# EXTRACTION BASED VERIFICATION METHOD FOR OFF THE SHELF INTEGRATED CIRCUITS

by

VIVEK NAGUBADI

Submitted in partial fulfillment of the requirements

For the degree of Master of Science

Thesis Advisor: Dr. Daniel G. Saab

Department of Electrical Engineering and Computer Science

CASE WESTERN RESERVE UNIVERSITY

August 2010

**CASE WESTERN RESERVE UNIVERSITY**

**SCHOOL OF GRADUATE STUDIES**

We hereby approve the thesis/dissertation of

Vivek Nagubadi

candidate for the Master of Science degree *.

(signed) Daniel G. Saab
(chair of the committee)

Francis "Frank" Merat

M. Cenk Cavusoglu

(date) 07-01-2010

*We also certify that written approval has been obtained for any proprietary material contained therein.

# Table of Contents

## List of Tables

## List of Figures

**Acknowledgments**

I take this opportunity to sincerely thank my advisor Prof. Daniel G. Saab for his constant encouragement and support in completing this work. His guidance and supervision have been invaluable and helped me from the initial level to the final level of my Master's program. Prof. Saab's wide knowledge and expertise has been essential to complete my thesis. It has been pleasure working with Prof. Saab.

I would like to sincerely express my gratitude to Prof. Francis "Frank" Merat and Prof. Cenk Cavusoglu for their support as members of my thesis defense committee.

Finally thanks to the each and every one in EECS department who have been instrumental in my growth. Lastly, I take this opportunity to sincerely thank the staff in Co-operative Education Department and International Student Services Offices for their services offered during my term at Case Western Reserve University.

Extraction Based Verification Method for off The Shelf Integrated Circuits

Abstract

by

VIVEK NAGUBADI

Off-the-shelf Integrated Circuits (ICs) are used in the design of many products. The IC is supposed to implement a set of available specifications describing the function of the IC. Users of off-the-shelf ICs need a simple and effective method to validate the specifications to insure that the IC implements exclusively the set of available specifications. In this thesis, we propose an approach to validate these specifications by a set of IC re-engineering experiments. The proposed approach is based on the construction of a high-level description of the packaged IC and on using the extracted description to validate the specifications. The approach uses the scan operations (available for manufacturing test of the IC) and the IC specification to disassemble the states/flip-flops and output functions of the packaged IC. Using the disassembled functions, a Register Transfer Level (RTL) model suitable for Computer-Aided Design manipulation is constructed. The disassembling is based on an ATPG scan experiment. Information on the scan chains is employed to construct the connectivity of the logic function. The connectivity is then used to discover the implemented logic. Using the proposed approach, we re-constructed over 90%of the system functions for an example IC. Scope for future work has been discussed at the end of the thesis work.

# Chapter 1

# Introduction to Reverse Engineering

Engineering can be classified into two sub-categories viz., Forward Engineering and Reverse Engineering. Forward Engineering is the discipline in which a set of specifications/requirements are realized into products and systems by applying technical, mathematical and scientific knowledge. Where as Reverse Engineering is a processes of back-tracing to the initial specifications or blueprint from a final product. Figure 1.1 illustrates a top view of both forward engineering and reverse engineering processes. With reverse engineering one has the ability to determine the internal working or implementation details of a product. Reverse Engineering has its origins in the analysis of hardware for military or commercial applications [1].

As the number of transistors on a chip are being doubled on a chip every two years according to the Moore's law, the complexity of the circuits has grown. Rapid technological advancement in the field of integrated circuits has made reverse engineering an area of interest for many. It has a broad spectrum of applications ranging from reverse engineering

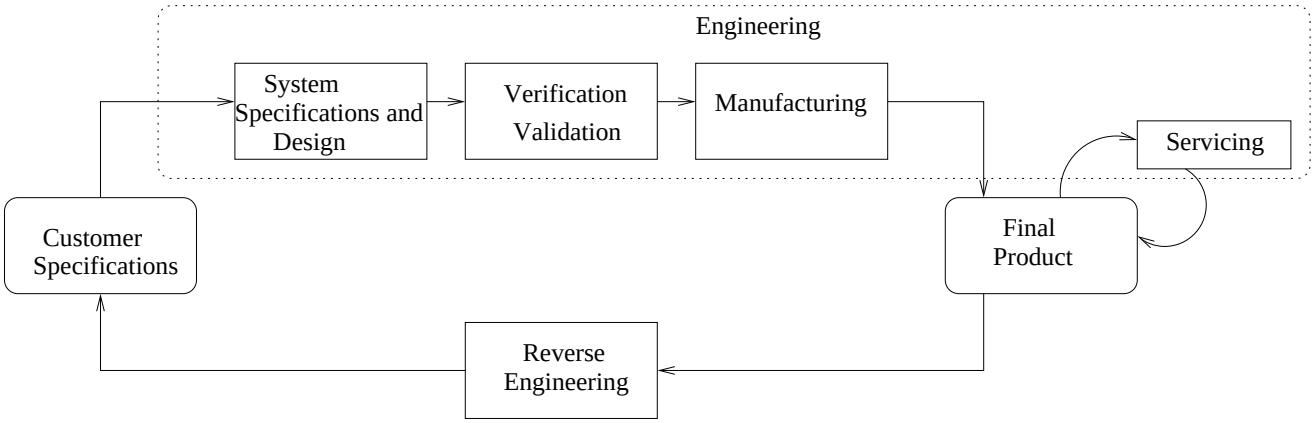


Figure 1.1: Engineering and Reverse Engineering Process Flow

obsolete products with lost documentation for the purpose of incorporating some of the existing features in the latest technology to analyzing competitor's products.

Software reverse engineering requires a combination of skills and a thorough understanding of computers and software development, but like most worthwhile subjects, the only real prerequisite is a strong curiosity and desire to learn. Software reverse engineering integrates several arts: code breaking, puzzle solving, programming, and logical analysis [6]. Reverse Engineering in semiconductors can be classified into several forms. Popular among the existing methods are product tear downs, system level analysis, circuit extraction [20]. The processes used to reengineer are often very difficult and time consuming. The amount of work involved in tracing back to the top-level specification depends on the type of method used.

## 1.1 Laws Governing Reverse Engineering

The legitimacy of reverse engineering products varies from country to country. In U.S. Reverse engineering of semiconductor devices is guarded under the Semiconductor Chip Protection Act (SCPA) of 1984. The SCPA is found in title 17, U.S. Code, sections 901-914 [2]. Prior to this act its not considered illegal to reproduce a competitors chip with identical layout. According to the SCPA law, reverse engineering is legal when done solely for the purpose of teaching, analyzing, or evaluating the concepts or techniques incorporated in the circuit, logic flow or organization of components used. The results obtained as a result of analysis should be made to be distributed. The law also states that without the consent of the actual owner the semiconductor chip cannot be reproduced for commercial purposes. The respective owners of the mask are required to file an application for registration of their mask with the U.S. Copyright office. The details related to the mask, such as pictorial information of IC layers and identifying material should be submitted along with the application.

In Europe, inspite of the popularity of reverse engineering concepts among engineers, the laws restrict the reverse engineering of hardware and makes reverse engineering of software legal only under very limited circumstances [3].

## 1.2 Applications of Reverse Engineering

Reverse Engineering has a wide range of applications depending upon the industry. Traditionally the most well-know and important application is in developing competing products. Below are some of the critical areas where semiconductor reverse engineering is applied.

- **Interoperability:**Interoperability is a technique of working together (inter-operate)

between diverse systems. Portability is the key for a system to operate in different environments. Examples include hardware and software interoperability.

- **Products with Lost Documentation:** Company's often want to incorporate part of the functionality of an obsolete product into its current technology. Besides this another important motivation for reverse engineering is, a product's documentation may have been lost and the person who designed it is no longer available. In these scenarios reverse-engineering obsolete integrated circuit can provide information about its functionality.

- **Analyze competitor's products:** In order to determine the pricing information, technical intelligence and functional details of the competitor's products, company's rely on reverse engineering. Product tear downs are generally employed to accomplish this task.

- **Copyright Violation:**Reverse Engineering is used to investigate products for copyright violation or patent infringement. The product under investigation is tear down into individual components and the obtained information is cross-verified against the information submitted while registering the copyright or patent.

- **Educational or Academic Purpose:** Reverse Engineering semiconductors is legal for the sake of educational or academic purpose. This is one of the several ways employed to study the functionality and operation of a product.

- **Security:** Computer security is one of the primary areas of concern now a days. Reverse Engineering is being used to develop malware to attack computers and also

by developers to find antidotes for the attacks. Also another area of application is to determine the level of security of the encryption algorithms.

# Chapter 2

# Background

## 2.1 Types of Reverse Engineering

Depending upon the area of interest, a wide variety of techniques exist to reverse engineer products. For example in software: decompilers, debuggers, system-monitoring tools, disassemblers are used to reverse engineer software products [6]. Similarly for mechanical products techniques such as rapid prototyping, scanning and converting the scanned data into a 3D model exists [15]. According to [20] the most commonly employed methods to Reverse Engineer Hardware are:

- Product Tear Down
- System Level Analysis
- Circuit Analysis
- Process Analysis

A combination of one or more of the above specified methods is generally employed. The remainder of this section explains in detail about these commonly employed techniques to Reverse Engineer integrated circuits.

- **Product Tear Down:** The goal of the product tear downs is to identify the components used in making the product under investigation. The information obtained from the product tear downs can be used to determine the system architecture, performance metrics, cost of the individual components used and in general pricing of the entire product including the manufacturing cost. Firstly, the product is systematically teared down into individual components and internal boards. In some instances optical and x-ray analysis are conducted on the components to determine the technology used and also the capacity of the individual components. Dissemination of the product into components is photographed and recorded so that others can use the information. The dis-assembly and analysis of a product has been common practice for major chip-makers and equipment manufacturers [20].

