P- 3453

VALIDATION OF TEST SETS FOR IMPROVING THE STUCK AT FAULT COVERAGE OF RTL CIRCUITS

By

P-3453

GU.DIVIYA

Reg. No. 0920106005

of

KUMARAGURU COLLEGE OF TECHNOLOGY

(An Autonomous Institution affiliated to Anna University of Technology, Coimbatore)

COIMBATORE – 641049

A PROJECT REPORT

Submitted to the

FACULTY OF ELECTRONICS AND COMMUNICATION ENGINEERING

In partial fulfillment of the requirements

for the award of the degree

of

MASTER OF ENGINEERING

IN

APPLIED ELECTRONICS

APRIL 2011

BONAFIDE CERTIFICATE

Certified that this project report entitled " **VALIDATION OF TEST SETS FOR IMPROVING THE STUCK AT FAULT COVERAGE OF RTL CIRCUITS** " is the bonafide work of Miss.**GU.Diviya [Reg. no. 0920106005]** who carried out the research under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

[illegible] 6/4/2011

Project Guide

Ms.S.Nagarathinam

[illegible]

Head of the Department

Dr. Rajeswari Mariappan Ph.D

The candidate with university Register no. 0920106005 is examined by us in the project viva-voce examination held on 21.04.11..........

[illegible] 21/4/11

Internal Examiner

[illegible]

External examiner

ACKNOWLEDGEMENT

A project of this nature needs co-operation and support from many for successful completion. In this regards, I am fortunate to express my heartfelt thanks to **Padmabhusan Arutselvar Dr.N.Mahalingam B.Sc.,F.I.E.,** Chairman and Co-Chairman **Dr.B.K.Krishnaraj Vanavarayar B.Com,B.L.,** for providing necessary facilities throughout the course.

I would like to express my thanks and appreciation to the many people who have contributed to the successful completion of this project. First I thank **Dr.J.Shanmugam Ph.D,** Director, for providing me an opportunity to carry out this project work.

I would like to thank **Dr.S.Ramachandran Ph.D,** Principal, who gave his continual support and opportunity for completing the project work successfully.

I would like to thank **Dr.Rajeswari Mariappan Ph.D,** Prof of Head, Department of Electronics and Communication Engineering, who gave her continual support for me throughout the course of study.

I would like to thank **Ms.S.Nagarathinam M.E,** Asst.Professor(SRG), Project guide for her technical guidance, constructive criticism and many valuable suggestions provided throughout the project work.

My heartfelt thanks to **Ms.R.Latha M.E(Ph.D),** Associate. Professor , Project coordinator, for her contribution and innovative ideas at various stages of the project to successfully complete this work.

I express my sincere gratitude to my family members, friends and to all my staff members of Electronics and Communication Engineering Department for their support throughout the course of my project.

ABSTRACT

Advances in semiconductor and electronic design automation technology have increased the size and complexity of VLSI circuits. Due to this, verification of modern circuit design has become the major bottleneck in entire design process. Most verification efforts are employed for verifying the correctness of the initial register-transfer level (RTL) descriptions written in hardware description language (HDL). Until now, the functional verification is mostly done by simulation with a massive amount of test patterns. A digital system, in general, consists of two main parts: a datapath and controller. The datapath is used to store and manipulate data and transfer data from one part of the system to another. The controller, on the other hand, controls the operation of the datapath. A validation test set is used to verify and activate controller behavior. In this paper, a methodology is used where the controller behaviors are exercised by test sequences in a validation test set and are reused for detecting faults in the datapath. Such controller behaviors are said to be compatible with the corresponding pre-computed test vectors/responses hence resulting in the detection of majority of faults in the datapath RTL modules. Also, the test generation is performed at the RTL and the controller behavior is predetermined, test generation time is reduced. The architecture is designed using VHDL and functional verification is done by using Modelsim.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLE

LIST OF ABBREVIATIONS

ATPG	**-**	**Automatic test pattern generation**
RTL	**-**	**Register transfer level**
HDL	**-**	**Hardware description language**
BIST	**-**	**Built in self test**
SBST	**-**	**Software built self test**
DFT	**-**	**Design for testability**

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without mentioning the people who made it possible, whose constant guidance and encouragement crowned my efforts with success.

I am thankful to **Dr.Ramachandiran, Principal,** Kumaraguru College of Technology, Coimbatore for giving me the necessary encouragement and guidance.

I express my sincere gratitude to **Dr.Rajeswari Mariappan Ph.D.,HOD,** Department of ECE, for giving me the necessary encouragement and guidance.

I am highly grateful to my guide **Mrs.A. Amsaveni** , Asst. Prof , Department of ECE, Kumaraguru College of Technology, Coimbatore, for providing guidance and encouraging me throughout the course of this project.

I express my profound gratitude to **Mrs.Vanitha,** Group Director, Digital Systems Group, Data Handling Division ,ISRO Satellite Center. Bangalore, who offered me an opportunity to take up this project and the facilities provided to me to work on this project.

I am highly grateful to my guide Head, **Mrs.A.Lakshmi,** Scientist'D', Data handling Division, Digital Systems Group, ISAC, for guiding me throughout the course of this project work.

I express my sincere gratitude to **Mrs.S.Srividhya**, Scientist'E', and **Mr.Nazeer Ahmed Shaik**, Scientist'D' from Data Handling Division of ISRO Satelite Center, Bangalore, who has provided their valuable experience, knowledge, and technical expertise which they shared with me to successfully accomplish this project.

I express my profound gratitude to **Mrs.V.Kamalapriya** ,Scientist 'D', ISRO Satellite Center, Bangalore, who offered me an opportunity to take up this project work.

Last, but not the least, I would like to express my gratitude to my family members, friends and to all my staff members of Electronics and Communication Engineering department for their encouragement and support throughout the course of this project.

CHAPTER -1

INTRODUCTION

1.1 OVERVIEW :

Due to the increasing complexity of modern circuit design, verification has become the major bottleneck of the entire design process. Most verification efforts are put on verifying the correctness of the initial register-transfer level (RTL) descriptions written in hardware description language (HDL). Until now, the functional verification is still mostly done by simulation with a massive amount of test patterns even though some formal verification techniques claim to be able to verify the equivalence of a design across several different design levels. Design verification deals with checking the conformance of the design to its functional specification. Simulation is the primary means used to achieve this in industry today. Sequential automatic test pattern generation (ATPG) for VLSI circuits is a hard problem with an exponential complexity. Various invasive [design for testability (DFT)] and noninvasive [software-based self-test (SBST), built-in self-test (BIST)] strategies have been suggested to reduce test generation time and improve fault coverage. While the invasive techniques tend to alter the characteristics of the design in terms of area and timing, the noninvasive techniques suffer from low fault coverage and/or large test application times.

Even though several gate level ATPG approaches have been successfully employed in practice, the increasing complexity and size of circuits, as well as the proliferation of register- transfer level (RTL) design styles has motivated development of test-generation techniques at the RTL. Previous work in the area of test generation at the RTL can be broadly classified into two categories, which are: 1) constraint-based testgeneration and 2) precomputed test-set based approach. Constraint-based test generation rely on the fact that module can be tested by abstracting RTL environment as constraints.

The extracted constraints, with the embedded module, present the gate-level ATPG tool with a circuit of significantly lower complexity than the original circuit. Precomputed test-set-based approaches first precompute the test sets for different RTL modules, and then attempt to determine functional paths through circuit for symbolically justifying a test set and the corresponding responses. The use of precomputed test sets enables RTL ATPG to focus its test effort on determining symbolic justification and propagation paths. For RTL circuits, testability measures have been used for selecting test architecture to improve fault coverage. The testability measures are based on controllability and observability analysis. Design validation deals with verifying the conformance of the design to its functional specification, thereby aiding in the elimination of design errors.

For RTL circuits with a clear controller/datapath it must possess free from stuck-at faults. Tests are generated for these faults using explicit state enumeration and used to detect stuck-at faults. In general, all these approaches attempt to measure test quality by looking at the faulty circuit behavior, i.e., behavior of the circuit with a fault inserted. The coverage metric in this case is closely modeled after the stuck-at fault model coverage metric. Once the fault is detected in the RTL path it is then subjected to validation and precomputation test sets to detect majority of faults in the path and reduce the test generation time.

A heuristic is used in this case to identify controller behaviors that can justify/propagate pre-computed test vectors/responses of datapath register-transfer level (RTL) modules. Such controller behaviors are said to be compatible with the corresponding precomputed test vectors/responses. The heuristic is fairly accurate, resulting in the detection of a majority of stuck-at faults in the datapath RTL modules. Also, since test generation is performed at the RTL and the controller behavior is predetermined, test generation time is reduced

1.2 SOFTWARES USED :

- Modelsim XE III 6.2g
- Xilinx 9.2i

1.3 ORGANIZATION OF THE REPORT

:

- **Chapter 1** deals with the introduction .
- **Chapter 2** deals with the testability,importance of testing , the various phases involved in testing and different testability analysis and the test generation .
- **Chapter 3** deals with the faults . definition of a fault , their analysis and the various fault models that exist in the circuits,the type of stuck-at faults and how these faults are propagated and their effects in the circuit performance.
- **Chapter 4** deals with the general MUX description and the steps involved in determining the test vector by externally inducing a single stuck-at fault in the circuit.
- **Chapter 5** discusses the simulation results
- **Chapter 6** shows the Conclusion and Future scope of the project

CHAPTER- 2

TESTABILITY

2.1 BASIS FOR QUALITY ASSURANCE :

Productivity is vital for survival. Increasing productivity has become a constant goal,the importance of which cannot be over-emphasized. To increase the productivity means to create more value at less cost with better quality. Quality implies the embodiment of reliability and cost-effectiveness in operation; and quality assurance relies on testing and maintainance.

2.2 TESTING :

Process of exercising a product and analyzing its resulting response to check whether the faults are introduced during the manufacturing or in operation phase.

2.3 IMPORTANCE OF TESTING :

Testing in its broadest sense, means to examine a product to ensure that the functions and exhibits the properties and capabilities it was designed to possess.The correct functioning of a circuit relies on its interconnections.The main purpose of testing is to detect the malfunctions and to locate their causes, so that the faults could be eliminated.

2.4 PHASES OF TESTING :

The phases of testing are :

- Technology development
- System development
- System-parts fabrication
- System assembly and bringup

- In – field system maintenance
- System – parts repair.

2.5 TYPES OF TESTING :

- Production (manufacturing) test ¾ test individual products to check if the faults are introduced during manufacturing phase.
- System test ¾ test a product in its operating environment to ensure that it works correctly when interconnected with other components.
- Operation and maintenance test ¾ test a product in the filed for diagnosis or "preventive" purpose.
- Prototype test ¾ testing to check for design faults during the system development phase. Diagnosis is required.
- Different levels: chip, board, or system.
- On-line, off-line, or concurrent testing.
- Functional test ¾ validating the correct operation with respect to its functional specification.
- Structural test ¾ testing of structural defects, such as open, short-ckt.

2.6 TEST GENERATION :

It is the creation of a sequence of stimulus patterns which when applied to the input's must be able to detect the fault The principle of test generation and fault simulation is to postulate the fault and detect the malfunction, then perform the set of tests devised to detect the fault one by one.

Test generation and fault simulation comprise the following steps.

- Fault specification
- Fault insertion
- Fault excitation
- Fault propagation
- Response evaluation

Main approaches for fault simulation are

- Selective tracing
- Parallel simulation
- Single fault propagation
- Concurrent simulation
- Speed-up techniques
- Parallel-value list

2.7 TEST GENERATION METHODS :

- Time frame expansion
- Simulation based method

In the case of simulation based method a fault simulator and test vector generator is used to derive the tests. Circuit is tested at the RTL level and at transistor level.

2.8 TESTABILITY ANALYSIS :

It usually refers to an estimation of the "fault coverage" that can be attained by the use of a generated test set, or an estimation of the "testability", often without actually having all the tests generated. In this case a probabilistic approach is used. In some cases, such "untestability" can be due to the presence of built –in redundancy which can be intentional or non-intentional.

Advantages of Testability :

- Reduce test efforts.
- Reduce cost for test equipments (ATE).
- Shorten turnaround time.
- Increased product quality

CHAPTER - 3

FAULTS

3.1 FAULT :

Fabrication of ASIC is a complicated process requiring hundreds of processing steps. Problems may introduce a defect that in turn may introduce a fault.

3.2 FAULT DIAGNOSIS :

On detection of testing of errors in testing,fault-diagnosis needs to be performed to locate the faults causing the errors. The main method for fault diagnosis has been to perform many additional tests, using different patterns to sensitize the alternate paths and to obtain various responses and then try to deduce from the observed results the possible cause of the errors by the process of backtracking. To avoid the delays and the expenses for repetitive and on-line computing, the input and output relationships for the faulty-circuit can be compiled into a "fault dictionary" to be looked up for comparisons and diagnostic decisions.

3.3 FAULT MODELS :

The various fault models are :

- Physical fault
- Logical fault
- Delay fault
- Open- circuit fault
- Short- circuit fault

Most of the short-circuit faults occur in interconnects; often they are known as **bridging faults (BF).** A **BF** usually results from metal coverage problem that leads to shorts. **In** case of **BF** it is classified into two categories of **Feedback** and **Non-feedback** bridging.

3.4 STUCK AT FAULT MODELS :

- Single stuck-at fault
- Multiple stuck-at fault
- Stuck-on fault
- Stuck-open fault

Two types of logical faults are :

- Stuck-at 0 fault
- Stuck-at 1 fault

3.5 FAULT EFFECT :

When the application of fault changes the circuit behaviour, the change is called as the fault effect.

3.6 FAULT PROPAGATION :

Fault effect travels through the circuit to other logic cells causing other fault effects. This phenomenon is known as fault propagation.

