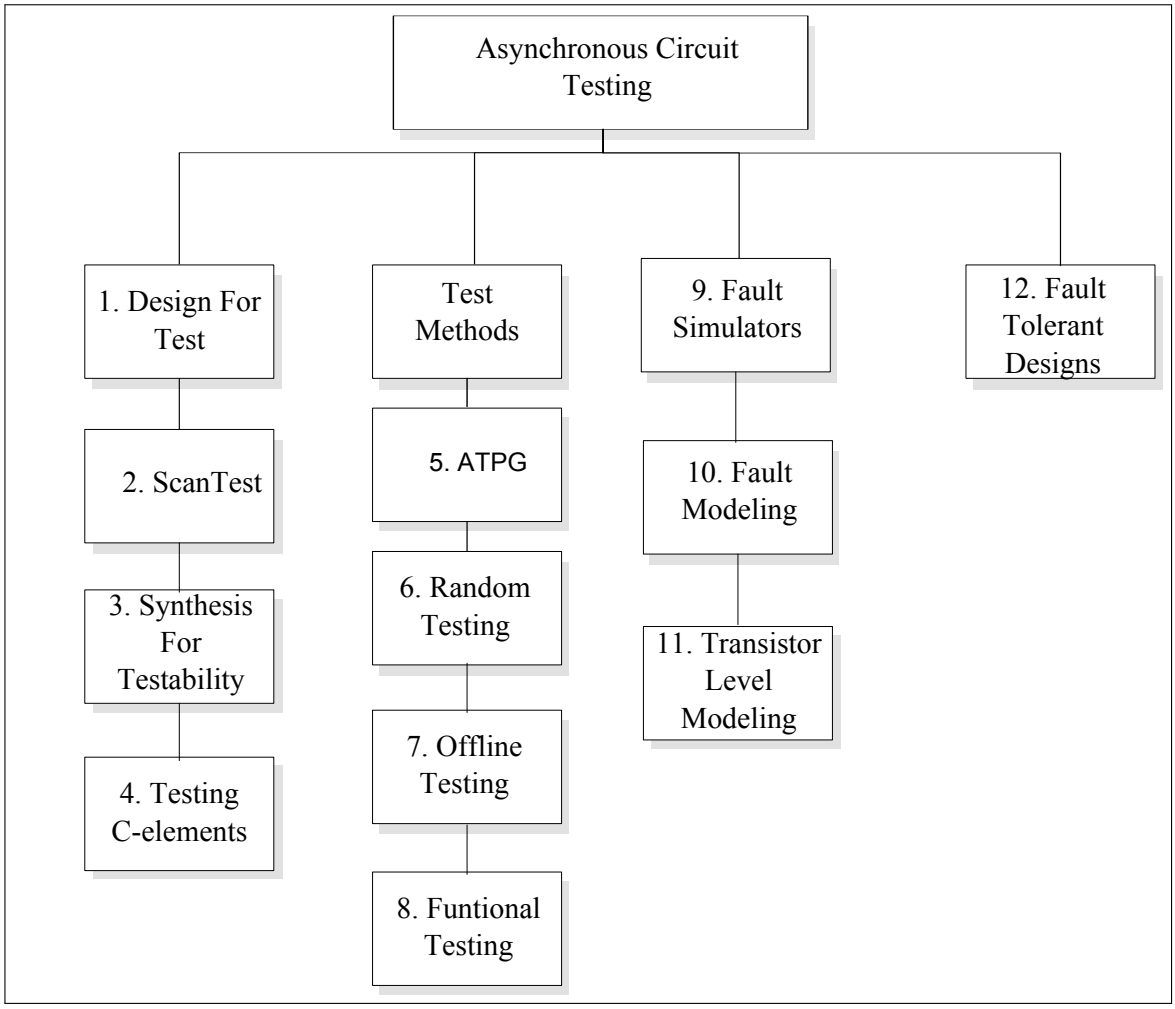


Figure 3.1: Asynchronous Circuit Testing - A Short Review

3.2.1 Design For Test

1. **Designing C-elements for testability** [PF95a]
   The designs of static symmetric and asymmetric C-elements that are testable for stuck-at faults and transistor level stuck-open faults was proposed in the work. Several CMOS implementation of these C-element designs were proposed. Also C-element design with scan features which aid the scan testable designs was proposed. Designs of conventional 2 transistors based inverter and a testable 4 transistors based inverter were examined in detail for the stuck-at and stuck-open faults. All the possible faults, the corresponding

circuit response for faulty and good circuit and the test sequences for the same were tabulated.

Next the design of static symmetric C-element was studied. Three different implementations were studied. All the good and faulty circuit responses for all the stuck-open faults in the symmetric C-element were analyzed then and the corresponding test sequences were generated. Next the design was examined for all the stuck-at faults. Totally 38 stuck-at faults were reported along with their test patterns to detect the faults. Next the asymmetric C-element design style was studied. The circuit comprising of 8 transistors was studied. Two different designs of asymmetric C-element were studied particularly. They were OR-AND type asymmetric C-element and AND-OR type asymmetric C-element. The stuck-open faults on these designs were first analyzed. It was reported that only 5 test sequences are enough to test all the stuck-open faults. The stuck-at fault diagnosis was carried further. Totally 32 stuck-at faults were reported and their corresponding fault response and the test sequences were tabulated. Analysis was made in the design implementation in 1 μm, double layer metal CMOS process and simulated using SPICE analysis.

Further, the scan testable designs of C-elements were introduced. A pseudo-static symmetric C-element with scan features was proposed first. The operating modes of the design were briefed further which involved normal mode, test mode and scan mode. The stuck-open faults and their corresponding output response on the circuit and the test sequence to detect them were described in detail. Finally the cost comparison of the testable C-elements were made with respect to the number of transistors, number of pins, area overhead, output nodal capacitance and the testability were made. Transistor overhead from 17 percent to 200 percent was reported over six designs. Area overhead of 17 percent to 115 percent was reported for the same six designs. Output nodal capacitance was ranging 2 to 11fF. Two of the six designs were reported to be stuck-at and stuck-open fault testable.

2. **Asynchronous Sequential machines designed for fault detection** [SM74]

   Design of asynchronous sequential machines which can allow detection of faults in them was introduced in this work. The circuits designed are assumed to be operating in the fundamental mode. The operation of the circuits was described based on the flow table. The definition of flow table is also introduced here. Definitions on transition pair, partition pi were defined which was later used in the description of the machine design. Definitions on internal states and proper stable states were introduced. A method for detection of internal state fault was briefed further. An additional equivalence class called fault equivalent class was introduced to facilitate the design of fault detecting asynchronous circuits. Stuck at 0 and stuck at 1 faults were considered here.

Two main conditions were introduced which should be met while designing the asynchronous sequential circuit such that it is fault detecting. First condition is that the equivalence class of different transition paths for a given input "I" must have at least a distance of two. The second condition is that for a single fault, the circuit must become stable in the equivalence class of the transition path or a fault equivalence class. Due to the static nature of the method proposed, the same technique can be extended to the faults other than stuck -at- 1 and stuck-at-0. The whole method is based on developing k-sets from the flow table of the given circuit. Then by applying a reduction rule and assigning a variable value of 1 for each state in the set and a value of 0 for the states not in the set. This produces a partition in such a way that the first condition for designing the circuit is satisfied. Thus given a flow table of the circuit, a five step procedure yields the design equation for the fault detecting equivalent of the asynchronous circuit. Finally a bound on the amount of logic required for designing the fault detecting circuit was derived. The upper bound on the number of gate inputs for fault detecting realizations is less than or equal to $\sum h_n + d_s(m+1) + m_t$, $1 \leq n \leq d_s$. Where d_s is the number of distinct non trivial k-sets after the list has been reduced, h_n is the number of stable states that are contained in the nth k-sets, m is the number of input variables, t is the number of trivial input columns.

3. **A TestableCMOS Asynchronous Counter** [CB90]

An asynchronous counter design was introduced in this work along with a DFT (Design for Test) logic inserted to make the counter testable. The counter was designed based on the two cycle transaction(transition signaling) method. The counter was composed of n identical two-cycle toggle modules and an XOR gate. Two designs, the asynchronous toggle module and asynchronous toggle module with inverter were introduced. A new asynchronous toggle module with scan capability was designed to facilitate the testability of the counter design. Test method for the counter for testing the stuck-at and stuck open faults was also introduced. Four test procedures namely toggle test, shift test, cycle test and XOR test were introduced which has to be carried out to completely test the counter design. It was shown that the asynchronous counter testing time was O(n) which is less compared to the synchronous counter testing time $O(n^2)$. The reduction in time was attributed to the parallel testing of the cells in the counter due to the presence of the scan path and two bit of state in each cell. The base counter design and the testable asynchronous counter design layout were presented. The two counters were 16 bit designs and were fabricated in 2 micron process. The experimental results of the chip were given. The performance of the base design was with a count rate of 21.0 MHZ and

that of the testable design was 22.6 MHZ. The area overhead was 6 % compared the base asynchronous design and 15% compared to the equivalent synchronous design.

4. **DFT for Fast Testing of self timed control circuits** [PKB95]
   A design for test method for fast testing of self timed control circuits was proposed in this work. The circuits used for the testing are compiled by a custom compiler namely OCCAM based circuit compiler. The OCCAM program description is converted in to an interconnection of pre-compiled self-timed-macro-modules/library and the test method was developed for the resulting circuits. The synthesis method for the OCCAM based program description in to self-timed circuits was briefed further. The translation is syntax directed. This method involves testing all the control paths simultaneously, which in turn means that all the paths in the design under test are excited concurrently. Four basic requirements for the testing method to be applicable were listed. They include 1) At a branching point all the branches should be activated, 2)When two branches are merged through a Merge element a single event should be produced at the output of the Merge element after both the branches finish their processing, 3) When the sharing of resources occur, it has to be guaranteed that progress on one control path is not stopped because of the progress in another control path, and 4) The control path should be decoupled from the data path during testing so that the control path can be tested separately.
   Certain modifications were done in the pre-compiled modules to satisfy the above mentioned 4 requirements for testing the circuits built using these modules. Modifications are done to three modules namely XOR, select and Call. Also modifications are done to the OCCAM program constructs to fulfill the requirement for testing. IF, LOOP and ALT constructs were modified and an example showing this modification for ALT construct was demonstrated.
   To demonstrate this method, the control path of the self-timed circuit to implement the GCD of two numbers was experimented. Faster testing time compared to another method involving scan testing was reported. Low testing time, no need for test vectors, and possibility of extension to other asynchronous circuit styles and automation are reported as the advantages of this method. Area overhead for the DFT comes from replacing the XORs with XOR/Celements and a performance degradation of 15 percent was reported. But it was justified with the percentage of area the control circuits take in an overall chip layout.

3.2.2 Scan Testing

1. **Scan Testing of asynchronous sequential circuits** [PF95b]
   A new method for testing the asynchronous sequential circuit based on micropipelines

was introduced in this work. The test method is based on Scan DFT method. Both the stuck-at faults and the delay faults were considered during the testing process. The basic structure is composed of a combinational logic block, registers in the feedback loop storing the state of the circuit, two C-elements and a delay element. The circuit has primary inputs(PI), primary outputs (PO), secondary input(SI) and secondary outputs (SO) along with the request signals Rin and Rout and the control acknowledge signals Ain and Aout. At the initial state, the registers and the two C-elements are set to zero. First the input data on the primary inputs are generated by the sender which activates the Rin signal in the circuit. The request signal is delayed long enough to stabilize the circuit with output data on the primary and internal/secondary outputs. The delay is facilitated by the delay element between the C-element of the request signal and the signal Rout. Once the circuit is stabilized, the Rout signal is activated for the receiver by the circuit. Also after receiving the acknowledge signal (Aout) and storing the new state in register 2, the circuit activates the acknowledge signal (Ain) to the sender. Thus this procedure of processing data is repeated with the repeated reception/activation of the Rin signal.

Three types of stuck-at faults were distinguished for the micropipelines namely 1) faults in the control part of the micropipelines, 2) faults in logic blocks, and 3) faults in the latches. A scan test approach for testing these faults was next introduced. The CMOS implementation of the scan latch structure was introduced. The performance of the proposed scan latch was compared with the basic latch design in terms of delay using SPICE simulations. The delay was basic design and proposed scan design were reported as 3.7ns and 6.2 ns respectively. Next a two-bit scan register design was proposed based on the scan latch. The register operates in three modes namely, normal mode, test mode and scan mode. Using the modules complete testable asynchronous sequential circuit design was demonstrated. The testable design is composed of two blocks namely the actual circuit under test and the scan testable control logic (STCL). The STCL block proposed is fully testable for stuck-at faults because of its asynchronous delay insensitive nature. A complete test strategy for the testable design to test the faults in control logic, combinational block and the latches were briefed in detail.

Next the path delay fault testability of the design was briefed. It is based on the well known path delay fault testability method for combinational circuits. Basically the test pattern pair applied to the input of the combination logic module detects the delay faults in the paths of the block. This test method involves loading the state vectors for the registers in the circuit and then the test vectors to the input of the circuit and monitoring the output signals of the circuit under test. In detail, the circuit is operated in test mode, to apply the test pattern p1 to the inputs along with generating a request signal on the input Ri. After receiving the acknowledgement event on the signal Aout the test p2 is applied

to the input. The test control signal is set to zero and request event is generated in the signal Ri. This results in the data path of the combinational block being activated. If there is a delay fault in the path, it will result in a delayed response of the circuit ,which aids the detection of the delay fault in these circuits. Thus a scan testable method for both the stuck-at and path delay faults was introduced in this method.

2. **Optimal scan for pipelined testing:an asynchronous foundation** [RAV96]

   A method for constructing optimal scan chain was proposed in this work. The objective of the optimal scan chain construction was to 1) reduce the area overhead for latch based design and 2) reduction of the number of pipeline scan shifts. The difficulties encountered in the scan testing and pipelined scan testing and standard LSSD(Level Sensitive Scan Design) based testing were detailed further.
   Several pipelined scan testing types namely 1) simple sequential scan, 2) smart sequential scan, 3) simple parallel scan, and smart parallel scan were described in detail. Then the issues in LSSD based scan testing were discussed in detail.
   Next the heuristic for the optimal scan chain construction was introduced. The main objectives of the algorithm were 1) to keep the L1/L2 partition for the scan latches balanced and to keep the scan latches for each datapath close together. The first objective aims at reducing the area overhead while the first objective aims at reducing the scan shift time.
   The algorithm was applied to three industrial asynchronous circuits and the results were reported in terms of the scan shift time reduction, number of tests, and scan latch shifts.The size of the circuits was randing from 20k to 45 k transistors with the number of data latches ranging from 417 to 1083 and the latch classes ranging from 30 to 101. The pipelined LSSD based scan testing method was reported to have reduced the number of scan shifts to around 60-75 percent and the pipelined L1L2* based testing with the reduction percentage of 79-86 percent was reported. For all the three designs the number of shifts needed for the pipelined L1L2* was reported to be about half the number needed for the pipelined LSSD.
   . The formal justification of these two problems to construct the optimal scan chain was briefed further. First it was proved that the optimization problem for L1L2* relates to the area overhead minimization. Next it was proved that the problem of constructing the feasible scan chain by adding dummy latches was to minimize the total scan shift time. Experimental results on three industrial asynchronous IC designs were shown as (1) less than 0.1% extra scan latches for Level-Sensitive Scan Design, and (2) scan shift reductions up to 86% over traditional scan schemes.

3. **Partial Scan Test for asynchronous circuits illustrated on a DCC Error Corrector**

[Ron94]

A new design for testability method for testing asynchronous circuits using partial scan was proposed in this work. Before introducing the method a gentle introduction to the VLSI programming in Tangram was given. An example of Tangram procedure named scanin was illustrated. The compilation of the Tangram programs in to asynchronous circuits via an intermediate representation called handshake circuits was detailed further. Next the asynchronous circuit implementation focusing the design of Digital Compact Cassette (DCC) Error corrector was discussed. For the implementation, four-phase handshake signaling and double-rail data encoding was used.

Detailed description of the DCC error corrector architecture was briefed further. The architecture is composed of a DRAM, controller and a detector. Testing the controller using partial scan method forms the crux of this work. Further the design and test aspects of the detector and the controller were discussed in detail. Next the Tangram programming procedure for the partial scan design of I2S transfer procedure was proposed. The scan facility was added in to control the value in the DRAM address counter. The equivalent handshake circuit compiled for this procedure was also demonstrated. The Test performance, circuit performance cost, reliability and the test solutions for the diagnostics and detection transferrers were discussed further. The scan test for the I2S transferrer was reported to be 19 DRAM addressing cycles which was 1800 times less than that of the design without the scan architecture. An area overhead of 3 percent for the scan circuitry was reported for the design analysed. A fault coverage of 99.9 percent was reported for the scan design proposed on the circuit under test.

3.2.3 Synthesis For Testability

1. **Synthesis of asynchronous circuits for stuck at and robust path delay fault testability** [NJC95]

   A method for synthesis of multi-level asynchronous circuit with the hazard free property and also completely testable was introduced in this work. Stuck-at and robust path delay are focused for this testing method. A minimization algorithm for the synthesis of hazard-free two level implementation of asynchronous circuits was first introduced. First steps for minimizing the non-primes were introduced with an algorithm named npni-row-dominate. Then the methods for minimizing the redundant covers were briefed. An algorithm named rni-row-dominate-unopt was introduced for the same.

   Then a method for converting the hazard-free two level implementation in to completely testable multi-level implementation was introduced. Four different procedures based on the type of two-level logic was detailed with example. The first procedure which stars

with non-Prime and irredundant Two-level logic was described. Then the second procedure starting with the redundant but prime two-level logic was introduced. The third procedure starts with the redundant and prime two-level logic. In the fourth procedure, the two level logic which is irredundant and prime was processed to convert them in to multi-level testable logic was described.

Experimental results for several benchmark circuits were reported.stuck-at and robust path delay testability of 100 percent with pin overhead of zero or few was reported.

2. **Synthesis of testability techniques for asynchronous circuits** [KLSV91]

   A logic synthesis method for asynchronous circuits without hazard and to detect path delay faults in them is proposed in this work. Two types of gate delay fault models were used for the path delay fault testing namely hazard-free robust path delay fault (hfrpdft) and robust gate delay fault (rgdft) models. A method for removing the hazards from the initial two level implementation of the circuit was introduced. A method for generating a guaranteed hfrpdft circuit was proposed. The crux of this method was to choose a binate variable x, in a given SOP representation S, of a Boolean function f, decompose in to x.G + x'.H + R, in such a way that variable x does not appear in G,H and R. The method implements area efficient design which is a hfrpdft. Another heuristic procedure to further improve the testability of the hfpdft was also proposed. This heuristic uses algebraic factorization to improve the delay fault testability of the circuit. Next a procedure to guarantee the generation of rgdft circuit was proposed. This method requires test inputs to make it robustly path delay testable.

3.2.4 Testing C-element

1. **Testing C-elements is not elementary** [BR95]

   This work analyses several designs of C-element for stuck-at fault testability. Interesting facts on the effect on the functionality of the circuit by the stuck-at faults in the C-element designs were analyzed. Totally 7 different C-element design implementations were taken into account and the testability of these designs for stuck-at faults were discussed further. First the majority gate implementation of the C-element was introduced and all the stuck-at faults in this design and the possible functional behaviours of the C-element due to these faults were tabularized. Test patterns for each faults were also derived in one of the columns of the table. Interestingly at most 2 test patterns were needed for testing all the detectable single stuck-at and multiple stuck-at fault models. From the analysis of the circuit, a guaranteed time of stabilization for the circuit given the circuit at any stable state and the new input value was derived as d-max = max $\{d_1, d_2, d_3\} + d_4 + dassumingd > max\{d_2, d_3\}$, where di is the transition delay of the gate i.

The majority gate element is composed of the AND gates g1,g2,g3 and the OR gate g4 and hence the corresponding transition delays d1 to d4. Totally 18 different faulty machines were derived for all the fault in the C-element. For the Wuu's circuit a test length of 7 was reported and 38.5 percent of the single stuck-at faults were reported not to result in a halting state. For Koche and Brunvand's circuit also required 7 vectors and 50 percent of the faults does not result in halt state. For Mayevsky's circuit, 7 test vectors are required and 20 percent of the faults does not result in halt state. Bartky's circuit again needed 7 test vectors but only 14.7 percent of the single faults results in circuits that does not halt. For the dynamic implementation, the test vectors needed were 7 and 6 out of 14 fault machines do not result in halt state. The static implementation has six additional transistor compared to dynamic one. For the asymmetric C-element 14 out 18 faults does not result in the halt state and it requires test length of only 4. Thus this work concludes that detection of faults in C-elements is not trivial and the testability properties are recommended to be considered during the design of the C-elements.

3.2.5 Test Pattern Generation for Asynchronous Circuits

1. **Heuristic for testing asynchronous circuits -** [Put70]
   A heuristic algorithm for automatic test pattern generation for asynchronous circuits was introduced in this work. This work is the most earliest work reported on ATPG algorithm for asynchronous circuits. The algorithm was implemented as an APL program. This method reads in the circuit netlist to be tested as a combinational asynchronous network which has feedback loops present in it. The test generation algorithm is a heuristic and thereby the test for the circuit is not guaranteed. This method involves two steps . In the first step the test pattern or vector is generated for the CUT with a given fault. Then the generated test vector is simulated for both the good and faulty machine to validate the test. Also in this method, given asynchronous sequential circuit is considered as an iterative design of combinational blocks. In detail, when an asynchronous sequential circuit S with primary inputs PI_1, PI_2....Pin and primary outputs $PO_1, PO_2 ... PO_n$ with "n" feedbacks is given as input, the heuristic finds the points in S where the feedback loops will be cut to convert S in to an acyclic circuits.
   Also, when delay elements are added in these cut sites, the circuit will operate as the original circuit in functionality. Once the cut points are selected, pseudo inputs $SI_1..SI_m$ are inserted for the m selected cut points. A Strongly Connected Component (SCC) based loop cutting procedure was employed to cut the feedback loops. Intrinsic weights and weights for each lines are introduced along with finding the SCCs to cut the optimal feedback loops. Once the acyclic circuits are obtained by cutting the loops, the modified

D-algorithm is applied to generate test patterns for the circuit. While generating the tests, some restrictions are applied due to the presence of pseudo outputs and pseudo inputs present in the modified circuits. A working example of potential test generation and fault simulation were shown. The whole heuristic was implemented in APL program namely CIRCUIT, CUT and GENERATE. Circuits with size of 50 blocks were used as CUT and it took 25 to 130s. It was concluded that the test generation problem execution time is not dependent on the circuit size but only on the topology of the circuit.

2. **Boolean Difference for fault detection in asynchronous sequential circuits** [HC71] Major reasons for difficulty in detecting faults in asynchronous sequential networks were outlined in this work. The four main reasons were 1) presence of feedback loops makes the test patterns order/time dependent, 2) The machine must be kept stable to apply the test patterns,3) an exact model of asynchronous sequential circuit is difficult to obtain and 4) large amount of information needed to execute the test being infeasible for large circuits. A Test pattern generation for asynchronous sequential networks based on Boolean difference is introduced in this work. The asynchronous sequential machine Huffman model was used. Several definitions namely stable state, set state, homing sequence, Boolean difference, Boolean difference chain and total states were used to define the test generation methodology. The test generation algorithm is based on describing the asynchronous circuit as a set of Boolean equations. Then a primary input is chosen from the set of inputs of the machine and a sensitizing path is found between the selected input and the selected output. Homing sequence is used to facilitate the test generation process by moving the machine to known stable state. Two requirements namely stability and compatibility have to be satisfied to generate test successfully using this method. An example of a gated latch was demonstrated to show the applicability of this method. It has been summarized that the feedback variable assignment and the homing sequence generation algorithms were not discussed in this work.

3. **Test pattern generation for circuits with asynchronous signals based on scan** [TF96] A constrained test pattern generation method was introduced in this work for scan testing circuits with asynchronous signals. The test patterns generated using this method were guaranteed to be valid even when a hazard occurs. Two different classifications of the scan register were first introduced, namely concurrent capture-update (C-C-U) class and the separate capture-update (S-C-U) class. The classification is based on the timing of the capture and update operations during the scan testing process. In the C-C-U class the update operation occurs right after the capture operation. A latch and an edge triggered flip-flop are examples of this class. In the S-C-U class, the timing of the update is separated from that of the capture. LSSD double latch is an example of this class.

Next two problems were discussed: 1) destruction of scan chain values by a capture clock and 2) destruction of the scan-in value by a hazard. The C-C-U class is more vulnerable to the first problem. A detailed example for these two problems was demonstrated. To solve these problems dynamic constraints were proposed. The features of these constraints were 1) make a decision that scan-in values are not to be destroyed by the capture clock, 2) justify a value of the D input so as not to destroy the scan-in values and 3) propagate uncontrollable value where a hazard is possible.
The dynamic constraints for the asynchronous faults and synchronous faults were proposed separately. The constraints for the C-C-U scan register that feeds the asynchronous inputs of registers or the control inputs of the tri-state devices are 1) when a scan-in value of a register is determined, justify the same value on the D input of the register and 2) when a value of the D input of the register is implied, make a decision to have the same value on the register. For the C-C-U scan register that feeds the D input of a register driven by the same clock, the constraints to be satisfied are 1) when a scan-in value of a register is determined, justify the same value on the D input of the register and 2) when a value of the D input of a register is implied, make a decision to have the same value on the register. For asynchronous faults, the following constraint was proposed. For every register that feeds asynchronous inputs of registers or the control inputs of tri-state devices, the constraint to be satisfied is that when a value of a register is destroyed by an activated asynchronous signal, propagate the uncontrollable value U from the output of the register.

An ATPG procedure for this test method was described further. The dynamic constraints were applied to both the decision process and the implication process which forms the main part of the ATPG flow. The justification or decision process was executed based on the result of the constraint checking process. When there is conflict between the constraints and the existing values of the circuit, backtrack is performed. Experiments were carried out on real chips for communication systems. Totally 5 chips were experimented and the results showing the number of gates, number of equivalent faults, the number of scan registers and the ratio of synchronous and asynchronous faults to the total number of faults in the design were reported. Faults ranging from 89 to 96 percent for synchronous faults and 4 to 11 percent for the asynchronous faults were reported as the characteristics of the chips being experimented. The resulting fault coverage for all the chips was reported to be in the range of 97 to 99 percent. The test execution time was reported to be between 355 to 11458 CPU seconds. This method seems to be efficient and feasible for industrial circuits.

4. **Synchronous test generation model for asynchronous circuits** [BCR96]
   A test generation model which is synchronous in nature was introduced in this work testing asynchronous circuits. Main advantages of this method are 1) synchronous sequential test generation methods can be used to generate the test for the model, 2) the test generated using this model can be easily translated in to a test for the asynchronous circuit under test and 3) tests will not suffer from test invalidation due to unstable states. Automatic test generation for asynchronous circuits was discussed first in comparison with the synchronous circuits. By adding a delay element in the feedback path of the synchronous sequential circuit, increase in testing capability of the asynchronous circuits was pointed out with an example. Three key factors for properly modelling the asynchronous circuits were proposed namely 1) a new input pattern should only be applied after the circuit becomes stabilized and when it is fault-free, 2) the effect of the faults should be observed only when the faulty and fault free circuits have stabilized , and 3) The circuit should be allowed to cycle through the internal unstable states before it gets stabilized on application of the new input pattern.
   The Synchronous Test Model (STM) was introduced next. The model is composed of the input and output signals and the asynchronous latches are replaced with clocked flip-flops. These flipflops are clocked at the period equal to the critical path delay of the circuit. These flipflops are called the model flipflops as they exist only in the synchronous model of the asynchronous circuits. The model is also composed of three blocks namely Input Logic Block (ILB), Output Logic Block (OLB) and the input/output signal flow generator (IOFG). These additional blocks will appear only in the synchronous model and not in the actual hardware. The IOFG is used to both apply the input pattern and observe the output signals of the core circuits. The implementation of these three blocks at the gate level was described further.
   The testing framework using the STM for testing the asynchronous circuits was briefed further. The tests for faults in the STM can be generated using a conventional synchronous sequential circuit test generator. The translation of test for STM to the test for the actual asynchronous circuits was shown to be a linear procedure. An automated procedure for the test generation based on this method was given as a 5 step procedure. The Automatic Test Generation procedure is as follows

   - Step 1: Construct the STM for the ACUT assuming either a user-specified cycle length or an estimated one.
   - Step 2: Create the target fault list that contains only faults in the ACUT.
   - Step 3: Perform test generation on the STM using any off the-shelf synchronous

test generator.

- Step4: Translate these test patterns into sequences for the ACUT.
- Step 5: Validate the translated patterns by fault simulation on the ACUT

The experimental results of applying this method over several asynchronous benchmarks were reported. Two experiments were mainly performed. The first experiment was to verify that the unstable states are the main source of test invalidation and the second method was used to validate the STM model proposed and the fault coverage efficiency of the method. The fault coverage ranges from 88.5 to 100 percent. The test invalidation was zero for all circuits in proposed method whereas it was ranging from 4 to 17.9 percent in the Ad-hoc method. Next the application of the proposed method to test the embedded asynchronous circuits in the synchronous circuit was proposed. The test results for the embedded circuits in the ISCAS benchmarks were reported. The embedding of the asynchronous circuit was nothing but replacing the flipflops in the ISCAS benchmarks with the actual gate level representation which is asynchronous in nature by itself. The fault coverage was ranging from 62.9 to 93.4 percent. The test efficiency was ranging from 89 to 99.9 percent.

Thus an effective synchronous model for testing asynchronous circuits and the embedded asynchronous circuit in synchronous systems was proposed and the results were convincing enough to apply to the industrial circuits. Other applications of this STM were reported as test generation for gated-clock circuits and feedback bridging faults.

3.2.6 Random Testing

1. **Random Testing of Asynchronous VLSI circuits** [Pet94]

This work is an attempt to find possible ways to test asynchronous VLSI circuits using random (or, more accurately, pseudo-random) patterns. The main results have been obtained in the field of random testing of stuck-at faults in micropipelines. An asynchronous random testing interface has been designed which includes an asynchronous pseudo-random pattern generator and an asynchronous parallel signature analyser. A program model of the universal pseudo-random pattern generator has been developed. The universal pseudo-random pattern generator can produce multi-bit pseudo-random sequences without an obvious shift operation and it can also produce weighted pseudo-random test patterns. Mathematical expressions have been derived for predicting the test

length for random pattern testing of logic blocks of micropipelines by applying equiprobable and weighted random patterns to the inputs. The probabilistic properties of the n-input Muller-C element have been investigated. It is shown that the optimal random test procedure for the n-input Muller-C element is random testing using equiprobable input signals. Using the probabilistic properties of the Muller-C element and multiplexers incorporated into the circuit a certain class of asynchronous networks can be designed for random pattern testability. It is also shown how it is possible to produce pseudo-random patterns to detect all stuck-at faults in micropipelines.

2. **Designing asynchronous sequential circuits for random pattern testability** [PFRG95]
   A method for designing asynchronous sequential circuits for random pattern testability was proposed in this work. The general structure of the asynchronous sequential circuit was discussed first. Issues regarding the testing of micropipelines were discussed in detail. The drawback of the scan testing that, in shifting the n-bit patterns in to the DUT (Design Under Test) before actually applying it to the test object was pointed out. This was pointed out as important fact to reduce the testing performance of the BIST structures in which application of a large number of pseudo-random patterns forms the part of the BIST procedure. To overcome this, a solution is proposed which involves implementing the scan testing by shifting the random patterns bit serially with concurrent observation of the test results.
   Design of random pattern testable asynchronous sequential circuits were introduced in detail further. Two modes of operation of these circuits namely normal mode and test mode were detailed further. The proposed testable circuit has the test structure with additional hardware. It contains an additional register to collect the test data from the internal outputs of the combinational block, a block of XOR gates for mixing the test data and multiplexer to switch the data flow during the test phase. In addition to this, to facilitate the control signalling properly, two XOR gates, multiplexers and a toggle element were added. The mechanism for applying the random test patterns to the inputs and compressing the output responses of the combinational block were detailed further. The signature analyser used for collecting the test data from the internal outputs of the combinational circuit was described in detail. The signature analyser used was adapted from the well known BILBO (Built-In-Logic-Block-Observer) signature analyzer.
   The advantages of random testing the sequential circuit were reported to be 1) low complexity in testing procedure, 2) faster testing time of n-1 times, where n is the number of latches of registers and 3) the number of test patters for detecting all the single stuck-at faults in the circuit is equal to the number test patterns for detecting all the stuck-at faults in the combinational part of the circuit under test. The reason point 3 is attributed to the following 3 factors namely 1) all the stuck at faults on the inputs of registers Reg1 and

Reg2 are equivalent to the appropriate faults on the internal inputs of the combinational logic block, 2) all the stuck-at faults on the inputs/outputs of the block of XOR gates and Reg3 are detected easily during the test of the combinational circuit and 3) stuck-at faults on the control lines involved in the control of the random testing of the circuit are detectable as they cause deadlock of the circuit or change the data flow during the test which can be identified easily.

Hardware overhead and performance degradation were reported to be the disadvantage of this method. An experiment was carried using the circuit called "register destination decoder" which is a part of the asynchronous version of the ARM processor. Testable implementation of this circuit was first designed. The testing mode and normal mode of the circuit were executed to detect the stuck-at faults in the circuits. The test set consisted of 47 test patterns including 1) one test pattern which contains all zeros, 2) sixteen 'running one' test patterns and 3) thirty test patterns everyone of which include only two ones and all zeros. Up to a reduction of 165 times was reported for testing this circuit using weighted random test patterns. The CMOS implementation of this circuit consisted of 1011 transistors. The testable design of the same circuit was reported to be comprised of 1290 transistors and thus giving a hardware overhead of 27 percent.

3.2.7 Offline Testing

1. **Offline testing of asynchronous circuits** [Kop05]

   This work introduces a new method for testing the asynchronous circuits which is obtained by the direct mapping technique from 1-safe petrinets. Signal transition graphs (STG) and 1-safe petrinet were used for the representation of the circuit under test. The original petrinet based circuit description is converted in to a two level architecture which is composed of a tracker and a bouncer. The tracker and the bouncer are connected by means of read-arcs. Direct mapping from STG/petrinet involves introducing a David cell for each place in the petrinet or STG. David cells are sequential and speed-independent circuits. The fault models used are based on the physical faults occurring in the David cells. The fault model proposed capture three different errors due to the physical faults occurring in the David cells. First error is called token disappearing fault which occurs when the David cell executes its input handshake, but does not starts its output handshake, causing a deadlock. Second error called stuck-at-full error occurs when the David cell has its output wire at in the stuck-at-active state, which starts the output handshake prematurely and never finishing it. Third error occurs when a David cell receives a token, and starts its output handshake without finishing it. All these errors lead to the deadlock. A pseudo clock was used to detect other two faults in addition to these three errors.

The crux of this work is that the chain of David cells present in the decomposed circuit is converted into shift registers. An approach for testing single stuck at faults was also proposed. The test generation algorithm involves three steps namely a) conversion of the tracker in to an acyclic structure, b) generation of verilog netlist with control signals and demux-mux and c) test pattern generation. This approach was demonstrated over the benchmark, up-down counter. Case study on the benchmarks composed of David cells ranging from 5 to 17 were reported with 93% and 100% testability. Overhead incurred due to the addition of AND gate at the David cell interface was also reported.

3.2.8 Functional Testing

1. **Fast functional testing of delay insensitive circuits**[Pag95]
   A fast functional testing method for the test generation for delay insensitive circuits was proposed in this work. The circuits tested were the four phase handshake signalling based circuits designed using Martin's method [BM88]. A new block called OR/C block was used to facilitate the testing process and also to preserve the delay insensitivity of the circuit under test. This block acts as an OR gate during normal operation of the circuit and as a C-element during the testing phase. The program flow graph of the circuit is used for the test sequence generation and the OR/C block insertion/replacement. A synthesis method for the delay insensitive circuits represented in CSP-like language was described using an example. The CSP-like specification is then represented as a program flow graph. The guarded command present in the program flow graph is used in the testing process of these circuits.
   A testing method by simultaneous execution of the guarded sequences was briefed further. An algorithm named "multi_path" for determining the paths to be traversed during this process was also introduced. Following the test method, steps ensuring the correct operation of the circuit during the testing process were analysed. The effect of communicating multiple values was discussed with an example. The effects of simultaneous execution of the guarded sequences were discussed further during the testing process by using an example. Behaviour of the environment during the testing process was discussed further. Due to the distributive nature of the delay insensitive circuits, the testing time is considerably reduced due to the simultaneous execution of more than one guarded sequences in the program flow graph of the circuits. An extra overhead of 1 pin is needed to implement the testability feature during the synthesis process.

3.2.9 Fault Simulators and Test methods

1. **Testability of Asynchronous Self-Timed control circuits with delay assumptions**[BM91]
   A Testability method for timed asynchronous control circuits was proposed in this work. These circuits were built using standard logic cell with assigned min-max value for the rise and fall times. The circuit model was represented as a total state graph (TSG) to practically realize all the possible state transitions of the circuit. Another state graph named, invalid state graph was also introduced which depicts the functionality of the circuit when it is faulty. Faults used in the testability of the self timed circuits in this technique are single stuck-at-0 and stuck-at-1 faults. The control circuits are represented by the signal transitions of the circuit which is composed of the partial orderings of the signal transitions in a signal transition graph (STG). The memory element C-element is assumed to be driven by the combinational logic block decoupled from other inputs to the C-element. This is to ensure the testability of the C-element. The testing environment assumed in this work is considered slow enough to allow the circuit to be in the stable state until other nodes in the circuit gets stabilized. In other word, the input of the circuit is not changed until the effect of the previous input to the circuit has stabilized. The stuck at faults in combinational logic block, C-element were dealt separately. The fault inside the C-element is not considered in this method. The C-element is assumed as an atomic gate and the faults in the two inputs and the output were considered for testability.
   A sufficient condition for full testability of an asynchronous control circuit was also proposed. The conditions is that for an asynchronous control circuit to be 100% fault testable for single stuck-at fault, when 1) for all the faults, the circuit is capable of traversing from one reachable state in which the fault can propagate to another state in which the outputs are different for the circuit, 2) for all the faults, the circuit will not traverse from a valid state to another state where the output of the circuits are as expected but the output of the memory elements are different. An automated testability checker tool was also implemented. The tool reads in the circuit under test with the minimum and maximum gate delays assigned to each internal gate and outputs the declaration of the testability of the circuit along with the list of states traversed for testing.

2. **FSIMAC Simulator** [SKR00]
   A fault simulator called FSIMAC for stuck at faults and gate-delay faults for asynchronous sequential circuits was developed in this work. The time frame unfolding method is used in this fault simulator to simulate the faulty and good machines which is sequential. The time-frame boundaries for the synchronous circuits are the boundaries of their clock, but in case of the asynchronous circuits , feedback loops present in them

bound the time frame. Hence a new feedback identification algorithm was proposed. This algorithm was a variant of the conventional feedback loop breaking algorithm except that this new proposed algorithm used breadth first search instead of depth first search during the scan element selection process. Min-max timing analysis and the 13 valued logic were used by the simulator for the timing analysis of the frames.

The main target circuits for this fault simulator were those of RAPPID resembling the extend burst-mode machines. The delay model used here is the bounded delay model as the min-max timing analysis approach is based on bounded delay model. An equal nominal gate delays for rising and falling transitions, and zero wire delays without sacrificing the simulation model. Some of the inertial delays were also modelled due to the presence of domino gates in the target circuits chosen. 13 valued waveform logic was used for the simulation, which is capable of dealing with the hazards during the circuit analysis. The signal waveforms were represented as a triple <b,m,e> with b denoting the begin state of the signal, e the end state and m the intermediate transition behaviour. The class of waveforms in the 13-valued waveforms are constant, transition, hazard, stabilizing, unstabilizing and undefined. This classification is based on the transition of the signals and also their stability.

A conversion method from 3-valued logic to 13-value logic was introduced. First, All the input waveform sequences were taken and the corresponding sequences of states from being state to end state is computed. Second, the function for the begin state and the end state were defined. Third, based on the function, the value of the m is computed by monitoring the transitions occurring during the state change. These three steps were applied to develop a 13-value logic by storing all the 13-valued functions as a pre-computed look up table. A demonstration of this fault simulator over the complex domino logic circuit was demonstrated with the HDL description and the files generated by the FSIMAC simulator.

A algorithm named feedback_detect based on breadth first search was described. The algorithm involves storing two indices namely "level" and "flag" for each gate in the graph description of the circuit. The level computes the number of gates between the PI and the current gate and the flag variable stores the completion of the level computation. Two traversal lists namely TRUE_LIST and FALSE_LIST were used to completely traverse all the vertices/gates of the circuit. Feedbacks were added in the separate list named FEED and its evaluation is registered in the list called Eval_Feed_List. Thought the identification of feedbacks is on the fly and the method of finding them is not elaborated. The inputs to the fault simulator FSIMAC are a) verilog gate level description , b) minimum and maximum gate delay bounds and c) a sequence of test vectors. First the fault free good machine is simulated and then the fault machine is simulated for each fault.

Fault models used are single stuck-at-0, stuck-at-1 and gate delay faults. For the sequential circuit, the simulations are done frame-after-frame with bounds being the feedbacks detected by the feedback_detect algorithm. Once the circuit is initialized by the input values provided by the user, the current frame is simulated using the first test stimuli and min-max timing analysis. When the primary output value becomes stable, the values of the output value of the fanin gate g for every feedback are fed as the next input value of fanout gate f for every feedback. For stuck at faults the Primary outputs for the good and faulty circuits are examined for fault detection. For the gate delay fault, time stamps for the primary output signals at the end of each frame is examined for fault detection and reporting. Several benchmarks from Phillips and Intel were experimented with the fault simulator and the results are reported.

3. **Testing two-phase transition signalling based self-timed circuits in a synthesis environment**[KA94]

   A testing technique for self-timed asynchronous circuits taking advantage of the automated synthesis method of self-timed circuits was introduced in this work. A synthesis environment named SHILPA was developed. The circuit description for the asynchronous circuit was based on the hopCP, a high level concurrent HDL, which is based on CSP. The circuits, are described as a collection of concurrent processes communicating through the synchronous channels via handshake through restricted shared variables. The transition signalling, known as two-phase or event-based signalling, is used in these designs attributing to its high performance and low power consumption. A clear example of the hopCP description of a self-timed asynchronous circuit was demonstrated. The represented behaviour of the circuits is converted in to an annotated Petrinet called HFG, where the places denote the states of the system, the actions/Boolean evaluations denote the transitions of the Petrinet. The HFG is then converted in to a self-timed circuit using a syntax-directed translation procedure called action refinement. Action refinement involves a set of petrinet based transformations to convert the HFG to a RTL level description. In the proposed synthesis framework, every block of the design is represented by an action block which implements the hopCP action. The action blocks were also classified in to three types namely Control action blocks, Function action blocks and Predicate action blocks which models the control flow, functions and the Boolean predicates respectively. Now this automated synthesis method is used to generate test for the synthesized circuits too.

   Two types of fault models were considered during the testing process: the stuck-at fault and the delay fault models. But only the stuck fault model was demonstrated in the work. The fault model assumption over here is based on the capability of transferring a 0 to 1 and a 1 to 0 transitions through a node and in which case the node is considered to be

void of stuck-at-0 and stuck-at-1 faults. Thus to test a node for the stuck-at test, two transitions have to be passed through the path from the input through the node to the output of the circuit. This process still has a bottleneck over the proper justification sequence needed to propagate the transitions. To overcome this, the SELECT module in the library of asynchronous circuits used by SHILPA is modified. Also, all the macromodules in the designs are considered to be atomic gates and thereby the faults inside the modules are not considered during the test generation. The design of the SCANSELECT module which aids the test generation process was briefed. The proposed SCANSELECT module is also considered as an atomic gate during the test pattern generation.
Next an algorithm for the test pattern generation was proposed. The algorithm takes in the output of the SHILPA synthesis system namely the NHFG, the set of resources and the physical netlist. The output of the testing algorithm proposed are 1) the test vectors for all the stuck-at faults, 2) control sequences to test the control part of the circuit and to setup the conditions to test the data path, and 3) the points on the circuit to be used in setting up the scan chain. The top level flow of the algorithm is as follows. Once the output of the SHILPA system is read in, the circuit is partitioned in to datapath and control path. Then the procedure called testCab is applied to test the control path which returns the control sequences for the control path testing. Next, the testDataPath procedure is run, which generates the test vectors for all stuck at faults and also the control sequences . Then the algorithm returns the the control sequence of the control path, the control sequences and the test vectors for the data path and the points selected for the scan chain implementation.
The whole synthesis and testing flow was applied to several asynchronous benchmarks and fault coverage of 100 percent was reported. The area overhead for the circuit will be contributed mainly by the modification in the SELECT module. The pin overhead was reported to be 7.

4. **Testing Redundant asynchronous circuits by variable phase splitting**[LKL94]

This work proposes a test generation approach for stuck-at and delay fault testing of asynchronous circuits without the addition of any logic. This method is based on partitioning the asynchronous circuit into combinational and memory elements. The full stuck and delay testability was achieved under weak conditions with an assumption of being able to drive both phases of the each combinational logic input independently. Any two level circuit implementing a unate function is automatically prime and irredundant if it is free from single cube containment. This property is mainly exploited in this method. The method proposed is called testing by variable phase splitting. Instead of modifying

the logic, the inputs of the circuit are modified to enable testability of the circuits. It is done by treating the positive and negative phase of each input variable as the separate entities.

Next a design for testability for the test methodology was introduced. Scan flipflops are needed to apply the proposed test methodology. The synthesis algorithms for three main classes of asynchronous circuits namely 1) Huffman circuits, 2) Burst Mode Machines and 3) Bounded delay circuits are shown to be synthesizing the circuits that can preserve the single stuck-at and the delay fault testability if the synthesis procedure obeys the constraint proposed during the synthesis. Experiments were carried out the asynchronous benchmarks and results with full testability were reported. A greedy algorithm was implemented for carrying out the experiment which ensures full testability by splitting each input signal. The heuristics used using this approach employ the order of splitting by considering signals that are at the near end of the untestable path and then the non-unate signals. The number of split signals used are reported to be very low and also the execution time reported were only of few seconds for the circuits with literal size ranging from 10 to 52.

Several other fault simulators for asynchronous circuits reported in the literature lately are [SM04a],[BR]

3.2.10 Fault Modelling

1. **High level fault modelling of asynchronous circuits** [Lu95]
   A high level fault model was proposed in this method for testing asynchronous circuits. The fault model is based on the signal transition graph. The fault model introduced here is called Transitional fault models. Complete fault machines of the C-element for the stuck on, bridging and stuck faults were derived. The C-element implementation style was dynamic C-element.Total 34 possible faults were realized. Out of 34, only 6 were modelled by the stuck-at fault model. Rest of the 31 faults were modelled based on the proposed transitional fault model. Definitions of two new transitional faults namely transition unable fault and extra transition faults for the C-element behaviour. Stuck-at-false and stuck-at-true faults are the proposed transition fault models. Stuck-at-false in the STG is that one of the pre-conditions of a transition is always false. This fault is represented by adding a ’0’ in the STG. Stuck-at-true fault is the fault in the STG in which one of the preconditions of the transition is always true. This is represented by adding a ’1’ in the arc corresponding to that precondition. Further the transition fault was divided in to single and multiple signal transition faults namely Single Signal Transition Fault (SSTF) and Multiple Signal Transition Fault(MSTF). When only one signal transition fault oc-

curs at a time in the STG, then it is called SSTF. When more than one SSTF occurs in the STG at a time, then the fault is called MSTF. Most of the functionally irredundant faults can be modelled using the transition faults.
Further the fault collapsing technique for the signal transition faults is introduced. Transition fault equivalence and Transition fault dominance were defined. The proposed fault models were used to generate test for the asynchronous benchmark namely asynchronous neuron. The analog fault simulator was used to map all the transistor level fault models to gate level transitional fault models. Then the gate level transitional fault models were used to generate test using the STG. It has been reported that more than 90 percent of the transistor level faults could be covered by the proposed fault models. Still a set of fault namely parametric faults could not be detected by these models

2. **Issues in fault modelling and testing of micropipelines** [PVS92]

   A testing technique for Micropipelines is introduced in this work. Micropipeline's advantages over the synchronous pipelines are 1) it has the minimum possible response time equal to the delay of all the stages, 2) the logic circuitry is simple, 3) optimal working speed of each stage is guaranteed and 4) problems regarding the devising clocking schemes for synchronous pipelines are not encountered. In terms of the testability further three more advantages namely 1) control parts of the micropipelines are concurrently testable, 2) test pattern generation for data part logic can be reduced to that of the combination circuit with an update in the test application method, and 3)testing latches requires test pattern test which can be obtained from the usual test pattern generation methods for combinational circuits. The stuck at fault model were used for the test pattern generation of micropipeline namely 1) faults in the control part of the pipelines, 2) faults in the logic blocks and 3) faults in the latches. The faults in the latches considered were the single stuck-at-faults, single stuck-at-capture faults and the single stuck-at-pass faults.

3. **Fault effects in asynchronous sequential logic circuits** [SWF93]
   This work studies three types of fault effects in the Huffman model of asynchronous circuits. The three types of fault effects are equivalent-state redundant faults, invalid=state redundant faults and state oscillations. In this work following assumption on the asynchronous circuit being analyzed is made, 1) the circuits are tested and operated in the fundamental with only one input changing at a time, 2) circuit has the reset state from which all the input test sequences are started, 3)two-level implementation in which only prime and non redundant implicants are present except the redundant logic for the static hazard protection and no shared logic for the next state equation and output equation and 4) the single stuck at fault model is used. The equivalent-state redundant faults are

reported to be generated when there is a violation in the fundamental mode constraints. The invalid state faults occur due to the presence of invalid states or improper assignment of don't care terms. State oscillations occur due to the presence of the critical races. Three properties for the non occurrence of the oscillations were introduced. Property 1 states that two-state oscillations will never occur in a race-free fault circuit. Property 2 states that if each input column contains either 1) atmost one k-connected path, k > 3 or b) k-connected paths , k <= 3, then no multistate oscillation occurs in a race-fee faulty circuit. Property 3 states that , if there exists two disjoint k-connected paths, k > 3 and dH(Sx,Sy)(distance between Sx and Sy) >1 for all Sx and Sy in different connected paths, then no multistate oscillations occur in the fault circuit. And finally a set of rules for synthesizing testable asynchronous sequential logic circuit was also given.

3.2.11 Switch/Transistor Level Testing

1. **A switch level test Generation for system for synchronous and asynchronous circuits** [ES95]

   A switch-level test generation system for synchronous and asynchronous circuits has been developed in which a new algorithm for fully automatic switch-level test generation and an existing fault simulator have been integrated. For test generation, a switch-level circuit is modelled as a logic network that correctly models the behavior of the switch-level including bidirectionality, dynamic charge storage, and ratioed logic. The algorithm is able to generate tests for combinational and sequential circuits. Both nMOS and CMOS circuits can be modelled. In addition to the classical line stuck-at faults, the algorithm is able to handle stuck-open and stuck-closed faults on the transistors of the circuit. In synchronous circuits, the time-frame based algorithm uses asynchronous processing within each clock phase to achieve stability in the circuit and synchronous processing between clock phases to model the passage of time. In asynchronous circuits, the algorithm uses asynchronous processing to reach stability within and between modules. Unlike earlier time-frame based test generators for general sequential circuits, the test generator presented uses the monotonicity of the logic network to speed up the search for a solution. Results on benchmark circuits show that the test generator outperforms an existing switch-level test generator both in time and space requirements. The algorithm is adaptable to mixed-level test generation.

2. **Test quality of asynchronous circuits: a defect oriented evaluation** [RB96]

   A detailed analysis on the test quality of the asynchronous circuits using defect based fault models was undertaken in this work. The transistor level implementation of the sequencer circuits with 14 transistors and 2 inverters was also presented. Next the design

and test aspects of handshake logic circuits were discussed.

Then, a detailed introduction to the defect-oriented testing was given. This approach namely inductive fault analysis and the tool in which it was implemented was also pointed out. The testability of the opens and shorts in the handshake circuits were analyzed using this tool named Analog SystemQ. Next the fault models namely stuck open, short and bridging fault models were described. Three different test methods were used for the analysis and evaluation. The first method is based on the deadlock detection. The second method is based on voltage testing and the third one being the IDDQ testing.

Next a detailed fault analysis for the component SEQ was carried out. First the fault free behaviour was simulated and the corresponding waveforms were recorded for all the nodes and also the I_{DD}. All the bridging fault and the stuck-at faults were considered for the analysis, which accounted for totally 91 bridging faults and 30 stuck at faults. A fault coverage 88 percent for the bridging faults and 97 percent for the stuck at faults were reported. Also 12 undetectable faults were reported. Based on this three classes of faults were classified for this sequencer circuit. The percentage of bridging and stuck-at faults detected by each of the three testing methods was represented using the Venn diagram. A DFT component named HOLD was introduced further and the transistor level implementation of it was shown with 14 transistors and 3 inverters. This element facilitates the lock-stepping of the circuit operation to create sufficient quiescent states for IDDQ and the scan test. The simulation results for the sequencer circuit s with the HOLD element was reported in the same way using the Venn diagram. Stuck-at fault coverage of 95 percent and 95 percent bridging fault coverage was reported for this DFT based design of the sequencer.

Next the fault analysis for the other handshake components was done. The DFT approach of adding HOLD element was carried out to analyze all these components. The components namely MUX, CASE, DO, PAR and HOLD were analyzed and the results were reported. Test performance and the cost of all the test methods were analyzed further. It was evident that the scan testing along with the IDDQ testing was needed to get good fault coverage. Costs with respect to area, power dissipation and delay were also reported.

3.2.12 Self Testing Asynchronous Designs

1. **Self-Timed is Self-Diagnostic** [DGY90]

   A self diagnostic design of asynchronous circuits was introduced in this work. A technique for implementing any Boolean equation into a self checking asynchronous design was proposed. A combinational module was implemented in ternary in which, the

logic values will be 0,1,U, where U is an undefined value. The sequential behaviour of the implemented combinational module was specified using a cycle of activities namely E1,S1,E2,S2,.....E4,S4, where E's are environments and S's are network functional constraints. Also the ternary logic was employed using the dual-rail logic with the logic values 0,1, and U represented as 10, 00, and 01 respectively. The combinational module was composed of 4 subnets name ORN, CEN, DRN and OUTN. The subnet ORN detects when each of the input has become defined or undefined. The subnet CEN is designed in such a way that it detects when all the inputs are defined or undefined. The arbitrary set of Boolean equations are implemented by the subnet DRN. Subnet OUTP retains the output of the combination logic module's outputs to undefined value until all the inputs become defined and only after that the correct outputs will be produced. A detailed self diagnostic model was described for the circuits with stuck-at-fault only. The self-diagnostic system was defined in this work as a design in which the occurrence of a single stuck-at-fault and a sequence of environment transitions E1-E4 , either produces the correct outputs or goes to a hung-up state or an illegal final state. Based on this definition, several theorems to prove the self-diagnostic design was further briefed. Sixteen different cases involving stuck-at-1 and stuck-at-0 faults on the wires of the proposed designs were discussed further. The detection of faults in the proposed self-diagnostic design was aided by the four phase signalling protocol used for the communication between the circuit and environment. Low hardware requirements was also an advantage of this technique compared to the usual self-checking logic design.