- **System Level Analysis:** Electronic systems primarily consist of hardware and software. System level analysis of these components can be done using reverse engineering or functional analysis. Reverse Engineering hardware is a hierarchial method, where the product is for tear-down into subsystems and the connections between them are noted. Next, the main board is delayered and the connections between layers are noted. For software, reverse engineering involves conversion of the machine code into human readable form. Functional analysis is similar to both hardware and software, where test cases are developed, stimulus is created for operating the system in its functional modes. The response is watched using signal generators, logic analyzers and oscillo-

scopes [20]. Similar to hardware and software, system-level analysis can be done on firmware, communications, tranducers etc.

- **Circuit Analysis:** According to [20] circuit extraction is back tracing of the integrated circuit back to circuit netlist it is intended to represent. Device extraction, interconnect extraction and parasitic device extraction are the three important processes involved in order to accomplish this technique. The packaging material is dissolved by applying an acidic solution without effecting the die. Delayering of the die obtained from the previous step is done by determining the chemical composition of the respective layers using scanning electron microscopes. Significant amount of distinction has to be made between the designed devices, which are created by the designer and the parasitic devices, which are inherent to the circuit layout. The extracted circuit can be used for logic simulation, timing analysis, power analysis, signal integrity and optimization.

- **Process Analysis:** If product tear down technique is used to identify the individual components of a final product, its with the use of process analysis techniques which helps determine the technology of the modules, how sub-system modules are manufactured and also to do failure analysis on the wafers. Process analysis for semiconductors is done using plan view imaging and cross-sectional analysis. While plan view imaging gives limited information, cross-sectional analysis gives more detailed information of the module. Transmission Electron Microscope (TEM) or Single Electron Microscope (SEM) are required to perform the cross-sectional analysis. In-order to determine the chemical composition of the module under investigation Energy Dispersive X-Ray Spectroscopy (EDS) is used. Secondary ion mass spectroscopy and Auger Analysis are also occasionally employed to find the chemical composition [20].

## 2.2 Related Prior Work

In the past few decades research on reverse engineering semiconductors has been an interest to many. Most of the techniques proposed so far either assumes that a gate-level netlist is readily available or access to the integrated circuit die/wafer is present or both. However for a packaged IC, access to the gate-level netlist and die/wafer may not be possible. In some scenarios there is limit on the inputs to the functions that can be realized for a circuit.

Scan architecture was developed to alleviate the cost of sequential ATPG and to reduce test requirement. Chip integrity and security was not addressed in the original scan design. With the recent trend of implementing these algorithms in hardware [22], [7] security is becoming an issue not only for cryptochips but also for general design [11], [19], [16], [14]. In fact, it was pointed out that scan chain can be used to attack Data Encryption Standard cryptochip, retrieve secrets keys , and compromise its security. As a result, techniques have been proposed to address this issue and make the scan chain more secure [9], [10], [12]. The remainder of this section gives a brief overview of the existing techniques.

- In [13] a reengineering technique is proposed for the legacy hardware and/or software systems which are functional. The entire procedure is divided into three phases, which consists of Legacy system understanding, Reengineering trade-off exploration and Detailed hardware/software design integration. The paper is based on the assumption that a structural transistor-level netlist of each of the circuit card assembly component can be extracted through etching and circuit geometry abstraction. The obtained netlist is then converted to the transistor-level netlist to a gate-level netlist, using a library-based approach that includes describing the gates transistor implementation.

Next search-based pattern-matching procedures are employed to extract the circuit's class and hence the key components in the legacy system are determined. In the second phase of reengineering trade-off exploration evaluation is done to determine the architectural changes required to upgrade the legacy system and also to determine the reengineered systems cost and quality. Virtual prototyping of the hardware components using clock-accurate and function-accurate models of the reengineered system is created to test the entire systems operation in the final phase. However for packaged ICs, the structural transistor-level netlist required for the first phase of Legacy system understanding is not readily and the process of etching described to obtain the structural transistor-level netlist may not be effective.

- Integrated Circuits are designed at multiple levels of abstraction which include behavioral level, register-transfer level (RTL), gate-level, transistor level. In [8] a technique has been proposed to construct the higher level of description for ISCAS-85 benchmark circuits. The gate-level netlists of the benchmark circuits are used to reverse engineer in order to determine the higher-lever structure. The technique starts with identifying the library modules such as multiplexers, decoders and adders from the gate-level netlist. Next repeated modules in the circuit are identified, if any. With the information obtained so far expected global structures such as signals or functions that use these modules are determined. The outputs of repeated modules are grouped into buses and any common names derived from the netlists are temporarily grouped together for further structural insights. The structural insight obtained is used to compute logic functions in symbolic or binary form. When no information can be derived from the gate-level netlists the technique demands the portion be represented as a

black box. Inorder to reverse engineer larger designs, the netlist is first converted to the schematic for convenience. Although this technique systematically recovers the circuits' hidden functional and structural information, it works on the fact that the gate-level netlist is readily available. However for packaged ICs obtaining gate-level netlist is a difficult task. Also the technique cannot realize logical functions of greater than five input signals.

- In [26] a technique for extracting integrated circuit design is disclosed. This method requires an optical means which includes a camera and a microscope to capture the sectional images of the integrated circuit that is positioned on a table. The camera is placed directly above the table and the table means can be controlled with the help of a computer or microprocessor to position the integrated circuit relative to the camera. Images of the IC are obtained section-by-section until all the layers of the die are photographed. There is a provision in the technique for video signal captured by the camera be sent to an image processor. An image processor generates an abstract representation which identifies the features such as size, type, relative location of transistors, width, length and relative location of all the metal interconnects from the captured images. Next, identified features are mapped to a library which contains raw pixel data for each of the reference library elements. The computer generates an abstract representation of the integrated circuit after all of the circuit cells on the die have been identified. This technique explores reverse engineering of an IC at die level, which prevents this method to be implemented on a packaged IC.

- In [24] discusses about the vulnerabilities of the scan based techniques. Although the work described in [24] is not directly related to the thesis, it gives a different perspective

on the scan chain design. The paper mentions about how scan chain techniques can be used to decipher secret keys stored in the Application Specific Integrated Circuit (ASIC) which implements cryptographic algorithms. Data Encryption algorithm is used to discover the structure of internal scan chains and breaking the round key is explained in detail. However our proposed technique does not need to read and decipher the information stored in the scan-chain register.

- In [5] proposed a technique to derive higher-level of description of an integrated circuit from a know structural-level description using an algorithmic process. The conceptual algorithm is based on transforming the circuit to successive higher level abstractions by identifying functional components in structural representations. This process is done in two phases: partitioning into candidate components and matching candidate components against a known set of library components. Matching the candidate components can be performed in a syntactic or semantic fashion. In syntactic fashion the identified structural component and the corresponding component in the library module should match exactly. Where as in the case of Semantic method the identified sub-circuit is matched based on the similar functionality within the library module. In order to reverse engineer larger circuits with sequential elements an iterative search is performed which includes additional techniques such as family matching, coarse partitioning, register identification, state machine identification, fine partitioning are employed. At every step the new sub-circuit identified, is added to the syntactic library, to search for additional instances of the new component. This research work is based on the following assumptions. Firstly, the algorithm assumes the availability of a netlist with no significant errors. Secondly, the motivation for the syntactic matching is restricted to

either verification or intellectual property analysis, which in turns assumes that either the design document or circuit functionality is readily available. Thirdly, it assumes the availability of a library of module descriptions for each level of abstraction. For a packaged IC or an IC with lost documentation the above assumptions may not be valid.

- In [25] formal verifications methods are suggested to be incorporated during both engineering and reengineering processes. VDELE (VHDL Design Environment for Legacy Electronics) is simulation technique used to deliver functional clones which duplicate the behavior of legacy hardware system. According to [25], VDELE technique alone can't guarantee the exact functional equivalence and with the incorporation of formal methods with VDELE can guarantee the exact functionality. This model depends on the existing manufacturing specification data sheet of the obsolete integrated circuit. It may not be possible to obtain the specification sheet for products with lost documentation.

## 2.3 Design for Testability

Over the past few decades integrated circuit design has been driven by rapidly increasing performance and functional density at reduced cost. This advancement in digital technology posed new challenges to the capabilities of existing test equipment. A combinational circuit with N inputs needs [2 power N] test vectors to test the circuit exhaustively and completely. Similarly, a sequential circuit with N inputs and M registers needs [2 power N+M] test vectors in order to validate this circuit [23]. This problem was addressed by incorporating features

to test in the original design itself. Controllability and Observability of nodes became a key in designing of integrated circuits.

- Controllability of a node with in an integrated circuits is defined as the flexibility in setting the node to 0 or 1
- Observability of a node with in an integrated circuit is defined as the ability to observe the state of the selected node at the output pins.

Design For Testability (DFT) is one of the manufacturing test principles which adds testability features to the integrated circuit design. If designed properly using these techniques they can provide more controllability and Observability of nodes in a circuit. Ad hoc testing, Scan-based testing, Built-In-Self-Test (BIST) are the three main approaches for Design for Testability.

### 2.3.1 Scan Based Design:

Scan based techniques are very effective design methods that makes possible the testing of complex VLSI chips . The registers generally operate in two modes of operation: scan mode and normal mode. The scan mechanism requires additional input/output pins and circuitry which provides a mechanism for the circuit to switch between scan mode and normal mode.

