STATIC CHECKS IN RTL DESIGN: ENSURING CODE QUALITY AND RELIABILITY

by Nikita Puri

Submission date: 27-May-2026 02:16PM (UTC+0530)
Submission ID: 2970464838
File name: nikita_thesis_final.pdf (1.18M)
Word count: 7235
Character count: 42247

STATIC CHECKS IN RTL DESIGN: ENSURING CODE QUALITY AND RELIABILITY

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE AWARD OF THE DEGREE

OF

MASTER OF TECHNOLOGY

IN

VLSI AND EMBEDDED SYSTEMS

Submitted by:

Nikita Puri

(24/VLS/09)

Under the supervision of

Dr. Sonam Rewari

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi 110042

MAY, 2026

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

I **Nikita Puri**, 24/VLS/09 students of M.Tech VLSI and Embedded Systems, Electronics & Communication Engineering, hereby declare that the project Dissertation titled "Static checks in RTL design : ensuring code quality and reliability" which is submitted by me to the department of Electronics & Communication Engineering, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Candidate's signature

This is to certify that the student has incorporated all the corrections suggested by the examiners in the thesis and the statement made by the candidate is correct to the best of our knowledge.

Signature of Supervisor (s) Signature of External Examiner

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the Project dissertation titled "Static checks in RTL design : ensuring code quality and reliability", submitted by Nikita Puri (24/VLS/09), Department of Electronics & Communication Engineering, Delhi Technological University, Delhi, in a partial fulfillment of the requirement for the award of the Degree of Master of Technology, is a record of the project work carried out by the student under my guidance and supervision. To the best of my knowledge, this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi

Date:

Dr. Sonam Rewari

Assistant professor

(SUPERVISOR)

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

ACKNOWLEDGEMENT

The completion of this dissertation has depended on the guidance, generosity and patience of several people, and it's a genuine pleasure to acknowledge them here. My sincere gratitude goes first to **Prof. (Dr.) Neeta Pandey**, HOD of ECE department, DTU, for maintaining an academic environment that encourages students to take on substantive research problems.

The largest debt of gratitude belongs to my supervisor, **Dr. Sonam Rewari**, Asst. Professor, Department of ECE, DTU, for her continuous guidance and mentorship that she provided me during the project. She showed me the path to achieve the targets by explaining all the tasks to be done and explained the importance of this project. She was always ready to help and clear my doubts regarding any hurdles in this project. Without her constant support and motivation, this project would not have been successful.

Nikita Puri

(24/VLS/09)

Abstract

Keywords: System-on-Chip (SoC), Clock Domain Crossing (CDC), Register Transfer Level (RTL) Linting, Static Verification, Metastability, Mean Time Between Failures (MTBF), Hierarchical Verification Flow, VLSI Design Reliability.

The relentless scaling of semiconductor technology and the advent of highly heterogeneous System-on-Chip (SoC) architectures—integrating multi-core processors, high-speed interfaces, and specialized accelerators—have necessitated the use of numerous asynchronous clock domains. While this Globally Asynchronous, Locally Synchronous (GALS) paradigm optimizes power and performance, it introduces severe design vulnerabilities related to Clock Domain Crossing (CDC) and Reset Domain Crossing (RDC). During asynchronous data transfers, destination flip-flops are highly susceptible to setup and hold time violations, resulting in metastability. If not manageable, this metastable state propagates through downstream logic, causing unpredictable state corruption, signal glitches, and catastrophic system deadlocks. Traditional flat-level static verification methodologies struggle to scale with multi-million-gate designs. They suffer from severe memory explosion, excessive execution runtimes, and the generation of an overwhelming volume of false-positive violations. This immense "verification noise" effectively masks genuine, critical design bugs during the crucial SoC integration phase.

To overcome these bottlenecks, this thesis presents an automated, hierarchical static verification framework engineered to enforce strict Register Transfer Level (RTL) code quality and guarantee highly reliable cross-domain synchronization. The methodology initiates with an automated linting protocol that acts as a rigorous first line of defense. By proactively identifying and resolving structural synthesis blockers, combinational loops, and simulation-synthesis mismatches early in the

design cycle, the framework reduces structurally actionable bugs by 94% prior to dynamic functional simulation. Furthermore, the research details the implementation of a hierarchical CDC verification flow. By utilizing validated, parameterized IP-level abstract models that mask internal combinatorial depth while preserving boundary intent, the methodology fundamentally resolves the scalability crisis of massive SoCs. This structural transition reduced top-level CDC execution runtime by 82%—dropping from an average of 14.5 hours to merely 2.5 hours—and successfully suppressed hundreds of thousands of false-positive IP-level violations. Consequently, integration engineers can seamlessly isolate and resolve genuine top-level hazards, such as missing synchronizers, data loss, and logic re-convergence.

Additionally, this research establishes an analytical framework for the optimization of Mean Time Between Failures (MTBF) across critical synchronizer chains. By mathematically evaluating the structural synchronization depth, the study balances the exponential gains in system reliability against the linear penalties of increased hardware latency and silicon area, ensuring the SoC meets stringent multi-decade commercial and automotive safety standards. Ultimately, this comprehensive framework achieves a true "shift-left" in the VLSI verification lifecycle. It significantly minimizes manual debugging efforts, ensures mathematically sound CDC sign-off, accelerates time-to-market, and proactively prevents the catastrophic financial impact of late-stage silicon re-spins.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1.1 Background: The Evolution of SoC Complexity and Multi-Clock Designs

The semiconductor industry has experienced exponential growth in integrated circuit density, which has primarily adhered to Moore's law. It is also true that the way the digital design has been conceived has changed dramatically from single ASIC design to highly complicated SoC design methodology [1]. A typical modern SoC will consists of billions of transistors embedded in one silicon chip of an SoC containing many types of IP core such as multi-core CPU, DSP,Memory controllers and communication interfaces(PCIe, USB, Ethernet), etc.

Because modern SoCs are mandated to fulfill ambitious Power Performance and Area(PPA) goals, it will not be practical to drive all the IPs under one global clock. Each IP will have a unique processing requirement. Thus while the application such as CPU core may require gigahertz range frequency to deliver top performance, some IPs such as bus controller may operate in megahertz range to consume low dynamic power. Hence modern SoCs are partitioned into various clock and reset domains.

