Tampere University

Vili Kautto

STATIC VERIFICATION TOOL IMPROVEMENT IN ASIC DESIGN FLOW

Tool evaluation using a real design

Master's Thesis
Faculty of Information Technology and Communication Sciences
Professor Timo Hämäläinen
University instructor Sakari Lahti
June 2022

ABSTRACT

Vili Kautto: Static verification tool improvement in ASIC design flow
Master's Thesis
Tampere University
Master's degree Programme in Electrical Engineering
June 2022

Verification is now the most time-consuming step in the design flow for digital circuits. Design organizations are constantly researching improvements to accelerate verification tasks so that a functional and efficient silicon can be released to a demanding market to improve the company's competitive position. Today, EDA (Electronic Design Automation) tools are a part of the development of each designed circuit and contribute to the verification work. Automating and simplifying the verification flow will help focus on resolving the underlying system issues. The company is interested in improving its verification flow to take advantage of the features available in the EDA market. Recognizing more synchronization structures, an improved hierarchical verification flow and incremental verification flow could potentially improve verification process throughput in the company. This study examines how changing the tool improves the static verification flow for the company.

This research examines the background of EDA tools and the most relevant theory to understand the tool roles and the static rule checks they make. In addition, the deployment of static verification tools will be discussed, and clock-domain crossing and lint checking tools will be introduced from an EDA toolkit. A modular inspection flow compatible with the server infrastructure will be built for this purpose. The company's proprietary and problematic synchronization structures will be implemented as an interface-level script, so that the inspection software understands the used structures to be suitable for specific use-cases. Design constraints are developed to improve the accuracy of the results. In addition, this study measures the performance of the programs. The use of computing resources is measured in the company's design environment and compared to the current verification flow. In addition, the quantity and quality of the reported messages are compared to the old flow, and the user experience and correctness of the results are briefly assessed. In the study, configured software checks are performed on the company's own subsystem under development, and the suitability of the software for the organization's purposes is determined by examining the results. Flow performance, duration, and utilization of computational resources are measured with the generated script and software reports. In addition, the functionality and user-friendliness of the software's graphical user interface are briefly reviewed.

The research finds that the clock-domain crossing verification flow is accelerated by over three times and the lint verification flow by a quarter. The inspections detect almost three times more potential issues in the code. The new tool flow requires under a third of the disk space and system memory consumed by the old verification flow. In addition, it is observed that the new software is overall more pleasant to use: The software is perceived to be somewhat more challenging to learn, but in return it provides more information to solve the underlying issues in the code. Finally, it is concluded that the company should consider adding the tool to complement its verification tool belt due to the performance, verification thoroughness and more moderate use of computation server resources.

Keywords: EDA, Lint, CDC, ASIC

TIIVISTELMÄ

Vili Kautto: Static verification tool improvement in ASIC design flow
Diplomityö
Tampereen yliopisto
Sähkötekniikan diplomi-insinöörin tutkinto-ohjelma
Kesäkuu 2022

Suunnittelun varmennus on nykyään aikaa vievin vaihe digitaalipiirien suunnitteluvuossa. Suunnitteluorganisaatiot pyrkivät jatkuvasti kehittämään nopeampaa varmennusta, jotta toimiva ja tehokas piiri saataisiin julkaistua vaativalle markkinasegmentille oman kilpailuaseman parantamiseksi. Nykyään EDA-työkalut (engl. Electronic Design Automation) ovat osana jokaisen suunnitellun piirin kehitystä ja osaltaan ne tukevat myös varmennustyötä. Varmennusvuon automatisointi ja yhdenmukaistaminen auttavat keskittymään järjestelmän ongelmien ratkomiseen. Eräs yritys on kiinnostunut varmennusvuonsa parantamisesta hyödyntääkseen tarjolla olevia ominaisuuksia tunnistaakseen synkronointirakenteet paremmin ja parantaakseen hierarkkista sekä inkrementaalista tarkastusvuota. Nämä saattaisivat osaltaan nopeuttaa ja helpottaa varmennustyön vaiheita järjestelmäpiirien suunnittelussa. Tutkimuksessa selvitetään millä tavoin työkalun vaihtaminen parantaisi yrityksen tarkistusvuota.

Tässä diplomityössä tutkitaan EDA-työkalujen taustaa ja asennettavien työkalujen kannalta olennaista taustateoriaa, jotta voidaan ymmärtää työkalujen asemaa digitaalisuunnittelussa sekä niiden tekemiä tarkistuksia. Tämän lisäksi työssä käsitellään staattisten tarkastustyökalujen käyttöönottoa ja hyödynnetään vaihtoehtoisia EDA-työkalukokonaisuuden kellorajapintojen ja koodirakenteen tarkastustyökaluja. Näiden työkalujen ympärille rakennetaan modulaarinen palvelininfrastruktuurin kanssa yhteensopiva tarkastusvuo. Tämän jälkeen määritetään ongelmallisena pidetyt yrityksen omat synkronointirakenteet rajapintatarkasteluna, jotta tarkastusohjelmisto ymmärtää käytetyt rakenteet käyttötilanteisiin sopiviksi. Tarkistusrajoitteita kehitetään tuloksien oikeellisuuden parantamiseksi. Tässä tutkimuksessa mitataan ohjelman suorituskykyä ja laskentaresurssien käyttöä yrityksen suunnitteluympäristössä nykyiseen varmennusvuohon verrattuna. Havaittujen ongelmien määrää ja laatua verrataan vanhaan tarkastusvuohon. Lisäksi työssä sivutaan ohjelmiston käyttökokemusta ja tuloksien oikeellisuutta. Tutkimuksessa suoritetaan konfiguroidut ohjelmistotarkistukset yrityksen omalle kehityksessä olevalle alilohkolle ja tuloksia tarkastelemalla määritetään ohjelmiston soveltuvuus organisaation käyttötarkoituksiin. Tarkastusten suorituskyky, kesto ja laskentaresurssien käyttö mitataan luodulla koodilla ja ohjelmistoraporteilla. Lisäksi ohjelmiston graafisen käyttöliittymän toiminnallisuutta ja käyttäjäystävällisyyttä tarkastellaan lyhyesti.

Tutkimuksessa havaitaan, että kellorajapintojen tarkasteluvuo nopeutuu yli kolminkertaiseksi ja koodin tarkastusvuo neljänneksellä. Vastaavasti tarkastuksissa havaitaan melkein kominkertainen määrä mahdollisia ongelmia koodissa. Uusi työkaluvuo vaatii alle kolmasosan vanha tarkastusvuon levytilasta ja järjestelmämuistista. Lisäksi havaitaan, että uusi ohjelmisto on kokonaisuutena miellyttävä käyttää, sillä vaikka se koetaan nykyistä hieman haastavammaksi opetella, tarjoaa se vastaavasti enemmän tietoa ongelmien ratkaisemiseksi. Lopuksi havaitaan, että yrityksen kannattaa harkita edellä mainitun ohjelmiston lisäämistä muiden käyttämiensä tarkastustyökalujen joukkoon tarkastusvuon suorituskykyisyyden, tarkistusten perusteellisuuden ja maltillisemman palvelinresurssien käytön ansiosta.

Avainsanat: EDA, Lint, CDC, ASIC

PREFACE

This MSc thesis work conforms to the Guide to Writing a Thesis in Technical Fields at Tampere University during the spring of 2022. The thesis study was commissioned by a company.

First of all, I would like to thank the company and especially the reviewed expert in the field for making this thesis work possible. Unfortunately, he is no longer working with us, but he had time to give me the necessary contacts, guidelines, a four-page long study scope definition and eventually free hands in project execution. Separately, I want to thank the people for rapidly filling his shoes regarding this thesis once he was gone.

Moreover, I would sincerely like to thank the provider of the alternative tools for the evaluation. More specifically, I also want to thank their EDA staff for their time, effort, humour, expertise, tenacity, and shared lunches that this evaluation has required. Even though I had no previous experience in product deployment, you always respected my and the company's needs for this work and delivered everything that was requested. I had a pleasant time working with you all, and I can without hesitation commend your support as reliable, flexible, fast and trustworthy.

In addition, I want to acknowledge the rest of the project top-level team for working hard and letting me focus on the thesis work so I could graduate to become a fully-fledged company employee sooner: Knowing you covered my back the whole time kept me motivated.

A detailed analysis would have not been possible without knowing a plethora of technical details about the design under testing. For that reason, I would like to thank everyone in the company who delivered the vital information despite me interrupting your work.

Finally, I would like to thank my friends and family for cheering me up and especially Samuli Pohjola and Heikki Lahtinen for proof-reading the thesis to improve it.

Tampere, 10th of June 2022,

Vili Kautto

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

ASIC	Application-Specific Integrated Circuit
CAD	Computer-Aided Design
CDC	Clock-Domain Crossing
CentOS	Community Enterprise Operating System
CI	Continuous Integration
CMOS	Complementary Metal-Oxide-Semiconductor
CPU	Central Processing Unit
DB	Database
DFF	D-type Flip-Flop
DUT	Device Under Test
EDA	Electronic Design Automation
FPGA	Field-Programmable Gate Array
GNU	GNU’s Not Unix
GUI	Graphical User Interface
HDL	Hardware Description Language
HTML	Hypertext Markup Language
IC	Integrated Circuit
IDE	Integrated Development Environment
I/O	Input and Output interface
IP	Semiconductor Intellectual Property core
MTBF	Mean Time Between Failures
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
OS	Operating System
PDF	Portable Document Format
RAM	Random access memory
RDC	Reset-Domain Crossing
RTL	Register-transfer Level
RTT	Round-Trip Time
RX	Receiving side of the data transmission
SSH	Secure Shell
SoC	System on a Chip
TCL	Tool Command Language
UPF	Unified Power Format
UVM	Universal Verification Methodology
UX	User experience
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLSI	Very Large-Scale Integration
VPN	Virtual Private Network

1. INTRODUCTION

Semiconductor structures can now be built to be smaller than ever before with constantly evolving manufacturing technology, while market demands increased feature sets in IC (Integrated Circuit) products [1]. Humans currently use more ICs than ever before, and at the time of writing, there is said to be an ongoing silicon circuit shortage, which limits the productions of cars, for example [2] [3]. Simultaneously IC power consumption, computational and silicon area demands have historically been mutually exclusive with each other and servicing all these conflicting areas of interest partly contributes to increasing design complexity. [4, p. 2]. As verification gap widens every year, an increasing number of chips will not meet the original development schedule goals, partly because of all-encompassing feature improvements [5].

As design complexity increases rapidly, the complexity of verification increases exponentially in relation, while the importance of releasing a product in time increases as well [4, p. 4] [6, p. 1] [7, p. 1] [8]. Design verification is trying to keep up with the pace of increasing product feature sets and is consuming most of the design time and effort nowadays [9, p. 2–3]. To ensure that a logic design conforms to the set of specifications, deploying EDA (Electronic Design Automation) tools is done to automate and standardize the verification process. Verification EDA tools are commonly used in the industry to systematically locate, report, and fix typical functional errors in the design before the product is manufactured. The sooner an error is found and fixed, the cheaper the repercussions will be in general [4, p. 194].

This financial incentive keeps costly EDA tools relevant in the very competitive ASIC (Application-Specific Integrated Circuit) design industry [10]. Nowadays, the highly competed market space of ASIC business aims to produce systems in time to satisfy the market needs, while avoiding errors to minimize the chance of needing to re-do a silicon product, which would waste a lot of time and money [11].

There are different vendors offering EDA solutions, which usually are targeted towards their own respective design flows [12, p. 8]. The goal of this thesis is to evaluate improvements made by switching EDA tools, such as CDC (Clock-Domain Crossing) and linters. This information in turn is then used to assess if an alternative design flow with a competing software can be used to perform tasks more efficiently. To achieve that, the requirements for the verification software must be established. In this research, an alternative static verification solution is evaluated for the company. The verification performance data being compared to the currently used design flow is included in the results and analysed for this purpose.

This document is structured as follows. Chapter 2 introduces a theoretical background behind using EDA tools and includes information about the static check tools evaluated in this research. Chapter 3 addresses clock-domain crossing theory. Chapter 4 discusses the research strategy for the software analysis and explains the test methodology, environment, and the created workflow. Chapter 5 focuses on displaying the measurement data and chapter 6 summarizes the research.

2. EDA AND STATIC CHECKS

This chapter introduces the incentives behind using EDA tools for verification, starting from the brief history of EDA tools in digital design. Some aspects in verification tools today are defined to justify the feature set to improve the design flow. Finally, influential digital design theory behind lint and CDC verification is investigated for the final study, in which a new static verification suite is deployed for evaluation. In this evaluation, Unfortunately, the terms of licensing do not allow disclosing the tool name. For that reason, the comparative study focuses on the benefits and drawbacks of the new workflow versus the old one used in the company.

2.1 The importance of EDA tools

EDA is one of the key enablers of the semiconductor industry: It is used in designing every IC (Integrated Circuit) nowadays and the semiconductors on the other hand drive improvements in EDA technology [13, p. 1–2]. The improvements enabled by advanced technology nodes are exploited in the design flow to reduce costs by using more advanced design tools in the process. Therefore, a new design will in turn be further refined, fuelling an observation called Moore's law. This phenomenon solidifies the design tools' position in ASIC design flow, as placing ever-increasing transistor nets manually was already becoming humanly impossible in the 1980's. [1]

In Figure 1, ASIC (Application-Specific Integrated Circuit) design costs are categorized as a function of a shrinking process node. Verification cost increases significantly as the technology node shrinks. This is because the number of design states increases exponentially as more features, which all need to be verified, fit into the same area. It should be noted that software development has an even steeper curve in Figure 1.

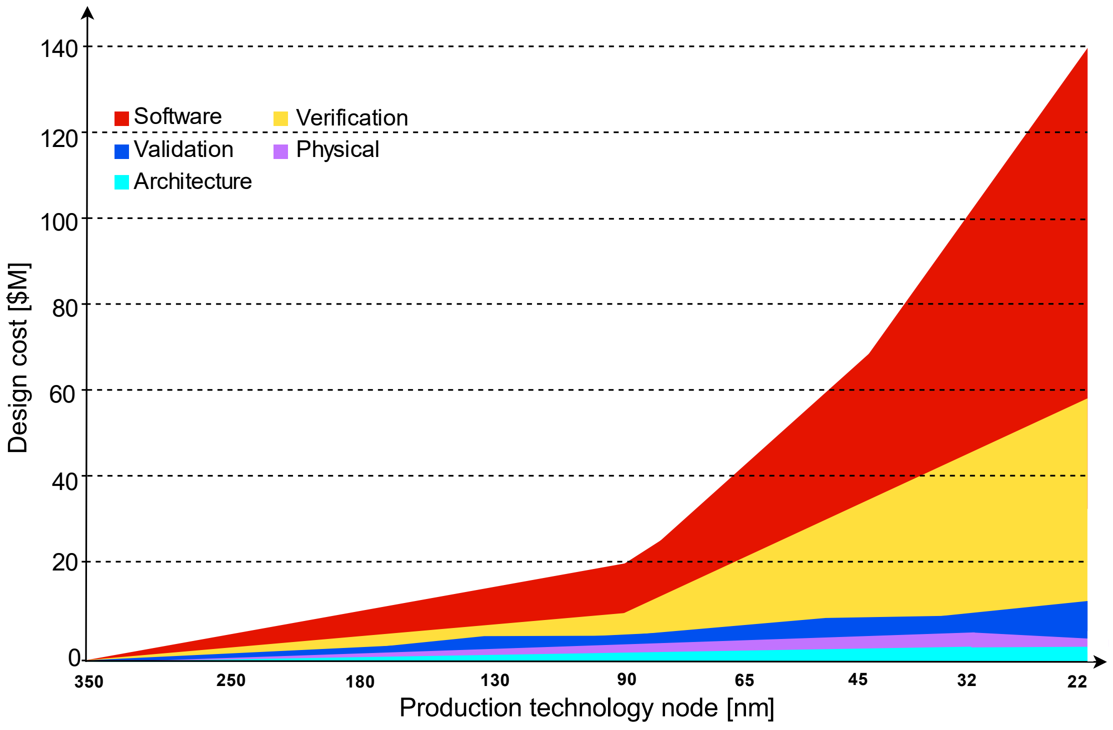


Figure 1: *Categorization of increasing design cost in ASIC development as a function of shrinking technology node. [4, p. 3].*

The long-lasting revolution of the semiconductor industry has required help from the EDA tools, which have proven to be an important asset in VLSI (Very Large-Scale Integration) design [6, p. 287] [14]. As the EDA tools always enable the next generation of computing, they must logically be ahead of the current paradigms in hardware design.

The EDA development started as automatic place-and-route layout tools were introduce to the market in the late 1970's. From the late 1980's onwards, logic synthesis tools became available to help improve design efficiency. It is estimated that some designers would benefit from using high-level synthesis tools to improve throughput. [5]

The trend in the semiconductor industry is to increase component reusability as much as possible. This increases productivity by reducing design verification gap: Focusing the efforts into the system level integration in SoC (System on a Chip) structure while using as many verified blocks and IP (Semiconductor Intellectual Property core) as possible minimizes redundant verification [15] [16]. One main benefit of SoC is design reuse enabled by modularity. Dire time-to-market demands, on the other hand, restrict design reuse, because designing a reusable platform takes more time in general and is typically not as high a priority as optimization or costs [5]. An example of a SoC structure containing hierarchical elements is presented in the following Figure 2, which displays the main hierarchical elements in the design, such as the main interconnects and multiple sub-IPs.