During the test mode, internal flip-flops are chained into a shift register. Figure 2.1 shows a sequential scan circuit. The flip-flops can be configured into a shift-register, with scanin/scanout as Input/Output, when primary input mode is asserted; otherwise the flip-flops load their normal inputs (functional mode). Using the scanin pin a test vector is scanned into the flip-flop shift-registers by asserting the mode signal and are clocked using

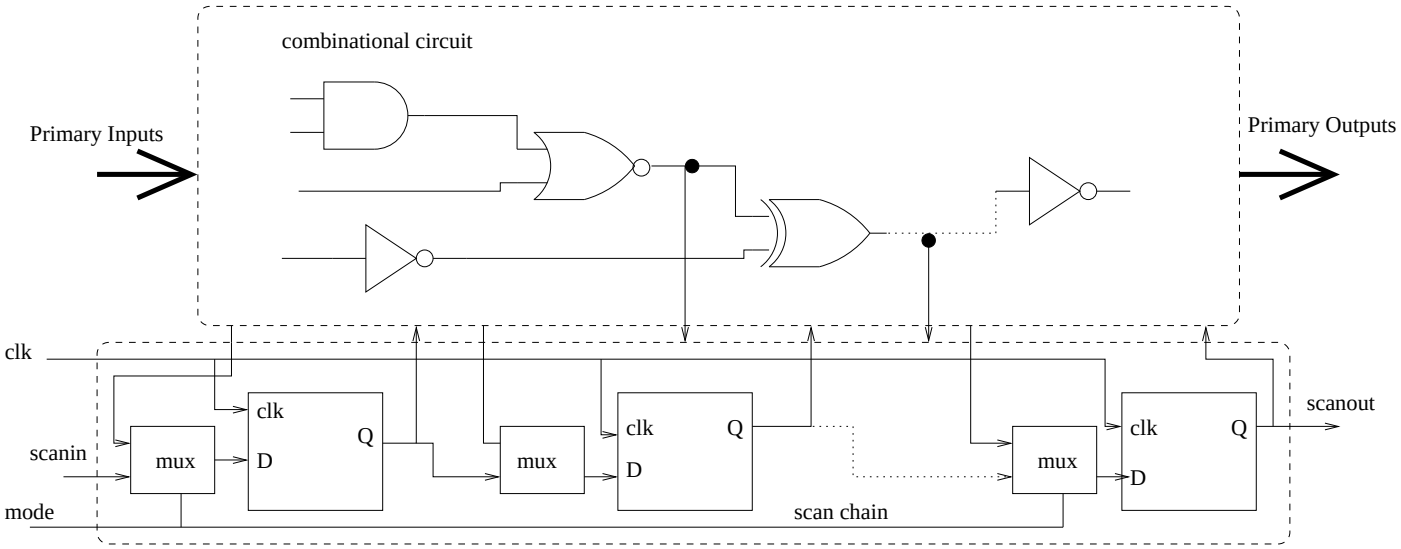


Figure 2.1: Example of a Scan Based Circuit

the system clock. When a test vector is fully scanned, the flip-flops are configured to be in normal mode for one clock cycle. At the end of the normal clock cycle the flip-flops load the normal system response. At this point, the flip-flops are again configured into a shift register to scan-out the stored normal response, concurrently scanning in the next test vectors. This process is repeated for all test vectors. In a scan based design, combinational Automatic Test Pattern Generation (ATPG) tools are used to generate tests for the combinational part of the circuit eliminating the need for the more expensive sequential Automatic Test Pattern Generation tools (ATPG).

Because of the high degree of controllability and observability of the nodes this technique provides, Automatic Test Pattern Generation (ATPG) tools provide high-fault coverage with reasonable memory and CPU requirements. ATPG techniques can be used for the combinational blocks and can easily be tested with the help of scan chain.

## 2.4 Contribution of this thesis:

In this thesis, examination of a packaged integrated circuit implantation of a sequential system is done and a technique to reverse-engineer the implemented design using scan functions is proposed. The approach is different from the previous techniques in that we compute a netlist for a packaged integrated circuit and do not need to read and decipher any specific [24] information stored in the register file. Unlike previous techniques, this approach targets packaged integrated circuit designs where the design netlist is not available. The thesis work employs the scan chain and available integrated circuit specification to construct the circuit connectivity. The gained connectivity knowledge is used with the scan chain to discover the logic of the connected function. Also, if the integrated circuit netlist is readily available we proposed an ambiguity analysis technique to map the flip-flops present in the gate-level netlist and the packaged IC.

# Chapter 3

# Modeling Packaged Integrated Circuit

After the integrated circuit is verified for its functionality, packaging is the final step in the semiconductor device fabrication. Currently, there are several existing packaging techniques. Depending upon the type of technique used, the yield and efficiency of electronic components are determined. The package provides mechanical and environmental protection for the chip and enables electrical connections to be made from the chip to other electronic components [18]. The remainder of this chapter presents a detailed description of the simulation technique developed to model packaged integrated circuits.

The goal of modeling a packaged IC is to replicate the exact representation of a circuit for simulation purposes. Our model primarily involves the following steps for complete representation and testing of a circuit under consideration.

- Node Creation
- Sequencing
- Simulation

## 3.1 Node Creation:

Figure 3.1 shows a scan based circuit. The goal of this step is to create a node for each and every unique gate in the circuit and determine the fanin, fanout and gatetype of each gate. Figure 3.3 shows a flowchart used to create nodes for each and every gate. The parser is a combination of parser and lexical analyzer whose role is to break the circuit into individual tokens. These tokens are sent to the search routine which searches the list and creates a node if there isn't any node for the gate. If a node already exists, then it calculates the fanin to that particular node. The fanin for primary inputs gates is considered to be 0.

At this point, we already have the information of the inputs of each and every gate. This information is used to calculate the output. Consider the circuit shown in Fig 3.4. The inputs to the Gate $Z$ are Gate $X$ and Gate $Y$. Which implies Gate $Z$ is a fanout to both Gate $X$ and Gate $Y$. From previous step we have calculated that fanin to Gate $Z$ has Gate $X$ and Gate $Y$. Calculation of fanout to each gate involves, traversing each and every node's fanin array and updating the fanout of the corresponding gate.

Figure 3.5 and 3.6 represents the algorithms to calculate the fanout of every gate. For primary outputs, fanout value is 0. Each gate at the end of this step is represented as node in Figure 3.2.

## 3.2 Sequencing:

Next step in modeling the circuit, invovles dividing it into subsequent levels. Figure 3.7 shows flowchart of the algorithm to levelize the circuit. All the primary input gates are considered to be at Level 0. Levelizing of further gates invovles, verifying if levels of the