3.7 SINGLE STUCK –AT FAULTS :

In the single stuck -at fault models it is assumed that the effect of the physical fault is to create only two kinds of logical faults. The two types of logical faults are

- Stuck –at -0 fault
- Stuck – at-1 fault

Stuck –at -0 fault propagates the value '0' as the output irrespective of the actual functioning of the circuit.

Stuck –at -1 fault propagates the value '1' as the output irrespective of the actual functioning of the circuit.

If the fault level is at the structural level , the phenomenon is known as structural fault propagation .

If the fault level occur inside the blocks but no need to propagate the fault through the path then it is known as behavioral fault propagation.

In case if both structural as well as behavioral fault occurs in the same circuit then it is known as mixed fault simulation.

3.8 MULTIPLE STUCK –AT FAULTS :

Two or more faults mask each other in the same circuit, but they do occur rare.

3.9 UNTESTABLE FAULTS :

Faults for which no test can he found. Faults that are redundant. Faults that do not change the input output behaviour of the circuit.

CHAPTER -4

TEST VECTOR

4.1 TEST VECTOR :

In order to stimulate a device off board, a series of logical vectors must be applied to the device inputs. These vectors are called test vectors and are mostly used to stimulate the design inputs and check the outputs against the expected values. In other words, on a tester, the test vectors replace the HDL testbench used by designers to verify the design functionality. In case of a functional failure analysis, it uses test vectors to duplicate the failure of the device under test. Since antifuse FPGAs cannot be reprogrammed, the quality of the test vectors is critical to ensure a complete and timely failure analysis. Due to the tester constraints, the test vectors are not fully capable of resembling the signal conditions on the real board. A standard format in generating the test vectors are as follows :

- Improves simulation by reducing errors
- Simplifies design verification on automatic test equipment
- Reduces project cycle times
- Reduces debugging costs

Guidelines that are recommended for functional verification of designs in FPGAs, and they cover several aspects of design verification,
including:

- Waveform file formats
- Timing and sampling

- Tester constraints
- Handling bidirectional signals
- Simultaneously-switching outputs (SSOs)

4. 2 VECTOR GENERATION :

In test vector generation, the input vectors must be designed to stimulate the logic module or the I/O that needs to be monitored. In simple designs in which it is easy to toggle the desired module by stimulating few inputs, users may write a simple test bench to do so. Furthermore, the simplest designs can be tested on a bench board. In such cases, instead of test vectors, only stimulus instructions are required. For example the stimulus instructions can be as simple as the following:

Pull up input1 of the design, pull the rest of the inputs low and toggle the clock.

However, in a complicated design, where it is difficult to relate the inputs to the destination internal node or I/O directly, the easiest way to generate test vectors is to generate them from the test bench. Note that the verification test bench should naturally stimulate the desired logic, so it should create accurate vectors easily. The vector generation consists of repeating few fundamental steps:

A text file should be opened;

The inputs are sampled at the beginning of each cycle;

The outputs are sampled after being settled;

Collection of the sampled signals is written in the file.

A test bench usually contains the following blocks under the top level module:

- Stimulus module
- Instantiation of the design top module
- Special Routines

The following should be added to the test bench for vector generation flow:

- Top-level routines for:
  - Check clock generation
  - Generating time stamp (optional)
  - Character conversion to tester format
- Processes for:
  - Formatting each line of the Vector file
  - Converting each bit of each signal to tester format
  - Writing completed vector line into the file

4. 3 VECTOR VERIFICATION :

The generated vectors should be functionally verified before being applied to the real silicon on the tester. Due to the real time delays on the silicon, the generated output vectors may not match the outputs of the silicon. This is similar to simulating the design functionality without back-annotating the delays after place and route. The mismatch may happen if the back-annotated delays (SDF file) are not incorporated in the vector generation. Therefore, the test vectors should be verified using the backannotated. delays (SDF file) to simulate the real time delays on silicon. In vector verification flow, the vectors are read from the vector file. The input vectors are applied to the design as force vectors and the output of the design is compared with the output vectors. Since the test vectors will be applied to the real silicon with the real time delays, it is recommended that the back-annotated netlist and SDF files be used in vector verification instead of the RTL level design. Similar to the vector generation the timing of applying inputs and sampling outputs is important. In simulation for vector verification, there should be enough time intervals between applying the inputs and checking the outputs. The following routines should be included in the top-level of the verification code:

- Reference (and check) clock generation
- Format conversion from the tester format

The necessary processes in the code are:

- Reading vector file
- Converting the format from the tester format by calling routines.
- Creating "force" vectors to be applied to the design and "observe" Vectors for comparison.

4. 4 TEST VECTOR GENERATION FLOW :

The number of ports of the design is limited to few ports in order to avoid confusion and unnecessary complication. After generating the files, it should be run through a format conversion script. However, the vectors, indicated by "out" should be converted. After finishing the format conversion, the files can be glued together in format similar to the VHDL flow vector file. A similar pin list should be generated too to accompany the vector file. The inputs are applied to the design inputs and outputs of the design are registered to be compared against the expected values in the vectors. The bidirectional vectors may act as inputs or outputs according to the state of the control signals.

4. 5 VECTOR TESTING :

To apply at-speed tests to detect timing-related faults, existing structural BIST needs to resolve various complex timing issues related to multiple clock domains, multiple frequencies and test clock skews which are unique in the test mode. In contrast, self-testing the devices using instruction sequences allows more natural application of at-speed tests.

Delay tests are applied by executing instruction sequences in a similar fashion as normal system operations.

Pure functional self-test may or may not give a desired level of test quality. This paper recent research are trying to provide an answer to this dilemma. However, regardless of what the answer might be, good understanding of the capability and limitations of functional self-test can provide with more insights in how functional selftest can complement structural BIST. These insights may further trigger new research to study hybrid solutions which combine the strengths of functional and structural self-test.

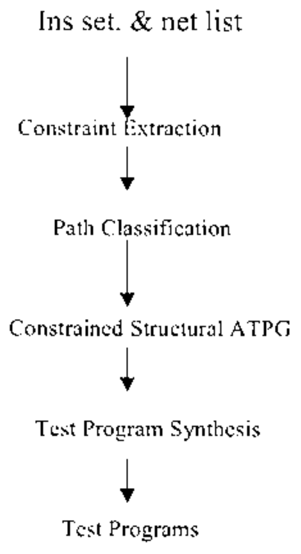


Figure 4.1 Test program generation process

4. 6 Test generation process :

The method consists of three phases. In the first phase, a path classification algorithm is applied to identify a tight superset of the path delay faults which can be tested using instructions. It is observed that a structurally testable path (i.e., a path testable through at-speed fullscan) in a microprocessor might not be testable by its instructions - simply because no instruction sequence can produce the desired test sequence which can sensitize the path and capture the fault effect into the destination output flip-flop at-speed. These paths are called functionally untestable paths. Elimination of functionally untestable paths can significantly help identifying achievable path delay fault coverage and also reduce the computational effort of the subsequent test generation process. We extract a set of constraints capturing correlations among input / output *(I/O)* signals and register d flip-flops of the datapath logic, while offering good test quality. Given the extracted constraints and the gate-level netlist, the classification will eliminate all functionally untestable paths from the fault universe. Our experimental results show that a large percentage of the total number of paths are functionally untestable.

In the second phase, a constrained, structural gate-level ATPG for path delay faults is used to generate deterministic tests for functionally testable paths. Given a path delay fault, the gate-level netlist, and the set of constraints extracted in the first phase, the constrained, combinational ATPG generates a minimally-specified test vector pair which specifies required bit-level values at primary inputs and register flip-flops.

The third phase is the program synthesis phase. In this phase, the vector pairs generated in the second phase are mapped into required word-level values at the inputs and the registers flip-flops in two consecutive cycles. An instruction-level justification process is then invoked. Based on the instructional set architecture and micro-architecture, the effects on internal registers and on major signal lines in the processor after the execution of each instruction are pre characterized and modeled.

This high-level abstraction is used to guide the instruction-level justification which significantly reduces the search space.

After finding a sequence of instructions that will bring the processor to state satisfying the required value assignments, a signature calculation subroutine is appended, which compresses the contents in all registers and stores the result in memory in order to form the complete test sequence for the target fault. The above procedure is repeated until all target path delay faults have been examined. In results, most vector pairs generated by the constrained ATPG can be successfully realized by instructions. Besides, since we use only the behavioral information of the processor in this phase, the search space is small and thus, the runtime is efficient.

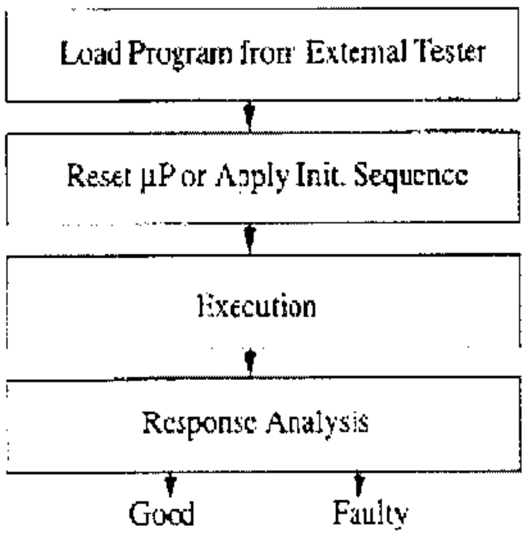

Load Program from External Tester
Reset µP or Apply Init. Sequence
Execution
Response Analysis
Good
Faulty

Figure 4.2 Test procedure

4. 7 Test procedure :

The test application procedure of a test program is given in Fig. The test program is loaded into the on chip memory by an external tester at a slow speed. Depending on the architecture of the processor core, the external tester might need to reset the processor or apply an initialization sequence so that the processor starts fetching and executing instructions from memory. When the test program is being executed at-speed, a set of signatures is recorded in memory. At the end of the test program, a response analysis subroutine is called to further compress the recorded signatures in memory and finally, compare the compressed signature with the correct signature.

4. 8 Control Event :

A control event is a unique set of control variable values when projected onto the set of variables observed by the datapath .

4. 9 Defining Control Event :

There are two parts: a state coverage metric (SCM) and a transition coverage metric.

$$SCM = \frac{Number\ of\ States\ Visited}{Total\ Reachable\ States}$$

$$TCM = \frac{Number\ of\ Transitions\ Taken}{Total\ Reachable\ Transitions}$$

CHAPTER -5

RTL CIRCUIT

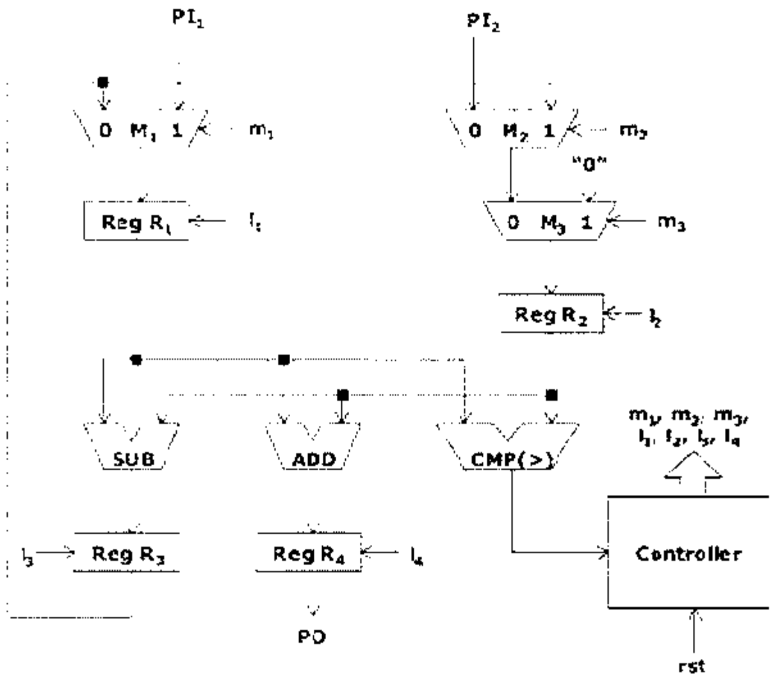


Figure 5.1 RTL circuit

5. 1 RTL DESCRIPTION :

- Consider the RTL circuit shown in Fig. 4.1.
- The circuit comprises registers R_1, R_2, R_3, R_4 .
- Primary inputs PI_1, PI_2 and reset *rst*.
- Primary output PO.
- Multiplexers M_1, M_2 and M_3.
- Comparator CMP, adder ADD.
- Subtractor SUB, and a controller.
- The bit-width of all modules in the datapath is four.
- The controller, shown in consists of seven states.
- S0, S1, S2, S3, S4, S5 and S6.
- For various combinations of input, (PS) and (NS) vary.

Output from the compare signal is fed as input to that of the controller block. Reset serves as input to the controller block and the final output of the circuit will be obtained as PO. The simultaneous operation of both the blocks constitute the intermediate results and the data to flow in and out of the circuit. The next state (NS) and values at the outputs of the controller are determined by its present state (PS), CMP's output, and *rst*. If *rst* is asserted then NS is always irrespective of PS and the value at CMP's output.

5. 2 CDFG :

A precomputed test vector for ADD consists of two subvectors (v_1,v_2) corresponding to the two inputs of the module. ADD uses a carry-lookahead implementation for computing the sum of the values held by its two inputs. A precomputed test set that detects all the stuck-at faults in ADD consists of the vectors: $\{(15,1),(1,15),(2,14),(5,12),(8,9),(0,15),(14,0),(13,1),(11,3),(7,7)\}$.