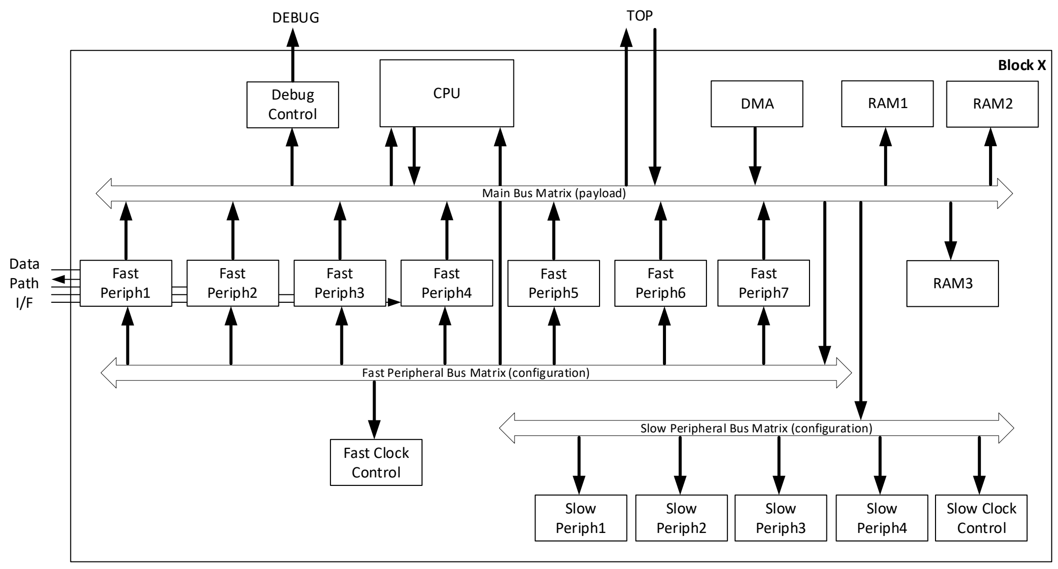


Figure 2: *An example of a hierarchical system architecture. This functional block is used as a test platform in this study.*

Reusability of IP increases in the designs produced in the industry, but as hierarchical SoC structures have become more common, the demand to verify the integration needs more attention than before [16]. As verification consumes the majority of pre-production efforts, reuse is a powerful method to direct efforts to the necessary system level checking [17, p. 83]. EDA tools fulfil an important role in the process of turning the HDL (Hardware Description Language) based IP into a physical IC on silicon [5]. The best-known EDA tool vendors are Synopsys, Siemens EDA (previously Mentor Graphics) and Cadence [12, p. 8]. These EDA tools will address the ever-increasing design complexity by enabling more efficient designing of electronic systems [18].

In Figure 3, An example of a cumulative number of bugs found in a design is provided as a function of time. Multiplying the values of both curves at every point will provide bug costs affecting the financial ASIC project overhead. These all summed together would yield the total cost.

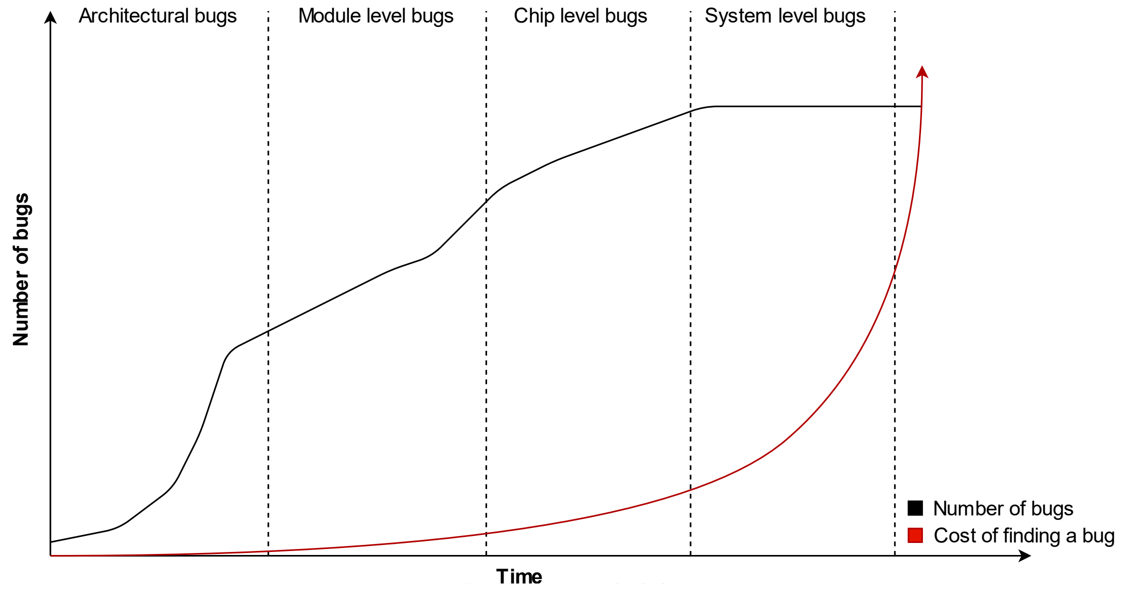


Figure 3: *Cost of finding a bug in an ASIC device as a function of time categorized by the hierarchical level. [17, p. 81]*

It is important to achieve a successful silicon wafer tape-out at the first try as re-spins are costly and time consuming [11]. To prevent re-spins, it is advised to accomplish RTL (Register-transfer Level) signoff at IP, subsystem and SoC hierarchy levels as the design matures. This means that the verification flow was not able to find any suspicious structures or clear mistakes in the design. Also, unclear cases have been reviewed and deliberately accepted on a case-by-case basis for each DUT (Device Under Test). Preferably this so-called “sign-off” process is done on IP, subsystem and SoC top levels. This allows to start the verification flow early in design flow to reduce the number of bugs and to improve the quality of the design [19] [20]. The tools discussed in this research are rule-checkers, which are meant to provide exhaustive lists of the possible violations to be forwarded to other types of verification [21, p. 88].

It is also possible to enable CI-tools (Continuous Integration), such as Jenkins to periodically run the EDA tool flow in the development environment to set a project wide verification standard, which most people can access and gather feedback from. This makes the results more accessible, peer-reviewable, and transparent, improving project management, clarifying status, and easing the integration tasks [22]. However, running CI is not always ideal either, as it increases the requirements for software licences and computing requirements remarkably. CI verification is more complex to setup for specialized runs and for that reason, CI is preferred only when doing the standard runs periodically with no human interaction in between the quality checks.

EDA tools typically work in conjunction with rule sets or methodologies, which typically can be configured to serve the needs, phases, and hierarchical levels of different VLSI designs. These sets will define verification errors and warning notifications ranging from readability improvement suggestions to unnecessary type casts and floating data pins. Naturally, the rule sets are not always purpose-built for each design by the software vendor, and therefore some configuration is typically recommended to reduce overlapping in violation messages. Redundant information is typically unwanted, as it misrepresents the severity of the real issue by giving it an inherent weight by error count. This causes some other issues in turn to have a smaller priority in many design flows without a justifiable reason since designer time is limited.

EDA tools aim to improve issue readability and process automation, reduce redundant information noise, ease creating and sharing reports, provide methodology, improve reliability and consistency, and enable analyses early in the flow. In this work, the focus is on CDC and lint functionality to improve the workflow for quality checking for ASIC development.

2.2 Lint

Humans make careless mistakes over time because of miscommunication, interruption and forgetfulness caused by a plethora of different reasons ranging from simple thought errors to misunderstandings. When creating HDL code for ASIC, these daily mishaps might be the cause of functional errors in the code and that in turn can make the product deviate from the specification or set standards. If left unnoticed, this can be a considerable hindrance when trying to create a holistic SoC product with high complexity and many hierarchical levels, as making sure that everything is working correctly in ASIC tape-out is critical. Generally holistic means “more than the sum of its parts”. In SoC design, system level considerations define the characteristics of its subsystems which the SoC is built on. [23, p. 21–24] [17, p. 13–14] [24, p. 20].

For these reasons, vendors have long been offering lint tools with mature rulesets to provide syntax and coding style checks. Lint is used with purpose-built rules to pinpoint, for example, suspicious declarations, unreachable code, and infinite loops. The main benefit is running the checks to catch bugs early in the design flow even before compilation and synthesis phases. [25, p. 54]

Even though some IDEs (Integrated Development Environment) can nowadays support some basic lint functionality to provide the earlier step of the design flow with some feedback, lint tools have developed to support synthesis and design compilers to offer even more in-depth information. Typical lint issues include, but are not limited to, unsynthesizable constructs, unintentional latches, unused declarations, driven and undriven signals, race conditions, incorrect usage of blocking and non-blocking assignments, incomplete assignments in subroutines, case statement style issues and out-of-range indexing [12, p. 36].

Lint tools, also known as linters, also have their limitations when it comes to error analysis. Even though it can identify structures well, questions rise when a linter observes a suspicious but allowed structure with more depth to it than the structure alone: Stability of a combinational loop, severity of contentious buses and unconnected nets remain out of the verification scope for lint alone. Additional tests and actions are required to justify keeping these structures in a design. Alternatively, a workaround can be developed to save verification time. [26, p. 86]

3. CLOCK-DOMAIN CROSSINGS

Ever-increasing design complexity typically accounts for an increased clock signal count and control logic, both of which in turn increase the number of clock-domains and flip-flops in a modern ASIC design. Signal synchronization when entering a different clock domain is done to prevent glitches and metastability from propagating through the design, which could cause a critical failure in functionality [21, p. 74–78]. Metastability refers to the unstable behaviour of a flip-flop caused by setup and hold-up time specification violation in the input, causing the bistable storage element to output an unstable voltage value between logical 0 and 1 [11] [27]. Metastability is a symptom of a CDC issue, which can lead to a rapid and unexpected device malfunction if handled poorly: Unreliable or lacking data transfer can hinder the reliability of the design unit in question [11] [21, p. 73–74]. The importance of CDC issues is underlined by the fact that roughly a quarter of silicon re-spins are results of a clocking issue [4, p. 249]. The typical cause for metastability is inspected in the following Figure 4, where mutually asynchronous clock signals clk1 and clk2 are displayed. In the picture, the dynamic time until the next rising edge of target clock clk2 can be observed. This could lead to setup time violations across flip-flops due to the clock domain crossing, causing unreliable data transfer and possibly metastability. In Figure 4, a setup time of 3.5 units is considered to be out of specification for the flip-flop. [11]

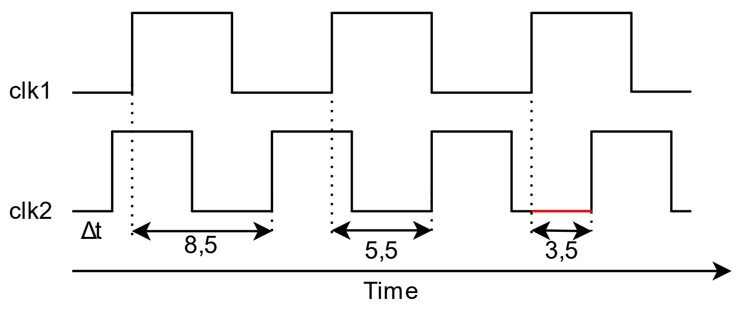


Figure 4: *Asynchronous clock signal clk1 and clk2. [21, p. 73–74]*

Shurivala and Gard define a separate clock domain as follows: “For any given pair of clocks, if the phase relationship between their active edges cannot be determined *predictably*, the two clocks are considered to be in separate domains. And, if the phase relationship between the edges can be determined *predictably* – then the two clocks are considered to be in the same domain.” [21, p. 88]

Metastability will temporarily increase the transient power consumption of the fanout cells utilizing CMOS (Complementary Metal-Oxide-Semiconductor) transistors, because a temporary short-circuit is developed for all the transistors driven by the metastable flop's output. Having a two-flip-flop synchronizer already mitigates this issue by limiting the number of target cells to the first input transistor in the synchronizer. The metastable value can also be interpreted differently in any of the driven inputs due to miniscule differences in electrical parameters in the signal transmission lines and fanout. [21, p. 75–76]

In addition to the previous issues, metastability has a tendency to propagate within the design unit if not synchronized accordingly. Therefore, the previously mentioned effects also propagate on the path on the RX (Receiving side of the data transmission) chain of structures, usually multiplying the unwanted effects. The delay of the synchronizer allows the temporarily unstable value to settle. Depending on the clock rate of the signal, the number of synchronizers and the tenacity of metastability, this is in most cases resolved with an intentional delay in the signal transmission. [27] [21, p. 74–77]

For example, in a design where data propagates though 2 sequential D-type flip-flops, both of which are operated by a clock signal with deviating oscillation frequencies. Over time, the latter asynchronous flip-flop will experience a data-path metastability when a timing violation inevitably occurs due to constantly changing phase shift with respect to the two clock signals. Metastability can be observed in Figure 5 receive-side data. [4, p. 150]

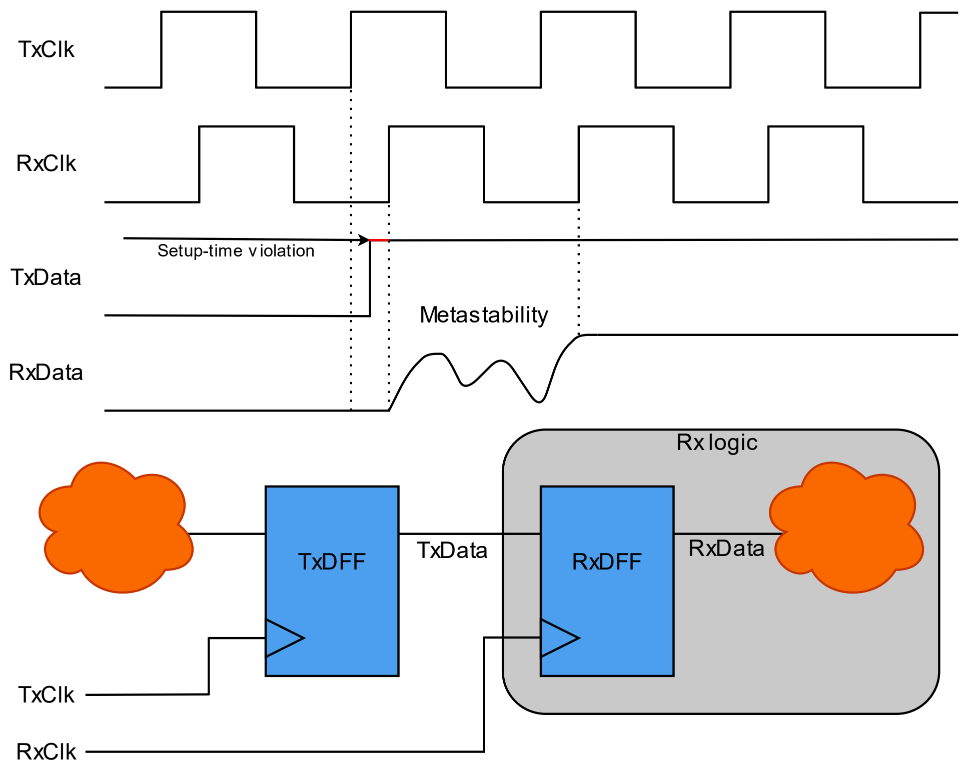


***Figure 5**: A demonstration of metastability in an unsynchronized design. [4, p. 150]*

Figure 5 describes the RxData region as metastable. The logical value for the signal level during the first full clock cycle of RxClk is undefined, as the voltage level does not represent a strict binary 1 or 0, but rather an intermediate transient value between them. In Figure 5, an RTL illustration is located below and corresponding timing graph above. Each connection in RTL is represented with a named domain. The metastability related oscillations in RxData should be noted even though TxData is firmly bound to a logical value. There is no clock frequency slow enough or delay in synchronization long enough to absolutely eliminate this issue, but there are methods to alleviate the frequency of metastability occurrence down to acceptable levels [4, p. 150] [28].

The possibility of metastability negatively impacts the MTBF (Mean Time Between Failures) of a finished product in which the synchronization scheme is not ideal [29]. Even though the probability of a metastable occurrence is rather low with synchronization, the ever-increasing clock frequencies due to technology improvements with numerous clock-domain crossings in cumulate the probability of an issue occurring. In addition, ASIC circuits are designed to be produced in high volume, which means that there will be multiple circuits existing simultaneously, which further increases the odds of cumulative singular failure in any circuit. Even though a double DFF (D-type Flip-Flop) synchronization offers moderate protection against metastability, in theory it merely increases the average number of clock cycles required to trigger metastability instead of completely preventing it. There is no limit to how long a single metastable incident can last in a circuit, and it can happen to any binary state cell holding information, such as latches. To improve MTBF, additional DFFs can be inserted in each synchronizer unit to target high-speed and high-reliability designs to add latency in order to solve the metastability. The chance of metastability occurring cannot be completely removed, but it can be mitigated to an extent which is reliable enough. [27] [21, p. 74–78]

To complement the previous Figure 5, using 2-FF synchronization would affect the RTL and timing graph in the following manner [21, p. 77] in Figure 6. The timing graph above corresponds to the RTL below. By implementing an additional synchronization step, the probability of metastability is now considerably lower and thus the design is much more robust against register timing violations.