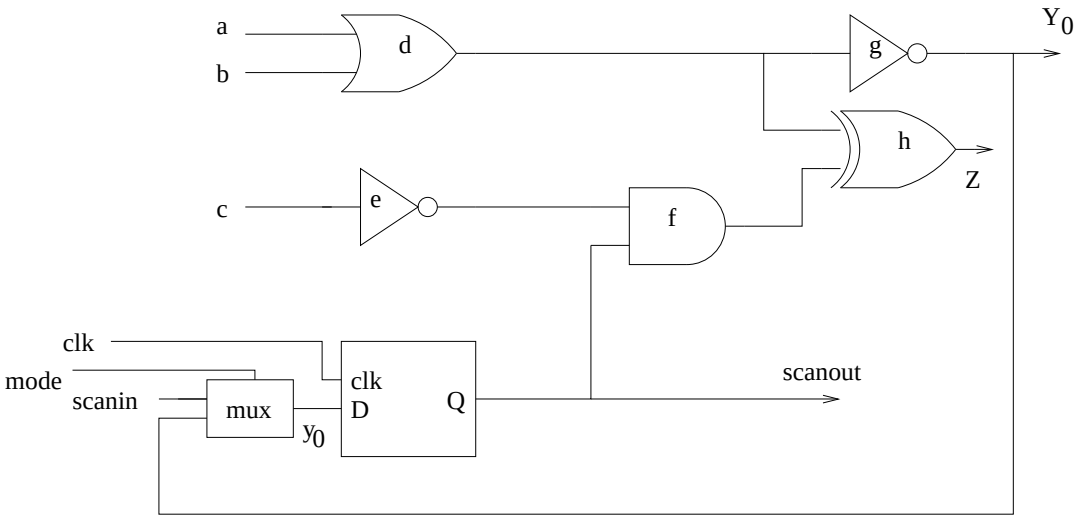


Figure 3.1: Sample Circuit

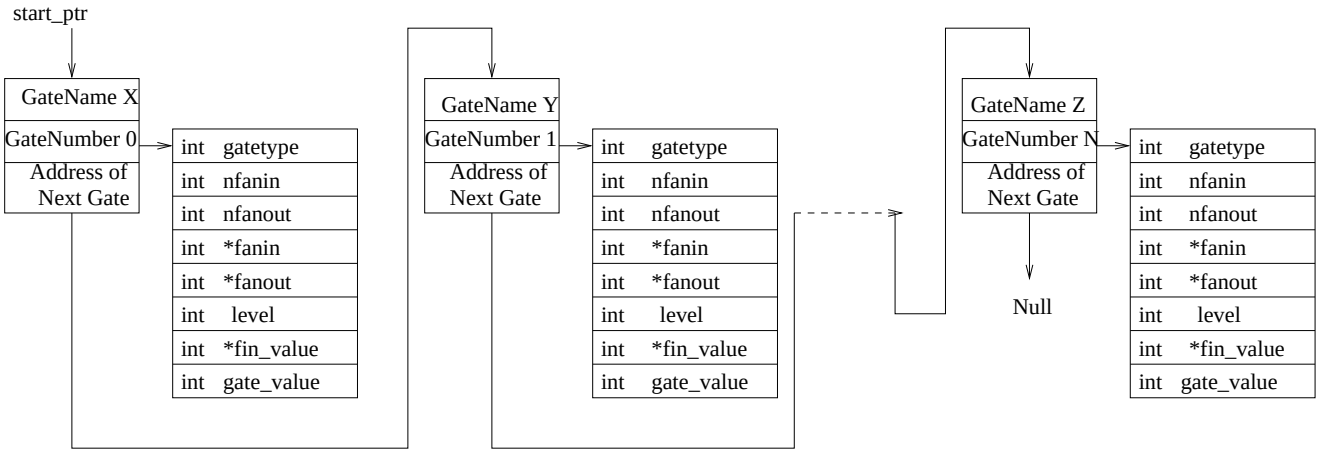


Figure 3.2: Gate Structure

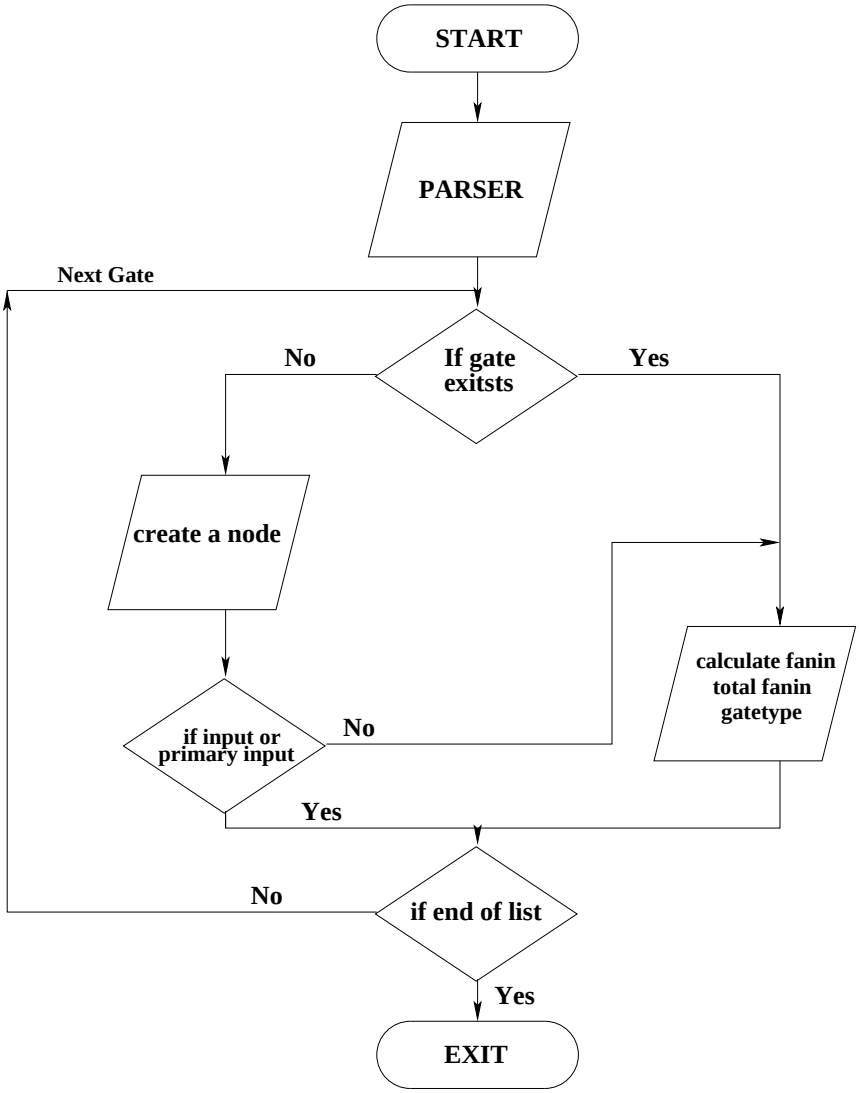


Figure 3.3: Flowchart to create a gate

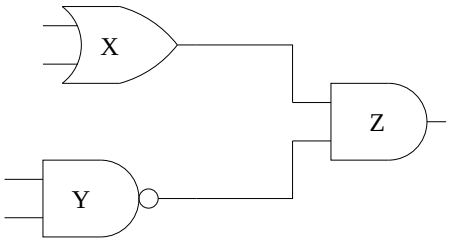


Figure 3.4: Illustration of fanout

```
nfanout(){
    for(int i=0;i<total_gates;i++){
        int total_fanin=ckt_gate[i].nfanin;
        for(int j=0;j<total_fanin;j++){
            int number=ckt_gate[i].fanin[j];
            ckt_gate[number].nfanout++;
        }
    }
}
```

Figure 3.5: Algorithm to calculate number of fanout for each gate

```
fanout(){
  int temp;
  for(int i=0;i<total_gates;i++){
      int total_fanout=ckt_gate[i].nfanout;
      ckt_gate[i].fanout=(int *)malloc(total_fanout)*sizeof(int));
      for(int j=0;j<total_fanout;j++){
          temp=ckt_gate[i].fanin[j];
          ckt_gate[temp].fanout[ckt_gate[temp].cout]=i;
          ckt_gate[temp].count++;
      }
  }
}
```

Figure 3.6: Algorithm to calculate fanout

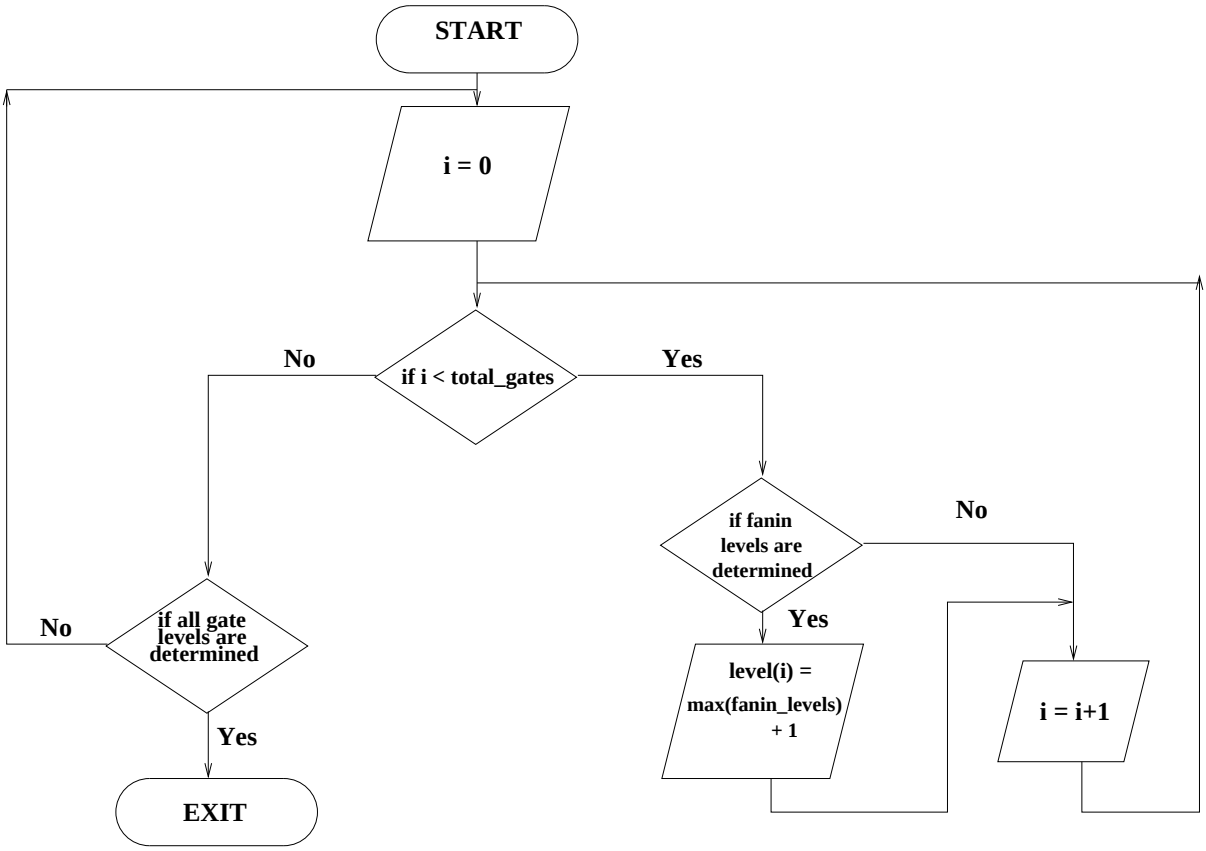


Figure 3.7: Flowchart to calculate Levels

fanin gates are determined. If they are, then the level of the gate under consideration will be (maximimum_level(fanin gates) + 1). This is an iterative approach and program will exit when all the individual gates present in the circuit are determined.

Consider the circuit shown in figure 3.1. Primary inputs $a$, $b$ and $c$ are considered to be at Level 0. Next the algorithm traverses the gatestructure 3.2 and search for gates which exclusively depend on primary inputs. They are labelled as Level 1. In the figure 3.1 $d$, $e$ depend on primary inputs. Similarly levels are calculated for every gate in the circuit.

## 3.3 Simulation:

Simulation is the final step in modeling the circuit, used to validate the circuit functionality as a whole. This step involves, generating random test vectors to apply as the input. The output observed at the primary outputs and next state elements for a particular test vector is cross-verified.

# Chapter 4

# Technique

Consider the sequential circuit with $i$ primary input lines, $o$ primary output lines and $n$ storage elements. The output of each of the $n$ storage elements is considered as a present-state line and the input as a next-state line. The clocking scheme is assumed to be known (in our case it is a single clock), and the value of the next-state line of a storage element in the previous time frame becomes the present-state line value of that element in the current time frame. The symbol $y_i$ is used to indicate the present-state variable corresponding to the storage element $i$ and $Y_i$ to indicate its next-state variable. The symbol $Z_i$ is used to indicate the primary output variable corresponding the $i$th circuit output variable.