Review:

3.2.13 Critical Analysis

Design For Test

Though several testable design methods targeting specific low level hardware were introduced for asynchronous circuits, they are very specific to certain design style or certain hardware for specific application. No general asynchronous DFT is currently available which can be applied in a generic manner for any type of asynchronous design style. Hence several recent methods addressed in literature still follows the synchronous design based test methods for asynchronous circuits.

SCAN Testing

Full scan testing methods introduced for asynchronous circuits are only robust test method currently available for testing asynchronous. But the issue with this method is that, the area over head will be higher compared to the original circuit. For the asynchronous control circuits with too many C-elements , it will be important to develop partial scan test generation method to reduce the test area overhead. Not many partial scan methods are reported in the literature for the asynchronous design. This paves way for the main motivation of this thesis to develop test methods for asynchronous circuits.

Synthesis for Testability

Synthesis for testability for asynchronous circuits is a very rarely dealt topic in asynchronous test community. As given in the review the methods in [KLSV91] and [NJC95] were reported in late 90's. Feasibility of these methods for the current technology nodes will be an important question to address.

Test Pattern Generation for Asynchronous Circuits

Four different ATPG techniques were reviewed in the previous subsections. It should be noted that these test methods were reported long back in 1970s. Only two recent ATPG methods were reported in the literature ([SM04a] and [Roi97]). But these methods are based on STG based and random test vector based ATPGs. The number of test patterns generated by these methods is very high. A very effective test method with optimal number of test patterns and still with very good fault coverage similar to synchronous ATPG methods is yet to be developed.

Fault Simulation

FSIMAC [SKR00], SPIN-SIM [SM04a] and [BR] are some of the fault simulators reported in the literature. They still follow fault simulation methods used for synchronous circuit design and are adapted to address the hazards in asynchronous circuits. Developing more fault simulators with fault models targeting faults on asynchronous circuits will improve the future test generation methods which will use these simulators.

Fault Modeling and Transistor Level Testing

Fault models for asynchronous circuit test are still under very early stages. Most of the test methods introduced for asynchronous circuit usually apply stuck-at fault model as they are the golden models for the past few decades. But transistor level fault models like stuck-open faults cannot be completely detected by this model. The discussion on this topic is addressed in the Chapter 9. Developing new fault models is currently necessary as the technology node already reached sub 30nm. Several defects occurring at these nodes and their impact on asynchronous design styles have to be addressed.

This thesis addresses most of the above mentioned topics. Firstly, as mentioned in the outline of the contribution of the thesis in the introduction chapter, three partial scan DFT methods were introduced in this thesis. This contribution aligns with the current challenge of addressing the partial scan test methods for asynchronous circuits. Next the fault model other than stuck-at fault model was considered to generate test for detecting transistor level faults. With the transistor feature size reducing this method addresses the important challenge of improved testing of asynchronous circuits for transistor level defects. Finally, a pre-processing method aiding the asynchronous test generation process was introduced. This will reduce the complexity of the test generation algorithm proposed for asynchronous circuits by reducing the problem size.

3.3 Conclusion

A brief literature review on testing for asynchronous circuits was carried out in this chapter. Several works related to the testing of asynchronous circuits were analysed. Automatic test pattern generation methods for generating test patterns for asynchronous circuits were also reviewed. The defect level test generation system were also found in the literature, but a very few works were reported in actual test generation at defect level. Following the review in this chapter, a comparative study of two test generation methods will be carried out in the next chapter to probe the test generation issues in asynchronous further deeper.

Chapter 4

Automatic Test Pattern Generation for Asynchronous Circuits: A Comparative Study

4.1 Introduction

This chapter deals with the analysis of two approaches for test pattern generation of asynchronous circuits. The first approach is uses a symbolic method based on state traversal, while the second one is based on an adaptation of the well-known scan insertion technique.

A comparative analysis of two different methods of test generation for asynchronous circuits is carried out in this chapter. The two methods are

- Automatic Test Pattern Generation based on symbolic reachability analysis [RCPP97]
- Scan insertion based test generation [BA05]

The organization of the chapter is as follows: Section 2 describes the State Transition Graph (STG) based automatic test pattern generation; Section 3 describes the test pattern generation based on the scan insertion technique; Section 4 gives a comparison of test generated by two approaches for a number of small benchmarks; the chapter is concluded in Section 5.

4.2 Automatic Test Pattern Generation based on Symbolic Reachability Analysis

This section briefly describes the approach of automatic test pattern generation used in [RCPP97]. It proposed a testing strategy with the following features:

- The behaviour of the asynchronous circuit is modelled as a synchronous finite state machine.
- Test patterns are generated using symbolic technique from the modelled FSM.

Test patterns can be synchronously applied to the asynchronous circuits and faults are made available at the output. An asynchronous circuit in this approach is modeled as an interconnection of gates and delay elements. The delay model used here is an unbounded gate delay model [KF91].

4.2.1 Definition

State Graph (SG) A state graph (SG) is a pair <S,E>, where s is the set of states and E $\subseteq$ S $\times$ S is the set of edges (transitions).

Circuit Stage Graph(CSG) A circuit state graph (CSG) is a 7-tuple <S,E,P,G,S F0,λ_P,λ_G> , where

- <S,E> is a State Graph, P = {p_1....p_m } is the set of primary inputs,
- G={g_1....g_n } is the set of gates
- $S_0 \subseteq$ S is the set of initial states
- The labeling functions λ_P : S $\rightarrow$ $\{0,1\}^m$, and
- λ_G: S $\rightarrow$ $\{0,1\}^n$ map each state, s, with binary vector consisting of the values s of primary inputs and gates, respectively.

The next state of a circuit under unbounded gate delay model depends on its present state. A gate is said to be "excited", if its output differs from the function it implements and "stable" otherwise. A next state function δ: SxG$\rightarrow$ S can be defined for each gate. Function δ(s, g_i) returns either the state reached by switching the output of g_i if it is excited, or s, if g_i is stable. A transition relation, R relates pairs of predecessor/ successor states. If state s' is an immediate successor of state s, it will be assumed that both states are in relation R, denoted sRs', or (s,s')

$\in$ R. By using the next state function of each gate, the transition relation associated with circuit gates are defined as:

$$R_\delta = \{(s,s') \in S \times S | \text{s is stable} \wedge s = s1 \vee (\exists g_i \in G) \text{such that } s' = \delta(s,g_i) \neq s)\}$$

For each pair (s,s') $\in R_\delta$, if s is stable, its successor is the same s; otherwise, the successor is obtained by switching an excited gate. The transition relation associated to input signals are defined as follows:

$$R_I = \{(s,s') \in S \times S | \text{ s is stable } \wedge \lambda_p(s) \neq \lambda_p(s) \wedge \lambda_G(s) = \lambda_G(s')\}$$

Thus the transition relation of the circuit in test mode is defined as R = $R_I \cup R_\delta$.

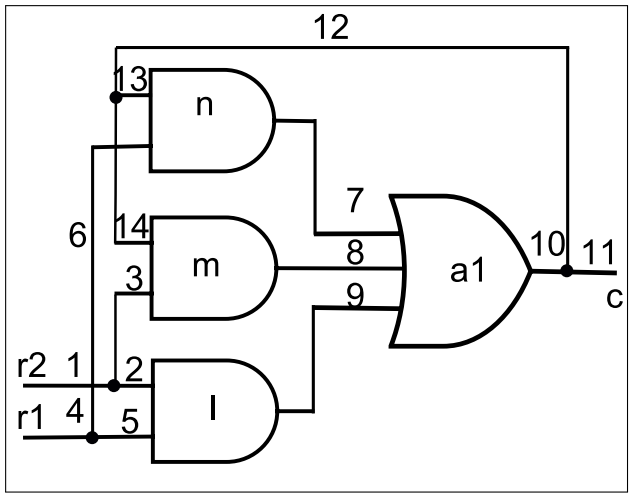


Figure 4.1: Majority Gate Based C-Element

4.2.2 Synchronous Abstraction of the Circuit State

To calculate the synchronous abstraction of the testable Circuit State graph, the pairs of states, (s,s'), such that s' is reached from s at the end of the test cycle is defined. Each pair has an associated input pattern based on the different values of inputs in s and s'. The set of all these pairs were called Test Cycle Relation (TCR). For practical reasons it was assumed that the circuit must settle in at most k transitions. The k-step test cycle relation (TCR^k) represents the pairs (s,s') distant at most k transitions. TCR^k for a given CSG in test mode <S,E,P,G,S_0,λ_P,λ_G> is defined as:

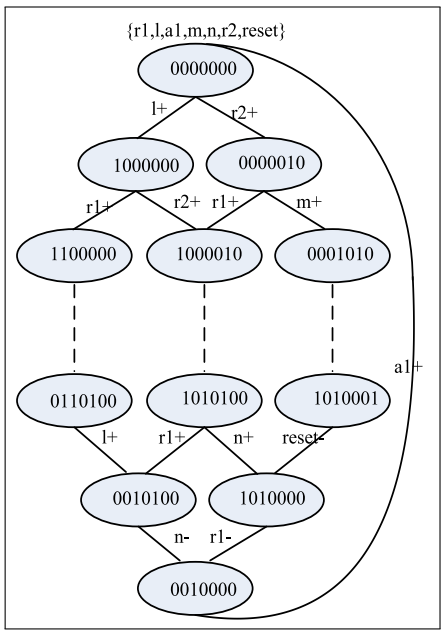


Figure 4.2: State Graph

$$TCR^k = \left\{ (s,s') \in S \times S | \exists s_1 ..... ,s_k \text{ such that } sR_I s_1 \wedge ({}^k\Lambda_{i=2} s_{i-1} R_\delta s_i) \wedge s_k = s' \right\}$$

Invalid pairs of states are removed in the next step. Vectors causing non-confluence are detected in pairs (s, s') and (s, s") such that s' and s" with the same input values exist. Patterns producing oscillation or unacceptably long test cycles are found if s' is unstable. The k-Confluent Stable State Graph, denoted as CSSG^k, is formed by those pairs in TCR^k that present neither non-confluence nor cause the circuit to be unstable after k transitions. Formally it is defined as

$$CSSG^k = \left\{ (s,s') \in TCR^k | \text{s' is stable } \wedge \nexists (s,s'') \in TCR^k \text{such that } [s' \neq s'' \wedge \lambda_I(s') = \lambda_I(s'')] \right\}$$

Thus each one of CSSG ^{k} 's nodes represents a stable state. An arc between two nodes s and s' exists, if s' is stable and the only state reachable from s in at most k transitions by applying some input pattern. An example to show the approach of the above theory is given below using the C-element, implemented by a majority gate, shown in Figure 4.1. The C-element shown is a model with two input signals, r1 and r2, and four gates. The circuit state graph modelled for this circuit is a 7 tuple <S,E,P,G,S_0,λ_P,λ_G>, where <S,E> is a State Graph, P = {r1,r2,reset} is the set of primary inputs (the reset signal is added by the Testify tool which initializes any memory element in the circuit), G={l,m,n,a1} is the set of gates and $S_0 \in S$ is the set of initial states. The labelling functions λ_P: S $\rightarrow \{0,1\}^3$ and λ_G :S $\rightarrow \{0,1\}^4$, map each state s with a binary vector consisting of the values s of primary inputs and gates, respectively. Thus the elements of set, S (set of reachable states), has a binary vector of length 7. In total, 128 states form the set S. The reachable states can be calculated by using a symbolic traversal algorithm

like the one used in [JRBD94]. The set for this circuit is obtained by enumerating over the (128 $\times$ 128) states. The next state functions for each gate defined for this circuit are (δ_l: S $\times$ l $\rightarrow$ S), (δ_m : S $\times$ m $\rightarrow$ S), (δ_n: S $\times$ n $\rightarrow$ S), (δ_y: S $\times$ y$\rightarrow$ S) which operate over the gates l, m, n and y, respectively.

From this circuit state graph model and next state functions, the transition relation R = $R_I \cup R_\delta$ are obtained, which forms a set of stable state pairs. Next the synchronous abstraction involving computation of TCR ^{k} and CSSG^k is made. The state graph evaluated for this circuit model is as shown in Figure 4.2. Testify generated 34 edges which form the transition relation between the states. For the sake of clarity, only part of the state graph is shown. After several iterations, the set of stable state pairs are ready for test generation. With these set of stable states, test pattern generation was performed in three phases: fault activation, state justification and state differentiation, as described in [RCPP97]. The test generation is carried out using Random TPG and Ternary simulation [RCPP97]. The stable state pairs picked for test generation for this circuit are (s1, s127), (s127, s1), (s2, s3), (s127, s89), (s64, s65), and (s127, s22). The encoded binary codes on these state pairs were generated which corresponded to the test patterns covering 24 fault sites. The test patterns obtained for this circuit were *(0000001, 1111111), (1111111, 0000001), (0000010, 0000011), (1111111, 1011001), (1000000, 1000001), (1111111, 0010111).* The size of the test pattern was 7, which is equal to the size of the binary encoded state variables in the state pairs. 12 patterns were generated for 24 faults. To validate the approach several benchmarks synthesized by Petrify were tested and the results are analyzed in Section 4.4.

4.3 Scan Latch Insertion Based Test Generation

This section describes the test pattern generation based on scan latch insertion [KF91]. Asynchronous circuits can be represented as combinational blocks with feedback loops. Effective test pattern generation involves breaking these feedback loops and inserting scan latches in these loops, thereby making the circuit completely combinational. Level sensitive latches are used as they restore the asynchronous operation during the normal mode of operation by keeping them transparent. The loops may be global or local feedback ones. In the test mode, the asynchronous circuit operates synchronously with the scan latches being fed with test patterns and the outputs scanned out.

The LSSD scan design [KF91] is shown in Figure 4.3. It was designed with a 2:1 multiplexer and two latches and operates using 2 phase, level sensitive clocks. The signals 'x' and 'y' provide the path for normal operation of the circuit. The signals SI and y form the test mode path. This design is fully stuck-at testable. Several optimized circuits [KF91] are possible for

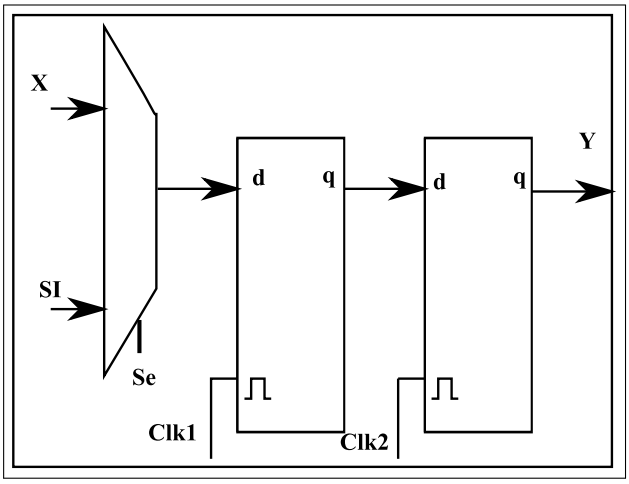


Figure 4.3: LSSD Latch Desgin

the scan latch design inserted in the feedback loop of the C-element. The simplest and robust scan design is shown here. The scan mode is used for several cycles to apply the test patterns to the scan latches. The scanned output reveals the potential faults in the design. To illustrate this approach, once again a majority gate based C-element is considered. The circuit consists of 2 input signals r1 and r2 with the output signal a1. Thus the LSSD Latch is inserted at the node 10 to break the feedback [KF91]. The modified circuit is shown in Figure 4.4.

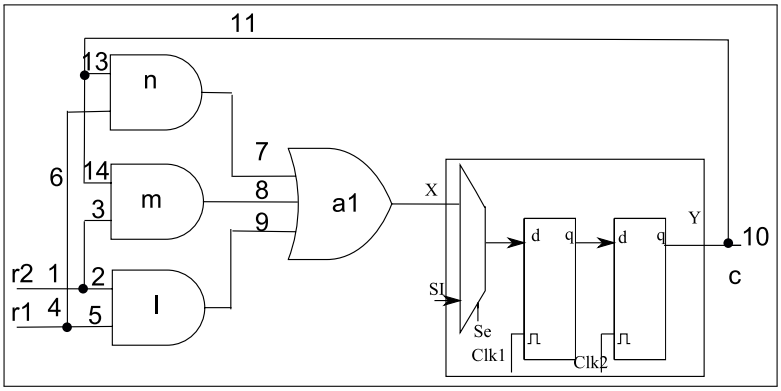


Figure 4.4: Celement Design with LSSD Latch

The test generation for the modified circuit is carried out using standard test pattern generation tools. This is an important aspect of this method, since such tools are fast, reliable and produce high-quality test patterns. This approach can be automated as shown below:

- Read in the design net-list
- Remove local loops by adding scan latches for each C-element (if present)
- Break the global feedback loops
- Insert the proposed scan latch at the feedback loop points

- Generate the modified net-list of the original design file with local and global loop scan insertion
- Apply the net-list to the ATPG tool to generate the test patterns

The fault coverage obtained over different benchmarks by using this method in comparison with that obtained using the symbolic technique is discussed in the next section.

4.4 Comparison of results

This section compares the results of the two proposed approaches by applying them to several benchmarks synthesized using Petrify which is used in the asynchronous community [CKK+96a]. The fault coverage and test patterns based on first method was generated using the tool Testify [Roi97] which is developed from the same approach. Table 4.1 gives details on fault coverage, number of test patterns, total number of faults, total number of detectable faults and total number of detected faults for several benchmark circuits.

For the second approach, the fault coverage and test patterns were generated by cutting the global loops manually and inserting the scan latch in the feedback paths. After inserting the latches, the netlist was fed into the Synopsys Tetramax ATPG tool to generate the test patterns and calculate the fault coverage. Table 4.2 gives the fault coverage for the same benchmarks and summarizes the test patterns generated using the scan insertion method.

4.4.1 Example

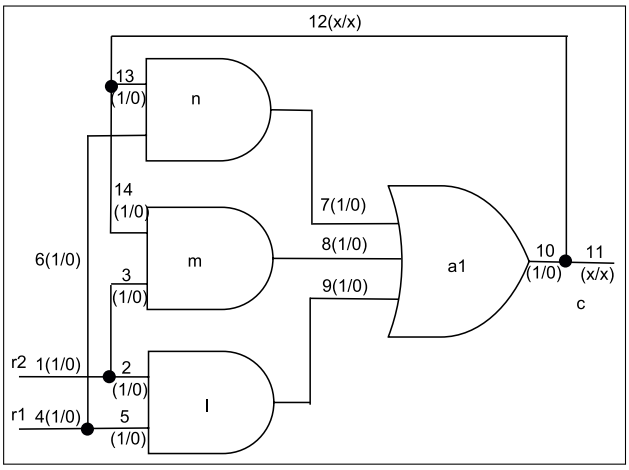


Figure 4.5: C-element-Faults detected by testify

For the C-element, the faults covered by testify are 24 out of 28 faults as shown in Figure 4.5. As evident from the figure, testify generated tests based on the primary input and the gates. So

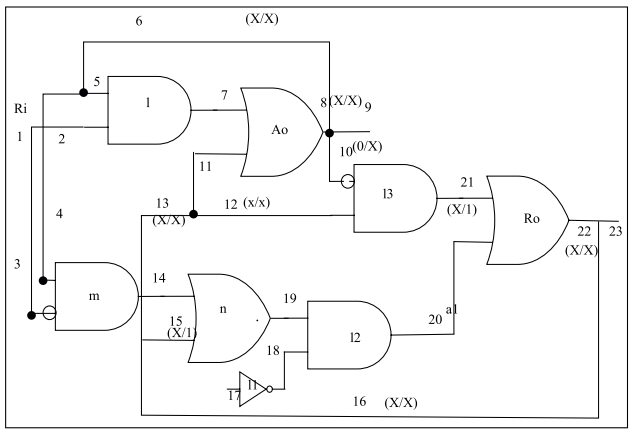


Figure 4.6: Half-Faults detected by testify

it could not detect the faults at the nodes 11 and 12 which are represented by (x/x). Although the test at node 10 covers the fault 11, it does not cover fault at node 12. The output of the gate a1, node 10, was taken into account as a single node which comprises of nodes 10, 11 and 12. But the fan-out nodes (13 and 14) from 12 are considered as test nodes as they form the input for the gates, n and m, respectively. Testify generated 12 test patterns of length 7 covering 24 fault sites in the circuit. The test patterns should be applied synchronously to stabilize the circuit at each pattern interval. Similarly for the benchmark circuit "half" (Fig.4.6), the faults covered by testify are only the inputs and outputs signals of all the gates. For this benchmark, even 5 more faults at input/output fault sites namely 10(0/x), 12(0/1), 15(x/1), and 21(x/1) were not detected by testify. Other intermediate node fault sites include 6(x/x), 13(x/x) 16(x/x), and 22(x/x). Testify generated 24 patterns of length 11. From these results, it is evident that any proposed test generation algorithm to be developed should focus on testing the intermediate nodes which will be overseen by the circuit models which are modelled with only the input and output signals of each gate.

4.4.2 Analysis

This section provides the insights for the undetectable faulty nodes in the asynchronous circuits. The intention is to give two working examples on how the fault simulation in two different methods compared, how it ignores the undetectable nodes and provides higher fault coverage without including these nodes.

4.4.2.1 Detectable Faults

First, we will compare the total number of nodes and total number of testable nodes listed by these two methods. Figure 4.7 shows comparison of total number nodes listed by the tools

Table 4.1: Fault Coverage using Symbolic Method

Benchmark	Number of Patterns	Number of Faults	Testable Faults	Number of Faults(D)	Fault Coverage (%)
chu133	10	60	40	40	100
chu150	19	64	52	52	100
converta	38	58	44	44	100
dff	34	52	44	44	100
ebergen	30	86	70	70	100
half	15	22	14	14	100
hazard	55	52	44	44	100
Master-read	-	160	130	126	96.92
mmu	203	166	136	128	94.12
mpforward	19	68	58	58	100
mr1	-	170	140	135	96.43
nak-pa	19	100	80	80	100
nowick	13	68	54	54	100
ram-read-sbuf	69	102	82	82	100
rcv-setup	12	46	36	36	100
rpdft	11	80	62	62	100
sbuf-ram-write	72	124	102	102	100
sbuf-send-ctl	60	106	86	85	96.51
sbuf-send-pkt2	101	146	116	113	97.41
seq4	145	104	86	86	100
seq_mix	245	178	142	138	97.18
trimos-send	72	162	132	124	93.94
vbe5b	22	52	42	42	100
vbe5c	16	36	28	28	100
wrdatab	342	194	158	153	96.84

Table 4.2: Fault Coverage for Scan Insertion based method

Benchmark	Number of Patterns	Number of Faults	Testable Faults	Number of Faults(D)	Fault Coverage (%)
chu133	5	34	34	28	84.85
chu150	8	48	44	44	100
converta	40	62	60	60	100
dff	49	52	50	50	100
ebergen	85	80	80	80	100
half	34	40	40	40	100
hazard	49	56	56	56	100
master-read	242	186	180	179	96.76
mmu	192	151	139	137	91.95
mp-forward	40	72	72	72	100
mr1	298	192	192	192	100
nak-pa	30	94	94	94	100
nowick	10	44	44	44	100
ram-read-sbuf	54	102	102	101	99.02
rcv-setup	25	26	26	26	100
rpdft	38	47	47	47	100
sbuf-ram-write	100	132	132	132	100
sbuf-send-ctl	117	114	114	114	100
sbuf-send-pkt2	127	128	124	122	96.03
seq4	110	138	138	138	100
seq_mix	128	158	154	152	97.44
trimos-send	254	181	181	181	100
vbe5b	38	56	56	56	100
vbe5c	44	58	54	48	87.93
wrdatab	243	184	184	183	99.46

Testify and Tetramax. It should be noted that for the same benchmark circuits the number of nodes accounted in the fault list varies. Since the full scan method uses extra pins for the scan-in and scan-out processes, the number of nodes will be higher. One interesting point to note here is that for some benchmarks the number of nodes accounted for full scan is equal to or less than the symbolic method. This is due to the fact that those benchmarks does not have the memory elements present in them. For example, benchmarks such as rpdft and rcv-setup will have almost the same number of nodes in the fault list for both methods. In Figure 4.7, the number of faults for these two benchmarks is lower for full scan compared to the symbolic method. This is because, the Tetramax tool reports the collapsed fault list while the Testify does not use any collapsing.

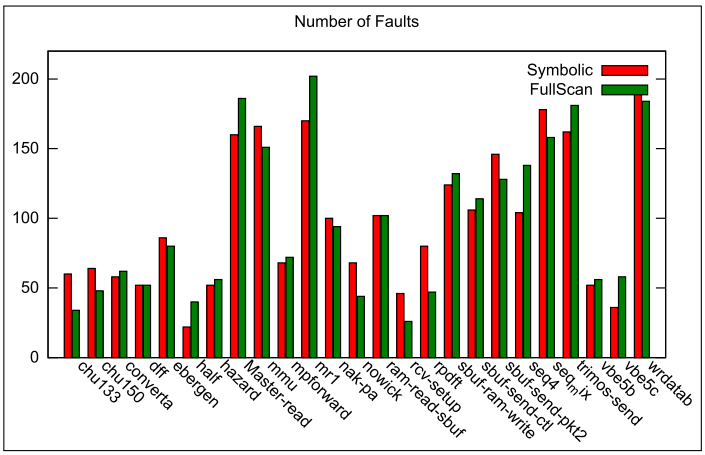


Figure 4.7: Total number of Faults - Symbolic versus Full Scan

Total faults Vs Testable Fault

From the fault coverage definition, the ratio of the number of faults detected to the number of testable faults is obtained as the metric for the testability of the circuit. Hence the actual total number of faults that may occur in the circuit is different from the total number of faults in the fault list. This is illustrated in Figure 4.8 for the Testify tool and in the Figure 4.9 for the Tetramax tool. As shown in Figure 4.8, the actual number of faults/nodes in the circuit is always higher than the number of faults considered for test. The legend "Testable(sym)" gives the total number of testable faults and the legend "Total(sym)" gives the total number of faults/nodes in the circuit simulated by the Testify tool. For example, the circuit "wrdatab" has 194 faults in total and the total number of testable faults considered for test generation was 158. Some of these faults are the electrically equivalent faults, while others are the feedback nodes. Similarly, in Figure 4.9, the comparison between total number of faults with the total number of testable faults is shown. The legend "Total(full)" gives the actual number of faults in the circuit and the legend "Testable(full)" gives the total number of testable faults. It should be noted that for most of the benchmarks the number of total faults and the number of testable

faults are almost equal or closer. Only circuits with drastic differences are mmu, mr1 and master-read. From these two figures (Fig. 4.8 and Figure 4.9), it is quite evident that Testify dropped a number of nodes from the testable fault list. This factor will affect the fault coverage metric eventually.

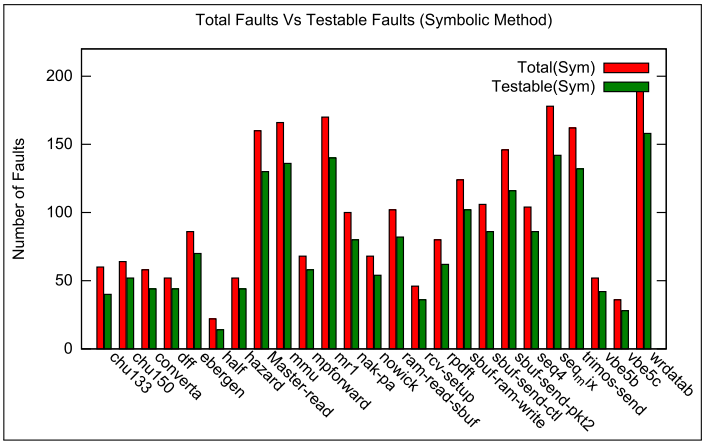


Figure 4.8: Total Faults Vs Testable Faults - Symbolic Method

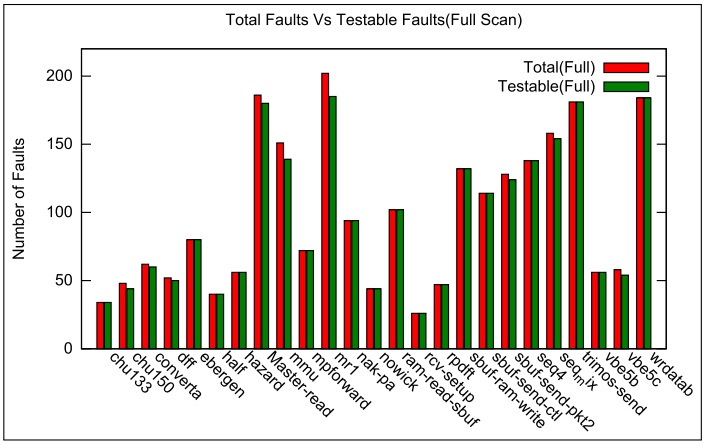


Figure 4.9: Total Faults Vs Testable Faults - Full Scan Method

4.4.2.2 Fault Coverage

Next we will compare the fault coverage of these two methods are discussed. Figure 4.10 gives the comparison between the fault coverage of the symbolic method and full scan method. For most of the benchmarks experiments, the fault coverage was the same for both these methods. It should be noted that although the full scan method considered more fault sites compared to the symbolic method and yet it has the same fault coverage percentage for most of the benchmarks. For the benchmarks chu133, master-read, mmu, sbuf-send-pkt2 and vbe5c, the full scan method had a lower fault coverage. For the benchmarks mr1, sbuf-send-ctl, seq_mix,

trimos-send and wrdatab, the full scan method had the higher fault coverage. The full scan method takes into account more fault sites than the symbolic method and gives higher or almost same fault coverage. Thus the full scan method detected higher number of faults.

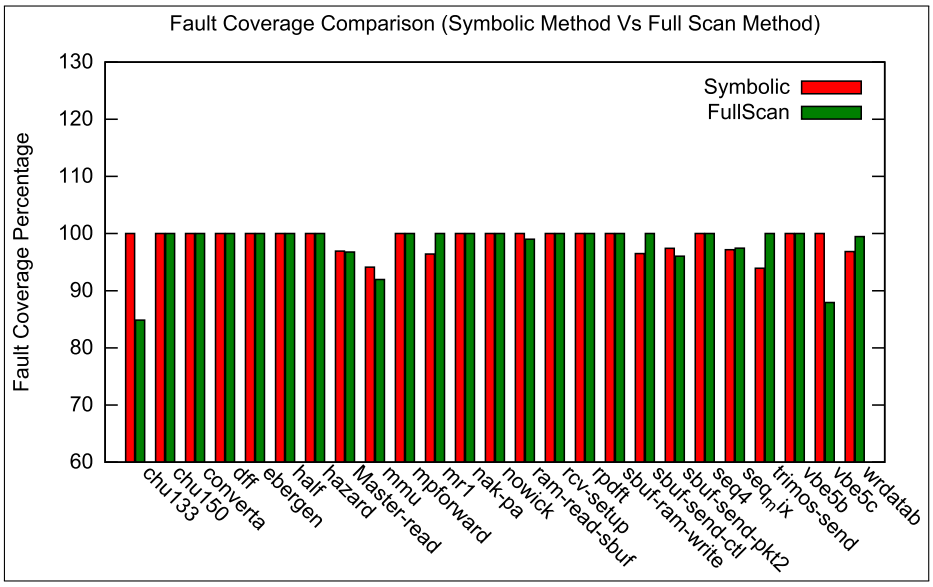


Figure 4.10: Fault Coverage Comparision - Symbolic Vs Full Scan

4.4.2.3 Number of Test Patterns

Finally, we compare the number of patterns generated by these two methods. It should be noted that the symbolic method generates pattern by enumerating the State Graph. However, for the full scan method, the test patterns are generated by the ATPG algorithm underlying the Tetramax tool, which enumerates the actual circuit nodes. Fig. 4.11 shows the comparison between the total number of test patterns generated by these two methods. The legends "Symbolic" and "Fullscan" gives the total number of patterns generated by the symbolic and the full scan methods, respectively. Since the full scan method uses scan latches, the number of patterns generated by this method is expected to be higher. However, for some benchmarks namely mmu, ram-read-sbuf, seq4, seq_mix and wrdatab the symbolic method generated higher test pattern than the full scan method. All these benchmarks had more C-elements present in them. For example, the benchmark "wrdatab" had 7 C-elements. On other hand, the benchmark "trimos-send" had 8 C-elements, but the full scan method produced more patterns compared to the symbolic method. But the number of testable faults detected were 182 collapsed faults for fullscan and only 132 for the symbolic method for this benchmark.

The difference in the total number of faults compared to the previous approach is attributed to two factors; the addition of scan latches, which increases the number of primary inputs and fault sites, and fault collapsing applied by the Tetramax tool in Full scan method. Test

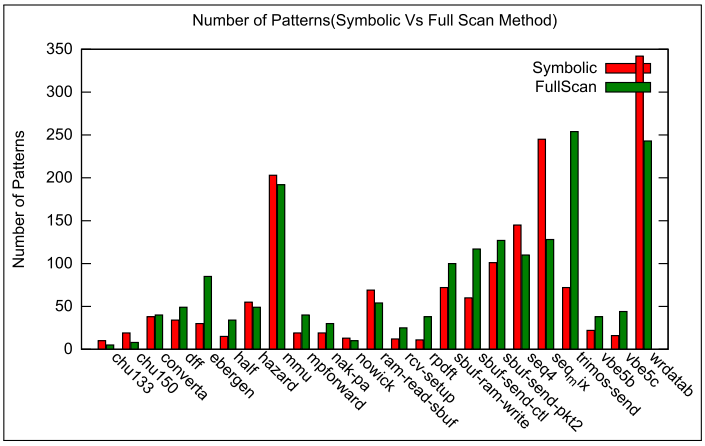


Figure 4.11: Comparision of Test Patterns - Symbolic Vs Full Scan

pattern generated using the symbolic method seems to be expensive in terms of number of test patterns and provides lower fault coverage than the full scan method. Also it generates longer test vectors compared to that of scan insertion approach. With the increase in test vector and number of pins the test patterns can be further reduced by using partial scan design instead of the full scan. It also reduces the area overhead due to these scan latches. Another advantage of the fullscan approach is that currently available synchronous test pattern generation tools can be used to generate test patterns, thereby makes this approach for testing asynchronous circuits feasible on industrial scale.

4.4.3 Factors affecting the fault coverage

Currently available ATPG tools such as Tetramax, detect the feedback paths and add the corresponding nodes to ATPG untestable faults list. Hence no effort is spent in the beginning of the test generation algorithm for creating test for these feedback nodes.

How does conventional fault simulation analyse faults?

Conventional fault simulation first assigns the test patterns to the corresponding primary inputs of the DUT. Based on these patterns all the nodes in the circuit are assigned with proper justification and propagation. The node values propagated to the primary outputs will be the same as the output pin vectors in the test pattern. After the good machine simulation, the faulty machine simulation for a particular fault is carried out. First, the simulator assigns logic 1/0 to the node to be tested for stuck-at faults 0/1, respectively. Then the primary input patterns are applied to the primary inputs to justify and propagate the logic values through all the other nodes in the circuit. The faulty value at the node being tested will also affect the justification and propagation process. As a result, the output pattern obtained will vary from that of the good machine. When this happens, the simulator considers that the test pattern has detected

the fault and reports it as detected. This works well for synchronous circuit testing, but for asynchronous circuits several more factors come in to play during the fault simulation.

Those factors affecting the testability/fault coverage of the asynchronous designs are listed below:

- Feedback paths present in the circuit
- Type of logic used for simulation
- Fault Simulator used
- Fault Collapsing
- location of the nodes in front of or after the C-elements
- Depth of the node in the circuit
- Observability of the nodes
- Controllability of the nodes

Conventional circuit structures affecting the test process:

Differences in the circuit structure of the asynchronous circuits compared to the synchronous circuits has a major impact on the test pattern generation. The feedback lines and the fanouts originating from those feedback lines make the circuit difficult to be testable. Since the synchronous circuit design representation always deals with acyclic/ loop free designs, most of the CAD algorithms developed were based on acyclic graphs and data structures aiding them. So there are no or very few methods using cyclic graphs for the test generation.

Type of logic used for simulation:

Representation of the node/line values in the circuit for logic simulation is another factor that affects the test pattern generation. Multi-valued logic had been used to represent the hazards in the circuits during the logic simulation. This eventually increases the effectiveness of simulating the asynchronous circuits which encounters the hazards and oscillations often. Often, 3 valued, 6-valued and 9-valued logics were used for asynchronous circuit logic and fault simulations.

Fault Simulator:
Although the fault simulator used for synchronous circuits enable the fault simulation for asynchronous circuits, several updates have to be made to effectively simulate them. For example, conventional fault simulator first generates the fault list of the circuit under test. During the fault list generation, feedback lines and the fanouts originating from them are omitted. This is due to the fact that the synchronous circuits do not have feedback lines. Thus by not adding

these lines/nodes to the fault list, the fault simulation can be carried out without interruption and the fault coverage can be obtained. But the fault coverage reported (even though higher values can be achieved) will not reflect the original fault coverage of the circuit under test. Secondly, the fault collapsing considered during the fault list generation and fault dropping phase will affect the fault coverage. This is discussed next.

Fault Collapsing:
Two issues namely "Fault Dominance" and "Fault equivalence", also affects the fault coverage reported by any ATPG tool. By definition, a test pattern for one fault is said to dominate another fault, if a subset of the test pattern of the latter detects the former. Two faults are considered to be equivalent, if the test pattern of one fault also detects the test pattern on another fault. These fault collapsing steps were not carried out in the first method based on STG discussed in this section. In [SM04b], fault dominance and fault equivalence were considered for asynchronous circuits and are different from the conventional definitions for the synchronous circuits.

Other factors, such as controllability and observability of the intermediates, are affected by the location of the C-elements in the circuits.

Changes required for asynchronous design:
In order to improve the testability of asynchronous designs, the following issues have to be addressed:

- Logic level simulation should be changed
- New method for realizing and simulating the feedback cycles should be developed
- The issue of whether feedbacks and oscillations need to be simulated during fault simulation should be addressed
- How does the fault on the feedback node affects the good and faulty machines, respectively?

4.5 Conclusion

A comparative study of two methods of test generation of asynchronous circuits namely, the Symbolic method and the Full scan method, was carried out on a set of representative benchmark circuits. The analysis of the results gave insights into factors affecting the testability of the asynchronous circuits. The drawbacks identified are considered for proposals for improvement of the new test generation methods presented in this thesis.

Chapter 5

ABALLAST-Asynchronous Circuit Test Generation based on Balanced Structures

5.1 Introduction

The first gate level test generation method proposed in this thesis is introduced, which uses cyclic to acyclic circuit conversion, partial scan based test generation and BALLAST methodology [GB90].

5.1.1 Problem statement

BALLAST methodology [GB90] of generating tests for sequential circuits is a promising approach for partial scan based test generation of synchronous sequential circuits. The main technique used in this method involved generating a balanced graph kernel from the circuit topology graph of the sequential circuit which was demonstrated to have equivalent combinational structure when the memory elements in the kernel are replaced by wires. Thus the test patterns for the sequential circuits are generated by treating them as combinational equivalent. The same technique can be applied to the asynchronous sequential circuit to generate tests. The challenges faced by applying this technique to the asynchronous circuits are:
1) Asynchronous circuits have both combinational gates and memory elements which makes them cyclic, whereas BALLAST method operates only on synchronous cyclic circuits with memory elements in each cycle; 2) Balanced kernel consists of memory element other than latches, whereas C-elements frequently appear as memory elements in asynchronous designs.

These elements constitute the local loop in the circuit; 3) The operation of the C-element cannot be controlled during its normal operation as compared to normal latches which are controlled by the clock.

Any proposed method to generate the test patterns efficiently for these circuits should address these three issues.

5.1.2 Motivation

The motivation underlying this work originates from the partial scan test generation method developed for synchronous circuits calle BALLAST methodology. Using the BALLAST partial scan methodology and cyclic-to-acyclic circuit conversion together, the ABALLAST methodology for partial-scan testing the asynchronous circuits is developed.

The contributions of this method are:

- Effective handling of the cyclic asynchronous circuits to accommodate them in the usual synchronous test generation flow
- Partial scan element selection based on balanced sequential structures
- Automatic Test pattern generation for the partial scan design generated

The chapter is organised as follows: Section 5.2 outlines some background information on partial scan test generation method and the BALLAST methodology; Section 5.3 describes the algorithms proposed in the test methodology; Section 5.4 detailss the algorithmic basis of the test methodology; Section 5.5 gives a working example of the proposed method and analyses the result obtained by applying this method to the asynchronous sequential circuits; Section 5.6 concludes the chapter.

5.2 Background

5.2.1 Cyclic and Acyclic Circuits

In this chapter, for cyclic to acyclic conversion, a circuit is represented by a Circuit Topology graph (CTG), where the nodes of the graph form the gates in the circuit and the arcs form the connection between the gates. Acyclic circuits are circuits comprising only of feed forward paths, where the output of one gate is fed to the input of the next gate and so on. Cyclic structures occur in asynchronous circuits due to the presence of local and global loops, due to feedback and feedforward paths. In these circuits, either the output of the gate is fed back to

its input or the output of the other gate in the forward path is fed to its input. The former case is called local loop and the later is called Global loop.

5.2.2 Loops in circuit

The CTG of a circuit will contain cycles due to the occurence of loops in the circuit. Some cycles occurring in the CTG are further studied here. Nested cycles and intersection of cycles are most commonly found structures. Nested cycles are formed when there is a self/global loop present inside the global loop of the circuit. Intersection of cycles is formed when a forward path of one global loop is fed to the gate in another global loop.

5.2.2.1 C-element

A majority gate based C-element is shown in Figure 5.1. The circuit is cyclic and consists of four gates and two feedback loops. The corresponding acyclic circuit is shown lower in the Figure 5.1. In this example, the number of copies of the feedforward path of the loop is taken as three, assuming the loops stabilize in three cycles.

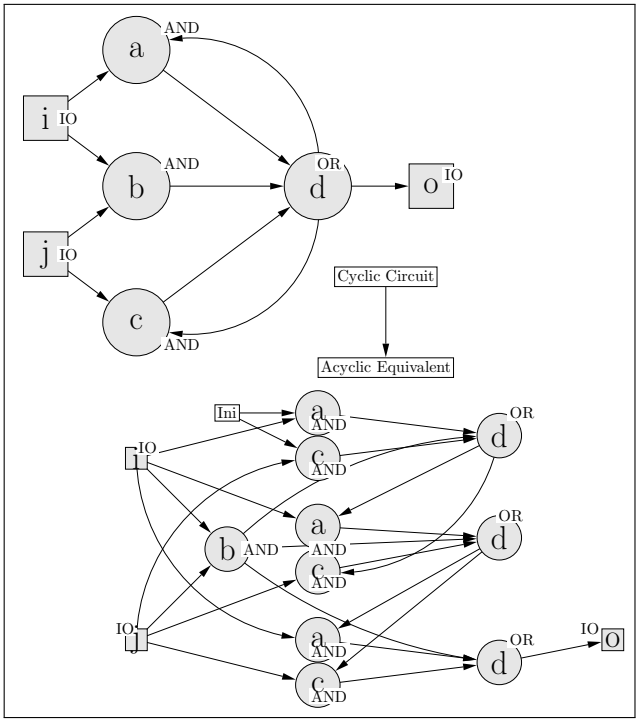


Figure 5.1: C-element - Cyclic to Acyclic Conversion

The equivalent acyclic circuit consists of 3 inputs, 1 output and 11 gates. Additional input formed in this circuit is the initialization input for the first copy of the forward path.

5.2.2.2 Benchmark "half"

The benchmark circuit "half" shown in Figure 5.2 consists of 14 gates and 4 feedback loops. The corresponding acyclic circuit consists of 4 inputs, 2 outputs and 52 gates. Additional input formed in this circuit is the initialization input for the first copy of the forward path.

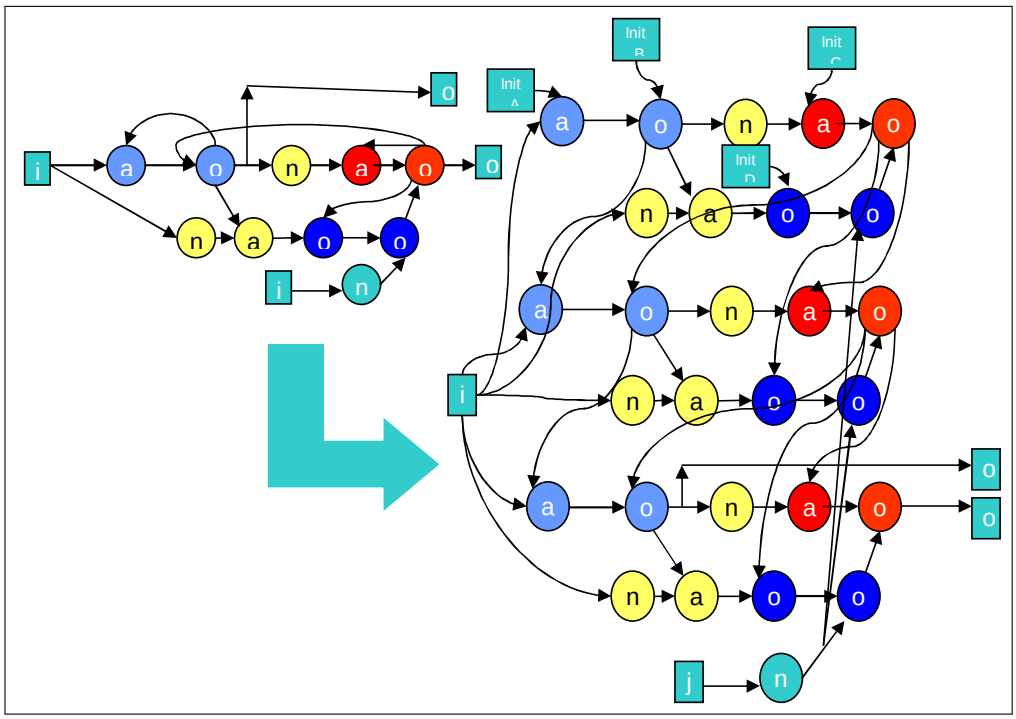


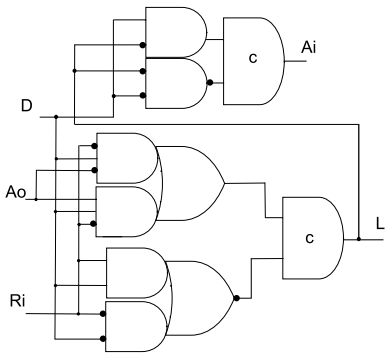
Figure 5.2: Benchmark "half" - Cyclic circuit and equivalent acyclic circuit

5.2.3 BALLAST

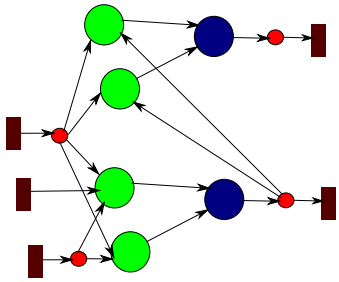
5.2.4 Circuit topology

The circuit topology used in the ABALLAST algorithm (detailed in Section 5.3) is shown in Figure 5.3. To convert the given circuit to the graph, all the elements of the circuit are classified as one of the following: combinational node, memory element, fanout node, or PI/PO node. The conversion of a circuit to the shown circuit topology involves specific rules [GB90], such as the following: all the combinational nodes fed by the same fanout nodes and PI/PO nodes can be grouped in to a single cloud [GB90]; two clouds connected consecutively can be merged; all the memory elements fed by the same clock can be grouped together while ensuring each element is fed by exactly one cloud, (in case of higher number of memory elements each group forms a register); no two memory element/registers can be connected consecutively. Figure 5.3(a) shows the circuit for the benchmark "chu150" with the equivalent general graph

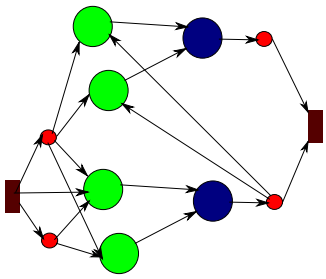
G(V,E) shown in Figure 5.3(b), where V forms the nodes of the graph and E are set of edges between the nodes. All the nodes of the graph correspond to the gates of the circuit and the edges correspond to the connection between the gates. Fig 5.3(c) shows the graph with all the PI/PO grouped, with them grouped as a single cloud in Fig 5.3(d). Figure 5.3(e) shows the grouping of two combinational nodes into another cloud, as they are fed by same input signals. In Figure 5.3(f) two other combinational nodes are grouped to form another cloud. These clouds are separated as the top cloud and are fed by the memory element. Fig 5.3(g) shows the arrangement of clouds and memory elements from the left to the right. The abstract view of the equivalent graph obtained without the fanout nodes is shown in Fig 5.3(g). If the clouds are converted to a set of nodes V and memory elements between them are converted to set of arcs A, the resulting Graph G(V,A,w) forms the topology graph on which ABALLAST algorithm can be applied. Set A can be partitioned further into (A-H,H), if memory elements are present with "hold" functionality. "w" is the weight of the arc based on the number of memory elements in it, when it represents a register.



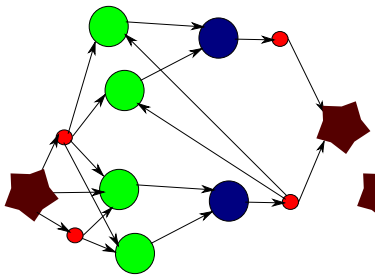
(a) chu150 Benchmark



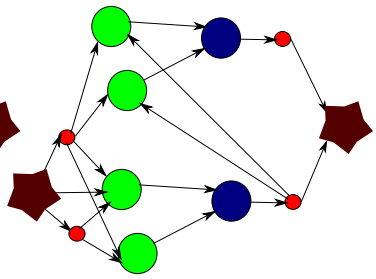
(b) Equivalent nodes with gates and fanouts as vertices



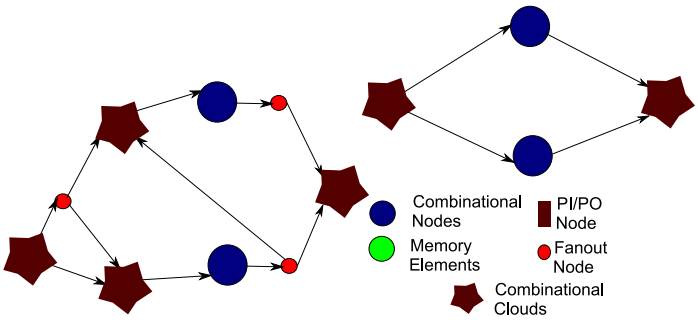
(c) I/O converted to Pseudo I/O node



(d) Pseudo I/O added to cloud



(e) Common gates merged into a single cloud



(f) Common gates and fanout merged into a single cloud

(g) Abstract view without the fanouts

Figure 5.3: Circuit to graph conversion for ABALLAST algorithm shown in (a), (b), (c), (d), (e), (f), and (g), respectively

5.2.4.1 BALLAST Method

In [GB90] a synchronous circuit model is represented by blocks of combination logic connected with each other either directly or through a register, which is a group of flip flops. A circuit structure S is given by G = (V,A,H,w), where G is the graph forming the circuit, V is the set of nodes in the graph representing the blocks of combination gates, A is the set of arcs between the nodes representing the register or the direct connection between them, H being

the set of arcs representing the "Hold" registers or "Scan" registers and w is the number of flip-flops in the register. A sequential circuit structure S with the circuit topology G is said to be balanced if, G is acyclic, all the directed paths between each nodes in the graph are equal and should an arc from the set H be removed, then the graph becomes disconnected. An example of a balanced circuit structure and the partial scan circuit generated by this method is shown in Figure 5.4 and Figure 5.5, respectively. The circuit topology/representation described in the previous subsection was used in this method which was introduced in [I.P94].The steps involved in test generation based on BALLAST methodology are as follows:

- Represent the circuit topology as a graph
- Make the graph acyclic and add the edges removed to the scan set
- Balance the resulting acyclic graph and add the edges removed during balancing to the scan set
- Generate test for the balanced circuit

To illustrate the BALLAST test flow, a simple example based on the same abstract circuit shown in Figure 5.5 is considered in Figure 5.7. In Figure 5.7.a, the circuit before the application of partial scan selection is shown. There are 6 registers in this circuit, marked by boxes and four combinational clusters shown as clouds. The primary inputs and primary outputs as clusters in the form of a black box. The equivalent graph representation is shown in Figure 5.6. The graph contains two feedback edges, which are removed and their corresponding registers are added to the scan set according to step 2. The resulting equivalent circuit is shown in Figure 5.7.b with two boxes denoting "scan". Step 3 involves balancing the remaining circuit called "kernel" marked enclosed in the box. Since the kernel in this example is already balanced (details on balancing is discussed in section 5.4.4 and is detailed in [GB90]), the procedure moves to the next step. In the fourth step of test generation, the non-scan registers are converted into wires (locations are marked as crosses in Figure 5.7.c) which is the combinational equivalent of a register/flipflop, when the clock is high. The resulting circuit is shown in Figure 5.7.c. In this way, the test patterns are generated for this circuit which will test the original partially scanned synchronous circuit in Figure 5.7.b.