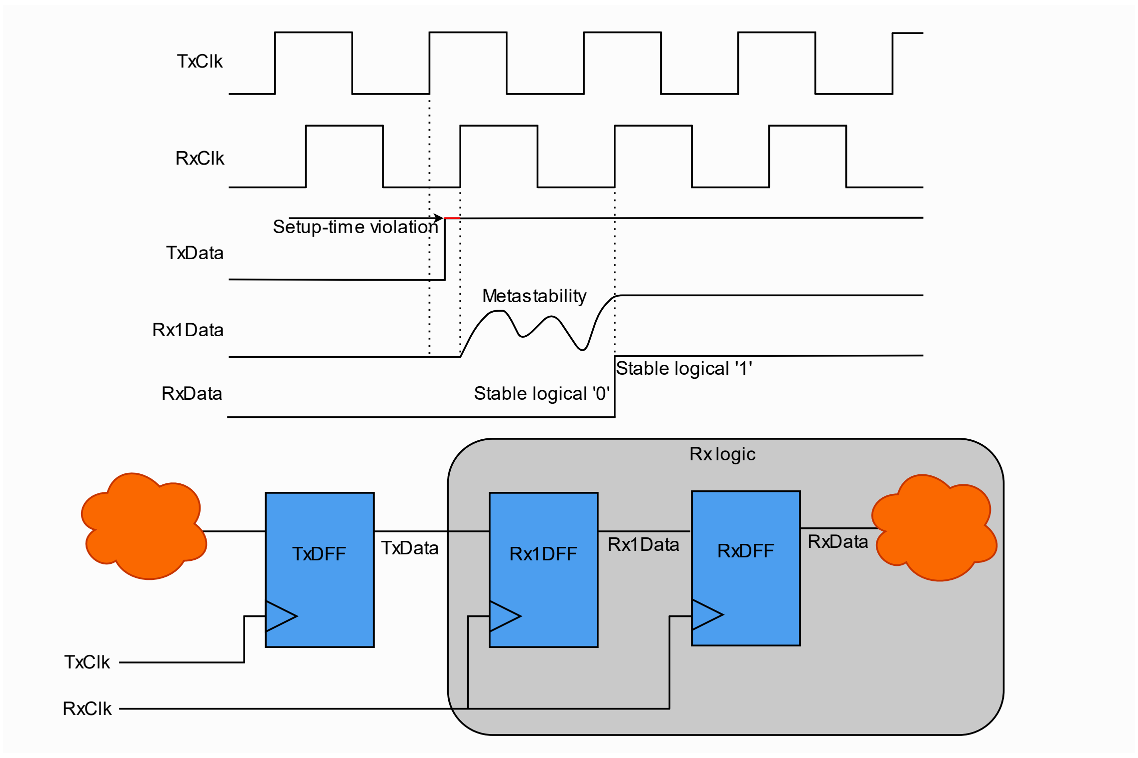


Figure 6: *A demonstration of alleviating metastability with a synchronizer. [4, p. 150]*

When compared to the previous Figure 6, the previous metastability issue has now been resolved at the destination RxData, as the logical value of RxData1 has a full clock cycle length intermediate step to settle into a desired logical value before it is in turn interpreted as the final RxData. Multiple fanout within the synchronizer is not allowed and therefore synchronizers limit the metastable fanout transistor count. If they were MOSFET (Metal-Oxide-Semiconductor Field-Effect Transistor) type transistors, then synchronizers would also limit the amount of wasted power during the metastable period caused by an open transistor channel. In addition, this structure also solves the data ambiguity issue between multiple fanout targets, as there exists only one target cell to interpret the temporarily metastable logical value of Rx1Data. [21, p. 74–77]

One solution is to gate the clock signals when they are not needed. This is mainly done to save power, as the line capacitance of a clock tree will typically consume a considerable amount of energy, but this also has the additional benefit of preventing metastability within the gated parts of the system: The likelihood of metastability is directly related to the number of propagated clock cycles through the registers. [30]

Even though the metastability issue can be alleviated by using synchronization, conforming to this solution for every clock domain crossing in an SoC will consume some silicon area in the final product. The gate count increases along with the added synchronizers, which will increase the design complexity by itself. In addition, implementing a synchronizer with multiple flip-flops will naturally increase latency in the communication link. This can be a meaningful drawback in some latency and area sensitive applications. [1, p. 111]

Glitches are also a typical source of issues in a design [31]. The synchronization scheme will only be effective whenever the data is stable: Data should be properly set up before an ‘enable’ signal is asserted. This potential issue can be solved by using a “full handshake” process with mutual acknowledgements for data transmission. This involves some additional latency but will in turn be more reliable in the application in which it is needed the most. [32]

Figure 7 introduces some general design rules to reduce the chance of a glitch occurring in the DUT. It is a recommended design practice to not insert combinatorial logic before registers in the control path or in in the feedback of the data path: The red connections depict the restricted areas for combinatorial logic in Figure 7.

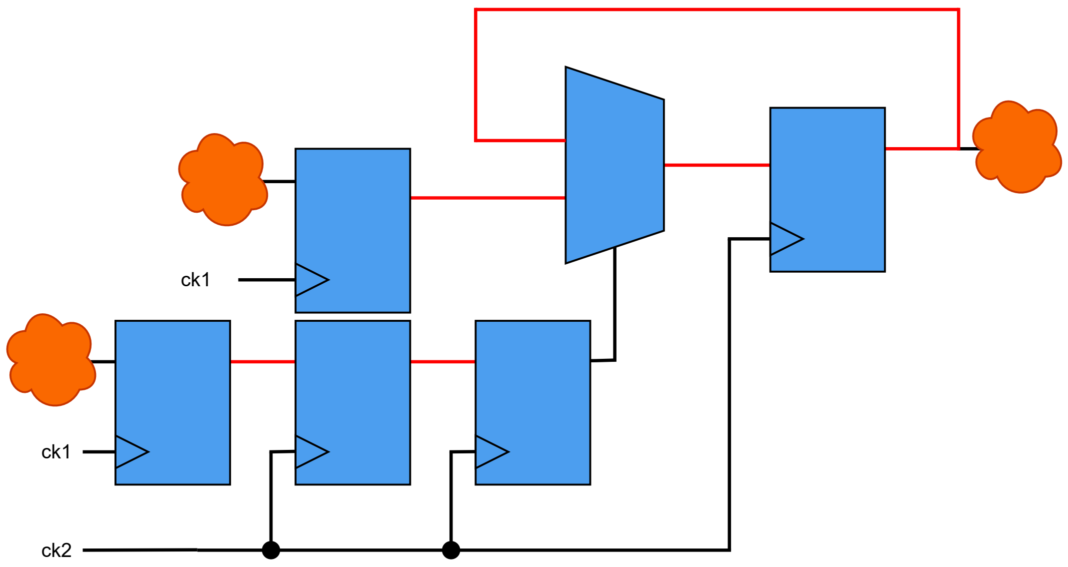


Figure 7: *Design rules to avoid glitches.* [32]

Glitches are effectively a result of mutually asynchronous clock frequencies combining in a logical port causing an output, which, for the transient time period of its existence, resembles a clock rate which may not correspond to any of the specified input frequencies. This short-lived pulse cannot be trusted to comply with any timing standards. A recurring timing violation will eventually cause metastability, which in-turn justifies handling glitch logic as errors in CDC tool. Glitches, however, sometimes propagate through the design, which can cause functional errors. [33]

3.1 Preferable features

ASIC design development is moving at a rapid pace, and the design tools should have setup properties to enable both conventional and new development strategies, such as top-down versus bottom-up development flow and flexible configuration possibilities for modern design solutions [10]. Running static-check software, even for a subsystem, takes some time in itself and making bug handling and reporting as effortless and quick as possible will help alleviate the increasing verification productivity gap [4, p. 3]. Some features are given a weight integer to emphasize their value in Appendix A.

The company was particularly interested in three main features of the new EDA tools. Testing incremental flow to reduce designer hours was proposed. Also, a hierarchical flow has been requested to be inspected for the same reason. In addition, customizable synchronization schemes should be configured to help the software understand the DUT [33, p. 24]. However, any other features that speed up or simplify the flow setup, runtime, user experience, debug or reporting will also be appreciated in their own right.

3.1.1 Flexible constraints

Verification tool flow sometimes benefits from using constraint files to limit the scope of the test. Typically, the reason behind this is to reduce the required computing time and violation messages from uninteresting areas to speed-up and simplify the runtime and debug processes. Moving into SoC structures makes accurate constraining even more prevalent, as typically the number of different clocks and resets in these systems is greater when compared to flat designs. Most rule-checkers expect some sort of information about the design to identify correct issues, which reduces noise on the exhaustive rule checks reports [21, p. 88].

Constraint file is a powerful tool against a phenomenon called “state space explosion”. The more complex the design is, the exponentially larger the number of possible DUT state combinations. They all need to be verified against the proposed use-case which eventually becomes laborious. This problem for model checking is especially common for systems with many interacting parts or a high number of functional states, such as a data bus [34, p. 1]. This is the main reason behind verification gap in digital design and partly explains the graph evolution in Figure 1 [4, p. 3]. By constraining the DUT correctly, the number of DUT test scenarios can be reduced, effectively limiting the amount of verification to sufficiently cover the realistic DUT use case. The goal is to express the design intent as a list of functional properties [17, p. 51–52].

Correct constraining is instrumental in displaying a correct state of the design, while simultaneously these constraints can hide redundant and unwanted details [35, p. 29–35]. Therefore, understanding the DUT purpose and technical details alike typically require obedient documentation of the DUT. This is important in linting and CDC. However, CDC’s dimensions exceed the structure of RTL due to dynamic clocking required by modern power management systems and evolving synchronization schemes used by design houses [4, p. 162]. In practice this means the constraints would effectively need to change in tandem with the transient operational mode of the design: Whenever DUT is powering on, idle, reset or in operational mode, different constraints can be required.

In addition, EDA verification software defaults to general good practices and design rules. Therefore, it should not make assumptions of analog macros, stability of combinatorial loops, unorthodox synchronization structures, gray-boxed third-party IPs in subsystems or limited use cases of a bus in design. All of these rely on external forms of verification, such as formal, analog, or careful simulation to cover the use case of the feature. With these scenarios in mind, it is safe to say that flexible and accurate creation of constraints is a very important feature for any verification tool, because they are essential for representing the correct verification status while reducing the computational complexity of model checking.

Constraint files are systematic command lists executed as a script. They can be used to exclude and abstract a computationally heavy block from testing, reduce noise by listing false paths unknown to the tool, and link clock signal frequencies with corresponding domains. Also, defining reset signals with their active states, constants with their values, and quasi-static signal paths with no CDC requirements are typical parameters to constrain for a CDC run. Noisy paths can be filtered either in constraints, later in GUI (Graphical User Interface) or in a waiver file. The rest of the related tool configuration can be done outside the constraint file with a scripting language, such as TCL (Tool Command Language) [36].

3.1.2 Incremental flow and reporting

Incremental flow properties will limit the scope of RTL check results to previous design changes. Getting rapid feedback from a verification check limited in this manner could improve the design flow remarkably and decrease the cost of bugs found in the DUT. It is unideal to completely run every verification check after each change, never mind reading daily all the lengthy reports generated by the tools. This type of incremental flow would shorten these reports, showing only the differences in the run results compared to the previous version of RTL. Reports could separate the previously reported issues to a different section and only the new issues would need to be emphasized. This lowers the effort of running the verification flow more often by reducing the designer time and computational footprint [34, p. 1].

Accurate reporting is crucial in ensuring success in verification [17, p. 101]. Metrics enable people who have limited technical knowledge about the verification status of the DUT to understand the design maturity [17, p. 102]. Having a sufficient selection of exportable report formats ready will help the organization find a suitable template for most needs. Having a single easily readable summary report as a baseline can offer most designers with sufficient information. This format could be established as the main report for CI-flow as well. As a more advanced feature, HTML (Hypertext Markup Language) report publishing is also an interesting way of complementing the verification results. It enables categorizing and visualizing the distribution of violations by hierarchical blocks or by violation category to form a solid understanding of the state of verification. In addition, seeing the common denominators of new violations briefly can also help track the root cause of the issue. As mentioned in the setup phase, reports can be used to validate the DUT setup for verification as well. Log files can be parsed for errors, inferred clocks and resets can be specified when found by the tool, and missing port domains can be elaborated faster when there is a list existing about the missing information, which can then be passed on to developers.

3.1.3 Ease of use

User experience is a complex topic and will not be focused on in this evaluation [37, p. 95]. Even if an unintuitive software is functional, any unnecessary learning curve in setup, analysis and debug phases will inevitably widen the verification productivity gap compared to designer throughput [5]. This could also simultaneously reduce the amount of verification passes done for DUT. In that case, not as many users would be competent with the tool, delaying the discovery of unintentional bugs in DUT and thus increasing the design costs according to Figure 1. In the worst possible case, this might cause delays in product releases [10]. Creating quantifiable metrics is difficult in UX (User experience), because even though some of the preferences might be shared between users, typically not everyone agrees with the shared approach. This would require a different research approach overall to produce results. Doing this could, however, be investigated further in upcoming research topics.

Batch runs can most of the time be configured to work similarly to each other. A best-case scenario is to enter a single command and let a script take care of the check because this way the workflow can be kept constant. Fully automatic verification would be preferable, but in practise human intervention is required [34, p. 4]. Standardized workflow between designers with adequate methodology enables results to be compared in a fair manner between runs. Standardizing this approach enables basic status checking by an external checker, who might not know about the functional details of the block to solve the issue by themselves. However, reporting the issue could nevertheless speed up the development process if any previously unknown cases are found. The generated database is openable in GUI after the check is done and the generated plain text report can either be viewed in GUI or in a text editor of choice. It can be difficult to differentiate batch workflows from each other, as their differences are seen mostly in run time duration, terminal verbosity, and output reporting types. These items are also separately configurable and sometimes affect each other: Generating only the required reports can reduce software runtime.

Of course, nowadays a GUI is a main part of an EDA verification tool for debug tasks: Displaying clock and reset trees, dynamic schematics, reorganizing and categorizing violations and reviewing design hierarchies is a completely normal procedure. An integrated source code viewer or editor is commonplace as well. Displaying only the required information for the DUT in a limited screen-space is a daunting challenge, and thus support for modern display setups can boost productivity. There are numerous combinations of views in both lint and CDC GUIs, which would benefit the designer by displaying only the necessary information. Being able to customize the software layout to solve different kinds of issues is beneficial. Even if the GUI layout is not ideal all the time, the information should be accessible effortlessly without leaving the main GUI to find it. This will improve the rate at which the user can understand the different aspects of the issue and its context. For example, automatically emphasizing constant signals and displaying their value in a schematic could speed up the analysis greatly.

3.1.4 Performance

Software responsiveness is an interesting topic. With EDA, software related crashes are unacceptable because editing files in between the analyses in the GUI is important. In an exemplary case of a segmentation fault, all pending changes in RAM (Random Access Memory) are lost for the crashed instance and the changes will have to be re-done, as no pending write operations were done to the non-volatile parts of the file system. This becomes problematic when the program has pending changes in the system memory to reduce file system write accesses during the analysis. This is typically done to improve performance in the GUI client and to reduce continuous system load. This aspect is even more important if the files are accessed over a network, which introduces some systematic delays in file accesses and unpredictable effects caused by shared server resources.

Resources, such as software licenses and RAM, are unallocated when rule checking has ended, and so a longer run time withholds these resources. As a ground rule, the shorter the run time is, the better. However, this comes with some caveats: Dedicating all server computing resources to favour a single process while freezing others is usually an unideal compromise to make, as this invalidates other people’s work. For that reason, granting elevated process priority should be handled with care. In addition, the perceived improvements in run-time are not linear: Being able to finish a daily check overnight is important. Similarly, finishing weekly regression checks within the weekend is also favourable. These conditions allow work to progress normally in a local environment when the results are available at normal working hours. To elaborate, regression runs can hinder other people’s ability to work, whereas overnight runs are crucial for gaining incremental information as feedback about recent changes. While complex, the SoC structure on the other hand typically allows to run the verification jobs on a specific hierarchical level, which can be completed in the set time frame. After that, verifying multiple suitable parts of the design at the same time will enable process parallelization and therefore shared computational resources start to matter more.

Design houses typically have numerous computationally intensive applications running in parallel, such as simulations, power estimations, formal verification runs and so on. These are typically run whenever needed. If the DUT status is known to be interesting in long-term verification, regressive runs could also be implemented. These regressions are run automatically with a timer, for example. Regressive checks could also be done with a differing hierarchical scope with the DUT in it. As today’s SoC designs contain a lot of different functional blocks, this could easily become very computationally intensive if multiple people and CI use it simultaneously. This phenomenon encourages preferring EDA tools with good support and scalability for the used server hardware: Even though there typically is a lot of memory available, some parallel processes could simultaneously heavily use CPU (Central Processing Unit) and RAM in certain use cases, making their availability desired. Therefore, having the EDA software inherently use less resources is preferable, especially if the runtimes do not suffer from the resource scarcity. Even though changing the process priority could help with the issue, any savings in other people’s performance usually come with a run time penalty to the others.

3.1.5 Tool documentation and reporting

A sufficient and transparent documentation of project details along with straightforward EDA flow improve both the quality and quantity of verification passes, eventually refining the final product. Understanding documentation is important, because otherwise the user might not have a clear understanding of the testing methodology: How strict are the applied reference rules, is there overlap in some of the violations, are there possibly uncovered but available checks and so on.