4.1 illustrates the general data flow and key steps of the approach. The object of our system is to determine if the manufactured IC implements its intended specifications. To accomplish this, our re-engineering approach discovers the circuit connectivity by generating and using a set of tests that detects specific input/output dependency, existing in the specification, and on using the test to assert that the discovered dependencies are implemented

in the manufactured circuit. The discovered/disassembled dependencies are combined to form an interconnected circuit. This is show in the boxed labeled State Identification and Interconnect Discovery. The two steps identify the number of states (flip-flops), generate the variables declaration, and create the interconnection of the declared variables using function instantiations. Algorithms that identify variable similarity are applied to identify buses and registers. Note that any design error that causes connectivity changes in the specification can be detected at this point. In addition, the interconnected functions are partially discovered. To disassemble the functions implemented by the IC, the discovered structure, the scan and the specifications are used to compute the IC exhibited logic. The IC exhibited logic is minimized and stored symbolically or in a truth table. During function discovery, we use the specification to identify control signals, data-path signals, buses and registers. This involves the derivation of formal verification properties from the specifications, verifying the properties using the IC and the scan functions, and assembling the exhibited IC functions. As a result of the verification, functions associated with interconnect are discovered. In fact, a complete IC verification implies complete function [17].

## 4.1 State Identification:

The scan chain consists of a shift register where each flip-flop corresponds to a state variable. The flip-flop can store a logic 0 and a logic 1 and the combination stored in the scan chain can be shifted and observed, one bit during each clock cycle at the scan out. To compute the number of flip-flops are in the chain, WE shift a logic 1 followed by a large number n of 0s (n is larger than the number of flip-flops in the chain).

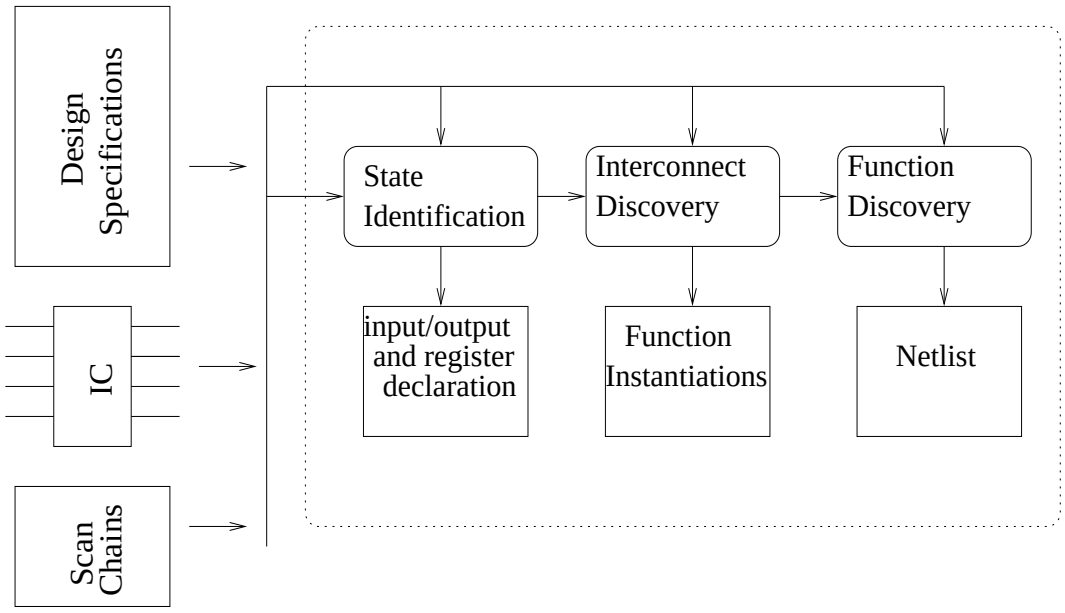


Figure 4.1: General Approach

Concurrently while shifting into the scan, we observe the scan out and record the clock cycle when the last 1 is observed at the scanout line. Since we are shifting one 1 followed by n zeros into the chain, the clock cycle K in which the last 1 appears indicates the size of the scan chain. In fact, the actual size is K - 1. Once the number of flip-flops is determined, each flip-flop is given a identification number between 1 and K - 1. These numbers are used to identify variables corresponding to present-states $y_i$s and next-states $Y_i$s.

### 4.1.1 Illustration of State Identification:

Consider the circuit show in 4.2 and assume the initial state of flip-flops ($y_0$, $y_1$, $y_2$) is (101) as shown in the table 4.1 below. A sequence of 000001 is serially shifted into the scan one

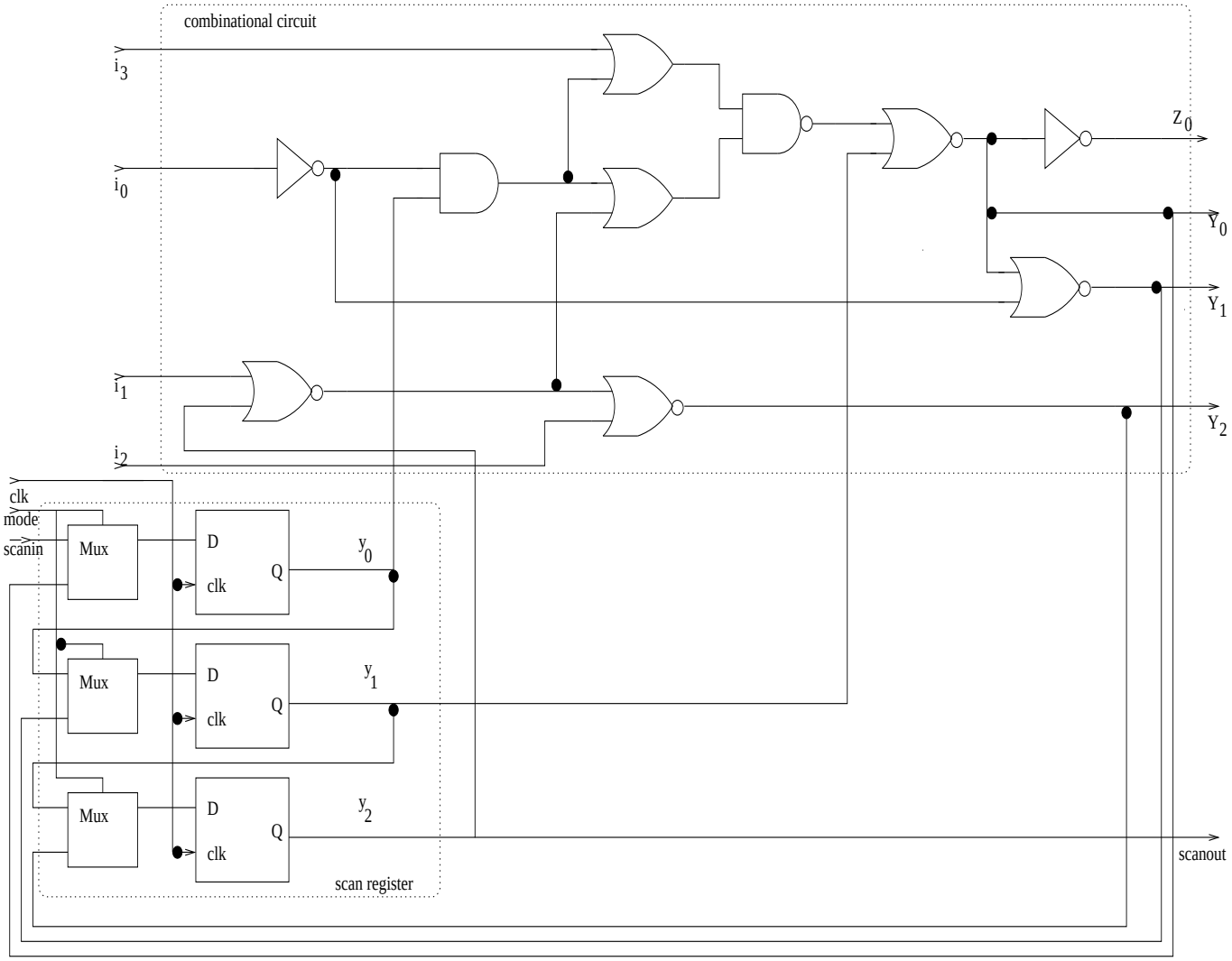


Figure 4.2: Example Circuit

Table 4.1: Demonstration of State Identification

| scanin | $y_0$ | $y_1$ | $y_2$ | scanout | clockcycle |
|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 2 |
| 0 | 0 | 0 | 1 | 1 | 3 |
| 0 | 0 | 0 | 0 | 1 | 4 |
| 0 | 0 | 0 | 0 | 0 | 5 |
| 0 | 0 | 0 | 0 | 0 | 6 |

bit/clock. The sequence observed at the scanout at each clock is 1101100. During the 4th clock cycle, the last logic 1 is observed at the scanout line. This indicates that the length of the scan chain for this circuit is 3 flip-flops.

## 4.2 Interconnect Generation:

The extracted circuit is to be described by a set of Boolean equations. For each of the output and next-state lines, we extract a Boolean description in terms of the primary inputs and present-state lines. To compute the Boolean expression for each output and next-state lines, we divide the task into two steps, the first is the computation of dependency matrix (show in 4.2) followed by function generation. The dependency matrix provides the global interconnection of the circuit and the Boolean function reveals the behavior of the circuit. The dependency can be computed either by deriving an expression for an output Z and literals appearing in the expression are included in the dependency list of Z. This is not possible in this case, since we have only access to the circuit input/output.

Definition 1: A function $f$ depends on variable $y$ if there exists an input combination such that $f$ is equal to $y(y')$

Consider the circuit shown in 4.2 and the input combination

$(i_0, i_1, y_2) = (1, 0, 0)$

The value of the output $Z_0$ for this input combination is

$Z_0 = i_3$'

Thus $Z_0$ depends on $i_3$. Therefore, applying a transition at $i_3$ causes a transition at $Z_0$ revealing the dependency. To compute the dependency matrix, WE need to find for each dependency an input combination that reveals that dependency at a primary output or at next-state lines.