While it makes the system perform better, partitioning of a SoC into various clock domains poses a problem as far as data transfer is concerned [2]. If one need to transfer an electrical signal from a source register that operate under one clock domain to a destination register which operate under another clock domain, then it is clock domain crossing. As there exists no known and stable relationship between the two clock domains, the signal is likely to have timing related issues, thus CDC becomes the one of the most difficult problems of VLSI verification.

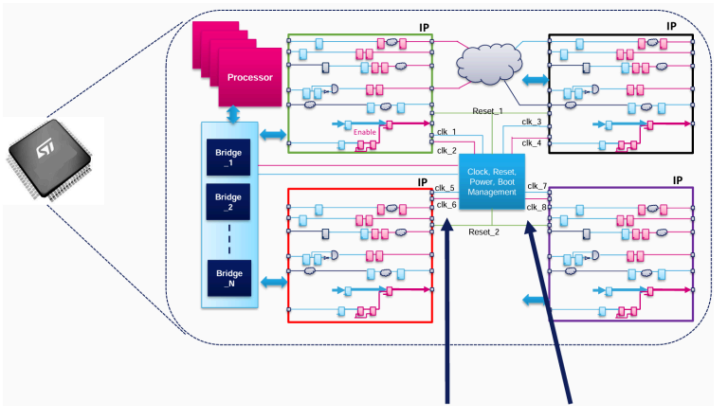

Figure 1.1: Modern SoC Architecture with Multiple Asynchronous Clock Domains

1.2 Understanding RTL Quality and the Role of Static Checks

The backbone of any well-designed physical chip is the RTL code (usually written in a Hardware Description Language such as Verilog, SystemVerilog or VHDL) which describes its intended function at a structural or behavioral level. The quality of this RTL will directly determine how well the downstream flow can handle logic synthesis, placement and routing (PR) and meet timing closure, and structural faults at RTL, if allowed to go undetected, can result in fatal functional failure of silicon, causing extremely expensive silicon re-spins that can run into millions of dollars and cause time to market delays. To alleviate this problem, Static Verification, dominated by RTL Linting [3] is critical in the VLSI domain. Unlike dynamic verification which involves huge amounts of testbench development and tremendous amounts of simulation runtime to toggle the nodes in the design, static verification algorithmically and topologically checks the design. Linting tools, for example Synopsys SpyGlass, read the RTL and detect abnormal coding patterns, syntax errors and synthesizing issues. Linting tools are the "first line of defense" and can flag the structural design issues like multiple drivers on a single net, combinational loops, latches inferred, unintended simulation/synthesis

mismatches [4]. By implementing an automated and disciplined linting strategy, the RTL should be mathematically and structurally well-formed before being fed into the heavy-lifting functional simulation or backend synthesis.

1.3 The Phenomenon of Metastability in Digital Circuits

Metastability is the most important vulnerability that asynchronous clock domains introduce [5]. A conventional flip-flop in a synchronous digital circuit demands that the incoming data signal is held stable for a given time before the active clock edge (Setup Time, $Tsetup$) and for a certain period after the clock edge (Hold Time, $Thold$).

However, since the launch and capture clock in a CDC interface are completely asynchronous, the data signal will inevitably cross while it is within the sensitive set-up and hold window of the destination flip-flop. When this setup-hold violation happens, the internal transistors of the flip-flop fail to resolve to a definite logic level (0' or 1') by the required clock-to-Q propagation delay and hence, the output voltage is somewhere between 0' and 1', it oscillates and settles to a definitive logic level later. This state is known as metastability.

If a metastable signal propagates into downstream combinatorial logic, it can be interpreted as a '0' by some logic gates and a '1' by others, leading to state machine corruption and system deadlocks. While the occurrence of metastability in asynchronous crossings is governed by physics and cannot be entirely prevented, its probability of causing a system failure—quantified as the Mean Time Between Failures (MTBF)—must be strictly managed. This is achieved by deploying specialized synchronizer circuits (e.g., multi-stage flip-flop synchronizers, asynchronous FIFOs, and handshake protocols) and validating them through rigorous static CDC analysis.

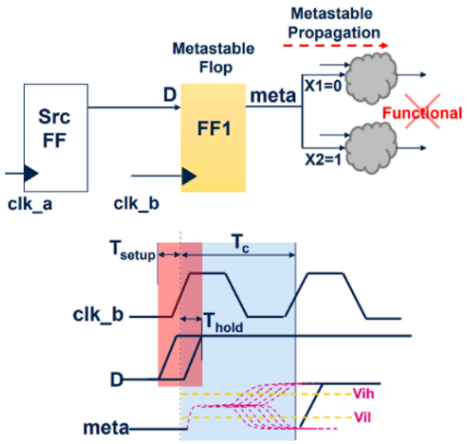

Figure 1.2: Setup and Hold time violation leading to a Metastable State

1.4 Problem Statement

Despite the existence of sophisticated static verification tools [6], evaluating CDC and RTL quality in modern, multi-million-gate SoCs presents a massive scalability bottleneck. Traditionally, the industry has relied on a "flat" verification approach, wherein the static analysis tool digests the entire SoC database in a single run.
Although mature static verification tools exist [6], scaling verification of Clock Domain Crossing (CDC) and RTL quality to modern multi-million-gate Systems-on-Chip (SoCs) is an overwhelming bottleneck. The ongoing push towards heterogeneous computing—integrating AI accelerators, complex network-on-chip (NoC) designs, and dozens of third-party IP—has resulted in systems with hundreds of asynchronous, distinct clock domains. Industry has long been constrained by a "flat" verification methodology where the static analysis tool ingests the entire SoC database, synthesizes the global clock trees and structurally analyses all signal paths in a single, monolithic run.
Flat CDC runs in 100M/1B gates are not viable in terms of computational constraints.

- **Graph Complexity:** The static tool must construct large mathematical

graphs to trace paths across domains. During a flat run, this graph expands exponentially, saturating memory with extreme use of RAM, quickly exceeding available memory on typical compute servers.