The goal of RTL test generation is to justify these vectors from the inputs of ADD to primary inputs PI_1,PI_2 , and *rst*, and propagate the responses to primary output .

The controller behavior derived from test sequence T_1 is reused during test generation by: 1) ensuring that the sequential circuit is unrolled and 2) maintaining the same values for signals *rst* and CMP as imposed by test sequence.

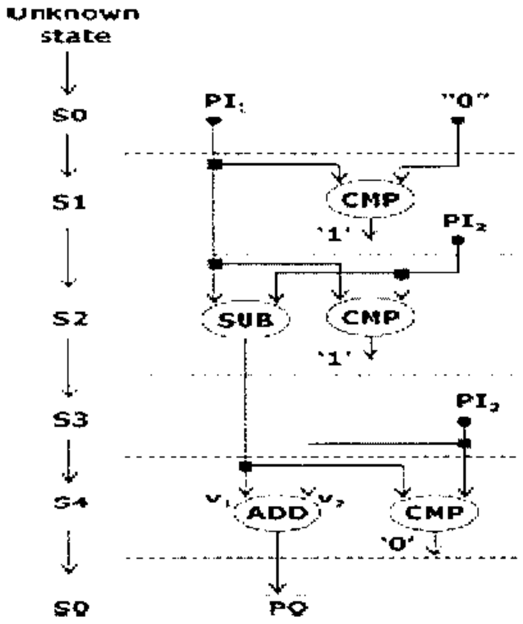


Figure 5.2 CDFG and state transition sequence exercised by Validation rest sequence

In this figure precomputed test vector (v_1,v_2) can be delivered to ADD only in state (S_4) and the response can be observed only in the subsequent state S_0 . However, in general, a given module can be exercised in multiple states by a given validation test sequence. For example, CMP in this Fig is exercised in states S_1, S_2, S_4 . By reusing the controller behavior extracted from T_1 , five out of ten vectors in the pre-computed test set can be delivered to the inputs of ADD and their responses observed at the primary output.

5.3 RTL FAULT MODELS :

As it is difficult to find the correlation between high-level fault models and gate level fault models. It mainly focus on the RTL fault simulation, but not the mapping between RTL and gate level fault models. For the VRM circuit model, we can define fault models in natural way. These are stuck-at bit faults of d-node and k-node, except of process and function node, which is associated with its CFG. Furthermore, c-nodes in CFG do not need fault models at all, since each c-node is associated with a DFG, in which its d-node faults are enough to represent assignment or condition faults including the left-value variable faults and internal nodes faults of the expression. So it is possible to obtain the good mapping between two level fault models.

5.4 RTL FAULT SIMULATION :

There are three typical fault simulations in the gate levels, i.e. parallel, deductive and concurrent. At the RTL, the logic values dealt will be for words, instead of bits. So the concurrent simulation may be the unique choice. Based on the VRM model, we propose a RTL concurrent fault simulation approach. Since we do not deal with the delay faults, we will use the 0-delay model. Furthermore, for simplicity, we only consider single clock and synchronous sequential circuits. So the simulation is cycle-accurate, and the circuit levelizing is performed in preprocessing to avoid the iteration of simulation. The concurrent fault simulation is a one-pass process. It simulates all active faults simultaneously for each test input. The super fault list (SFL) at RTL is defined. After the good simulation, then we can select all active faults and inject to VRM for fault simulating, which convert the selected fault into a super fault, add it to the SFL of its corresponding node, and initiate a list event.

During good simulation, the simulation event can be input value/edge, status value/edge, clock edge and internal signal value/edge. The simulation starts from CKT level. When there is a process, then go into its CFG in terms of DFGs. For the concurrent fault simulation, due to SFLs, the event will be the logic list event. At the CKT and DFG level models, like the gate level, the fault simulation or event propagation is simple.

Along the good path, simulate all possible faults simultaneously, which do not cause any faulty branch. When the first faulty branch met due to some faults (called as branch fault), then need simulate all possible faulty paths such that only propagate these branch faults one by one. This must be a serial fault simulation in order to avoid the appearance of multiple faults. In that case, it needs a LIFO stack to manage all possible branch super faults to be simulated correctly.

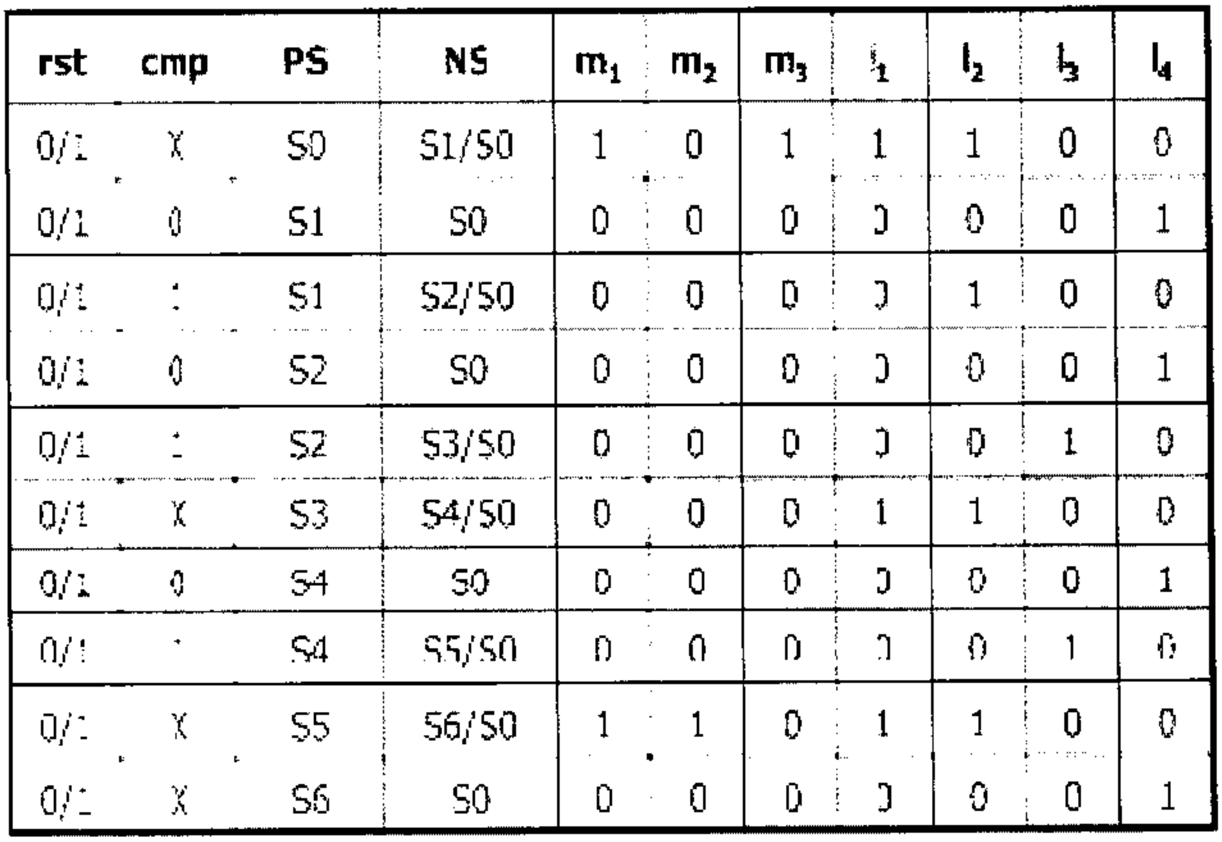
The RTL concurrent fault simulation approach is feasible. Since the active part of a Verilog program for each test input is a small rate, the number of selected faults for the input to simulate and the size of total super fault lists will not be too large. So one can implement efficient fault simulators for VLSI, which even needs less overhead comparing to the gate-level one. For the VFSim, the RTL fault simulation result reflects the gate-level one in a sense. Next is to include the VRM extension, and acceptable RTL fault models which can map to gate-level ones in statistical sense.

CHAPTER - 6

CONTROLLER

6. 1 STATE TRANSITION TABLE :

- The state transition table for the controller is obtained.
- It deals with various input' s and their corresponding output' s.



rst	cmp	PS	NS	m_1	m_2	m_3	l_1	l_2	l_3	l_4
0/1	X	S0	S1/S0	1	0	1	1	1	0	0
0/1	0	S1	S0	0	0	0	0	0	0	1
0/1	1	S1	S2/S0	0	0	0	0	1	0	0
0/1	0	S2	S0	0	0	0	0	0	0	1
0/1	1	S2	S3/S0	0	0	0	0	0	1	0
0/1	X	S3	S4/S0	0	0	0	1	1	0	0
0/1	0	S4	S0	0	0	0	0	0	0	1
0/1	1	S4	S5/S0	0	0	0	0	0	1	0
0/1	X	S5	S6/S0	1	1	0	1	1	0	0
0/1	X	S6	S0	0	0	0	0	0	0	1

Figure 6.1 Controller State Transition Table

6. 2 CONTROLLER DESCRIPTION :

Assume that T_1 is a validation test sequence consisting of the following six vectors: {(1000000000), (000010000), (000000000), (000000000), (000000101), (000000000)}.Each vector consists of nine bit values held by different primary inputs in the following order: *rst*, $PI_1[3:0]$, $PI_2[3:0]$. The behavior of the controller is governed by the values held by the input signals to the controller (*rst* and CMP's output).

The pre-computed test vector/response of an embedded RTL module in the datapath of a circuit can be justified/propagated by reusing the controller behavior extracted from a validation test sequence. It is defined to be compatible with the pre-computed test vector/response. A heuristic is quickly used to identify a compatible controller behavior for a given pre-computed test vector/response. Hence, test generation is performed only for those precomputed test vectors/responses for which a compatible controller behavior has been selected by the heuristic. This significantly reduces the number of test generation attempts required to justify/propagate the precomputed test vectors/responses of all the RTL modules in the datapath of a circuit.

The heuristic consists of two steps. The validation test sequences are first fault-simulated. The stuck-at fault simulator is augmented to capture the cycles in which a fault gets activated and eventually detected. If a detected fault belongs to an RTL module in the datapath, then the second part of the heuristic is used to estimate whether the controller behavior extracted from the corresponding validation test sequence is compatible with any one of the module's pre-computed test vectors/responses. This is done under the assumption that pre-computed test vectors (responses) will take the same number of cycles to get justified (propagated) as taken for the detected stuck-at fault.

CHAPTER - 7

TEST GENERATION METHODOLOGY

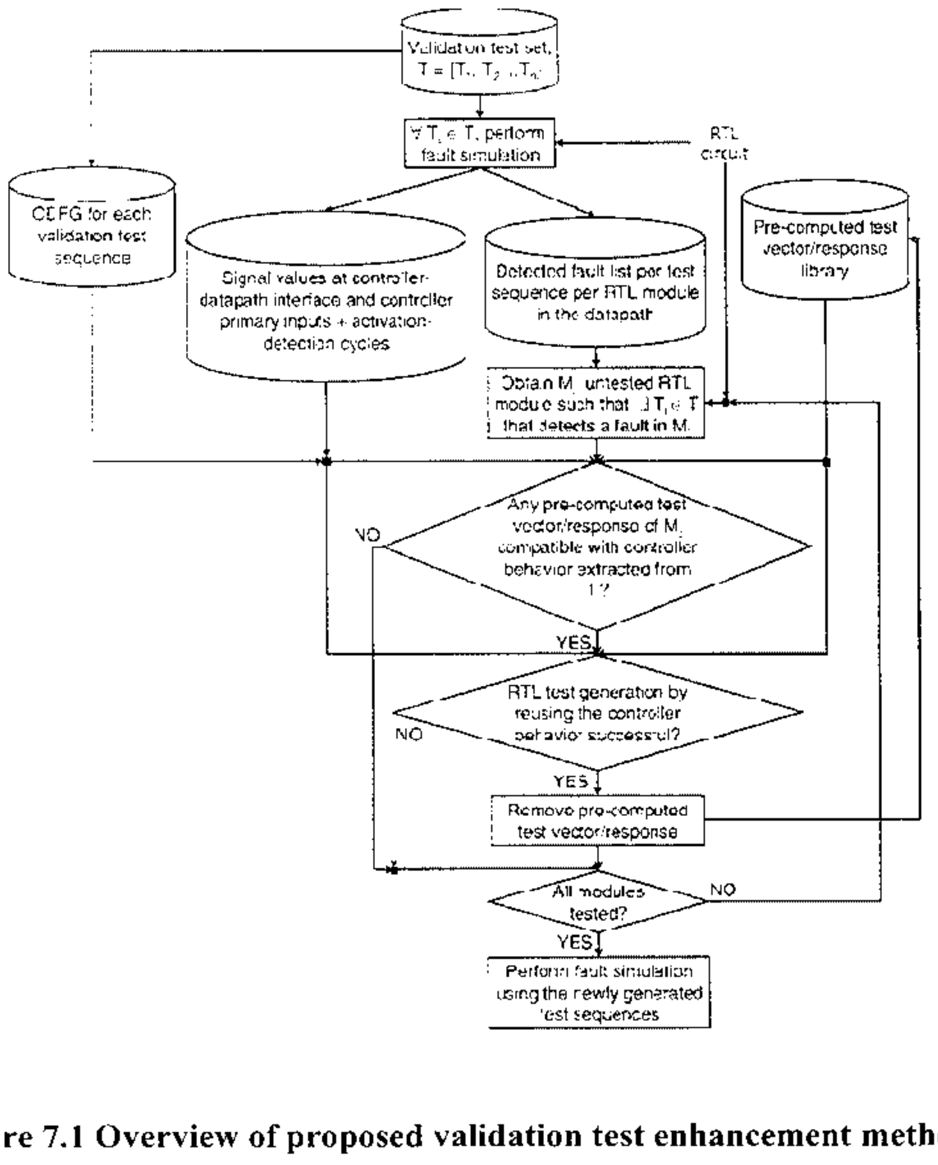

Validation test set, T = {T1, T2 ..TN}
∀ Ti ∈ T, perform fault simulation
RTL circuit
CDFG for each validation test sequence
Signal values at controller-datapath interface and controller primary inputs + activation-detection cycles
Detected fault list per test sequence per RTL module in the datapath
Pre-computed test vector/response library
Obtain Mj untested RTL module such that ∃ Ti ∈ T that detects a fault in Mj
Any pre-computed test vector/response of Mj compatible with controller behavior extracted from Ti?
NO
YES
RTL test generation by reusing the controller behavior successful?
NO
YES
Remove pre-computed test vector/response
All modules tested?
NO
YES
Perform fault simulation using the newly generated test sequences

Figure 7.1 Overview of proposed validation test enhancement methodology

7. 1 TEST ENHANCEMENT DESCRIPTION :

Fig.6.1 shows the proposed validation test generation methodology. The initial data set consists of an RTL circuit description, a synthesized gate-level netlist and the validation test set T . Assume that T contains n test sequences (T_1, T_2, $T_3 \ldots T_n$). For each test sequence , we perform fault simulation to determine the faults detected in different RTL modules in the datapath.