Algorithm 1 shows the dependency generation procedure. The algorithm is based on generating random test patterns Test_pattern and performing input/output sensitivity analysis. The test patterns are generated using the specification. These tests target the sensitivity of the specification to primary inputs. This helps our analysis and increases the chances of discovering input/output/register dependencies. The analysis consists of complementing input bits in Test_pattern and monitoring changes, caused by this complementation, on the outputs and next state functions. In the case of a change in one of the outputs or next state $F_i$, the variable corresponds to the complemented bit is in the support of the function $F_s$. The algorithm increases the search time for a given Test_pattern flipping (reward) when a dependency is found. Otherwise, the search is aborted after a fixed number of bit inversions.

The complete procedure is shown in algorithm 1. The input to this algorithm is the set of K inputs $I_1$, $I_2$, . . . ., $I_n$, $I_{n+1}$, $I_{n+2}$, . . . ., $I_{n+s}$. The first $n$ inputs correspond to the primary circuit inputs and the next SSS corresponds to the present state lines. The algorithm generates the dependency for the $N$ functions $F_1$, $F_2$, . . . ., $F_o$, $F_{o+1}$, $F_{o+2}$, . . .

., $F_{o+s}$. The first $o$ corresponds to the primary circuit outputs and the next $s$ corresponds to the next state lines. The computed dependency for each function $F_i$ is saved in D($F_i$). The procedure selects the first of the four random inputs, from a test generated from the circuit specifications, that causes a change in at least 10% of the function $F_i$'s. A *gain* indicator is set to TRUE and *Ggain* (global gain) to FALSE. The *gain* and *Ggain* are used to reward good search. In the inner loop, the sate of the output functions are saved in Fs_state, the gain is set to FALSE, and bits in Test_pattern are completed one at a time. For each complemented bit operation, the output state response is recorded in New_Fs_state. The New_Fs_state and the saved previous output states Fs_state are compared for changes. For each changed output $F_n$, the corresponding complemented input $I_k$ that caused $F_n$ to change is added to the dependency set D($F_n$). Every time a dependency is found, the *gain* is set to TRUE to reward the search. After every complementation, implication, comparison and state update are repeated once for every input variable. If during a single iteration no dependency is found, then the *gain* search is terminated and a new random Test_pattern is generated. The random Test_pattern generation is repeated Iter_Limit times and is rewarded if any gain is recorded in the previous iteration. For our experiment, we set the Iter_Limit to 1000.

### 4.2.1 Illustration of Interconnect Generation:

In the previous section, we have calculated number of flipflops present in the packaged IC using State Identification. To demonstrate the Interconnect Generation, consider the Figure 4.2 shown. It represents ISCAS '89 sequential s27 circuit. From the specificaiton we know the input/output configuration of the IC and we have calculated the number of flip-flops

present in the scan chain. According to the algorithm 1 set of NI inputs are $i_0$, $i_1$, $i_2$, $i_3$, $y_0$, $y_1$, $y_2$ and set of NF functions are $Z_0$, $Y_0$, $Y_1$, $Y_2$.

Table 4.2 contains the test vector applied and the calculation of dependency. First column shows the test vector ($i_0 i_1 i_2 i_3 y_0 y_1 y_2$) randomly generated, followed by the flipped variable of the test vector. Column III shows the flipped test vector and columns IV and V shows the previous output and current output generated when the flipped test vector is applied. Dependency is calculated by comparing the previous_output and current output.

In Figure 4.2, for example $Y_0 = y_1{}' i_3 i_0 y_0 + y_1{}' i_1{}' y_2{}'$;

From the above expression, we can say that $Y_0$ depends on $y_2$, $y_1$, $y_0$, $i_3$, $i_2$, $i_2$, $i_0$. Knowing the dependency of primary outputs and next state lines is a crucial step in re-engineering the design using the scan chain and associated operations. Thus for every circuit, we need to compute the dependency matrix similar to the dependency matrix show in table 4.3. In this matrix, a"1" is used in the dependency matrix to indicate that a dependency exists. Next-state function of flip-flop $Y_2$ depends only on its present state and input lines $i_1$ and $_2$; hence, this flip-flop is independent of the other two flip-flops.

Dependency can be classified into two types viz., structural and functional. The dependency calculated using the above algorithm **??** generates functional dependency. In Figure 4.2, flip-flop $Y_1$ is assumed to be dependent on $y_0$. But if its logic equations are computed, however it is found to be independent of $y_0$:

$Y_1 = i_3{}' i_0 + i_1 i_0 + y_2 i_0 + y_1\ i_0$;

$Y_1{}' = i_0{}' + y_2{}' y_1{}' i_3 i_1{}'$;

Structural dependency for 4.2 is given in table 4.4. Note that strucutal dependency may contain superfluous variables.

Table 4.2: Dependency Table

| Test_Vector $i_0i_1i_2i_3y_0y_1y_2$ | FlipVariable | FlipVector $i_0i_1i_2i_3y_0y_1y_2$ | Prev_Output $Z_0Y_0Y_1Y_2$ | Output $Z_0Y_0Y_1Y_2$ | Dependency |
|---|---|---|---|---|---|
| 1110011 | N/A | N/A | N/A | 1100 | N/A |
| | $i_0$ | 0110011 | 1100 | 0010 | $Z_0(i_0)$, $Y_0(i_0)$, $Y_1(i_0)$ |
| | $i_1$ | 0010011 | 0010 | 0010 | N/A |
| | $i_2$ | 0000011 | 0010 | 0011 | $Y_2(i_2)$ |
| | $i_3$ | 0001011 | 0011 | 0011 | N/A |
| | $y_0$ | 0001111 | 0011 | 1001 | $Z_0(y_0)$, $Y_1(y_0)$ |
| | $y_1$ | 0001101 | 1001 | 1001 | N/A |
| | $y_2$ | 0001100 | 1001 | 1000 | $Y_2(y_2)$ |
| 1111111 | N/A | N/A | N/A | 1100 | N/A |
| | $i_0$ | 0111111 | 1100 | 1000 | $Y_0(i_0)$ |
| | $i_1$ | 0011111 | 1000 | 1000 | N/A |
| | $i_2$ | 0001111 | 1000 | 1001 | $Y_2(i_2)$ |
| | $i_3$ | 0000111 | 1001 | 1001 | N/A |
| | $y_0$ | 0000011 | 1001 | 0011 | $Z_0(y_0)$, $Y_1(y_0)$ |
| | $y_1$ | 0000001 | 0011 | 1001 | $Z_0(y_1)$, $Y_1(y_1)$ |
| | $y_2$ | 0000000 | 1001 | 1000 | $Y_2(y_2)$ |
| 1100111 | N/A | N/A | N/A | 1101 | N/A |
| | $i_0$ | 0100111 | 1101 | 1001 | $Y_1(i_0)$ |
| | $i_1$ | 0000111 | 1001 | 1001 | N/A |
| | $i_2$ | 0010111 | 1001 | 1000 | $Y_2(i_2)$ |
| | $i_3$ | 0011111 | 1000 | 1000 | N/A |
| | $y_0$ | 0011011 | 1000 | 0010 | $Z_0(y_0)$, $Y_1(y_0)$ |
| | $y_1$ | 0011001 | 0010 | 1000 | $Z_0(y_1)$, $Y_1(y_1)$ |
| | $y_2$ | 0011000 | 1000 | 0010 | $Z_0(y_2)$, $Y_1(y_2)$ |

```
module top(Z0, clk, i0, i1, i2, i3);
input clk, i0, i1, i2, i3;
output Z0;

    dff XG1(y0,Y0, clk);
    dff XG2(y1, Y1, clk);
    dff XG3(y2, Y2, clk);
    fy0 XG4 (Y0, y0, y1, y2, i0, i1, i3);
    fy1 XG5 (Y1, y0, y1, y2, i0, i1, i3);
    fy2 XG6 (Y2, y2, i2, i1);
    fZ0 XG7 (Z0, y0, y1, y2, i0, i1, i3);
endmodule
```

Figure 4.3: State and Interconnect Generation

Table 4.3: Functional Dependency

| | $y_2$ | $y_1$ | $y_0$ | $i_3$ | $i_2$ | $i_1$ | $i_0$ |
|---|---|---|---|---|---|---|---|
| $Y_0$ | 1 | 1 | 1 | 1 | . | 1 | 1 |
| $Y_1$ | 1 | 1 | . | 1 | . | 1 | 1 |
| $Y_2$ | 1 | . | . | . | 1 | 1 | . |
| $Z_0$ | 1 | 1 | 1 | 1 | . | 1 | 1 |

In Figure 4.3 the State Identification step generates the first seven lines in the extracted model. Interconnect generation step discovers the circuit interconnections using the specification, the scan, the manufacturing test, the IC and a sensitivity experiment. This is represented as the last four lines in the 4.3.