An integrated help viewer in GUI which opens the contained documentation for the necessary setup, violation and methodology information is considered a bare minimum standard. Every tool is slightly different when it comes to these aspects. Results are the most influential part of any verification software as they determine the actions to be taken. Usually, the software is configured according to the standard of an organization or even that of a project. The goal of this decision is to reflect what kind of errors are hoped to be seen in a design. There usually is a small contradiction of interests depending on who is asked, as the verification pass is typically wished to be seen as free of all serious design violations. Simultaneously the designers want to be certain that the software has considered all possible scenarios. Minimizing the runtime with constraints while limiting the number of false positives speed up the debug work. Therefore, the setup affects the observed results. For the most part, the interesting parameters are constraints, the chosen methodology, applied waivers and filters, the hierarchy selected for the DUT, the software version, and the tool config.

The reports should contain at least an output log, a list of all the notifications along with their severity, notification type and the hierarchical path of occurrence. For the transparency of the setup, including a collective list of enabled settings along with the results made by the tool is a good practice. A good addition would be a list of inferred information, which should be validated by the designer of the DUT to help setup the software or the DUT. A summary with violation types and counts will help form a clear picture of the situation of the DUT. Enabling report viewing and regeneration in the GUI to review the changes made during the debug process is recommended. In some tools the reports can also be exported into an online database with HTML or parsed for a CI system to display the status to a wider audience.

These autogenerated reports for comprehensive systems can be very long and monotonous. The intention is to provide a lot of categorized information for each violation while keeping the file readable and accurate. An incremental flow can help a lot when assessing the DUT status while reading a report, as the newer changes are separated from the ones known from the previous run. A good report is ordered by sections to keep the reader aware of the part being handled. This allows the user to focus more on the problem itself.

4. RESEARCH STRATEGY AND ENVIRONMENT

To get a better grasp of some desired features, a study scope enquiry session was held with an expert. To complement this approach, the expert was also asked to fill a structured form, in which each pre-determined feature of the suite is given a weight integer from 0 to 5 represent the execution importance of each mentioned feature in effectively accomplishing a useful verification flow. The sub-sections of the filled chart are covered in the results to provide perspective on important features to be covered in section 4.

The new workflow will be established from the ground-up with new lint and CDC tools for comparison with the old flow. The default development environment is used in the company to offer a comparable view on the server performance. The main testing will be based on the measurement data in which runtimes are recorded along with reported violations and resource usage.

4.1 The study inquiry

In the history of digital design, EDA tools have not been an integral part of it from the beginning [38, p. 6]. To figure out the main reasons for what makes them worth using today, the reasons behind the change in industry trends causing their widespread use today should be understood. As in any evaluation, the benefits should outweigh the drawbacks to justify a change. To answer why this specific EDA tool might be useful, the requirements need to be mapped and then the benefits of each feature must be evaluated in the testing. To help with assessment, an inquiry session was arranged to gain some insight. The interviewed expert also agreed to fill a structured chart to help estimate weight scalars to clarify the important tool focus areas. This chart offers subtopics covered in sections 4.4–4.8.

To understand why EDA tools are popular nowadays, an employee in the company agreed to be asked about the scope of this study and its background. He is regarded as an expert in the field with over ten years of working experience with ASIC and FPGA (Field-Programmable Gate Array) design. In addition, the expert agreed to fill out a structured evaluation chart in Appendix A with weight integers to represent the importance of each feature in his experience. The chart contained some categorized features: For each category, a rounded average integer was estimated. The expert was given four instructions found in the sheet for evaluating the features:

1. “Fill in the weights by importance as an integer from 1 to 5. 0 = NA”
2. ”Cells with thick borders represent the whole (average integer) importance of its sub-cells”
3. “The goal is to observe differences in importance: Avoid giving too high grades on average”
4. “Both lint & CDC are to be evaluated where applicable”

The first rule enables us to clearly qualify the importance of each feature. The greater the filled integer is, the higher the perceived importance is. Filling in a “0” serves an option for not understanding or being aware of such a feature, possibly improving data integrity, as this eliminates any filled guesswork in the results.

Moreover, the second rule clarifies that the collective category grade exists only to group the sub-results into a more readable and generalizable form. The average was underlined so as to not be mistaken for the sub-feature field. It is meant to be a rounded integer which can be left unprocessed, but any difference from the actual average of the sub-grades can be considered as deviating emphasis from the expert’s part. This possibly enables us to get a better understanding of the expert’s point-of-view by computing and comparing the averages.

It is worth mentioning that the point of rule number 3 exists to avoid filling in the maximum grade to each row. Giving a set number of maximum points was considered for this purpose, but that would have risked the integrity of the test, because in that case, the importance of each feature should be in line with each other. However, in this case that is impossible, as some valuable features are enabled by more basic ones and moreover, there is overlap between some of the rows. Also, the exact number of points given should then have been justified by a criterion, which is not representative of the real world: Features, just like the aforementioned point count, are not mutually exclusive. For these reasons, an unrestricted weight assessment was granted. However, a remainder was left to emphasize the differences. The original evaluation chart was provided by the EDA tool provider but has been modified slightly to encompass both the tools’ features.

Finally, the fourth rule allows us to use the data for software evaluation for the applicable parts. It is an acknowledged possibility that these grades deviate for each software: Maybe the reporting accuracy requirements are different between lint and CDC, for example. However, the EDA tool suite has been built to either directly share the features between lint and CDC tools where applicable, or to have been appended only to CDC due to the redundancy in lint. This addresses the possible tool data coupling issue in the results.

4.2 Evaluation environment

In the company, the tested software was configured on a CAD (Computer-Aided Design) server containing CentOS (Community Enterprise Operating System) 7 Linux operating system distribution, which was a system requirement for the new software suite. Despite its age, it is still supported by RedHat, and it provides the required OS (Operating System) libraries for the software to be installed successfully. Servers are of the following kind:

- At the time of testing, the most recent tool version was used.
- The used operating system in CAD server is CentOS 3.10.0-1127.8.2.el7.x86_64 x86_64 GNU/Linux
-
- CAD server contains dual socketed 12-core 64-bit Intel Xeon E5-2680 v3 CPU's running 2.50 GHz with disabled simultaneous multithreading: This has been done to improve security against known side-channel attacks as the architecture is relatively old and considered vulnerable [39]. The allocated 384 GiB of DDR4 should not limit any testing during the quiet hours. Evaluation activity on CAD servers is limited to flow setup tasks, workflow functionality verification and opening the GUI for database analysis, because the Grid Engine does not support GUI operations.
- The recently deployed Altair Grid Engine contains simulation servers with dual-socketed AMD Epyc 7443 CPUs and 767 GiB of system memory for each system. Unlike for the CAD servers, the SMT was enabled for these CPUs. RHEL 8.5 operating system was used instead of CentOS 7. As these new servers were deployed at the time of testing, the load was inherently low, since not many users had yet started using the refreshed Grid during the evaluation period because the old system was still in place. To create a representable picture of using lint & CDC tools for the Grid, the run command was standardized according to the company guidelines with the enabled load balancer, which might affect the initial job submission latency and cause minor deviations in run times. Because the licenses are used in tandem with additional CPU multithreading in both software suites, the number of used CPU threads was limited to 4 in the performance measurements to standardize CPU allocation and not to be limited by the available software license count in either software. Some EDA tools consume

more software licenses when the computation process has been parallelized enough.

The project file system is based on a remote server in another country accessed via SSH (Secure Shell) from Linux remote virtual desktop using TigerVNC viewer and a VPN (Virtual Private Network). It is worth pointing out that this multiple-step infrastructure is expected to introduce some systematic and deviating delays in the user experience, and it is an acknowledged trade-off to improve security while offering access to the files needed by multiple designers from many countries. As the design files are never supposed to leave the server, testing methodology complies to these rules while simultaneously reflecting the real-world experience as well. The following network latency Table 1 was measured with 100 RTT (Round-Trip Time) ping requests with a test interval of one second while VPN was connected for the nearest server:

Table 1: *Measured RTT ping latencies of the deployment.*

Latency	User and VNC	VNC and CAD	VNC and Grid	CAD and Grid
Min RTT [ms]	22	0.12	0.21	0.13
AVG RTT [ms]	22	0.18	0.40	0.25
Max RTT [ms]	37	0.49	8.60	3.40

Two main themes can be concluded from Table 1: The main cause of latency in the network is the secure connection from the end machine to the different servers, all of which in turn are located relatively close to each other. The latency will also multiply in between GUI-interactions, as each command must first reach their destination before a corresponding acknowledgement can be displayed, such as a button press animation after clicking the symbol itself. This makes assessing GUI responsiveness difficult in this testing environment and for that reason measuring GUI action time is out-of-scope for this research.

However, the runtime test methodology will mask the connection latency well, as the start and stop times are checked server-side and then the difference is sent back to the terminal. The main impact of latency in the research is limited to the responsiveness of the GUI: Every action done in the GUI is delayed systematically by the network round-trip latency. However, the software will likely be considered responsive enough, but the deployment type is not ideal for more in-depth user experience testing with action clocking, for example. It is worth emphasizing that every other software deployed this way will suffer the same usability penalty, which is not the software's fault. Overall, the network conditions are steady most of the time, but some temporary deviations are to be expected. This information is used to conduct measurements with average run times and absolute error bars to eliminate the possibility of an outlier result misrepresenting software behaviour.

Verification tasks in the company are run on centralized simulation servers called Altera Grid, which dynamically assign the job a priority and resources by default. They contain an integrated load balancer, causing some initial latency and thus are more difficult to standardize performance metrics on, but are the recommended approach for running heavy command line tasks. This behaviour also justifies using 5-run average results with runtime deviation displayed to conclude performance.

Due to these challenges inherent to this type of deployment, the user interface responsiveness will not be assessed deeply. The GNU (GNU's Not Unix) module deployment is used due to its benefits in the used server deployment. Software data is in this case local for the Grid, enabling low processing overhead and effortless backups without user-specific installations, which also saves on storage space.

For both CDC and lint, the software default methodology was used. This approach is to be complemented with TCL scripts and constraints when deemed beneficial: Some settings were adjusted according to the reporting verbosity, and the software was also instructed on how many DFFs there should be in the synchronizer structure. Also, some custom synchronizer IPs were created to reflect real-world functionality. The goal is not to waive any violations and focus more on constraining the specifications to reduce noise in the results. The software's built-in feature to utilize different design maturity goals was also exploited to reduce result noise. This is analysed in chapters 4.1–4.2. Statistics were also taken for runs in which only clocks and ports were defined, while all standard IP specific waivers were disabled for a fair comparison. A script was used to transform existing design constraints into a TCL constraint form. This was necessary for three reasons: To support the required constraint format, to minimize human error in translation, and to save some time.

A frozen snapshot of a development branch release was used to produce controlled and repeatable results. This approach also enables the utilization of some of the design constraints made for the release and review, if any of the known issues are found by the tool. A local internal Git-based version control was created for redundancy and to better keep track of changes.

The project development is very active, and many people are sharing server resources with each other simultaneously. This in mind, performance tests were run during quiet hours when the resources are plentiful to virtually eliminate the chance of overloading the infrastructure with additional load and to keep the performance bottlenecks typical.

For ideal results, background server activity should be eliminated, but for the nature of this deployment this was unfortunately not possible. This choice was made to show the performance in typical development conditions.

The test was run a on a proprietary functional subsystem, which operates in a SoC and is relatively complex: It contains hierarchical elements, such as a CPU subsystem, memory, analog macros and a lot of I/O (Input and Output interface). This block's RTL was kept constant through testing. From a CDC point of view, these blocks contain tens of different clock signals with varying clock frequencies and over ten different reset signals. The DUT is currently in active development and therefore, it is worth emphasizing that any seen violations will not reflect the status of a finished product. With these factors in mind, thousands of errors were initially expected to be seen. The errors might originate from an unfinished design, incomplete design constraints and human design errors.

4.3 The new workflow setup

Keeping the testing environment simple is important, as it is best for everyone involved that the tests are run correctly. Automating the new workflow will help verify the wanted results [17, p. 122–126]. Makefiles and file lists were created for establishing the basic workflow. The flow contains 4 main commands with the following functionality which are typically run in the following order:

1. “compile”: Gathers synthesis and simulation definitions and design files into a single database to be read later. This could be integrated into the other commands if needed. Uniform commands were used across all testing, as no changes will take place before compilation in the flow testing done for this study. File system paths were instantiated as exported variables and the DUT top level was defined.

2: “setup”: Reads the compiled database and applies constraints into the RTL. Infers and informs about lacking clock and reset constraints and verbosely informs the user about the shortcomings of this setup [40]. As any parameters changed for the testing affect the results from the main CDC run onwards, this command is also considered uniform because all of its inputs are kept constant.

3: “cdc”: Runs chosen methodology goal containing the verification rules for the DUT and generates reports. Creates the database to be loaded into the GUI.

4: “GUI”: Loads previously generated results into the GUI for more nuanced inspection.

In addition, all these Makefile targets were configured to report and save all the encountered issues found while executing the target in order to keep track of the issues.

After a successful compilation, the focus was on specifying and correcting the inferred clocks and port domains with the designers of the DUT. Obtaining a relevant set of specifications proved to be somewhat troublesome. The most problematic part was the unified clock, reset and power management subsystem within the DUT. The available specification of its functionality was limited and consisted mostly of the intentions of the subsystem’s port interface. Constraining this module required more in-depth analysis of the subsystem, as it contained debug backdoors with intentionally missing synchronizers to save on design silicon area and very dynamic clock behaviour with multiple active power domains. The known false positives found in this phase were listed into the filtered results, which are still displayed and not waived. However, these are not interesting, because they are caused by known issues within an unfinished subsystem. The clock signal is gated to save power whenever the reset is active, which allowed the filtering of some of these known false error generating paths away from the results. An IP design specification revealed that these issues have been waived previously and a separate confirmation about the validity of these waivers was received. In this testing, waivers had been disabled to also simplify the comparison between the software.

The installation was made available to everyone on CAD servers by using a GNU module with authorization from IT-department. This approach also enabled the use of the Altair Grid Engine. A verification flow was deployed in a hierarchical subsystem of the SoC. The most recent version of the DUT was chosen to stabilize the environment, since the main development branch may contain unexpected, and in worst case, setup flow crippling changes. The constraints had also been standardized for the release, which also simplified the setup process.

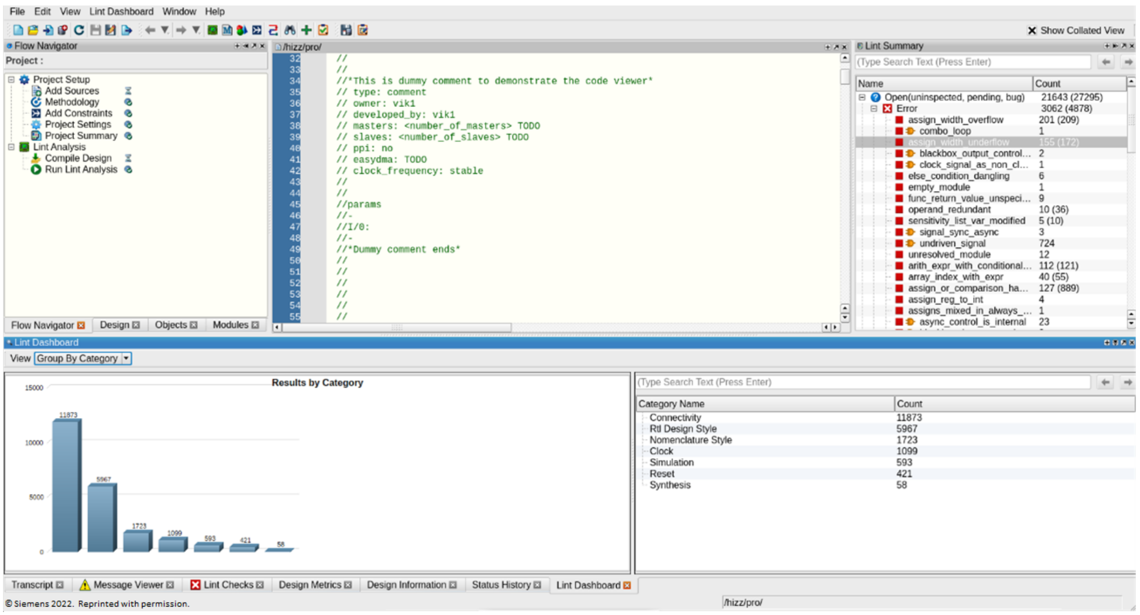
The constraints were initially derived from a two-month-old release. However, they were not complete and needed some clarifications. Hence, revised constraints were requested. In addition, the modularity of the design flow was improved with a block for which the constraints were already available and documented. This also allowed the verification of the correct functionality of the software, as the correct base for the constraints was available. Version control was implemented for the project to track recent changes made in the workflow.

The company’s proprietary synchronizer IP blocks are based on edge-case design rules made possible by the process node guidelines. As the EDA tool is not aware of the process-specific design technology, all these special synchronizers were initially flagged as dangerous by the tool, which is expected behaviour. However, this caused a problem, as thousands of design violations caused unwanted “noise” in the results. This made it more difficult to find the real issues. To get more accurate results, this noise needed to be fixed, preferably without using waivers. There is always a possibility to gently mislead the tool by creating a custom RTL synchronizer and to use a global macro to load the customized units instead of the proper synchronizers. After first discussing the issue, a suggestion to use customized synchronizer gray-boxes modelled with a TCL-script was implemented. This means that the interfaces were defined exhaustively along with their mutual relations. This replaced the need of using macros and custom-made RTL to replace the verified structure, which would then have needed to be verified separately to be used exclusively in CDC runs. The models specified the proven outputs for the given inputs of the synchronizer blocks. A model was created for each synchronizer unit to cause a pre-specified violation. The modelling required some detective work with the designer of power and clock domains, but eventually the script was able to represent the synchronizers’ functionality. After that, the CDC tool was run and the GUI was used to elaborate the DUT status.