If a validation test sequence T_i detects one or more faults in a datapath RTL module M_j , then we determine whether the controller behavior extracted using T_i is compatible with any of the precomputed test vectors/responses of M_j . The heuristic uses the information obtained from the augmented fault simulator to verify quickly whether justification/propagation of a pre-computed test vector/response in a CDFG corresponding to a validation test sequence will result in a conflict. If the metric is below the threshold value for all conflicts, then the controller behavior is declared to be compatible with the pre-computed test vector/response. Note that only a single pass through the CDFG is required for a given validation test sequence and a given pre-computed test vector/response. If the controller behavior is compatible, then RTL test generation is performed by reusing the controller behavior to find a test sequence. If a test generation run is successful, then the corresponding pre-computed test vector/response is removed from the precomputed test vector/response library. This process is repeated until all the RTL modules in the datapath have been targeted. Fault simulation is then performed using the newly generated test sequence to determine the stuck-at fault coverage obtained using the proposed methodology. Reusing the controller behavior for RTL test generation results in very low test generation times. Also, a majority of stuck-at faults in the datapath RTL modules get detected.

CHAPTER – 8

JUSTIFICATION PROCESS

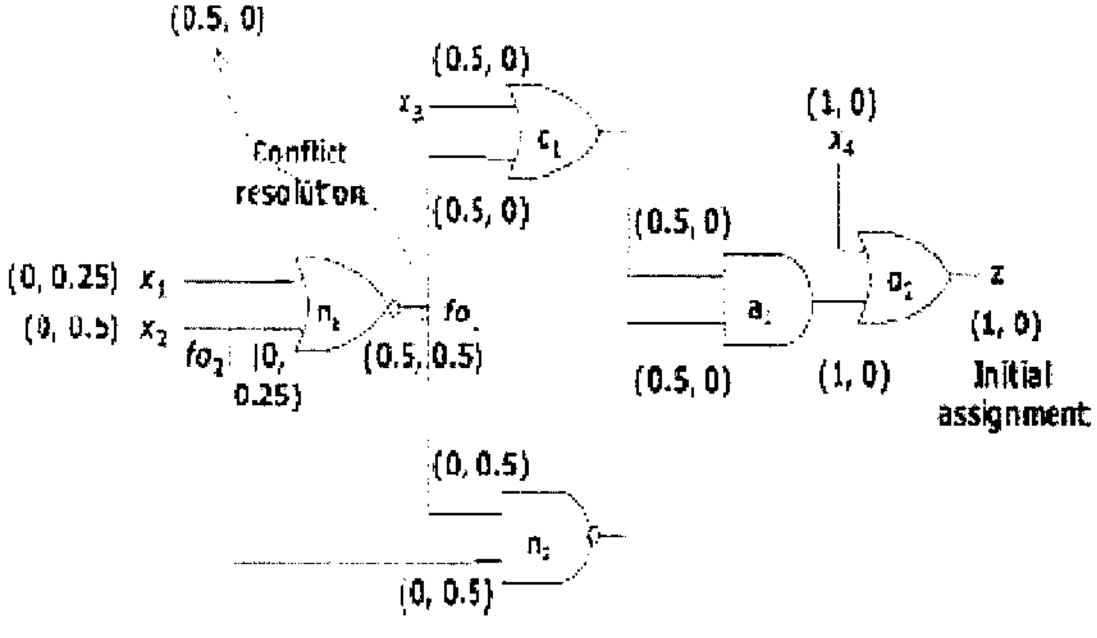


Figure 8.1 JUSTIFICATION OF REQUIREMENT VALUES

Consider the gate-level circuit. It consists of one NOR gate n_1 , one NAND gate n_2 one AND gate a_1 , two OR gates o_1 and o_2 , four primary inputs x_1,x_2,x_3 and x_4 , one primary output z , and two fan-out nodes f_1 and f_2. The requirement values are captured in the format (r_o, r_1). Assume that an initial requirement value assignment of (1,0) is made at while the requirement values for all the other nodes are initialized to (0,0).

8. 2 JUSTIFICATION ANALYSIS :

Given a validation test sequence, we first extract the CDFG exercised by the sequence. Next need is to determine efficiently and accurately whether justification/propagation of any precomputed test vector/response of an RTL module in the CDFG will result in a failure. For this purpose, it define two measures called *1-requirement (r_1)* and *0-requirement (r_0)* each line in the CDFG. They capture approximate requirements for a line to hold binary value "1" or "0", respectively, for justifying/propagating a pre-computed test vector/response in the CDFG. The range of values that r_0 and r_1 and can assume is $0 \leq r_0, r_1 \leq 1$. Also, for any line, if r_1 $(r_0)>0$ and r_0 $(r_1)=0$. Given a set of initial assignments, the nonzero requirement values are justified to primary inputs in the CDFG. The initial assignments correspond to a pre-computed test vector/response and the values at the controller inputs. Justification through RTL modules is performed using their gate-level implementation. The justification process obeys the following rules for gate-level modules.

- AND gate: Let the number of gate inputs be n . Let i and o represent the gate input and output lines, respectively. Then

  if $r_0(i) < r_0(o)/n$, then $r_0(i) = r_0(o)/n$
  if $r_1(i) < r_1(o)$, then $r_1(i) = r_1(o)$.

- OR gate: Let the number of gate inputs be n . Let i and o represent the gate input and output lines, respectively. Then

  if $r_0(i) < r_0(o)$, then $r_0(i) = r_0(o)$
  if $r_1(i) < r_1(o)/n$, then $r_1(i) = r_0(o)/n$.

- NOT gate: Let i and o and represent the gate input and output lines, respectively then,

  if $r_0(i) < r_1(o)$, then $r_0(i) = r_1(o)$
  if $r_1(i) < r_0(o)$, then $r_1(i) = r_0(o)$

- Fan-out point: Let a line i have k fan-outs $o_1, o_2, \ldots.ok$. Then

  $r_0(i)=\max(r_0(o_p), r_0(i))$
  $r_1(i)=\max(r_1(op), r_1(i)),\ 1 \leq p \leq k.$

The CDFG is levelized before the justification process. Hence, a single pass is sufficient to justify the requirement values to the primary inputs. In some cases, during the justification process a condition might arise wherein the 0- and 1-requirements of a particular line are both assigned values greater than 0. Such a conflicting situation arises when a given line in a CDFG might need to hold both Boolean values "0" and "1" in order to satisfy the initial assignment of requirement values. The extent of the conflict on a line is measured with the help of the following metric :

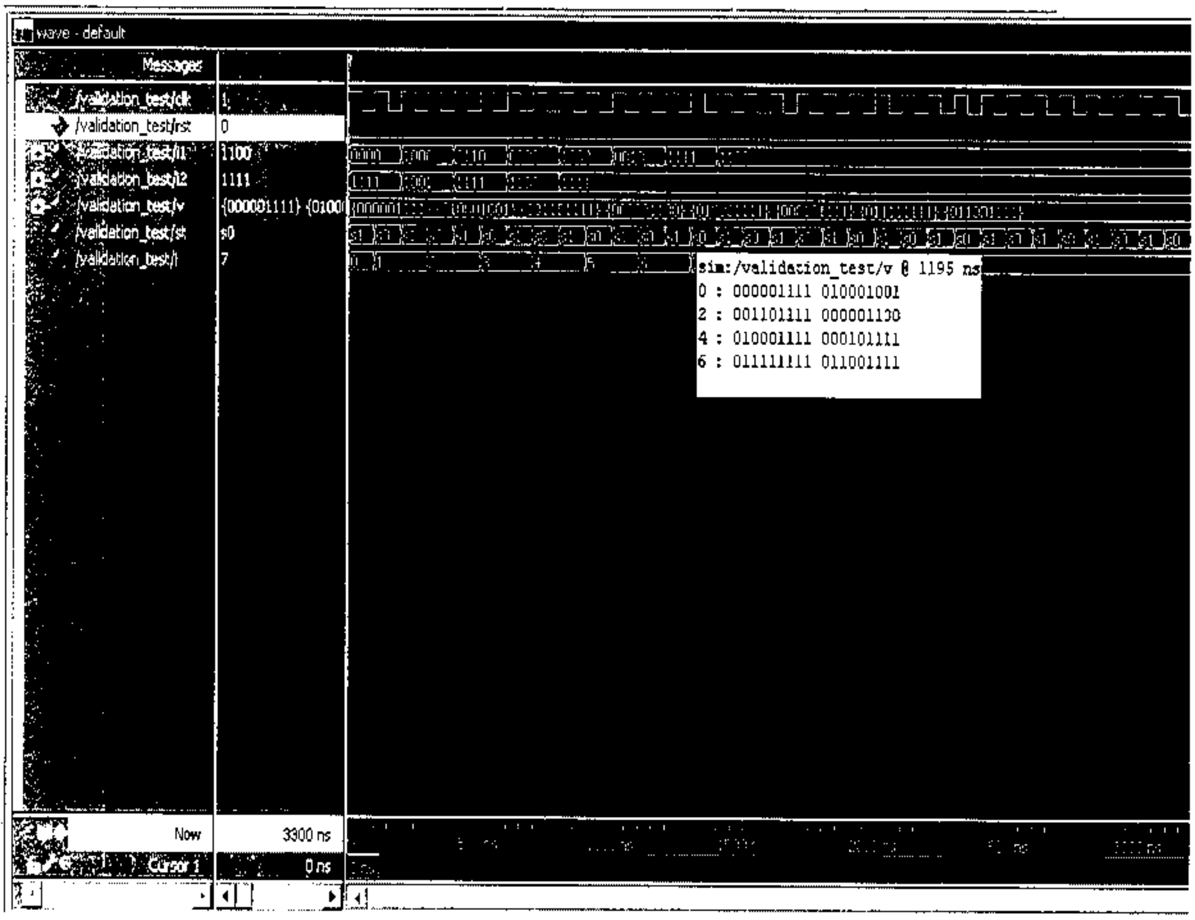
$$\gamma = \frac{1}{2^{1-(r_0(l)*r_1(l))} - 1} - 1.$$

When either of the parameters hold the value 0, γ evaluates to 0, indicating no conflict. However, when both $r_0(l)$ and $r_1(l)$ are equal to 1, γ evaluates to infinity indicating that l needs to hold both complementary Boolean values, which is not possible. Also, γ is a monotonically increasing function in both $r_0(l)$ and $r_1(l)$ in the range $0<r_0(l),r_1(l)\leq 1$. Based on the experiments presented a threshold value of 1.0 was selected for γ . Hence, if γ for a conflicting situation is greater than 1.0, then justification is stopped and the corresponding controller behavior is declared to be incompatible with the pre-computed test vector/response. However, if $\gamma \leq 1.0$, then the smaller of the two requirement values is changed to 0.

The values are stored based on the levels of the corresponding lines. The values are justified starting from lines with the lowest level number (primary outputs in the CDFG) to lines that have the maximum level number (primary inputs in the CDFG). In case of a conflict on a line o (both r_0 (o), r_1 (o) > 0) ,γ is evaluated. Depending on its value, we either continue with the justification process or declare the controller behavior corresponding to the CDFG as incompatible with the targeted precomputed test vector/response. If the justification process completes with all intermediate values, if any, falling below the threshold value, then the controller behavior is declared to be compatible with the pre-computed test vector/response .

The requirement values are captured in the format (r_o, r_1). Assume that an initial requirement value assignment of (1,0) is made at , while the requirement values for all the other nodes are initialized to (0,0). The initial nonzero assignment at is justified using the procedure propagating to the primary inputs. In other words, we want to quickly estimate whether there exists an assignment of Boolean values to primary inputs that imply a Boolean value "0" at . Fig.7.1 shows the requirement values at different nodes at the end of the justification process. A conflict arises at fan-out node f1, since both $r_0(f_1)$ and $r_1(f_1)$ get assigned a value of 0.5. γ for this conflict evaluates to 0.46. Since γ is less than (1.0), the conflict is resolved in this case. The heuristic predicts that there exist primary input value assignments that resolve this conflict. This is indeed true as there are three vectors that result in a Boolean value "0" at z .