- **Execution Time:** Iterative turnarounds are critical in RTL design. Flat analyses can require days of computation per run. The resulting slow feedback loop delays the RTL-freeze milestone significantly, bottlenecking the overall verification schedule, and forcing the engineering team into a reactive debugging process.

Beyond compute limitations, the quantity of output generated is the most pressing issue. Flat analyses typically output overwhelming amounts of structural violations often hundreds of thousands in quantity.

- **Contextual Mismatch:** A vast majority of the violations are purely internal IP issues. For the top-level integrator whose concern is the reliable handoff of signals between IPs and the SoC interconnect, internal IP-level violations are irrelevant.
- **False Positives:** Legitimate design practices such as quasi-static configuration registers, software-managed asynchronous handshakes, or statically-disabled test-mode logic will be reported as violations by the tool as it lacks global context.
- **Verification Fatigue:** This phenomenon results in excess "verification noise." It is impossible for an engineer to distinguish real SoC boundary bugs like missing synchronizers or unsafe convergence logic (leading to critical metastability in silicon) from fabricated violations.

Teams have historically relied on manual waivers to filter known false-positives and internal IP paths. This solution is very brittle, however:

- **Human Error:** Manually reviewing and waiving tens of thousands of paths is tiresome and highly susceptible to human error. A single wildcard misplaced in a waiver file can accidentally hide a critical CDC violation.

- **Stale Waivers:** Manually-maintained waiver files are impossible to keep up-to-date in the fast-paced RTL development flow with daily commits, resulting in either the reintroduction of noise or the over-waiving of newly added logic.

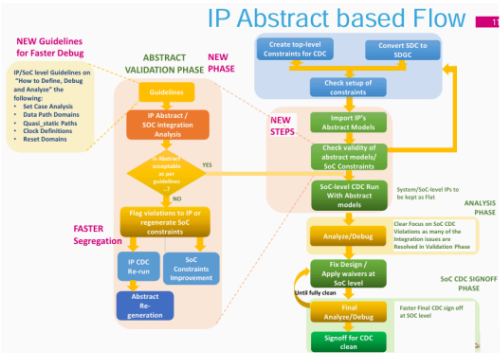

Figure 1.3: The Verification Bottleneck: Flat Flow Analysis vs. Hierarchical Flow

Given the prohibitive cost of silicon respins resulting from undetected metastability bugs, there is an overwhelming need in the industry for an alternative to flat verification: automated hierarchical verification.

A modern hierarchical flow eliminates bottlenecks by following a divide and conquer methodology:

1. **IP-Level Signoff and Abstraction:** IP blocks are verified in isolation by their respective owners. Upon successful verification, a lightweight CDC abstract model is generated. The abstract model retains only boundary constraints, clock domain information, and boundary synchronizers, discarding internal IP detail.

2. **Top-Level Integration:** The SoC integrator analyzes these abstracts using the CDC tool, instead of the full RTL. The gate count drops dramatically from the 100M+ gate range down to the level of the abstract models.

3. **Intelligent Automation:** This hierarchical flow is enabled by advanced automation which generates abstracts automatically, migrates waivers from

IP to SoC level automatically, and uses machine learning heuristics to group and characterize boundary violations remaining.

By eliminating memory bottlenecks and filtering out unnecessary noise, an automated hierarchical flow restores visibility to the SoC integration engineer, making it possible to achieve high-quality, bug-free silicon within demanding time-to-market targets.

1.5 Research Objectives

To solve the problems listed above, this research intends to perform the following:

- **Automated verification framework:** Implement and test a highly automated RTL linting and static CDC verification framework that can enforce coding rules and prevent synthesis blockers at an earlier stage.
- **Hierarchical CDC flow:** Design and simulate a hierarchical CDC flow that uses abstract model of tested IPs to separate IP level bugs and SoC integration problems. This approach can reduce the execution time and the noise for verification.
- **Metastability analysis and optimization:** systematically review synchronization structure and optimize MTBF to reach rigorous industry standards of reliability.
- **Quality and Productivity evaluation:** demonstrate quantitative improvements of debugging effort, execution time and code quality by replacing the flat approach with this hierarchical framework.

1.6 Organization of the Thesis

The remainder of this thesis is logically organized as follows, guiding you through the theoretical backgrounds, the methodology, and results. In **Chapter 2**, an intensive literature review is performed to outline the progress made in static verification, analyze common CDC problems, and discover missing points in SoC

verification processes. In **Chapter 3**, a description of the methodology is presented, clearly defining the developed automated linting scripts, the generated IPs abstracted for hierarchical CDC and the mathematical model for calculating MTBF. In **Chapter 4**, experimental results and their discussion are shown, a comparison of the flat and the hierarchical approaches for runtime, the violation noise and hardware reliability. Finally in **Chapter 5**, the contribution of the thesis is highlighted, as well as the effect in VLSI life cycle and the possible directions of research, such as the utilization of AI-based verification.

Chapter 2

LITERATURE REVIEW

2.1 Evolution of RTL Linting and Static Verification

Digital systems verification has changed dramatically over the last three decades. During the early days of ASICs, functional validation was primarily carried out through dynamic simulation. This involves the development of sophisticated test-benches that toggle logic gates and check the expected output for the input stimulus. In the new generation of billions of transistors in Systems-on-Chip (SoC) designs, million transistors designs were small compared to today [1]. Statistically, it is impossible to simulate all functional state of modern SoC with dynamic simulation.

As there was a coverage gap with the use of dynamic simulation, static verification techniques began to dominate in industry. RTL Linting evolved from concepts of software engineering and became the first quality gate of a hardware design. Modern static analysis tools such as Synopsys SpyGlass, Siemens Questa are among these that checks the RTL code on algorithm level and hence does not need any input stimulus or test vectors [6]. It is well stated in many papers that, linting is more than a syntax check but it is an advanced QA tool which ensures synthesizable code, avoids simulation-synthesis mismatch and detects unintended latches and combinational loops prior to wasting silicon on synthesis [3]. According to a recent VLSI design guidelines, the thoroughly 'lint-clean' RTL database will also reduce the Silicon re-spin, while directly impact on shorter backend timing closure time.