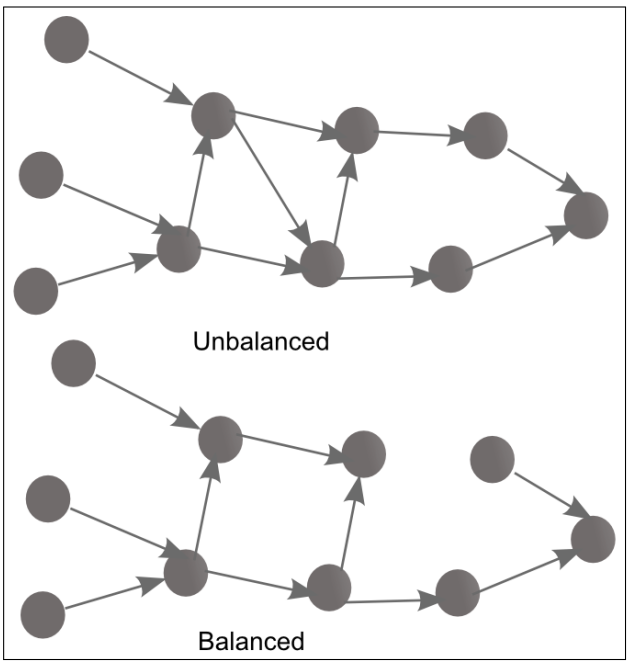


Figure 5.4: Unbalanced and Balanced structures

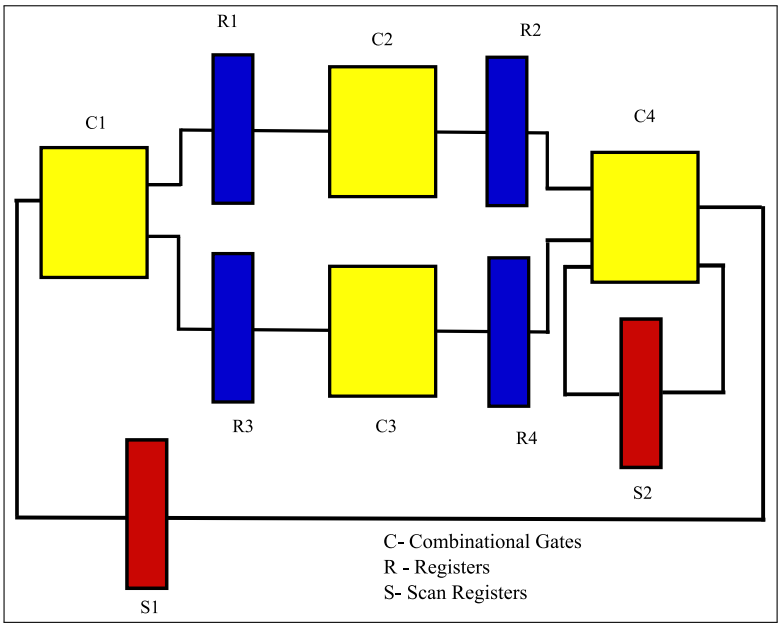


Figure 5.5: Partial Scan Circuit using Balanced structures

Limitations of BALLAST for Asynchronous Circuits

When the suitability of the BALLAST method is explored for asynchronous circuits, the following limitations were encountered:

- Should C-elements be represented as registers or be added in the combinational cloud?

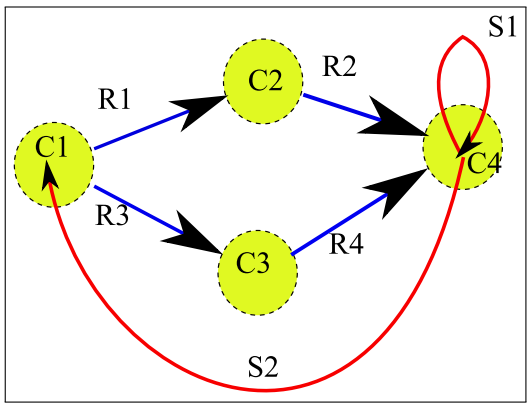


Figure 5.6: Example Circuit - Graph Representation

- What is the scan equivalent of the C-elements?
- What is the combinational equivalent of the C-element?

To illustrate these limitations a comparison is made on how BALLAST will handle the synchronous circuit and the asynchronous circuit. The comparison is shown in Figure 5.8: the figures on the left-hand side gives the synchronous BALLAST flow, and the figures on the right-hand side give the BALLAST flow of the asynchronous circuits. As shown in Figure 5.8.d, when converting the circuit into the graph structure, the representation of the C-elements into registers or into combinational gates is not addressed in this method. Even, when they are considered, as the registers and the partial C-elements set is chosen following the same flow, conversion of the scan equivalent of these C-elements are not shown, as this method was designed for synchronous circuits. Finally, the combinational equivalent of the C-elements that are not scanned in the kernel are also another concern when generating test patterns based on this method.

These questions form the motivation for the development of the ABALLAST method which is derived as an extension to BALLAST. How these questions are answered and the test flow of Asynchronous BALLAST (ABALLAST) is described in next section.

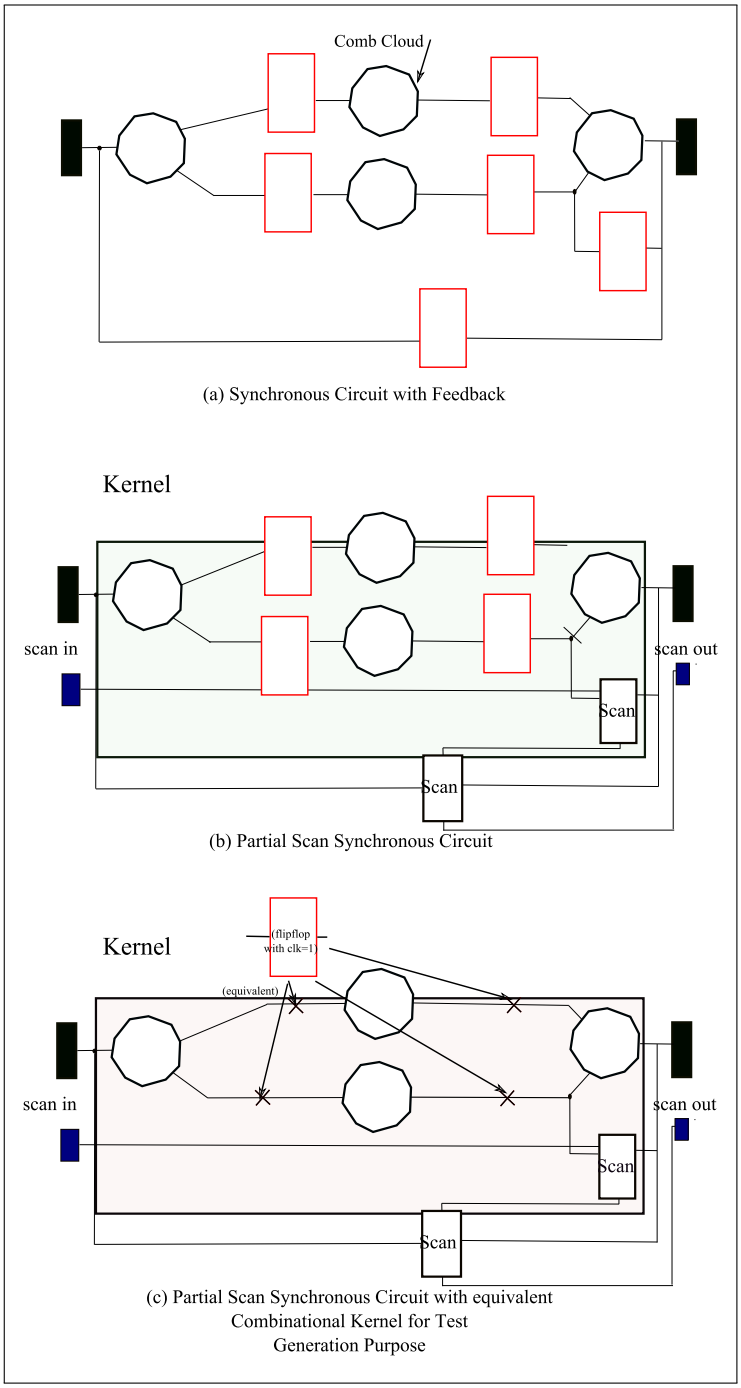


Figure 5.7: BALLAST Method Example

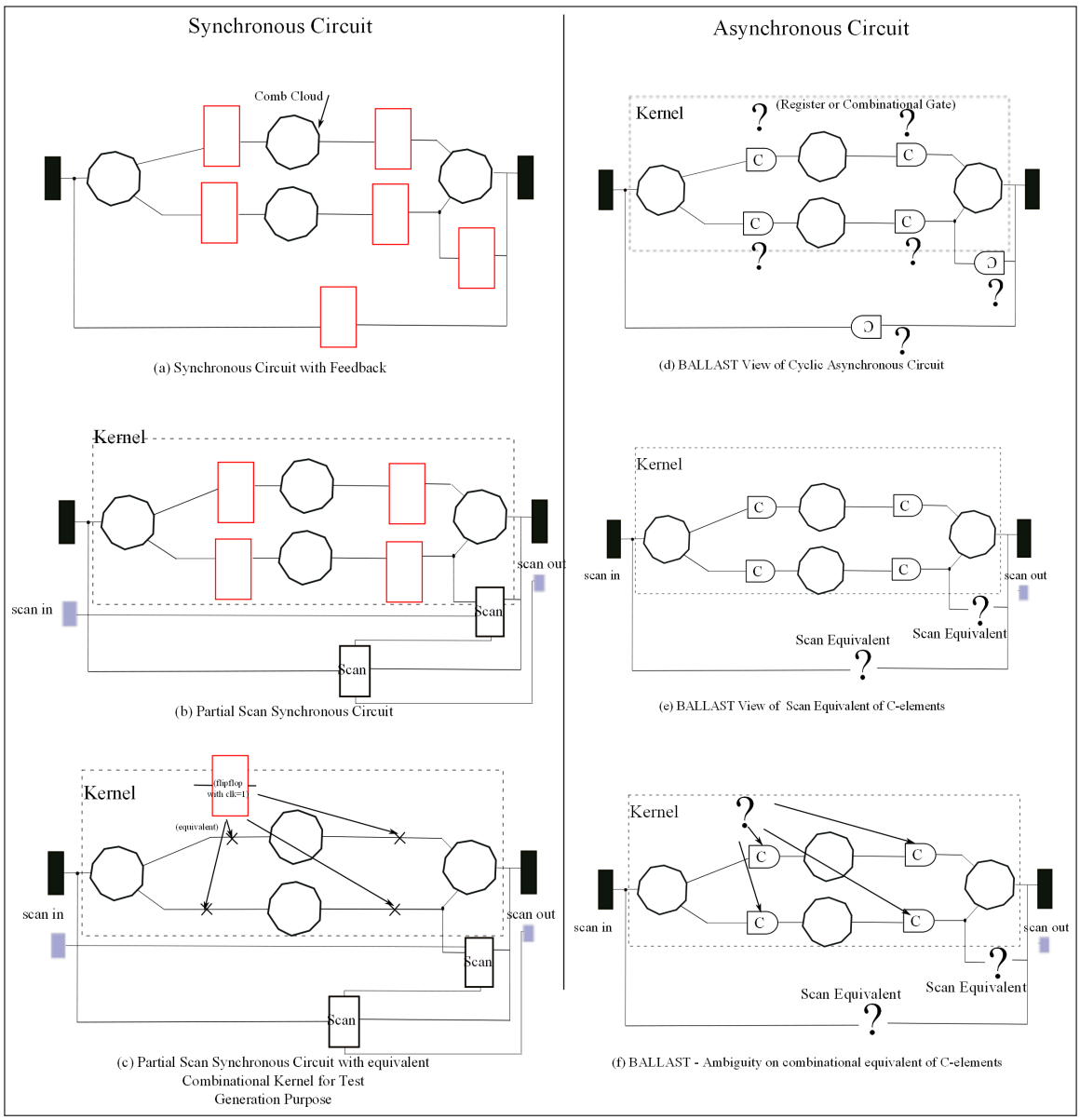


Figure 5.8: BALLAST Method on Synchronous and Asynchronous Circuits

Further details on balancing the graph structure is dealt in detail, when describing the test methodology in the next section.

5.3 Test Methodology

This section describes the overall test flow of the ABALLAST methodology as illustrated in Figure 5.9. There are two main stages in the test flow:

1. A partial scan DFT methodology based on BALLAST is applied to the circuit in order to improve its testability.

2. The circuit is further transformed into a fully acyclic circuit only for the purpose of generating test vectors using conventional ATPG tool. The generated vectors are then slightly modified and applied to the DFT circuit produced in the first stage.

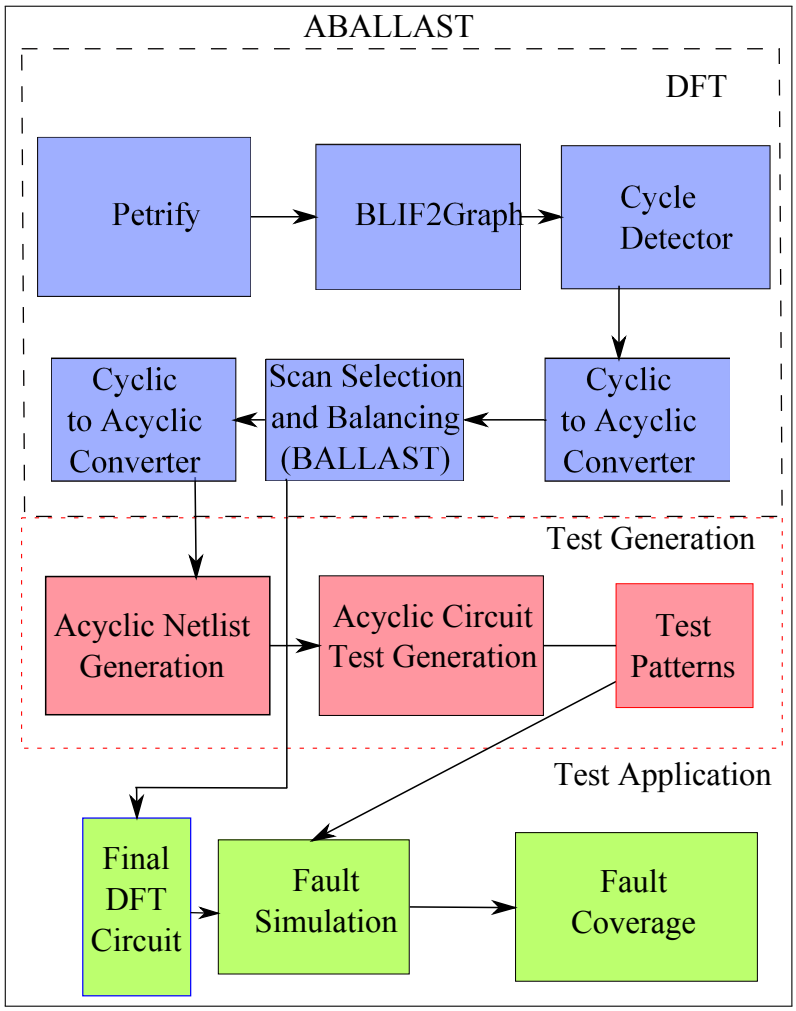


Figure 5.9: Test Methodology

The steps involved in this method are listed and explained below:

Convert the netlist into a suitable graph representation

The first step is to convert the circuit synthesized by Petrify into a graph representation. This is done by using the tool BLIF2Graph, which parses the BLIF (Berkeley Library Interchange Format) netlist into a graph where the vertices correspond to the circuit gates and edges to the wires connecting the gates. However, BALLAST requires a representation where the elements (C-elements and latches) are the edges and all interconnections and combinational gates are modelled as vertices (clouds). This high level extraction is performed by ACLARION which

was introduced in the previous chapter.

Detect and break global loops

The BALLAST algorithm cannot process a circuit with loops, so these should be broken. Note that, at this stage, only the global loops are exposed to the algorithms; local loops, such as those inside memory elements, are hidden as each memory element is considered a single gate.

The extracted graph from the previous step is processed by the CycleDetector to find all the cycles in the graph. This is carried out by detecting all the “backward” edges in the graph.

The back edges are determined using the GR algorithm explained in algorithm 12. Once the loops are determined, the cyclic to acyclic conversion is straight forward: the back edges are simply removed. Since graph edges correspond to C-elements, an edge removal means that the corresponding C-element is converted into a scanned C-element. For example, consider the graph shown in Figure 5.6 of the circuit shown in Figure 5.5. The straight arrows with larger head correspond to the registers R1,R2,R3, and R4 in Figure 5.5. The dotted circles/vertices corresponds to the combinational clouds. The bent edges/arrows forming the loops are the registers S1 and S2, which will be removed by the cyclic-to-acyclic converter to make the graph acyclic and hence they are converted to Scan-C-elements. The resulting acyclic circuit (without the edges S1 and S2) will be balanced by the BALLAST algorithm.

Extract kernel using BALLAST. Produce partially scanned circuit

The BALLAST tool takes the acyclic graph and generates the balanced graph structure by removing some edges, if required. Any edges removed result in more C-elements being scanned. Thus in addition to the C-elements being converted into scan C-elements in the previous step, the C-elements returned by BALLAST are now converted into scan C-elements. The result of this step is the final circuit containing Design-for-Test (DfT) structures to aid testing. This is essentially the circuit to be fabricated and the test coverage, at the end of the test flow, is measured on it.

Detect local cycles and unroll them to generate acyclic circuit

Standard ATPG tools cannot produce test vectors for the circuits generated from the previous step because they still contain local loops inside C-elements. However, since this is a partial-scan method, the remaining C-elements are not scanned in order to keep the DfT area overhead

low. Instead a method similar to time frame unrolling [GB90] is used to convert these C-elements into their acyclic equivalents. Essentially the loop is unrolled a number of times and eventually an extra primary input is added.

As mentioned earlier, the circuits of type shown in Figure 5.10 will still contain local loops (the C-elements inside the kernel in Figure 5.10.b form these local loops). For this circuit, there are 4 C-elements left in the circuit which are not scanned.

This step converts all the non-scan C-elements into their acyclic equivalent by substituting them with acyclic (unrolled) instances. The resulting acyclic, balanced circuit is ready for processing by the test generator without it complaining about the presence of feedback loops. The graph description is now converted back to netlist (Verilog in this case) and can be directly sent to the test generator.

Generate test vectors for the resulting circuit

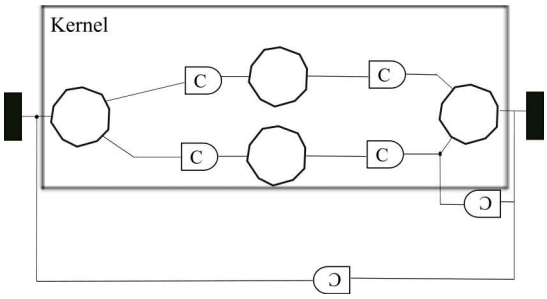
The acyclic netlist is fed into Synopsys's Tetramax to generate test vectors as shown in Figure 5.10.c.

Convert test vectors and fault-simulate partially scanned circuit

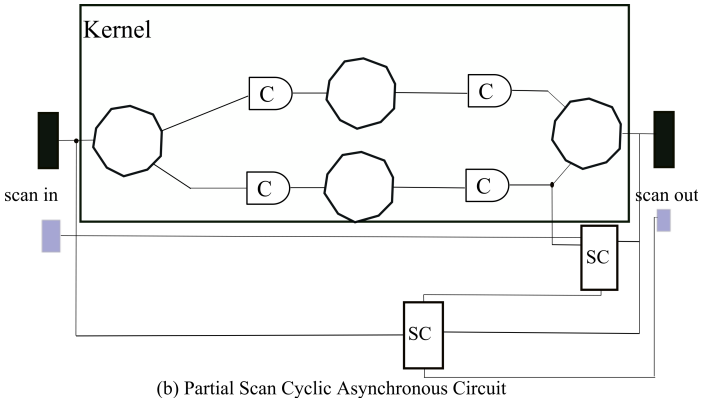
The length of the test vectors generated will be equal to the number of I/O pins of the final acyclic circuit which includes the initialization pins added when the local loops were broken. The actual DUT will not have these pins and hence the test vectors have to be trimmed by removing the bits which correspond to the initialisation pins.

The converted vectors are applied to the DUT and the fault coverage is obtained using a fault simulator (Synopsys's Tetramax) as shown in Figure 5.10.d.

Thus to summarize the whole methodology, the resulting equivalent partial scan circuit will be of the form shown in Figure 5.10.b which contains both non-scan and scan C-elements (named C and SC) respectively. The actual test pattern will be generated for the circuit in Figure 5.10.c, which has its non-scan C-element converted into acyclic equivalent (named "AC" in the shaded box named "Kernel" in Figure 5.10.c). The initialization pins for the acyclic C-elements are marked "ini" in Figure 5.10.c. The test patterns generated for this circuit is applied to the circuit in Figure 5.10.b, which is shown in Figure 5.10.d. The pseudoinput shown here is the input equivalent to "ini" in the acyclic equivalent circuit in Figure 5.10.c.

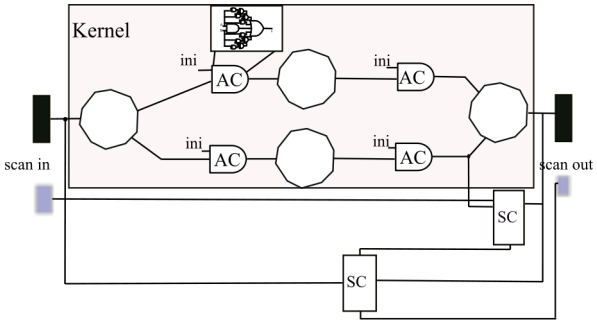


(a) Cyclic Asynchronous Circuit

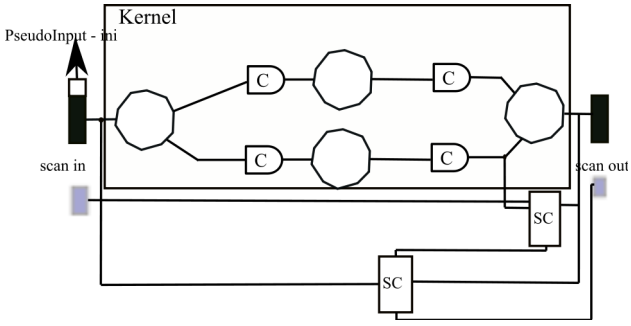


(b) Partial Scan Cyclic Asynchronous Circuit

(b) Partial Scan Cyclic Asynchronous Circuit



(c) Partial Scan Acyclic Asynchronous Circuit



(d) Partial Scan Testable Cyclie Asynchronous Circuit

Figure 5.10: Test generation for cyclic circuit with state holding element

5.3.1 Special Case - Cyclic circuits without C-elements

There is also a possibility that the benchmark will not contain C-elements and only have global loops with combinational gates as in the case of the abstract circuit shown in Figure 5.11. The dark circles denote only the combinational gates and no memory elements are present in this circuit. Thus the corresponding graph representation will be the vertices representing the combinational gates and edges representing the connections between them. As there is no C-elements present in them, it is not necessary for the circuit to go through the scan selection algorithm. For these circuits the circuit pre-processing for test pattern application ends at this stage and can be sent to the test pattern generator by converting them to equivalent HDL (Hardware Description Language) file.

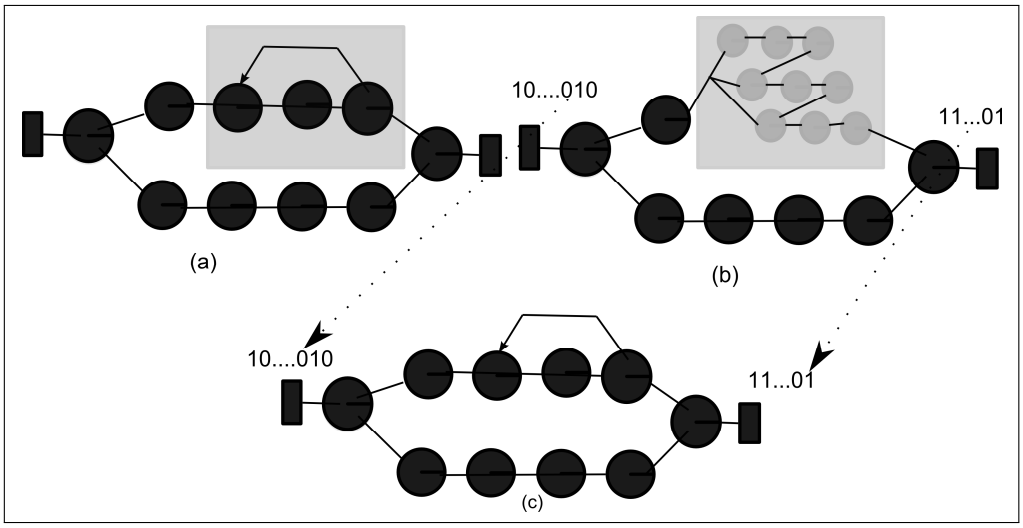


Figure 5.11: Test generation for cyclic circuit without state holding elements

5.4 Algorithms

The circuit model and the algorithms involved in cycle detection, cyclic-to-acyclic conversion and graph balancing are discussed further in greater detail.

5.4.1 Circuit Topology Description

The circuit C is represented as a directed graph G (V, E) | {V is the set of vertices of the graph, E is the set of edges}. The set of vertices, V, corresponds to the set of gates present in the circuit. The set of edges, E, corresponds to the connections between the gates. All types of

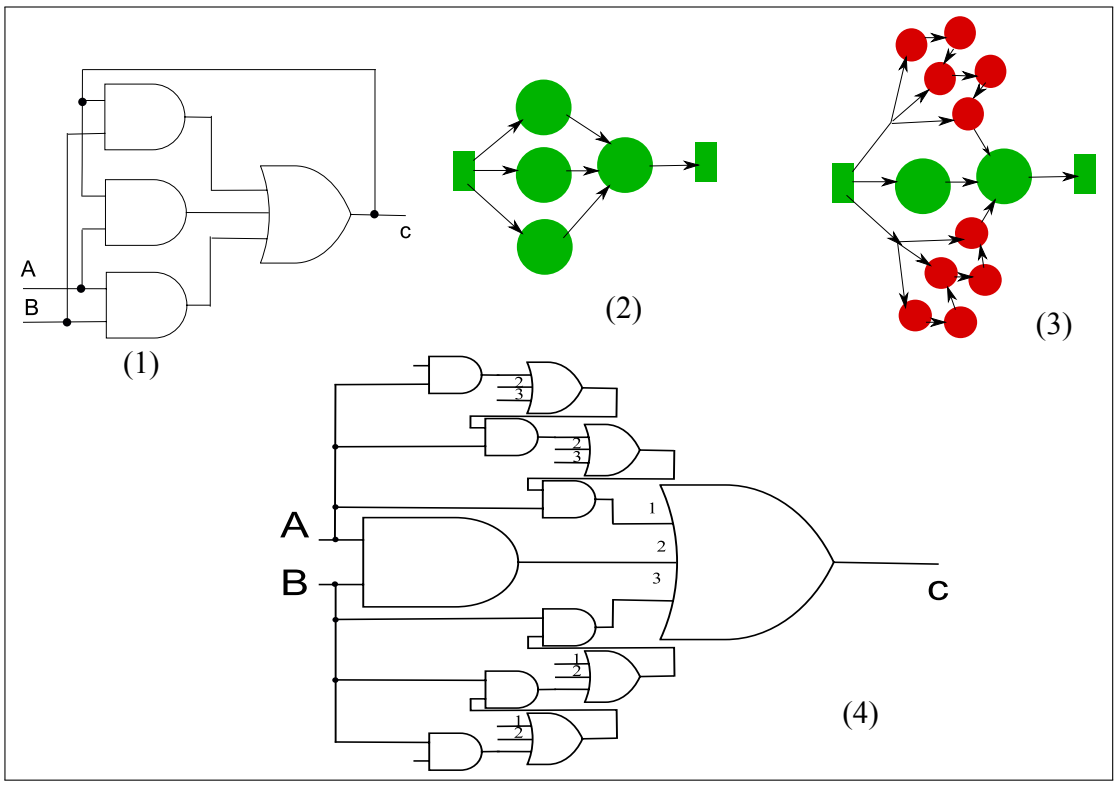


Figure 5.12: Cyclic to Acyclic conversion - C-element

gates (C-elements and other two input gates) and PI and PO are considered as a generic node in the graph.

5.4.2 Cycle detection

Cycle detection problem in a graph is equivalent to the problem of finding the feedback arc set in graph theory. Given a graph represented by G (V, E), with V representing the set of vertices and E representing the set of edges of the Graph G, finding the set of R(s) which forms the feedback arc sets of G is called Feedback Arc Set Problem. This problem is an N-P hard problem [Kar72] and was first studied in [Sla61]. The FAS problem being N-P hard problem is solvable in polynomial time for planar graph was shown in [Luc76],[LY78]. Finding FAS up to a size of $1/2\|E\|$using the heuristics in Figure 1 was shown in [BS90].

Algorithm 1 Heuristic for Finding Feedback Arc Set

$F = \emptyset$ **while** $G \neq \emptyset$ **do**
    select a vertex v in $G(v)$ **if** d- *(v)* $<$ d+ *(v)* **then**
        add all arcs incoming to v to F
    end
    else
        add all arcs outgoing from v to F
    end
    remove v and all arcs incident to it from G
end
return F

This algorithm runs in O (|V||E|) times, where |E| is the number of edges and |V| is the number of vertices. By exploiting vertex sequence ordering, a fast heuristic was introduced in [ES93]. In this method, all the vertices of the graph are ordered in sequence. When these vertices are placed in a horizontal line in this sequence, all the leftward arcs will form the feedback arc set of the graph. It runs in O(|E|) times. It has an asymptotic performance bound of r (G) <= m/2 – n/6, where r (G) is FAS of minimum cardinality. d(u) = d^+(u) + d^-(u) is the degree of the vertex u $\in$ V. d^-(u) is the indegree of the vertex and d^-(u) is the out degree of the vertex. The algorithm is shown in Algorithm 12 [ES93].

Algorithm 2 GR
input: G: DiGraph; var S: Vertex Sequence

```
1  s1 = ∅  s2 = ∅  while G ≠ ∅ do
2      while G contains a sink do
3          choose a sink u   s2 ← us2  G ← G- u
4      end
5      while G contains a source do
6          choose a source u  s1 ← s1u  G ← G-u
7      end
8      if G ≠ ∅ then
9          choose a vertex u for which del (u) is a maximum   s1 ← s1u   G ← G- u
10     end
11     S ← s1s2
12 end
```

The procedure GR computes two sequences s1 and s2 based on three types of vertices in the graph. Each node/vertex of the graph is removed from the graph and added to either s1 or s2.

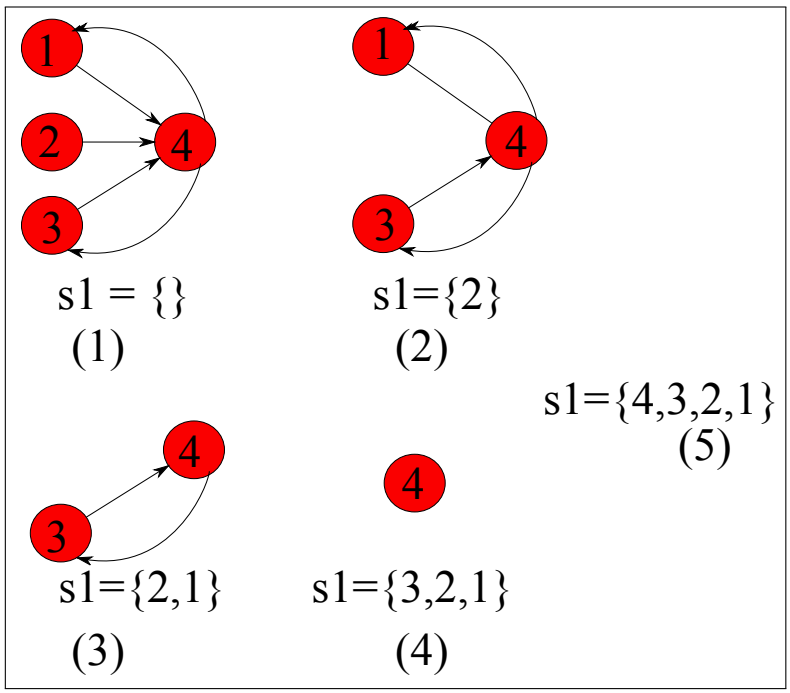


Figure 5.13: Graph traversal in GR procedure

If the removed node is a sink, then it is added to the sequence s2. If the node is a source then it is added to the sequence s1. When all the sinks and sources are removed from the graph, the remaining vertices are added to the sequence s1 in the descending order of δ(u)(where δ(u) = $d^{+}(u) – d^{-}(u)$). Once the graph becomes empty, both the sequences are concatenated to form a single sequence S.

Theorem 1: Algorithm GR computes either an empty vertex sequence or a vertex sequence s for which $R(s) \leq m/2 - n/6$.

The proof of the theorem is provided in [Edw03] or alternatively in [BS90]. An example showing the GR algorithm over the graph equivalent of C-element is shown in Figure 5.13.1 shows the graph equivalent of a C-element with four vertices and 5 edges. It has 1 source and 3 nodes with both d+(u) ,d-(u) != 0. Now the vertices are sorted based on the value of δ(u). δ(4) = -1,δ(1) = δ(3) = 0, and δ(2) = 1. By GR algorithm, the graph doesn't have a sink and hence it checks for sources. Source 2 is present and is added to the sequence s1. 2 is removed from G and the resulting G is shown in Figure 5.13.2.

Since the resulting graph does not contain any sink or source, the vertices are removed based on the value of δ(u). Vertices 1,3,and 4 are removed from the graph in the order as shown in Figure 5.13.3, 5.13.4, and 5.13.5. The sequence formed by concatenating s1 and s2 is 2134. When the vertices of the graph are placed in the horizontal line a set of rightward edges and a set of leftward edges are formed as shown in Figure 5.14. The leftward edges in the graph

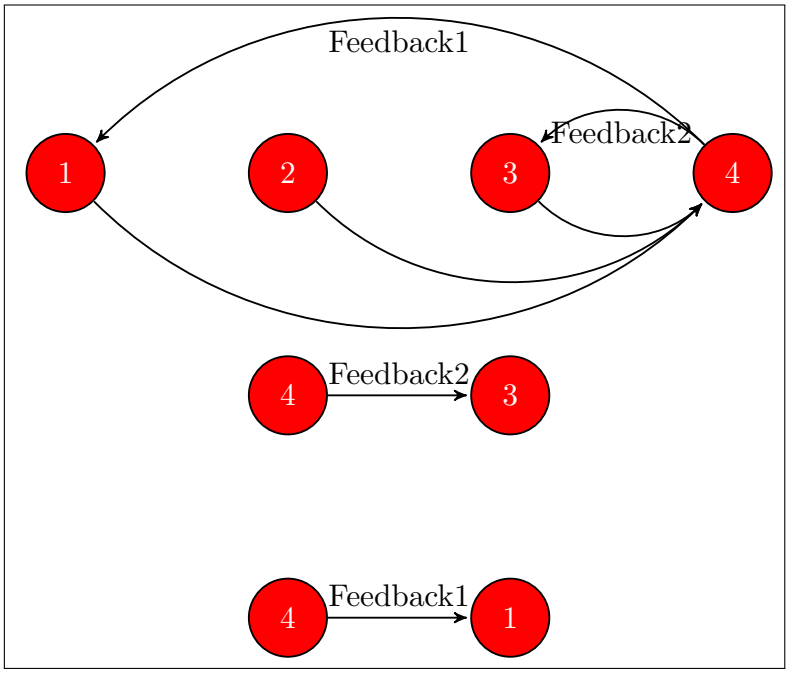


Figure 5.14: Ordered Vertex Sequence

constitute the feedback arc set, (R(s)). In this example, edges (4,3) and (4,1) form the feedback edges as shown in Figure 5.14 After the sequence is obtained, the R(s) of the sequence will be used by the cyc_to_acyc algorithm to generate the acyclic equivalent.

5.4.3 Cyclic to Acyclic Conversion

The steps for generating the acyclic circuits from the cyclic one are shown in Figure 5.12. The given circuit is first converted into a circuit topology graph (CTG). All the nodes correspond to the gates and all the edges corresponding to connections between the gates. The conversion algorithm requires user specified number of cycle copies. Cycle copies are equal to the number of cycles it takes for the circuit to stabilize.

Algorithm 3 Cyclic to Acyclic

Input numofcopy G (V,E) output Gc (V,E) `CycToAcyc(`G *(*V,E*)*,Gc *(*V,E*)*`)` {

Run Algorithm.2 on G (V,E) ;

Update list ”*Feedback_edge*” from Algorithm 2.

Ga (V) = G (V);

Ga (E) = `CreateVertices(`G *(*E*)*`)`

Ga (V,E) = `CyclePathDuplication(`Gc *(*V,E*)*,numofcopy`)`

Ga (V) = `CreateVertices(`Gc *(*V*)*`)`

Ga (E) = `CreateEdges(`Gc *(*E*)*`)`

Ga (V,E) = `ConnectIONodes(`G *(*V,E*)*,Ga *(*V,E*)*`)`

return Ga (V,E) }

`CreateEdges(`G *()*`)` {

$e1$ = G (E) **while** $e1$ **do**

  if $e1 \neq feedback\_edge$ **then**

    add $e1$ *to* Ga (E)

  end

end

return Ga (E)

}

`CyclePathDuplication(`Gc *(*V,E*)*,numofcopy`)` {

for numofcopy = 1 **to** numofcopy **do**

  $Vncopy$ = Gc (V); $Encopy$ = Gc (E) $v3$ = lastnode of $vncopy$ $v4$ = firstnode of $Vncopy$ +1

  $ec = v3 \rightarrow v4$ add ec *to* Gc (E) Gc (V,E) = Gc (V,E) + $Vncopy$ + $Encopy$

end

}

Algorithm 4 Cyclic to Acyclic Procedures

```
AddCycleVertices(Gc (V)) {

u2 = Gc (V)   while u2 do
    v2 = Ga (V)  while v2 do
        if v2 ≠ u2 then
            u2 → Ga (V)
        end
    end
end
return Ga (V)   }

AddCycleEdges(Gc (E)){ e2 = Gc (E)  while e2 do
    if e2 ≠ e1 then
        add e2 → Ga (E)
    end
end
return Ga (E)   }

ConnectIONodes((G (V,E),G (V,E)) {

for all input nodes i in G do
    if there is an edge e = G(E) with i as source then
        for j = 1 to numofcopy do
            add edge e = (i,Vncopy) rightarrow Ga (E)
        end
    end
end
for all output nodes out in G do
    if there is an edge e = G(E) with v as target then
        add edge e = (vncopy,out)→ Ga (E)
    end
end
return Ga (V,E)
}
```

A cyclic graph description G(V,E) and the set of edges, Ec, of the cycle graph Gc(V,E) obtained from the cycle detector forms the input for the cyclic-to-acyclic conversion. Acyclic graph Ga(V,E) is constructed by adding the vertices from the cyclic graph and adding only the feedforward edges. Vertex and Edge set in Gc(V,E) is added to the acyclic graph Ga(V,E) . The graph Ga (V,E) is then updated with the edges corresponding to the I/O nodes and the

forward path copies. To obtain this, the cyclic graph I/O nodes connection are compared with corresponding nodes in Ga(V,E) and the corresponding edges are updated. For example, if the first node of the cycle path is fed by an input node, then all the clone nodes are fed by the input node. In case of a node from cycle forward path feeding an output node, the clone node of the last forward cycle copy is fed to the output node leaving other clone nodes. This ensures that, only the final forward path node output is connected.

The resulting acyclic circuit will contain additional copies of the original gates depending on the number of cycles the circuit takes to stabilize. For example, if the circuit is assumed that it will stabilize in three cycles, the resulting acyclic circuit will have three forward path copies of the path in the corresponding cycle in the cyclic circuit. The number of copies will also depend on the number of cycles present in the original cyclic circuit. Thus the resulting graph forms an acyclic equivalent of a cyclic graph.

5.4.4 ABALLAST

ABALLAST algorithm (Algorithm 20) involves the generation of a balanced acyclic asynchronous circuit, which forms the asynchronous circuit with all the C-elements in the kernel converted into acyclic equivalent and the kernel being a balanced structure. BALLAST methodology was introduced in [GB90], where the balanced graph structures were used to select partial-scan flipflops for synchronous sequential circuit test generation. The circuit is represented as a graph G (V, A, H, w), where V, the set of vertices corresponds to the combinational blocks or clouds [GB90] in the circuit; E, the set of edges represent the registers between the clouds; H, a subset of A is the Hold registers; and w, being the cost of converting the registers to scan registers. A balanced circuit of the graph G, is given by G (V, A-R, H-R, w) where, R being the arcs removed from the graph to make the graph balanced. Registers in the set of removed arcs R will form the scan registers of the circuit and remaining register along with the clouds form the balanced structure or kernel. Here three steps are carried out to make the cyclic circuit to be testable: first, the circuit is checked whether it is acyclic; if the circuit is acyclic, then the procedure ‘balance’ is applied to make the acyclic circuit balanced; the procedure ‘check’ verifies whether the circuit is balanced.

‘

Algorithm 5 ABALLAST

input : G represented as set of edges, number of nodes, number of vertices

output: Kernel Graph Gk and Scan Element List R

```
13 ABALLAST(G (V,E)) {
14 Check(G (V,E))  if Success then
15 |  R = Balance(G (V,E))
16 end
17 else
18 |  return Failure
19 end
20 Gk (V,E) = GetKernel(G (V,E))  Gak (V,E) = CycToAcyc(Gk (V,E))  return Gk (V,E), R  }
```

Algorithm 6 ABALLAST Procedures

```
GetKernel(G (V,E)) { Check(G (V,E))  Rk = Balance(G (V,E))  for ∀ r ∈ Rk do
|  GK (V,E) = GK (V) - r
end
return GK (V,E)  }
}
```

Although, the circuit is balanced and the scan registers are found, the kernel obtained by using the check and balance procedures contain C-elements which only account for the local loops. In BALLAST, only synchronous circuits are used and hence the kernel is ready for generating test. But in case of ABALLAST, the cyc_to_acyclic algorithm is applied to the kernel again to convert the C-elements present in them to acyclic equivalent. Thus, acyclic equivalent of the kernel and the list of registers to be scanned are obtained as the output of this ABALLAST algorithm. It should be note that, in case of BALLAST method, only the list of registers to be scanned is obtained as output. This netlist will be used for test generation. The acyclic circuit will be then fed to the Synopsys 's Tetramax. The test vector generated for the acyclic circuit using Tetramax is now used to fault simulate the equivalent cyclic circuit.

5.5 Evaluation methodology

The proposed test flow was evaluated by applying it to a number of asynchronous benchmark circuits and comparing the results to 3 other methods with respect to fault coverage and DfT area overhead.

5.5.1 Choice of Benchmarks

The benchmarks selected for evaluation of the proposed method are taken from [SM04a]. These benchmarks are basically a latch controller and interface circuits commonly used in asynchronous circuits. The benchmarks provided in the above mentioned source is only in STG format. Hence the STG specifications are synthesized to gate level specifications using the Petrify tool. The gate library used for synthesizing the STG specifications is the inbuilt library in the Petrify tool.

5.5.2 Methods Evaluated

The proposed method was compared with the following existing methods: SPIN-SIM [SM04a], Eichelberger's method [Eic65] and the full-scan method [BA05].

5.5.2.1 Eichelberger's Method

In Eichelberger's method [Eic65], a novel method for detection of hazards in both combinational and sequential circuits was introduced. It was implemented as a program in [SM04a] and the fault simulation results were compared for 10000 random vectors.

5.5.2.2 SPIN-SIM

SPIN-SIM is a simulation-based test approach [SM04a] adapted from [Eic65] to integrate with the fault simulator for synchronous sequential circuit (namely HOPE) resulting in a fault simulation strategy for asynchronous circuits. Issues addressed in this method are 1) adaptation of Eichelberger's method , 2) Preserving relative Transition order, 3) Judicious time frame unrolling, and 4) handling complex gates

Some of the drawbacks of this method are: 1) pseudo gates are used for the C-elements, 2) C-elements are considered as a single gate and faults inside the C-element are not considered in the fault list, 3) Random vectors are used for fault simulation which counts to 10000 vectors, 4) Fault collapsing is based on the method introduced in the HOPE simulator (for synchronous circuits), and only a subset of the fault classes are used during collapsing.

5.5.2.3 Full Scan Method

This method [BA05] is a straight forward DFT method involving the replacement of all the memory elements in the design by their equivalent scan latch design. Using this approach

considerably increases the area overhead. Few partial scan methods have been proposed for asynchronous circuits testing and the method advocated in this thesis seeks to implement a new partial-scan method.

5.5.3 Metrics used for Evaluation

The metrics used to compare the different test methods are:

- Fault Coverage
- Scan Area Overhead
- Test Vectors

Before defining the metrics used in the evaluation, the fault classes used in Tetramax are introduced. The five main fault classes represented by Tetramax are:

- DT - Detected
- PT - Possibly Detected
- UD - Undetectable
- AU - ATPG Untestable
- ND - Not Detected

The subclasses of these faults are listed in Table 5.1, which are used to report the fault coverages. Most of the fault names are intuitive, and the detailed definitions can be obtained from the Tetramax userguide.

The definitions of the three metrics mentioned are defined next:

- Fault Coverage - This the ratio of the number of faults detected to the total number fault sites in the circuit. The fault coverage of the circuit therefore depends on the total number of detectable faults taken into account in the fault list. The fault coverage in the proposed method is calculated by the Tetramax tool. The fault collapsing, test generation and fault simulation steps are carried out by the Tetramax tool. The equation for the test coverage used in Tetramax is as follows.

$$Fault\ \ Coverage\ \ =\ \ \frac{DT\ \ +\ \ (PT\ \ \times\ \ PT_{credit})}{All_{Faults}}\ \ \times\ \ 100 \tag{5.1}$$

  "PT" in equation 5.1 stands for "Possibly Detected" fault, and "PT_credit" is set to 1.

Table 5.1: Fault Classes in Tetramax

Fault Classes		
DT detected	DR	Detected Robustly
	DS	Detected by simulation
	DI	Detected by Implication
PT:Possibly Detected	AP	ATPG Untestable-Possibly Detected
	NP	Not analyzed - Possibley Detected
UD Undetectable	UU	undetectable unused
	UT	undetectable tied
	UB	undetectable blocked
	UR	undetectable redundant
AU ATPG untestable	AN	ATPG untestable-not detected
ND not detected	NC	not controlled
	NO	not observed

- Scan Area Overhead - It is the amount of extra logic used to convert all the memory elements present in the DUT to make it testable. The percentage of scan area overhead reduction is the ratio of the number of scan latches in the partial scan design to the number of scan latches used in the full scan design. It is given as a percentage in equation 5.2:

$$Scan\ Area\ Overhead\ Reduction\ Percentage = 1 - \frac{N_p}{N_f} \times 100 \qquad (5.2)$$

  where, N_p is the number of scan elements in the partial scan, and N_f is the number of scan elements in the full scan design.

- Number of Test Vectors - This is the number of stimulus and the corresponding responses of the DUT needed to test all the detectable faults accounting for the fault coverage of the DUT.

5.6 Results and Analysis

Two circuits, namely the majority gate based C-element and benchmark *half*, are discussed in detail. The C-element was used in Section 5.3 to demonstrate local loop detection and unrolling. Its fault coverage is evaluated in this section.

5.6.1 C-element

The majority gate implementation of C-element is shown in Figure 5.12.1 and the corresponding unrolled, acyclic circuit is shown in Figure 5.12.4. The C-element with its gate nodes labelled is shown in Figure 5.15. The Tetramax convention has been adopted here to ease fault analysis. The fault sites and the detection results for the original circuit and results of fault simulation of the acyclic circuit patterns over the original circuit obtained from Tetramax is given in Table 5.2.

The test vectors obtained for the acyclic circuit are: 111, 000, 100, 010, 111, 101, 011 for the pins a,b,c respectively, with c being added as an initialisation input. In order to apply the vectors to the original circuit, which only has 2 inputs (a, b), the last bit of each vector is removed. The list of faults in the C-element and the detection results are shown in Table 5.2. The third column gives the results for running the Tetramax on the original circuit. The fourth column gives the results for fault simulation of the acyclic circuit patterns over the original circuit. All faults are detected by the test pattern which gives a 100 % fault coverage.

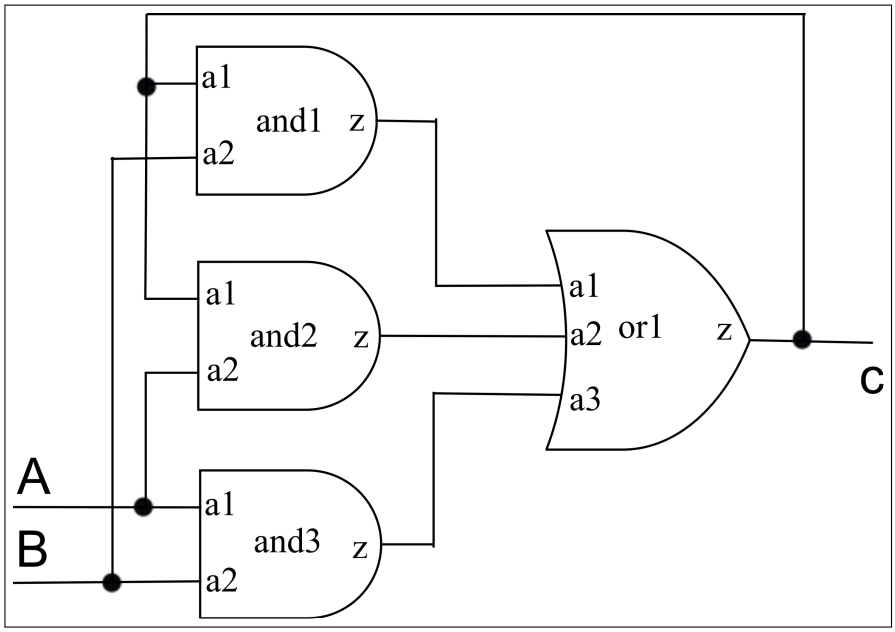


Figure 5.15: C-element -Majority Gate Implementation

Table 5.2: Fault Sites and Detection Results

Faults	Stuck-at type	Detection Result(orignal)	Detection Result(acyclic patterns)
A	1	NO	DS
B	1	NO	DS
And1A1	1	AN	NP
And1A2	1	AN	NP
And2A1	1	NO	DS
And2A2	1	AN	DS
And3A1	1	NO	DS
And3A2	1	AN	DS
Or1Z	1	DS	DS
C	1	DS	DS
A	0	NO	DS
B	0	NO	DS
And1/z	0	AN	DS
And2/z	0	AN	DS
And3/z	0	AN	DS
Or1/z	0	DS	DS
C	0	DS	DS

5.6.2 Benchmark "chu150"

Test generation for benchmark circuit chu150 is shown in Figure 5.16. The complex gate implementation of chu150 synthesized by petrify has 2 C-elements forming the two local feedback loops (Figure 5.16.1). The graph representation of the circuit forming the clouds and state holding elements is shown in Figure 5.16.2. Then the balanced graph equivalent with cyclic kernel formed with only one C-element is obtained (Figure 5.16.3). The removed element forms the scan latch based C-element. Now the balanced circuit with cyclic kernel is obtained (Figure 5.16.4). In order to generate the efficient test vectors, the cyclic kernel with one C-element should be converted in to an acyclic kernel; the cyclic-to-acyclic algorithm, which is applied to the cyclic kernel converts the local loop or C-element present in kernel in to an acyclic equivalent as shown in Figure 5.16.5. The circuit with acyclic kernel (Figure 5.16.5 and Figure 5.16.6) is now ready for test generation. The test pattern is obtained for this circuit using Tetramax. These test patterns are then fault simulated over the circuit with cyclic kernel

which is the original partial scan circuit as shown in Figure 5.16.7. Fault coverage obtained by the test vectors generated is 95.83%.

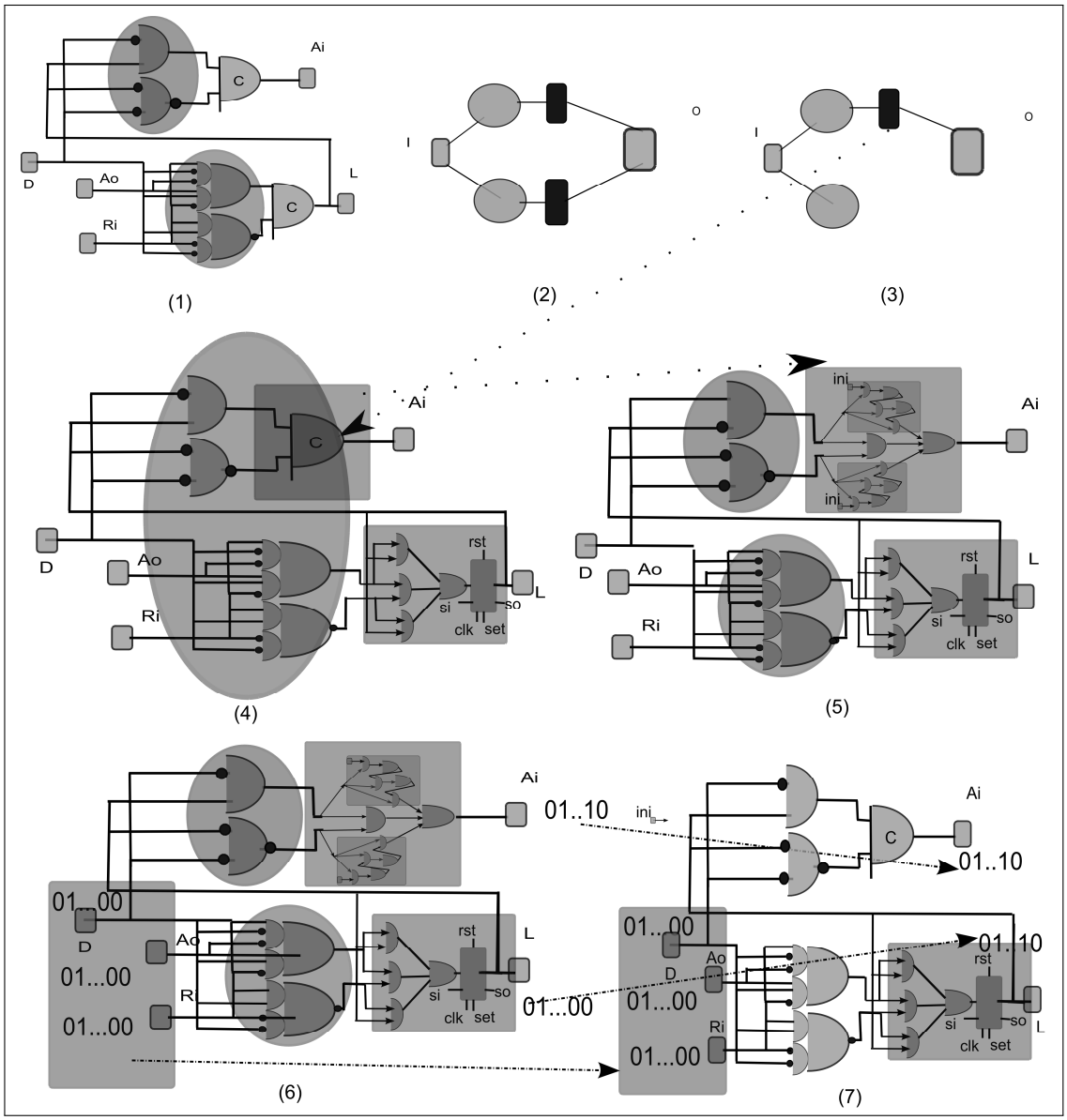


Figure 5.16: Test Generation for chu150

5.6.3 Results

Results of the proposed method are described in detail in this section. First, the fault coverage of the experimented benchmarks are shown and they analysed. Second, the scan area overhead is discussed. Finally the number patterns generated for attaining the reported fault coverages is discussed.