CHAPTER – 9
SIMULATION RESULTS



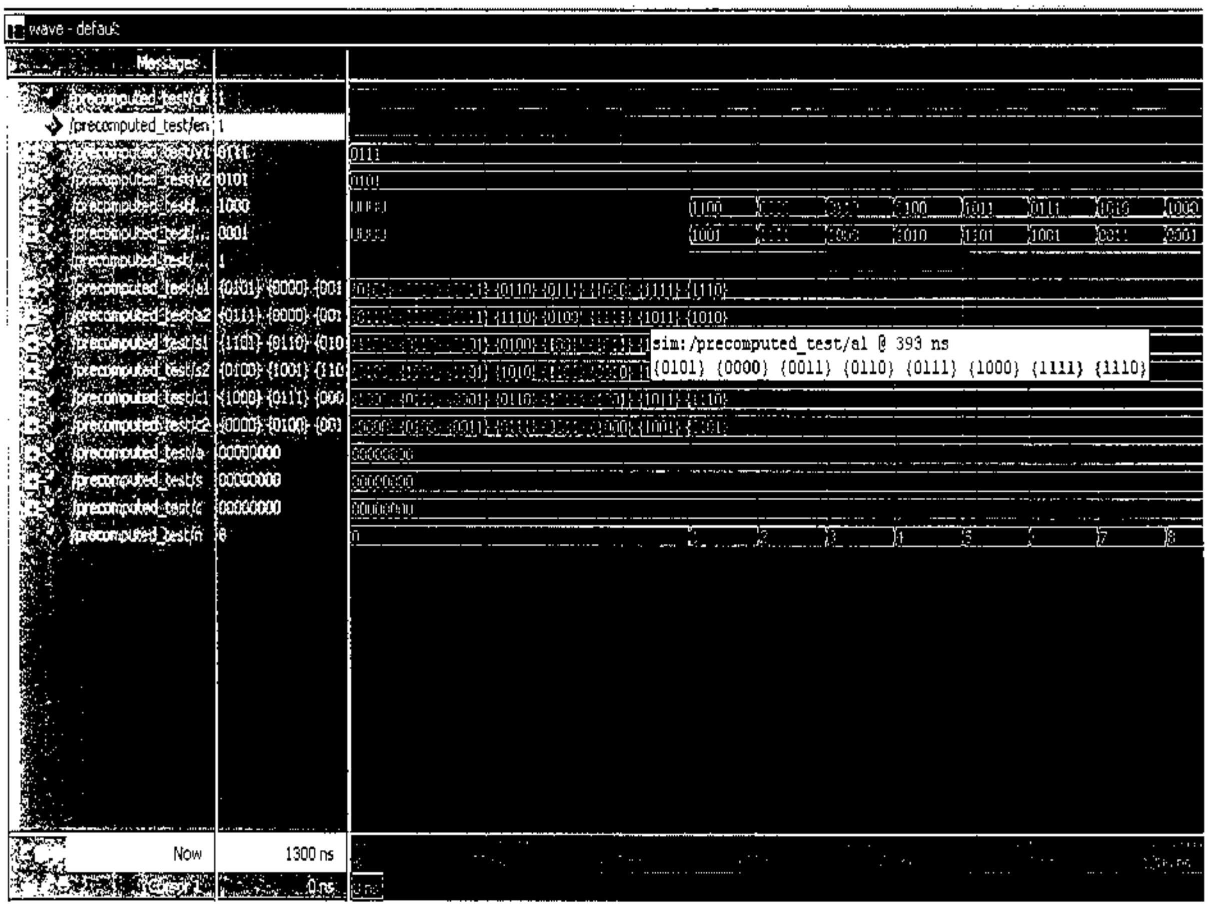
```
wave - default
Messages
/validation_test/clk     1
/validation_test/rst     0
/validation_test/i1      1100
/validation_test/i2      1111
/validation_test/v       {000001111} {0100
/validation_test/st      s0
/validation_test/i       7
Now       3300 ns
Cursor 1  0 ns
```

```
sim:/validation_test/v @ 1195 ns
0 : 000001111 010001001
2 : 001101111 000001110
4 : 010001111 000101111
6 : 011111111 011001111
```

Figure 9.1 Simulation result for validation test set

The input's are : CLK, RST

The output's are : S_0, S_1



```
/precomputed_test/clk   1
/precomputed_test/en    1
/precomputed_test/y1    0111
/precomputed_test/y2    0101
/precomputed_test/...   1000
/precomputed_test/...   0001
/precomputed_test/...   1
/precomputed_test/a1    {0101} {0000} {001
/precomputed_test/a2    {0111} {0000} {001
/precomputed_test/s1    {1101} {0110} {010
/precomputed_test/s2    {0100} {1001} {110
/precomputed_test/c1    {1000} {0111} {000
/precomputed_test/c2    {0000} {0100} {001
/precomputed_test/a     00000000
/precomputed_test/s     00000000
/precomputed_test/c     00000000
/precomputed_test/n     8
```

```
sim:/precomputed_test/al @ 393 ns
{0101} {0000} {0011} {0110} {0111} {1000} {1111} {1110}
```

Figure 9.2 Simulation result for pre- computation test set

The input's are : CLK, RST, EN

The output's are : Test vector set

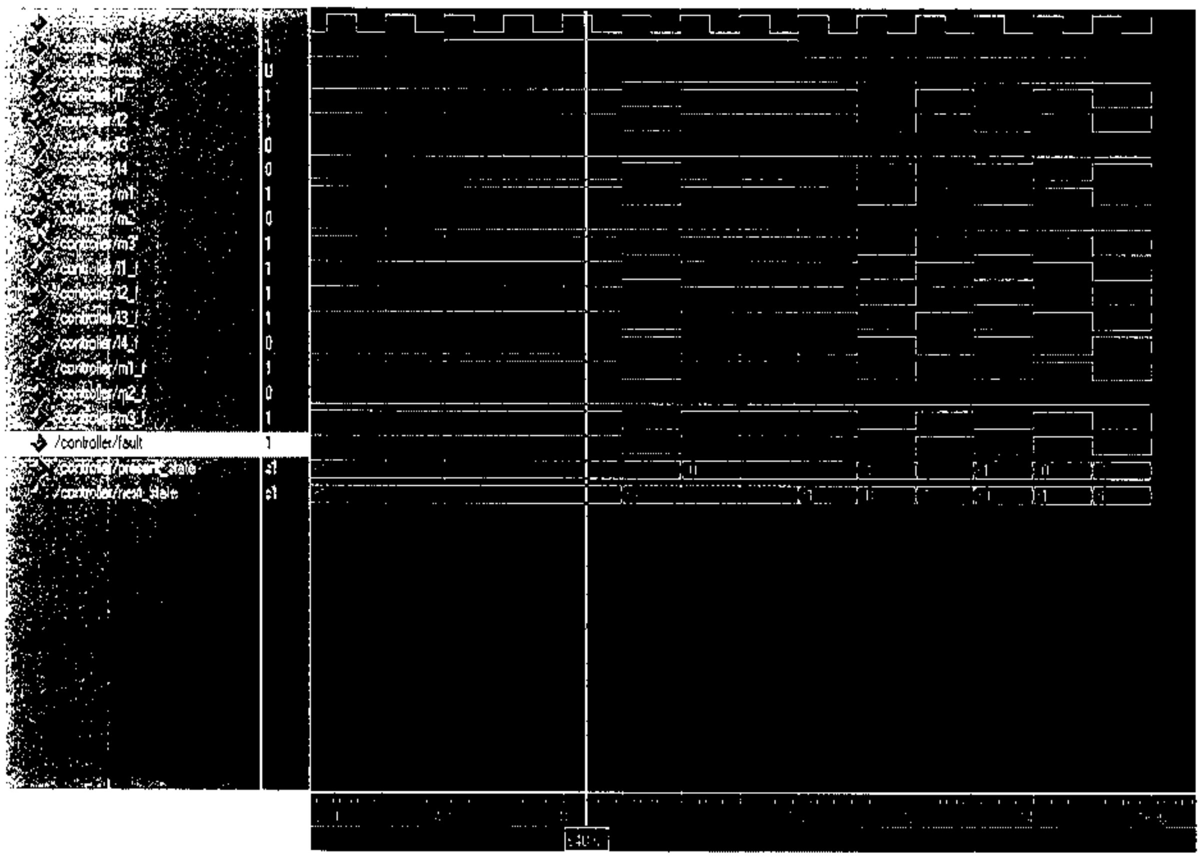


Figure 9.3 Simulation result for proposed fault

The input's are : CLK, RST, CMP

The output is : presence of fault indicating as '1'

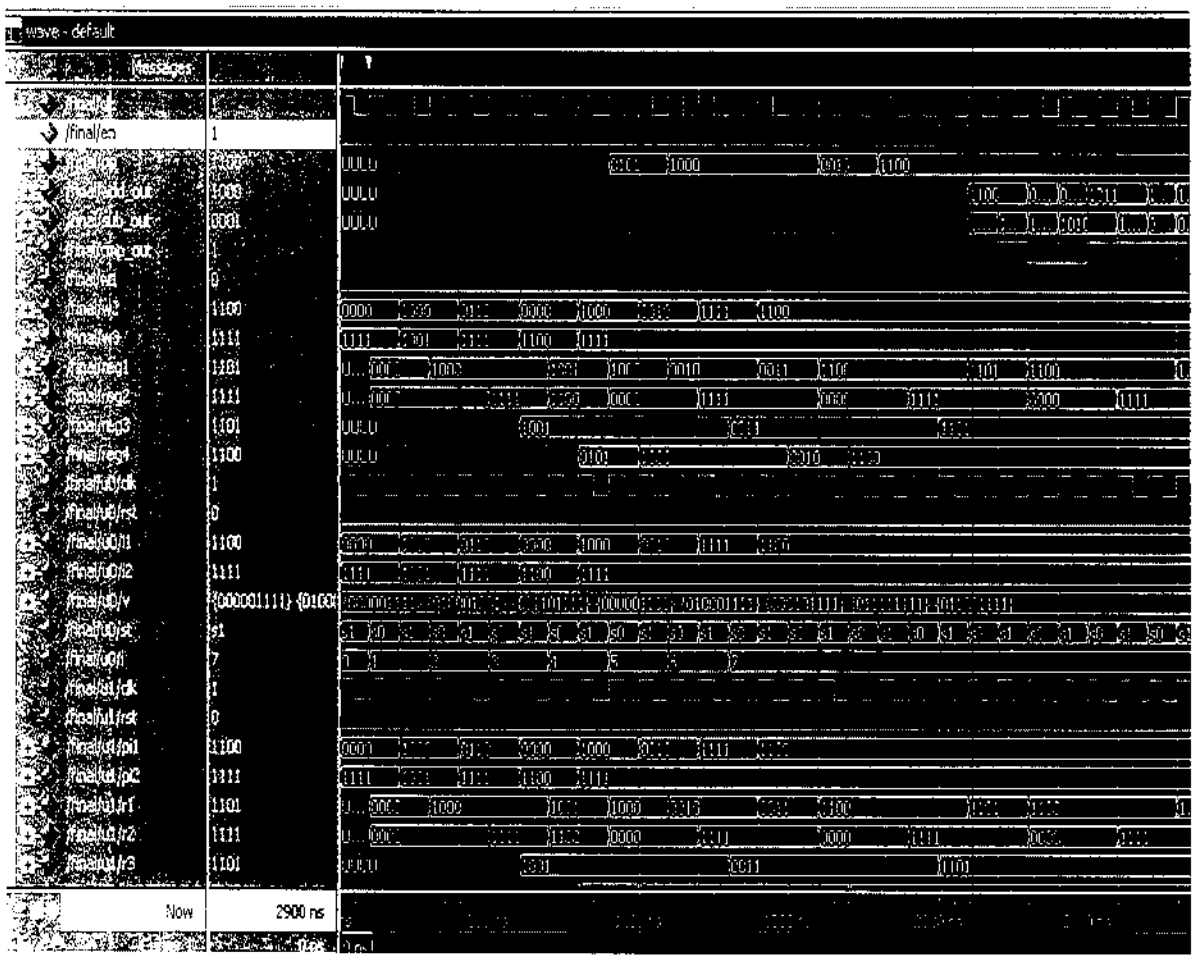
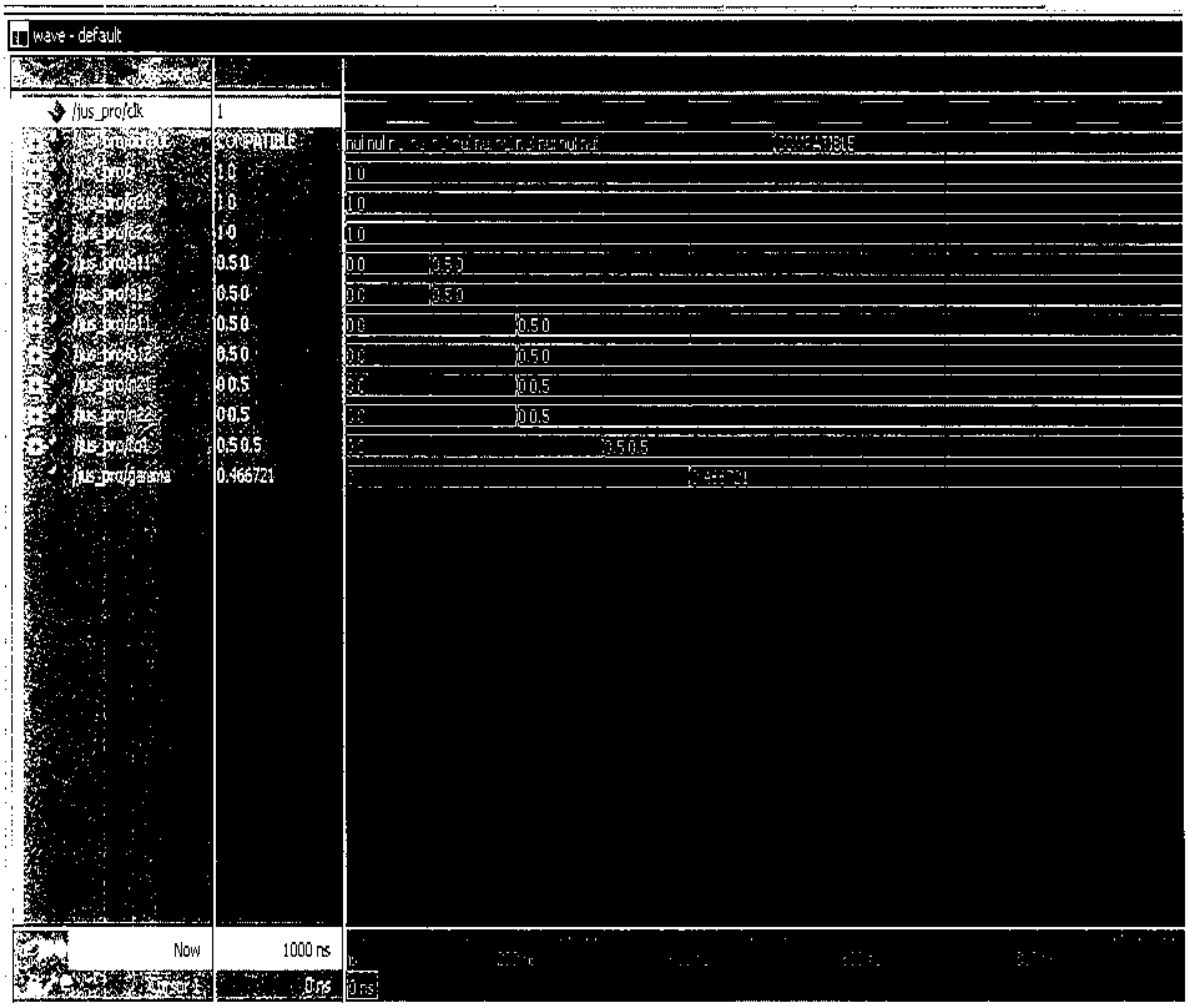