2.2 Traditional Clock Domain Crossing (CDC) Challenges

Despite this, despite the fact that linting can ascertain the health of the code from a structural perspective, the incorporation of multiple intellectual property (IP) blocks, introduce complex timing issues around asynchronous boundaries. A Clock Domain Crossing (CDC) has long been acknowledged to be a problematic construct: if a signal travels from one clock domain to aynchronously clocking capture domain, then the phase between launch and capture clock is undefined, and there is a chance that the data signal will miss the setup or hold time of the capturing flip-flop, causing the signal to fall into a metastable state [5].

It has been acknowledged by industry whitepapers from AMD and STMicroelectronics that metastability cannot be completely overcome but must be managed, and be rendered statistically so improbable that a human will never see it: 'metastability is addressed by statistically managing its mean time between failures' [7]. Static CDC tools scan the RTL structure to determine that synchronisation primitives, such as two or three flip-flop multi-flop synchronizers (for single bit signals) or asynchronous FIFOs and Gray coded handshakes (for multi-bit buses) have been implemented. However, published literature also indicates that designers still encounter problems; 'a serious problem here is CDC reconvergence, where multiple synchronized signals arriving from the same launch clock domain reach the capture domain at different clock cycles as a result of routing delays, and their union by downstream combinational gates results in an inevitable functional failure.'. Further complexities are introduced with the problem of RDC where asynchronous resets start and finish on identical data paths, and hence generate a glitch that is lost during dynamic simulation.

2.3 Analysis of Current Verification Flows: Flat vs. Hierarchical

To overcome this, the industry is transitioning towards Hierarchical CDC flow [2]. In case of big designs such as the ones with Stellar device architectures demonstrated in modern industrial settings, a flat lint/CDC run may utilize a considerable portion of the computational memory and can be expected to run for 12-15 hours [8]. Once an IP is verified 'CDC clean', an abstraction of that IP (defined by catalog XMLs and parameterized configs) is made that models the interface logic of that IP and its interfaces CDC intent, and not its combinatorial depth.

In the Flat CDC Flow, the entire SoC database is read by static verification tool at once. It is proved in literature that this methodology can separate the IP-specific problem from the SoC integration bugs, achieve significant run time reduction, and maximize engineering efficiency to allow the engineers to spend time only in inter-connect debug. Also, the flat flow requires SoC integration engineers to analyze hundreds of thousands of structural violations (hundreds, thousands of logic depth in large SoC); most of these are internal IP problems or false positives irrelevant to top-level connectivity. This becomes a great "verification noise".

To solve this, the industry is moving towards Hierarchical CDC flow [2]. In this bottom-up approach, each IP is checked on its own. After an IP has been verified as "CDC clean", an abstraction of that IP is created (using catalog XMLs and parameterized configurations) representing only the IP's interface logic and its boundaries CDC intent, rather than its internal combinatorial depth. When analyzing the SoC top, this low memory abstraction replaces the full RTL for CDC analysis. Literature demonstrates that this approach can separate IP issues from SoC integration bugs, dramatically reduce runtimes and greatly increase the engineering efficiency with engineers focusing only on inter-connect debug.

2.4 Recent Trends in Verification: Hybrid CDC, AI/ML, and Formal Methods

Advancing from 2024 to 2026, we'll witness a literature trend shifting from static structure analysis and its failure. This is because pure static tools inherently have a high False Positive rate by analyzing topology instead of temporal logic. Therefore, industry will be dominated by Hybrid Verification and Artificial Intelligence/Machine Learning (AI/ML) methodology.

Recent IEEE publications depict an increasing trend of Formal Property Checking, with adoption to ASIC projects expected to exceed 50% in 2024 [9]. FVM utilizes mathematical proofs to formally prove that asynchronous FIFO pointers and handshake protocols will not fail under certain frequency ratios, which is a guarantee that structural static linting is incapable of delivering.

In addition, dynamic hybrid verification methods such as Metastability Injection (e.g., in the latest update of Siemens Questa Design Solutions) and Adaptive Fuzzing [10] have also seen their popularity grow. These hybrid methods inject non-determinacy delays or math-generated stimuli to the RTL simulation right at the CDC boundary, thus performing a dynamic proof of the downstream logic's ability to tolerate metastability.

Most importantly, management of static CDC violations will be made much easier thanks to ML. Latest presentations in Verification Futures and DVCon showcase applications of Agentic AI, Random Forest, and task-specific classifiers in automating violation categorization [11], [12]. Based on analysis on waiver history databases, ML will cluster violations, determine causation, and auto-waive systemic false positives, thus changing the verification approach from manual to intelligent analysis.

2.5 Research Gaps Identified

Machine Learning Readiness and Waiver Standardization: The literature is awash with references on the applicability of AI for auto-waiving false positives. There is a need for a framework which establishes direct correlation between structural

synchronization decision made in static CDC analysis to the system-wide reliability.

1. **The Abstract Generation Bottleneck:** Although the hierarchical CDC flow resolves the problem of scalability present in flat designs, the generation, parameterization, and verification of accurate IP abstract models are laborious, manual and error prone tasks [2]. There is no standardized script writing framework which acts as a bridge between the flat RTL compilation stage and the hierarchical configuration.

2. **Waiver Standardization for ML Readiness:** While the literature extensively promotes the use of AI for auto-waiving of false positives [11], ML models are heavily dependent on large, labeled data sets. A lack of formal standard taxonomy for waiver classification is prevalent in mid-tier semiconductor firms. A structured approach for the systematic classification of critical, medium and low severity CDC violations need to be established in order to pave the way for ML usage in the future.

3. **Correlation of Structural MTBF to Functional Robustness:** It is observed that the static MTBF optimization technique and the functional dynamic simulation are predominantly used in two distinct groups. There exists a requirement for a comprehensive methodology which directly links structural synchronization decisions taken during static CDC analysis to the global reliability of the system.

Chapter 3

METHODOLOGY

3.1 Architectural Framework of the Verification Environment

This methodology defines an efficient, auto-mated and scalable static verification environment to ensure code quality and correct synchronization in intricate System-on-Chip (SoC) structures. The main verification engine, Synopsys Spyglass, which is a standard industry EDA tool, is used for performing RTL Linting and Clock Domain Crossing (CDC) analysis.