4.4 GUI

Because the interface in both tools is based on the shared GUI, many features resemble each other, even though the verification tasks of the tools are different. Keeping shared features in similar places across the different software suites can help the user adapt to new verification environments. This section is for presenting the GUI layout and features which are displayed in the following Figures 8–11.

In Figure 8, the open subcategories on the top half are Flow Navigator (top left), source code inspector (top middle), and Lint Summary (top right). The bottom half has the Lint Dashboard open, which contains result visualization by error category (bottom left), and its data in written format (bottom right).



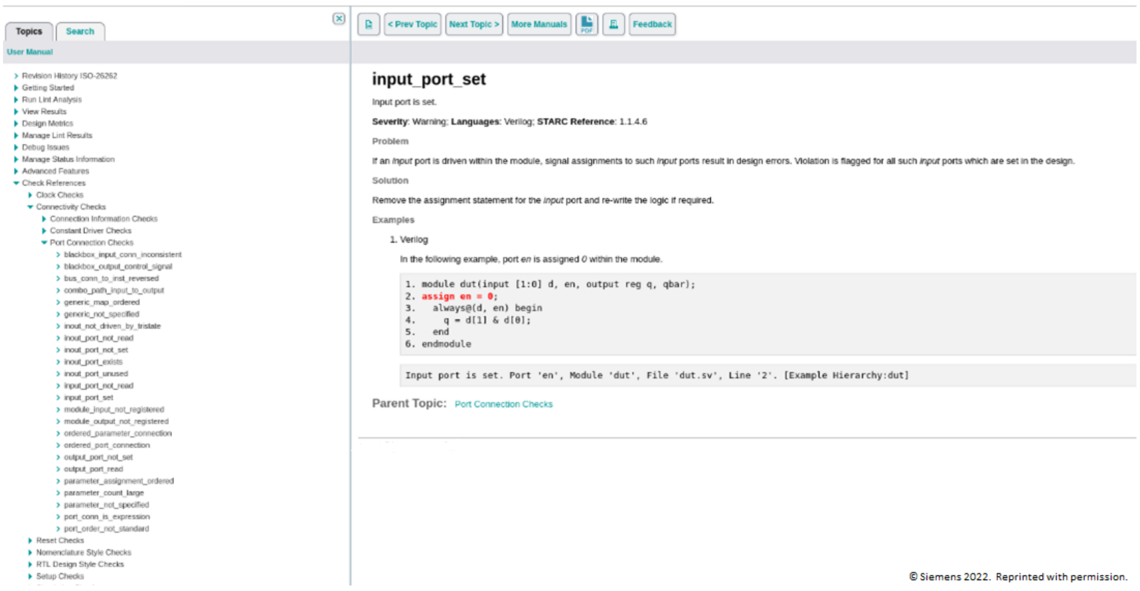
```
//
//
//*This is dummy comment to demonstrate the code viewer*
// type: comment
// owner: vik1
// developed_by: vik1
// masters: <number_of_masters> TODO
// slaves: <number_of_slaves> TODO
// ppi: no
// easydma: TODO
// clock_frequency: stable
//
//
//params
//-
//I/0:
//-
//*Dummy comment ends*
//
//
//
//
//
//
```

Name	Count
Open(uninspected, pending, bug)	21643 (27295)
Error	3062 (4878)
assign_width_overflow	201 (209)
combo_loop	1
assign_width_underflow	155 (172)
blackbox_output_control...	2
clock_signal_as_non_cl...	1
else_condition_dangling	6
empty_module	1
func_return_value_unspeci...	9
operand_redundant	10 (36)
sensitivity_list_var_modified	5 (10)
signal_sync_async	3
undriven_signal	724
unresolved_module	12
arith_expr_with_conditional...	112 (121)
array_index_with_expr	40 (55)
assign_or_comparison_ha...	127 (889)
assign_reg_to_int	4
assigns_mixed_in_always_...	1
async_control_is_internal	23

Category Name	Count
Connectivity	11873
Rtl Design Style	5967
Nomenclature Style	1723
Clock	1099
Simulation	593
Reset	421
Synthesis	58

Figure 8: *The new lint tool GUI main window.*

In the following Figure 9 the help viewer is displayed. The documentation navigator menu is visible on the left side. It should be noted that a search tab is also available. Documentation is available in both PDF (Portable Document Format) and HTML data formats. This documentation HTML file has been opened in the default web browser of the operating system.



- Revision History ISO-26262
- Getting Started
- Run Lint Analysis
- View Results
- Design Metrics
- Manage Lint Results
- Debug Issues
- Manage Status Information
- Advanced Features
- Check References
  - Clock Checks
  - Connectivity Checks
    - Connection Information Checks
    - Constant Driver Checks
    - Port Connection Checks
      - blackbox_input_conn_inconsistent
      - blackbox_output_control_signal
      - bus_conn_to_inst_reversed
      - combo_path_input_to_output
      - generic_map_ordered
      - generic_not_specified
      - inout_not_driven_by_tristate
      - inout_port_not_read
      - inout_port_not_set
      - inout_port_exists
      - inout_port_unused
      - input_port_not_read
      - input_port_set
      - module_input_not_registered
      - module_output_not_registered
      - ordered_parameter_connection
      - ordered_port_connection
      - output_port_not_set
      - output_port_read
      - parameter_assignment_ordered
      - parameter_count_large
      - parameter_not_specified
      - port_conn_is_expression
      - port_order_not_standard
  - Reset Checks
  - Nomenclature Style Checks
  - RTL Design Style Checks
  - Setup Checks

input_port_set

Input port is set.

Severity: Warning; **Languages**: Verilog; **STARC Reference**: 1.1.4.6

Problem

If an *input* port is driven within the module, signal assignments to such *input* ports result in design errors. Violation is flagged for all such *input* ports which are set in the design.

Solution

Remove the assignment statement for the *input* port and re-write the logic if required.

Examples

1. Verilog

In the following example, port *en* is assigned *0* within the module.

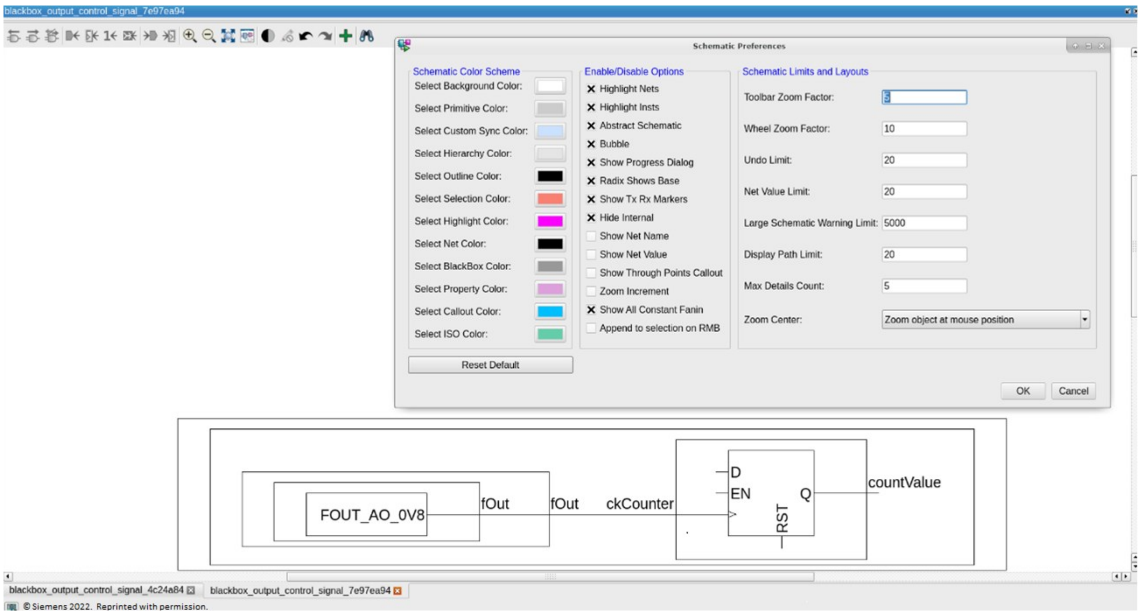
```
1. module dut(input [1:0] d, en, output reg q, qbar);
2. assign en = 0;
3.   always@(d, en) begin
4.     q = d[1] & d[0];
5.   end
6. endmodule
```

```
Input port is set. Port 'en', Module 'dut', File 'dut.sv', Line '2'. [Example Hierarchy:dut]
```

Parent Topic: Port Connection Checks

Figure 9*: Lint tool help viewer about an input_port_set error documentation.*

In the following Figure 10, the schematic viewer is demonstrated. The empty looking boxes in the schematic represent abstracted hierarchical levels, as viewing their details further is not relevant for this type of error. The user, however, is free to inspect unrevealed information by double-clicking each box or line.


blackbox_output_control_signal_7e97ea94
Schematic Preferences
FOUT_AO_0V8
fOut
fOut
ckCounter
D
EN
Q
RST
countValue
blackbox_output_control_signal_4c24a84
blackbox_output_control_signal_7e97ea94

Schematic Color Scheme

Select Background Color:	⬜
Select Primitive Color:	⬜
Select Custom Sync Color:	⬜
Select Hierarchy Color:	⬜
Select Outline Color:	⬛
Select Selection Color:	🟥
Select Highlight Color:	🟪
Select Net Color:	⬛
Select BlackBox Color:	⬜
Select Property Color:	🟪
Select Callout Color:	🟦
Select ISO Color:	🟩

Enable/Disable Options

- ✓ Highlight Nets
- ✓ Highlight Insts
- ✓ Abstract Schematic
- ✓ Bubble
- ✓ Show Progress Dialog
- ✓ Radix Shows Base
- ✓ Show Tx Rx Markers
- ✓ Hide Internal
- ☐ Show Net Name
- ☐ Show Net Value
- ☐ Show Through Points Callout
- ☐ Zoom Increment
- ✓ Show All Constant Fanin
- ☐ Append to selection on RMB

Schematic Limits and Layouts

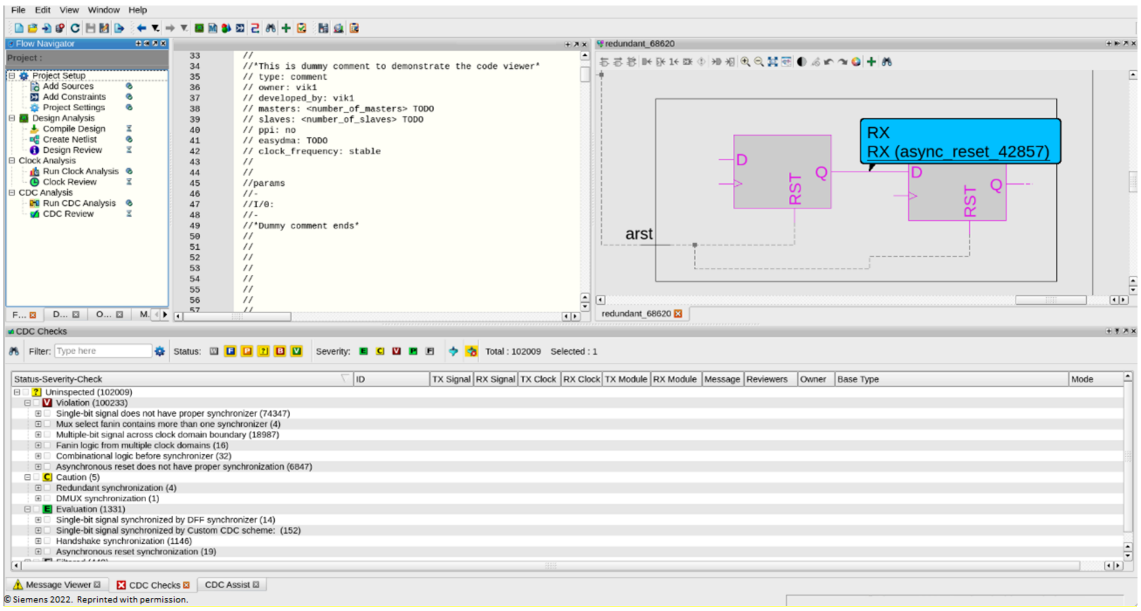
Toolbar Zoom Factor:	5
Wheel Zoom Factor:	10
Undo Limit:	20
Net Value Limit:	20
Large Schematic Warning Limit:	5000
Display Path Limit:	20
Max Details Count:	5
Zoom Center:	Zoom object at mouse position

Reset Default   OK   Cancel

© Siemens 2022. Reprinted with permission.

Figure 10: *Lint schematic viewer about a black-box error with default customization preferences menu also visible above.*

In the following Figure 11 the CDC tool main window is displayed. The open subcategories are code inspector (top right), CDC Checks (bottom) and Flow navigator (top left). Notice that the black space in CDC checks is reserved for detailed violation data, such as RX signal etc. Multiple enabled status indicators and the number of filtered violations with design specifications should be noted.



```
//
//*This is dummy comment to demonstrate the code viewer*
// type: comment
// owner: vik1
// developed_by: vik1
// masters: <number_of_masters> TODO
// slaves: <number_of_slaves> TODO
// ppi: no
// easydma: TODO
// clock_frequency: stable
//
//
//params
//-
//I/0:
//-
//*Dummy comment ends*
//
//
//
//
//
//
//
//
```

CDC Checks

Filter: Type here   Status: W F P ? B V   Severity: E C V P F   Total : 102009   Selected : 1

Status-Severity-Check	ID	TX Signal	RX Signal	TX Clock	RX Clock	TX Module	RX Module	Message	Reviewers	Owner	Base Type	Mode
Uninspected (102009)												
Violation (100233)												
Single-bit signal does not have proper synchronizer (74347)												
Mux select fanin contains more than one synchronizer (4)												
Multiple-bit signal across clock domain boundary (18987)												
Fanin logic from multiple clock domains (16)												
Combinational logic before synchronizer (32)												
Asynchronous reset does not have proper synchronization (6847)												
Caution (5)												
Redundant synchronization (4)												
DMUX synchronization (1)												
Evaluation (1331)												
Single-bit signal synchronized by DFF synchronizer (14)												
Single-bit signal synchronized by Custom CDC scheme: (152)												
Handshake synchronization (1146)												
Asynchronous reset synchronization (19)												

© Siemens 2022. Reprinted with permission.

Figure 11: *The new CDC tool main window.*

The GUI was perceived to be intuitive, even though this feature is difficult to demonstrate with the previous pictures. Now that the tools are working properly, the software's capabilities can be assessed when compared to the old tool flow.

5. ASSESSING DIFFERENCES

The features requested by the company are investigated in this section. The measured performance and violation numbers will help quantify the software, while the categorization of errors helps qualify the violations. These factors help form a representable picture of the software’s capabilities in typical verification activities. Different methodology goals are tested to observe the noise control strategy within the software. Finally, setup experiences are gauged with respect to the structured review chart of Appendix A.

The proprietary synchronization structures were implemented with a TCL script to reduce false issue reporting in the results. This reduces overhead in the server environment, as then global macros for alternative synchronizer RTL implementation for CDC checks are not required. Functionality of hierarchical CDC checks was tested successfully and implemented with a method similar to the old workflow. In hierarchical flow, any sub-IPs are converted into gray-boxes with specified I/O relations, which can be used as simulation models for the next hierarchical block in line. Hierarchical lint is not available in the provided version of the software and therefore, it is not explored in this research.

Incremental runs were configured to work with lint. For this purpose, several sub-IPs were masked as hierarchical units to pass the constraints to the highest hierarchical DUT component. This was done for the sake of this test, and it masked a small part of the violations observed before. The default Makefile command was run to create a reference database for incremental inspection. Then, the sub-Ips were unabstracted again, revealing all the previously masked violations in the IPs. After that, a modified Makefile command was run to create a comparison database with more violations embedded into it. A separate report was finally generated to first address the incremented issues, which had been previously masked. In addition, the rest of the violations were also listed, but as previously known issues, which were separated to a later section of the report. This property makes incremental reporting purely complementary, as it is not hiding any results by itself.

A number of tests were run with the new verification tools. The tests are shown in the following table 2:

***Table 2:** All practical test cases in their respective order before addressing the items in Appendix A.*

Test #	CDC	Lint
1	Subflow length	-
2	Total flow length	-
3	Error categorization	-
4	Noise reduction	-
5	-	Subflow length
6	-	Total flow length
7	-	Error categorization
8	-	Custom ruleset coverage
9	Resource consumption	Resource consumption

The performance measurements were taken from the new workflow with different check methodology goals to better understand the thoroughness of testing compared to the achievable run times. Using this feature is beneficial in hiding most of the unwanted false violation, as typically the root cause violations are displayed in the “Start” methodology goal, which would likely cause multiple issues when using more exhaustive methodology goals. The findings were compared against the known issues found with the original flow. It is worth noting that one of the improvements was to separate the compilation from the checking targets to reduce the application runtime.

In preliminary testing it was noted that “Compile” and “Setup” sub-flows were not affected by the set methodology goal level outside the testing margin of error. This was true for both tools. According to the evaluation provider, this behaviour is to be expected, as the mentioned targets receive identical inputs regardless of the goal and processing them is identical regardless of the target. This means that the main CDC target runtime was left to be inspected with different methodology goals. Tool workflow is introduced in the following Figure 12.