Figure 9.4 Simulation result for final circuit

The input's are : CLK, EN

The output is : P_O



```
/jus_pro/clk         1
/jus_pro/[illegible] COMPATIBLE   nul nul [illegible]              COMPATIBLE
/jus_pro/[illegible] 1 0          1 0
/jus_pro/[illegible] 1 0          1 0
/jus_pro/[illegible] 1 0          1 0
/jus_pro/a11         0.5 0        0 0   0.5 0
/jus_pro/a12         0.5 0        0 0   0.5 0
/jus_pro/[illegible] 0.5 0        0 0         0.5 0
/jus_pro/[illegible] 0.5 0        0 0         0.5 0
/jus_pro/n21         0 0.5        0 0         0 0.5
/jus_pro/n22         0 0.5        0 0         0 0.5
/jus_pro/[illegible] 0.5 0.5      0 0               0.5 0.5
/jus_pro/gamma       0.466721                             [illegible]
```

Figure 9.5 Simulation result for justification analysis

9.6 TIME ANALYSIS REPORT :

Table 9.1 Timing report

Parameters	Existing method	Proposed method
Input arrival rate	8.175ns	2.466ns
Output arrival rate	4.664ns	3.762ns
Time response	7.519ns	7.313ns

Thus the test generation time has reduced in case of proposed method.

CHAPTER -10

CONCLUSION AND FUTURE SCOPE

10.1 CONCLUSION :

Thus in this project the behaviour of both the blocks independently and combined are obtained. In this project, a novel method is presented by using a validation test set to generate test sequences that have good stuck-at fault coverage for datapath RTL modules. The scheme first derives the controller behaviors from validation test sequences and reuses them for simplifying justification/propagation analysis corresponding to pre-computed test vectors/responses of datapath RTL modules. A heuristic is used to identify controller behaviors that are compatible with a given set of pre-computed test vectors/responses. It requires only a single pass through the CDFG corresponding to a validation test sequence and is accurate, resulting in a small number of test generation runs. Test generation is performed at the RTL and the controller behavior is prespecified, which results in very small test generation times. The test generation time obtained is said to be less in case of simultaneous operation of both the blocks than the previous method.

10.2 FUTURE SCOPE :

The future scope of this project is that the test generation could be obtained for asynchronous circuits and for the circuits associated with glitches.

REFERENCES

[1] F . FALLAH, P . ASHAR , and S . DEVADAS ,"Simulation vector Generation from HDL descriptions for observability-enhanced Statement coverage, "in.Proc.Des.,Jun . 1996.

[2] R.C.HO and M.A.HOROWITZ,"Validation coverage analysis for complex digital designs," in Proc.Int.Conf.-Aided Des.,Nov.1996.

[3] I.GHOSH , A.RAGHUNATHAN, and N.K.JHA,"A Design for testability technique of RTL circuits," in Proc.Int.Conf.Comput. – Aided Des ., Nov.1996.

[4] W.C.LAI, A.KRSTIC, and K.T.CHENG,"Test program synthesis for path delay in microprocessor cores,"in Proc.Int.Test Conf., Oct.2000.

[5] N.K. JHA, and S. GUPTA, Testing of Digital Systems, Cambridge, U.K.,Cambridge University Press, 2003.

[6] L. CHEN, S. RAVI, A. RAGHUNATHAN, and S. DEY, " A Scalable Software based self- test methodology for programmable processors, " in Proc. Des. Autom. Conf., Jun. 2003, pp. 548 – 553.

[7] D. J. MOUNDANOS, J. A. ABRAHAM, and Y. V. HOSKOTE "Abstraction techniques for validation coverage analysis and test generation,"*IEEE Trans. Comput.*, vol. 47, no. 1, pp. 2–14, Jan. 1998

[8] M. BENJAMIN, D. GEIST, A. HARTMAN, Y. WOLFSTHAL, G. R. SMEETS, "A study in coverage-driven test generation," in Proc. Des. Autom. Conf., Jun. 1999, pp. 970–975.

[9] T. E. MARCHOK, A. EL-MALEH, W. MALY, and J. Rajski, "Complexity of sequential ATPG," in Proc. Eur. Des. Test Conf., Mar. 1995, pp. 252–261.

[10] Synopsys Inc., Santa Clara, CA, "Design compiler," 2007. [Online]. Available: http://www.synopsys.com

[11] W. H. PRESS, B. P. FLANNERY, S. A. TEUKOLSKY, and W. T. VETTERLING, *Numerical Recipes in C*.Cambridge, U.K.: Cambridge University Press, 1993.

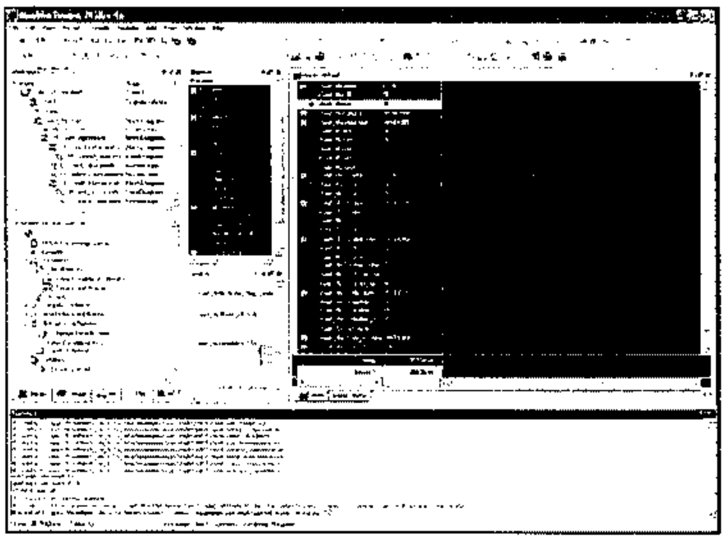
[12] Z. NAVABI, *VHDL: Analysis and Modeling of Digital Systems*. New York: McGraw-Hill, 1993.

APPENDIX I

Design Creation to Realization

ModelSim Designer

D A T A S H E E T



ModelSim Designer combines easy-to-use, flexible creation with a powerful simulation and debug environment

The Easy to Use Solution for the FPGA Designer

ModelSim® Designer is a Windows®-based design environment for FPGAs. It provides an easy to use, advanced-feature tool at an entry-level price. The complete process of creation, management, simulation, and implementation are controlled from a single user interface, facilitating the design and verification flow and providing significant productivity gains.

ModelSim Designer combines the industry leading capabilities of the ModelSim simulator with a built-in design creation engine. To support synthesis and place-and-route, ModelSim Designer is plug-in ready for the synthesis and place-and-route tools of your choice. Connection of these additional tools is easy, giving FPGA designers control of design creation, simulation, synthesis, and place-and-route from a single cockpit.

A flexible methodology conforms to existing design flows. Designers can freely mix text entry of VHDL and Verilog code with graphical entry using block and state diagram editors. A single design unit can be represented in multiple views, and the management of compilation and simulation at all levels of abstraction is a single click away.

Major product features:

- Project manager, version control, and source code templates and wizards
- Block and state diagram editors
- Intuitive graphical waveform editor
- Automated testbench creation
- Text to graphic rendering facilitates analysis
- HTML Documentation option
- VHDL, Verilog and mixed-language simulation
- Optimized native compiled architecture
- Single Kernel Simulator technology
- Advanced debug including Signal Spy™
- Memory window
- Easy-to-use GUI with Tcl interface
- Support for all major synthesis tools and the Actel, Altera, Lattice, and Xilinx place-and-route flows
- Built-in FPGA vendor library compiler
- Full support for Verilog 2001
- Windows platform support

Options: SWIFT interface support, graphics-based Dataflow window, Waveform compare, integrated code coverage, and Profiler (for more details on options, please see the ModelSim SE datasheet)

Mentor Graphics®

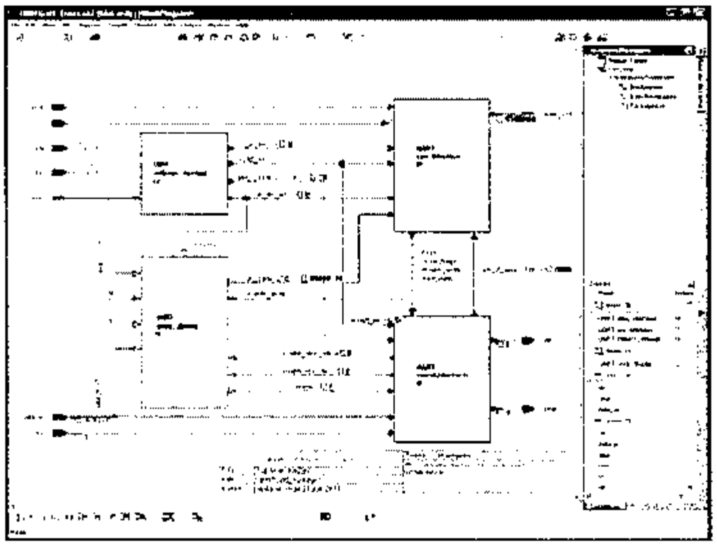
Implementation tasks are achieved through tight integration with the most popular FPGA synthesis and place-and-route tools in the industry. ModelSim Designer can either run these tools directly or externally, via scripts. Either way, the design data is maintained and used in a convenient and consistent way.

Project Manager

The flexible and powerful project manager feature allows easy navigation through a design in order to understand design structure and execute all necessary tasks. The project manager provides easy access to all design data for each individual design unit, throughout the design process. Synthesis results, place-and-route results, back-annotated netlists, and SDF files are associated with the relevant design unit. Project archiving is a few simple clicks away, allowing complete version control management of designs. The project manager also stores user-generated design documents such as Word, PowerPoint, and PDFs.

Templates and Wizards

Creation wizards walk users through the creation of VHDL and Verilog design units, using either text or graphics. In the case of the graphical editor, HDL code is generated from the graphical diagrams created. For text-based design, VHDL and Verilog templates and wizards help engineers quickly develop HDL code without having to remember the exact language syntax. The wizards show how to create parameterizable logic blocks, testbench stimuli, and design objects. Novice and advanced HDL developers both benefit from time-saving shortcuts.


The Block Diagram editor is a convenient way to visually partition your design and have the HDL code automatically generated.

Intuitive Graphical Editors

ModelSim Designer includes block diagram and state machine editors that ensure a consistent coding style, facilitating design reuse and maintenance. These editors use an intuitive graphical methodology that flattens the learning curve. This shortens the time to productivity for designers who are migrating to HDL methodologies or changing their primary design language.

Designers can automatically view or render diagrams from HDL code in block diagrams or state machines. When the code changes, the diagram can be updated instantly with its accuracy ensured. This helps designers understand legacy designs and aids in the debugging of current designs.

Automated Testbench Creation

ModelSim Designer offers an automated mechanism for testbench generation. The testbench wizard generates VHDL or Verilog code through a graphical waveform editor, with output in either HDL or a TCL script. Users can manually define signals in the waveform editor or use the built-in wizard to define the waveforms. Either way, it is intuitive and easy to use and saves considerable time.

Active Design Visualization Enhances Simulation Debugging

During live simulation, design analysis capabilities are enhanced through graphical design views. From any diagram window, simulations can be fully executed and controlled. Enhanced debugging features include graphical breakpoints, signal probing, graphics-to-text-source cross-highlighting, animation, and cause analysis. The ability to overlay live simulation results in a graphical context speeds up the debug process by allowing faster problem discovery and shorter design iterations.