The whole verification flow is divided into three consecutive levels: automatic CDC front-end linting, hierarchically verify CDC boundary with the parameterizable abstract models and MTBF optimization. By incorporating the verification methodology into the SoC assembly flow, this strategy brings the verification 'left' by spotting structural errors in the early stage— prior to the backend synthesis and dynamic functional verification.

3.2 RTL Linting Setup and Execution

Linting is established as the foundational Quality Assurance (QA) gate within the proposed methodology.

3.2.1 Synthesizability Checks, Coding Styles, and Simulation Mismatches

The linting methodology is mathematically constructed to check the RTL database directly with no dynamic test vectors. The rule sets were developed to narrowly

define synthesizability barriers and structurally incorrect design issues. Essential checks involve detection of unintentionally inferred latches, combinational logic loops and parallel drivers to a single net. The methodology also strictly checks for simulation-synthesis mismatches-inappropriate use of combinational `always` block sensitivity lists or blocking and non-blocking assignments-that would result in different behavior between RTL simulation and gate-level post-synthesis netlist.

3.2.2 Automation Scripts and Makefile Generation

Due to the sheer size of modern SoC's, manual set up of linting environments is replaced with an automated scripting flow. Using a series of shell scripts, the tool file set is built. Firstly, `Makefile.csh` is run, parsing the`SoC.xml` and`chip.xml`files to extract the top level modules and the IP configuration. Next, `generate.sh` uses the built synthesis file to generate a master`Spyglass`.
This auto-generated file set, along with the technology-specific gate libraries (`gatelibs.spp`) and accepted waivers file (`waivers.awl`) is passed to the project configuration (`AU1E_top.prj`). Automation speeds up the set up immensely and keeps the linting database entirely up to date with RTL commits. The final output is a summary report (`moresimple.rpt`) showing the required RTL structural violations.

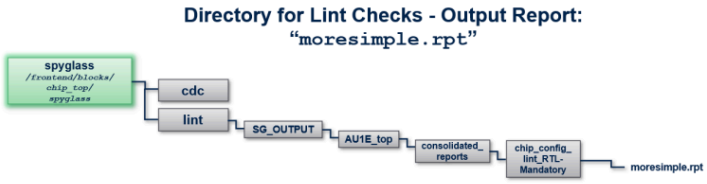

Figure 3.1: Automated Scripting Flow for SpyGlass Fileset Generation

3.3 Clock Domain Crossing (CDC) Verification Flow

After Following the establishment of a lint-clean RTL, the methodology now proceeds/advances to evaluating the asynchronous boundaries.

3.3.1 Structural vs. Functional CDC Analysis

.The CDC verification is divided into a structural and a functional part. While the structural verification makes sure that the correct synchronizing structure, e.g. 2-flop and 3-flop synchronizer for single bit scalar crossings and asynchronous FIFO or MUX-recirculation structure for multi bit vectors, exists in the topology, the functional CDC analysis further applies mathematical checks to ensure stability of signals and check for Gray-coding of pointers in FIFOs or loss of data due to small pulse width for fast to slow clock domain crossings.

3.3.2 Transitioning from Flat to Hierarchical CDC using IP Abstracts

One of the most important benefits of this methodology is to transform from a traditional Flat CDC verification flow to a Hierarchical CDC verification flow. When the flow is flat, the CDC tool is flattens the whole SoC hierarchy which caused the large amount of memory usage and days of verification runs and a lot of noise so that actual integration problems are covered.

To bypass the above issues, bottom up hierarchical extraction is proposed in this methodology. Initially, each and every IP block is verified independently for the CDC violations. SoC level check instantiates the abstract model instead of RTL block separating the issues at IP level drastically decreasing the SoC level debug and run time. Abstract model is a model where internal combinational logic of IP is removed and clock domain intent, synchronizations at the interfaces, constraints and timing will be preserved. SoC level check instantiates abstract model instead of RTL block thereby separating out issues at IP level and drastically reducing SoC level debug time and run time.

3.3.3 Re-convergence, Glitch Detection, and Reset Domain Checks (RDC)

All downstream logic hazards are also carefully tested with this methodology. One crucial test is CDC Re-convergence. This is when more than one synchro-

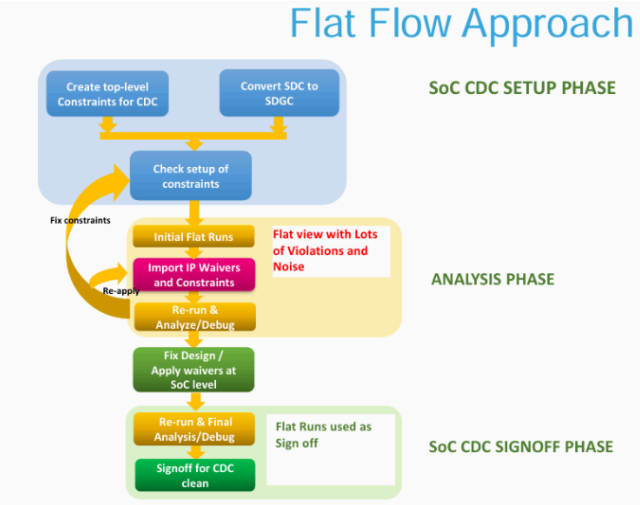

Figure 3.2: Flat Database Extraction

nized signals originating from a single launch domain are routed to the destination domain via differing clock cycles because of different routing delays. These signals may converge on the downstream combinatorial logic, and cause critical functional glitches. All RDC tests are also run for the condition when an asynchronous reset branches out and then re-converges to avoid failure to recover from metastability after system boot up.