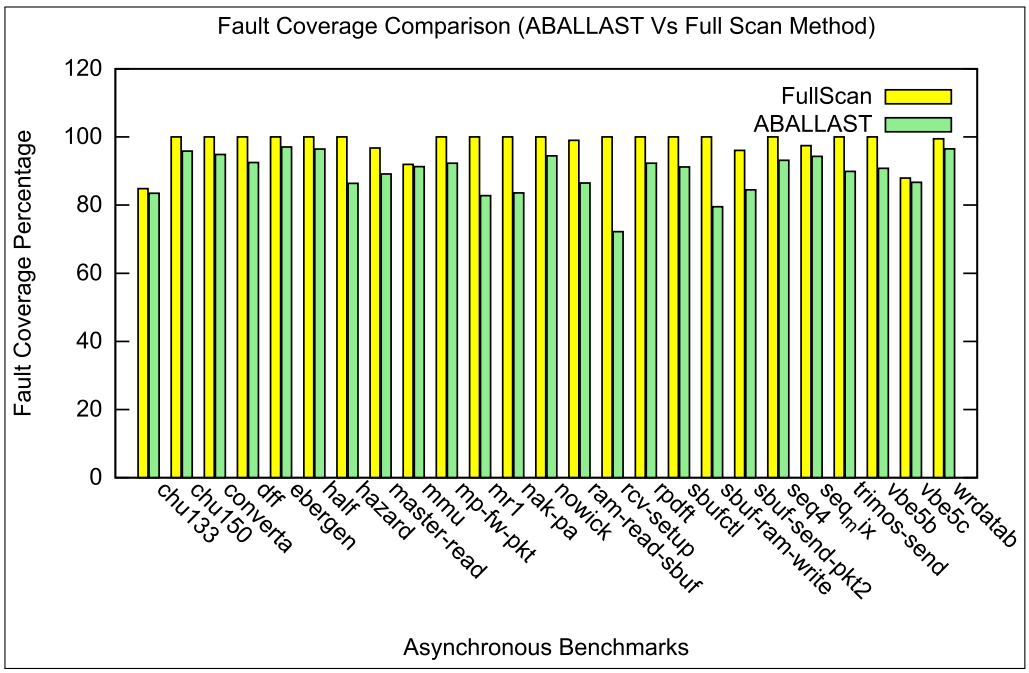


Figure 5.17: Fault Coverage comparison - ABALLAST vs Full scan

5.6.3.1 Fault Coverage

The fault coverage results and comparison with full scan method are shown in Table 5.3. The fault coverage for the benchmarks chu150, mp-fw-pkt and sbufctl were higher than all other benchmarks in the range closer to 90 percent. nakpa , rcv-setup and mr1 had lower fault coverage. Reduction in fault coverage of nak-pa and rcv-setup is due to the fact that the actual effect of partial scan design cannot be seen in these circuits due to presence of only one or two memory elements. Benchmarks such as, half, hazard, nak-pa, rcv-setup, rpdft , vbe5c and vbe5b also fall in to this category. They either have one or two C-elements or none at all. The main reason for reduction in the fault coverage is the presence of global loops. For the benchmarks: ebergen, half, chu150, sbuf-ram-write and wrdatab, ABALLAST achieved fault coverages of over 95 %. Significant improvement in fault coverage for the benchmark, "wrdatab", was achieved which has many C-elements and more global loops. As shown in Figure 5.17, yellow lines gives the fault coverage for full scan method and green lines give the fault coverage for ABALLAST. Clearly full scan has higher fault coverage than ABALLAST. It should be noted that for the benchmarks masterread, mmu and vbe5c the fault coverage is comparable to fault coverage obtained by full scan.

The comparison results of several benchmarks with other two methods mentioned in the evaluation methodology section is shown in Table 5.4. It shows the number of faults and the fault coverage for the circuits. They are obtained from [SM04b]. The fault coverage for the method in [Eic65] (shown in [SM04b]) was between 21.4% - 100%. For the method in [SM04b], the

fault coverage ranges between 85.7 - 100%. The benchmarks with columns 2,3 and 4 marked "-" are not reported in [SM04b]. From the comparison, ABALLAST achieves higher fault coverage for all the benchmarks with higher number of C-elements present in them. For example, the benchmarks chu150, converta, dff, ebergen, half , masterread, mr1,sbuf-send-ctl and seq4 had higher fault coverage than those in [Eic65]. ABALLAST also had higher fault coverage than the method in [SM04b] for the benchmarks converta, dff, and ebergen.

5.6.3.2 Scan Area Overhead

The comparison of scan area overhead is shown in the Table 5.5. As defined earlier, this metric gives the difference in the number of scanned memory elements chosen by the scan methods. The second and third column lists the number of C-elements scanned for full scan method and ABALLAST, respectively. The last column in the table gives the scan area over head percentage. Figure 5.18 depicts the difference in the number of scan C-elements chosen. The proposed method clearly chose fewer C-elements compared to the full scan method. Benchmarks that show better reduction in patterns also have shown better reduction in the number of C-elements scanned. For some benchmarks, namely nowick, rcv-setup, rpdft, vbe5b and vbe5c, only red line is shown in the graph. This means that no C-element was scanned by the ABALLAST. The fault coverage gets reduced by 10 % for this reason. Since these are smaller benchmarks, they can be full scanned to get the maximum coverage.

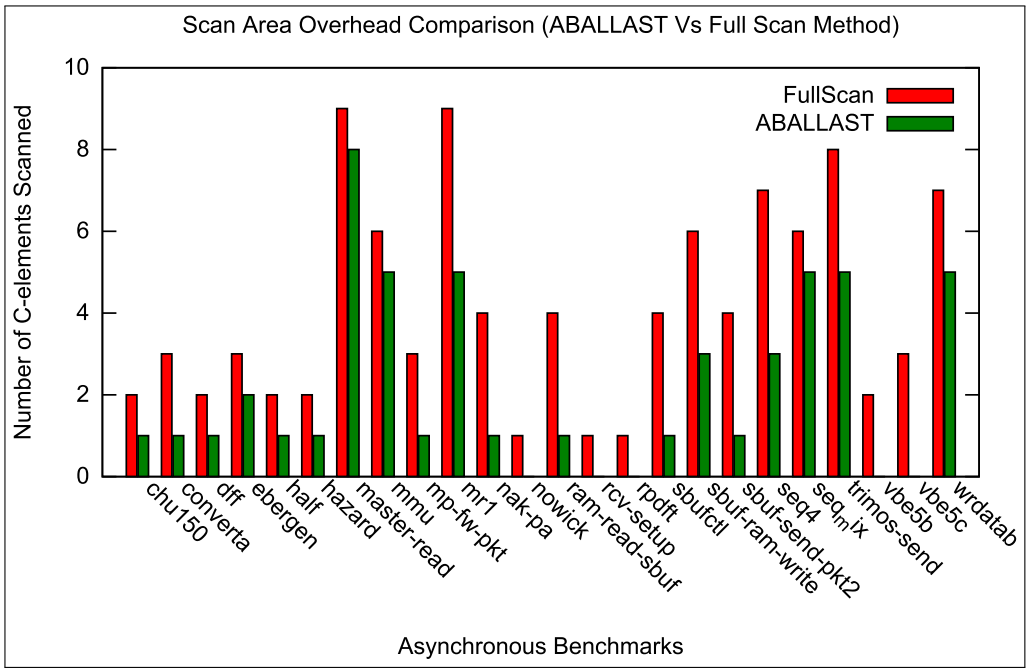


Figure 5.18: Scan area overhead comparison

5.6.3.3 Number of patterns

The number of test patterns generated by the test pattern generator is shown in the Table 5.6. Most of the benchmarks had lower number of patterns for the proposed method compared to the full scan method due to the reduction in the number of scan flipflops. As shown in Figure 5.19, the taller green lines for the benchmarks mr1, mmu, trimos-send and wrdatab indicates the full scan method requiring higher number of patterns to achieve the fault coverage. It should be noted that ABALLAST required 20 - 40 % lower number of patterns. For example, to test the benchmark mr1 ABALLAST needed only 29 test patterns, whereas full scan required 48 patterns. But, in the case of benchmarks chu150,converta, half and seq_mix the number of test patterns increased for the proposed method.

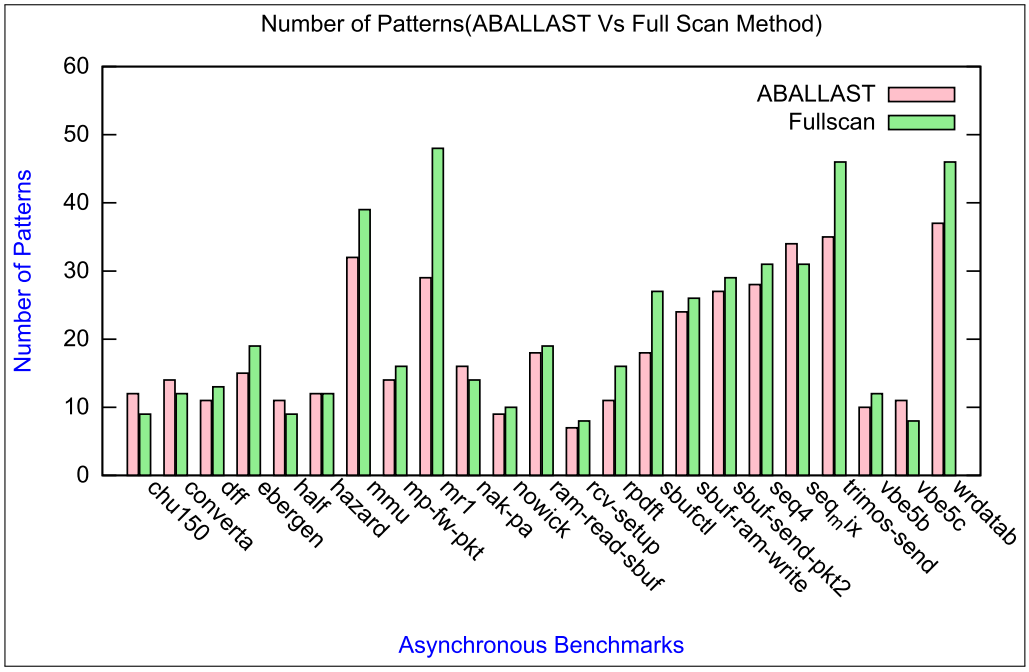


Figure 5.19: Comparison of Number of Patterns Generated -ABALLAST Vs Full Scan

5.6.3.4 Analysis of Undetectable Faults

Table 5.7 gives the complete distribution of fault classes for all the benchmarks. The last four columns gives the four different fault types, namely, Not Observable (NO), Not Controllable (NC), Detected(DT) and Possibly Detected (PT), respectively. Fault classes of interest are the NO and NC faults. These are the main faults causing the fault coverage of the asynchronous circuits to be lower. Figure 5.20 shows the distribution of these faults over all the benchmarks. The blue stack is the number of faults detected and striped blue stacks (found at the tip of each line) are the number of possibly detected faults. Given that ABALLAST having 80 -100 %

fault coverage, the blue stacks dominate the graph. But, the focus of interest in this graph is the two stacks: NO (red mesh pattern) and NC (checked green pattern). From the zoomed area in the graph, NO faults dominate all the benchmarks. For example sbuframwrite had 11 NO faults and trimos-send had 13 NO faults. The main reason underlying the difficulty in observability is the local loops in the unscanned C-elements. To probe this further, some of the benchmarks were selected and analyzed in the next subsection.

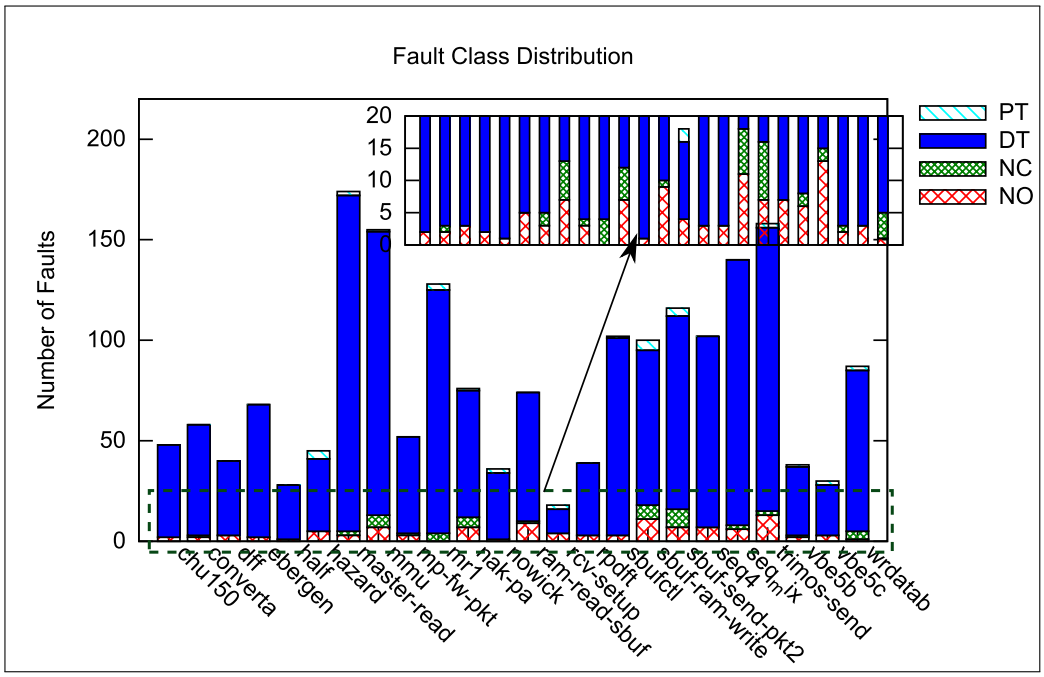


Figure 5.20: Distribution of different fault classes, PT- Possible Detected, DT- Detected, NO - Not observable, NC - Not Controllable

The list of undetectable faults and their classification based on their controllability and observability and structural location is given in the Table 5.8.

Un-Controllable(NO) and Un-observable(NO) Faults

Most of the faults which occured were unobservable as opposed to being uncontrollable. For example, in the benchmarks ram-rd-sbuf, out of 10 undetectable, 9 of them were unobservable. The reason for this condition is the location of the fault sites: most of the faults are in the nodes that are either feeding or fed by the C-element.

Faults sites before or after the C-element

Out of 10 faults in the same benchmark, 7 of them are either fed by or fed to the C-elements (in this case 2 of the 7 nodes are feeding C-element, and the other 5 are fed by C-element).

The trend of undetectable fault sites in the global loop path is very low, and is seen only in the benchmarks, sbufsndctl and ram-rd-sbuf.

Faults inside unscanned C-elements

For some of the benchmarks, most of the faults occured in the unscanned C-elements. For example, in the benchmark sbuf-ram-write, 9 out of 17 faults occurred in the nodes inside the C-elements.

5.6.3.5 Number of Copies

It will be interesting to see how well the fault coverage improves when we increase the number of copies of the forward path added to the acyclic circuit conversion. To explore this, the number of copies of the forward path of the cycles is increased in the cyclic to acyclic conversion process. That is, only one copy of the forward path was considered when cutting the loops and now it will be more than one copy cascaded with one other. In total, 8 different experiments for all the benchmarks were carried out, with each one having 1 to 8 copies of the forward path for the cyclic-to-acyclic operations, respectively. The trend of fault coverage and the number of patterns generation for the test are next analyzed.

Fault Coverage

Table. 5.9 shows the fault coverage impact over the number of copies made. In Figure 5.21 shows the fault coverage comparison of the proposed method with 1 to 8 copies of forward path. Figure 5.21. (a -c) shows the trend for benchmarks with lower number of C-elements. Figure 5.21.d has the benchmarks with higher number of C-elements. From figures Figure 5.21.(d) it is evident that the fault coverage is highly impacted by the increase in number of copies as the benchmarks have higher number of C-elements and global loops.

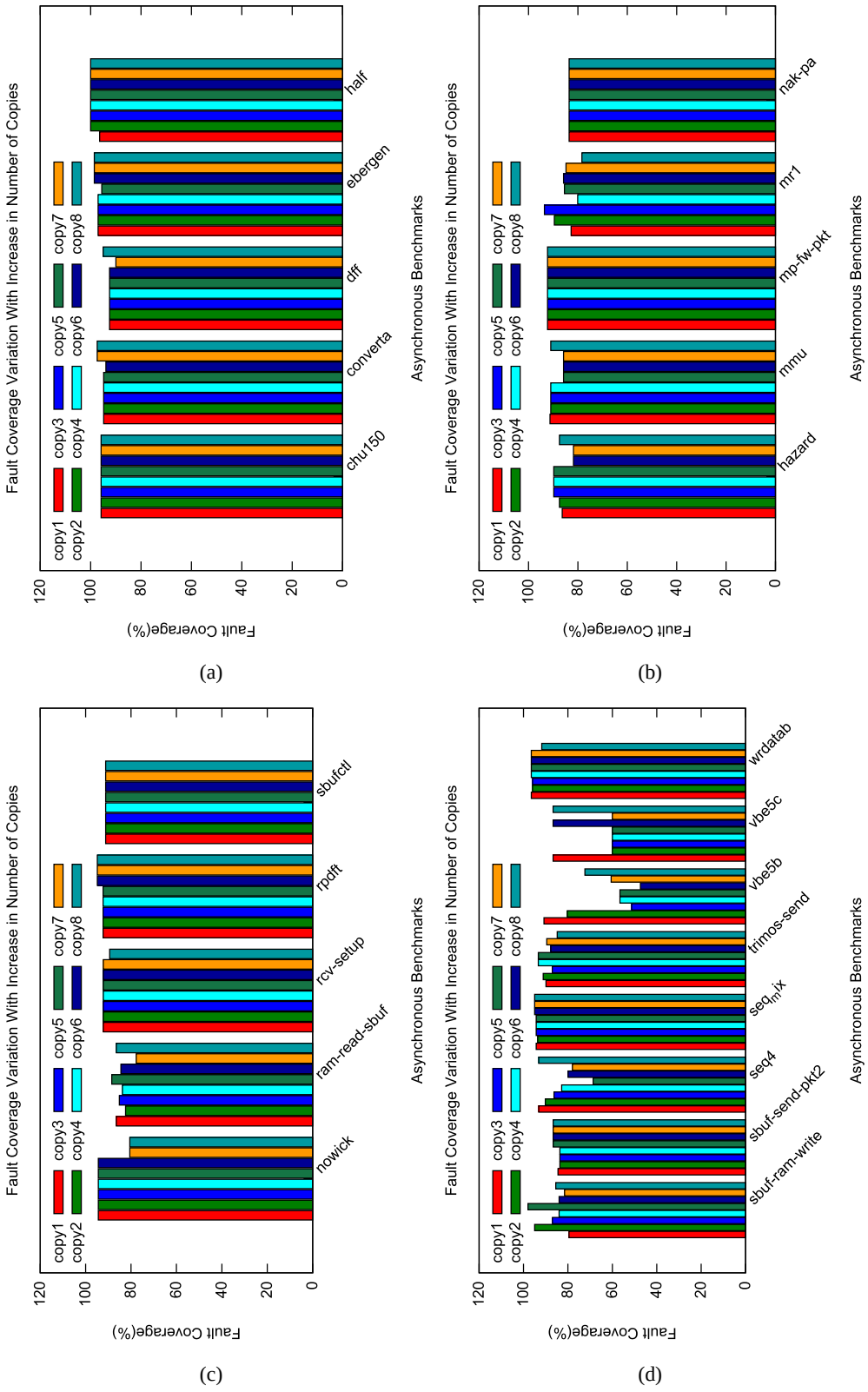


Figure 5.21: Fault coverage of Benchmarks with copies 1 to 8 shown in (a), (b) ,(c), and (d)

For example, the fault coverage of the seq4 and trimos-send had a high variation in fault cov-

erage. For seq4 benchmark the fault coverage varied from 68.63 - 93.14%. For the benchmark trimos-send, the variation was from 84.85 - 93.35%. Benchmark vbe5b exhibited the highest difference in faultcoverage. The single copy version of the benchmark achieved 90.79% fault coverage while the siz xopy version achieved only 47.37%. For some benchmarks namely, chu150, nowick, nak-pa and mp-forward-pkt had no impact due to the increase in copies. Most of these had either one or two C-elements or had no global loops.

Number of Patterns

Table 5.10 lists the number of patterns generated for 1 to 8 copies circuits by ABALLAST. Figure 5.22 shows the impact on number of patterns generated with 1 to 8 copies of forward path. The benchmarks that had no impact on the fault coverage was due to the fact that the increase in number of copies did not find more patterns for those benchmarks. As shown in Figure 5.22.(a-d), the benchmarks chu150, nak-pa, and mp-fwd-pkt had same number of patterns generated for all the 8 versions. But the benchmark nowick had variation in the number of test patterns. For example, the 1-6 copies of circuit had generated same number of patterns (9 patterns). But, the circuits with 7 and 8 copies generated lower number of patterns. But, it should be noted that with lower number of patterns these two circuits provided the same fault coverage as previous 5 versions. Hence increasing the number of copies reduced the number of patterns needed to test the same circuit in this benchmark. For benchmarks mr1, ram-read-sbuf, wrdatab increasing the number of copies increased the number of patterns. For other benchmarks the increase was not monotonic with increase in number of copies. The impact of making several copies of the circuit for test generation over the fault coverage is shown as a 3D plot in Figure 5.23. The level in the middle of the plot shows the lower impact on fault coverage over the benchmarks with fewer C-elements. The impact on the number of patterns is shown in the Figure 5.24. It should be noted that the benchmark, wrdatab, shows a steep rise in the number of patterns for the "8" copy circuit. Increasing the fault coverage of the acyclic equivalents of the circuit will increase the fault coverage of the DUT. So some advanced method has to be applied to find the test for the redundant faults present in the acyclic equivalent circuit.

Complexity The complexity of the ABALLAST method is the summation of the complexity of the three steps namely, BALANCE, the Cyclic-to-Acyclic Conversion. The complexity of the Balance procedure is O(nm^3). This is derived from the fact that the balance procedure computes the minimum cutest for O(m) times and the size of each cutest is bounded by m (at worst case) and the procedure check (which has the bound of O(mn) is called over each of these cute. For the cyclic2acyclic conversion, the performance is dominated by the cycle detection process with the upper bound of O($m/2$).

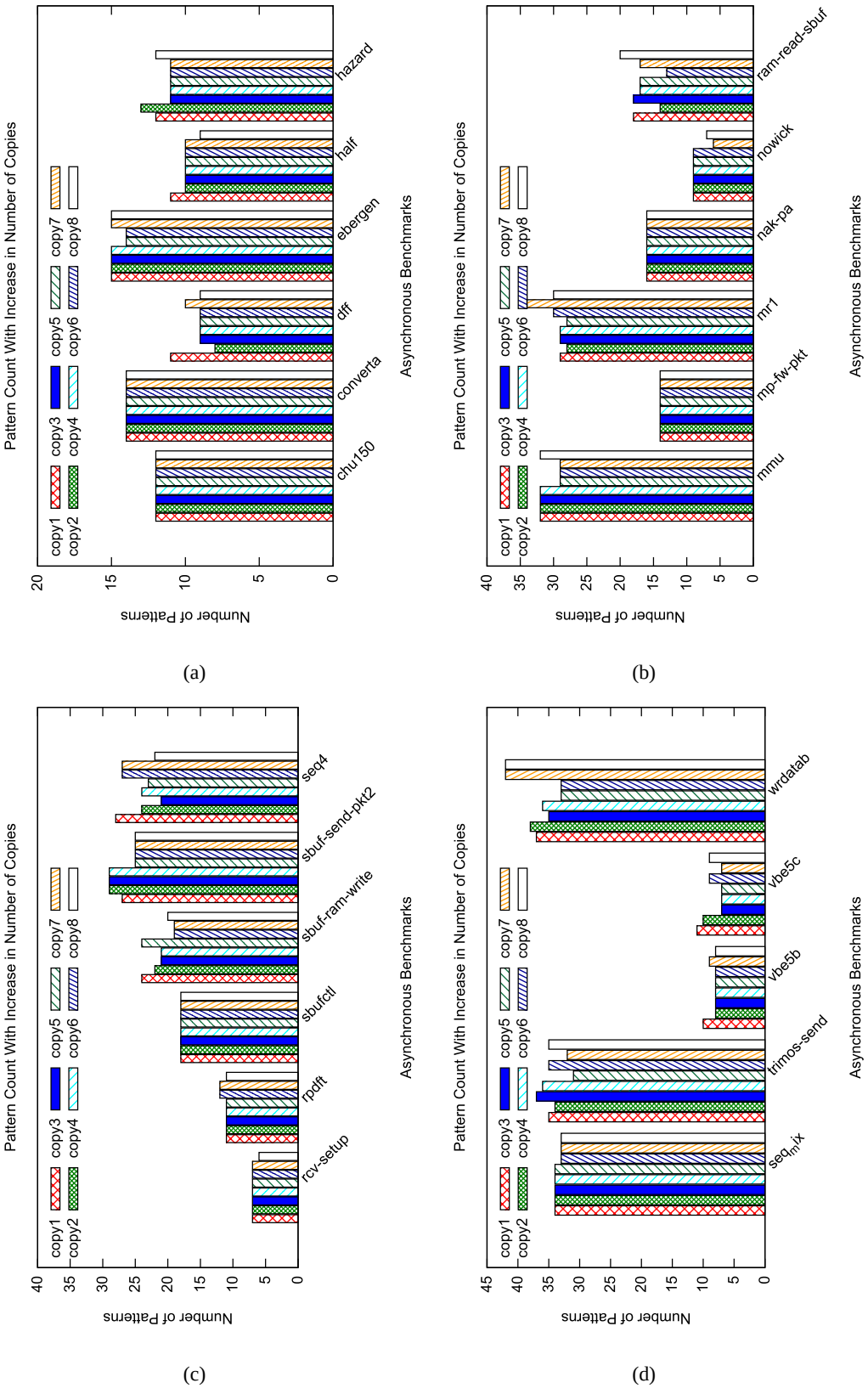


Figure 5.22: Number of Test Patterns generated for Benchmarks with copies 1 to 8 shown in (a), (b) ,(c), and (d)

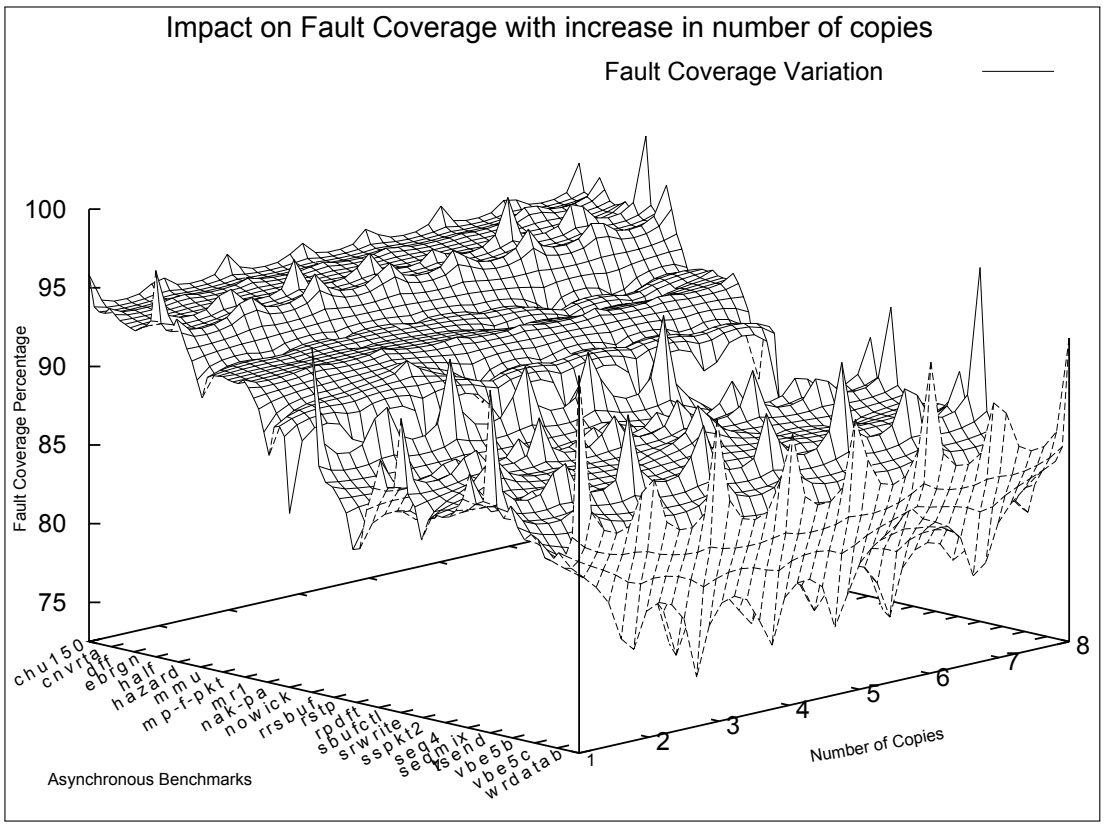


Figure 5.23: 3D Plot depicting the impact on fault coverage

5.6.3.6 Summary

The prime factors affecting the testability of the asynchronous circuits are summarised.

Factors affecting the fault coverage are:

- Depth of the node in the circuit
- Memory elements present in the circuit
- Feedback paths present in the circuit
- Observability of the nodes
- Controllability of the nodes
- Type of logic used for simulation
- location of the nodes in front of or after the C-elements

Conventional circuit structures affecting the test quality are:

- Reconvergent fanout
- Feedback paths

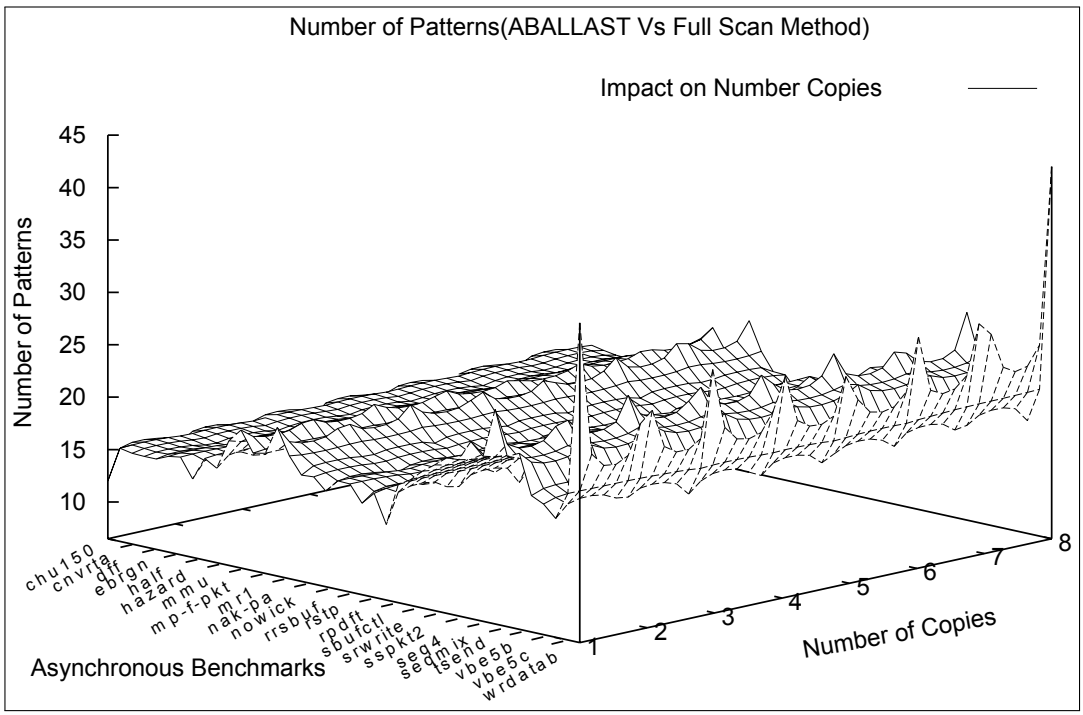


Figure 5.24: 3D Plot depicting the impact on number of patterns

- Blocking Scan paths

Hence, the following changes are required for improving the testability of the asynchronous design:

- Logic level of the fault simulation should be changed.
- New method for realizing and simulating the feedback cycles seperately should be devised.
- DFT for feedback paths should be designed

5.7 Conclusion

A test pattern generation method for asynchronous circuits was presented.This test method provided the following:

- An effective way of handling the cyclic asynchronous circuits such that they can be used with the Tetramax test generator
- Partial scan element selection based on balanced sequential structures
- Automatic Test pattern generation for the partial scan design generated

This method gives a better fault coverage compared to [Eic65]. But it does not out-perform [SM04a]. In [SM04a], the test generation was based on random test patterns and custom fault simulation, which constributes to the higher number of test vectors and detectable faultlist. The area overhead is effectively reduced compared to the full scan based design.

Table 5.3: Fault Coverage

Benchmarks	Full Scan	ABALLAST
chu150	100	95.83
converta	100	94.83
dff	100	92.50
ebergen	100	97.06
half	100	96.43
hazard	100	86.36
master-read	96.76	96.55
mmu	91.95	91.29
mp-fw-pkt	100	92.31
mr1	100	82.77
nak-pa	100	83.55
nowick	100	94.44
ram-read-sbuf	99.02	86.49
rcv-setup	100	72.22
rpdft	100	92.31
sbufctl	100	91.20
sbuf-ram-write	100	96.43
sbuf-send-pkt2	96.03	84.48
seq4	100	93.14
seq_mix	97.44	94.29
trimos-send	100	89.87
vbe5b	100	90.79
vbe5c	87.93	86.67
wrdatab	99.46	96.49

Table 5.4: Result – Fault Coverage Comparison

Benchmark	No of faults	[Eic65]	[SM04a]	Proposed
chu150	56	97.1	97.1	95.83
converta	54	56.8	91.9	94.83
dff	44	21.4	85.7	92.5
ebergen	74	47.8	95.7	97.06
half	22	40	100	96.43
hazard	48	87.9	97	90.91
masterread	144	65.1	97.7	97.13
mmu	-	-	-	91.61
mp-forward-pkt	60	100	100	92.31
mr1	152	10.8	93.5	83.78
nak-pa	82	100	100	84.21
nowick	56	100	100	97.22
ram-read-sbuf	90	100	100	86.49
rcv-setup	40	100	100	77.78
rpdft	62	100	100	92.31
sbuf-send-ctl	94	59.3	94.9	91.67
sbuf-ram-write	110	100	100	82
sbufsend-pkt2	-	-	-	86.21
seq4	96	54	95.2	93.14
seq_mix	-	-	-	94.29
trimos-send	-	-	-	90.51
vbe5b	-	-	-	92.11
vbe5c	-	-	-	90
wrdatab	-	-	-	97.08

Table 5.5: Scan Area Overhead

Benchmarks	Full Scan	ABALLAST	Area Overhead Reduction (%)
chu150	2	1	50.00
converta	3	1	66.67
dff	2	1	50.00
ebergen	3	2	33.33
half	2	1	50.00
hazard	2	1	50.00
master-read	9	8	11.11
mmu	6	5	16.67
mp-fw-pkt	3	1	66.67
mr1	9	5	44.44
nak-pa	4	1	75.00
nowick	1	0	100.00
ram-read-sbuf	4	1	75.00
rcv-setup	1	0	100.00
rpdft	1	0	100.00
sbufctl	4	1	75.00
sbuf-ram-write	6	3	50.00
sbuf-send-pkt2	4	1	75.00
seq4	7	3	57.14
seq_mix	6	5	16.67
trimos-send	8	5	37.50
vbe5b	2	0	100.00
vbe5c	3	0	100.00
wrdatab	7	5	28.57

Table 5.6: Number of Patterns

Benchmarks	Full Scan	Proposed
chu150	12	9
converta	14	12
dff	11	13
ebergen	15	19
half	11	9
hazard	12	12
mmu	32	39
mp-fw-pkt	14	16
mr1	29	48
nak-pa	16	14
nowick	9	10
ram-read-sbuf	18	19
rcv-setup	7	8
rpdft	11	16
sbufctl	18	27
sbuf-ram-write	24	26
sbuf-send-pkt2	27	29
seq4	28	31
seq_mix	34	31
trimos-send	35	46
vbe5b	10	12
vbe5c	11	8
wrdatab	37	46

Table 5.7: Fault Class Distribution

Benchmarks	NO	NC	DT	PT
chu150	2	0	46	0
converta	2	1	55	0
dff	3	0	37	0
ebergen	2	0	66	0
half	1	0	27	0
hazard	5	0	36	4
master-read	3	2	167	2
mmu	7	6	141	1
mp-fw-pkt	3	1	48	0
mr1	20	4	121	3
nak-pa	7	5	63	1
nowick	1	0	33	2
ram-read-sbuf	9	1	64	0
rcv-setup	4	0	12	2
rpdft	3	0	36	0
sbufctl	3	0	98	1
sbuf-ram-write	11	7	77	5
sbuf-send-pkt2	7	9	96	4
seq4	7	0	95	0
seq_mix	6	2	132	0
trimos-send	13	2	141	2
vbe5b	2	1	34	1
vbe5c	3	0	25	2
wrdatab	1	4	80	2

Table 5.8: Undetectable Fault Locations

Bench	Total Un-detectable	NO faults	NC Faults	feeding cele	fed by cele	inside cele	faults in Global loop
chu150	2	2	0	2	-	-	-
nakpa	13	8	5	6	-	4	-
mp-fw-pkt	4	3	1	3	-	-	-
ram-rd-sbuf	10	9	1	2	5	3	3
sbuf-ram-write	17	10	7	3	1	9	-
sbufsndctl	9	9	0	-	-	6	3

Table 5.9: Fault Coverage Comparison of proposed method using 1 to 8 copies of forward path during acyclic conversion

Benchmarks	copy1	copy2	copy3	copy4	copy5	copy6	copy7	copy8
chu150	95.83	95.83	95.83	95.83	95.83	95.83	95.83	95.83
converta	94.83	94.83	94.83	94.83	94.83	93.97	97.41	97.41
dff	92.5	92.5	92.5	92.5	92.5	92.5	90	95
ebergen	97.06	97.06	97.06	97.06	95.59	98.53	98.53	98.53
half	96.43	100	100	100	100	100	100	100
hazard	86.36	87.5	89.77	89.77	89.77	81.82	81.82	87.5
mmu	91.29	90.97	90.97	90.97	85.81	85.81	85.81	90.97
mp-fw-pkt	92.31	92.31	92.31	92.31	92.31	92.31	92.31	92.31
mr1	82.77	89.53	93.58	80.07	85.47	85.81	84.8	78.38
nak-pa	83.55	83.55	83.55	83.55	83.55	83.55	83.55	83.55
nowick	94.44	94.44	94.44	94.44	94.44	94.44	80.56	80.56
ram-read-sbuf	86.49	82.43	85.14	83.78	88.51	84.46	77.7	86.49
rcv-setup	72.22	72.22	72.22	72.22	72.22	72.22	72.22	69.44
rpdft	92.31	92.31	92.31	92.31	92.31	94.87	94.87	94.87
sbufctl	91.2	91.2	91.2	91.2	91.2	91.2	91.2	91.2
sbuf-ram-write	79.5	95	87	84	98	84	81.5	85.5
sbuf-send-pkt2	84.48	83.62	83.62	83.62	86.64	86.64	86.64	86.64
seq4	93.14	90.2	86.27	82.84	68.63	79.9	77.94	93.14
seq_mix	94.29	93.57	94.29	94.29	94.29	95	95	95
trimos-send	89.87	91.14	87.03	93.35	93.35	87.66	89.56	84.81
vbe5b	90.79	80.26	51.32	56.58	56.58	47.37	60.53	72.37
vbe5c	86.67	60	60	60	60	86.67	60	86.67
wrdatab	96.49	95.91	95.91	96.49	96.49	96.49	96.49	91.81

Table 5.10: Comparision of Number of patterns generated for the circuits with 1 to 8 copies of forward path during acyclic conversion

Benchmarks	copy1	copy2	copy3	copy4	copy5	copy6	copy7	copy8
chu150	12	12	12	12	12	12	12	12
converta	14	14	14	14	14	14	14	14
dff	11	8	9	9	9	9	10	9
ebergen	15	15	15	15	14	14	15	15
half	11	10	10	10	10	10	10	9
hazard	12	13	11	11	11	11	11	12
mmu	32	32	32	32	29	29	29	32
mp-fw-pkt	14	14	14	14	14	14	14	14
mr1	29	28	29	29	28	30	34	30
nak-pa	16	16	16	16	16	16	16	16
nowick	9	9	9	9	9	9	6	7
ram-read-sbuf	18	14	18	17	17	13	17	20
rcv-setup	7	7	7	7	7	7	7	6
rpdft	11	11	11	11	11	12	12	11
sbufctl	18	18	18	18	18	18	18	18
sbuf-ram-write	24	22	21	21	24	19	19	20
sbuf-send-pkt2	27	29	29	29	25	25	25	25
seq4	28	24	21	24	23	27	27	22
seq_mix	34	34	34	34	34	33	33	33
trimos-send	35	34	37	36	31	35	32	35
vbe5b	10	8	8	8	8	8	9	8
vbe5c	11	10	7	7	7	9	7	9
wrdatab	37	38	35	36	33	33	42	42

Chapter 6

AGLOB - Asynchronous Circuit Test Generation Based on Breaking Global Loops

6.1 Introduction

This chapter proposes a novel partial scan design methodology and a technique for generating test patterns for asynchronous circuits. Generating test patterns with high stuck-at fault coverage and achieving a lower area overhead compared to existing full scan methods forms the motivation for this work. Some work related to this chapter is detailed below.

Partial scan and full scan test methods for asynchronous circuits developed so far are for specific asynchronous design styles and methods. The roadblock for testing in all these design methods seems to be the cyclic circuits present in them. Also converging the methods to an industrial test generation tools poses another problem. This chapter is motivated towards developing a partial scan based ATPG method aiding the synchronous CAD tools to generate tests for asynchronous cyclic sequential circuits.

Two main contributions made in this work are: 1) extending the synchronous partial scan method to be used for test generation of cyclic asynchronous circuits, and 2) cyclic-to-acyclic circuit conversion method to prepare the circuit for test pattern generation. Fault coverage of 76-96% was obtained using this method. The organization of the chapter is as follows: Section 2 gives the background; Section 3 describes the proposed algorithms for the test method; Section 4 describes the test methodology; the results are analyzed in Section 5 with two working examples, with conclusions in section 6.

6.2 Background

Asynchronous circuits use combinational loops to store state. There are two types of loops, namely global and local loops. Local loops are the combinational loops present in the state-holding gates such as C-elements or set-reset latches. The familiar flip-flop also contains a local loop, but it is hidden from test tools since a flip-flop is a cell on its own in standard cell libraries and does not pose any problems in testing. Global loops are longer loops formed outside these gates and are used for creating asynchronous state machines. Asynchronous full-scan methods [BPvBK03] break all these loops in test mode using LSSD-type scan latches. This simplifies testing as the circuit is transformed in to a purely combinational one in test mode. However, the area overhead is enormous, hence motivating our work on partial-scan methods.

6.3 Test Methodology

Several steps involved in this test methodology are discussed in this section. Figure 6.1 shows the components involved in test generation. As the circuits dealt in this method are asynchronous circuits, the state graph level description of the circuits is synthesized using Petrify[CKK$^+$97] . The synthesized circuits are converted to graph level representation. BLIF2Graph generator converts the circuit representation to graph in which nodes represent the gates and edges represent the connection between the gates. In order to apply conventional scan selection method[CA90] , the abstract representation of the graph called s-graph [CA90], with only memory elements are needed. The abstract level of graph with memory elements as nodes and paths between the elements as edges is created. In the next step, the strongly connected components are identified which aids the scan selection algorithm. A graph represented by G (V,E) ,where V forms the set of vertices and E forms the set of edges is said to be a strongly connected graph if there exists a path from each vertex of the graph to every other vertex. Strongly connected components of the graph are its maximal strongly connected subgraphs. The algorithm for finding the strongly connected components is a linear time O (V+E) for the graph represented as an adjacency list [THCR01]. It uses depth-first search to find the components of the graph. By applying the scan selection method(AGLOB1,AGLOB2) the memory elements to be scanned are selected. With synchronous designs, the circuit is ready for scan test generation as the global loops are broken, but for asynchronous design, the circuits still contain the local loops. The C-elements that have not been selected to be scanned constitute these loops. Therefore the circuit has to be passed to the cyclic-to-acyclic converter.

6.3.1 Cyclic-to-Acyclic Conversion

Cyclic-to-acyclic conversion of the circuit should be performed for the effective test generation of asynchronous circuits using a synchronous TPG tool. The conversion removes all the feedback loops formed in the cyclic circuit. As a result the tool's visibility of the fault sites will increase so that it will be able to generate test patterns of high fault coverage. The produced patterns will then be applied to the acyclic (partial-scanned) circuit. Several methods for generating an acyclic circuit from cyclic circuits have been introduced [Edw03],[Mal93],[Wei72],[Niv04]. Unfortunately, these methods are restricted for cyclic circuits without state holding elements and which do not oscillate. Oscillations are predominant in asynchronous cyclic circuits and also state holding elements like C-elements are commonly found in them.

Thus, the acyclic partially scannable equivalent of the cyclic partially scannable circuit is obtained. Now the design is passed through a conventional test pattern generator. Synopsys's Tetramax was used for test generation and fault simulation.

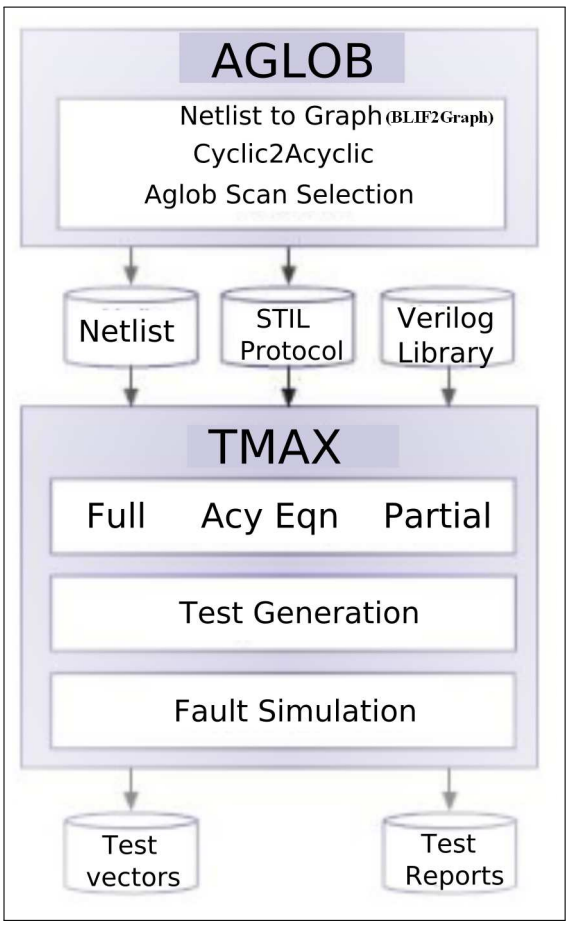


Figure 6.1: Test Methodology

6.4 Algorithms

The circuit model and the algorithms involved in Global loop breaking, scan selection, cyclic to acyclic conversion are discussed further in detail.

6.4.1 Global loop breaking

In [CA90], the method of global loop breaking involves representing the design in an abstract circuit topology graph. All the vertices in the graph represents the flip-flops in the design and the edges forms the path between the flops comprising of combinational gates and wires. Then the graph is processed to find the strongly connected components [HS89] present in it, which constitutes to the global loops in the circuit. All the cycles or loops are stored as a list to be processed by the flip-flop selection algorithm. The Breakloop algorithm, outlined below, selects the minimum number of flipflops in the design. Scanning the selected flipflops will cut all the global loops in the design.

The flipflops selected will form the scan elements in case of synchronous design. In the case of an asynchronous circuit, the C-elements present in the circuit are also considered as local loop or memory elements. Thus C-elements are added as vertices during the graph representation, before applying the scan selection algorithm. After applying the scan selection algorithm, the selected C-elements and latches will form the scan elements for the design. As an example, Figure 6.2.a shows a benchmark seq4 with 7 memory elements with the combinational gates and I/O pins shown as circles. The graph representation of the circuit is shown in Figure 6.2.b. As explained earlier, the vertices represent all the memory elements present in the circuit. Four strongly-connected components can be identified from the graph and the vertices list forming each component is shown in Fig 6.2.c. These components form the cycles present in the circuit. Note that, the vertex Ce3 appears in the second, third and fourth cycle. Hence when the scan selection algorithm is applied vertex Ce3 will be selected in the first pass. Selection of this vertex will remove cycle 2,3 and 4 from the cycle list. Vertex Ce1 will be selected in the second pass, which is present in the cycle 1. Thus the cycle list is emptied after the selection of vertex Ce1 and the algorithm is halted. Thus the resulting partial scan circuit with Ce1 and Ce3 forming the scan elements is shown in Fig 6.2.d. Though the scan elements are selected, the resulting partially scanned circuit may contain C-elements that are not scanned. These C-elements constitute the local loops of the circuit. Hence another step is needed to handle these local loops and create the acyclic equivalent of the design.

Two algorithms are proposed for selecting the scan chains, namely AGLOB1 and AGLOB2 and one algorithm for converting the cyclic circuits to acyclic ones. The conventional scan

Algorithm 7 Conventional Scan Selection algorithm

Conventional_SCC

For a s-graph G(V,E) {

```
  If the graph has cycle {
   Find all the Cycles of the graph (heuristically)
   Generate a list of cycles
   Find the frequency of occurance of each vertex in all SCC
   Choose the vertex/c-element with higher frequency
   add to scan elements set
   Remove the SCC's containing the vertex }
   }
```

selection algorithm is shown in Algorithm 7.

6.4.1.1 Algorithm 1- AGLOB1

In the first algorithm AGLOB 1 (shown in Algorithm 8)the c-elements and flipflops are selected based on maximum occurrence of them in all the cycles. This is similar to the conventional scan selection algorithm in which all the flipflops are selected based on their occurrence. We have extended this algorithm to be used in selection of C-elements in the asynchronous circuit and adding cyclic-to-acyclic conversion to the resulting circuit. Thus, in AGLOB 1 finding set of the memory elements is followed by converting the resulting partial scan circuit to acyclic circuit.

6.4.1.2 Algorithm 2-AGLOB2

The second algorithm AGLOB 2 (shown in Algorithm 9) deals with selecting the C-elements based on maximum degree of the vertices/C-elements present in the circuits. The degree of a vertex is the sum of incoming arcs and outgoing arcs. Once the scan elements are selected, the partial scan circuit is converted to its acyclic equivalent by applying the Cyc2Acyc algorithm.