HTML Documentation

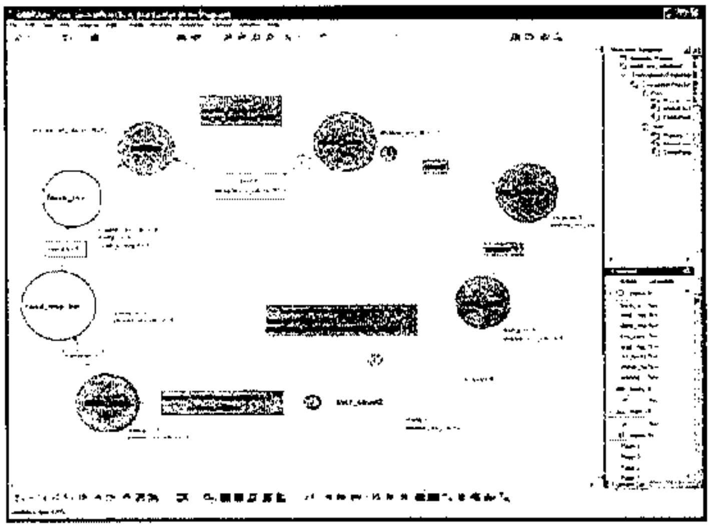
The HTML (HyperText Markup Language) Documentation option allows the user to output the complete design in HTML format. This allows the graphics, HDL, associative design files, and design hierarchy to be shared with anybody that uses an HTML browser. Design hierarchy and details can be viewed and navigated in any HTML browser, aiding communication and documentation.

Memory Window

The memory window enables flexible viewing and switching of memory locations. VHDL and Verilog memories are auto-extracted in the GUI, delivering powerful search, fill, load, and save functionality. The memory window allows pre-loading of memories, thus saving the time-consuming step of initializing sections of the simulation to load memories. All functions are available via the command line, making them available for scripting.

Intelligent GUI

An intelligently engineered GUI makes efficient use of desktop real estate. ModelSim Designer's intuitive layout of graphical elements (windows, toolbars, menus, etc.) makes it easy to step through the design flow. There are also wizards in place which help set up the design environment and make going through the design process seamless and efficient.


The State Diagram Editor uses interactive animation to track the simulation through the state machine and to check that all states were exercised.

Synthesis and Place-and-Route Integration

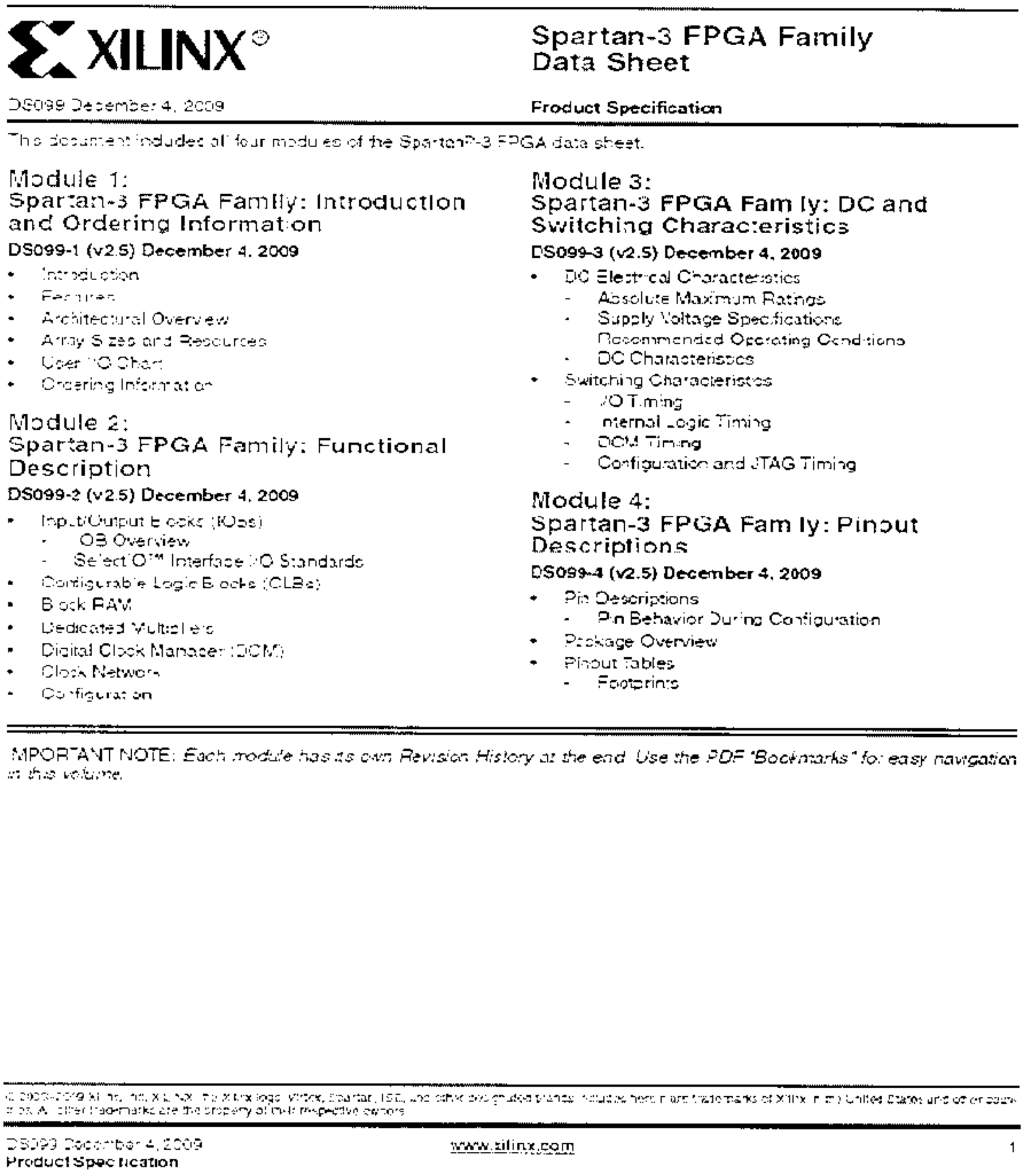
The industry's popular FPGA synthesis tools, such as Mentor Graphics Precision-RTL and most FPGA vendor synthesis tools, can be integrated into ModelSim Designer with push button convenience. Actel Designer and Libero, Altera Quartus, Lattice ispLEVER, and Xilinx ISE place-and-route software are similarly integrated. Place-and-route results, together with SDF information, are automatically managed by ModelSim Designer after the process is complete, making them ready for post-place-and-route, gate level simulation.

FPGA Vendor Library Compiler

ModelSim Designer provides an intuitive mechanism to compile the necessary vendor libraries for post-place-and-route simulation. The compiler detects which FPGA vendor tools have been installed and compiles the necessary libraries as soon as the tool is launched, taking care of this step once and for all.

Visit our web site at www.model.com/modelsimdesigner for more information.

APPENDIX II



XILINX®

Spartan-3 FPGA Family Data Sheet

DS099 December 4, 2009

Product Specification

This document includes all four modules of the Spartan®-3 FPGA data sheet.

Module 1: Spartan-3 FPGA Family: Introduction and Ordering Information

DS099-1 (v2.5) December 4, 2009

- Introduction
- Features
- Architectural Overview
- Array Sizes and Resources
- User I/O Chart
- Ordering Information

Module 2: Spartan-3 FPGA Family: Functional Description

DS099-2 (v2.5) December 4, 2009

- Input/Output Blocks (IOBs)
  - IOB Overview
  - SelectIO™ Interface I/O Standards
- Configurable Logic Blocks (CLBs)
- Block RAM
- Dedicated Multipliers
- Digital Clock Manager (DCM)
- Clock Network
- Configuration

Module 3: Spartan-3 FPGA Family: DC and Switching Characteristics

DS099-3 (v2.5) December 4, 2009

- DC Electrical Characteristics
  - Absolute Maximum Ratings
  - Supply Voltage Specifications
    Recommended Operating Conditions
  - DC Characteristics
- Switching Characteristics
  - I/O Timing
  - Internal Logic Timing
  - DCM Timing
  - Configuration and JTAG Timing

Module 4: Spartan-3 FPGA Family: Pinout Descriptions

DS099-4 (v2.5) December 4, 2009

- Pin Descriptions
  - Pin Behavior During Configuration
- Package Overview
- Pinout Tables
  - Footprints

IMPORTANT NOTE: *Each module has its own Revision History at the end. Use the PDF "Bookmarks" for easy navigation in this volume.*

XILINX®

Spartan-3 FPGA Family: Introduction and Ordering Information

DS099-1 (v2.5) December 4, 2009

Product Specification

Introduction

The Spartan®-3 family of Field-Programmable Gate Arrays is specifically designed to meet the needs of high volume, cost-sensitive consumer electronic applications. The eight-member family offers densities ranging from 50,000 to five million system gates, as shown in Table 1.

The Spartan-3 family builds on the success of the earlier Spartan-IIE family by increasing the amount of logic resources, the capacity of internal RAM, the total number of I/Os, and the overall level of performance as well as by improving clock management functions. Numerous enhancements derive from the Virtex®-II platform technology. These Spartan-3 FPGA enhancements, combined with advanced process technology, deliver more functionality and bandwidth per dollar than was previously possible, setting new standards in the programmable logic industry.

Because of their exceptionally low cost, Spartan-3 FPGAs are ideally suited to a wide range of consumer electronics applications, including broadband access, home networking, display/projection and digital television equipment.

The Spartan-3 family is a superior alternative to mask programmed ASICs. FPGAs avoid the high initial cost, the lengthy development cycles, and the inherent inflexibility of conventional ASICs. Also, FPGA programmability permits design upgrades in the field with no hardware replacement necessary, an impossibility with ASICs.

Features

- Low-cost, high-performance logic solution for high-volume consumer-oriented applications
  - Densities up to 74,880 logic cells
- SelectIO™ interface signaling
  - Up to 633 I/O pins
  - 622+ Mb/s data transfer rate per I/O
  - 18 single-ended signal standards
  - 8 differential I/O standards including LVDS, RSDS
  - Termination by Digitally Controlled Impedance
  - Signal swing ranging from 1.14V to 3.465V
  - Double Data Rate (DDR) support
  - **DDR, DDR2 SDRAM support** up to 333 Mbps
- Logic resources
  - Abundant logic cells with shift register capability
  - Wide, fast multiplexers
  - Fast look-ahead carry logic
  - Dedicated 18 x 18 multipliers
  - JTAG logic compatible with IEEE 1149.1/1532
- SelectRAM™ hierarchical memory
  - Up to 1,872 Kbits of total block RAM
  - Up to 520 Kbits of total distributed RAM
- Digital Clock Manager (up to four DCMs)
  - Clock skew elimination
  - Frequency synthesis
  - High resolution phase shifting
- Eight global clock lines and abundant routing
- Fully supported by **Xilinx ISE®** and **WebPACK™** software development systems
- **MicroBlaze™** and **PicoBlaze™** processor, **PCI®**, **PCI Express® PIPE Endpoint**, and other IP cores
- Pb-free packaging options
- Automotive **Spartan-3 XA Family** variant

Table 1: **Summary of Spartan-3 FPGA Attributes**

Device	System Gates	Equivalent Logic Cells[1]	CLB Array (One CLB = Four Slices)			Distributed RAM Bits (K=1024)	Block RAM Bits (K=1024)	Dedicated Multipliers	DCMs	Maximum User I/O	Maximum Differential I/O Pairs
			Rows	Columns	Total CLBs						
XC3S50[2]	50K	1,728	16	12	192	12K	72K	4	2	124	56
XC3S200[2]	200K	4,320	24	20	480	30K	216K	12	4	173	76
XC3S400[2]	400K	8,064	32	28	896	56K	288K	16	4	264	116
XC3S1000[2]	1M	17,280	48	40	1,920	120K	432K	24	4	391	175
XC3S1500	1.5M	29,952	64	52	3,328	208K	576K	32	4	487	221
XC3S2000	2M	46,080	80	64	5,120	320K	720K	40	4	565	270
XC3S4000	4M	62,208	96	72	6,912	432K	1,728K	96	4	633	300
XC3S5000	5M	74,880	104	80	8,320	520K	1,872K	104	4	633	300

Notes:
1. Logic Cell = 4-input Look-Up Table (LUT) plus a 'D' flip-flop. "Equivalent Logic Cells" equals "Total CLBs" x 8 Logic Cells/CLB x 1.125 effectiveness.
2. These devices are available in Xilinx Automotive versions as described in **DS314**: Spartan-3 Automotive XA FPGA Family.

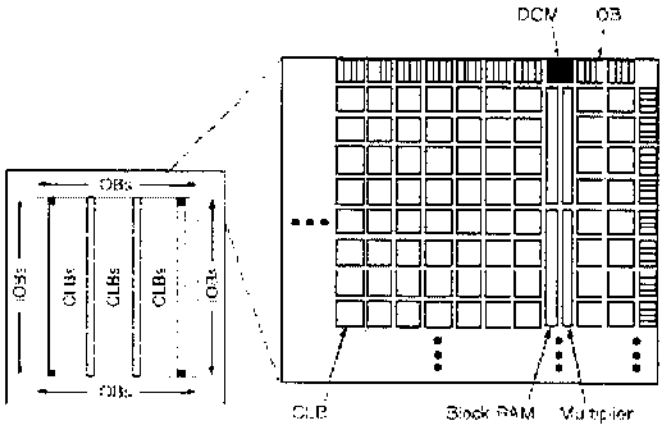
Architectural Overview

The Spartan-3 family architecture consists of five fundamental programmable functional elements:

- Configurable Logic Blocks (CLBs) contain RAM-based Look-Up Tables (LUTs) to implement logic and storage elements that can be used as flip-flops or latches. CLBs can be programmed to perform a wide variety of logical functions as well as to store data.
- Input/Output Blocks (IOBs) control the flow of data between the I/O pins and the internal logic of the device. Each IOB supports bidirectional data flow plus 3-state operation. Twenty-six different signal standards, including eight high-performance differential standards, are available as shown in Table 2. Double Data-Rate (DDR) registers are included. The Digitally Controlled Impedance (DCI) feature provides automatic on-chip terminations, simplifying board designs.
- Block RAM provides data storage in the form of 18-Kbit dual-port blocks.
- Multiplier blocks accept two 18-bit binary numbers as inputs and calculate the product.
- Digital Clock Manager (DCM) blocks provide self-calibrating, fully digital solutions for distributing, delaying, multiplying, dividing, and phase shifting clock signals.