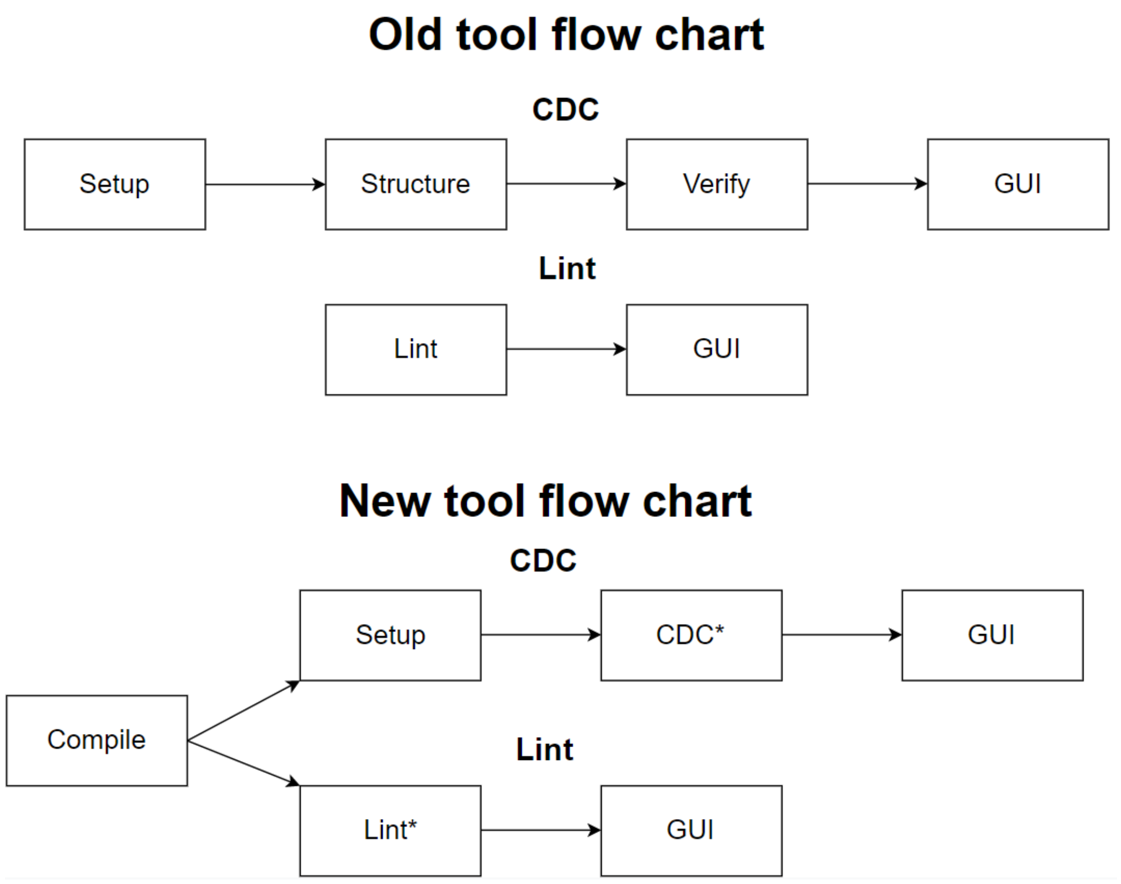


Figure 12: *New and old tools flows. Sub-flows are represented in separate boxes. The sub-flow part names ending with *-symbol are run with multiple different goals.*

Deviating load on the servers causes small runtime deviations emphasized by the dynamic load balancer in the server environment. For this reason, error modelling with 5 runs was added to improve the statistical value of the results.

IP methodology was used throughout the testing with a deviating goal. It is important to note the increased number of violations caused by an increased number of CDC checks in the main verification run to know how influential the used methodology goal is for results, noise and run times. The names of the goals represent the proposed phase of utilization for them in the IC design flow.

5.1 CDC performance and violations

The following Figure 13 was created from the results. In the figure the runtime difference of the coarsest and most exhaustive default goals (Start versus Release), which is about 28.3 %, can be observed. The effect on the total flow length between using these two goals is 12.0 %, which contains the measured 5-run averages of Compile, Setup, CDC and GUI added together for the total flow time. Even though these steps might not be run sequentially one after another in an actual analysis, this still represents the total time required for a complete analysis. The CDC methodology goals are mutually exclusive with each other and only one of them must be considered for a total flow duration at the time. The displayed error bars thorough this research represent the absolute minimum and maximum results for statistical value.

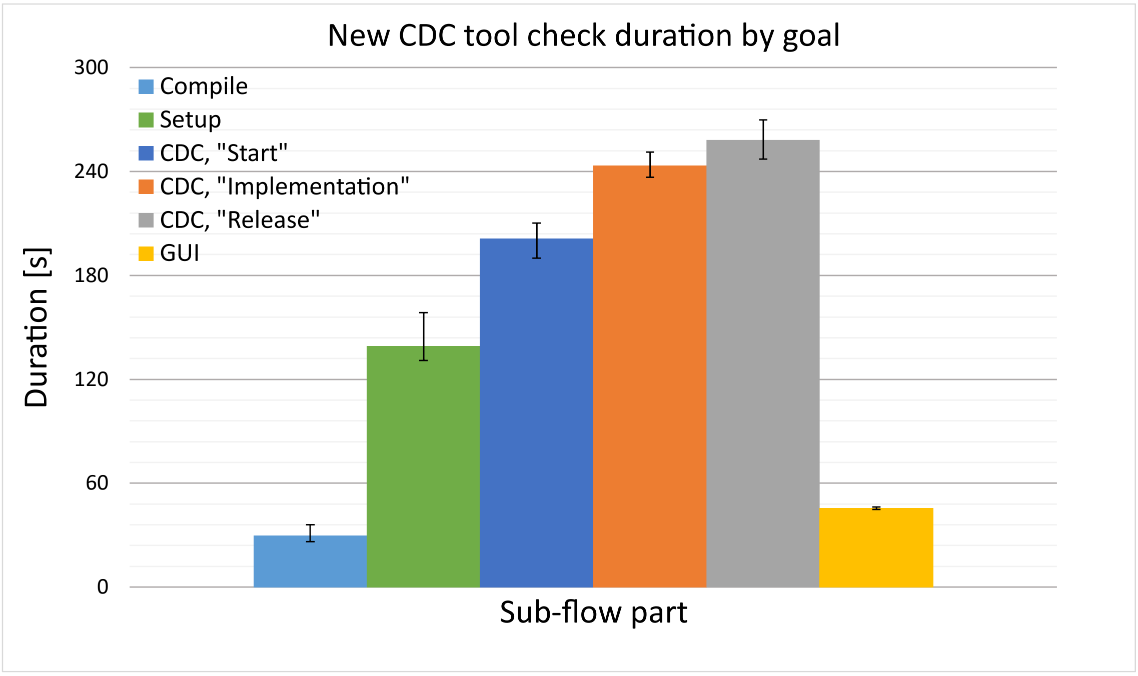


Figure 13: *The new CDC sub-flow run times.*

Even though this difference in “Start” and “Release” goals is not much in absolute time, it is worth noting that DUT is merely a subsystem of a whole SoC and therefore, the relative result is more interesting, as the design will scale further. It can be concluded that the absolute run time is not a major limitation in using the Release goal in either this subsystem or in a simpler one with the new CDC tool.

For comparison, the old flow run times were also recorded. None of the sub-flow results are directly comparable with the ones observed in the new workflow but the total results are compared to each other to evaluate the differences between them. It should be noted that the compilation is done in each run of this flow, which affects the results negatively. The old flow contains an integrated compilation in every phase and a corresponding “structure” goal does not exist in the new flow. Therefore, these phases could not be comparably exported into the previous Figure 13, because of the unidentical partitioning of the flow. The old workflow sub-flow parts and their total sum are represented in the following Table 3, in which a separate table from the new flow results was formed due to the differences in each sub-flow. However, the total runtime is to be compared later in Figure 15.

Table 3: *The old CDC workflow part length 5-run averages categorized and combined.*

Old workflow	Setup	Structure	Verify	GUI	Total
Time [s]	267.323	328.950	1212.660	29.863	1838.796

Here the length differences in the total runtime can be observed, which are mainly caused by the comparatively lengthy execution of "Verify" sub-flow. Every checking job is slower in the old workflow when compared to the new flow. Opening the GUI, however, is consistently faster on the old flow. The results have been compiled into a graph in Figure 14 to clarify the differences.

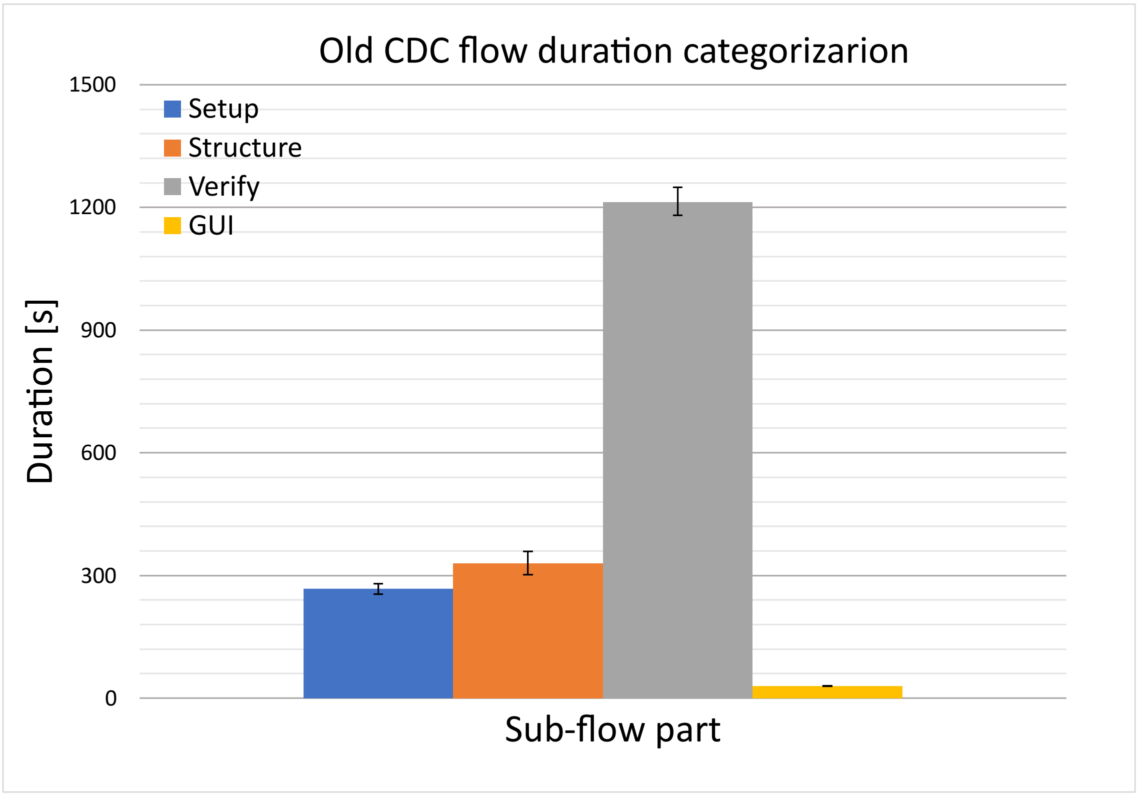


Figure 14: *Old flow part run times measured in seconds.*

The sums for both workflows were then computed. For the new flow, the most comprehensive default "Release" goal was chosen for this purpose to display a scenario with all the recorded violations and the maximum run time. In the following Figure 15 the total durations of the workflows with each other are compared. The previously noted "Verify" goal for the old flow delays the completion considerably and contributes to the new flow being about 389 % faster. The other sub-flows were more closely aligned in duration with each other across the software.

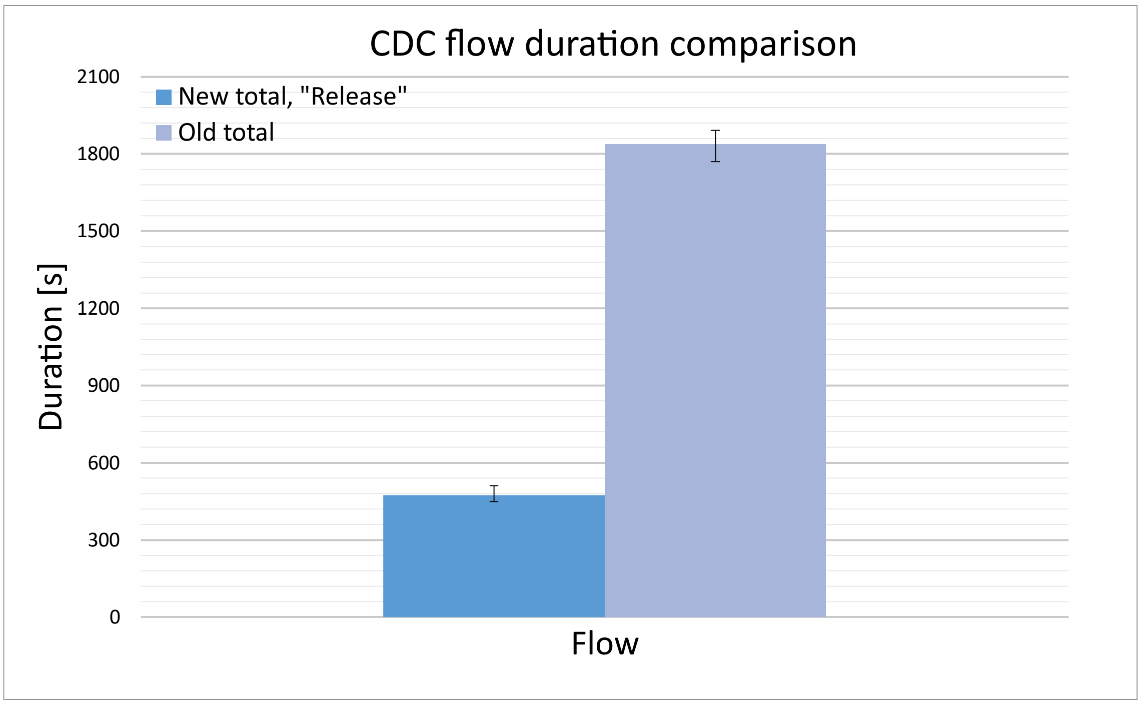


Figure 15: *Total flow duration both CDC workflows.*

Next, the detected violation types by category observed in the new workflow are summarized. According to the manual, violations are categorized as problematic CDC schemes. Cautions consist of potential problems. Evaluation keeps track of detected and accepted schemes which are not considered problems by the tools on their own. Proven status counts schemes with proven protocol assertions. The Filtered section encompasses all the automatically and manually gathered false positives caused by limited use of the signal path in specifications, which the software cannot assume.

Constant CDC paths are filtered. The filtered list is not exhaustive. However, it is still identical between the tests, since its only purpose is to display how well earlier methodology goals supress unwanted noise. The new CDC tool reported items are presented in the following Table 4:

Table 4: *Reported violations in the new CDC tool sorted by severity.*

Type	Start	Implementation	Release	Old flow
Violation	267	625	5073	296
Caution	9	9	514	524
Evaluation	229	1322	1332	8302
Proven	4	4	4	-
Filtered	456	19254	19308	-

From Table 4, it can be concluded that filtering false violations from the results while using different methodology goals results in a different proportion of filtered violations. Without using waivers, “Start” contains 47.3 % of false messages, which cumulates quickly when unaddressed violations contribute to a more comprehensive methodology, resulting in a 90.7 % noise ratio in “Implementation” and a 73.6 % noise ratio in the “Release” goal. As the “Release” goal includes more rule checks to increase the violation count, the noise ratio drops consequentially. The main objective of using different methodology goals is to clean the DUT from the most typical root causes first, which would cause multiple errors later while using a more comprehensive goal.

The old workflow uses similar violation classification categories, and it yielded the results available in the previous Table 4. The old flow classifies most of the issues into the “Evaluation” category. The old workflow does not contain corresponding data classification types for “Proven” and “Filtered” issues. The results of Table 4 have been compiled into a single graph in Figure 16, which displays all previously reported items. The issue type numbers vary within multiple orders of magnitude, and for that reason an exponentially increasing Count axis was created for the figure.

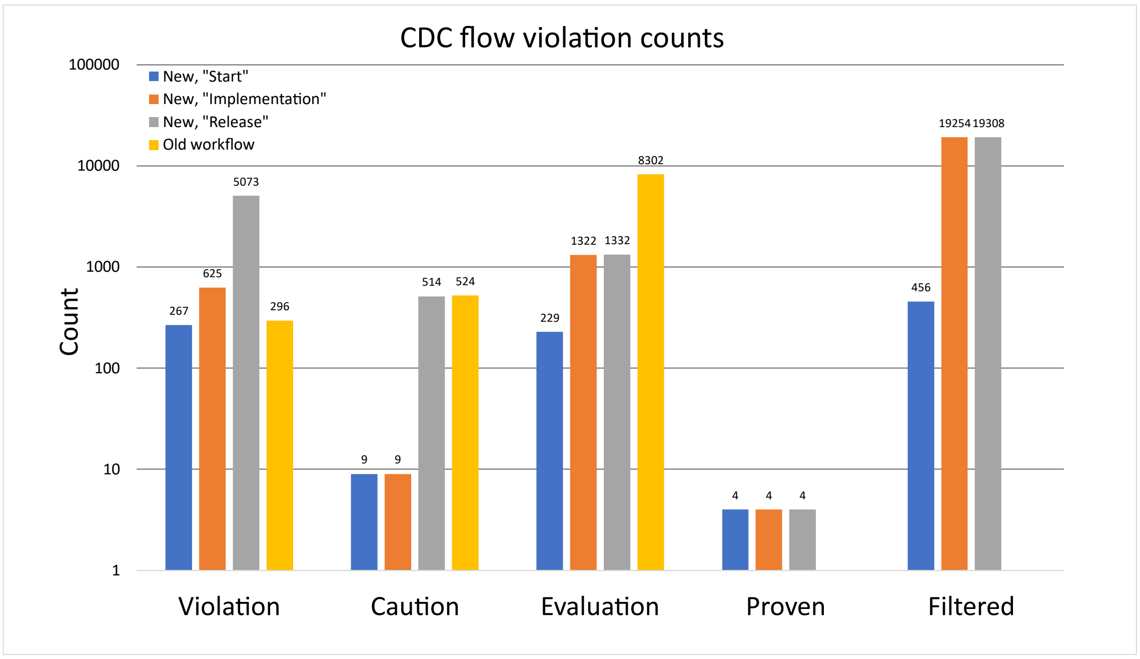


Figure 16: *CDC violation results from both workflows.*

From the results it can be concluded that the new CDC tool is able to report more total issues when compared to the old flow used in the company. Methodology goals can be used to keep the resulting noise in control while the DUT is still in development. The new CDC tool is also remarkably faster when compared to the old flow.