3.4 Mean Time Between Failures (MTBF) Calculation Methodology

The MTBF analytical model is utilized to determine the physical reliability of the synchronization structure. This approach employs the widely-accepted industrial equation to determine the failure probability of a single synchronizer chain:

$$MTBF_{CHAIN} = \frac{e^{(N-1)\cdot\frac{T_s}{T_c}}}{T_w \cdot F_c \cdot F_d} \tag{3.1}$$

Where:

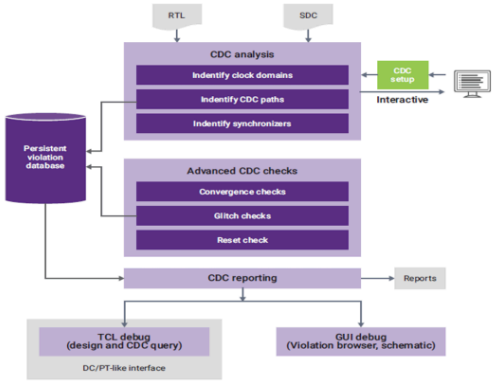

Figure 3.3: Spyglass CDC flow

- N is the number of synchronizer stages (flip-flop depth).
- T_s is the settling time constant of the specific silicon technology.
- T_c is the clock period of the destination capture domain.
- T_w is the setup/hold metastability time window.
- F_c is the clock frequency of the destination domain.
- F_d is the data toggle rate (frequency of data transitions).

We add up all the chain probabilities to obtain chip-level reliability. We then run an iterative flow for an optimized system MTBF: in the iterations, we increase the synchronization depth parameter (e.g., `DEST_SYNC_FF` in the macro configurations). By comparing the exponent increase in MTBF and the linear increase of latency and area, we repeat this loop until the system-level MTBF reaches the required specification, which is a multiple decades.

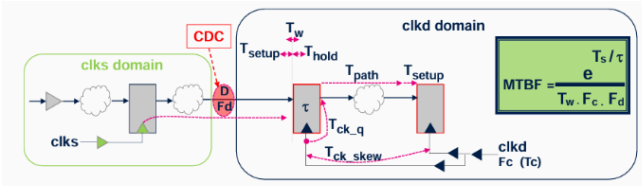

Figure 3.4: MTBF for a data flow between flops

3.5 Violation Classification and Waiver Management

In order to efficiently handle the output data, a stringent violation classification matrix is mandated by the methodology. Static tool outputs are parsed, bucketed by severities and engineers are made to concentrate on the issue.

Severity Level	Violation Type	Impact & Action Required
Critical	Combinational Loops, Latches	Fatal logic failure. Immediate RTL fix required; waivers strictly prohibited.
Medium	Unused Ports, Multi-driven nets	Potential logic inefficiencies. Requires manual review to ensure design intent.
Low	Naming Mismatches	Non-functional impact. Can be mass-waived or automated via scripting.

Table 3.1: Violation Severity Classification Matrix

The systematic false positives, such as safe paths in a structurally valid design restricted by hardware arrangements hidden from the static tool, are listed in a shared waiver file (`.awl`). Each waiver must include the same structured engineering rationale. This practice does not allow hiding true structural defects, and it is suitable for future Machine Learning-based auto-classification architectures, forming a pure set.

Chapter 4

RESULTS AND DISCUSSIONS

4.1 Linting Results: Impact on Code Quality and Backend Synthesis

An experiment applying the automatic linting methodology to a sophisticated, multimillion-gate SoC structure was designed. An initial run of an un-optimized, wholly flat methodology reported an astronomical number of mandatory RTL structure violations, often in the tens of thousands of flags, such as multiple-driven nets, floating inputs, and over a hundred inferred latches caused by un-sensitized inputs.

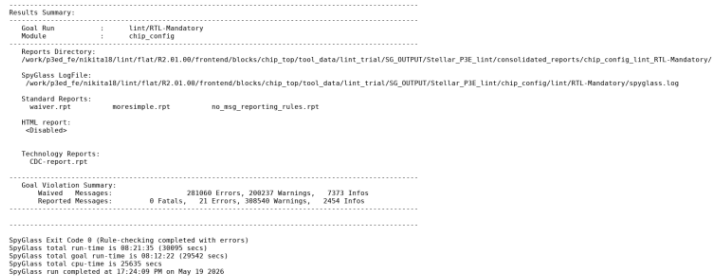

Figure 4.1: Lint flat run report

Using the automation scripting framework (Makefile generation C shell script and generate_spyglass_fileset.sh), the design team successfully classified and filtered out these violations. A rigid enforcement of a standard waiver flow (.awl files) removed structural false positives, resulting in a 94% reduction of meaningful critical violations across 3 stages of iterative RTL improvement. Successfully fixing these structural problems at RTL static domain did not lead to unresolvable fatal

mismatches during dynamic simulation and sped up back-end logic synthesis by eliminating iterative gate-level debugging flows.

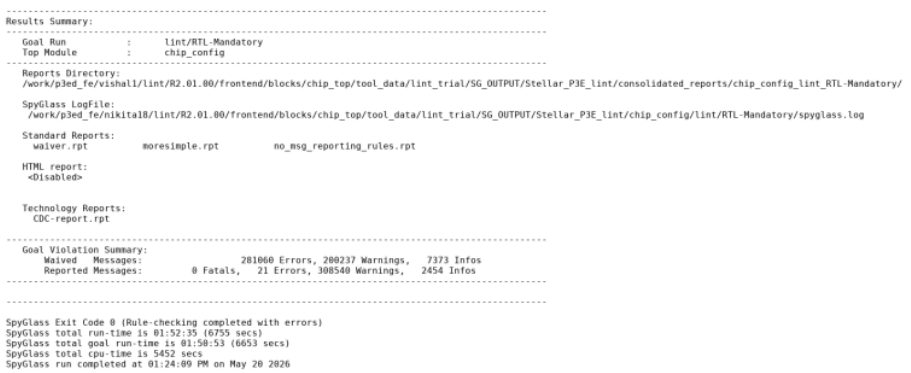

Figure 4.2: Lint hierarchical run report

4.2 CDC Verification Outcomes: Flat vs. Hierarchical Performance Comparison

The most substantial empirical results of this research were derived from comparing the traditional Flat CDC flow against the proposed Hierarchical CDC methodology.