These elements are organized as shown in Figure 1. A ring of IOBs surrounds a regular array of CLBs. The XC3S50 has a single column of block RAM embedded in the array. Those devices ranging from the XC3S200 to the XC3S2000 have two columns of block RAM. The XC3S4000 and XC3S5000 devices have four RAM columns. Each column is made up of several 18-Kbit RAM blocks; each block is associated with a dedicated multiplier. The DCMs are positioned at the ends of the outer block RAM columns.

The Spartan-3 family features a rich network of traces and switches that interconnect all five functional elements, transmitting signals among them. Each functional element has an associated switch matrix that permits multiple connections to the routing.



Notes
1. The two additional block RAM columns of the XC3S4000 and XC3S5000 devices are shown with dashed lines. The XC3S50 has only the block RAM column on the far left.

Figure 1: **Spartan-3 Family Architecture**

Configuration

Spartan-3 FPGAs are programmed by loading configuration data into robust, reprogrammable, static CMOS configuration latches (CCLs) that collectively control all functional elements and routing resources. Before powering on the FPGA, configuration data is stored externally in a PROM or some other nonvolatile medium either on or off the board. After applying power, the configuration data is written to the FPGA using any of five different modes: Master Parallel, Slave Parallel, Master Serial, Slave Serial, and Boundary Scan (JTAG). The Master and Slave Parallel modes use an 8-bit-wide SelectMAP port.

The recommended memory for storing the configuration data is the low-cost Xilinx Platform Flash PROM family, which includes the XCF00S PROMs for serial configuration and the higher density XCF00P PROMs for parallel or serial configuration.

I/O Capabilities

The SelectIO feature of Spartan-3 devices supports 18 single-ended standards and 8 differential standards as listed in Table 2. Many standards support the DCI feature, which uses integrated terminations to eliminate unwanted signal reflections.

Table 2: **Signal Standards Supported by the Spartan-3 Family**

Standard Category	Description	V_{CCO} (V)	Class	Symbol (IOSTANDARD)	DCI Option
Single-Ended					
GTL	Gunning Transceiver Logic	N/A	Terminated	GTL	Yes
			Plus	GTLP	Yes
HSTL	High-Speed Transceiver Logic	1.5	I	HSTL_I	Yes
			III	HSTL_III	Yes
		1.8	I	HSTL_I_18	Yes
				HSTL_II_18	Yes
			III	HSTL_III_18	Yes
LVCMOS	Low-Voltage CMOS	1.2	N/A	LVCMOS12	No
		1.5	N/A	LVCMOS15	Yes
		1.8	N/A	LVCMOS18	Yes
		2.5	N/A	LVCMOS25	Yes
		3.3	N/A	LVCMOS33	Yes
LVTTL	Low-Voltage Transistor-Transistor Logic	3.3	N/A	LVTTL	No
PCI	Peripheral Component Interconnect	3.0	33 MHz[1]	PCI33_3	No
SSTL	Stub Series Terminated Logic	1.8	N/A (±6.7 mA)	SSTL18_I	Yes
			N/A (±13.4 mA)	SSTL18_II	No
		2.5	I	SSTL2_I	Yes
				SSTL2_II	Yes
Differential					
LDT (ULVDS)	Lightning Data Transport (HyperTransport™) Logic	2.5	N/A	LDT_25	No
LVDS	Low-Voltage Differential Signaling		Standard	LVDS_25	Yes
			Bus	BLVDS_25	No
			Extended Mode	LVDSEXT_25	Yes
LVPECL	Low-Voltage Positive Emitter-Coupled Logic	2.5	N/A	LVPECL_25	No
RSDS	Reduced-Swing Differential Signaling	2.5	N/A	RSDS_25	No
HSTL	Differential High-Speed Transceiver Logic	1.8		DIFF_HSTL_II_18	Yes
SSTL	Differential Stub Series Terminated Logic	2.5		DIFF_SSTL2_II	Yes

Notes:
1. 66 MHz PCI is not supported by the Xilinx IP core although PCI66_3 is an available I/O standard.

Table 3 shows the number of user I/Os as well as the number of differential I/O pairs available for each device-package combination.

Table 3: **Spartan-3 Device I/O Chart**

Package	VQ100 VQG100		CP132[1] CPG132		TQ144 TQG144		PQ208 PQG208		FT256 FTG256		FG320 FGG320		FG456 FGG456		FG676 FGG676		FG900 FGG900		FG1156[1] FGG1156	
Footprint (mm)	16 x 16		8 x 8		22 x 22		30.6 x 30.6		17 x 17		19 x 19		23 x 23		27 x 27		31 x 31		35 x 35	
	User	Diff	User	Diff	User	Diff	User	Diff	User	Diff	User	Diff	User	Diff	User	Diff	User	Diff	User	Diff
XC3S50	63	29	89[1]	44[1]	97	46	124	56	-	-	-	-	-	-	-	-	-	-	-	-
XC3S200	63	29	-	-	97	46	141	62	173	76	-	-	-	-	-	-	-	-	-	-
XC3S400	-	-	-	-	97	46	141	62	173	76	221	100	264	116	-	-	-	-	-	-
XC3S1000	-	-	-	-	-	-	-	-	173	76	221	100	333	149	391	175	-	-	-	-
XC3S1500	-	-	-	-	-	-	-	-	-	-	221	100	333	149	487	221	-	-	-	-
XC3S2000	-	-	-	-	-	-	-	-	-	-	-	-	333	149	489	221	565	270	-	-
XC3S4000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	489	221	633	300	712[1]	312[1]
XC3S5000	-	-	-	-	-	-	-	-	-	-	-	-	-	-	489	221	633	300	784[1]	344[1]

Notes:
1. The CP132, CPG132, FG1156, and FGG1156 packages are being discontinued and are not recommended for new designs. See http://www.xilinx.com/support/documentation/spartan-3_customer_notices.htm for the latest updates.
2. All device options listed in a given package column are pin-compatible.
3. User = Single-ended user I/O pins. Diff = Differential I/O pairs.

Package Marking

[illegible] shows the top marking for Spartan-3 FPGAs in the quad-flat packages. [illegible] shows the top marking for Spartan-3 FPGAs in BGA packages except the 132-ball chip-scale package (CP132 and CPG132). The markings for the BGA packages are nearly identical to those for the quad-flat packages, except that the marking is rotated with respect to the ball A1 indicator. [illegible] shows the top marking for Spartan-3 FPGAs in the CP132 and CPG132 packages.

The "5C" and "4I" part combinations may be dual marked as "5C/4I". Devices with the dual mark can be used as either -5C or -4I devices. Devices with a single mark are only guaranteed for the marked speed grade and temperature range. Some specifications vary according to mask revision. Mask revision E devices are errata-free. All shipments since 2006 have been mask revision E.

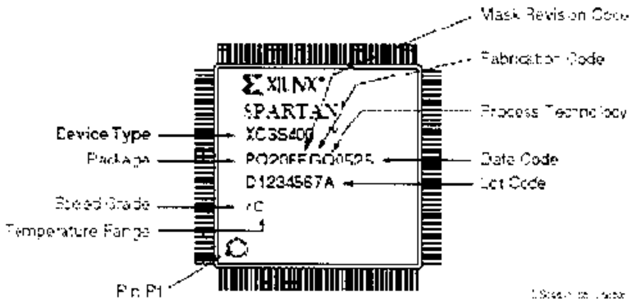


Figure 2: **Spartan-3 QFP Package Marking Example for Part Number XC3S400-4PQ208C**

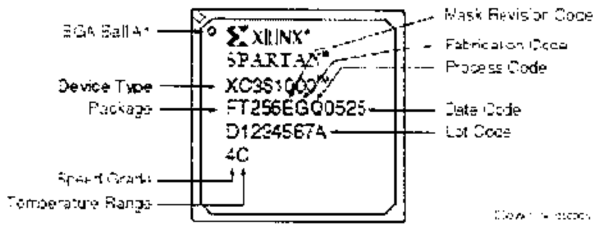


Figure 3: **Spartan-3 BGA Package Marking Example for Part Number XC3S1000-4FT256C**

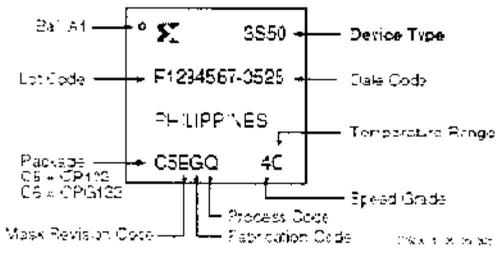
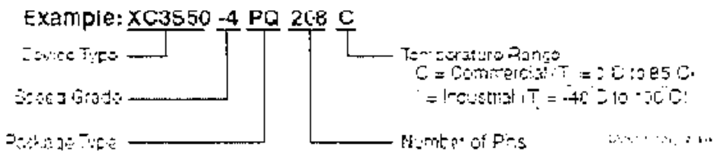


Figure 4: **Spartan-3 CP132 and CPG132 Package Marking Example for XC3S50-4CP132C**

Ordering Information

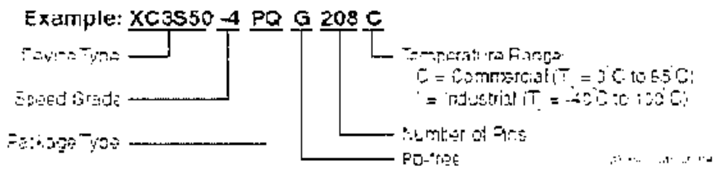
Spartan-3 FPGAs are available in both standard and Pb-free packaging options for all device/package combinations. The Pb-free packages include a special 'G' character in the ordering code.

Standard Packaging


Example: XC3S50 -4 PQ 208 C
Device Type
Speed Grade
Package Type
Temperature Range
C = Commercial (T_J = 0°C to 85°C)
I = Industrial (T_J = -40°C to 100°C)
Number of Pins

Pb-Free Packaging

For additional information on Pb-free packaging, see **XAPP427**: "Implementation and Solder Reflow Guidelines for Pb-Free Packages".


Example: XC3S50 -4 PQ G 208 C
Device Type
Speed Grade
Package Type
Temperature Range
C = Commercial (T_J = 0°C to 85°C)
I = Industrial (T_J = -40°C to 100°C)
Number of Pins
Pb-free

Device	Speed Grade		Package Type / Number of Pins		Temperature Range (T_J)	
XC3S50	-4	Standard Performance	VQ(G)100	100-pin Very Thin Quad Flat Pack (VQFP)	C	Commercial (0°C to 85°C)
XC3S200	-5	High Performance[1]	CP(G)132[2]	132-pin Chip-Scale Package (CSP)	I	Industrial (–40°C to 100°C)
XC3S400			TQ(G)144	144-pin Thin Quad Flat Pack (TQFP)		
XC3S1000			PQ(G)208	208-pin Plastic Quad Flat Pack (PQFP)		
XC3S1500			FT(G)256	256-ball Fine-Pitch Thin Ball Grid Array (FTBGA)		
XC3S2000			FG(G)320	320-ball Fine-Pitch Ball Grid Array (FBGA)		
XC3S4000			FG(G)456	456-ball Fine-Pitch Ball Grid Array (FBGA)		
XC3S5000			FG(G)676	676-ball Fine-Pitch Ball Grid Array (FBGA)		
			FG(G)900	900-ball Fine-Pitch Ball Grid Array (FBGA)		
			FG(G)1156[2]	1156-ball Fine-Pitch Ball Grid Array (FBGA)		

Notes:

1. The -5 speed grade is exclusively available in the Commercial temperature range.
2. The CP132, CPG132, FG1156, and FGG1156 packages are being discontinued and are not recommended for new designs. See http://www.xilinx.com/support/documentation/spartan-3_customer_notices.htm for the latest updates.

Revision History

Date	Version No.	Description
04/11/03	1.0	Initial Xilinx release.
04/24/03	1.1	Updated block RAM, DCM, and multiplier counts for the XC3S50.
12/24/03	1.2	Added the FG320 package.
07/13/04	1.3	Added information on Pb-free packaging options.
01/17/05	1.4	Referenced Spartan-3 XA Automotive FPGA families in Table 1. Added XC3S50CP132, XC3S2000FG456, XC3S4000FG676 options to Table 5. Updated Package Marking to show mask revision code, fabrication facility code, and process technology code.
08/19/05	1.5	Added package markings for BGA packages (Figure 3) and CP132/CPG132 packages (Figure 4). Added differential (complementary single-ended) HSTL and SSTL I/O standards.
04/03/06	2.0	Increased number of supported single-ended and differential I/O standards.
04/26/06	2.1	Updated document links.
05/25/07	2.2	Updated Package Marking to allow for dual-marking.
11/30/07	2.3	Added XC3S5000 FG(G)676 to Table 7. Noted that FG(G)1156 package is being discontinued and updated max I/O count.
05/29/08	2.4	Updated max I/O counts based on FG1156 discontinuation. Clarified dual mark in Package Marking. Updated formatting and links.
12/04/09	2.5	CP132 and CPG132 packages are being discontinued. Added link to Spartan-3 FPGA customer notices. Updated Table 5 with package footprint dimensions.