5.2 Lint performance and violations

In the following Table 5, it can also be seen that choosing the correct methodology goal will reduce the run time in lint. Lint compile time is different from the CDC due to an additional lint flag required by the compiler. Like with the new CDC tool, the methodology levels are mutually exclusive in the flow and according to run duration Table 5, it has some effect on run durations. Interestingly, the “Implementation” goal takes consistently longer to complete, even though it should have less enabled checks compared to the “Release” goal, which in turn had shorter run times. The 5-run average absolute error margins imply that this is a reliable result, and this behaviour was re-tested later with similar results, which means the original measurement results were kept. The Compile and GUI results were shared between the tests.

Table 5: *The new lint tool runtime results*

New flow runtimes	Compile	Lint	GUI	Total
Start	18.930	214.602	33.582	267.114
Implementation	18.930	276.621	33.582	329.133
Release	18.930	254.931	33.582	307.443

To clarify these results, the following chart was compiled from the results in Figure 17. Like in the CDC tool, “Start”, “Implementation” and “Release” are mutually exclusive goals in lint to reduce noise in different phases of design and only one of them shall be used at the time.

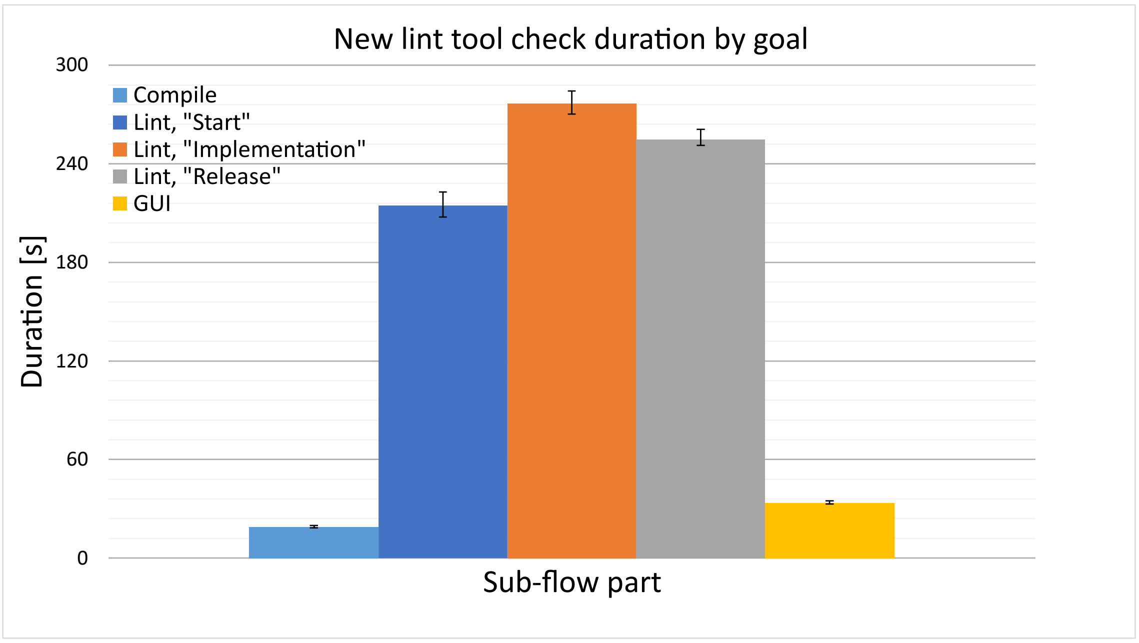


Figure 17: *The new lint tool sub-flow durations.*

For total lint flow, the data of “Release” goal was compared, for it has the most comprehensive checks within the tool. The data collected for Figure 18 shows that the new lint tool is also measurably faster in these checks, and it would be even faster with a reduced methodology goal, according to the previous Figure 17. However, the speed benefit is less pronounced in lint when compared to the difference in CDC runs. Generally, the length of the whole new flow is slightly shorter in lint than in CDC.

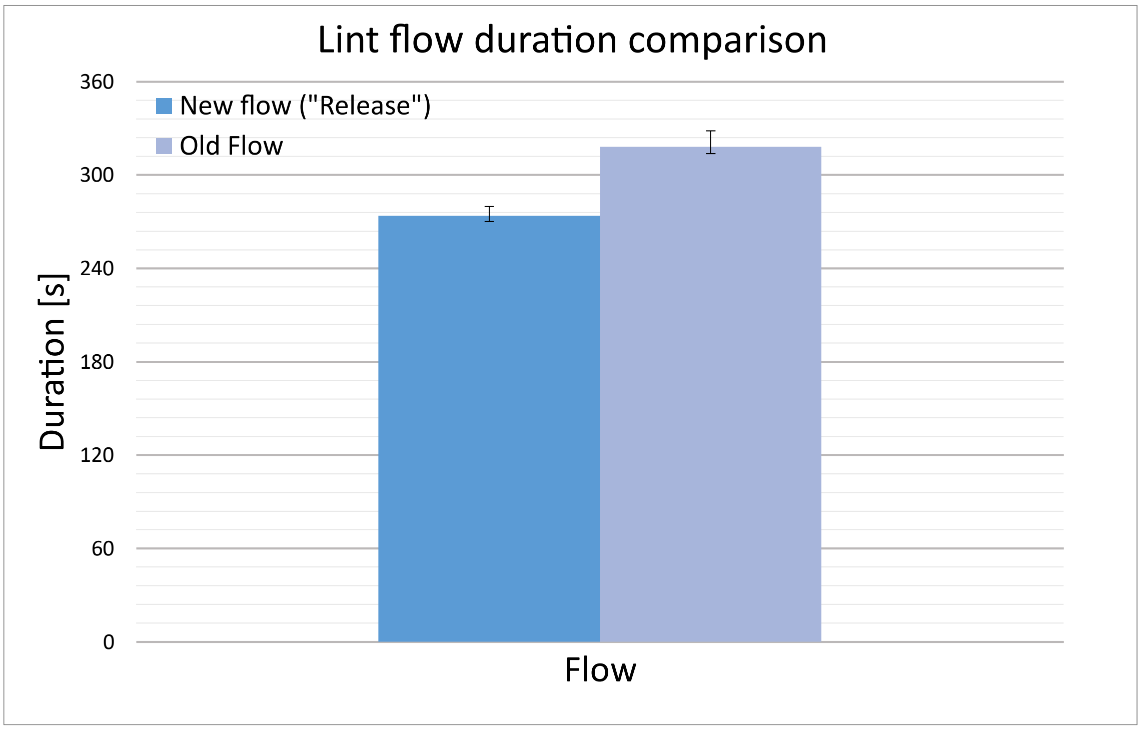


Figure 18: *Lint check pass durations from both flows.*

With differing methodology goals, the lint issue counts represented in the following Table 6 were observed. The detected violation counts increase when changing the tool, and most of the issues are categorized with “Caution” in the old flow. Also, all the issues found in the old lint low were found in the new flow with all enabled rules, albeit with different severity. However, even the “Release” goal was unable find every specific error from the old flow and so rules outside of that specific goal would need to be considered if ruleset parity was important. For that purpose, generating a customized ruleset is recommended to standardize accuracy and reduce runtimes: Server load, licence utilization and resource consumption are all reduced in conjunction. If required, differing severities between flows are easy to fix with a simple ruleset adjustment. Implementing new rules altogether is a much more laborious ordeal in comparison.

Table 6: *Lint violation counts*

New flow runtimes	Old flow	Start	Imple-mentation	Release
Violation	9	274	3197	3456
Caution	6624	1953	10788	10740
Evaluation	22	3728	7454	13669

The data from Table 6 has been compiled into the following chart to improve readability in Figure 19.

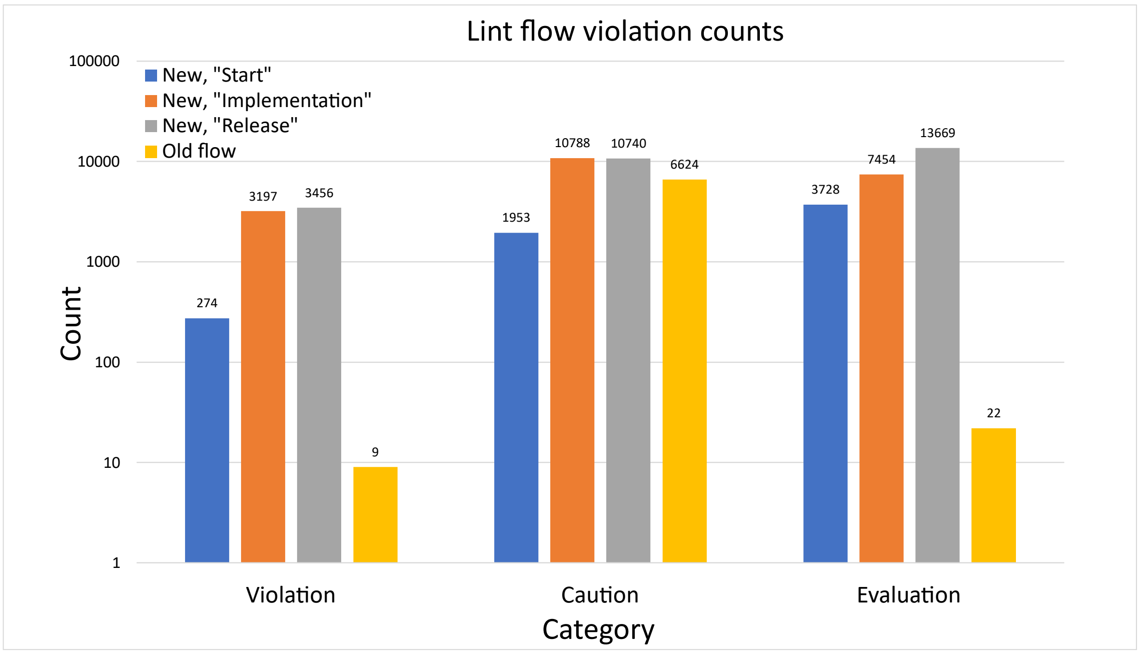


Figure 19: *Lint issues of both lint flows in a bar graph. Note the exponential "Count" axis.*

For reference, all of the goal related messages were summed together to estimate the tools' abilities to find issues in the design, because the severities were not consistent between tools. The old flow found a total of 6655 issues, whereas the new lint tool found 5955 with "Start", 21439 issues with "Implementation" and 27865 issues using the "Release" goal. According to the testing, the new lint tool "Implementation" goal should only be used to reduce noise, as the run times took consistently the longest time to finish while using the goal in question.

Lint was verified to find all the errors listed by the old tool flow with the customized ruleset. Therefore, it can be concluded that new lint flow can find every error discovered by the company's current verification flow. More issues can also be found while having lower runtimes. This is an excellent basis for custom ruleset creation, which could reduce the runtime and resource allocation further due to the reduced number of checks taking place during the verification run.

5.3 Server resource utilization

In the company, these tools are run in a centralized server environment, which means that the computation servers pass any excess resources to another user who might need them. System memory and disk space along with computation power are finite resources. Using them sparingly allows the deployed infrastructure to last longer without further investments into additional hardware. Lower runtimes could potentially also reduce licence usage. In the following Table 7, the recorded disk usage from the tool installations can be seen. Also, system memory allocation while running the checks or the memory consumption when the database has initially been loaded into GUI can be observed.

Table 7: *Server resource consumption measured while the tests are running and while the resulting database has been opened in the GUI.*

Resource property [MB]	New workflow	Old workflow
Installation size	7079	25528
CDC check RAM usage	2212	7681
CDC DB in GUI, RAM usage	9033	26587
lint check RAM usage	2181	5025
lint DB in GUI, RAM usage	10164	13918

The installation of the new tool consumed 7.08 GB of storage space. Because the old tool requires 25.5 GB of storage space, this is considered an improvement of 360.6 %. However, due to the module-based nature of the deployment, the impact of this difference can be considered small, since these modules are installed only once for the software to be available to everyone with development access. Therefore, the difference will not cumulate as a function of the user base, and the effective financial impact of acquiring the differing amounts of storage is limited even for more costly memory types [41].

We can also observe that the system memory allocation was greatly reduced for the already shorter run time, regardless of the test type. CDC runtime memory allocation was reduced by 71.2% and GUI allocation by 66.0 %. In lint, the memory footprint reduction during the runtime was 56.6 % and in GUI 27.0 %. The combined effect of reduced memory allocation and shorter runtimes means that less memory is reserved for the check overall, and more resources can be utilized elsewhere on the computation servers, which are collectively used by the users.

The contents have also been compiled into a chart for improved readability in Figure 20. A general observation can now be made: The new workflow with the new CDC tool used remarkably less server resources in our tests. For the new tool, the “Release” methodology goal was used to model a worst-case scenario, as it uses the most rules for the verification pass.

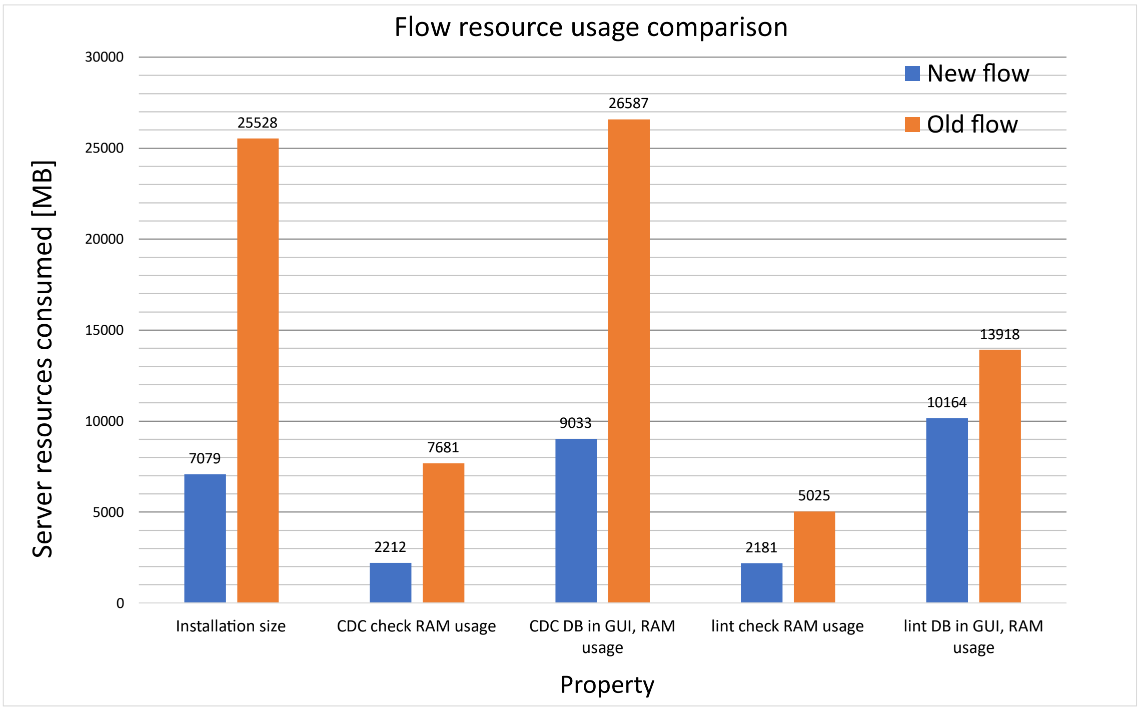


Figure 20: *A comparing bar chart for visualizing the resource allocation for both the software suites based on the previous Table 7.*

According to the previous results, it can concluded that the new tool suite consumes 27.7–73.0 % of the memory reserved by the old workflow depending on the task, which in turn could enable running multiple instances in parallel to reach the memory usage of the old workflow. This is a possible scenario in such a server deployment. This could also improve the sufficiency of computation resources now and in the future.

5.4 Methodology & Flow

In Appendix A, the expert emphasized some software features over the others. Even though user experience is not quantified in this research, remarks are made on each subtopic to cover the table in Appendix A. The following sub-sections of this chapter are directly related to the contents of Appendix A within the scope of this research. Some aspects of Appendix A were considered out-of-scope for this research, such as dynamic CDC, RDC (Reset-Domain Crossing), UPF (Unified Power Format) analysis and FPGA library support, as replacing and evaluating the old workflow with the new one is the main assignment.