6.4.2 Cyclic-to-Acyclic Conversion

Once the scan elements are identified, for the purpose of test pattern generation, the resulting circuit must be converted into an acyclic one by replicating the appropriate parts of the circuit. This is similar to the time frame unrolling method, used in sequential pattern generation [[MAF90]]. The conversion method, (Algorithm 10), requires a user specified number of cycle

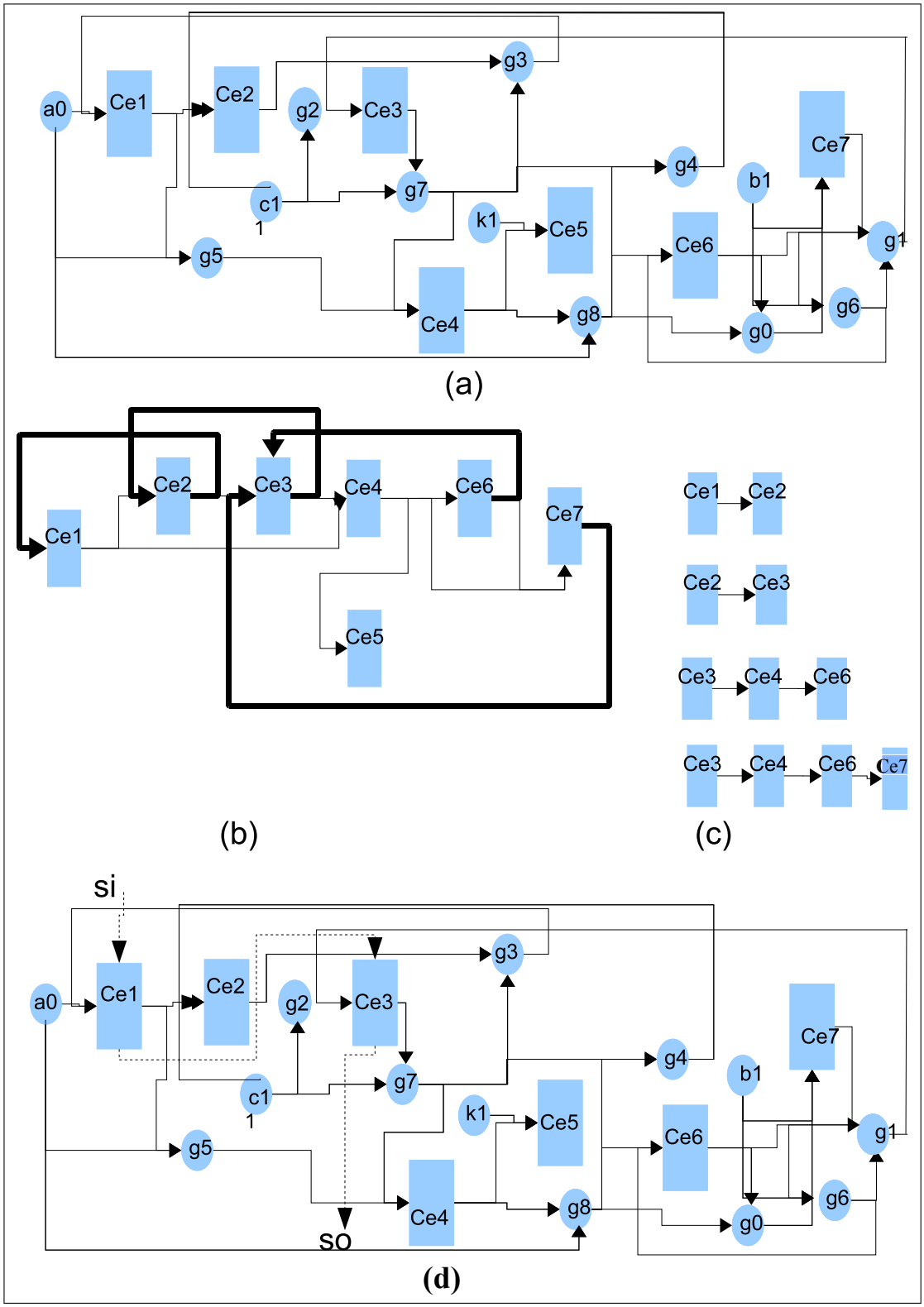


Figure 6.2: Scan Selection

Algorithm 8 AGLOB 1 -Asynchronous SCC based Scalgorithm

For a s-graph G(V,E) {

```
If the graph has cycle {
  Find all the SCC of the graph
  Generate a list of SCC's
  Find the frequency of occurance of each vertex in all SCC
  Choose the vertex/c-element with higher frequency
  add to scan elements set
  Remove the SCC's containing the vertex
  }
Create G(V-S,E),Where S is scan element set,
     and Selected Scan elementsS(n)
Check G(V-S,E) for c-elements.
If present{
   Acyclic Graph Ga(V-S,E) = Cyc2Acyc(G(V-S),E);
 }
Output G(V-S),E), Ga(V-S),E), S
}
```

Algorithm 9 AGLOB 2

For a s-graph G(V,E) {

```
if the graph has cycle {
 find the degree of each vertex /c-element
 choose the vertex with high degree
 remove the vertex from the graph
 }
Create G(V-S,E),Where S is scan element set,
 and Selected Scan element Set S
Check G(V-S,E) for c-elements.
If present{
  Acyclic Graph Ga(V\_S,E) = Cyc2Acyc(G(V-S,E));
  }
Output G(V-S),E), Ga(V-S),E), S
}
```

copies. The number of cycle copies is equal to the number of time frames it takes for the circuit to stabilize. For example, if the circuit is assumed that it will stabilize in three time frames, the resulting acyclic circuit will have three forward path copies of the path in the corresponding

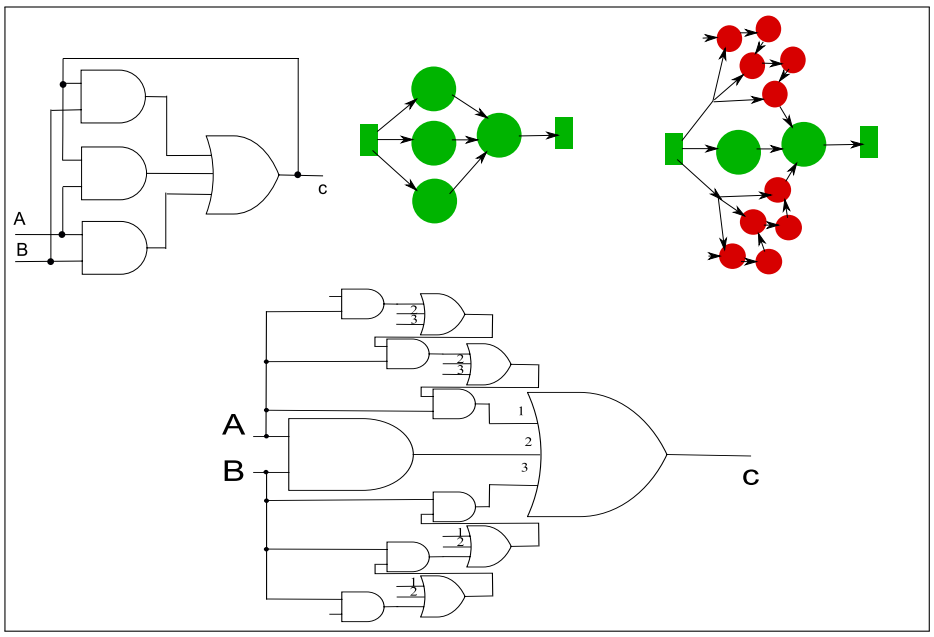


Figure 6.3: C-element Cyclic to Acyclic Conversion

cycle in the cyclic circuit. Since the feedback path is broken, the node where the feedback is broken is initialized with an input pin. The number of copies (ncopy in procedures of Algorithm 10) will also depend upon the number of cycles present in the original cyclic circuit and whether they are nested or intersected. It will be provided by the user based on the design library used. The typical example of converting the C-element from its cyclic to acyclic equivalent is shown in Figure 6.3

Algorithm 10 Cyclic-to-Acyclic Conversion

Algorithm: CycToAcyc

Inputs: Cyclic Graph G (V, E), Cycles graph Gc (V,E), ncopy

Output: Acyclic Graph Ga (V,E)

```
Cyc_To_Acy (G(V,E),Gc(V,E)) {
 Ga(V) = G(V));
 Ga(E) = Create_Edges(G(E))
 Gc(V,E) = cycle_path_duplication(Gc(V,E),ncopy) ;
 Ga(V) = add_cycle_vertices(Gc(V))
 Ga(E) = add_cycle_edges(Gc(V))
 Ga(V,E) = add_IO_nodes(G(V,E),Ga(V,E))
 return Ga(V,E)
 }
```

Algorithm 11 Procedures for Algorithm 10

Procedures:

```
 Create_Edges(G(E)) {
     e1 = G(E)
 While (e1){ // adding edges
         if (e 1 != feedback\_edge)
          add e1 ->Ga(E)
         return Ge(E)
         }
     }
 cycle_path_duplicate(Gc(V,E),ncopy) {
     For ncopy =1 to copy {
          // making ncopy duplications of cycle path
      Vncopy = Gc(V), \[for example:vncopy = v1 , if ncopy =1\]
Encopy = Gc(E)
          v3 = lastnode of (Vncopy), v4 = firstnode of (Vncopy+1)
          ec = v3,v4
        add ec -> Gc(E)
    Gc(V,E) = Gc(V,E) + (Vncopy,Encopy)
          }
    }
```

Algorithm 12 Procedures for Algorithm 10

```
add_cycle_vertices(Gc(V)) {
 u2 = Gc(V)
 While (u2) { //adding vertices in cycle
 v2 = Ga(V)
 while (v2){
 if v2 != u2 , add u2 -> Ga(V)
 }
    }
   return Ga(V)
   }
add_cycle_edges(Gc(E)) {
 e2 = Gc(E) //adding edges in cycle
 While (e2) {
 if (e2 !=e1) , add e2 -> Ga(E)
 }
 return Ga(E)
  }
```

Algorithm 13 Procedures for Algorithm 10

```
connect\_IO\_nodes(Ga(V,E),G(V,E)) \{
 for all input nodes i in G,
 if there is an edge e = G(E) , with e =( i, v),
 for ncopy =1 to copy,
 add edge e = (i,vncopy) -> Ga(E)
 for all output nodes out in G.
 if there is an edge e = G(E) , with e =(v,out),
 for ncopy =1 to copy,
   add edge e = (vncopy,out) -> Ga(E)
return Ga(V,E)
```

6.5 Working Example and Results

The overall methodology is explained further by showing the flow through two example circuits, namely the majority gate-based C-element and benchmark ram-read-sbuf [CKK+97].

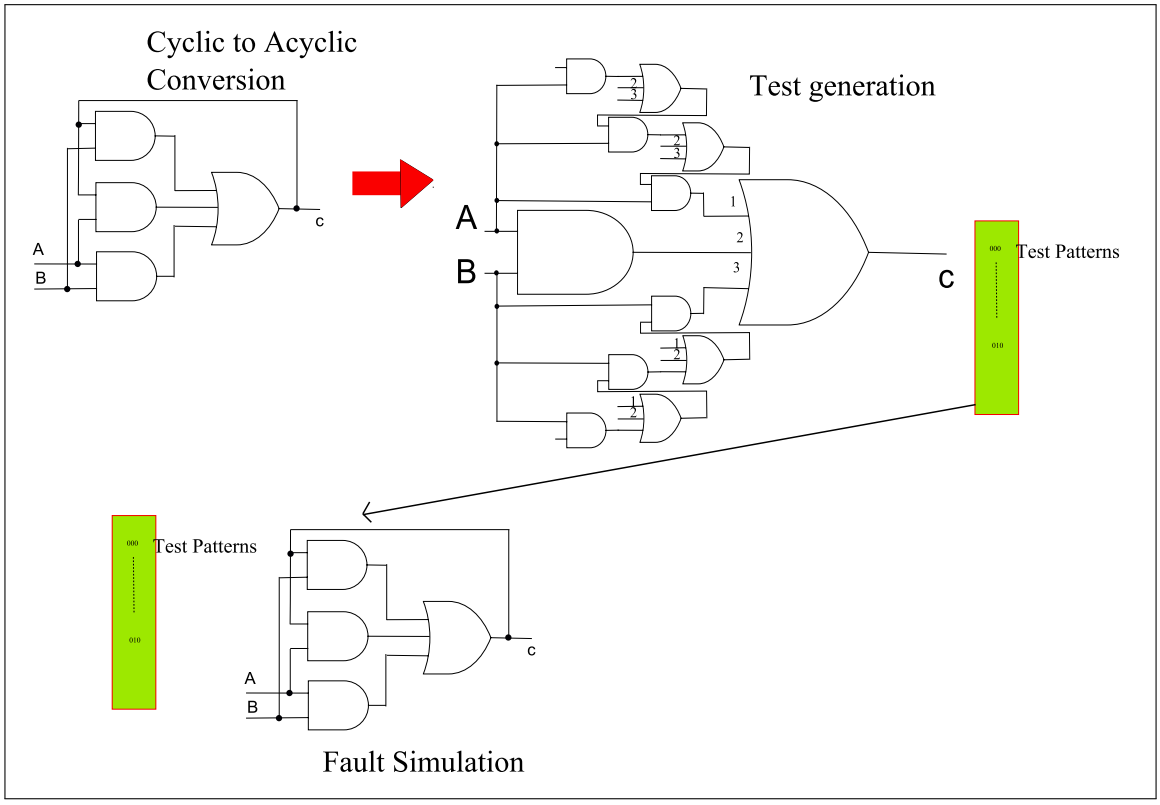


Figure 6.4: C-element Testing

6.5.1 c-element

A majority gate-based C-element is shown in Figure 6.3. The circuit is cyclic and consists of four gates and two local feedback loops. Since the c-element does not have memory elements, the scan selection algorithm does not select any scan element. This example is provided to show the cyclic-to-acyclic conversion in the absence of a memory element. Thus if no memory element is present and the circuit has loops, the cyclic to acyclic converter will produce an equivalent acyclic circuit.

The acyclic circuit in Figure 6.4 consists of 3 inputs, 1 output and 11 gates. The converted acyclic circuit is fed to the Synopsys's Tetramax to obtain the test patterns. The test patterns obtained are 111, 000, 100, 010, 111, 101, 011 for the pins A, B, and C, respectively, with C being the initialization pin. The actual patterns used to test the real cyclic circuit are therefore the first two bits of the above sequence. Tetramax was also used for fault simulation and the fault coverage is 100%.

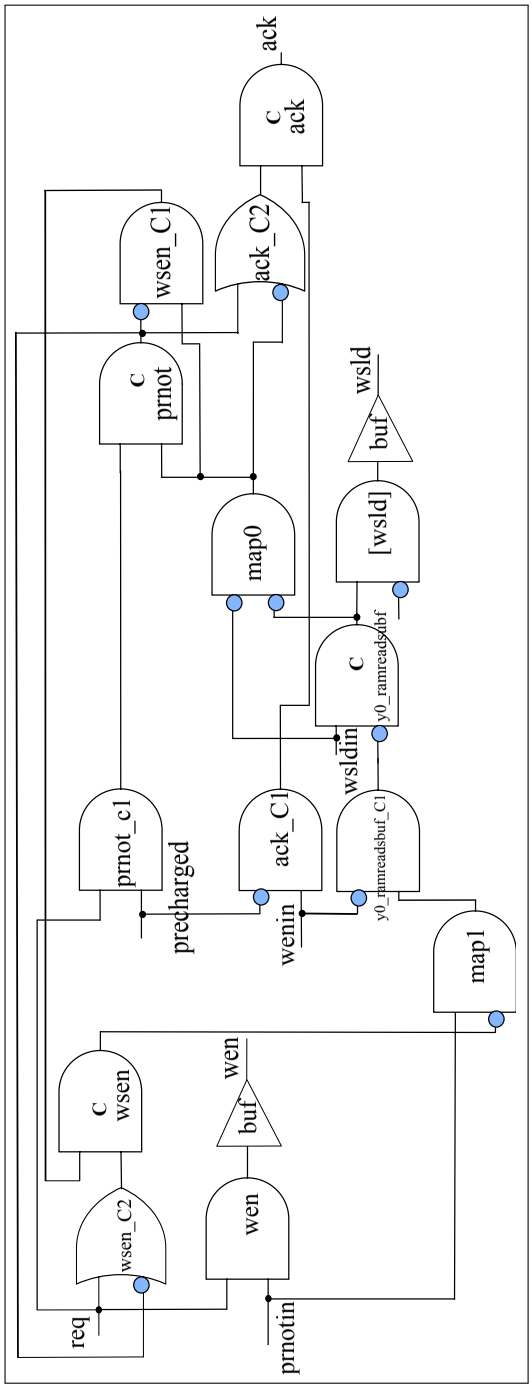


Figure 6.5: Benchmark "ramreadsbuf"

6.5.2 ram-read-sbuf

The benchmark "ramreadsbuf" (shown in Figure 6.5 is cyclic and consists of 10 combinational gates,2 buffers and 4 c-elements, constituting 4 local loops and 2 global loop. Three strongly-connected components are identified forming a cycle list with 1 cycle. By applying scan selection algorithm, the c-element, "y0_ramreadsbuf", will be selected. It will be selected at the first pass as it constitutes the cycle 1, emptying the cycle list to halt the algorithm. As explained earlier, the circuit has 3 c-element left without being scanned and hence it constitute to the local loops. So the circuit is fed to the cyclic-to-acyclic converter. The resulting circuit will be a partially scanned circuit free from local loops. The acyclic circuit is fed to the Tetramax tool, to generate the test patterns. These test patterns are then fault simulated over the orginal partially-scanned DUT to obtain the fault coverage. The test coverage for this benchmark is 96.34% for the ABLOB1 method and 94.59% for the AGLOB2 method.

6.5.3 Experiments and Results

The proposed methods were applied to 24 asynchronous circuits synthesized using Petrify[CKK+97]. The experimental results and their analysis based on the evaluation metrics namely fault coverage, number of patterns and the area overhead are discussed in detail in this subsection. The analysis is made based on comparing the two methods, AGLOB1 and AGLOB2, with the Full scan first. Then the two proposed methods are compared with each other. The fault coverage comparison of the proposed methods with Full scan method is shown in the Table 6.1, and the Table 6.2 shows the comparison of the number of test patterns generated for each method.

AGLOB1 Vs Full Scan

The fault coverage comparison of the AGLOB1 method with the full scan design is shown in Fig 6.6. Except for the benchmarks master-read, mmu, seq_mix and nakpa, AGLOB1 generated test provided fault coverage of 90% and above, for all the circuits. For the benchmarks ebergen, nowick and subf-ram-write, this method achieved fault coverages of more than 97%. It should be noted that for the benchmark mr1, which has the highest number of C-elements and global loops, this method achieved fault coverage of 95.51%. Comparison of the number of patterns generated by the AGLOB1 method with that of the Full scan method is shown in the Figure 6.7. Clearly, the number of patterns generated for the test is reduced for the AGLOB1 method. This is especially true for the benchmarks mr1, mmu, master-read , trimos-send and wrdatab, for which the reduction in the number of patterns was very high. For trimos-send, the reduction was more than 50%. For this benchmark full scan generated 46 test patterns whereas the AGLOB1 generated only 21 test patterns. The reduction in fault coverage due to more than halving the number of test patterns is approximately 10 %. For mr1, with 5% reduction in the

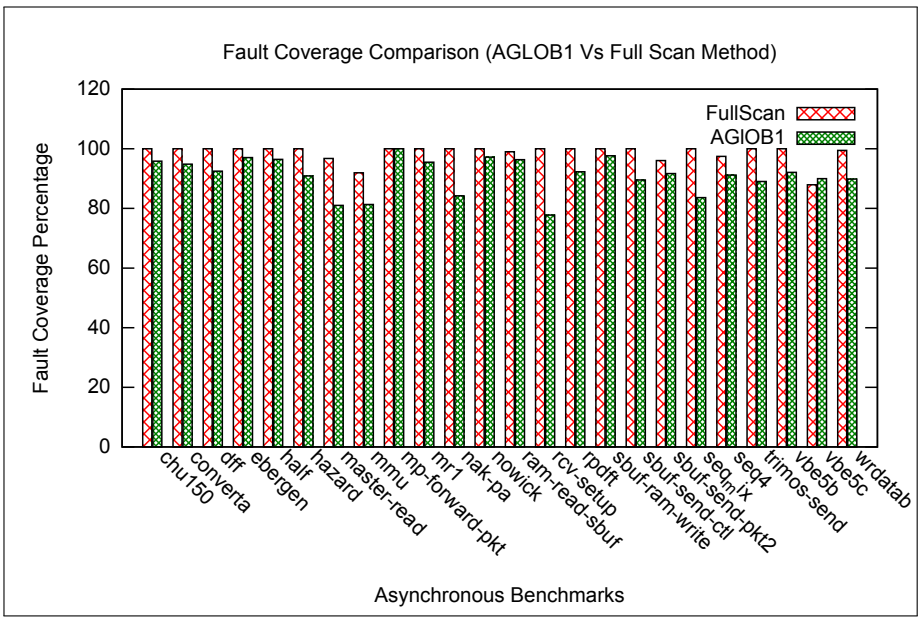


Figure 6.6: Fault Coverage - Full Scan Vs AGLOB1

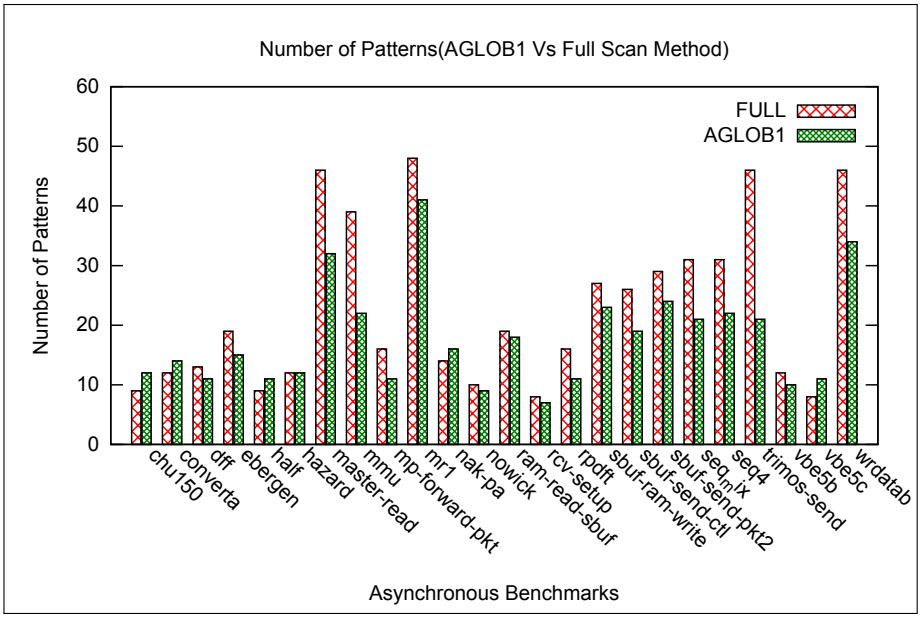


Figure 6.7: Number of Patterns - Full Scan versus AGLOB1

fault coverage, AGLOB1 can generate tests with 10% reduction in number of test patterns. It covered 95% of faults with only 41 patterns, whereas the full scan method needed 47 patterns.

AGLOB2 Vs Full Scan

The fault coverage comparison of the AGLOB1 method with the full scan design is shown in Figure 6.8. AGLOB2 achieved fault coverage closer to full scan method for most of the

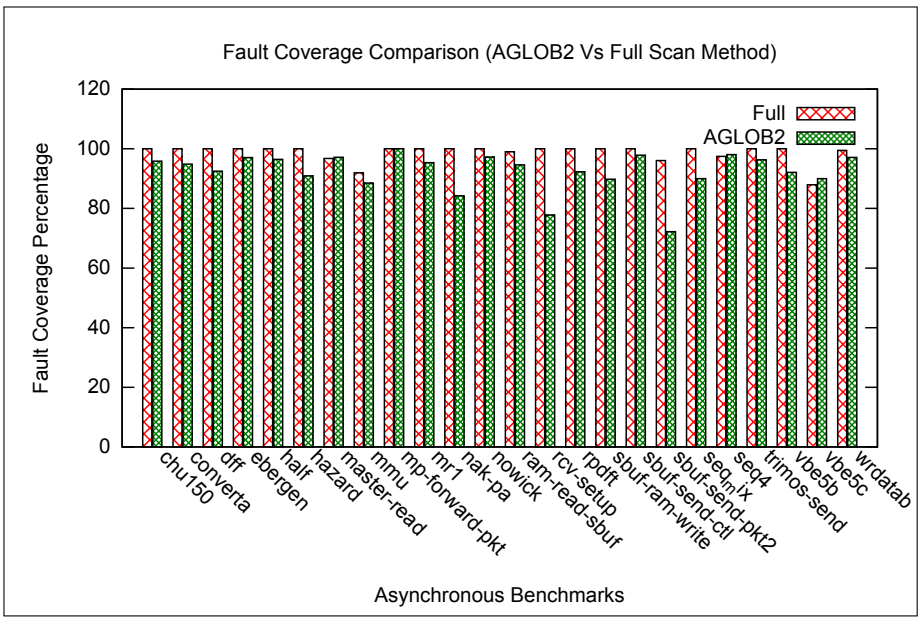


Figure 6.8: Fault Coverage - Full Scan versus AGLOB2

circuits. Only for three benchmarks, namely nak-pa, rcvsetup and sbuf-send-pkt2, was the fault coverage lower. For most of the circuits, the fault coverage was between 94 - 98%. For the benchmarks, wrdatab, sbuf-send-ctl, and ebergen, the fault coverage was greater than 97%. As this method concentrates on the nodes with higher degree, it eventually cut most of the loops and provided higher fault coverage. Next the number of patterns generated for the AGLOB2 and the full scan method were compared as shown in the Figure 6.11. The reduction in the number of patterns was not as good as AGLOB1, which may be attributed to the higher fault coverage. However, for the benchmarks, master-read, trimos-send and wrdatab, the reduction in the number of patterns compared to the full scan method was higher. For trimos-send and wrdatab, AGLOB2 generated 35 and 37 patterns, respectively, whereas for the full scan method there were 46 patterns each.

AGLOB1 Vs AGLOB2

Finally, the fault coverage comparison for the two proposed methods AGLOB1 and AGLOB2 were carried out, as shown in Figure 6.10. For the benchmarks with the lower number of C-elements, these two methods have achieved similar fault coverage. This is due to the fact that, when the number of C-elements are lower, and if one of them is inside the global loops, then both these algorithms will choose the same element. This is exhibited clearly in the benchmarks, chu150, converta, dff, half and hazard. For the benchmarks with higher number of C-celements and global loops, AGLOB2 achieved higher fault coverage. This is clearly seen from the result for the benchmarks, sbufsend-ctl, seq4, trimos-send and wrdatab. For all these benchmarks, AGLOB2 achieved nearly 8% higher fault coverage than the AGLOB1.

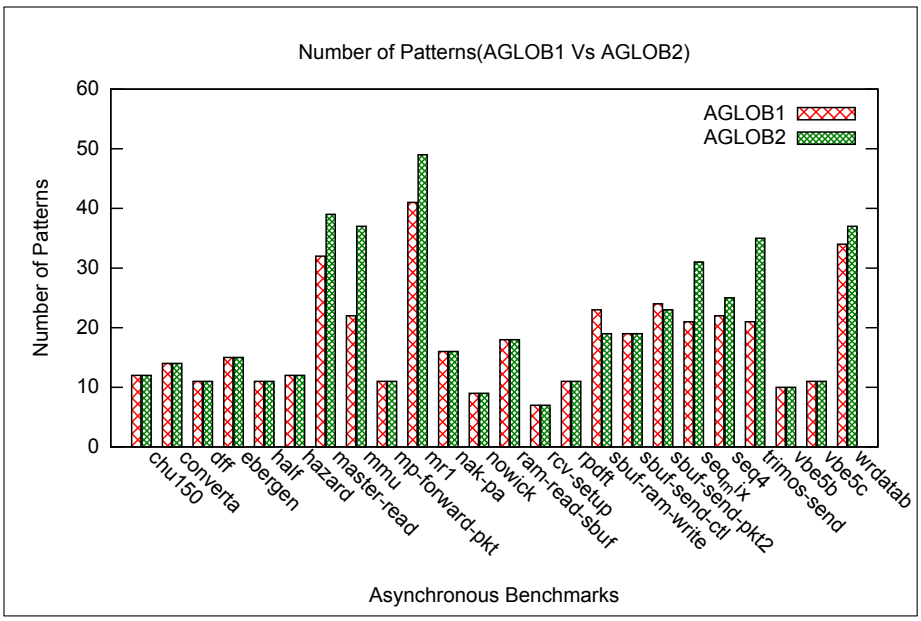


Figure 6.9: Number of Patterns - AGLOB1 vs AGLOB2

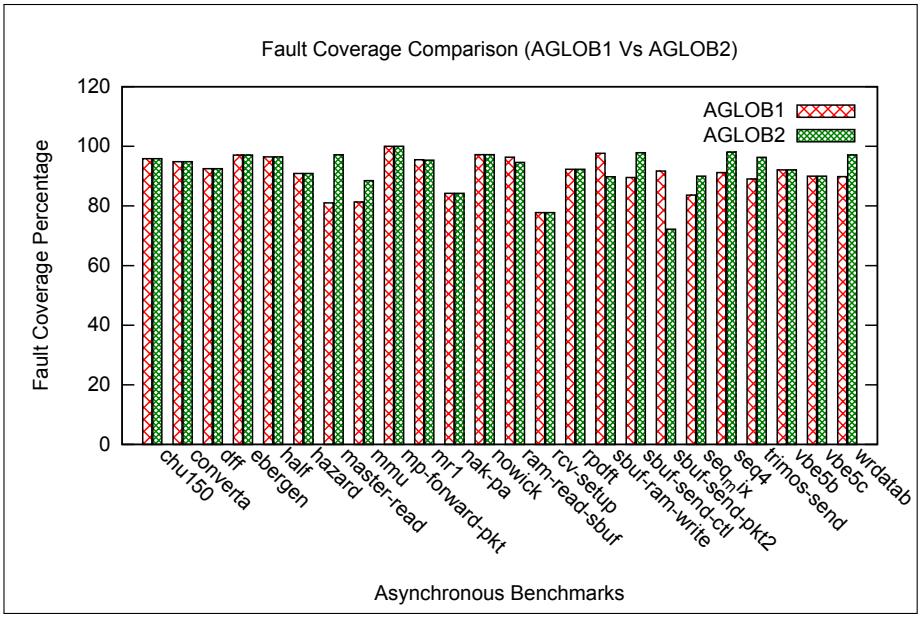


Figure 6.10: Fault Coverage - AGLOB1 vs AGLOB2

Comparison of number of patterns generated by the AGLOB1 method, with that of AGLOB2 method is shown in Figure 6.9. As mentioned earlier, AGLOB2 generated higher number of patterns compared to AGLOB1, due to the fact that AGLOB 2 selected more C-elements than AGLOB1. But interestingly, for some benchmarks ALGOB2 generated same number of patterns as AGLOB1, but attained higher fault coverage. This can be seen for the benchmark sbuf-send-ctl. Both the methods generated 19 patterns as test, but AGLOB2 had higher fault coverage of 97.83% and AGLOB1 achieved only 89.86%.

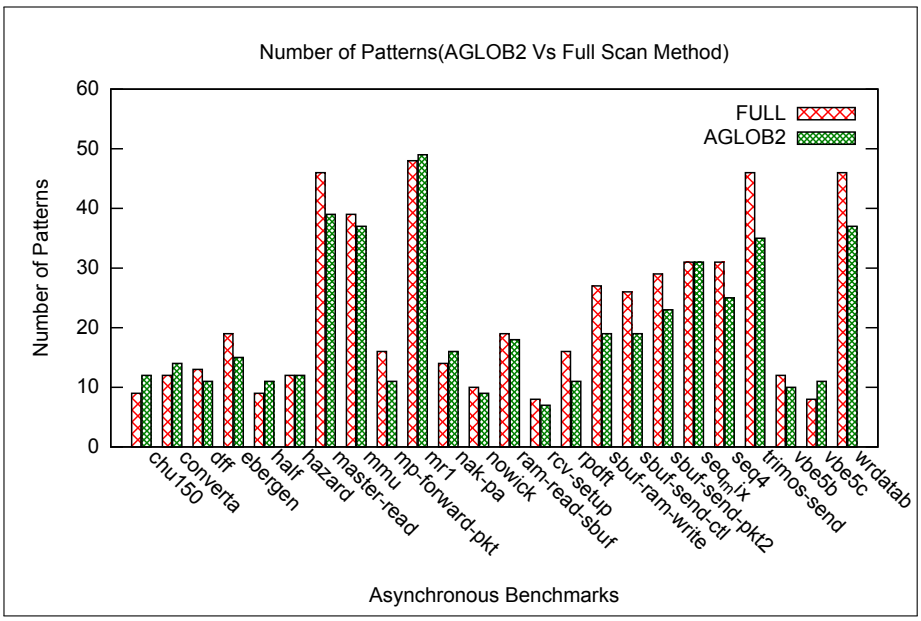


Figure 6.11: Number of Patterns - Full Scan vs AGLOB2

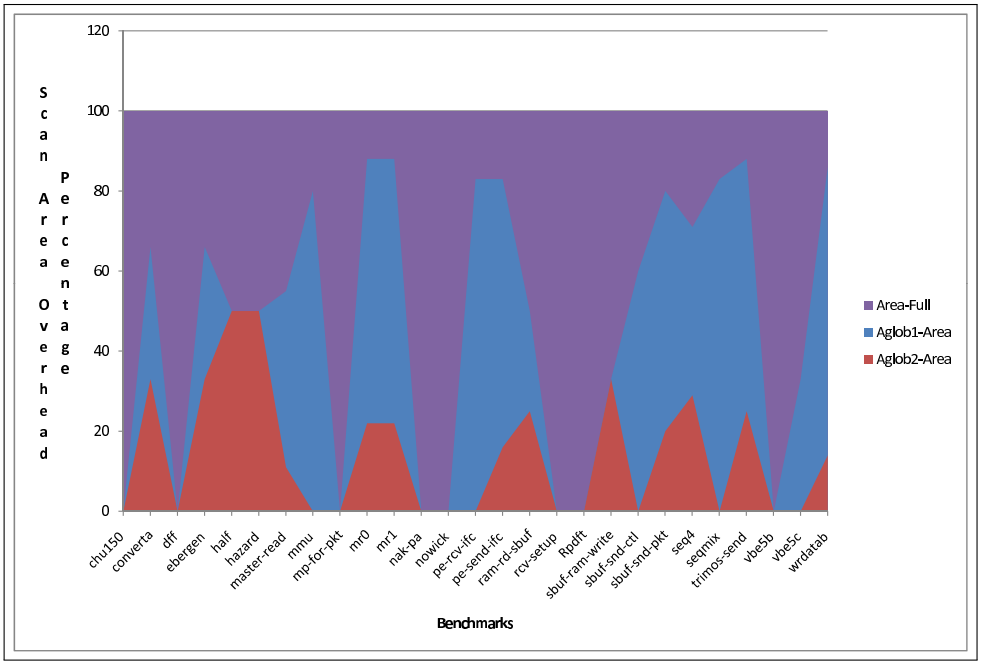


Figure 6.12: Results - Area Overhead comparison

The area overhead for the AGLOB1 and AGLOB2 method is shown in Table 6.3. The Figure 6.12 shows the graph which differentiates the area overhead percentage for Full scan, AGLOB1 and AGLOB2 methods. In several benchmarks scan elements were not required at all (100% reduction), while at the very least these methods required half the number of scan elements compared to full-scan.

Complexity

Since both AGLOB1 and AGLOB2 enumerates the S-graph (containing memory elements as vertices) the complexity of these two algorithms is O(mn), where n is the number of memory elements and n is the number of connections between them. For the cyclic-to-acyclic conversion the upper bound is dominated by the GR algorithm of O(m/2).

6.6 Conclusion

A partial scan test pattern generation method for asynchronous circuits based on strongly connected components(SCC) and cyclic to acyclic conversion was introduced in this chapter. The selection of the state elements that will be “scanned” is based on enumerating the SCC of the equivalent S-graph of the circuit similar to conventional method and generating the acyclic version of the resulting partial scan circuit. Test coverage was improved compared to test generated from original latch free circuit without applying DFT. The proposed method has been applied to a number of benchmarks achieving improvement in fault coverage compared to the original circuit. In total, 24 circuits tested with the fault coverage range of 0 - 82.35% for original circuit, improved to the range of 66.24 - 97.83% with proposed method. Further improvement of fault coverage closer to full scan is achievable by adding minor DFT circuit. Future work will involve exploring more algorithms for scan selection and cyclic to acyclic conversion of asynchronous circuits.

Table 6.1: Fault Coverage Comparison

Benchmarks	Full	AGLOB1	AGLOB2
chu150	100	95.83	95.83
converta	100	94.83	94.83
dff	100	92.5	92.5
ebergen	100	97.06	97.06
half	100	96.43	96.43
hazard	100	90.91	90.91
master-read	96.76	81.01	97.13
mmu	91.95	81.3	88.49
mp-forward-pkt	100	100	100
mr1	100	95.51	95.35
nak-pa	100	84.21	84.21
nowick	100	97.22	97.22
ram-read-sbuf	99.02	96.34	94.59
rcv-setup	100	77.78	77.78
rpdft	100	92.31	92.31
sbuf-ram-write	100	97.66	89.77
sbuf-send-ctl	100	89.53	97.87
sbuf-send-pkt2	96.03	91.67	72.22
seq_mix	100	83.62	90
seq4	97.44	91.18	98.04
trimos-send	100	89.04	96.3
vbe5b	100	92.11	92.11
vbe5c	87.93	90	90
wrdatab	99.46	89.86	97.09

Table 6.2: Comparison of Number of Patterns

Benchmarks	FULL	AGLOB1	AGLOB2
chu150	9	12	12
converta	12	14	14
dff	13	11	11
ebergen	19	15	15
half	9	11	11
hazard	12	12	12
master-read	46	32	39
mmu	39	22	37
mp-forward-pkt	16	11	11
mr1	48	41	49
nak-pa	14	16	16
nowick	10	9	9
ram-read-sbuf	19	18	18
rcv-setup	8	7	7
rpdft	16	11	11
sbuf-ram-write	27	23	19
sbuf-send-ctl	26	19	19
sbuf-send-pkt2	29	24	23
seq_mix	31	21	31
seq4	31	22	25
trimos-send	46	21	35
vbe5b	12	10	10
vbe5c	8	11	11
wrdatab	46	34	37

Table 6.3: Area Overhead - expressed as percentage of extra scan elements

Benchmarks	AGLOB1(%)	AGLOB2(%)
chu150	-	-
converta	66	33
dff	-	-
ebergen	66	33
half	50	50
hazard	50	50
master-read	55	11
mmu	80	0
mp-fwd-pkt	0	-
mr1	88	22
nak-pa	0	-
nowick	-	0
ram-rd-sbuf	50	25
rcv-setup	-	0
rpdft	-	0
sbuf-ram-write	33	33
sbuf-snd-ctl	60	0
sbuf-snd-pkt2	80	20
seq4	71	29
seq_mix	83	0
trimos-snd	88	25
vbe5b	-	-
vbe5c	33	-
wrdatab	86	14

Chapter 7

ASCP - A Set Covering Problem based Test Generation for Asynchronous Circuits

7.1 Introduction

A partial scan test generation method for asynchronous circuits based on the set covering problem is introduced in this chapter. A cycle enumeration algorithm with linear time complexity is used to efficiently enumerate the cyclic paths in the asynchronous circuits. The set covering problem is mapped over the partial scan selection problem to find the flipflops/C-elements to be scanned for test purposes. The scan selection procedure was run over 27 asynchronous benchmarks to compare the fault coverage and area overhead with the full scan design. Scan Area overhead reductions between 11% to 100% were achieved.

Contributions of this work are:

- A partial scan selection procedure for asynchronous circuits
- Facilitating the automatic test pattern generation for asynchronous circuits.
- Integration of the partial scan procedure with an industrial ATPG tool

This Chapter is organized as follows: Section 2 gives a background on cycle enumeration and set covering problems; Section 3 briefly describes our approach for partial scan selection; The algorithms for the methodology are described in Section 4; experimental results are analyzed in Section 5, with conclusions presented in Section 6.

7.2 Preliminaries

Due to the cyclic nature of the circuits being considered, the following preliminary definitions are added for clarity.

Definition.1 S-graph

A S-graph S(V,E) is a graph induced from the original graph G(V,E) by removing the node set S1(V,E) , where the vertices in S1(V,E) contains only the vertices corresponding to the flipflops/memory elements.

Definition.2 Path

A Path from vertex v1 to vertex v2 is a set of vertices encountered when traversing from v1 to v2 by visiting each of them one time.

Definition.3 Cycle A cycle in the graph is a set of vertices visited when traversing from vertex v1 and back to the same vertex.

7.3 Algorithms

7.3.1 Cycle enumeration

In [Uno03], a linear time cycle enumeration algorithm was proposed. This was based on the path enumeration algorithm introduced by [RT75]. EnumPath takes in the graph G(V,E), source s, target vertex t, s-t path P and an empty set. If the source is the same as the target, then the s-t path is added to the empty set. Otherwise, h, the adjacent vertex to s is chosen. A breadth first search is made from the target vertex t in the graph G-(s,h). If a path Q exists from s to h then a recursive call of EnumPath is made with Q as the path, and G-(s,h) as the graph, otherwise the vertex s is removed from the graph and EnumPath (shown in Figure 7.1) is recursively called over the graph G-s with h as the source. The empty set I, is updated during all the calls. The time complexity of this algorithm is O(|V|.(|E|+|V|)) for each path/cycle since one iteration takes up to O(|V|+|E|) time, and the depth of the recursion is O(|V|). The time complexity is further reduced by noting the vertices visited in the previous iteration. Therefore the complexity is O(|V|+|E|).

7.3.2 SCP algorithm

An efficient algorithm for the set covering problem was proposed in [EA00]. The set covering problem can be formulated as follows.

Given a m-row, n-column matrix a_{ij}, and a n-dimensional integer vector(w_j), the problem

```
EnumPath (G=(V,E), s, t, P, I)
If s = t then
      output I$\cup$ {s} ;
      return
h := the next vertex to s in P
Breadth first search starting from t in G-(s,h)
If a path Q from s to t exists then
      call EnumPath (G-(s,h), s, t, Q, I)
call EnumPath (G-s, h, t, P, I$\cup$ {s})


EnumCycle (G=(V,E))
For each edge (t,s)
    Remove (t,s) from G
    Call EnumPath (G,s,t,$\emptyset$)
End for
```

Figure 7.1: Function - EnumPath

consists of finding a subset of columns covering all the rows and having minimum total weight. A row i is covered by a column j if the position a_{ij} is equal to 1. In terms of a constrained optimization problem, this can be formulated as,

Minimize $\sum_{i=0}^{n} w_j x_j$, Subject to the constraints

$x_j \in 0,1, j = 1,...n$

$\sum_{j-1}^{n} a_{ij}.x_j \geq 1, i = 1,..,m.$

The variable x_j denotes whether the column j belongs to the solution or not. The m constraint inequalities are used to denote the requirement of each row being covering by at least one column. The weight, w_j, is a positive integer giving the weight of the column.The algorithm is shown in Figure 7.2 - 7.4.

The algorithm underlying the test methodology is shown in Figure 7.5 . The graph operated over by the algorithm shown is the S-graph, which is the graph composed of only the memory elements as vertices. The list L is generated by running the Enum_Cycle function in Figure 7.5. The resulting list of cycles and the corresponding vertices present in the cycle are represented as a matrix set with value 1 when the vertex is present in the cycle, or 0 otherwise. The constructed matrix is then processed by the function Wscp in Figure 7.2. The list of scan elements selected by Wscp is stored in Set S. Using the set S, the circuit under test is updated by replacing the

```
 Wscp()
 Begin
   Recompute_core()
  Sbset <- {1..ncol}
  S <- {};
  For {1.. param.number_of_iterations} do
    If(core_selection()) Recompute_Core(); Endif;
     S <- Greedy(S);
     S <- Optimize(S);
     If (value (S) <= value(Sbset) ) Then Sbest <- S; Endif;
     S <- Select_Partial_Cover(Sbest);
  Endfor
  Return Sbest
End
```

Figure 7.2: Function - Wscp

```
Funciton Greedy(var S)
  Begin
    While(S is not a cover) Do
      //select and add one column to S
      S <- S + select_add();
      //remove 0 or more columns from S
      While (remove_is_okay() ) Do
        S <- S - select_rmv();
      EndWhile;
    End While
    //S is a cover, without redundant columns
    Return S;
  End
```

Figure 7.3: Greedy Heuristic

```
Function Optimize (var S)
Begin
  Sup <- select_superior();
  While (sup not empty) do
    //select best column from Sup
    Best <- select_best();
    Sup <- Sup -best;
    // add superior and remove redundant columns from S
    If(best superior)
      S <- S + best;
      S <- S - select_redundant();
    Endif
  Endwhile
  //s is a cover, without redundant columns
  Return S;
End
```

Figure 7.4: Function - Optimize

corresponding set of C-elements into the scan-testable C-elements. Then, the resulting circuit is converted into acyclic circuit by running the CyclictoAcyclic function. The acyclic circuit is used for test generation and the test pattern generated is used to test the cyclic, but partial scan circuit (The resulting partial-scan ready circuit is still cyclic, as there will be few C-elements not being scanned). At this point the coverage of the circuit is checked for at least an user given percentage of coverage (X%), if the coverage is less than X%, the list containing the number of cycles each contribute to is created. If the contribution is more than 75% of the cycles, then the vertex is added to the scan list. Thus scan set is updated further for improved fault coverage. The detailed test flow is described in the next section.

```
Ascp (){
  List L = Enum_Cycle(S-graph);

  M = List of cycles x List of c=elements;

  S= list of scan elements = WSCP (M)

update: Update the circuits in the design to scan testable using S.

  Run cyclic to Acyclic Conversion.

  Run the test generation

  Check the fault coverage

  If fault coverage > X \% Go to "report"

  Else {

    List L = the number of cycles each vertex contribute

      For each element Ei in L

           If the contribution is > 75 \%

           add to scan list S.

       End For

     Go to "update"
    }

report:  Report fault coverage

}
```

Figure 7.5: Algorithm:ASCP

7.4 Methodology

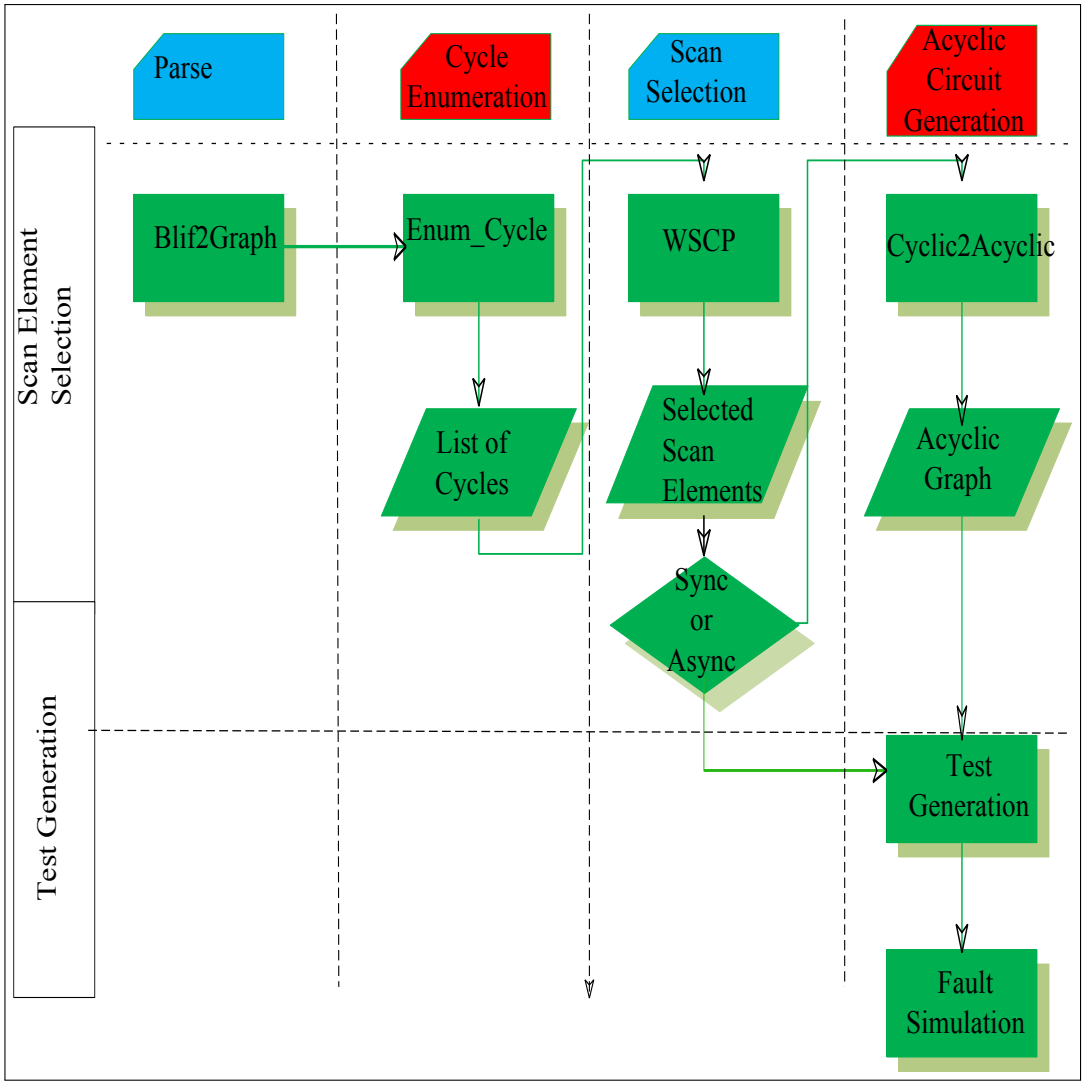


Figure 7.6: Test Methodology

The test methodology for the proposed partial scan test generation is described in this section. The Figure 7.6 gives the overall flow of the test method. The upper half involves the DFT method and the lower half involves the test generation and fault simulation as marked in the left-most column. The first phase of the flow is the circuit parsing, performed using the tool BLIF2graph. The next phase is called cycle enumeration, which involves enumerating all the cycles present in the s-graph. By applying the linear algorithm described in Figure 7.1 in Sub-section 7.3.2, all the cycles are listed with the corresponding vertex names. This list of cycles is passed to the "set cover" solution phase. Here the list of cycles is treated as rows and the vertices are treated as columns. Thus the minimum set cover computed by "wscp" will provide the list of vertices which forms the selected scan elements. At this stage, if the circuit

is a synchronous one, then it is passed directly to the test generation tool to produce the test patterns. If the circuit is an asynchronous circuit, then it undergoes another phase of cyclic-to-acyclic conversion. This should be taken care of in the asynchronous circuits, as the C-elements itself forms a self loop which is overlooked by the previous phases. Also, if some of the C-elements constitute around 75% of all the cycles, then they are also added to the scan list. Thus the acyclic equivalent of the asynchronous circuit is obtained at the end of the cyclic-to-acyclic phase. This circuit is then sent to the test pattern generator for testing the circuit.

7.5 Experiments and Results

The same set of benchmarks as before was chosen for experiments using the ASCP method. Table 7.1 shows the results for these benchmarks. The column marked 'cele' gives the number of C-elements in the circuits.The third column (marked 'scan') in the table gives the number of scan C-elements selected. The fourth column gives the fault coverage for the full scan method and the fifth one gives the fault coverage for the ASCP method. The area overhead from original and reduction from full scan is shown in sixth column. Table. 7.2 gives the comparison of number of patterns generated by the ASCP method with the fullscan method.

Fault Coverage

Figure 7.7 shows the graph comparing the fault coverage for the ASCP method and the full scan method. For the benchmarks, rcvsetup,hazard,chu150, converta and seq_mix, the fault coverage was between 80-90%. Out of all the 24 benchmark circuits, ASCP method achieved the maximum value of 98.03% fault coverage for the benchmark trimos-send which has more global loops and C-elements present in them. The fault coverage for this benchmark was 100% for the full scan. Also for the benchmarks ebergen, sbuf-send-pkt, seq4, vbe5b, vbe5c, and wrdatab the fault coverage was more than 95%. This method achieved 100% fault coverage for the benchmarks mp-forward-pkt, vbe5c and vbe5c. There reason for this increase is that the algorithm selected all the C-elements for scan. Thus the result obtained was similar to a full scan. On the other hand, for the benchmark rcv-setup, no C-element was selected and the resulting circuit was same as the original circuit and hencee the fault coverage was very low. In circuits with one C-elements like this, scanning the single C-element will provide better fault coverage.

Number of Patterns

In the graph in Figure 7.8, the comparison is made between the number of patterns generated by the ASCP method and the Full scan method. The number of test pattern generated by the ASCP method is low compared to the full scan method for a majority of the benchmarks. Especially for the benchmarks masterread, mmu, trimossend and wrdatab, the number of patterns

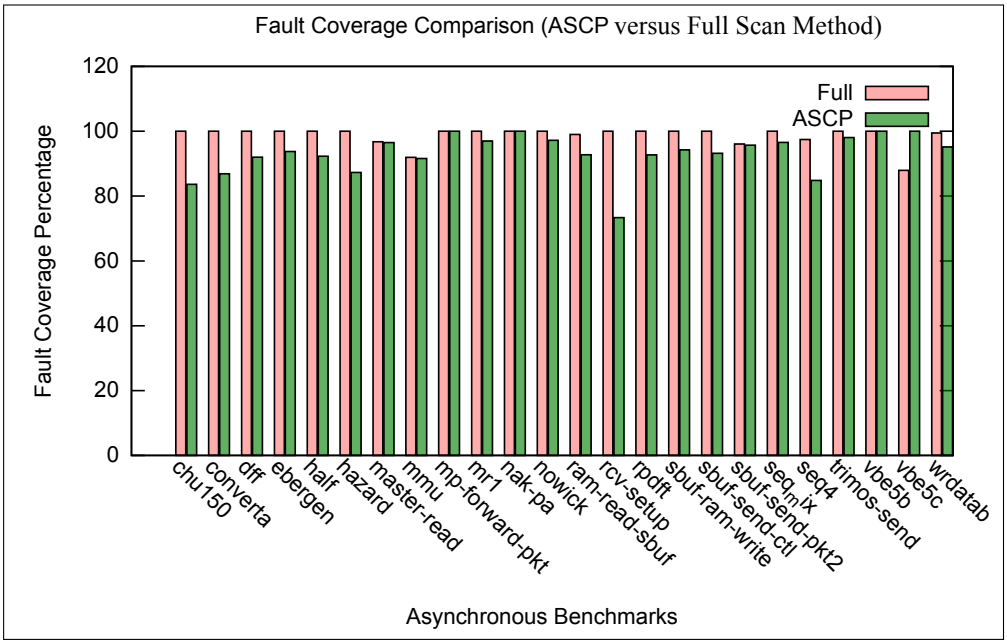


Figure 7.7: Fault Coverage Comparison - ASCP versus Full Scan Method

were less than 50% of that of the full scan method. For master-read the number of patterns generated by the ASCP method was 23 whereas for the full scan it was 46 (exactly 50% more). Also for the benchmarks, sbuf-send-pkt2 and wrdatab, the test patterns for ASCP were 8 and 18,respectively, whereas for full scan method it is, 29 and 46, respectively. Interestingly, for sbuf-send-pkt2, the fault coverage attained was almost the same as that of full scan with only 8 test patterns generated by ASCP. For wrdatab as well, almost 40% reduction in test pattern decreased the fault coverage only by approximately 4%.

Area Overhead

Figure 7.9 gives the area overhead comparison for the proposed ASCP method and the full scan method. The scan area overhead for the ASCP method was proportional to the number of patterns. As seen from the graph, the benchmarks master-read, mmu , trimos-send and wrdatab had lower number of C-elements scanned compared to the full scan method. But the number of patterns generated were less and the fault coverage was almost above 95%. For the ones with only red lines, the algorithm did not choose any C-elements as there were only one or two C-elements present and they were not inside a global loop to be chosen to be broken. But the fault coverage obtained without scanning the C-element were reasonable.