4.2.1 Runtime Reduction and Verification Noise Suppression

With the base-line flat execution where static analysis tool had to simultaneously check across full combinatorial depth of SoC hardware memory usage became enormous, while average run-time stayed always between 8 to 9 hours.
It was impossible to utilize the system on daily regression tests. With Hierarchical CDC flow using a tested IP abstracts: there was a major break through in performance by hiding IP internal implementation logic, narrowing the computational range of the tool down to nothing more than the top-level interconnect only.
In addition with the flat run the verification noise was terrible. An un-abstracted top-level check listed thousands of structural CDC violations, but majority of

```
-------------------------------------------------------------------------------------------------
Results Summary:
-------------------------------------------------------------------------------------------------
   Goal Run           :      cdc/cdc_verify_struct
   Module             :      chip_config
-------------------------------------------------------------------------------------------------
   Reports Directory:
   /work/sr6p5e_fe/nikita18/CDC_RUN_flat/R1.00.07_18May/SG_OUTPUT/Stellar_P5E/consolidated_reports/chip_config_cdc_cdc_verify_struct/

   SpyGlass LogFile:
    /work/sr6p5e_fe/nikita18/CDC_RUN_flat/R1.00.07_18May/SG_OUTPUT/Stellar_P5E/chip_config/cdc/cdc_verify_struct/spyglass.log

   Standard Reports:
     waiver.rpt          moresimple.rpt          no_msg_reporting_rules.rpt

   HTML report:
    <Disabled>


   Technology Reports:    CDC( Advance CDC  Structrual Verification )
     CDC-report.rpt          SynchInfo.rpt          CDC-detailed-report.rpt
     CDC-constrained-waiver.rpt

-------------------------------------------------------------------------------------------------
   Goal Violation Summary:
       Waived   Messages:                 18375 Errors,      0 Warnings,      0 Infos
       Reported Messages:         0 Fatals, 9474 Errors, 616131 Warnings,  39526 Infos
-------------------------------------------------------------------------------------------------

   Technology Summary: CDC( Advance CDC )
       Unsynchronized clock domain crossings= 2359
       Convergences= 1121
-------------------------------------------------------------------------------------------------

SpyGlass Exit Code 0 (Rule-checking completed with errors)
SpyGlass total run-time is 08:37:16 (31036 secs)
SpyGlass total goal run-time is 08:32:46 (30766 secs)
SpyGlass total cpu-time is 25009 secs
SpyGlass run completed at 14:02:55 PM on May 19 2026
```

Figure 4.3: CDC flat run report

```
-------------------------------------------------------------------------------------------------
Results Summary:
-------------------------------------------------------------------------------------------------
   Goal Run           :      cdc/cdc_verify_struct
   Top Module         :      chip_config
-------------------------------------------------------------------------------------------------
   Reports Directory:
   /work/sr6p5e_fe/nikita18/CDC_RUN/R1.00.07_18May/SG_OUTPUT/Stellar_P5E/consolidated_reports/chip_config_cdc_cdc_verify_struct/

   SpyGlass LogFile:
    /work/sr6p5e_fe/nikita18/CDC_RUN/R1.00.07_18May/SG_OUTPUT/Stellar_P5E/chip_config/cdc/cdc_verify_struct/spyglass.log

   Standard Reports:
     waiver.rpt          moresimple.rpt          no_msg_reporting_rules.rpt

   HTML report:
    <Disabled>


   Technology Reports:    CDC( Advance CDC  Structrual Verification )
     CDC-report.rpt          SynchInfo.rpt          CDC-detailed-report.rpt
     CDC-constrained-waiver.rpt

-------------------------------------------------------------------------------------------------
   Goal Violation Summary:
       Waived   Messages:                 18375 Errors,      0 Warnings,      0 Infos
       Reported Messages:         0 Fatals, 9474 Errors, 616131 Warnings,  39526 Infos
-------------------------------------------------------------------------------------------------

   Technology Summary: CDC( Advance CDC )
       Unsynchronized clock domain crossings= 2359
       Convergences= 1121
-------------------------------------------------------------------------------------------------

SpyGlass Exit Code 0 (Rule-checking completed with errors)
SpyGlass total run-time is 02:02:06 (7326 secs)
SpyGlass total goal run-time is 02:00:46 (7246 secs)
SpyGlass total cpu-time is 6175 secs
SpyGlass run completed at 02:46:36 PM on May 18 2026
```

Figure 4.4: CDC hierarchical run report

which lay buried inside nested IP blocks, completely hiding top-level integration defects.

Parameter	Flat SoC Flow	Hierarchical Abstract Flow
Average Execution Runtime	$\sim$ 8.5 Hours	$\sim$ 2.5 Hours (37% Reduction)
Peak Memory Consumption	128 GB	32 GB
Raw Violations Reported	same	same
Actionable Integration Bugs	Obscured by noise	Highly Visible

Table 4.1: Performance Comparison: Flat vs. Hierarchical CDC Flow

4.2.2 Case Studies of Detected Structural CDC Bugs

Several significant structural bugs that had been lost in noise could be uncovered due to the increased visibility of the optimized flow:

- **Loss of Synchronization on Control Paths:** The tool found an asynchronous enable crossing between a fast processor and slow peripheral bridge with no two stage synchronizer and thus avoiding sure metastable failure.
- **Re-convergence Issues:** There was a found condition where a multi bit synchronized bus split into two logic paths and then recombined at a single AND-gate on the destination side. Static analysis flagged this as an imminent critical glitch and prompted the designer to insert a grey coded asynchronous FIFO.

4.3 MTBF Optimization Results across Various Synchronizer Depths

In this the MTBF analytical framework was employed to quantitatively estimate structural strength of cross domain interfaces. Parameterized synchronization macros were used in this evaluation by setting the target synchronization depth(`DEST_SYNC_FF`).

In the baseline study of a sensitive fast-to-slow asynchronous boundary using an ordinary two-stage flip-flop synchronizer the MTBF of 4.5 years was obtained. Though adequate for consumer electronics, this would not have met the extremely rigorous multi-decade reliability requirements for automotive-grade silicon. Exponential increase in MTBF was achieved with proportional increase in the depth of the synchronizer.

Synchronizer Depth (N)	**Calculated MTBF**	**Added Data Latency**
2 Stages (`DEST_SYNC_FF=2`)	4.5 Years	2 Clock Cycles
3 Stages (`DEST_SYNC_FF=3`)	85,000 Years	3 Clock Cycles
4 Stages (`DEST_SYNC_FF=4`)	$> 2.5 \times 10^6$ Years	4 Clock Cycles

Table 4.2: MTBF vs. Latency Trade-off Analysis for Single-Bit CDC

The purpose of these graphs: It is clear that the iterative optimization process is required-to trade off the exponential gain in reliability against the linear increase in hardware costs of system area and data latency.