## 4.3 Function Discovery:

At this point, we have a set of dependent inputs for every function. Generating the function is the process of enumerating the input combinations of the function dependent variables,

**Algorithm 1** Dependency generation algorithm

```
Procedure_dependency(){
set of NI inputs: {I_1, I_2, , I_n, I_{n+1}, I_{n+2}, .., I_{n+s}}
set of NF functions: {F_1, F_2, . . . ., F_o, F_{o+1}, F_{o+2}, . . . ., F_{o+s}}
D(F_i): set of inputs supporting function F_i
set all D(F_i) = empty
iter = 0;
Test_pattern = Generate_random_input();
while (iter ≤ Iter_limit) do
  i = 0;
  repeat
    Test_pattern = Generate_random_input();
    New_Fs_state = apply_input(Test_pattern);
    i = i + 1;
  until (NumberofFschangingstate ≥ 10%ofNFandi ≤ 4);
  gain = TRUE;
  Ggain = FALSE;
  while (gain ) do
    gain = FALSE;
    Fs_state = New_Fs_state;
    for (k = 0 ; k ≤ NI; k + +) do
      flipbit(i_k, Test_pattern);
      New_Fs_state = imply(Test_pattern);
      diff = Fs_state XOR New_Fs_state;
      gain = diff != 0 : TRUE ? gain;
      if ( bit position n in diff == 1) then
        D(F_n) = D(F_n) U {I_k};
        Fs_state = New_Fs_state;
      end if
    end for
    if (gain) then
      Ggain = TRUE;
    end if
  end while
  iter = Ggain : 0 ? iter+1;
end while
}
```

Table 4.4: Structural Dependency

| | $y_2$ | $y_1$ | $y_0$ | $i_3$ | $i_2$ | $i_1$ | $i_0$ |
|---|---|---|---|---|---|---|---|
| $Y_0$ | 1 | 1 | 1 | 1 | . | 1 | 1 |
| $Y_1$ | 1 | 1 | 1 | 1 | . | 1 | 1 |
| $Y_2$ | 1 | . | . | . | 1 | 1 | . |
| $Z_0$ | 1 | 1 | 1 | 1 | . | 1 | 1 |

applying each input combination to the circuit, and computing the logical behavior fo the output functions. This process is very expensive and can only be performed on functions with limited number of inputs. In our case, we generated logical behavior for circuit which has functions with less than 32 inputs. To speed up this process, we employ concurrent enumeration of multiple functions with overlapping inputs. This reduces the total number of input combinations to be applied.

### 4.3.1 Illustration of Function Discovery:

At this point, we have all the interconnection of the circuit. In the next step, the functions are discovered. In the next step, the functions are discovered. An example of a discovered function is shown in next page. Functions are compared for similarity which are used to identify registers and to minimize the extracted model. In this case, we find that $Z_0$ and $Y_0$ are complements. This causes the algorithm to change the function for line

f($Z_0$) to f$y_0$ XG7 ($Z_0$', $y_0$, $y_1$, $y_2$, $i_0$, $i_1$, $i_3$);

not XG8 ($Z_0$, $Z_0$');

```
module fy2 (out, y2, i2, i1);
output out;
input y2, i2, i1;
    assign out = y2 & ~i2 & i0 | i1 & i2 & ~i0    ;
endmodule
```

Figure 4.4: Discovered Function

# Chapter 5

# Results

The procedure outlined in Chapter 4 was implemented in a computer program. The input data to the program is in the form of a bench file. We evaluated the procedure using ISCAS'89 benchmark circuits [4] and an implementation of an AES encryption chip [21].

To explain the results shown above, we define the following.

- The total number of outputs $Z_i$ and next state $Y_i$ functions in the circuit is defined by $T_{fun}$. Note that $Y_i$ is computer by this approach. In our experiment, the approach discovered all the states using the shifting experiment and monitoring experiment described in the previous section.

- The total number of discovered functions in the circuit is defined by $D_{fun}$. A function which all its inputs are known is a discovered function. In our experiment, we use the circuit netlist to compute the structural dependencies for each function. We compare the results from our approach to the structural dependencies of each function.

- The function coverage is defined as $F_{cov} = D_{fun}/T_{fun}$ X 100.

- The total number of primary inputs $I_i$ and present states $y_i$ in the circuit is given by $T_i$. For the circuit shown in Figure 1, $I_i$, $y_i$ and $T_i$ are equal to 4, 3 and 7, respectively.

- Each function in the circuit is structurally dependent on K inputs. The total number of structurally dependent inputs for all $T_fun$ is given by $T_{dep}$. For the circuit show in Figure 1 $T_{dep}$ is equal to 27.

- The total number of discovered dependencies for all $T_{fun}$ is defined by $D_{dep}$. Note, that structural dependencies may contain some superfluous variables while discovered ones do not.

- The dependency coverage is defined as $DP_{cov} = D_{dep}/T_{dep}$ X 100. Because of the superfluous terms in the structural dependency, the $F_{cov}$ may not be 100.

- $N_{tries}$ is the total number of test vectors applied to the circuit using the algorithm shown in Figure 3

As mentioned above, the results of this experiment are show in tables 5.1, 5.2, 5.3. Table 5.1 shows the state identification results. Each row in this table shows the number of flip-flops (FF) in the circuit, the number of inputs (In), the number of outputs (Out), the number of shifts applied ( Shifts), and the number of state variables identified ($y_i$). As a result of this procedure, the position of each identified state variable in the scan register is revealed. WE performed 2000 shifts for these circuits. In general, WE may want to perform more shifts to reduce uncertainties in the number and position of the state variables.

Table 5.2 shows results of the dependency computation procedure. The first column in that table gives the circuit name, the second column gives the number of gates, the

Table 5.1: State Identification

| Circuit | Flip-Flops | Inputs | Outputs | Number of Shifts | Present State($y_i$) |
|---|---|---|---|---|---|
| s27 | 3 | 4 | 1 | 2000 | 3 |
| s208 | 8 | 11 | 2 | 2000 | 8 |
| s298 | 14 | 3 | 6 | 2000 | 14 |
| s344 | 15 | 9 | 11 | 2000 | 15 |
| s349 | 15 | 9 | 11 | 2000 | 15 |
| s382 | 21 | 3 | 6 | 2000 | 21 |
| s386 | 6 | 7 | 7 | 2000 | 6 |
| s400 | 21 | 3 | 6 | 2000 | 21 |
| s420 | 16 | 19 | 2 | 2000 | 16 |
| s444 | 21 | 3 | 6 | 2000 | 21 |
| s526 | 21 | 3 | 6 | 2000 | 21 |
| s641 | 19 | 35 | 24 | 2000 | 19 |
| s713 | 19 | 35 | 23 | 2000 | 19 |
| s820 | 5 | 18 | 19 | 2000 | 5 |
| s832 | 5 | 18 | 19 | 2000 | 5 |
| s953 | 29 | 16 | 23 | 2000 | 29 |
| s5378 | 179 | 35 | 49 | 2000 | 179 |
| s1238 | 18 | 14 | 14 | 2000 | 18 |
| s1488 | 6 | 8 | 19 | 2000 | 6 |
| s1494 | 6 | 8 | 19 | 2000 | 6 |
| aescipher | 533 | 261 | 130 | 2000 | 533 |

third column give the number of primary outputs and state variables, the fourth gives the number of scan operations performed, the fifth column gives the number of functions that are completely identified, the number of dependencies discovered, followed by the total number of dependencies in the circuit, the function coverage and the dependency coverage are shown in the last columns. In this table, a total of 17 functions were completely discovered out of 20 functions. The total number of inputs supporting the 20 functions, for s298, is 86 inputs out of which 83 were identified and only 3 inputs were not identified. In this circuit, we cannot tell whether these three inputs are real inputs to these functions. The reason for this ambiguity is that we are comparing our results to the structural input dependency. For this circuit, the function coverage is 85% and the dependency coverage is 96.5%. We would like to note that the $DP_{cov}$ is very close to 100% for most of the circuits. This shows that the procedure discovered most of the dependencies and very few were not discovered. In this experiment, we used the structural dependency to compare the results produced by our approach. As show earlier, structural dependencies contain superfluous variable. If we can determine the superfluous variables, then the $DP_{cov}$ might become 100% for more circuits.

Table 5.3 shows the size of the extracted functions. In this table, every row corresponds to one of the circuits. The first column shows the circuit name, the second column shows the total number of functions, the third column shows a pair of numbers consisting of the size of the largest function in therms of the number of inputs and the number of such function, the last column shows a pair of numbers consisting of the size of the next largest function in terms of the number of inputs and the number of such function. We noticed that in every circuit most of the functions are small with that expect of a few. This shows that the construction by enumeration is possible. In addition, design specification can be used

Table 5.2: Dependency Generation Results

| Circuit | gates | $T_{fun}$ | $N_{tries}$ | I$D_{fun}$ | $D_{dep}$ | $T_{dep}$ | $F_{cov}$ | , D$P_{cov}$ |
|---|---|---|---|---|---|---|---|---|
| s298 | 119 | 20 | 1957437 | 17 | 83 | 86 | 85% | 96.5% |
| s344 | 160 | 26 | 26572 | 26 | 121 | 121 | 100% | 100% |
| s382 | 158 | 27 | 2678858 | 24 | 172 | 175 | 88.8% | 98.2% |
| s386 | 159 | 13 | 30645 | 13 | 129 | 129 | 100% | 100% |
| s400 | 163 | 27 | 2781972 | 24 | 172 | 175 | 88.8% | 98.2% |
| s420 | 218 | 17 | 3710268 | 14 | 179 | 186 | 82.3% | 96.2% |
| s444 | 181 | 27 | 2679378 | 24 | 172 | 175 | 88.8% | 98.2% |
| s526 | 193 | 27 | 2679300 | 24 | 164 | 167 | 88.8% | 98.2% |
| s641 | 379 | 43 | 5776344 | 42 | 499 | 500 | 97.6% | 99.8% |
| s713 | 393 | 42 | 5776344 | 41 | 485 | 486 | 97.6% | 99.7% |
| s820 | 289 | 24 | 1051600 | 24 | 213 | 213 | 100% | 100% |
| s832 | 287 | 24 | 2576675 | 22 | 211 | 213 | 91.6% | 99.0% |
| s953 | 395 | 52 | 802901 | 52 | 351 | 351 | 100% | 100% |
| s5378 | 2779 | 228 | 22770072 | 216 | 2232 | 2313 | 94.7% | 96.4% |
| s1238 | 508 | 32 | 3505944 | 26 | 375 | 387 | 81.2% | 96.8% |
| s1488 | 653 | 25 | 80896 | 25 | 266 | 266 | 100% | 100% |
| s1494 | 647 | 25 | 80896 | 25 | 266 | 266 | 100% | 100% |
| asecipher | 68805 | 659 | 812910 | 659 | 7860 | 7860 | 100% | 100% |