Complexity

The complexity of the ASCP method is the summation of the complexity of the WSCP algorithm and the cycle enumeration WSCP. have been extensively experimented with the larger graph benchmarks (larger compared to the millions of gates). The cycle enumeration algorithm

Table 7.1: ASCP Versus Full Scan - Fault Coverage Comparison

Ckt	cele	Scan	Full%	ASCP%	Area%
chu150	2	1	100	83.61	50
converta	3	1	100	86.89	66.66
dff	2	1	100	92	50.66
ebergen	3	1	100	93.75	66.66
half	2	1	100	92.31	50
hazard	2	1	100	87.27	50
master-read	9	3	96.76	96.48	66.66
mmu	6	2	91.95	91.6	66.66
mp-for-pkt	3	1	100	100	66.66
mr1	9	8	100	96.95	11.11
nak-pa	4	1	100	100	75
nowick	1	0	100	97.22	100
ram-rd-sbuf	4	2	99.02	92.73	50
rcv-setup	1	0	100	73.33	100
Rpdft	1	0	100	92.68	100
sbuf-ram-write	6	2	100	94.23	66.66
sbuf-snd-ctl	5	3	100	93.18	40
sbuf-snd-pkt	5	3	96.03	95.69	40
seq4	7	4	100	96.55	42.88
seq_mix	6	3	97.44	84.81	50
trimos-send	8	4	100	98.03	50
vbe5b	2	1	100	100	50
vbe5c	3	1	87.93	100	66.66
wrdatab	8	4	99.46	95.15	50

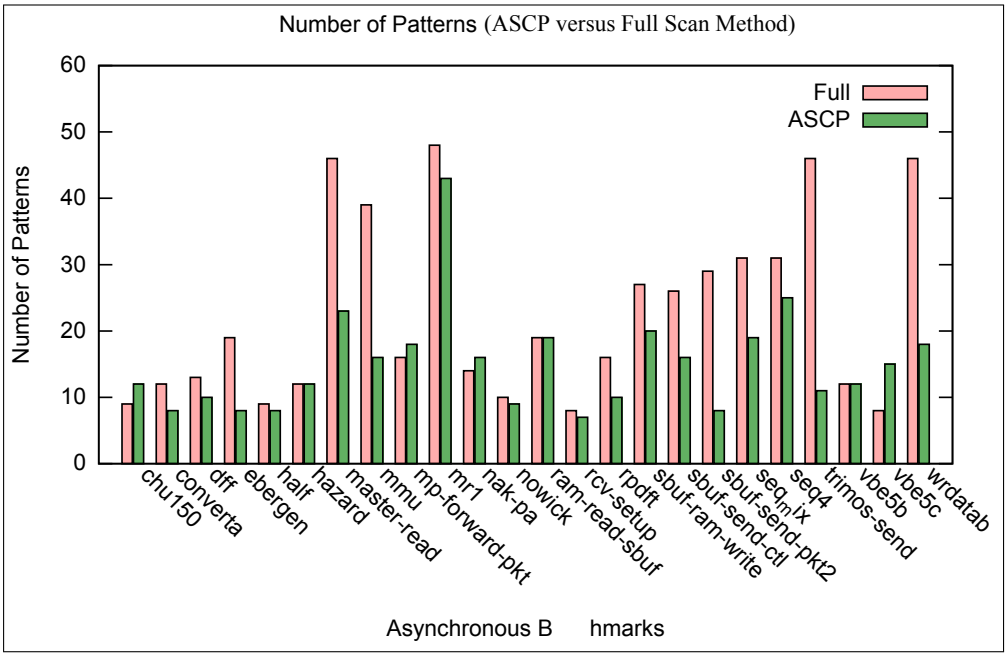


Figure 7.8: Number of Patterns - ASCP versus Full Scan Method

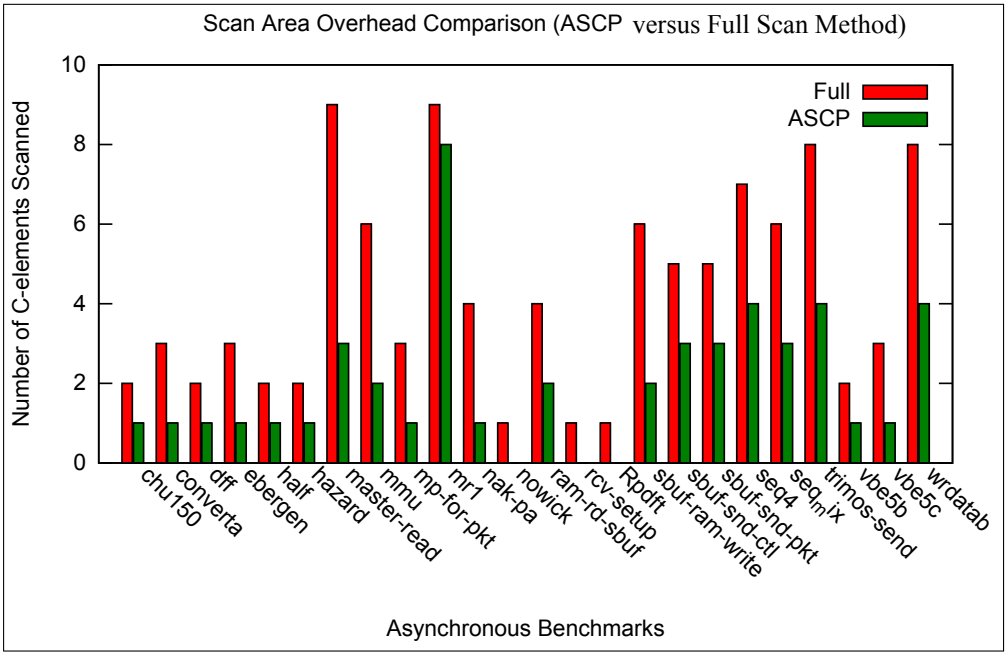


Figure 7.9: Comparison of number of scanned C-elements for 27 benchmarks (X-axis=Circuit name, Y-axis = Scan Area Overhead Percentage)

has a complexity of O (|V||E|).

7.6 Conclusion

A partial scan selection method was introduced. A cycle enumeration algorithm with linear time was used to efficiently enumerate the cyclic paths in the asynchronous circuits. The set covering problem was mapped over the partial scan selection problem to find the flipflops/C-elements to be scanned for test purposes. The scan selection procedure was excercised in 24 asynchronous benchmarks. The method proposed shows reasonable fault coverage with the trade-off of area overhead and reduced area overhead compared to the full scan circuit with a trade off in fault coverage.

Table 7.2: Comparison of Number of Patterns

Benchmarks	FULL	ASCP
chu150	9	12
converta	12	8
dff	13	10
ebergen	19	8
half	9	8
hazard	12	12
master-read	46	23
mmu	39	16
mp-forward-pkt	16	18
mr1	48	43
nak-pa	14	16
nowick	10	9
ram-read-sbuf	19	19
rcv-setup	8	7
rpdft	16	10
sbuf-ram-write	27	20
sbuf-send-ctl	26	16
sbuf-send-pkt2	29	8
seq_mix	31	19
seq4	31	25
trimos-send	46	11
vbe5b	12	12
vbe5c	8	15
wrdatab	46	18

Chapter 8

ACLARION - High level circuit extraction for Asynchronous Circuit Testing

8.1 Introduction

This chapter is motivated by the requirements of a high-level extraction tool, which can represent the asynchronous circuit at a higher level of abstraction to identify the interconnection of combinational logic, registers and fanout nodes, yet preserve the netlist connectivity of the design.

The main contribution of this chapter is a high-level circuit extraction method for asynchronous circuits. Often sequential circuit test generation involves grouping of several memory elements together. For example, in the partial scan design introduced in Chapter 2, the main motivation was to select the subset of memory elements. Usually, the design netlist is described in terms of combinational gates, memory elements, fanouts and interconnections. If these circuits/design representations can be represented at a higher level, then the problem set for the scan selection algorithms can be considerably reduced. This chapter is motivated towards developing such an extraction method which will reduce the size of the circuit representation so that the higher level extracted representations can be used for the other DFT algorithms.

The organization of the chapter is as follows: Section 8.2 gives the background required for the description of the extraction method; Section 8.3 describes the basic functions required for the implementation of the ACLARION extraction method and an overview of the methodology; Section 8.4 describes the proposed heuristics for the Register clustering process; Section 8.5

describes the heuristics for the combination logic unit (CLU) clustering; Fanout clustering heuristics are introduced in detail in Section 8.6. Experimental results are analyzed in Section 8.7 with one working example, with the conclusion in Section 8.8.

8.2 Background

Asynchronous circuits use combinational loops to store state. There are two types of loops, namely global and local loops. Local loops are the combinational loops present in the state-holding gates like C-elements or set-reset latches. The familiar flip-flop also contains a local loop, but it is hidden from test tools since a flip-flop is a cell on its own in standard cell libraries and does not pose any problems during testing. Global loops are formed outside these gates and are used for creating asynchronous state machines. Asynchronous full-scan methods [BPvBK03] break all these loops in test mode using LSSD-type scan latches. This simplifies testing as the circuit becomes purely combinational in test mode. However, the area overhead is enormous, hence motivating our work on partial-scan methods.

S-graph:
A S-graph S(V,E) is a graph induced from the original graphG (V,E) (where V is the set of combinational gates/memory elements and E is the set of interconnections) by removing the node set S1(V,E) , where the vertices in S1(V,E) contains only the vertices corresponding to the flipflops/memory elements.

Path:
A Path from vertex v1 to vertex v2 is a set of vertices encountered when traversing from v1 to v2 by visiting each of them one time.

Cycle:
A cycle in the graph is a set of vertices visited when traversing from vertex v1 and when the traversal ends in the same vertex v1.

8.2.1 Clarion

Clarion is a circuit extraction tool [I.P94], in which a circuit is represented as a s-graph with 5 different nodes namely PI node, PO node, combinational node, sequential element node, and fanout node. The PI and PO nodes are single nodes connecting all the primary inputs and

primary outputs of the circuit in to one node, respectively. A method for high level circuit extraction based on this graph was developed for synchronous circuit in [I.P94].

8.3 High Level Circuit Extraction

The extraction method proposed in this chapter is based on clustering the combinational gates, memory elements and fanout nodes which was used for the circuit extraction of synchronous circuits in [I.P94]. Thus the three clustering methods used: 1) Combinational logic Clustering, 2) Register Clustering, and 3) Fanout Clustering forms the basis of this technique. All the three clustering processes are described next.

8.3.1 Method

The functions used in the construction of the heuristic for the Asynchronous Clarion (AClarion) are namely Span, Union, Intersection and Span. The steps involved in function Span is shown in Figure 8.1. The function takes as input the graph G, the vertex, vertex identifier (vertex_label in Figure 8.1), c-element index and an empty set called spanset. The recursive function makes a depth first search (DFS) over the graph until all the vertices spanning from the vertex input until it reaches the c-element boundaries. This function a plays major role in finding the input span and output span in the main algorithm. The gates spanned are added to the empty set provided as input called spanset.

The function union is an implementation of the Union operation, taking in two sets, s1 and s2, along with the referenced empty set result. The elements in the sets s1 are enumerated and added to that of set s2 and the resulting set is assigned to the set result.

The function Intersection is an implementation of the intersection operation, taking the sets, s1 and s2, and enumerates the elements in set s1 and s2 to find the common elements and add them to the set result.

The pseudocode of the function Overlap is shown in Figure 8.3. This function implements the overlap operation "σ" which is used to form the equivalence classes, namely input overlap and output overlap in the main Aclarion algorithm. The function takes in a list of sets named setlist, two sets s1 and s2, an array named intercheck and another empty set called loopcheck. The set s2 is assigned to this set loopcheck. The intercheck array holds the intersection information of all the sets in the setlist.

```
1
2  /*********************************************
3  Function Span
4  *********************************************/
5  void span ( graph g, vertex ver, vertex_label
6            gate_name, int  is_cele, set  & spanset )
7  {
8          for(vertex ai2 = adjacent_vertices(*ver,g))
9          {
10                Iff ai2 != c-element{
11                 {
12                    spanset.insert(ai2);
13                   span (g,ai2,gate_name,is_cele,spanset);
14                 }
15         else
16                 {
17                 spanset.insert(ai2);
18                 }
19         }
20 }
21
```

Figure 8.1: Function Span

The overlap operator is defined as follows:
For two sets s1 and s2, s1 σ s2, if
1) s1 $\cap$ s2 $\neq \phi$,
2) s1 $\cap$ s3 = ϕ , s2 σ s3, s3 $\in$ S.

The first condition is achieved by direct application of the intersection function over the sets s1 and s2 to any intersecting elements in them in the first step. If there is an intersection then the overlap function returns 1. If this condition is not true, then all the sets in set s1 are enumerated to find any set with which the set s1 is having an intersection. When the intersection is found, then that set s3 is checked for a overlap with the set s2. Thus, a recursive overlap function is executed with set s2 and s3. When the called overlap returns 1, then the function returns 1 else the function returns 0. To avoid the looping of the function over the sets of the setlist, the variable noloop is used to set a flag to check for the same set not being enumerated again and again in the setlist.

These three functions are used extensively in implementing the several functions of the Aclarion algorithm. In the next section the heuristics constructed to implement the Aclarion extraction method are described in detail. The overall framework of Aclarion is shown in Figure

8.2.

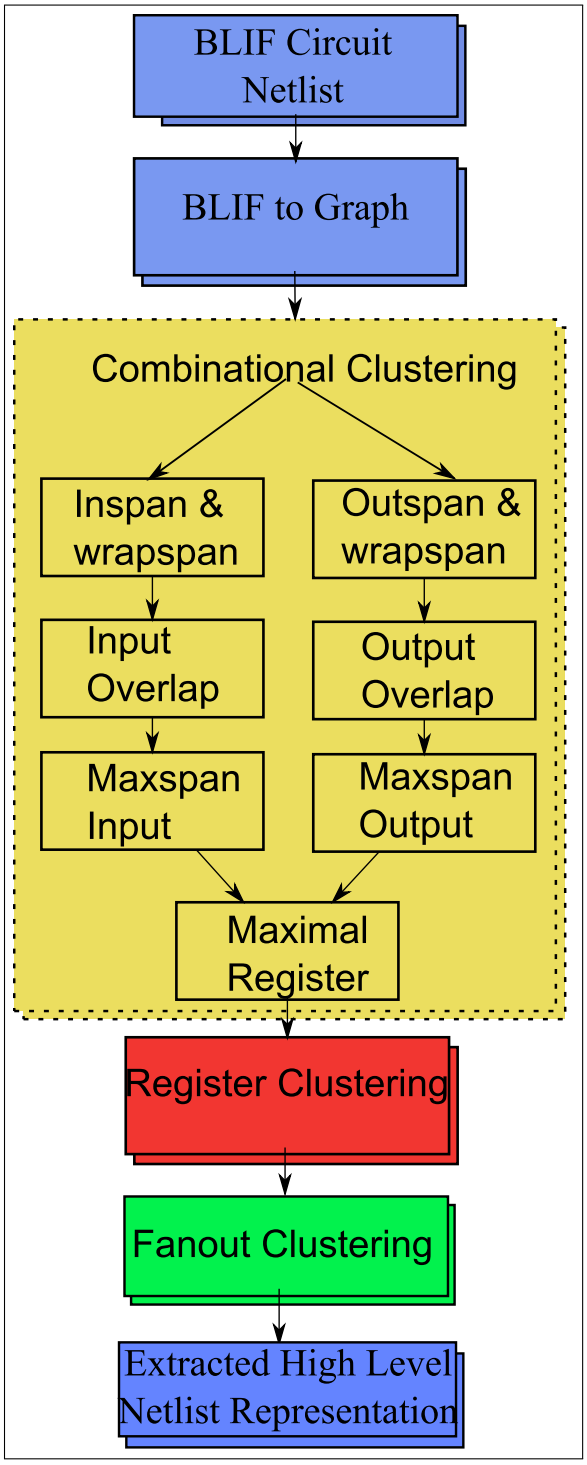


Figure 8.2: ACLARION Framework - Top-level View

```
/************************************
Function Overlap
************************************/
int overlap (list_of_sets  setlist,set  set1,set set2,
                      array intercheck,set  loopcheck)
{
  loopcheck = set2;
  set_iterator setiter;
  if((set1 != set2) && (intercheck[set1][set2] == 1))
  {
    loopcheck.clear();
    return 1;
  }
  else
  {
    for(unsigned int j=set1; j < size of setlist; j++)
    {
      if(set1 != j)
      {
        if(intercheck[set1][j] == 1 )
        {
          int noloop = 1;

          for(setiter = loopcheck.begin(); setiter
                != loopcheck.end();  setiter ++)
          {
            if((unsigned int)*setiter == j)
            {
              noloop = 0;
            }
          }
          if(noloop == 1)
          {
            if(overlap(setlist,set2,j,intercheck,loopcheck)== 1)
            {
              loopcheck.clear();
              return 1;
            }
          }
        }
      }
    }
    loopcheck.clear();
    return 0;
  }

```

Figure 8.3: Function:Overlap

8.4 Register clustering

To describe the register clustering process, the following terms have to be first defined.

8.4.0.1 Input span

Input span of a memory element is defined as the number of gates that spans along the path from the input of that element until the path encounters another memory element.

8.4.0.2 Output span

Output span of a memory element is defined as the number of gates that spans along the path from the output of that element until the path encounters another memory element.

8.4.0.3 Wrapped span

Wrapped span is the input/output span defined in terms of cyclic/asynchronous circuits. Thus the wrapped output span is the output span of the memory element in asynchronous circuits including the feedback/loop paths in the circuit. The wrapped input span is the input span of the memory element in asynchronous circuits including the feedback/loop paths in the circuit. The definition of wrapped span was introduced in [I.P94] and was not implemented as it was for synchronous circuit. In this proposed method, wrapped span is used for the clustering of registers/c-elements.

8.4.0.4 Maximal input span

A maximal input span is the equivalence class formed by the relation overlap on the set of input spans of the circuit.

8.4.0.5 Maximal output span

A maximal output span is the equivalence class formed by the relation overlap on the set of output spans of the circuit.

8.4.0.6 Maximal receiving register

The set of sinks of the input spans of the maximal input span is called maximal receiving register.

8.4.0.7 Maximal driving register

The set of sources of the output spans of the maximal output spans is called the maximal driving register.

8.4.0.8 Maximal Register

A maximal register R is defined as the maximal set of storage nodes given that for some R_r^i and R_d^j,[I.P94] R belongs to R_r^i and R belongs to R_d^j .

Thus by finding the Maximal registers for the given circuit, the memory elements in the circuit can be clustered to form a set of maximal registers.

8.4.1 Method

The sequence of steps in register clustering process are:

- Find the Output Span and Input span of all c-elements present in the circuit
- Find the maximum output span and maximum input span
- Find the maximum driving register and maximum receiving register
- Find the maximal register

The register clustering forms the vital part of the entire circuit extraction process. The scattered memory elements around the circuit are clustered strategically to bring out the high-level interconnection between the combinational gates. This is achieved by constructing an equivalence class on the set of spans through the overlap relation. The main functions involved in the register clustering process are Outspan, Outspan_wrap, Inspan, Inspan_wrap, OutputOverlap, InputOverlap, Maxspan_Output, Maxspan_Input, and Maximal_Register.
The pseudocode of the function Outspan is shown in Figure 8.4. The function takes in the graph "g" and outputs the list of spans for all the memory elements present in "g". This suffices for the synchronous circuits as they do not have loops or feedbacks in them due to their acyclic nature. But for the asynchronous circuits as mentioned in the earlier definition of wrapped span, the feedbacks occur in them due to their cyclic nature.

Figure 8.4 shows the pseudocode for the outspan_wrap function. The function takes in the graph "g", the list of outputspan created by the the outspan function and outputs an array output_wrapspan_check. The array stores the flag of all the spans which are wrapped (containing feedbacks) and those that are not (without feedback loops).

```
/*******************************************
Pseudocode: OutSPAN --- Depth First Search
*******************************************/
Input: Graph g1
Output: list_outspanset - list of outputspans

Outspan (graph g1){
        For each vertex v in graph G{
          If (v = c-element/latch) {
            currVertex = v;
      graph g2 = g1;
      set outspanset =   span(g2,currVertex);
      list_outspanset[vertex] = outspanset;
      }
    }
        return list_outspanset;
}


/*******************************************
Pseudocode: OutSPAN_Wrap --- Depth First Search
*******************************************/
Input: Graph g1,list_outspanset
Output:Array output_wrap_span_check

Outspan_wrap(graph g1,list_outspan_set){
        N = number of vertices of g1;
        Output_span_wrap[n]= 0;
        For each v in graph g1{
        If (v = c-element/latch) {
        For each vertex v1 in list_outspanset[v] {
                If(v1 == v){
                        output_wrap_span_check[v] = 1;
                        Break;
                        }
                }
        }
  return output_wrap_span_check[n];
}
```

Figure 8.4: Function:Outspan and Output WrapSpan

This information plays a major role in clustering the asynchronous cyclic circuits.

```
/************************************************
InSPAN --- Depth First Search
************************************************/
Input: Graph g1
Output: list_inspanset - list of input spans.

Inspan (graph g1){
        Graph g2 = reverse graph of g1;
        For each vertex v in graph g2{
                If (v = c-element/latch) {
                  currVertex = v;
            graph g2 = g1;
            set inspanset =   span(g2,currVertex);
            list_inspanset[vertex] = inspanset;
            }
          }
        return list_inspanset;
        }

/*********************************************
Input wrap span of all vertices*/
*********************************************/
Input: Graph g1,list_inspanset
Output:Array input_wrap_span_check

inspan_wrap(graph g1,list_outspan_set){
        N = number of vertices of g1;
        input_span_wrap[n]= 0;
        For each v in graph g1{
                If (v = c-element/latch) {
        For each vertex v1 in list_outspanset[v] {
                If(v1 == v){
                        input_wrap_span_check[v] = 1;
                        Break;
                        }
                }
        }
```

Figure 8.5: Function:Input Spand and Input WrapSpan

The Inspan function’s pseudocode is shown in the Figure 8.5. The function operates on the graph "g" to find all the inputspan of all the c-elements and latches present in the circuit. To use the span function defined previously, the graph is first reversed to form a new reversed graph "g2" and then for each memory element vertex element in the graph, the span function

is executed over the graph for that element.

```
/************************************************
OutputOverlap
************************************************/
Input: graph g,list_outspanset;
Output: matrix output_overlapcheck;

OutputOverlap(g,list_outspanset){
  size = size of list_outputspanset;
  array intersection[size][size], intersectioncheck[size][size];
  set loopcheck;
  For each set s1 in list_outspanset{
    For each set s2 in list_outspanset{
      Intersection[s1][s2] = intersection(s1,s2);
      If (s1 == s2 or interstion[s1][s2] = empty){
         intersectioncheck[s1][s2] = 0;
        }
      else{ Intersectioncheck[s1][s2] = 1;}
      }
    }

  For each set s1 in list_outspanset{
    For each set s2 in list_outspanset{
      Output_overlapcheck[set1][set2] =
         overlap(setlist,set2,set2,intercheck,loopcheck);
      }
    }
  }
```

Figure 8.6: Function:Output Overlap

The list of all the input spans containing the set of gates is output by this function. Figure 8.5 also give the pseudocode for the function inspan_wrap. This function takes in the graph g and the input span list and constructs the array input_wrap_span_check. This array stores the information on the feedback loops on the input spans similar to the outputspan_wrap function. Once all the outspans and inspans are constructed, the overlap operator is used over these spans

to find out the overlapping of the sets of the list of spans. Figure 8.6 gives the pseudocode for the function Outputoverlap. The graph g and the list of outspans are passed as input to these functions. First, an array named intersection of size n1 x n1 is constructed, where n1 is the size of the list of outspans. The sets that are intersecting are assigned the flag 1 in the place in the array corresponding to these sets. Then a new array named Output_overlapcheck is constructed. For each set s1 in the list of outspans, the overlap of this set with the other set are examined. This is carried out by passing the set s1 and other sets to the overlap function along with the intersection array and an empty set called loopcheck.

```
/*********************************************
InputOverlap
*********************************************/
Input: graph g,list_inspanset;
Output: matrix input_overlapcheck;

InputOverlap(g,list_inspanset){
  size = size of list_inputspanset;
  array intersection[size][size], intersectioncheck[size][size];
  set loopcheck;
  For each set s1 in list_inspanset{
    For each set s2 in list_inspanset{
      Intersection[s1][s2] = intersection(s1,s2);
      If (s1 == s2 or interstion[s1][s2] = empty){
         intersectioncheck[s1][s2] = 0;
      }
      else{ Intersectioncheck[s1][s2] = 1;}
    }
  }
}
For each set s1 in list_inspanset{
  For each set s2 in list_inspanset{
    Input_overlapcheck[setk][setl]
           =overlap(setlist,setk,setl,intercheck,loopcheck);
    }
  }
}
```

Figure 8.7: Function:Input Overlap

The overlap function returns a 1, if there is a overlap else it will return a 0. Thus the Output_overlapcheck array is constructed and returned as an output for this function.

Similarly, the overlap of all the sets in the list of inspans are examined by the function input_overlap. The inputs to this function are the graph g and the list of inputspans. Intersection

array is constructed first for all the sets in the list of inputspans. Then this array along with an empty set loopcheck is passed to the function overlap for each set. The resulting matrix input_overlapcheck is returned as the output from the function. Thus the input_overlapcheck and the output_overlapcheck matrices will be used to construct the maximum spans for the inspans and outspans.

Figure 8.8 gives the pseudocode for the function MaxSpan_Output. This function clusters the sets in the list of outspans which overlap with each other. Basic steps involved in this process are: 1) enumerating the matrix output_overlapcheck these two sets to a disjoint set, and 2) distinguish the wrapped span from unwrapped spans and construct two different maximum outputspans. For the first step which is straight forward to enumerate the output_overlapcheck. It should be noted that the overlapping sets are added to the disjoint set "ds", which creates the sets of c-elements forming the maximal outputspan. In the step 2, the sets in the list of outputspan are enumerated and based on the flag in the output_span_wrap_check array, the sets in ds corresponding to the memory element of the span set s1 is found and added to the maxoutputspan and maxoutputspan_wrap sets, respectively. Thus two maximum outspan sets for wrapped and unwrapped spans are constructed and returned by this function.
Figure 8.9 shows the pseudocode for the function Maxspan_input. The input to this function are graph "g" and the list of inputspans. The steps in this function are almost similar to that of the function Maxspan_output. Based on the flags in the input_overlapcheck matrix, the disjoint set ds is constructed for the list of memory elements whose spans overlap with each other. Then the two maximum input spans, namely maxinputspan and maxinputspan_wrap, are constructed based on the flag information in the array input_wrap (same array named input_span_wrap_check created by the function inspan_wrap). The output of this function are the two sets, namely maxinputspan and the maxinputspan_wrap. Finally, the maximal registers are clustered by using the sets, maxoutputspan, maxoutputspan _wrap, maxinputspan, and maxinputspan_wrap.

The pseudocode of the function maximal_register is shown in the Figure 8.10. As given in the definition earlier, this function merges all the maximal spans that are overlapping to form a maximal register. The function maximal_registers takes in the four sets of the maximum of spans. First step involves enumerating the unwrapped maximum spans. Thus the sets in the maxinputspan and maxoutputspan are enumerated and when set s1 in the maxinputspan has an intersection with set s2 in the maxoutputspan, a set forming the intersection of s1 and s2 is added to the Maximal Register list named MaxRegister.

```
/*********************************************
Maxspan - Output
*********************************************/
Input: graph g,list_outspanset;
Output: set maxoutputspan, maxoutputspan_wrap;

MaxSpan_output(graph g,list_outspanset){
  disjoint_set ds;
  list_of_set maxoutputspan, maxoutputspan_wrap;
  For each vertex v in graph g{
    create ds(v1); // adds a set with element v1 to ds
    }
  For each set s1 in list_outputspan {
    For each set s2 in list_outputspan {
      If(output_overlapcheck[s1][s2] =1){
        Union(s1's c-element/latch,s2's c-element/latch) in ds;
        }
      }
    }
  For each set s1 in list_outputspan {
    int i = order of the set containing the s1's c-element/latch;
    If(output_wrap[s1] != 1){maxoutputspan[i].insert(s1)}
    If(output_wrap[s1] = 1){ maxoutputspan_wrap[i].insert(s1)}
    }
  }
```

Figure 8.8: Function: Maxspan Output

```
\*****************************************************
 Maxspan - Input
 *****************************************************\
Input: graph g,list_inspanset;
Output: set maxinputspan, maxinputspan_wrap;

MaxSpan_input(graph g,list_inspanset){
  disjoint_set ds;
  list_of_set maxinputspan, maxinputspan_wrap;
  For each vertex v in graph g{
    create ds(v1); // adds a set with element v1 to ds
    }
  For each set s1 in list_inputspan {
    For each set s2 in list_inputspan {
      If(input_overlapcheck[s1][s2] =1){
        Union(s1's c-element/latch,s2's c-element/latch) in ds;
        }
      }
    }
  For each set s1 in list_inputspan {
     int i = order of the set containing the s1's c-element/latch;
     index = set index of s1's c-element in ds.
     If(input_wrap[index] != 1){
                                  in the disjoint set ds.
       maxinputspan[i].insert(s1)
       }
    If(input_wrap[index] = 1){
      maxinputspan_wrap[i].insert(s1)
      }
    }
  }
```

Figure 8.9: Function:Maxspan Input

```
\*************************************
  Maximal Register
*************************************\
Input: list_of_set maxinputspan,maxinputspan_wrap,
            maxoutputspan,maxoutputspan_wrap;
Output: list_of_set MaxRegister;

MaxRegister(maxinputspan,maxinputspan_wrap,
    maxoutputspan,maxoutputspan_wrap){
  \\ inserting unwrapped maxspans
  For each set s1 in Maxoutputspan {
    For each set s2 in Maxinputspan {
      If(! ( Intersection(s1,s2) not empty){
        set1.insert( Intersection(s1,s2));
        }
      }
    }
    MaxRegister.insert(set1);
  \\ inserting wrapped maxspans
  For each set s1 in Maxoutputspan_wrap {
    For each set s2 in Maxinputspan_wrap {
      If(! ( Intersection(s1,s2) not empty){
        set2.insert( Intersection(s1,s2));
        }
      }
    }
  Maxregister.insert(set2);
  return MaxRegister;
  }
```

Figure 8.10: Function:Maximal Register

The second step involves enumerating the sets in the maxinputspan and maxinputspan_wrap sets and finding the intersecting sets. Then the sets formed with the intersection elements are then added to the Maximal Register list,MaxRegister. Thus the Maximal register list is output by this function. This concludes the register clustering process in the overall flow of the ACLARION. The next section details the Combinational logic unit (CLU) clustering process.

8.5 Combinational logic clustering

After clustering the memory elements to form set of maximal registers, the next step involve clustering all the combinational gates or combinational logic unit (CLU) in the circuit. Outspan and inspan sets constructed during the registering clustering process make CLU clustering easier. The pseudocode for the function implementing the CLU clustering named CLU_clustering is shown in the Figure 8.11 and Figure 8.12. The following steps are involved in this procedure:

- Create the list containing the set of gates in Maximal registers
- Construct the matrix to set flag for the presence of a vertex (c-elements) of the graph in the maximal register list.
- Create the matrix for storing the flag information for presence of vertex in a maximal register
- Create a disjoint set which has the union of the set of maximal registers for each non-memory element vertices
- Create the list of clouds having the clouds of combination gates using the disjoint set created
- Update the disjoint set based on the connectivity of the fanout nodes
- Update the list of clouds using the updated disjoint set

The input to this function is the list of maximal registers named MaxRegister output by the function MaxRegister. The input to the CLU_clustering function are the list of the maximum registers MaxRegister, the list of inscan set and the graph g. The pseudocode for the function CLU_Clustering is shown in Figure 8.11 and Figure 8.12. The first step involves enumerating each set s1 in the list MaxRegister and all the memory elements of set s1. Then the gates in the input spans of these memory elements are stored in the multiset Max_Register_Gates. The indexing of the sets is similar to those in the list MaxRegister. The gates of each inspans of the c-elements are retrieved from the list list_inscanset. The second step involves creating the matrix which stores the flag information on the presence of a vertex in the maximal register set. To construct this matrix named Max_reg_check, the vertices vi of the graph are enumerated along with the sets si in the list MaxRegister. If the set si contains the vertex vi then the Max_reg_check[vi][si] is flagged 1 otherwise it is flagged 0.
The third step involves creating list of maximal registers in which each vertex in the graph which is not a memory element. It is simply to construct list of maximal registers to which each combinational and fanout node belongs to. This is achieved by enumerating the set s2 in the list Max_Register_Gates and the vertices in the graph g. The vertices and the sets are

verified with the matrix Max_reg_check to see whether the set s2 contains the vertex v. If it contains then the set s2 is added to a temporary set max_sets. Once all the sets are enumerated, the max_sets is added to the list of sets Max_Register_set. By the end of enumeration of all the vertices, the Max_Register_set will be having the list of sets corresponding to each non-memory element vertices. The fourth step involves enumerating all the vertices and checking whether each vertex belongs to same set of Maximal registers. For this, each vertex which is not a memory element is enumerated in the graph g. The set in Max_Register_set corresponding to this vertex is compared to the same for all the other vertices. If both the sets are the same then the union operator is applied to these sets corresponding to these two vertices in the disjoint set ds. The fifth step involves creating the set of clouds using the combinational gates. For each vertex which is a combinational gate in the graph g, The vertex is added to the list of clouds named cloudset with index N1 equal to the index of the set corresponding to this vertex in the disjoint set ds.

The sixth step involves analysing the fanout nodes which can be added to this cloudset. For each fanout vertex in the graph g, outedges of that vertex v4 is enumerated. If the target vertex v5 of each outedge is not a memory element, the set of maximal registers for the vertex v4 and v5 are compared in the list Max_Register_set. A flag 1 is set to the variable outedge_check, if all the target vertices have the same set of maximal registers with the vertex v4, otherwise it is set to 0. If the outedge_check is 1, then the union operation is applied to the set corresponding to the vertex v5 and the set corresponding to the vertex v4 in disjoint set ds. By now all the fanout nodes which drive the same register as the clouds in the list of clouds cloudset will be added to the corresponding set in the disjoint set.

```
Input:  graph g, list_of_set MaxRegister, list_inscanset
Output: list_of_clouds;

CLU_Clustering(MaxRegister)
{
 List_of_set Max_Register_Gates;
 Array max_reg_check;
 disjoint_set ds;
 list_of_multiset max_Register_sets, max_sets;
 list_of_sets  list_of_clouds;
 For each set s1 in MaxRegister{
   For each c-element/latch c1 in s1 {
    N = order of c1 in list_inscanset;
    Max_Register_Gates[s1] =  comb gates in  list_inscanset[N];
    }
 }
 For each vertex v in graph g{
   If(v = c-element/latch){
    For each set s1 in MaxRegister{
      If (intersection(v,s1) != empty{
        Max_reg_check[v][s1] = 1;
        }
      else{Max_reg_check[v][s1] = 0;}
      }
    }
  }
 For each vertex v(except memory elements) in graph g{
   For each set s2 in Max_Register_Gates{
    If(Max_reg_check[v][s2] = 1){
      Max_sets.insert(s2);
      }
    }
  Max_Register_set.insert(Max_sets);
  Max_sets.clear();
  }
 For each vertex v in graph g{
   create ds(v1); // adds a set with element v1 to ds
  }
   -continued
```

Figure 8.11: Function:CLU Clustering - part1

```
- continuation from part 1
For each vertex v1 in graph g{
  For each vertex v2 in graph g{
   If( (v1 != c-element or fanout )and
             (v2 != c-element or fanout) and v1 != v2){
     If(Max_Register_set[v1] == Max_Register_set[v2]){
       Union(set of v1, set of v2) in ds;
       }
     }
   }
 }
For each vertex v3 in graph g{
  If(v3 = comb gate){
   N1 = order of the set corresponding to v3 in ds;
   Cloudset[N1].insert( v3);
   }
  }
  Int outedge_check =1;
 For each vertex v4 in graph g{
   If(v4= fanout node){
     For each outedge oe of v4{
       Vertex v5 = target of oe;
       If(v5 != c-element){
         If(Max_Register_set[v5] = Max_Register_set[v4]){
           outedge_check = 1 * outedge_check;
           }
         else{outedge_check = 0;}
          }
          If(outedge_check = 1){
           Union(set of v4, set of v5 in ds);
           }
       }
     }
   }
 For each vertex v4 in graph g{
   If(v4 = fanout){
     N1 = order of the set corresponding to v4 in ds;
     Cloudset[N1].insert( v4);
     }
  }
}

```

Figure 8.12: Function:CLU Clustering:part 2

The final step involves updating the cloudset using the updated disjoint set ds. For this, each fanout node v4 in the graph is enumerated and the index of the set corresponding to the enumer-

ated vertex v4 is set to N1. Then the vertex v4 is added to the cloudset to the set corresponding to the index N1. The resulting cloudset has the set of clouds which consists of all the combinational gates and some of the fanout nodes which drive the same maximal registers in them.

Once the CLUs and some of the fanout nodes are clustered into clouds, the only nodes left to be clustered are the leftout fanout nodes. The next section discusses the clustering process involving these nodes to create the final complete high level extraction of the circuit.

8.6 Fanout clustering

Once the CLU and registers are clustered the fanout nodes in the circuit will be left out, which should be grouped in a way that it streamlines the whole structural view. There are two types of fanout clustering possible, namely uniform and non-uniform. A uniform fanout cluster/cloud is the set of fanout nodes fed by a register/CLU node in such a way that each fanout node feeds exactly the same set of CLUs. A non-uniform fanout cluster/cloud is the set of fanout nodes fed by a register/CLU node and atleast a pair of fanout nodes feed different sets of CLUs.

8.6.1 Algorithm

The heuristics involved in the fanout clustering process are detailed in this subsection. At this stage as mentioned earlier, all the clustered clouds of CLU and the registers are available to construct the high level view with only the fanout node clustering begin left pending. It should be noted that some of the fanout nodes were already added to the CLU cloud which drive the same registers. The fanout nodes not included are the nodes which do not drive the same clouds and register. The steps involved in the fanout clustering process are:

- Find the maximum registers driving the each cloud
- Find the clouds driving each fanout node
- Find the fanout nodes driving the clouds
- Construct the disjoint set to enumerate and cluster the fanout based on their connectivity with the clouds and the maximum registers
- Update the list of clouds cloudset based on the clustering information in disjoint set to form the new cloudset update_cloudset.

The input to the Fanout_clustering function are the list of clouds generated by the CLU_Clustering function, list of the maximum registers MaxRegister, the list of outscan set and the graph "g". The pseudocode for the function Fanout_Clustering is shown in Figure 8.13 and Figure 8.14.

The first step involves finding all the maximal registers driving the clouds in the list of clouds cloudset. Each set s1 in the cloudset is enumerated for the presence of single element set with the fanout node as its element. This is because after the CLU clustering process only the fanout nodes that do not drive the same maximal registers as the clouds in the cloud set are left out and are added as a separate cloud with only that node as the component of the cloud. For each of these set s1, each c-element c1 in the graph "g", each gate g1 in the set corresponding to the c-element c1 in the list of outspans list_outputspan is enumerated. If the gate g1 equals the fanout element s1, then each gate g2 in each maximal register set mr1 is enumerated. If the gate g2 equals the c-element c1, then the order N2 of the set s1 in the cloudset is calculated and the set mr1 is inserted to the list Maxdrivefo at the index N2.

The next step involves finding all the clouds driving the fanout node. For this, each cloudset s2 is enumerated in the list of clouds cloudset. For each outedge of the gate in the set s2, if the target equals the fanout element of s1, then the set s2 is added to the temporary set clouddriveset. After enumerating all the sets s2 in the cloudset, the clouddriveset set is added to the set clouddrivefo which holds the sets of clouds driving one particular fanout node.

All the fanout nodes driving each cloud is constructed in the next step. For each set s2 in the list of clouds cloudset, all the inedges of the gates of s2 is enumerated. If the source of the inedges is the same as the fanout element in s1, then the set s2 is added to the temporary set fodrivingset. After enumerating all the sets s2 in the cloudset, the fodrivingset is added to the fodrivingcloud list, which stores the list of sets having all the fanout nodes corresponding to one cloud.

```
\*****************************************
 Pseudocode: Fanout_Clustering
\*****************************************\
 Input: list_of_clouds,Max_Registers,graph g,list_outputspan
 Output:list_of_clouds updated_cloudset
 FAnout_clustering(graph g,list_outputspan, Max_Registers,cloudset){
   For each cloud set s1 in list_of_clouds{
    If(s1 = single fanout element set){
      For each c-element/latch cl1 in graph g{
        For each gate g1 in list_outputspan[cl1]{
          If(g1 =  element in s1){
            For each max register mr1 in Maximum_Registers{
              For each gate g2 in mr1{
                N2 = order of s1 in cloudset;
                If(g2 = cl1){Maxdrivefo[N2].insert(mr1)}
                }
              }
            }
          }
        }
      }
    }
  Set clouddrivingfo,clouddrivingset;
  For each cloudset s2 in list_of_clouds{
    If(s1 != s2){
      For each gate g4 in s2{
        For each out_edge oe in g4{
          If(target of oe = element of s1){
            Clouddriveset.insert(s2);
            }
          }
        }
      }
    }
  Cloudrivingfo[s1] = clouddriveset;
  Clouddriveset.clear();
  Set fodrivingcloud,fodrivingset;
  For each cloudset s2 in list_of_clouds{
    If(s1 != s2){
      For each gate g4 in s2{
        For each in_edge ie in g4{
          If(source of ie = element of s1){
            fodrivingset.insert(s2);
            }
          }
        }
      }
    }
  -continued
```

Figure 8.13: Function Fanout Clustering - part 1

```
-continuation of part1
fodrivingcloud[s1] = fodrivingset;
fodrivingset.clear();
}
Int MDCheck,focloud_check;
  For each cloud set s1 in list_of_clouds{
   If(s1 = single fanout element set){
     For each cloud set s2 in list_of_clouds{
       If(s2 = single fanout element set and (s1 != s2){
         if( Maxrdrivefo[s1] == Maxrdrivefo[s2] ||
          MDcheck =1;
          }
         else{MDcheck=0;}
         if(fodrivingcloud[s1]= fodrivingcloud[s2]){
           focloud_check=1;
           }
         else{focloud_check=0;}
         if(MDcheck * focloudcheck = 1){
           union(set of s1's element, set of s2's element) in ds;
         }
       }
     }
   }
 }
}
List_of_clouds updated_cloudset;
For each vertex v in graph g{
 If(v!=latch/c-element){
   N4 = order of the set that v belongs to in ds;
   updated_cloudset[N4].insert(v);
   }
}

return updated_cloudset;
}
```

Figure 8.14: Function Fanout Clustering - part 2

Once all the sets s1 in the cloudset has been enumerated the list Maxdrivefo will have the list of all the maximal registers driving the fanout nodes, clouddrivefo will have the sets of clouds driving each fanout node and the list fodrivingcloud has the sets fanouts driving all the clouds.

With these three lists, the next step of updating the disjoint set based on the connectivity of the fanout node with clouds and maximal registers is carried out. Two flag variables, namely

MDcheck and focloud_check, are used in this step. For each set s1 in the list of clouds, s1 is compared with all the other sets in the cloudset for checking the common maximal driving registers driving them. Also, each set s1 is compared with all other sets for checking the common clouds driving them. In former case, MDcheck flag is set to 1 and in the later case focloud_check is set to 1. Otherwise, both the flags are assigned the value 0. If both the flags are 1, then the union operation is applied to the set corresponding to the fanout node element of s1 and the set corresponding to the fanout node element of s2 are in the ds. This streamlines the excluded fanout nodes to be added to the corresponding clouds to which they belong if they drive the same clouds and are driven by the same maximal registers.
The final step involves updating the cloudset. For this all the vertices that are not memory elements are enumerated in graph g and the order N4 of the set each vertex belongs to in the disjoint set ds is calculated. Then the vertex is added to the updated cloudset update_cloudset with index N4. After all the vertices are enumerated, the resulting list of clouds update_cloudset will have the list of clouds containing the fanout nodes and the combinational gates. This list is returned by the fanout_clustering function.

Using the list updated_cloudset, list MaxRegisters and the connectivity information from the graph g, the overall high level structural view of the circuit can be constructed. The resulting graph will be several order of magnitude smaller than the original graph. The experimental results of this method applied to several asynchronous benchmark circuits are analyzed in the next section.

8.7 Experiment

This circuit extraction method for asynchronous circuit was implemented in C++ as an extractor tool. Several asynchronous benchmarks were used for the experimental analysis. The results obtained for the benchmark "master-read" is shown in Figure 8.15. Figure 8.15 shows the color-coded partition of the fanout, CLU and c-elements. To distinguish the clustering clearly, the Figure 8.16 shows the clusters of CLU with number (with nodes of same cluster having same number), fanout nodes named as fanout and the c-element left with their alphabetical name. The high-level extracted structural view of the benchmark is shown in Figure 8.17. Table 8.1 shows the resulting high-level structural representation consisting of 3 combinational clouds, 6 fanout nodes and 2 maximal registers. The column named "cele" gives the number of C-elements present in the benchmark circuit. The column named "#gates" gives the number of gates for the benchmark. The column named "# clouds" gives the number of clouds formed in the extracted view. The maximal register column has two subcolumns named "size" and "#", which give the number of C-elements in each register and the number of registers respectively.

The last column gives the time taken for execution of the implemented extractor tool.

The methodology is capable of executing over the industrial tool as the clarion was demonstrated on the industrial synchronous circuits. The profiling of the sourcecode revealed the function span being called extensively due to the construction of inspans and outspans. Further optimization on the span function usage will reduce the execution time of the tool.

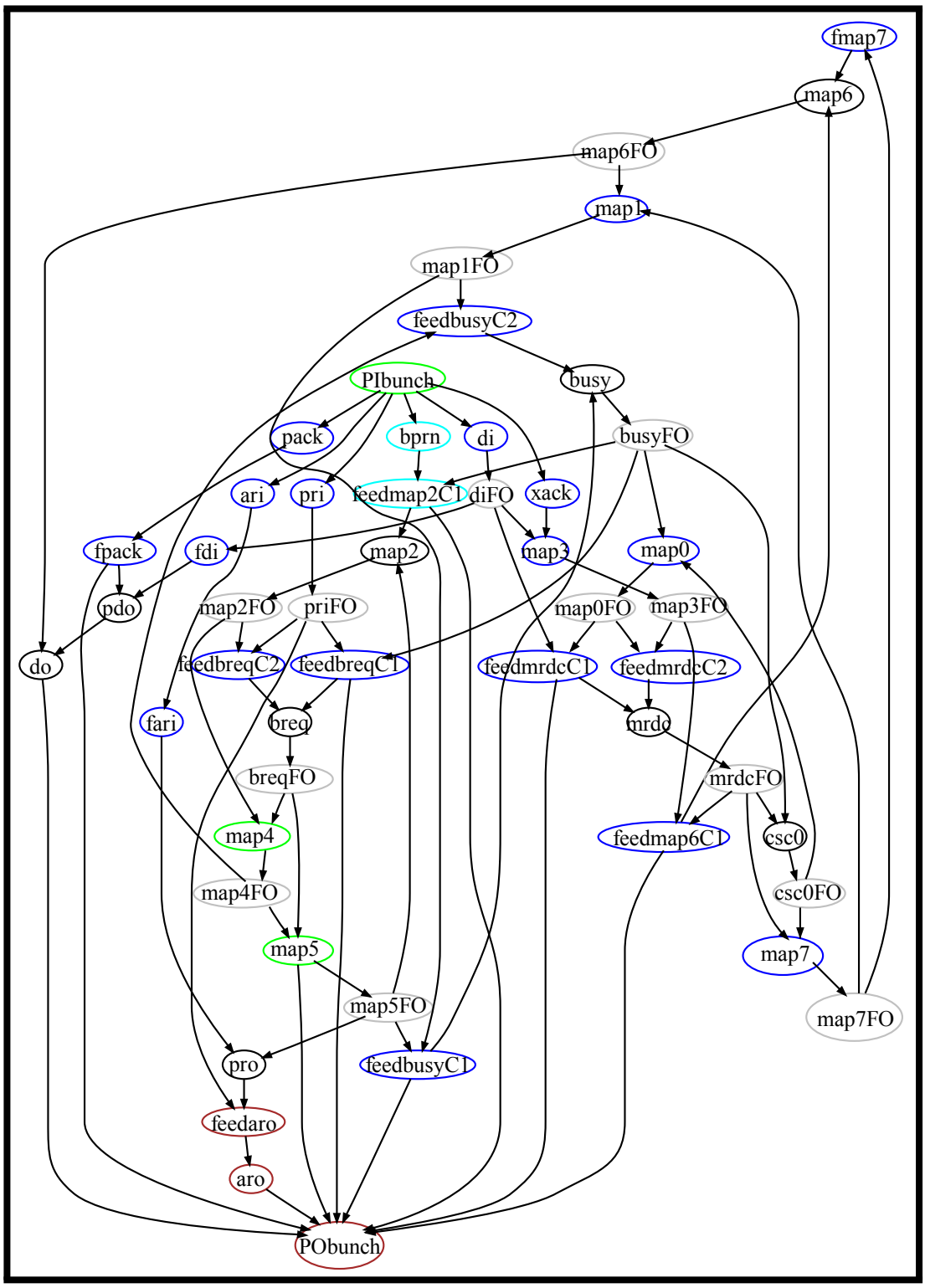


Figure 8.15: Master-read Benchmark

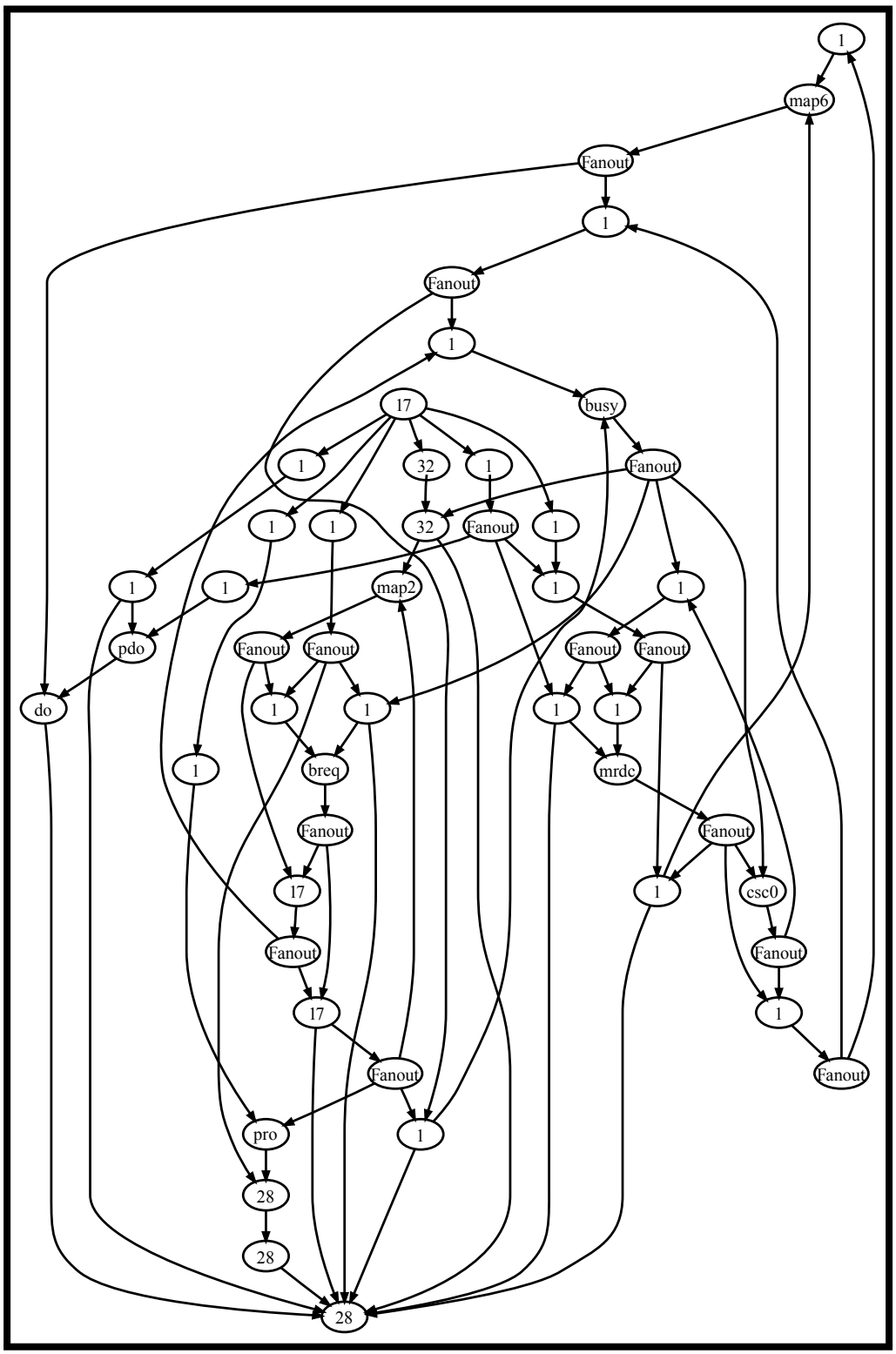


Figure 8.16: master-read benchmark - numbered clouds

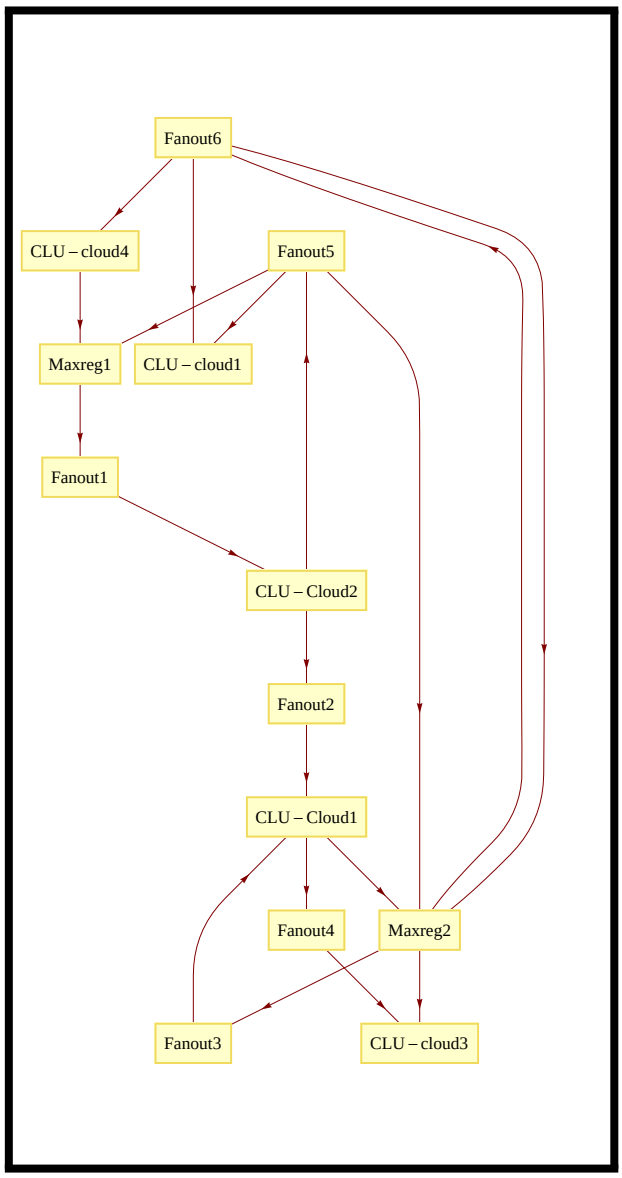


Figure 8.17: Extracted High Level View - master-read benchmark

8.8 Evaluation

To evaluate the proposed extraction methodology different asynchronous test methods proposed in this thesis namely ABALLAST, AGLOB1, AGLOB2 and ASCP were excercised with the high level netlist extracted by ACLARION. Evaluation metrics namely graph size in vertices and edges, fault coverage, and number of patterns generated are compared and analyzed for these methods applied over the original circuit and the high level extracted view of the circuit.