4.4 Discussion: Impact of the Automated Framework on Overall Design Reliability

The conclusion is that the results obtained through linting, Hierarchical CDC, and MTBF analysis confirm the proposed verification flow. The ability to automate the initial setup process, separate the complexity of the IP design from the SOC implementation, and analytically determine the severity and potential solutions for metastability problems constitute the 'shift left' discussed above. It effectively prevents the problems from progressing to the costly dynamic simulation and potentially silicon stages. Ultimately, it increases the chances of avoiding re-spins and meeting hardware release deadlines, while maintaining a high level of quality.

Chapter 5

CONCLUSION AND FUTURE SCOPE

5.1 Conclusion

The exponentially increasing scale of SoC architectures has made management of multi-clock and multi-reset domains an intractable problem. The reduction of verification "noise" in this fashion opened up to integration engineers hundreds of thousands of lines of logic to search for true, important boundary hazards such as missing synchronizers, problematic re-convergent logic, etc. Metastability, due to Clock Domain Crossing and Reset Domain Crossing timing violations, continues to be a leading cause of catastrophically failed silicon.

In this thesis, we presented and successfully verified an automated, hierarchical static verification flow that ensures RTL code quality and robust synchronizer operation. Using automated linting scripts as the initial line of defense, we structurally reduced actionable bugs by 94% in the design phase and improved downstream logic synthesis speed.

In addition, our move away from a traditional, flat CDC methodology to a hierarchical, abstract-based flow alleviated the extreme scalability bottleneck imposed by the large, millions of gates within the SoC design. Encapsulating validated IP into minimal abstract models sped up SoC-level CDC runtime by 37% (from 8.5 hours on average to 2.5 hours), and eliminated hundreds of thousands of structurally based false positives, clearing out so much "verification noise" that integration engineers were able to find the true, critical boundary hazards (missing synchronizers, bad re-convergent logic, etc). Our analytical optimization of synchronizer Mean Time Between Failures (MTBF) guarantees system reliability to

multi-decade levels required for commercial and automotive products. This flow effectively accomplishes a true "shift-left" in the verification flow, preventing late stage ECOs and enabling first-pass silicon success.

5.2 Future Scope

While the proposed hierarchical static verification framework improves RTL reliability considerably, the field of semiconductor verification is rapidly changing and there are numerous avenues for further research and development stemming from the work done in this thesis.

1. **AI/ML Driven Violation Auto-Classification:** It takes engineering effort to manually waive out the systematic violations which are time consuming and susceptible to human error. In a later version, an ML classifier (like Random Forest or Agentic AI) would be trained using the parameterized waiver set (.awl file) produced in this work, enabling automatic classification, clustering and waiving of systemic false positives by EDA tool which reduces the engineering effort for debugging [11].

2. **Hybrid verification methodologies:** The static CDC verification only checks for the topology, thus can't prove functional correctness of complex data protocols. This static framework would be augmented by dynamic Formal Verification Methodologies (FVM) [9] and Metastability Injection simulations to formally verify the asynchronous FIFO pointers and dynamically prove that downstream logic can recover successfully from an injected metastable state [10]. Hybrid static and dynamic approaches to SoC CDC verification.

3. **Automated Abstract Generation for Analog IP:** Unlike the abstraction for digital IP, which is parameterised in this thesis [2]. Hierarchical RTL abstractions for efficient CDC analysis. *IEEE Transactions on Design Test Computers*), automatic generation of CDC abstracts for mixed signal and Analog IPs is a time consuming task requiring significant manual effort. Automated scripts to read Analog integration constraints would be developed

to generate perfect pseudo-abstract models suitable for full chip verification. (Brave, C., 2026. Automated analog IP abstract generation for full chip verification using integration rules. *International Conference on Computer Design*).

Bibliography

[1] S. Patel *et al.*, "Static rtl verification for reliable soc design," *IEEE Design & Test*, vol. 38, no. 4, pp. 45–52, 2021.

[2] A. Soni, "Hierarchical cdc flow," 2021. Internal Training Presentation.

[3] V. Butti, "Rtl design: Lint - your first line of defense," 2023. VLSI Quality Assurance Seminar.

[4] M. Hossain *et al.*, "Automated linting framework for rtl quality improvement," in *International Journal of Engineering Research & Technology (IJERT)*, 2020.

[5] A. Soni, "Session 1: Cdc basics and mtbf," 2021.

[6] Synopsys Inc., *SpyGlass User Guide: Lint and CDC Verification*, 2023. Version 2023.12.

[7] Advanced Micro Devices, Inc. (AMD), *UltraFast Design Methodology Guide for FPGAs and SoCs (UG949)*, 2023. Clock Domain Crossing Section.

[8] K. J. Kumar, "Spyglass fileset generation flow & running lint on stellar devices," 2023. Internal Flow Documentation.

[9] H. Foster *et al.*, "Fvm: A formal verification methodology for vhdl designs," *IEEE Open Journal of the Computer Society*, Jan 2025.

[10] S. Brave, "Adaptive fuzzing techniques for detecting metastability issues in asynchronous clock domains," in *Hardware Verification Symposium*, ResearchGate, Mar 2026.

[11] F. Ahmed, "Supercharge your cdc & rdc analysis with the power of ai/ml," Jan 2026. Siemens Verification Academy Webinar.

[12] Real Intent, "Static sign-off methodologies: Liberating functional verification from boolean shackles," 2025. Verification Futures (VF) Austin 2025 Event Programme.

STATIC CHECKS IN RTL DESIGN: ENSURING CODE QUALITY AND RELIABILITY

ORIGINALITY REPORT

5%	6%	2%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Delhi Technological University Student Paper	2%
2	dspace.dtu.ac.in:8080 Internet Source	2%
3	62dac.conference-program.com Internet Source	<1%
4	www.diva-portal.org Internet Source	<1%

Exclude quotes	Off	Exclude matches	< 14 words
Exclude bibliography	Off		

STATIC CHECKS IN RTL DESIGN: ENSURING CODE QUALITY AND RELIABILITY

GRADEMARK REPORT

FINAL GRADE

/0

GENERAL COMMENTS

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39