Table 5.3: Max Inputs

| Circuit | $T_fun$ | MaxI, | NxtMaxI, |
|---|---|---|---|
| s27 | 3+1 | 4,1 | 3,2 |
| s208 | 8+2 | 8,2 | 7,6 |
| s298 | 14+6 | 8,2 | 7,6 |
| s344 | 15+11 | 13,2 | 6,3 |
| s344 | 15+11 | 13,2 | 6,3 |
| s382 | 21+6 | 14,4 | 10,3 |
| s386 | 6+7 | 12,3 | 10,5 |
| s400 | 21+6 | 14,4 | 10,4 |
| s420 | 16+2 | 34,1 | 17,1 |
| s444 | 21+6 | 14,4 | 10,4 |
| s526 | 21+6 | 14,2 | 13,1 |
| s641 | 19+24 | 27,1 | 22,3 |
| s713 | 19+23 | 27,1 | 22,3 |
| s820 | 5+19 | 21,1 | 20,2 |
| s832 | 5+19 | 21,1 | 20,2 |
| s953 | 29+23 | 18,1 | 17,5 |
| s5378 | 179+49 | 61,1 | 56,1 |
| s1238 | 18+14 | 23,1 | 22,1 |
| s1488 | 6+19 | 14,6 | 12,1 |
| s1494 | 6+19 | 14,6 | 12,1 |
| aescipher | 130+533 | 35,128 | 15,2 |

for signature matching to deduce the functions with large numbers of inputs.

# Chapter 6

# Scope of Future Work

The thesis work discussed so far assumes no knowledge of gate-level netlist of the packaged IC. Only the input/output specification of the IC is assumed to be know. If the gate-level netlist of an IC is readily available, the scope of this work can be extended to determine the flip-flops present in the packaged IC.

Let us assume that verilog netlist of the IC under investigation is available. Using the industry-standard tools, the verilog code can be converted to a bench circuit. Random test vectors are generated and applied to both the packaged IC and the bench file. Next-state values of the flip-flops present in the IC are scanned out. These values are mapped against the output of the flipflops in the bench file which is extracted from the verilog netlist.

Ambiguity analysis is performed to correlate the flipflops present in the packaged IC and extracted circuit by applying random test vectors and comparing the outputs of next-state elements present. Flowchart of the process is shown in next page. Better understanding of the circuit and functionality of the flip-flops can be known.

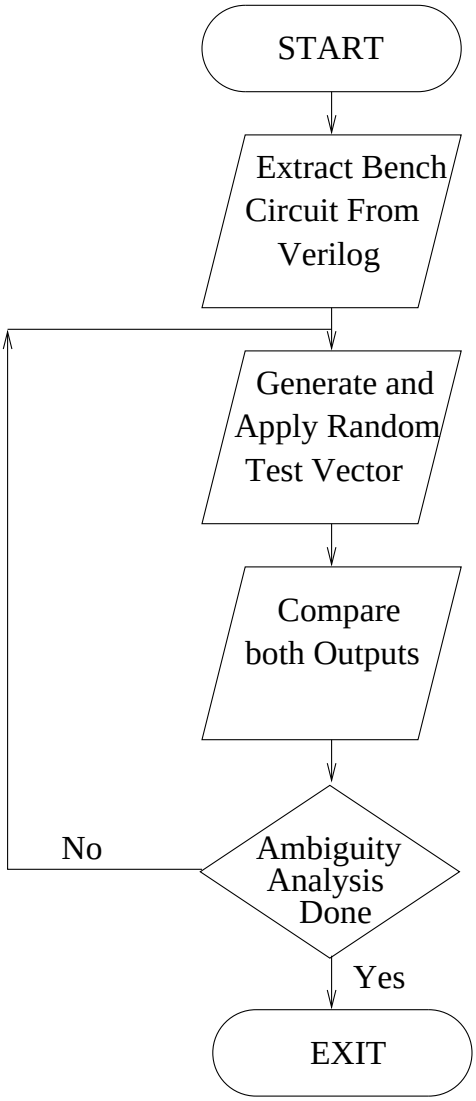


Figure 6.1: Flowchart of Ambiguity Analysis

# Chapter 7

# Conclusion

In this research work, we presented a new approach for generating a circuit netlist by re-engineering the design through an experiment that uses the scan functions and the design specification. The proposed approach for re-engineering a general sequential circuit is by exploiting and intelligently using the scan chain functions and the design specifications to construct the connectivity of the circuit blocks. The connectivity is then used to discover/compute the logic expressions and deduce the logical behavior of each block. The experimental results show that using the proposed approach we are able to construct over 90% of the system functions for a set of benchmark circuits. If the verilog code of the packaged IC is readily available then the scope of this thesis work as discussed in Chapter 6 can be extended to determining the mapping of the flip-flops present in the packaged IC.

# Bibliography


[1] http://en.wikipedia.org/wiki/Reverse$_e ngineering$.

[2] Semiconductor chip protection act. http://www.copyright.gov/title17/92preface.html, 1984.

[3] The design right (semiconductor topographies) regulations. http://www.opsi.gov.uk/si/si1989/Uksi$_1$9891100$_e n_1 .htm$, 1989.

[4] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. *IEEE 1989 International Symposium on CAS*, 1989.

[5] GREGORY H. CHISHOLM, STEVEN T. ECKMANN, CHRISTOPHER M. LAIN, and ROBERT L. VEROFF. Understanding integrated circuits. *IEEE Design Test of Computers*, pages 26–37, 1999.

[6] Eldad Eilam. *Reversing: Secrets of Reverse Engineering*. Wiley Publishing, Inc., 2005.

[7] K. Hafner, H. C. Ritter, T. M. Schwair, S. Wallstab, M. Deppermann, J. Gessner, S. Koesters, W.-D. Moeller, and G. Sandweg. Design and test of an integrated cryptochip. *IEEE Design and Test of Computers*, 1991.

[8] MARK C. HANSEN, HAKAN YALCIN, and JOHN P. HAYES. Unveiling the iscas-85 benchmarks: A case study in reverse engineering. *IEEE Design Test of Computers*, pages 72–90, 1999.

[9] D. Hely, M.-L. Flottes, F. Bancel, B. Rouzeyre, N. Brard, and M. Renovell. Scan design and secure chip. *Proceedings of the 10th IEEE International On-Line Testing Symposium*, 2004.

[10] M. Tehranipoor J. Lee, C. Patelm, and J. Plusquellic. Securing scan design using lock and key technique. *Proceedings of the 20th IEEE International Symposium on DFT in VLSI Systems*, 2005.

[11] P. Kocher, R. Lee, G. McGraw, A. Raghunathan, and S. Ravi. Security as a new dimension in embedded system design. *Proceedings of 41st Annual Conference on Design Automation*, 2004.

[12] J. Lee, M. Tehranipoor, C. Patelm, and J. Plusquellic. A low-cost solution for protecting ips against scan-based side-channel attacks. *VTS*, 2006.

[13] VIJAY K. MADISETTI, YONG KYU JUNG, MOINUL H. KHAN, JEONGWOOK KIM, and THEODORE FINNESSY. Reengineering legacy embedded systems. *IEEE Design  Test of Computers*, pages 38–47, 1999.

[14] S. Mangard, M. Aigner, and S.Dominikus. A highly regular and scalable aes hardware architecture. *IEEE Trans. Comput.*, 52, 2003.

[15] Vinesh Raja and Kiran J. Fernandes. *Reverse Engineering An Industrial Perspective*. Springer-Verlag London Limited, 2008.

[16] S. Ravi, A. Raghunathan, and S. Chakradhar. Tamper resistance mechanisms for secure embedded systems. *Proc. of the 17th International Conference on VLSI Design*, 2004.

[17] D.G. Saab, V. Nagubadi, F. Kocan, and J. Abraham. Extraction base verification method for off the shelf integrated circuits. *Quality Electronic Design, ASQED 1st Asia Symposium*, 2009.

[18] Thomas J. Swirbel. Method for fabricating a low cost integrated circuit (ic) package. http://www.faqs.org/patents/app/20080241998.

[19] K. Tiri and I. Verbauwhede. A vlsi design flow for secure side-channel attack resistant ics. *Proceedings of Design Automation and Test in Europe*, 2005.

[20] Randy Torrance and Dick James. Reverse engineering in the semiconductor industry. *IEEE 2007 Custom Intergrated Circuits Conference (CICC)*, pages 429–436, 2007.

[21] R. Usselmann. Aes cipher ip core. http://www.opencores.org, 2002.

[22] I. Verbauwhede, P. Schaumont, and K. Kuo. Design and performance testing of a 2.29 gb/s rijndael processor. *IEEE Journal of Solid-State Circuits*, 2003.

[23] Neil H. E. Weste and David Harris. *CMOS VLSI Design: A Circuit and Systems Perspective*. Pearson Publishing, 2004.

[24] Bo Yang, Kaijie Wu, and Ramesh Karri. Scan based side channel attack on dedicated hardware implementations of data encryption standard. *ITC International Test Conference*, pages 339–344, 2004.

[25] Frank C.D. Young and James A. Houston. Formal verification and legacy redesign. *IEEE AES Systems Magazine*, pages 31–36, 1999.

[26] Kenneth K. Yu and C. Neil Berglan. Automated system for extracting design and layout information from an integrated circuit. *United States Patent 5,086,477*, 1992.