Graph size reduction

Table 8.1: Circuit Extraction Results

Benchmarks	Cele	#gates	#clouds	#fanout	MaxReg	MaxReg
					#	size
chu150	2	8	3	5	1	2
converta	3	6	3	4	1	3
dff	2	6	8	5	2	1,1
ebergen	3	11	10	7	2	2,1
half	2	1	4	1	2	1,1
hazard	2	6	5	6	2	1,1
master-read	9	16	10	14	2	8,1
mmu	6	20	3	13	2	2,4
mp-for-pkt	3	8	4	5	1	3
mr1	9	18	12	16	2	7,2
nak-pa	4	12	4	5	1	4
pe-rcv-ifc	6	33	3	13	1	6
pe-send-ifc	6	23	6	11	2	2,4
ram-rd-sbuf	4	12	3	8	1	4
rcv-setup	1	6	3	4	1	1
rpdft	1	11	3	6	1	1
sbuf-ram-write	6	14	6	11	1	6
sbuf-snd-ctl	5	12	10	8	2	2,3
sbuf-snd-pkt2	5	18	12	11	2	3,2
seq4	7	9	3	10	1	7
trimos-send	8	18	9	14	2	1,7
vbe5b	2	6	3	5	1	2
vbe5c	3	2	5	3	2	1,2

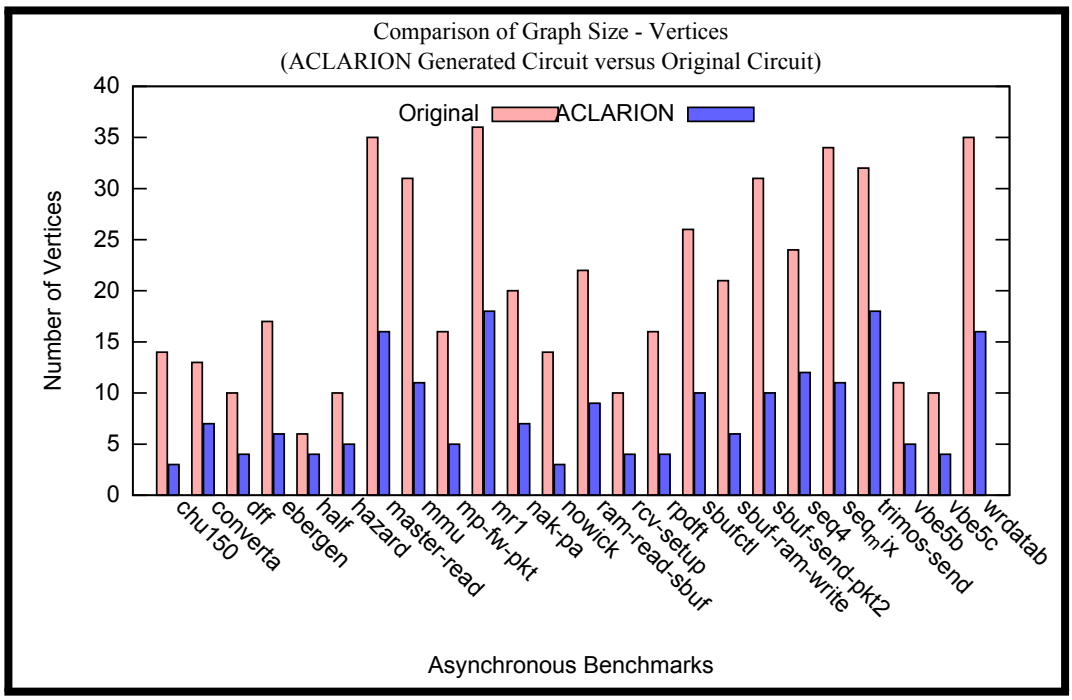


Figure 8.18: Graph Size Comparision - Vertices

The main motivation of generating the high level extraction is to reduce the problem size before applying further test algorithms applied. The extracted view generated by ACLARION does indeed reduce the graph size as it is clustering several combinational gates and registers. Figure 8.18 gives a comparison of the number of vertices of the original graph with the graph of the extracted view. The original graph/netlist is termed as "original graph" and the extracted view graph/netlist is called as the "ACLARION graph". Almost all the benchmarks had greater than 50 % reduction in the number of vertices. Bigger benchmarks such as wrdatab, mmu and master-read had considerably greater reduction. Figure 8.19 shows a comparison of the sizes of the edges for the original and ACLARION graph. In relation to the number of vertices, the number of edges is even lower. This is due to the fact that the interconnections between the gates and memory elements are reduced when the latter were clustered. For the benchmark wrdatab, the number of edges was reduced by 50%.

The interesting point to probe is how well the fault coverage and number of patterns obtained by the test methods are retained when these methods process the extracted view of the same netlist. To do this analysis all the benchmarks were run in two different experiments for the all the test methods. First experiment involved running the test methods over the original benchmarks and the second experiment involved running the test methods over the extracted benchmarks. After running these two experiments, the fault coverage and the number of patterns generated for both the experiments are compared for each test method. It should be ACLARION netlist type was used by ABALLAST method, it is already running on the extracted view, so the analysis

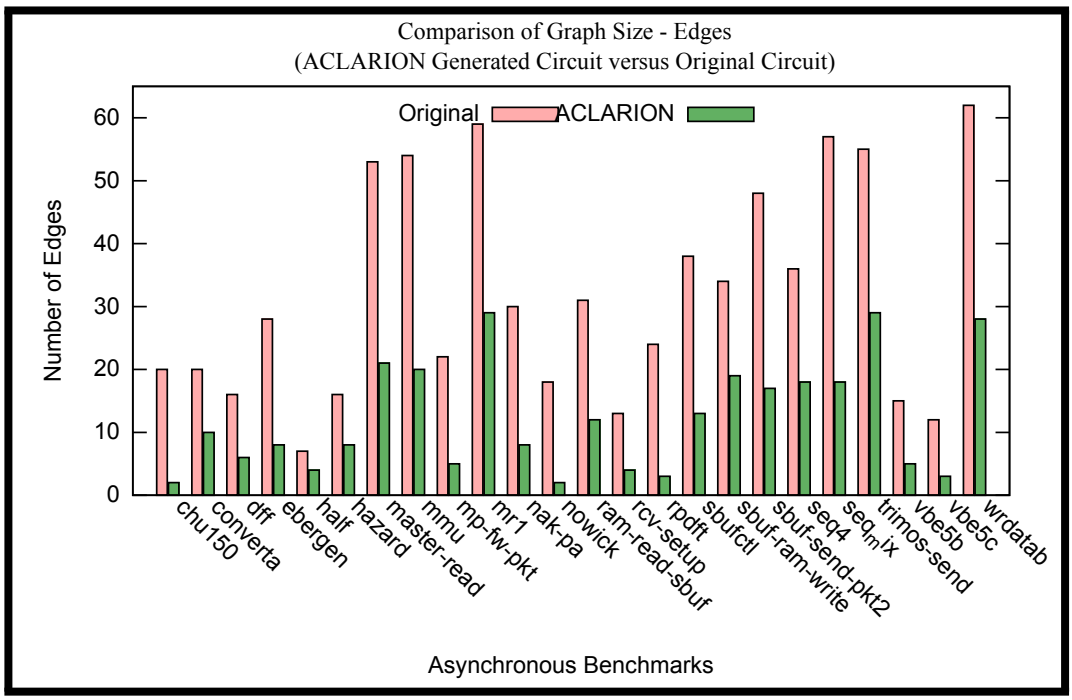


Figure 8.19: Graph Size Comparision - Edges

does not include ABALLAST method.

The following subsection discusses the impact on fault coverage and the impact on the number of vertices.

8.8.1 Impact on Fault Coverage

Now the impact of fault coverage on the test methods AGLOB1, AGLOB2 and ASCP are discussed.

AGLOB1 Test Method

The graph in Figure 8.20 gives the comparison of fault coverage for the AGLOB1 method operated over original graph and the ACLARION graph. Along with this comparison the full scan method is also included as the third data. For the smaller benchmarks, the fault coverage based on original graph and the extracted view is almost same as there is not be much difference between the two graphs. This is very well exhibited by the benchmarks chu150, converta, ebergen, vbe5b and vbe5c as they have same fault coverage for both the graphs. But a more important observation is on the benchmarks masterread, mmu,mr1, sbuf-send-ctl and wrdatab. For all the larger benchmarks, the fault coverage was improved with the extracted netlist view

but at the cost of increased scan element .

AGLOB2 Test Method

Next the fault coverage comparison for AGLOB2 method is analyzed. Figure 8.21 gives the comparison of the fault coverage obtained using original and ACLARION graph. In contrast to the AGLOB1 method, this method improved the fault coverage with reaching the full scan equivalent. For example, the benchmarks mmu and mr1 had higher fault coverage compared to the original graph and almost same as full scan. But for the benchmark mp-forward-pkt2, the fault coverage was improved without reaching the full scan memory scanning.

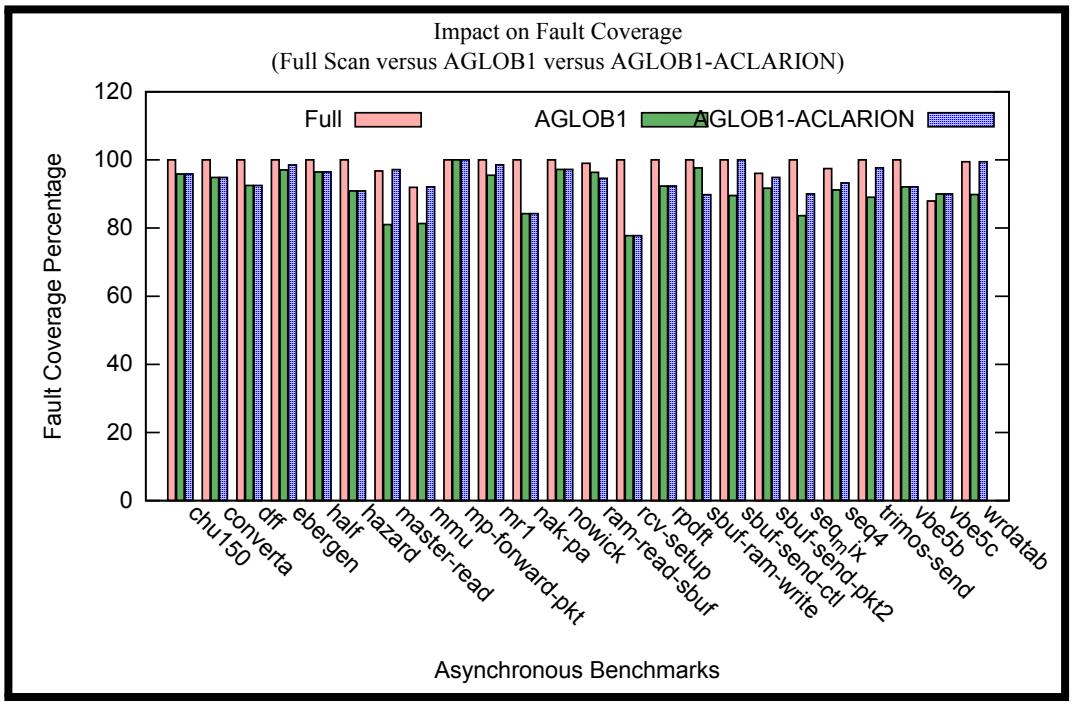


Figure 8.20: Impact on Fault Coverage - Full Scan versus AGLOB1 versus AGLOB1-ACLARION

ASCP Test method

In Figure 8.22, impact of the extraction on fault coverage over the ASCP method is shown. In this method, most of the benchmark exhibited improvement in fault coverage for the ACLARION graph-based experiments, with the exception on the benchmarks mr1, seq_mix, and trimos-send. The overall impact on fault coverage of all these methods will be shown at the end of the next subsection.

8.8.2 Impact on Number of patterns

In this subsection the impact of extraction over the number of test patterns generated is analyzed. For the three methods analyzed, the red bar in the graph gives the number of patterns

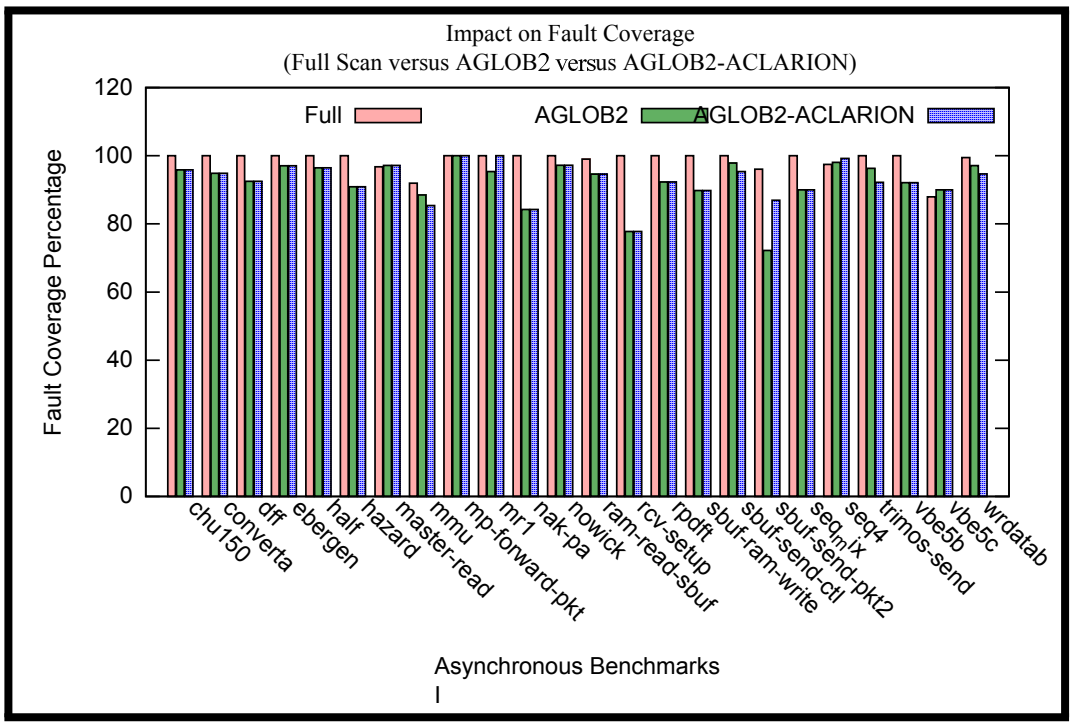


Figure 8.21: Impact on Fault Coverage - Full Scan versus AGLOB2 versus AGLOB2-ACLARION

generated for full scan method, the green bar gives the patterns for the original graph and the blue bar gives the patterns for the ACLARION graph.

AGLOB1 Test Method

Figure 8.23 shows the comparison for the number of test patterns generated. Clearly the extracted view had higher number for most benchmarks patterns as it selected more C-elements compared to the original graph. For master-read, mmu and trimos-send the number of patterns increased, whereas for the benchmarks sbuf-ram-write and sbuf-send-ctl the number of patterns reduced.

AGLOB2 Test Method

For the AGLOB2 method, the comparison is shown in the Figure 8.24. For this method, extracted netlist view reduced the number test patterns generated for most benchmarks. Benchmarks mmu, master-read and trimos-send had higher reduction in number of patterns.

For AGLOB1 and AGLOB2, the benchmarks, "master-read", "mmu", and "trimos-send" had their number of patterns increased. The reason for increase in the pattern number is of two-fold. First one is that, they had 9, 6, and 8 C-elements, respectively and the way the subset of these C-elements selected for partial-scan impacts the number of patterns being generated. And, when ACLARION extracted the clouds, several C-elements constituted a cloud, which resulted in increase of the number of partial-scan C-elements. With this increase, test pattern generation involved more scan test patterns to be added to test these scannable C-elements.

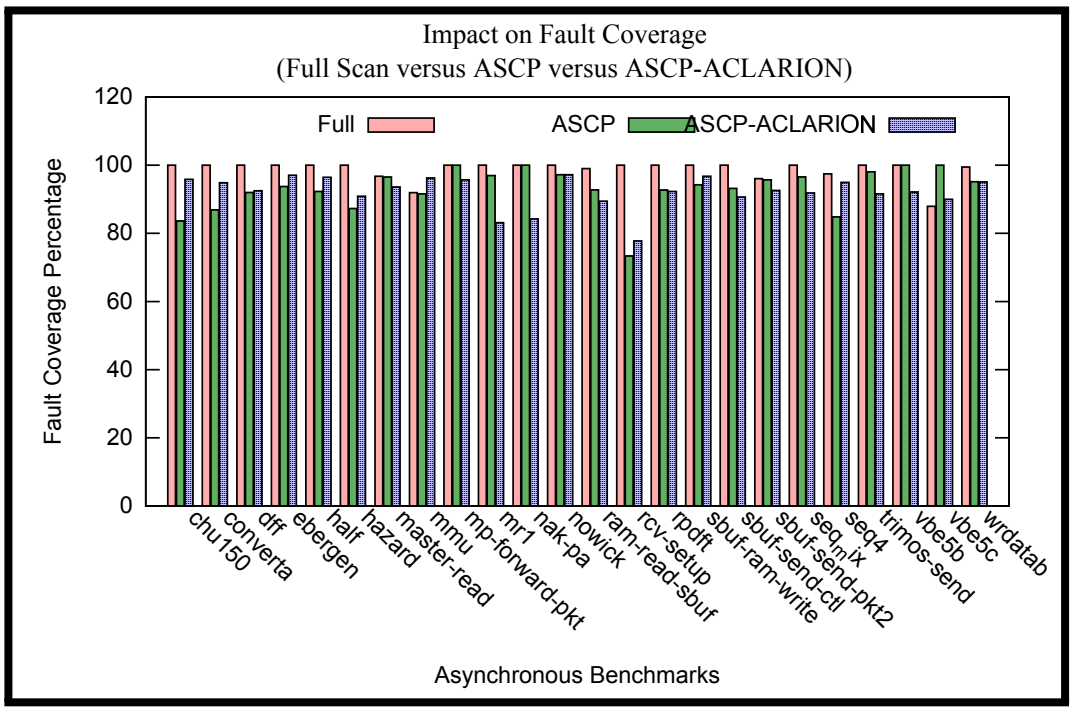


Figure 8.22: Impact on Fault Coverage- Full Scan versus ASCP versus ASCP-ACLARION

Second reason is that, when some of the C-elements, which are not scanned are located at the higher depth of the circuit, reaching those nodes required more test patterns. Vice versa, for the benchmarks "sbuframwrite" and "sbufsendctrl", the C-elements not chosen for scan were the ones, which were closer to the input/output nodes, compared to the original partial-scan circuit generated without ACLARION. Hence, the number of patterns for these circuits reduced.

ASCP Test Method

Finally the impact on number of test patterns for the ASCP method is shown in Figure 8.25. For this method also, the number of test patterns were reduced for most benchmarks. This is because, the ASCP method had lesser information on the location of the C-elements and the scan-selection was guided only by the efficent selection of lower number of scan C-elements. So, even when the number of scannable C-elements were reduced, the selected C-elements were not guaranteed to be at a lower depth of the circuit. But, interestingly for the benchmark seq_mix, there was a steep rise in the number of test patterns. For this case, the location of the subset of C-elements selected for partial-scan, resulted to be at the higher depth of the circuit. There were totally 6 C-elements and when the full-scan method chose to scan all these C-elements, the scan-chain formed by the full-scan reduced the depth of this path and hence the number of patterns were lesser. And, for the ACLARION generated circuit, the partial-scan path was longer, which resulted in the higher number of test patterns.

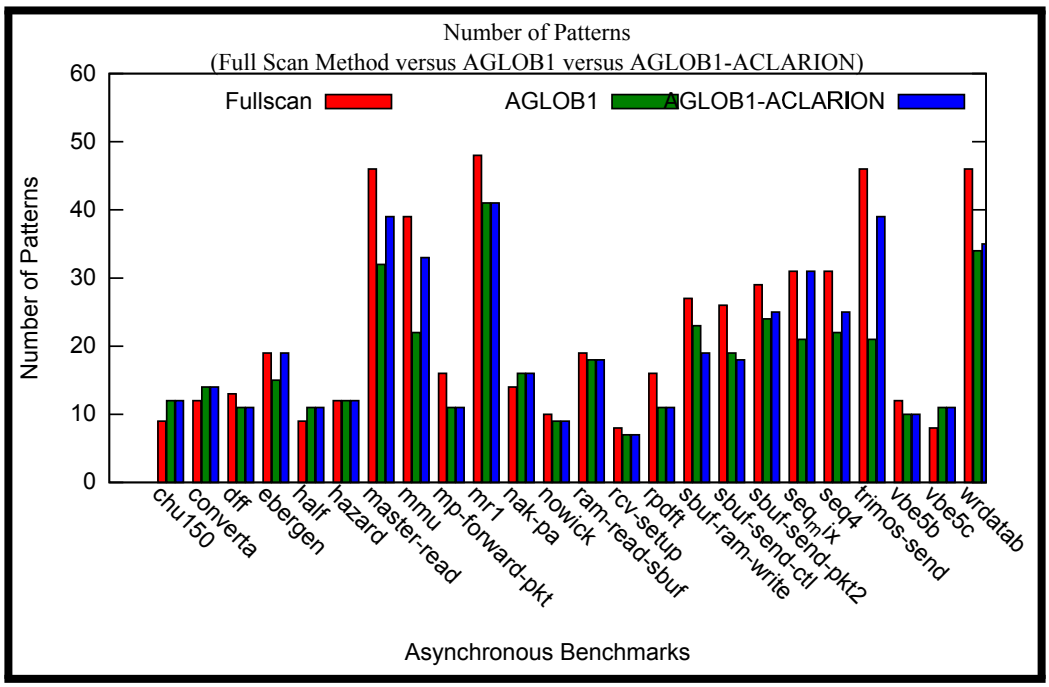


Figure 8.23: Impact on Number of Patterns - Full Scan versus AGLOB1 versus AGLOB1-ACLARION

With and without ACLARION

As analyzed in the previous subsections, the extracted netlist gave improved fault coverage for most benchmarks, but with some exceptions. To see the overall effect of the extracted netlist on the fault coverage obtained with the different test methods, the 3D plot of the fault coverage of the AGLOB1, AGLOB2 and ASCP methods for the original and extracted view is shown in the Figure 8.26. The left side of the plot on Yaxis (tics 1,2, and 3 are named AGLOB1,AGLOB2 AND ASCP) clearly shows lower fault coverage compared to the right hand side(tics 4, 5 and 6 are named AGLOB1(ACL meaning ACLARION), AGLOB2(ACL) and ASCP(ASCP). The blue regions on the left shows the lower fault coverage and the peaks on the right hand side shows the higher fault coverage for the extracted netlist view. The middle blue region is due to the benchmark rcvsetup which does not have any C-elements to be chosen by the partial scan methods. Finally, a 3D plot showing the fault coverage for the methods namely Full scan, ABALLAST, AGLOB1, AGLOB2,ASCP, AGLOB1-ACLARION, AGLOB2-ACLARION and ASCP-ACLARION are plotted in Figure 8.27. To show the peaks the graph is plotted as monochrome. It is evident from the graph that the ABALLAST, and all the methods with ACLARION graph based test generation had higher fault coverage. This can be seen from the peaks on the left side of the Y axis(initial one being the fullscan) and the peaks on the extreme right hand side. There are lower number of spikes in the middle which attributes to the methods applied over the original graph.

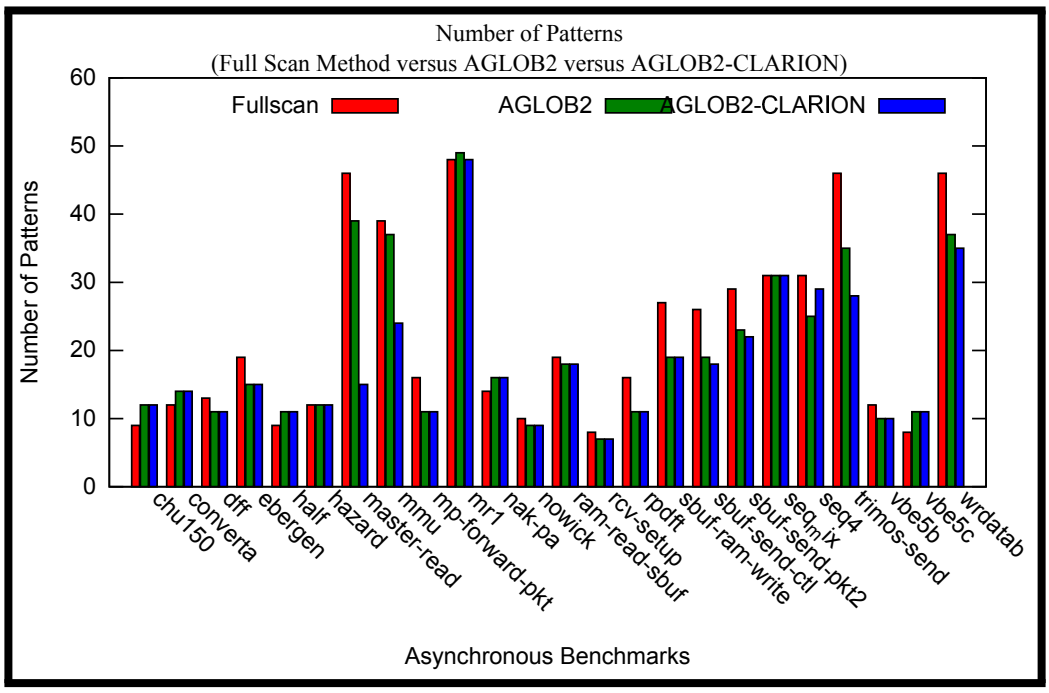


Figure 8.24: Impact on Number of Patterns - Full Scan versus AGLOB2 versus AGLOB2-ACLARION

Complexity // The complexity of the ACLARION method is mainly dominated by the combinational clustering and the register clustering steps. The merging of the memory elements has a complexity of O (n^3). The complexity of the combinational clustering is O (nm), where n and m are number of vertices and, edges respectively.

8.9 Conclusion

A high-level circuit extraction method for asynchronous circuits was proposed. Basic functions required for the implementation of the ACLARION extraction method and the overview of the methodology were described. The proposed heuristics for the Register clustering process was introduced next. The heuristics for the combination logic unit (CLU) clustering was discussed futher. Fanout clustering heuristics were introduced next. Experimental results were analyzed for various asynchronous benchmarks with one working examples demonstrated.

The circuit extraciton method developed can be applied to any test generation system for asynchronous circuits. The test generation time can be drastically reduced by operating on the high level graph generated by this method.

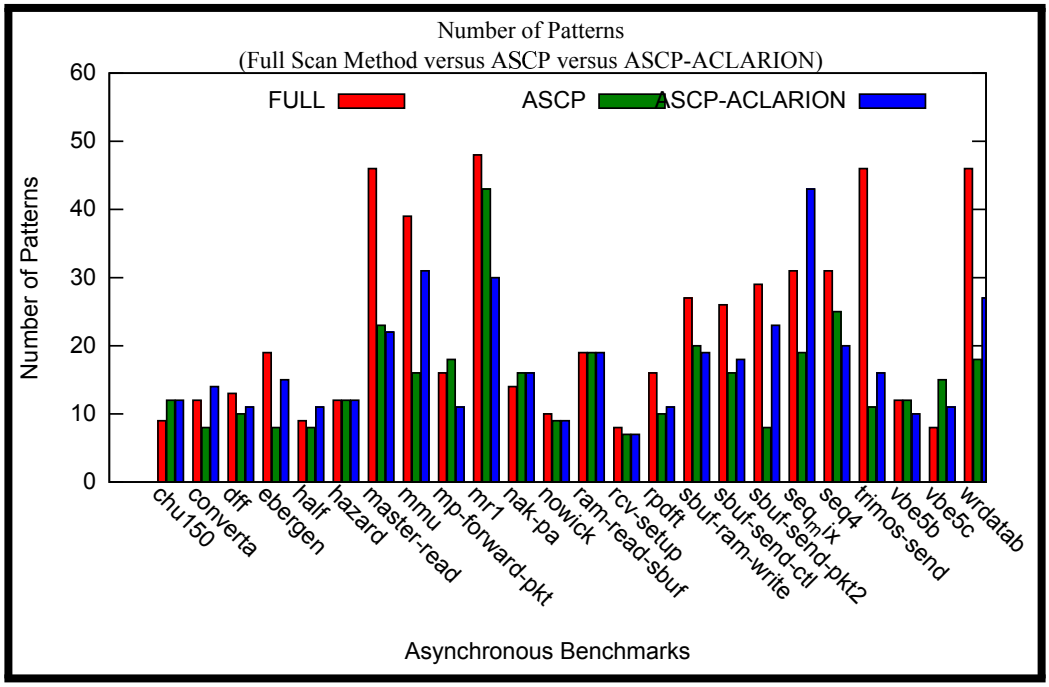


Figure 8.25: Impact on Number of Patterns - Full Scan versus ASCP versus ASCP-ACLARION

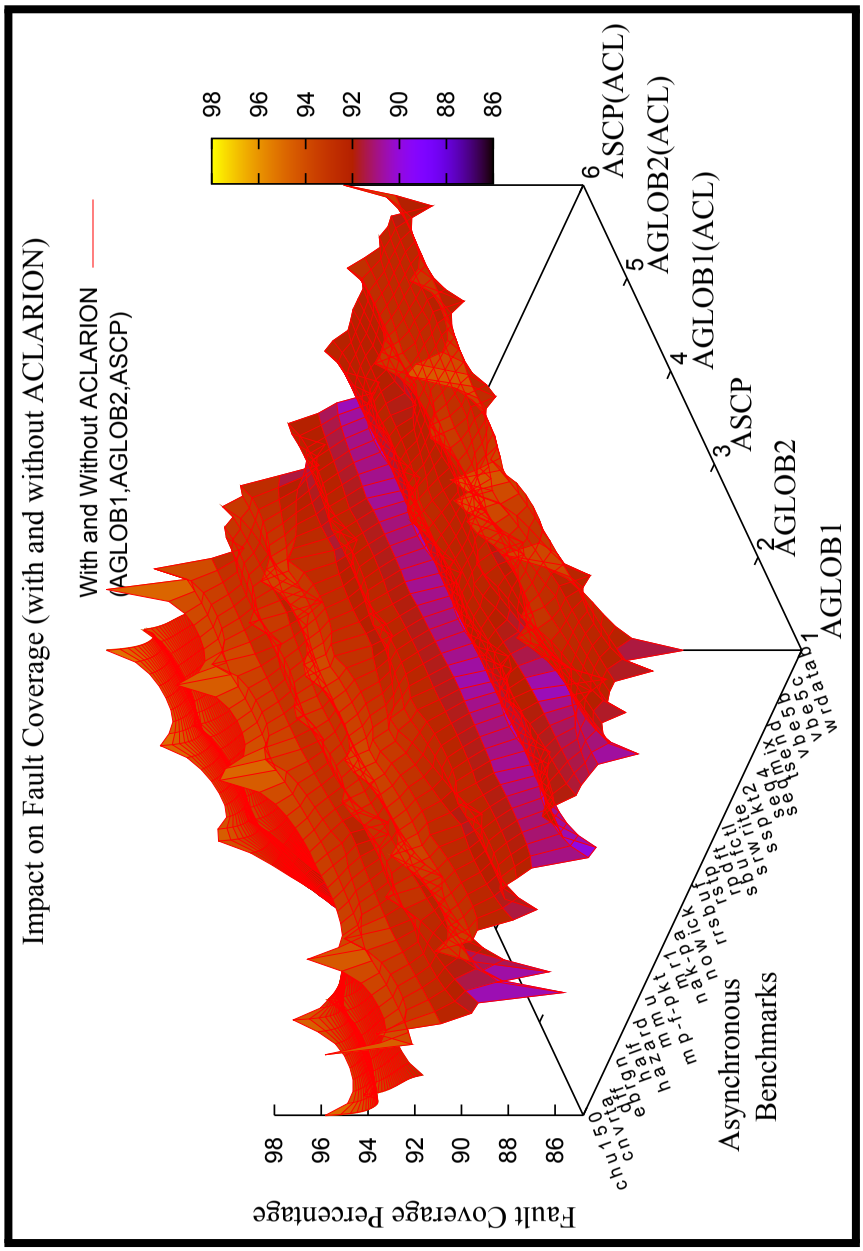


Figure 8.26: Impact on Fault Coverage for three methods- with and without ACLARION

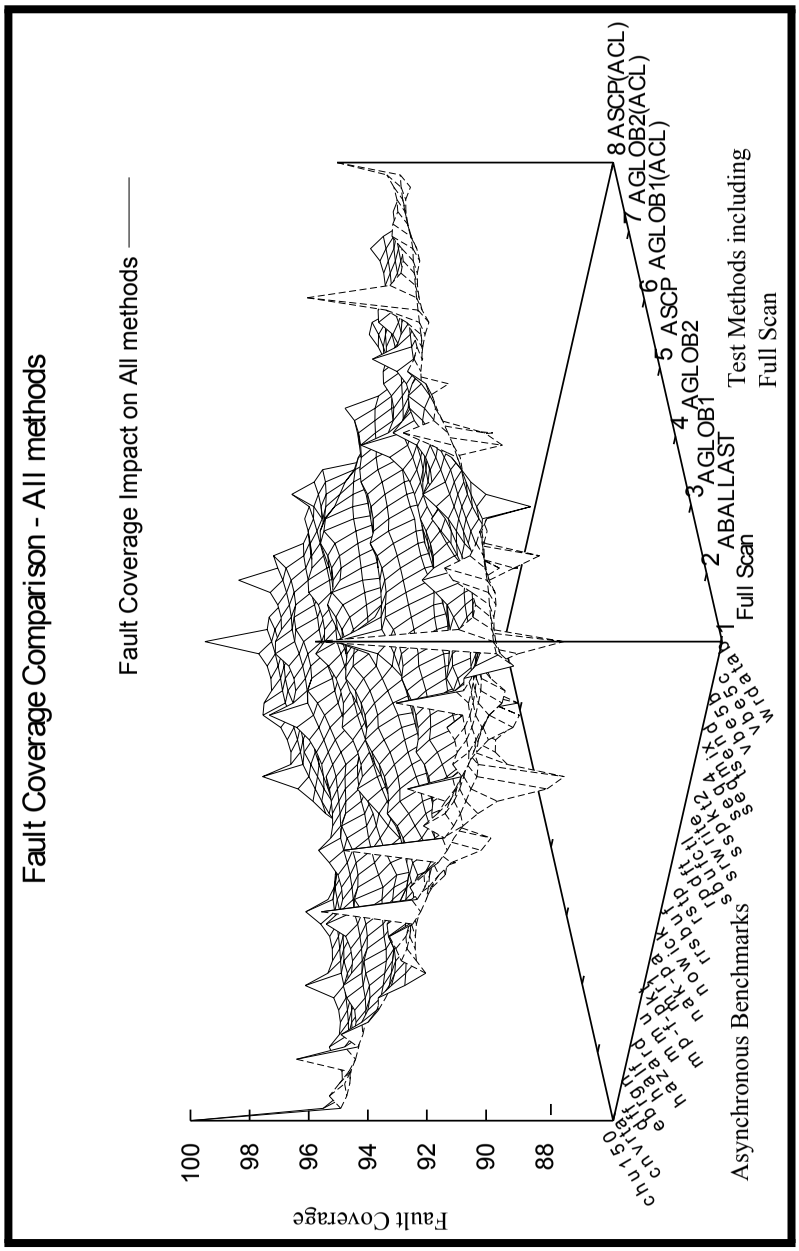


Figure 8.27: Fault Coverage Comparison - All methods

Chapter 9

ATRANTE - Transistor Level Test Generation for Asynchronous Circuits

9.1 Introduction

With the increasing number of transistors per chip, the number of faults due to physical defect is increasing. This will lead to a higher probability that devices will malfunction. These physical defects are mainly due to photo lithography errors, electromigration, corrosion, and oxide effects to name a few. These defects cause adverse effects on circuit behaviour. Hence several test methods and models have been created to detect the faults in the circuits [VCHS09, RDB08, IRR+01, LM05, FMHG05a, MAF90]. Gate-level stuck-at fault model is the most widely used model to test the circuits. The high-level abstraction of the circuits at the gate-level is considered to test the designs in the stuck-at-fault based test generation.

Although these models usually give significant fault coverage of the design, not all the physical defects can be mapped accurately to these models [FMHG05b]. Hence transistor level fault models are considered which increase the accuracy of the faults due to the physical defects at deeper level [FS88b] [Mal87]. In other words, the test effectiveness of the test patterns can be improved by adding test patterns generated using transistor-level fault models compared with those generated using stuck-at fault model. Again using detailed transistor-level models will increase the simulation time and the size of the test patterns. So there should be a model that provides high fault coverage with low test time. Several test generation methods for asynchronous circuits are proposed which is based on State Transition graph (STG) [Roi97][EOL02]. But these methods deal with gate-level stuck-at-fault model-based test generations. But test generation methods at transistor level for asynchronous circuits are not found in literature to the best of the author's knowledge. [ES95] deals with the switch level test generation problem for

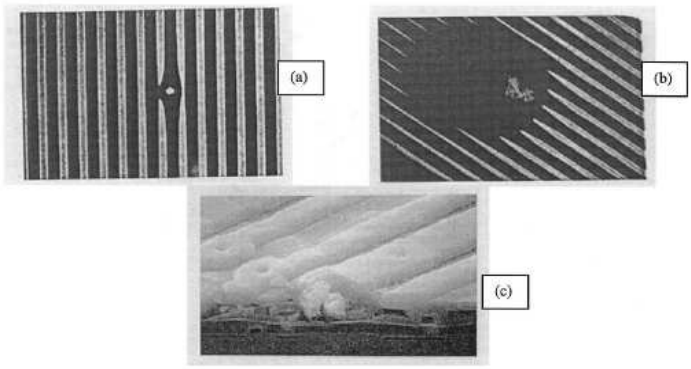


Figure 9.1: Open Defects. a) A foreign particle causing a line to open and a line thinning, b) A contaminating particle causing 7 line opens, c) Defect which caused an open in metal 2 and short in metal 1. [RM00]

asynchronous circuits but only stuck open and stuck on faults are considered.

9.1.1 Motivation: Why transistor level testing?

As the stuck-open and stuck-on faults are transistor-level faults (not a logic node level as in Stuck-at ones) , the behaviour of the circuit is changed from combinational to sequential due to the floating transistors occurring in the circuit due to the stuck-open faults. This issue has already been introduced and dealt with in detail for synchronous circuits [VCHS09, RDB08, IRR+01, LM05, FMHG05a]. It has been shown that even though the conventional stuck-at fault model covers some of the transistors having the stuck-open or short fault, it will not detect all the transistor-level faults. The commonly occuring open defects are shown in the Fig.9.1. It has been shown in [RM00] that several other defects cause the transistor to become open or shorted and these defects occur commonly in the manufacturing process.

To detect these kinds of faults, two different directions were taken traditionaly. First one is called "two pattern sequence" test generation. In this method, for each transistor fault, two test patterns are applied to detect the fault. In the second way, additional hardware or different CMOS logic design style was used to address this fault. The issue of sequence-based test was explained in detail in [LM02]. An example showing the occurrence of a stuck-open fault and the detection of fault based on the sequences of patterns was clearly shown in [LM02]. It was shown that even though these sequences detect the transistor-level stuck open faults, their order is very important for the detection. Change in the order will make the test invalid. The same issue is encountered with asynchronous circuits with the added complexity of having C-elements in the design. Testing the stuck-at fault for C-elements is a complex task by itself. On further testing these transistor-level fault needs correct order/sequences of test patterns to

test the single transistor fault than a set of stuck-at test patterns. Thus testing the asynchronous circuits at transistor level will increase the testability of the design for stuck open and short faults.

The main motivation behind this work is on developing a test pattern generation method at the transistor level for asynchronous circuit covering transistor stuck open and stuck on faults.

This chapter explores an automatic test pattern generation methodology using a fault model called "Transition fault model", which covers the above-mentioned faults. The main contribution of the chapter is in presenting a novel method of test generation for asynchronous circuits using State-Transition Graph (STG) at transistor level and the fault simulation method for the same. The conventional switch level modelling techniques and the STG based representation of asynchronous circuits are merged to develop this new method.

This chapter is organised as follows: Section 9.2 gives the background information to understand the proposed method; Section 9.3 states the problem illustrated with an example; Section 9.4 describes the test methodology; Section 9.5 presents the experimental results with analysis, followed by an ending remarks in Section 9.6.

9.2 Background

Background on asynchronous circuit and transistor level testing is given as follows.

9.2.1 Asynchronous Circuit Representation

State-Transition Graph (STG) is an interpreted Free Choice Petri Net introduced by [CG86] for representing asynchronous control circuits. The behaviour of the circuit is modelled as a set of transition rules with respect to I/O signals. A state graph is a finite automaton, which is an extended version of STG with all the state encoded with binary values

9.2.1.1 Petrinet

A Petrinet [CKK+97] is a compact model to represent concurrent systems. A Petri net is a quadruple N = {P, T, F, m0}, where P is a finite set of places, T is a finite set of transitions, F is the flow relation, and m0 is the initial marking. A transition is enabled at marking, m1, if all its input places are marked. An enabled transition, t, may fire, producing a new marking, m2, with one less token in each input place and one more token in each output place. A free choice

Petri net (FCPN) where the value changes on input, output or internal signals of the specified circuit are the interpretation of the transitions.

9.2.1.2 Signal Transition Graph (STG)

STG is an interpreted Free Choice Petri Net (FCPN) introduced by [Chu87] for representing asynchronous control circuits. It is a quadruple {T, P, F, m0}, where T is a set of transitions described by a x {+, -}, where a+ represents a 0 to 1 transition on signal a, and a- represents a 1 to 0 transition; P is a set of places which can be used to specify conflict or choice; F represents flow transition relation between transitions and place; m0 is the initial marking. An example of an STG is shown in Figure 9.2

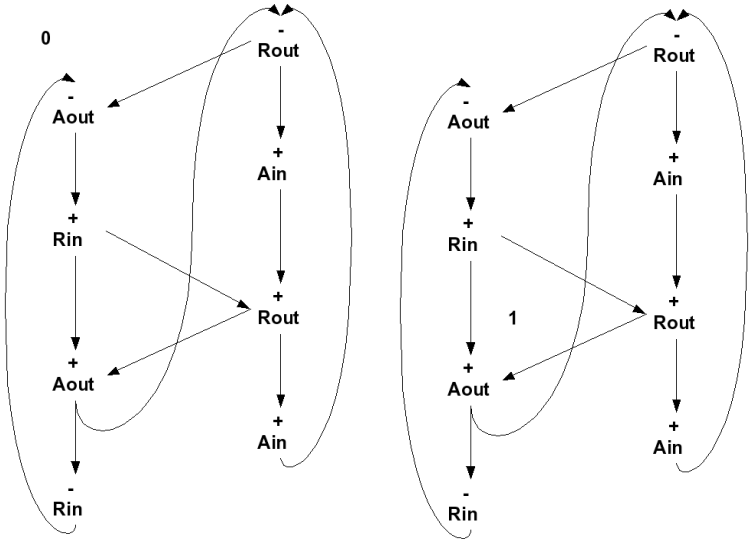


Figure 9.2: Stuck-at-false fault and Stuck-at-true fault

9.2.1.3 State Graph (SG)

A state graph [CKK+97] is a finite automaton given by G = $< A, S, T, \delta, s0 >$, where A is the set of input and non-input (output and internal) signals such that, T is a set of signal transitions, each transition can be represented as $(+ai, j) or (-ai, j)$ for the j-th $0 \rightarrow 1$ or $1 \rightarrow 0$ transition of signal a . $\delta : SxT \rightarrow$ S is a partial function representing the transition function such that if δ(s,t) = s', then signal t is said to be enabled and it takes the system from state s to s'. s0 is the initial state. Each state in the state graph is labelled with a binary vector according to the signal values of the system at that state.

9.3 Problem Statement

The test pattern generation problem is N-P hard. To focus further, the problem chosen in this chapter is to create the test patterns to detect defect level faults. To address this problem, two fault models are used. First, the STG level fault model (stuck-at-true/false) is used to generate test pattern (shown as the top layer in Figure 9.3). Second, the test patterns are fault simulated using defect level fault models (stuck-open/on) (shown as the bottom layer in Figure 9.3).

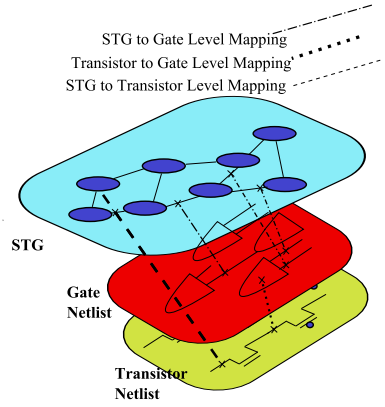


Figure 9.3: Fault Mapping in STG based asynchronous circuit netlist

9.3.1 Motivating Example

An example of transistor level test of C-element is shown in this subsection. A working example of the test pattern generation for a single fault is described further. A static implementation of the C-element is considered for the example (Figure 9.5) [ES95]. The implementation has 12 transistors, 3 i/o and 7 other internal nodes, including Vdd and Gnd. In total 63 faults can be modelled for the c-element, whereas, in the case of stuck-at fault only 6 faults can be modelled corresponding to the 3 I/O pins. To explain the test generation the stuck-open fault on transistor P3 (shown in Figure 9.6). Figure 9.10 gives all the possible transitions in the transistors that can occur in the C-element(in Fig. 9.6). For 8 input combinations ('a,b,c') of C-element, possible transition characteristics of the transistors in the C-element (good circuit) is shown. In Fig. 9.11, the possible transition characteristics of the C-element that is faulty is shown. For both the cases, the transistors that are switched on are denoted by the down-arrows (red). The transistors that are turned off are denoted by a cross (blue). For the faulty C-element, the P3 transistor switched on due to the stuck-on fault is denoted by a down arrow (shown in side the bubble). Thus it should be noted from Figure 9.11, that possible transitions which affects the operation of the P3 transistor are Figure 9.11.(c), Figure 9.11.(d), Figure 9.11.(g), and Figure 9.11.(h).

Table 9.1: MOS Gate Output Table[JA85]

S1(Pull Up Network)	S0(Pull Down Network)	Y(Output)
1	1	0
1	0	1
0	1	0
0	0	m=previous state

Table 9.2: Truth Table for Good(G) and Faulty(F) machine

State	S0		S1		Y	
	G	F	G	F	G	F
000	1	1	0	0	0	0
001	1	1	0	0	0	0
010	1	1	0	0	0	0
011	0	0	0	0	m=1	m=1
100	1	1	0	0	0	0
101	0	0	0	0	m=1	m=1
110	*0*	*1*	*1*	*0*	*1*	*0*
111	0	0	1	1	1	1

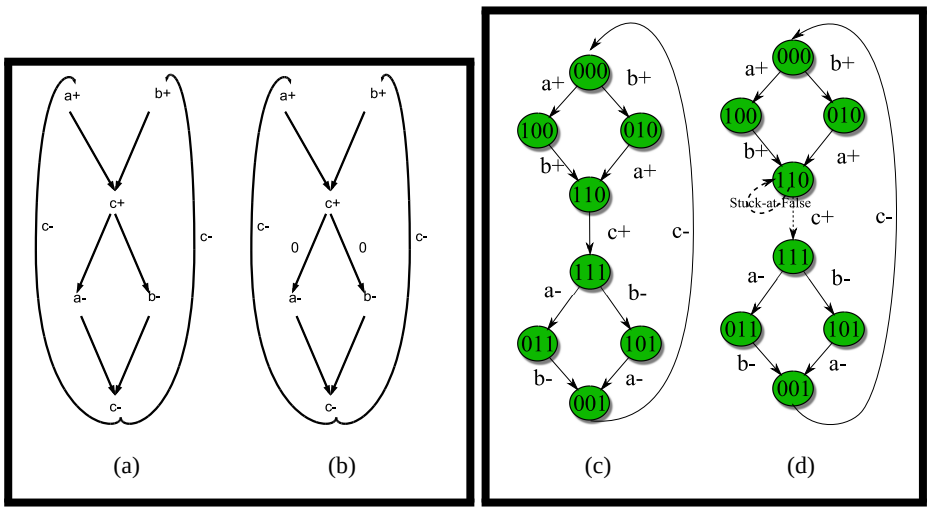


Figure 9.4: SG and STG for faulty circuit with transistor P3 Stuck-on fault shown in (a), (b), (c), (d)respectively.

The faulty circuit is shown in Figure 9.6. The corresponding STG and SG (Stage Graph) of the good circuit are also shown in Figure 9.4. The MOS gate table introduced in [JA85] is shown in Table 9.1. This table derives the logic value of the output node based on the pull-up and pull-down transistor network logic value. The MOS table equivalent to the C-element in Figure 9.6 is shown in Table 9.2. In this table, the pattern for abc, "110" differs in faulty and good circuit. The output stays at ’0’ for a faulty circuit, whereas, it is ’1’ for a good circuit.This faulty circuit behaviour causing the inhibition of transition from state 110 to 111 is shown as dotted lines in Figure 9.4.

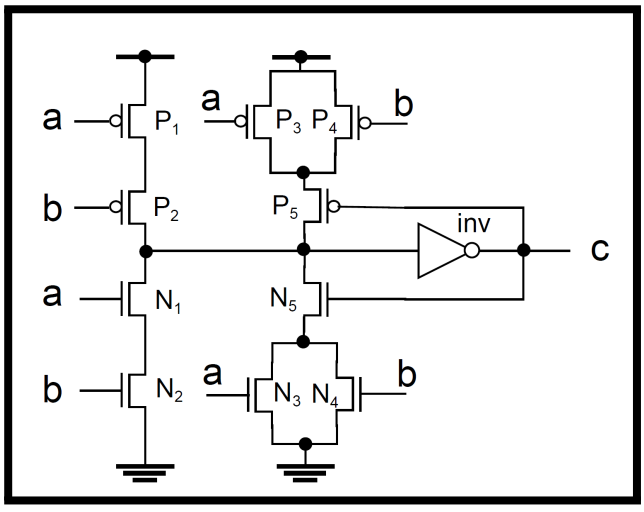


Figure 9.5: C-element Design

Test pattern for the transistor p3 stuck-on fault is obtained by traversing through the SG from the initial state to the state next to where the inhibition occurs. Thus the test pattern for this

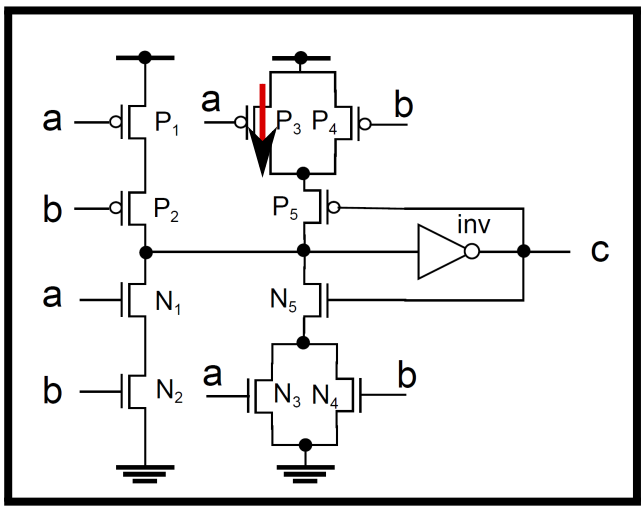


Figure 9.6: Transistor P3 Stuck-on fault in the C-element

fault will be "000-100-110-111" or "000-010-110-111". It should be noted that the circuit is operated in the fundamental mode during the testing process and hence the circuit is allowed to stabilize before applying a new pattern. The equivalent STF is represented by a '0' mark on the arc from b+ to c+. Thus test patterns for the remaining faults can be obtained by traversing the good circuit with the extracted logic of faulty machine. Once the test patterns are obtained they can be fault simulated to obtain the fault coverage. Fault coverage results for several benchmark circuits are discussed in the "Experiments" section.

9.4 Test Method

The proposed test method involves three major steps: BLIF2Spice netlist conversion, Test pattern generation and Fault simulation. Figure 9.7 shows the components involved in test generation and the test flow for the proposed method. The netlist of the Circuit under Test (CUT) is a BLIF (Berkeley Logic Interchange Format) file and its library file (genlib) generated by the petrify tool [CKK+97]. Along with the netlist the corresponding STG file of the circuit is parsed in. The netlist file is pre-processed by the custom *"BLIF2Spice"* tool written in perl and bash script. The pre-processed netlist is then sent to the switch/transistor level fault simulator. Meanwhile, the test pattern generator will read the STG file of the CUT and generate the test vector for all the faults in the CUT. The test generator is written in C++ using Standard Template Libraries (STL). A custom transistor level (spice deck) fault library for all the gates in the library generated by the petrify tool is generated and is available for the fault simulator. The details of the *"blif2spice"* tool, test pattern generator and the fault simulator are described further.

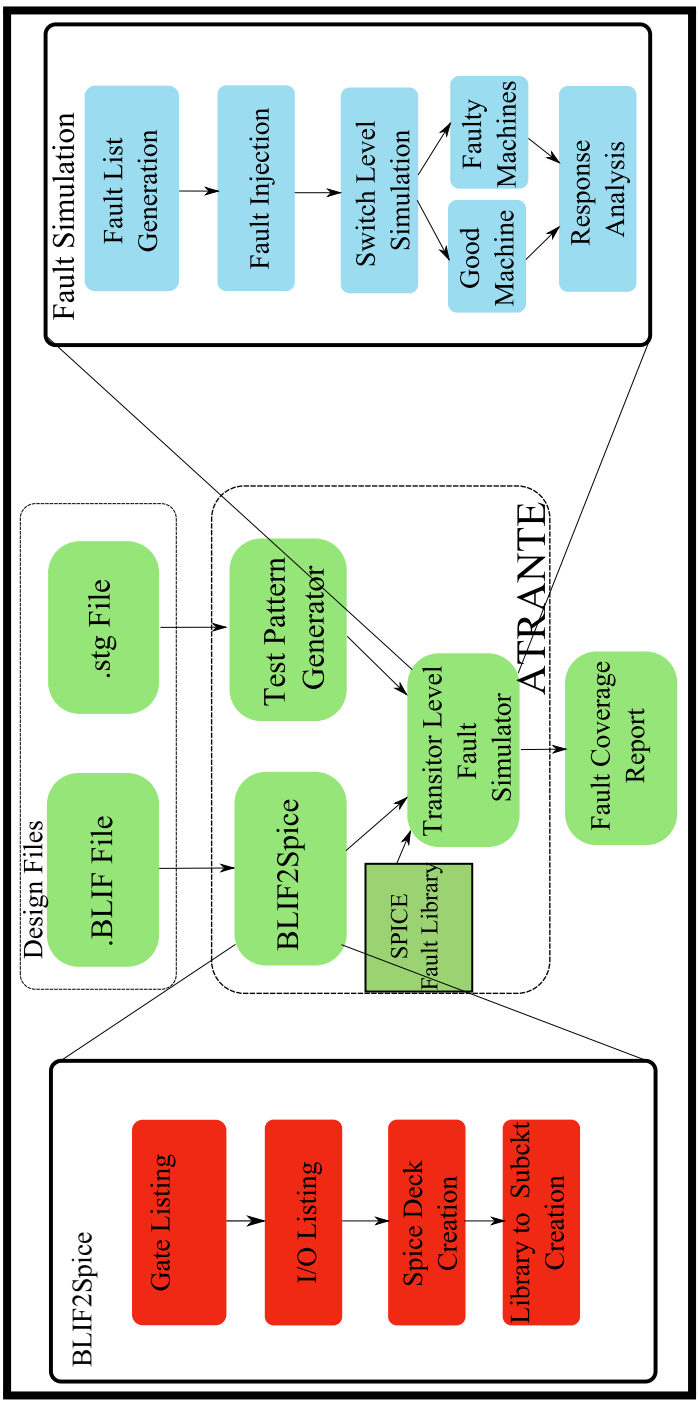


Figure 9.7: Test Methodology

First, the *"blif2spice"* tool is a pre-processor tool used to parse the BLIF file and convert them in to standard library specific spice deck. Standard libraries of 0.18 um technology are used for this conversion. Each gate in the blif file is converted in to its equivalent spice representation for 0.18 um technology. Thus the output of the tool will be a spice netlist of the CUT. Second, the test pattern generator follows the algorithm give in the algorithm 14. Details of the test generation algorithm are described in the subsection Test Generation. The tool implemented

with this algorithm reads in the STG file, SG file and enumerates the STG and SG based on the Algorithm 14 to generate test patterns and the output file is the test vector file (.vec file) with the list of test patterns.
Finally, the fault simulator is built using the Perl scripts and the commercially available switch level logic simulator. The fault simulator reads in the spice fault library (fault-lib.sp), spice netlist from *"blif2spice"* (CUT.sp) and the test pattern file (.vec file).

First the fault list is generated by the *"faultmachine-generator"* script. And then, this script injects all possible stuck-open and stuck-ON in to the CUT and generates all the possible fault machines. The detailed process is shown in the subsection Fault simulation. Once all the fault machines are generated, the good machine simulation is carried out using the switch level logic simulator. The response of the good machine is stored. Then the fault machines are simulated and their response is compared with the good machine response to report the fault to be detected or undetected. This report is finally printed as the test report. One further optimization that can be carried out will be the fault dropping, which further reduces the fault simulation time.

9.4.1 Fault Model

The fault models used in this method are: 1) stuck-at-true, and 2)stuck-at-false for test generation. For fault simulation, stuck-at-open and stuck-at-close faults are covered. Thus this method covers the defect level faults using the STG level fault models. To be precise, functional level fault model is used to cover defect level fault.

9.4.1.1 Stuck-at-True Fault

Stuck-at-true fault is the fault in the STG level that one of the pre-condition of the transition is always true. This fault is represented by a "1" in the arc (Rout+ to Aout+ in Fig. 9.2) corresponding to that precondition. Input stuck-at faults and faults causing extra transitions can be mapped in to this type of fault. For example, the stuck at true fault in Fig. 9.2, describes that the good circuit's STG will always have Aout+ transition after Rout +, but the faulty circuit with this fault will have Aout + transition before Rout + transition due to the stuck at fault in the corresponding node in the circuit. At circuit level, this might be due to the pin Aout being short with other node with value 1 and the path driving Rout to Aout being open.

9.4.1.2 Stuck-at-False Fault

Stuck-at-false fault is the fault in the STG level that one of the pre-condition of a transition is always false. This fault is denoted in the STG by "0" in the arc (Rin- to Aout- in Figure 9.2)

corresponding to that precondition. Figure 9.2 shows an example of this fault. This fault may occur when the node connected to Aout is stuck at 1 and hence Aout will never go to 0. Output stuck-at faults and faults causing inhibition of transitions can be modeled with this fault.

9.4.2 Test Algorithm

The test generation algorithm underlying this method is described in this subsection. The STG for the good machine is used to generate the test patterns. The test patterns for fault models "stuck-at-true" and "stuck-at- false" are generated by enumerating the states over the STG. For every test pattern, the traversal starts from the initial state of the STG. This provides the assumption that the generated test pattern have to be applied after restting the circuits to the intial state everytime before applying the test patterns. From the initial state, the traversal continues through the STG to reach the faulty state provided by the fault model. The path traversed from the initial state to the faulty state provides the set of states and the set of state values. These state values are the test for the faulty state. The set of these states are stored as test sequences for the corresponding fault.

The above mentioned steps are continuously applied for all the stuck-at-true and stuck-at-false faults to obtain all the test patterns for the DUT. The test generation algorithm is shown in Algorithm 14. The algorithm takes a STG (graph g1) and a SG (graph sg1) as input. For each edge "ei" of the STG is compared with each edge (eis) of the SG. When the source vertex of the ei equals the transition name/edge name of the sg1, then following two steps are carried out. First, for each vertex of the sg1, a comparison is made to check whether the vertex is same as the source vertex of the eis. If they are same, then for each vertex Vp in the predecessor list of the vertex, the pattern corresponding to each state is stored in the "testvectortrue" pattern list. The above steps are again carried out for generating the "testvectorfalse" pattern list, except that instead of comparing the source vertex of the edge "eis", the target vertex is compared with the transition/edge name of the STG. Thus two list of vectors namely "testvectortrue" and "testvectorfalse" are created which contains the test patterns for all the stuck-at true and stuck-at false faults.

Example: As an example to describe the test generation process Figure 9.8 shows the step-by-step process of test generation for the fault in a C-element. The SG graph and STG graph shown in Figure 9.4 is used to generate the tests. This example shows the test for stuck-at-false fault on the arc "a+". As mentioned before, first the algorithm selects the fault "a-" stuck-at-true from the STG and enumerates the SG and finds the edge named a-. After finding the edge, since the fault is stuck-at-true fault, it checks whether the output bits in the source and target pattern of the edge "a-" are changing. In this example (shown in Figure 9.8.a), the source and target

patterns of the edge "a-" are 111 and 011. Since the third bit is same, the algorithm traces the successor list of the target to find the node with its pattern having a flipping output bit. Thus in this example, the successor of the target pattern is the node with pattern "001"(shown in Figure 9.8.b). But its output is not flipping yet. So the successor of this node "000" reached. The pattern has its last bit flipped. Thus the algorithm chooses this pattern as the first pattern. From this node, it traces back each node to add the patterns of those nodes to the test vector list. Thus in Figure 9.8. c, the node with pattern 001 is reached and is added to the test vector list with the updated list 001,000. Similarly, the test vector list gets updated from {001,001}, {011,001,000}, ... {000,100,110,111,011,001,000} as shown in the figures Figure 9.8. d - h.

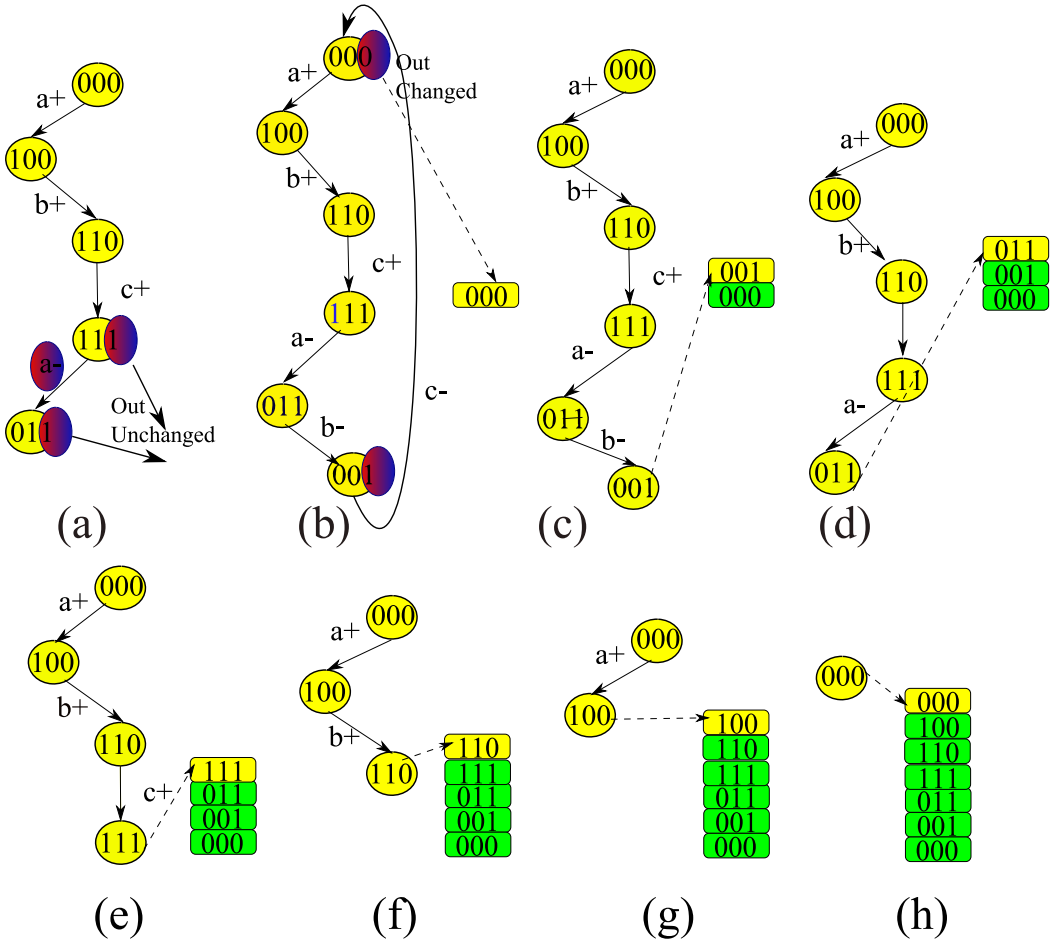


Figure 9.8: Test Generation Example

9.4.3 Fault Simulation

The fault simulation process is detailed in this section with a relevant example. During fault Injection, the faulty transistor representing either stuck-open or stuck-on fault is plugged in to the good circuit spice deck. The hierarchical view of the good circuit and faulty circuit

from DUT to transistor level description is shown in the Figure 9.9. The DUT gate level description of the circuit (shown at the top level of hierarchy) is mapped to the standard gate module library with common gates(including C-element) which is shown in the second level of the hierarchy. Until these two levels, both the good and faulty circuit description will be same. For the third(bottom) level, the spice deck for each gate is created as a library file. The library file includes not only good circuit spice decks(bottom left) but also the fault circuit spice decks(bottom right). Thus to inject a fault in the DUT, the faulty spice deck corresponding to the faulty transistor is plugged in to the DUT by replacing the original spice deck of the gate in which the transistor fault has to be injected. For example, to inject a stuck-on fault on one of the p-transistor in an OR gate, the OR-gate spice deck will be replaced by the OR-Faulty gate spice deck.

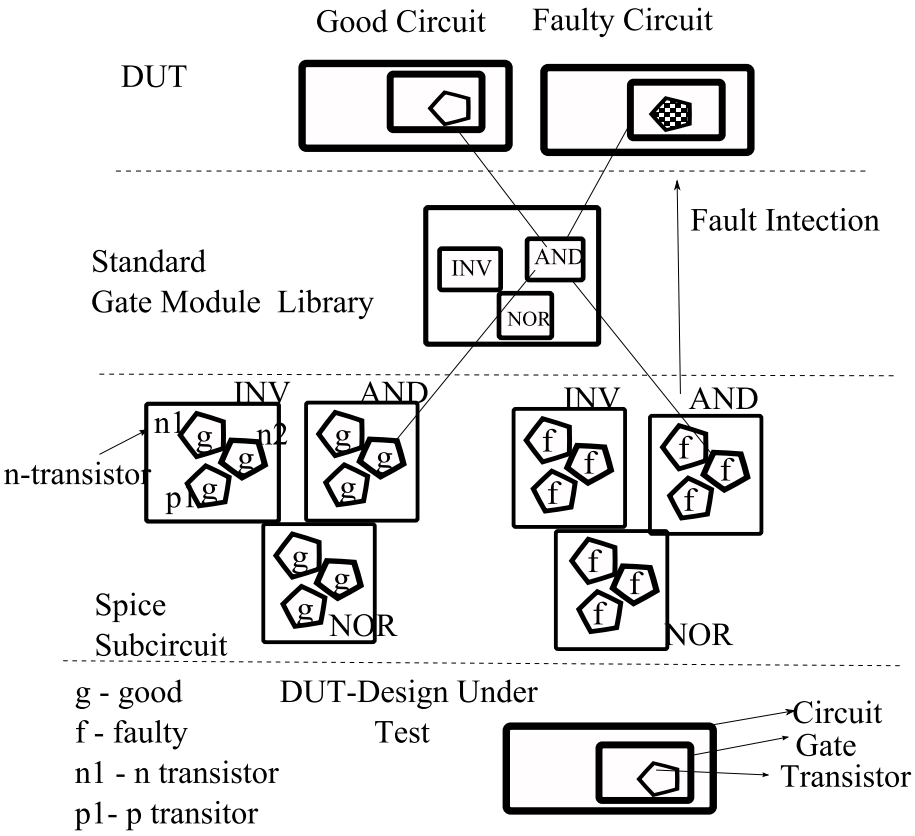


Figure 9.9: Fault Injection

9.5 Experiment Results

This section provides the experimental results on the proposed test generation system. Both the test generation algorithm and the fault simulation method were implemented in C++ and scripts to automate and integrate the whole ATPG system. The logic simulator is a commercial switch level simulator.

9.5.1 Test Generation and Fault Simulation

The results of the test generation algorithm are shown here. The results of test pattern generation algorithm for the asynchronous benchmarks are shown in the Table 9.4. The transistor level characteristic of the benchmarks is shown in the Table 9.3. The fault simulation results are shown in the Table 9.5.

Table 9.3: Transistor Level Circuit Characteristics

Ckt	No of Inputs	No of Outputs	No of transistors	No of STTF	No of Tx faults
chu150	3	3	32	64	240
converta	2	3	32	64	503
dff	2	1	52	104	288
ebergen	2	3	16	32	336
hazard	2	2	54	108	108
mstr-rd	6	7	80	160	840
mp-fd-pkt	3	5	52	104	264
nak-pa	4	6	48	96	528
nowick	3	3	42	84	336
rm-rd-sbf	5	6	54	108	576
rcv-setup	3	2	40	80	216
rpdft	4	1	44	88	408
sbf-rm-wr	5	7	58	116	504
sbf-snd-ctl	3	5	62	124	552
sbf-snd-pkt2	4	5	60	120	672
tri-snd	3	6	60	120	840
vbe5b	3	3	32	64	216
vbe5c	3	3	30	60	120
wrdatab	4	6	66	132	1008

Table 9.4: ATRANTE Test Generator Results

Ckt	No of states	No of STTF	Test Vector Size	No of Test Patterns	Cycles
chu150	26	32	6	48	2
converta	18	32	5	28	1
dff	36	52	4	36	1
ebergen	18	16	5	30	2
hazard	12	24	4	22	1
mstr-rd	2108	80	14	8942	7
mp-f-pkt	22	52	8	46	-
nak-pa	58	48	10	124	-
nowick	20	42	6	47	-
pe-send-ifc	117	164	10	442	12
rm-rd-sbf	39	54	11	67	-
rcv-setup	14	40	5	23	-
Rpdft	22	44	5	49	-
sbf-rm-wr	64	58	12	119	2
sbf-snd-ctl	27	62	8	50	8
sbf-snd-pkt	28	60	9	49	6
tri-snd	336	60	9	1296	44
vbe5b	24	32	6	46	-
vbe5c	24	30	6	43	-
wrdatab	216	66	10	723	222

9.5.2 Analysis

Detailed analysis on the fault coverage and performance of the algorithm is shown in this example.

9.5.2.1 Fault Simulation

Since the fault coverage results reported in [RCPP97],[EOL02] are gate stuck-at-fault coverage percentage, they will be a subset of the total fault coverage. The fault coverage comparison is shown in Table 9.5. Only three benchmarks were reported in [EOL02] which were synthesized using the same library used in [RCPP97]. Hence the comparison can be made only with these benchmarks. Furthermore, test patterns for several benchmarks were generated using the test

Table 9.5: Fault Coverage Results from Fault Simulator

Ckt	No of Faults	No of Detected	No of Un-Detected	Fault Coverage	No of patterns
chu150	112	99	13	88.39	240
converta	87	80	7	91.95	503
ebergen	89	84	5	94.38	336
hazard	108	98	10	90.74	108
mp-fd-pkt	140	130	10	92.85	264
nak-pa	244	234	10	95.90	124
nowick	160	139	21	86.87	336
rm-rd-sbf	312	268	44	85.89	576
rcv-setup	94	89	5	94.68	216
rpdfft	172	159	13	92.44	408
vbe5b	110	101	9	91.81	216
vbe5c	89	84	5	94.38	120

pattern generator implemented. The results of the tests generated by the test pattern generator is shown in the Table 9.4. The transistor level circuit characteristics of all the benchmarks used by the test generator is shown in the Table 9.3

Most of the faulty machines are redundant faults (are confirmed to operate error free in the presence of the fault) and hence the fault coverage will be actually high, when these faults are dropped during fault coverage calculations. The test patterns achieved higher fault coverage for the benchmark nak-pa. Totally 244 fault machines were simulated and 235 faults were detected leading to a fault coverage of 97%.

9.5.2.2 Comparison with State-of-the-art

An attempt on complete ATPG system for asynchronous circuits at transistor level is not reported in the literature until now(up to author's knowledge). Hence direct comparison with the current state-of-the-art is not possible. Comparison that can be made will be with the work in [Eic65] and [SM04a], but they are for gate level fault. Comparison with gate level test methods will not be appropriate.

9.5.2.3 Complexity and Scalability

The method introduced is completely scalable for the asynchronous control circuits. The test generation algorithm traverses through two graphs(STG) for comparing the edge values and also enumerates the adjacency vertices of source edges. The complexity of the test generation algorithm is $|E^2|.|V|.|V_a|$, where E is number of edges of the graphs, V, the number of vertices and V_a is the number of adjacent vertices in worst case.

9.5.2.4 Limitations

In this analysis, fault simulation was carried without any fault collapsing being applied. Hence the fault simulation time can be considerably reduced. Also, the number of test can be reduced when the fault dropping is carried out during every test pattern simulation. Also the fault simulation process uses switch level logic simulator to detect faults, efficient switch level fault simulation tool at transistor level will improve the fault simulation time and reduce the resource/memory needed for the same.

9.6 Summary

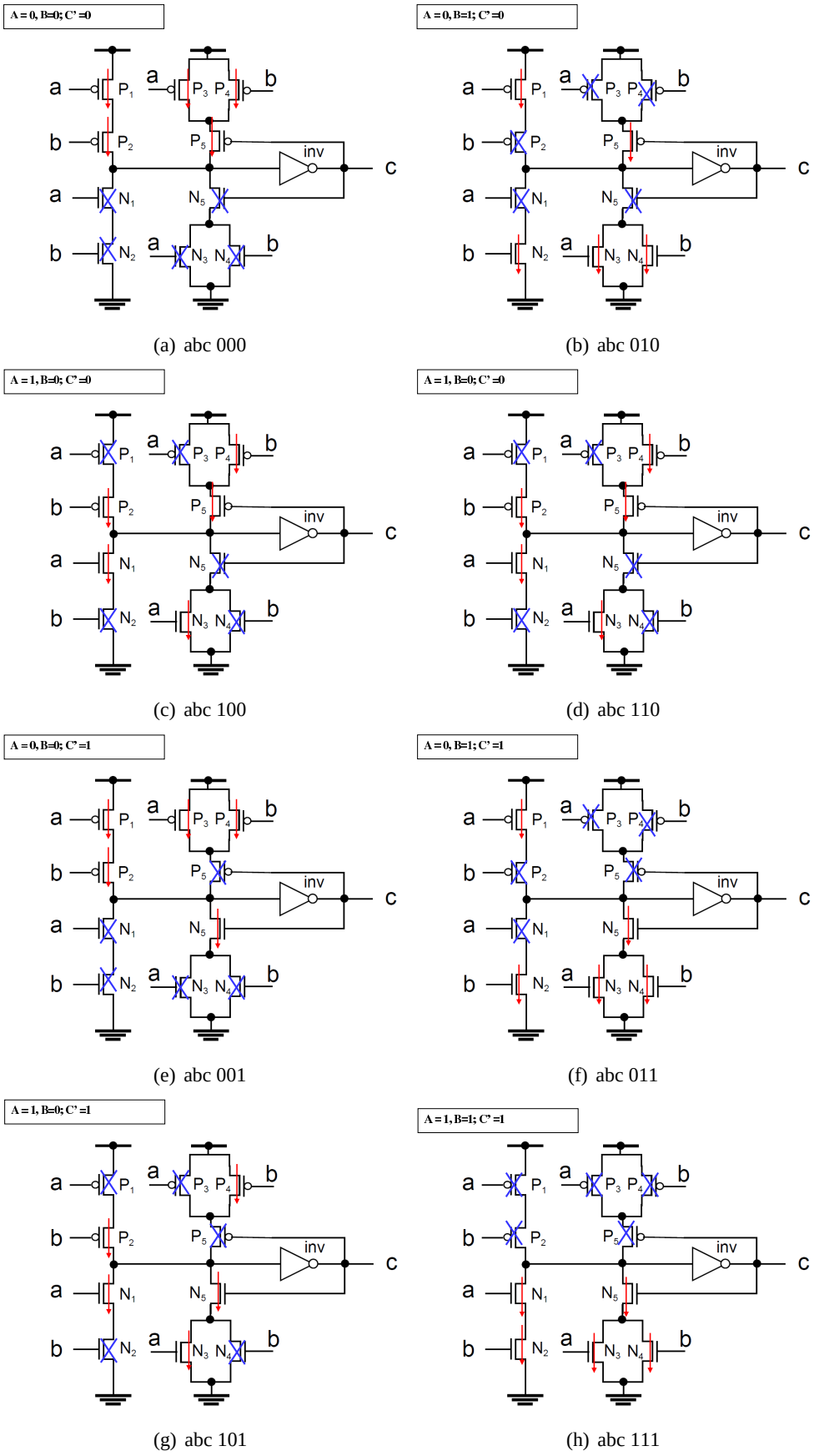
In this chapter, a transistor level test generation methodology based on transition fault model on STG was proposed. A test generation algorithm was proposed and test results of the implemented test pattern generator were reported. Fault coverage of 88-97% at the transistor level is obtained by applying this method. The test patterns generated in the proposed method is higher compared to the gate level fault simulation. This is because, only the circuit structure is enumerated for the fault simulation in gate level simulation, whereas in the proposed method, the state graph is enumerated and the test patterns are generated for the transistor level faults. Though fault coverage for several benchmarks were reported by using a transistor level logic simulator in this chapter, not all the benchmarks could be fault simulated in the same manner. A robust transistor level fault simulator is needed for fault coverage reporting. Implementing a custom transistor level fault simulator will be the future work/extension.

Algorithm 14 Proposed Test Generation Algorithm

Input: State Transition Graph STG,State Graph SG

Output: Test Pattern vector - "Testvector"

```
21 begin
     Data: vector pattern-true,pattern-false  vector<vector> testvectortrue, testvectorfalse  int
           parent
22   resetstate ⟵ resetstateofSG  shortest_path(State graph SG)  foreach each edge ei in
     STG g1(V,E) do
23     foreach each edge eis in graph sg1(Vs,Es) do
24       if source vertex of ei = name of the edge eis then
25         foreach vertex usg in graph sg1 do
26           if (usg = source of eis) & (out-bits flip) then
27             foreach vertex ‘usg2 in succ. list of usg do
28               if outbit of usg2 ≠ usg then
29                 usg1=usg2  exit
30               end
31             end
32             if no usg1 then
33               “‘No pattern!”
34             end
35           end
36           else
37             foreach vertex Vp in pred. list of usg do
38               repeat
39                 pattern.push_back(Vp)
40               until Vp = reset_state;
41               pattern.push_back(reset_state)  testvectortrue.push_back(pattern)
                 pattern.clear
42             end
43           end
44           if (usg = target of eis) & (out-bits flip) then
45             Same steps as line 9 to 17
46           end
47           else
48             Same steps as lines 20-27  test stored in testvectorfalse
49           end
50         end
51       end
52     end
53   end
54   return testvector = testvectortrue + testvectorfalse;
55 end
```



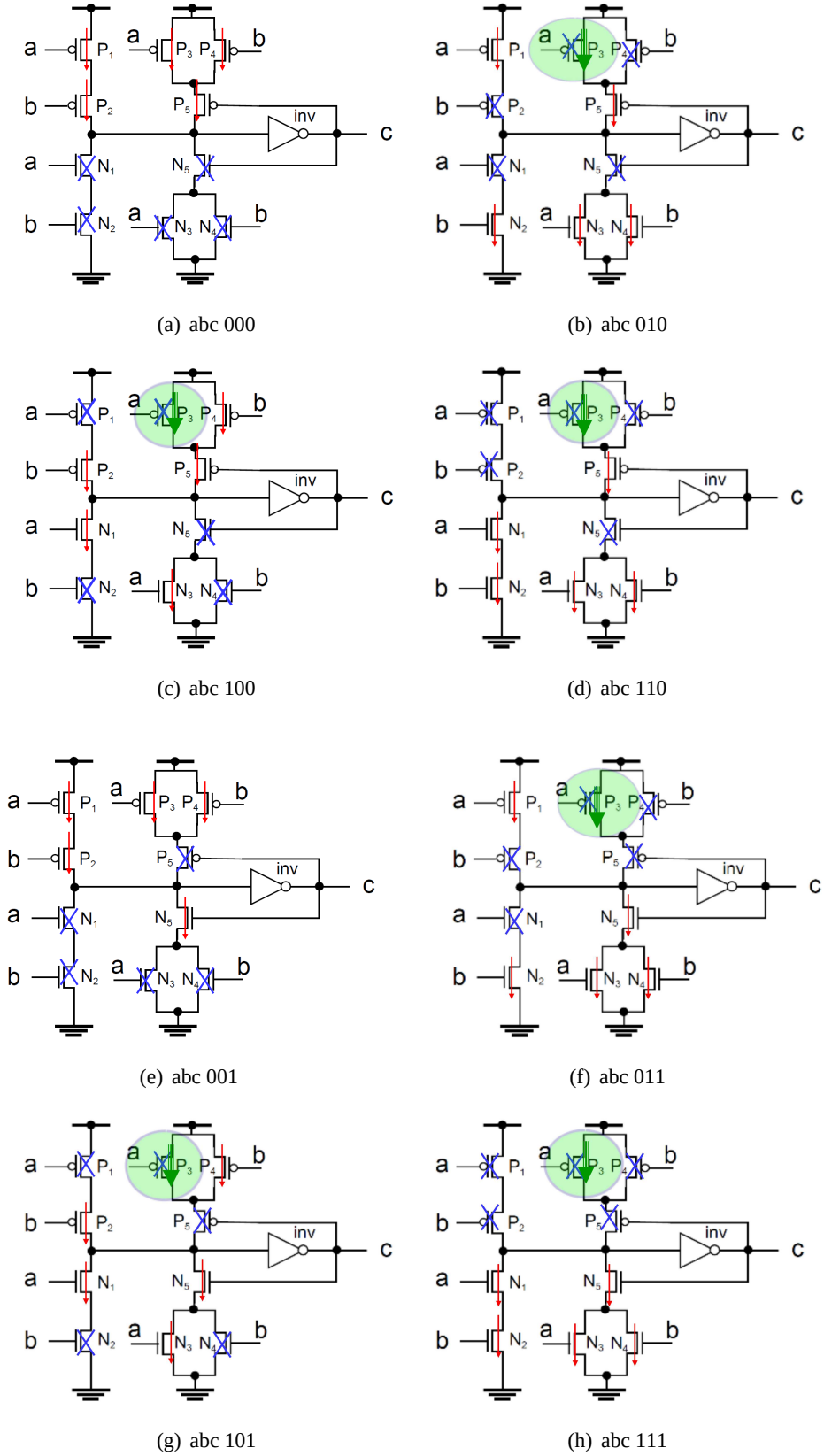
(a) abc 000 (b) abc 010

(c) abc 100 (d) abc 110

(e) abc 001 (f) abc 011

(g) abc 101 (h) abc 111

Figure 9.10: Eight different transitions in the transistor level Symmetric C-element design shown in (a), (b), (c), (d), (e), (f), (g), and (h) for values of abc' = 000,010,100,110,001,011,101,111 respectively.



(a) abc 000 (b) abc 010

(c) abc 100 (d) abc 110

(e) abc 001 (f) abc 011

(g) abc 101 (h) abc 111

Figure 9.11: Eight different transitions in the transistor-level symmetric C-element design with transistor p3 stuck-on shown in (a), (b),(c),(d),(e),(f),(g), and (h) for values of abc' = 000,010,100,110,001,011,101,111 respectively.

Chapter 10

Conclusion

10.1 Summary

This thesis has explored the possibility of generating good test patterns to test the asynchronous circuits with higher fault coverage and optimum area overhead. Four different methods were introduced in this thesis each of them exploiting the circuit structure of the asynchronous circuits, models and algorithms from graph theory and the currently available industrial tool for synchronous circuits to develop robust test generation methods. A brief summary of the thesis is presented below.

A detailed background on Asynchronous circuit design and testing challenges in the asynchronous paradigm was provided in **Chapter 2**. The chapter further covered the topics on testing (especially ATPG and scan design) with details on full scan and partial scan design. Several ATPG methods were described followed by the introduction of several fault models.

Chapter 3 covered the literature review over the related works involving the design for testability (DFT) and test generation of asynchronous circuits. The literature review in this chapter included the topics: 1) Design for test (DFT) for asynchronous circuits, 2) ATPG methods for asynchronous circuits, 3) Self checking designs of asynchronous circuits, 4) Testable asynchronous circuit design, 5)Test Generation at defect/transistor level, and 6) Delay fault testing of asynchronous circuits.

Chapter 4 carried out a comparison study on two automatic test pattern generation methods. Background on the State Transition Graph (STG) based automatic test pattern generation was described. The test pattern generation based on the scan insertion technique are introduced. Then a comparison of test generated by these two approaches for a number of small benchmarks are presented. The chapter was concluded by stating the drawbacks and improvements to be incorporated in the proposed test methods.

ABALLAST method was introduced in **Chapter 5**. The chapter presented detailed background on the balanced structures which was used in the BALLAST method. The test methodology proposed applied the balanced structures in the asynchronous circuit context and effective test pattern generation method was developed. The algorithms involved in this method were briefed in detail. This method used the "check" and the "balance" routine used in BALLAST method to check and create the balanced structure of the asynchronous netlists. A new cyclic to acyclic conversion algorithm proposed forms the main contribution of this method.
AGLOB Method was introduced in **Chapter 6**. This chapter introduced two different methods of partial scan selection for asynchronous circuits namely Aglob1 and Aglob2. The cyclic to acyclic circuit conversion technique was also used here to develop the test patterns for the asynchronous circuits. Global loops present in the asynchronous circuits e broken to create an equivalent asynchronous circuit that can provide higher fault coverage. Area overhead was reduced considerably in this method.
Chapter 7 introduces the method based on Set Covering Problem. Background on Set Covering Problem and cycle enumeration methods were provided. The method of weighted set covering problem to find the minimum set cover was chosen as it had reported good performance. This method reported good fault coverage and reduction in area overhead. A critical analysis of impact of number of C-elements present in the benchmark and its corresponding fault coverage was carried out. Eventhough, a concrete conclusion could not be reached on the impact, it gave a good insight on the impact of the circuit structure and the location of memory elements over the fault coverage of the same. Following this analysis, overall case study was carried out finally. All the three gate level test methods of test generation were studied with respect to fault coverage, test coverage and area overhead. Detailed results of these three methods were analyzed.

In **Chapter 8** a high level extraction method for asynchronous circuits was constructed. This method was based on partitioning all the memory elements into registers and combinational gates in to combinational cloud. Several asynchronous benchmarks were applied to this method to extract their corresponding high level representation. These high level representations can be used to process the design at high level for test pattern generation. This will considerably increases the performance of the algorithm/test method developed on top of these extracted views.
Chapter 9 introduced the method ATRANTE, a transistor level test generation method. This chapter provided further details justifying the need for transistor level test generation in the introduction. Then the test methodology for this method was briefed. The pruning space for the test pattern generation and fault list generation was the State Transition Graph (STG) rather than the circuit netlist. The fault model used in this method of test generation was a model

different from conventional circuit oriented models (ex.stuck-at model). The new fault model is called transition fault model which was modelled over the STG specification of the circuits. This method provided additional fault coverage for the faults at transistor level compared to the other test generation methods that used the same STG based pruning space. The fault coverage was obtained by mapping the transition fault model to the gate level and transistor level faults in the original circuit.

10.2 Future Works

Following future avenues are possible to work further and continue this thesis.

1. **Delay Fault Testing**
   Delay fault testing of asynchronous circuits is the area which is not explored much until now. Only few works are available in the literature for delay fault testing of asynchronous circuits as mentioned in the chapter 3. Since the asynchronous circuits are composed of delay components, testing the delay faults of asynchronous circuits is very important. Delay fault testing is still an active research in the synchronous design paradigm too. Developing DFT methods and ATPG techniques for the delay faults in asynchronous circuits will be needed in coming years as ITRS predicts more than 50% of the design in the middle of the next decade will be DFT blocks requiring delay fault test.

2. **Fault Simulator at Transistor Level**
   In [FS88a],[FS88b], [Cor91] inductive fault analysis for defect level faults were analysed extensively. The fault simulator at transistor level were explored in past decades, but due to the complexity of transistor level simulation and resource constraints the advancement is slowed down. But with the current advancement in parallel programming and many-core processing power, new simulator implemented by parallel programming techniques can be anticipated to handle the complexity of these simulators.

3. **New ATPG algorithm design**
   The gate level test methods proposed in this thesis (**Chapter 6, Chapter 7 and Chapter 8**) have incorporated synopysys's Tetramax in the methodology for generation of test patterns for asynchronous circuits. Hence the effectiveness of the methods is confined within the test generation and fault reporting effectiveness of the Tetramax tool. So developing new ATPG algorithms for gate level testing of asynchronous circuit that can compare with the algorithm of Tetramax will be a promising contribution towards robust ATPG for asynchronous circuits. One such effort was made in Chapter 9 after the lessons learned from the Chapter 6 to Chapter 8. The new algorithm should take into account

the hazards caused by the asynchronous circuits and should also be capable testing asynchronous circuits that is operating in non-fundamental mode also. This way the state of the art for asynchronous circuit test generation can be advanced.

4. **New Fault Models**
   Fault models currently available for gate level testing are sufficient for testing stuck-at faults. But for defect level faults it is necessary to model new fault models. Also with the current technology node reaching deep submicron level, it is necessary to build new fault models to handle the defects that will be rising due to the compression in feature size in these nodes.

5. **Ant colony optimization based test generation**
   Bio-inspired methods for developing test generation methods for asynchronous circuits are not explored yet up to the knowledge of the author. These algorithms are effective for developing scan selection algorithms. For example, Ant Colony Optimization based set covering problem can be formalized to develop a new partial scan selection method for asynchronous and synchronous circuits. Some literatures are emerging in the field of synchronous circuit testing. Hence the same idea can be passed on for asynchronous circuit test generation.

6. **Reversible Asynchronous test generation**
   Reversible computing is a newly emerging computing paradigm which is promising for the beyond CMOS Era. For these types of computing architectures, asynchronous circuits based designs are best match. So developing new test generation methods for these Reversible Asynchronous designs will be a long term investment in terms of test generation for future designs.

Thus in author's opinion, the basics of test generation principles needs complete refinement and advancement in terms of fault models, test generation, fault simulation and DFT methods for developing successful test methods for asynchronous circuits.

10.3 Conclusion

This thesis was motivated towards developing four different test generation methodologies for the asynchronous circuits. ABALLAST method presents a partial scan and automatic test generation methodology based on a novel adaptation of BALLAST for asynchronous circuits and time frame unrolling. Balanced structures are used to guide the selection of the state-holding elements that will be scanned. Fault coverage was improved from range 16 .20 %-69.57 to 76.78 -94.37%. Three CAD tools written in C/C++ namely "Aballast","Cyclic2Acyclic" and

"Blif2graph" were outcome of this work. In AGLOB two test generation algorithms((aglob1) and(aglob2)) were proposed in this project which uses cyclic to acyclic circuit conversion, partial scan based test generation and SCC based, graph density based memory element selection as aids. The fault coverage was improved from 0 - 85% to 71 - 98 %. A CAD tool named AGLOB12 in C++ was also an outcome of this work. For ATRANTE method, the main motivation of developing ATPG is supplemented by transistor level test generation. Here the Petrinet based representation of the asynchronous circuits and efficient mapping of transistor level faults to STG based fault models were used to implement this ATPG methodology. The test patterns generated covered the transistor level faults in addition to the gate level faults. The CAD tool ATRANTE developed for this tool is believed to be the first asynchronous transistor level test generator. ASCP is a test methodology developed based on a good set covering problem solution. Future work can be focused towards developing methodologies for delay fault testing. Developing a new fault simulator for the asynchronous circuits will aid a swift test development research. New ATPG method for transistor/defect level test method could be a promising track to carry on. New fault models are needed to accurately address the faults to be tested in asynchronous circuits.

Bibliography

[ABE05] A.Efthymiou, John Bainbridge, and Doug A. Edwards. Test pattern generation and partial-scan methodology for an asynchronous SoC interconnect. *IEEE Trans. VLSI Syst*, 13(12):1384–1393, 2005.

[ARM] ARM. ARM.

[BA05] F.te Beest and A.Peeters. A multiplexer based test method for self-timed circuits. In *In Proceedings. 11th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 166–175, 2005.

[BCR96] Savita Banerjee, Srimat T. Chakradhar, and Rabindra K. Roy. Synchronous test generation model for asynchronous circuits. In *Proc. International Conference on VLSI Design*, January 1996.

[BE00] A. Bardsley and D. A. Edwards. The Balsa asynchronous circuit synthesis system. In *Forum on Design Languages*, September 2000.

[Bee03] Frank J. te Beest. *Full scan testing of handshake circuits*. PhD thesis, Twente University, Enschede, The Netherlands, May 2003.

[BM88] Steven M. Burns and Alann J . Martin. Synthesis directed translation of concurrent programs into self-timed circuits. *In J. Allen a.nd F. Leighton editors, Proceedings of the Fifth MIT Conference on Advanced Resenrch in. VLSI,MIT Press*, pages 35–50, 1988.

[BM91] Peter A. Beerel and Teresa H.-Y. Meng. Testability of asynchronous self-timed control circuits with delay assumptions. In *Proc. ACM/IEEE Design Automation Conference*, pages 446–451. IEEE Computer Society Press, June 1991.

[BPvBK03] Frank te Beest, Ad Peeters, Kees van Berkel, and Hans Kerkhoff. Synchronous full-scan for asynchronous handshake circuits. *Journal of Electronic Testing: Theory and Applications*, 19:397–406, 2003.

[BR] Girard P. Pravossoudovich S. Bernardi P. Bosio, A. and M. S. Reorda. An efficient fault simulation technique for transition faults in non-scan sequential circuits. *In Proceedings of the 2009 12th international Symposium on Design and Diagnostics of Electronic Circuits&Systems (April 15 - 17, 2009). DDECS. IEEE Computer Society, Washington, DC*, pages 50–55.

[BR95] J. A. Brzozowski and K. Raahemifar. Testing C-elements is not elementary. In *Asynchronous Design Methodologies*, pages 150–159. IEEE Computer Society Press, May 1995.

[BS90] B. Berger and P.W. Shor. Approximation algorithms for the maximum acyclic subgraph problem. *In Proc. First ACM SIAM Symposium on Discrete Algorithms*, pages 236–244, 1990.

[BY07] Arjan Bink and Richard York. ARM996HS: The first licensable, clockless 32-bit processor core. *IEEE Micro*, 27:58–68, March 2007.

[CA90] K. T. Cheng and V. D. Agrawal. A partial scan method for sequential circuits with feedback. *IEEE TRANSACTIONS ON COMPUTERS*, 39(4):544–548, April 1990.

[CB90] Gerald Carson and Geatano Borriello. A testable CMOS asynchronous counter. *IEEE Journal of Solid-State Circuits*, 25(4):952–960, August 1990.

[CG86] T.-A. Chu and L. A. Glasser. Synthesis of self-timed control circuits form graphs: An example. In *Proc. International Conf. Computer Design (ICCD)*, pages 565–571. IEEE Computer Society Press, 1986.

[Chu87] Tam Anh Chu. Synthesis of self-timed VLSI circuits from graph-theoretic specifications. In *International Conference on Computer Design*, pages 220–223. IEEE Computer Society Press, 1987.

[CKK+96a] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. Technical report, Universitat Politècnica de Catalunya, 1996.

[CKK+96b] Jordi Cortadella, Michael Kishinevsky, Alex Kondratyev, Luciano Lavagno, and Alexandre Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. In *XI Conference on Design of Integrated Circuits and Systems*, Barcelona, November 1996.

[CKK+97] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325, March 1997.

[Cor91] F. Corsi. Inductive fault analysis revisited [integrated circuits]. *IEE Proceedings G Circuits, Devices and Systems*, 138(2):253–263, April 1991.

[DGY90] Ilana David, Ran Ginosar, and Michael Yoeli. Self-timed is self-diagnostic. Technical Report EE PUB No. 758, Department of Electrical Engineering, Technion, November 1990.

[EA00] E.Marchiori and A.Steenbeek. An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling. *Real World Applications of Evolutionary Computing. Springer-Verlag, LNCS 1083*, pages 367–381, 2000.

[Edw03] S. A. Edwards. Making cyclic circuits acyclic. In *Proc. Design Automation Conference*, pages 159–162, 2–6 June 2003.

[Eic65] E. B. Eichelberger. Hazard detection in combinational and sequential switching circuits. *IBM Journal of Research and Development*, 9:90–99, March 1965.

[Ela] Elastix. Elastix corporation.

[EOL02] S-H.Kim E. Oh and D.-I. Lee. High level test generation for asynchronous circuits using signal transition graphs. *Journal of the Korean Physical Society*, 40-1:193–198, 2002.

[E.P97] E.Paster. Structural methods for synthesis of asynchronous circuits from Signal Transistion Graphs. *Thesis: Universitat Polytecnica de Catalyuna*, 1997.

[ES93] X.Lin Eades, P. and W.F. Smyth. A fast and effective heuristic for feedbac arc set problem. *In Proc. Letter*, 47:319–323, 1993.

[ES95] Kent L. Einspahr and Sharad C. Seth. A switch-level test generation system for synchronous and asynchronous circuits. *Journal of Electronic Testing: Theory and Applications*, 6(1):59–73, February 1995.

[FMHG05a] Xinyue Fan, W. Moore, C. Hora, and G. Gronthoud. A novel stuck-at based method for transistor stuck-open fault diagnosis. pages 9 pp. –386, nov. 2005.

[FMHG05b] Xinyue Fan, W. Moore, C. Hora, and G. Gronthoud. A novel stuck-at based method for transistor stuck-open fault diagnosis. In *Test Conference, 2005. Proceedings. ITC 2005. IEEE International*, pages 9 pp. –386, 2005.

[FS88a] F. J. Ferguson and J. P. Shen. A CMOS fault extractor for inductive fault analysis. 7(11):1181–1194, Nov. 1988.

[FS88b] F. J. Ferguson and J. P. Shen. Extraction and simulation of realistic CMOS faults using inductive fault analysis. In *Proc. 'New Frontiers in Testing'. International Test Conference*, pages 475–484, 12–14 Sept. 1988.

[Fuj85] H. Fujiwara. Fan: A fanout-oriented test pattern generation algorithm. *Proc. of ISCAS 85*, pages 671–674, June 1985.

[GB90] R. Gupta and M. A Breuer. The ballast methodology for structured partial scan design. *IEEE Trans. Comput.*, 39, 4:538–544, Apr. 1990.

[Goe81] P. Goel. An implicit enumeration algorithm to generate tests for combinational logic circuits. *IEEE Transactions on Computers*, 30:215–222, 1981.

[Han] HandshakeSolutions. www.handshakesolutions.com.

[HBB94] Henrik Hulgaard, Steven M. Burns, and Gaetano Borriello. Testing asynchronous circuits: A survey. Technical Report TR 94-03-06, Department of Computer Science and Engineering, University of Washington, Seattle, 1994.

[HC71] M.Y. Hsiao and D.K. Chia. Boolean difference for fault detection in asynchronous sequential machines. *IEEE Transactions on Computers*, 20(11):1356–1361, 1971.

[HS89] R. V. Hudli and S. C. Seth. Testability analysis of synchronous sequential circuits based on structural data. In *Proc. Meeting the Tests of Time. International Test Conference*, pages 364–372, 29–31 Aug. 1989.

[I.P94] C.Njinda I.Parulkar, M.A.Breuer. Extraction of a highlevel structural representation from circuit descriptions with applications toDFT/BIST. *31st Conference on Design Automation,*, pages 345–350, June 1994.

[IRR+01] A. Ivanov, S. Rafiq, M. Renovell, F. Azais, and Y. Bertrand. On the detectability of cmos floating gate transistor faults. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 20(1):116 –128, jan 2001.

[JA85] S. K. Jain and V. D. Agrawal. Modeling and test generation algorithms for MOS circuits. (5):426–433, May 1985.

[JPRS67] W. G. Bouricius J. P. Roth and P. R. Schneider. Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits. *IEEE Trans. On Electronic Computers*, Vol. EC-16, No. 10:567–579, Oct. 1967.

[JRBD94] D. E. Long K. L. McMillan J. R. Burch, E. M. Clarke and D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on CAD*, page 401 424, 1994.

[KA94] Prabhakar Kudva and Venkatesh Akella. Testing two-phase transition signalling based self-timed circuits in a synthesis environment. In *Proceedings of the 7th International Symposium on High-Level Synthesis*, pages 104–111. IEEE Computer Society Press, May 1994.

[Kar72] R.M. Karp. Reducibility among combinatorial problems. *in Complexity of Computer Computations (Plenum Press,New York)*, pages 85–103, 1972.

[KB95] Ajay Khoche and Erik Brunvand. A partial scan methodology for testing self-timed circuits. In *Proc. IEEE VLSI Test Symposium*, pages 283–289, 1995.

[KF91] M. Roncken R.Saeijs K.V.Berkel, J. Kessels and F.Schalij. The VLSI programming language Tangram and its translation into handshake circuits. In *In Proc European Conference on Design Automation*, pages 384–389, 1991.

[KKL+98] Michael Kishinevsky, Alex Kondratyev, Luciano Lavagno, Alex Saldanha, and Alexander Taubin. Partial-scan delay fault testing of asynchronous circuits. *IEEE Transactions on Computer-Aided Design*, 17(11):1184–1199, November 1998.

[KLSV91] Kurt Keutzer, Luciano Lavagno, and Alberto Sangiovanni-Vincentelli. Synthesis for testability techniques for asynchronous circuits. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 326–329. IEEE Computer Society Press, November 1991.

[Kop05] Koppad.D. Off-line testing of asynchronous circuits. *VLSI Design, International Conference on*, 0:730–735, 2005.

[KP01] Joep Kessels and Ad Peeters. The Tangram framework: Asynchronous circuits for low power. In *Proc. of Asia and South Pacific Design Automation Conference*, pages 255–260, February 2001.

[KSS02] Alex Kondratyev, Lief Sorensen, and Amy Streich. Testing of asynchronous designs by inappropriate means. synchronous approach. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 171–180, April 2002.

[LKL94] Luciano Lavagno, Michael Kishinevsky, and Antonio Lioy. Testing redundant asynchronous circuits by variable phase splitting. In *Proc. European Design Automation Conference (EURO-DAC)*, pages 328–333. IEEE Computer Society Press, September 1994.

[LM02] J.C.-M. Li and E.J. McCluskey. Diagnosis of sequence-dependent chips. In *VLSI Test Symposium, 2002. (VTS 2002). Proceedings 20th IEEE*, pages 187 – 192, 2002.

[LM05] James Chien-Mo Li and E.J. McCluskey. Diagnosis of resistive-open and stuck-open defects in digital cmos ics. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(11):1748 – 1759, nov. 2005.

[Lu95] Shih-Lien Lu. Implementation of micropipelines in enable/disable CMOS differential logic. *IEEE Transactions on VLSI Systems*, 3(2):338–341, June 1995.

[Luc76] C.L. Lucchesi. A minimax equality for directed graphs. *Doctoral Thesis, University of Waterloo,Ontario, Canada.*, 1976.

[LY78] C.L. Lucchesi and D.H. Younger. A minimax theorem for directed graphs. *J.London Math Soc.*, 217:699–374, 1978.

[MAF90] M .A .Breuer M. Abramovici and A.D .Friedman. *Digtal Systems Testing and Testable Design*. computer Science Press, 1990.

[Mal87] W. Maly. Realistic fault modeling for vlsi testing. In *Proc. 24th Conference on Design Automation*, pages 173–180, 28–1 June 1987.

[Mal93] S. Malik. Analysis of cyclic combinational circuits. In *Proc. IEEE/ACM International Conference on Computer-Aided Design ICCAD-93. Digest of Technical Papers*, pages 618–625, 7–11 Nov. 1993.

[MV00] M.L.Bushnell and V.D.Agarwal. *Essentials of Electronic Testing for Digital,Memory and Mixed-Signal VLSI Circuits,Series:Frontiers in electronic testing*. Springer Verlag, Boston, 2000.

[Mye01] Chris Myers. *Asynchronous Circuit Design*. John Wiley & Sons, 2001.

[Niv04] Gabriel Nivasch. Cycle detection using a stack. *Inf. Process. Lett, Elsevier North-Holland, Inc., Amsterdam, The Netherlands*, 90,3:135–140, 2004.

[NJC95] S. Nowick, N. Jha, and F.-C. Cheng. Synthesis of asynchronous circuits for stuck-at and robust path delay fault testability. In *Proc. International Conference on VLSI Design*, January 1995.

[Pag95] Sandeep Pagey. Fast functional testing of delay-insensitive circuits. In *Proc. of the Asian Test Symposium*, pages 375–381, 1995.

[Pet94] O. A. Petlin. Random testing of asynchronous VLSI circuits. Master's thesis, Department of Computer Science, University of Manchester, 1994.

[PF95a] O. A. Petlin and S. B. Furber. Designing C-elements for testability. Technical Report UMCS-95-10-2, Department of Computer Science, University of Manchester, 1995.

[PF95b] O. A. Petlin and S. B. Furber. Scan testing of asynchronous sequential circuits. In *Proc. of the Great Lakes Symposium on VLSI*, pages 224–229, March 1995.

[PFRG95] O. A. Petlin, S. B. Furber, A. M. Romankevich, and V. V. Groll. Designing asynchronous sequential circuits for random pattern testability. *IEE Proceedings, Computers and Digital Techniques*, 142(4), 1995.

[PKB95] Sandeep Pagey, Ajay Khoche, and Erik Brunvand. DFT for fast testing of self-timed control circuits. In *Proc. of the Asian Test Symposium*, pages 382–386, 1995.

[Put70] Gianfranco R. Putzolu. A heuristic algorithm for the testing of asynchronous circuits. *IEEE Transactions on Computers*, 20(6):639–647, June 1970.

[PVS92] S. Pagey, G. Venkatesh, and S. Sherlekar. Issues in fault modeling and testing of micropipelines. In *Proc. of the Asian Test Symposium*, Hiroshima, Japan, November 1992.

[RAV96] Marly Roncken, Emile Aarts, and Wim Verhaegh. Optimal scan for pipelined testing: An asynchronous foundation. In *Proc. International Test Conference*, pages 215–224, October 1996.

[RB96] Marly Roncken and Erik Bruls. Test quality of asynchronous circuits: A defect-oriented evaluation. In *Proc. International Test Conference*, pages 205–214, October 1996.

[RCPP97] Oriol Roig, Jordi Cortadella, Marco A. Peña, and Enric Pastor. Automatic generation of synchronous test patterns for asynchronous circuits. In *Proc. ACM/IEEE Design Automation Conference*, pages 620–625, June 1997.

[RDB08] H. Rahaman, D.K. Das, and B.B. Bhattacharya. An adaptive bist design for detecting multiple stuck-open faults in a cmos complex cell. *Instrumentation and Measurement, IEEE Transactions on*, 57(12):2838 –2845, dec. 2008.

[RM00] Rochit Rajsuman and Senior Member. Iddq testing for cmos vlsi. *Proceedings of the IEEE*, 88:544–566, 2000.

[Roi97] Oriol Roig. *Formal Verification and Testing of Asynchronous Circuits*. PhD thesis, Univsitat Politècnia de Catalunya, May 1997.

[Ron94] Marly Roncken. Partial scan test for asynchronous circuits illustrated on a DCC error corrector. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 247–256, November 1994.

[RT75] R.C. Read and R. E. Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5:237–252, 1975.

[SKR00] Roncken M. Stevens K. Chaudhuri P. P. Sur-Kolay, S. and R Roy. Fsimac: a fault simulator for asynchronous sequential circuits. *In Proceedings of the 9th Asian Test Symposium (December 04 - 06, 2000). ATS. IEEE Computer Society, Washington, DC.*, page 114, 2000.

[Sla61] P. Slater. Inconsistencies in a schedule of paired comparisons. *Biometrika*, 4:303–312, 1961.

[SM74] D. H. Sawin and G. K. Maki. Asynchronous sequential machines designed for fault detection. *IEEE Transactions on Computers*, C-23(3):239–249, March 1974.

[SM04a] F. Shi and Y. Makris. Spin-sim: Logic and fault simulation for speed-independent circuits. *In Proceedings of the international Test Conference on international Test Conference (October 26 - 28, 2004). ITC. IEEE Computer Society, Washington, DC*, 2004.

[SM04b] Feng Shi and Y. Makris. Spin-test: automatic test pattern generation for speed-independent circuits. In *Proc. ICCAD-2004 Computer Aided Design IEEE/ACM International Conference on*, pages 903–908, 7–11 Nov. 2004.

[SO08] Kewal K. Saluja Satoshi Ohtake. A systematic scan insertion technique for asynchronous on-chip interconnects. *Proceedings of the 1st International Workshop on the impact of low power design on test and reliability(LPonTR)*, 2008.

[SWF93] M.-D. Shieh, C.-L. Wey, and P. D. Fisher. Fault effects in asynchronous se-

quential logic circuits. *IEE Proceedings, Computers and Digital Techniques*, 140(6):327–332, November 1993.

[TF96] M. Teramoto and F. Fukazawa. Test-pattern generation for circuits with asynchronous signals based on scan. In *Proc. International Test Conference*, October 1996.

[THCR01] Charles E. Leiserson Thomas H. Cormen and Ronald L. Rivest. *Introduction to algorithms*. MIT Press, Cambridge, MA, USA, 2001.

[Tie] Tiempo. Tiemp inc.

[Uno03] Takeaki Uno. An output linear time algorithm for enumerating chordless cycles. *92th SIGAL of Information Processing Society Japan*, pages 47– 53, 2003.

[VCHS09] Julio Vazquez, Victor Champac, Chuck Hawkins, and Jaume Segura. Stuck-open fault leakage and testing in nanometer technologies. *VLSI Test Symposium, IEEE*, 0:315–320, 2009.

[WA08] S.Kakarla W.K.Al-Assadi. Design for test of asynchronous null convention logic (ncl) circuits. *International Test Conference*, pages 1–9, 2008.

[Wei72] Herbert Weinblatt. A new search algorithm for finding the simple cycles of a finite directed graph. *J. ACM, ACM Press,New York, NY, USA.*, 19-1:43–56, 1972.