The new workflow offers multiple levels of methodology, for example IP and SoC levels with a different emphasis on quality checking. The chosen DUT determines the recommended methodology. The DUT design maturity, on the other hand, determines the recommended methodology goal, which helps find root causes of typical errors that can be found in a more comprehensive methodology goal later. Therefore, correcting typical root causes first helps correct numerous violations later. This helps with result noise control over the design period. This way the user should be exposed to less noise overall, which reduces engineering time, verification gap and design costs according to Figure 1.

The CDC verification methodology “identifies errors using structural analysis to recognize clock domains, synchronizers, and low-power structures via the Unified Power Format (UPF). It generates assertions for protocol verification along with metastability models for re-convergence verification. All properties and design intent are inferred by the software.

The technology checks all potential CDC failures, statically verifying that all signals crossing asynchronous clock domain boundaries are guarded by CDC synchronizers. It then illustrates DUT issues found with familiar schematic and waveform displays.” [4, p. 166]

The software documentation was clearly categorized. All necessary information related to design quality checking using new tools was found either by categorization or by using the available search function. Error messages are logically labelled in the new software suite, and each type of error is given an RTL code example in the documentation where applicable. Documentation is available in both PDF and HTML formats. Integration with current flow was overall simple with the new suite. Both CDC and lint tools were setup quickly, once a modular Makefile based design flow with necessary file lists and parameters was formed. The tool integration was presented in depth in chapter 3.3. It is worth emphasizing that even though no action blocking issues were encountered in integration, symbolic file system links used in the project file system were not supported right away, so a workaround was developed for that purpose. This was fixed two weeks later in a following software patch.

5.5 Quality of results

Static CDC and lint were evaluated in this research. In CDC, the DUT is considered clean of CDC violations. Should this be the case, the old flow was found to be less noisy. In test runs, no waivers were used to disregard any errors in either flow. However, the assumption of the DUT having no CDC errors is based on the old flow not observing any violations. Therefore, it is possible to find completely undiscovered violations using the new workflow, which would have to be verified with the lead designer of the DUT. However, such engineering time is not readily available, but the effectiveness of the flow noise reduction capabilities was witnessed in Figure 16. If there indeed are no CDC issues in the DUT, then by choosing the “Start” goal it could be concluded that the new CDC tool can have less noise than the current verification flow if the goal is picked accordingly. For CDC, the company uses the tool default settings for verifying design correctness, whereas in lint the company has compiled a customized rule set for the purpose. The CDC rules were structured differently between the new and the old flow, as some of the checks had different levels of overlap in the violations. There also was not a golden rule set for CDC rules unlike for lint in the old tool. For these reasons, the direct error comparison is not deemed beneficial on a DUT which is CDC cleaned with the old tool. For these reasons, the default rule sets were inspected by analysing error categorization and the noise suppression capability of the new tool.

In lint, the DUT was not free of violations and the list of known errors was available. All the known errors were found when every check was enabled. This included checks outside the domain of “Ip Release“ coverage, so it can be concluded that the software is competent in finding existing lint violations previously found with the old tool with a custom ruleset. Of course, having a customized ruleset is always preferable to reduce noise and run times, both of which save on engineering time. Noise suppression capability by using methodology goals is demonstrated in Figure 19, but violations from the old flow were not completely covered in that specific goal alone. Again, figuring out the relevancy of every previously uncovered violation requires extensive support from the designers, and thus this goal is out of the scope of this research. This is something that could be focused on in software integration phase once a possible acquisition decision has been made and custom ruleset configuration is on-going.

Clarity of results was on-par with the old software when it comes to the error messages alone. However, with improved status flow and clearer error tags, the new workdlow is overall better in this category. The status flow allows the user to append a “Pending” violation status list in the main database instead of waiving it or keeping a separate list to follow the violation statuses. This is increasingly more useful once the DUT is scaled up and error messages increase in frequency. On top of that, the new lint flow offers graphic design metrics for displaying the DUT status briefly. This might be more suitable for displaying the DUT status for management personnel, who might not be all that interested in the technical details of each error. The offered graphic charts display an overall “Quality Score” percentage, which has been broken down to different bar charts, such as “Nomenclature Style” and “RTL Design style”. This is also an improvement over the current lint workflow. The actionable information density can be configured with the desired methodology and goal.

The default synchronization structures were also flagged by the new tool, so a way needed to be found to teach the intended functionality to the software. The old workflow uses a custom CDC RTL for these synchronizers, as a more elegant solution was needed. As mentioned before, the new CDC flow supports TCL-based custom synchronizers which proved to be effective, contained within the tool, and neat.

5.6 Tool integration

As in the old workflow, the new one also uses a shared GUI between its quality checking tools. This can make finding shared GUI elements faster if the user has already used another tool with the same GUI. This, however, is already a commonplace feature and is available in the old GUI. The same improved status flow was also shared between the tools in both the old and new workflows. Also, the compilation between these tools can be shared if the “-lint du” flag was set for the compiler of the new flow to enable some error checks. For the new flow, compilation was deliberately separated from the checking commands to save some time, whereas in the old flow the compilation is always completed in each quality check run. File list based project dependencies were originally made to support the old tool, but they worked well for the new tool with the modular workflow. Therefore, there were no integration issues the digital design flow the new tool.

5.7 User experience

It is problematic to objectively measure the user experience overall. However, addressing evaluated features and their extent is still covered with a remark. Workflow preference is indicated in each section. Overall, the ease of use is on a similar level in both tools and there are some similarities between them.

Ease of setup was simple in the new flow, and no serious issues were experienced in that part. The tool was able to notify whether the constraints were descriptive enough or not for it with the “Setup” goal. The tool was also able to provide suggestions to improve constraints. A report with suggested missing information was created after the “Setup” goal was run. Therefore, even though available constraints were unideal to begin with, the workflow to improve them was rather straightforward. Clock and reset grouping suggestions were also made, but some of the could not be used as is, after a specification review with the designer. However, this same functionality was also available in the old tool flow. The list of settings containing all the options affecting the current database was considered especially helpful for tracing whether setup changes took effect.

The status flow was perceived as a comparatively more useful feature in the new workflow. The customizable statuses with a “Pending” state were considered very helpful. This enables keeping track of the interesting issues right in the tool database itself instead of writing waivers or keeping another separate violation list with additional notes. This could be even more useful if the user contacts the designer about an issue and sends a file system path to the database, where the pending violations are already emphasized and ready to be analysed. In addition, the other GUI features are also usable for the designer instead of the violation message alone with this approach.

Setup portability between CDC and lint was effortless once the modular tool flow had already been created. At that point, setting up the lint tool took about an hour. According to the tool support personnel, RDC would also be straightforward to deploy at this point, should it be required. It is difficult to assess how portable the old workflow was to setup without participating in it but setting up another tool in an hour sets the standard quite high on its own. CDC took quite a bit more time as the modular flow needed to be created at that point, and custom synchronizers needed to be done.

Viewing results is straightforward once the database loading command is given in both the tools. Alternatively, reports are also generated in the user specified file system path for review. However, the violation handling is better in the new flow due to the improved status flow. Violations can be sorted by severity, rules, alphabetical order, or module in both the tools and filtering, and waiving of violations can be done effortlessly. The new tool differentiates itself the most with the lint status visualizations, in which bar graphs are compiled to reflect the DUT maturity and error categorization. The new flow offers the opportunity to include different features into the report. This rough but effective means of customization will help the user match the data reporting.

Viewing schematic is otherwise considered to be on par with the old flow, but the ability to see the clock domains right away by using the cursor is an improvement. Both the tools offer the ability to view source code from the schematic, but the new tool allows the users to choose whether to see the module itself or an instance based on it. Constraining can be done in both the tools, either by editing the constraint file with a preferred editor within the file system or by using a GUI. In both tools, GUI arranges the constraint into separate text fields with a specific information type, which may reduce the occurrence probability of a typographical error. Both the tools have support for incremental reporting and the report layout is logical in both instances. If the scope of changes is known, both tools allow viewing violations within a certain module to aid in that. Integration between CDC steps can be automated to a high degree on both tools, and they support either running more limited single goals, or all of them by extending the Makefile script. Dynamic CDC was out of scope for this research, as it was not in use.

The new tool suite support all required HDL languages: VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language), Verilog and SystemVerilog. New constraints used a different data format from the old flow, so a script was used to transform the existing constraints for use with the new tool. Incremental analysis support was successfully tested in the new lint tool. However, it did not support hierarchical flow, which was available in CDC. By default, the old flow supports a wide variety of different formats, whereas the new flow allows the user to customize the reports themselves.

5.8 Support

The new tool documentation was sufficient in for the user to be able to diagnose and fix the presented issues. Search functionality speeds up finding the correct issue and documentation is available in HTML and PDF formats. The current verification flow requires users to separately download documentation from the installation folder in the CAD server. More details about documentation were introduced in section 3.4.

Customer support was clearly taken seriously by the tool provider. Their support was consistently available to even visit the configuration site if necessary. The support was not hesitant to contact the tool development team if systematic issues were found or they could not answer the presented questions. Initially, the symbolic links used in the project file system were not supported as is. After contacting the provider with a demonstration of the issue, they replied in two hours after finding the fix and included support for the feature in the following software release, which was about 2 weeks later. Later this fix was verified to have taken effect. This case demonstrated both excellent customer support and issue responsiveness and contacting the provider with an issue required minimal overhead.

6. CONCLUSIONS

This research encompassed mapping out the most important criteria when choosing a lint or CDC software for the evaluation to replace the old verification flow. The benefits and drawbacks of using the new lint & CDC workflow for this purpose were measured. Background information was researched from literature sources and an inquiry was made to gain insight to the theoretical issues behind using these tools and which features are considered important. The history of EDA was investigated to justify using EDA tools nowadays. Even though these tools offer an important perspective into verification, there exist a lot of other tools and additional features that the tool vendors offer to complement the EDA tool belts of design organizations.

In this research, it was concluded that using EDA tools in ASIC development enables cheaper error correction earlier in the design flow, which makes EDA tools important in today's competitive ASIC market. By automatization and simplification of design flow, more complex errors are caught, and less time is spent on solving human errors. This set the financial motivation for this research. Typical causes behind lint and CDC verification were also covered.

Tool requirements were investigated, and the provided alternative tool suite was integrated into the company's ASIC design flow modularly by using shared file lists, parameters and Makefiles. Ensuring that specification is accurate and detailed, while having plentiful computing resources and a well-integrated design flow kept the results consistent, reliable, and representative of the DUT status.

The following features were inspected additionally with the alternative verification flow: Customizable CDC synchronizers, CDC hierarchical flow and lint incremental flow. Also, the tools' ability to conform to the provided functionality chart containing proposed functionality was assessed. This is imperative in gaining optimal benefit from the generally pricey EDA tools. By recording the software run-times and resource utilization on the servers, a presentable impression about the performance against the old static verification flow used by the company was formed. The results show that the new workflow can reserve less than a third of the storage space and system memory across the tests when compared to the old flow. It also completes the assigned task in a similar time, or in the best case remarkably faster. The overall resource sparsity and fast run times both contribute to superior resource efficiency in the new verification flow.

This research finds that by using the new tool flow, the clock-domain crossing verification flow is accelerated by over threefold and the lint verification flow by about a quarter, while the inspections detect the same number of issues in the code, possibly even more. The new tool flow requires under a third of the disk space and system memory compared to the old flow. In addition, it is observed that the new software is overall more pleasant to use: The software is perceived to be somewhat more challenging to learn, but in return it provides more information with which to solve the underlying issues in the code.

Qualifying and quantifying the reported items and assessing performance on the computation servers helped us form a representable impression of the new software suite while addressing the company's needs. The new software was able to discover more schemes in both lint and CDC, making for a great starting point for creating a custom ruleset to further improve run times and result readability. To control the reporting of false errors in the all-inclusive design results, different methodology goals can be used to optimize designer time. For the most part, the new CDC flow demonstrated feature parity, but the software differentiates itself with better GUI feature contextuality, superior performance and resource efficiency. The company should consider adding the tool to complement its verification tool belt due to the performance, verification thoroughness and more moderate use of computation server resources.

More research could be done about this subject by using a different software deployment over a local system. Additional compliance tests could be done while using a software from a different vendor or by extending to RDC (Reset-Domain Crossing) and formal testing, or when deploying CDC test plans and CI infrastructure. Complementing result data could be gathered while using a different tool for the same purpose.

Their effects on design flow could be analysed separately. Additionally, more specific constraints could be generated and applied to formal verification features, such as dynamic CDC or UPF. The scope of this research, however, was to form and measure improvements to the old verification workflow with the deployment of an alternative lint & CDC. Therefore, these more advanced tools outside the old flow were out-of-scope for this research.

It would also be interesting to observe software performance and GUI responsiveness in a local system with a more controlled environment to remove variables from the performance testing, such as file access latency over the Internet and dynamic server load. Exploring different hardware configurations with varying performance “bottlenecks” would also be interesting to observe. In addition, a more comprehensive software comparison for a signoff quality DUT would also be interesting for a tape-off ready chip.

This would be especially helpful in in-depth noise analysis and ruleset creation. Recreating the standard ruleset as accurately as possible could further improve the accuracy and run times of the new software suite, which could be interesting to investigate as well. Other quality checking tools could also be investigated to get a better idea of the EDA market offerings to complement these results. This would also help form a more comprehensive insight about the best EDA tool for the company’s needs.

REFERENCES

[1] S. Kostin et al, "VLSI-SoC: System-on-Chip in the Nanoscale Era – Design, Verification and Reliability," in *24th IFIP WG 10.5/IEEE International Conference on Very Large Scale Integration, Revised Selected Papers. Vol. 508*, Tallinn, Estonia, 2016.

[2] "Chips in a Crisis," *Nature Electronics,* vol. 5, no. 4, p. 317, 2021.

[3] Y. Jiang, J. Shu and M. Song, "Coping with shortages caused by disruptive events in automobile supply chains," *Naval research logistics,* vol. 1, no. 69, p. 21–35, 2022.

[4] A. B. Mehta, ASIC/SoC Functional Design Verification: A Comprehensive Guide to Technologies and Methodologies, Springer International Publishing AG, 2006.

[5] K. Wakabayashi and T. Okamoto, "C-Based SoC Design Flow and EDA Tools: An ASIC and System Vendor Perspective," *IEEE Transactions on Computer-aided Design of Integrated Circuits And Systems,* vol. 19, no. 12, p. 1507–1522, 2000.

[6] D. Gajski, S. Abdi, A. Gerstlauer and G. Schirner, Embedded System Design Modeling, Synthesis and Verification, 1 ed., New York: Springer US, 2009.

[7] V. Bertacco, Scalable hardware verification with symbolic simulation, New York: Springer, 2006.

[8] V. Kamakoti et al, "Automatic Constraint Based Test Generation for Behavioral HDL Models: Design verification and validation," *IEEE transactions on VLSI systems,* vol. 4, no. 16, p. 408–421, 2008.

[9] J. Bergeron, Writing Testbenches: Functional Verification of HDL Models, New York: Springer, 2000.

[10] W. A. Ibrahim, "Novel EDA Tool for VLSI Test Vectors Management," *Journal of electronic testing,* vol. 5, no. 23, p. 421–34, 2007.

[11] A. Firdous and S. M. Kusuma, "Speeding up of Design Convergence using Spyglass," in *Global Conference for Advancement in Technology (GCAT), IEEE*, Bangalore, India, 2019.

[12] K. Mohamed, IP Cores Design from Spesifications to Production: Modeling, Verification, Optimization, and Protection, Springer International Publishing AG, 2015.

[13] S. Shinde, H. Guhilot, P. Gaikwad and R. Kamat, Harnessing VLSI System Design with EDA Tools, 1 ed., Dordrecht, Netherlands: Springer Netherlands, 2012.

[14] A. Kumar Das, "A Linting tool without a compiler in it," in *2021 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, Bangalore, India, 2021.

[15] H. Hellmich, A. Erdogan and T. Arslan, "Re-usable low power DSP IP embedded in an ARM based SoC architecture," *IEE Colloquium (Digest),* vol. 7, no. 82, p. 43–47, 2000.

[16] M. Jassi, Y. Hu, D. Mueller-Gritschneder and U. Schlichtman, "Graph-Grammar-Based IP-Integration (GRIP)-An EDA Tool for Software-Defined SoCs," *ACM transactions on design automation of electronic systems,* vol. 23, no. 3, p. 1–26, 2018.

[17] S. Vasudevan, Effective Functional Verification Principles and Processes, New York: Springer US, 2006.

[18] D. MacMillen, R. Camposano, D. Hill and T. Williams, "An industrial view of electronic design automation," *IEEE transactions on computer-aided design of integrated circuits and systems,* vol. 19, no. 12, p. 1428–1448, 2000.

[19] ICT Monitor Worldwide, "RTL sign-off apps boost design verification," 2017.

[20] Y. Q. Aguiar de et al, "Permanent and single event transient faults reliability evaluation EDA tool," *Microelectronics and reliability,* vol. 64, p. 63–67, 2016.

[21] S. Churiwala and S. Garg, Principles of VLSI RTL Design: A Practical Guide, 1 ed., New York: Springer Science and Business Media, 2011.

[22] S. Latif, "Continuous Integration for Fast SoC Algorithm Development," Tampere University, Tampere, 2019.

[23] J. R. Pérez, Human error reduction in manufacturing.:, Milwaukee, Wisconsin, USA: ASQ Quality Press, 2018.

[24] J. Bruce, W. Spencer and D. Twang, Memory Systems: Cache, DRAM, Disk, Elsevier Science & Technology, 2007.

[25] J. Andrews, Co-verification of hardware and software for ARM SoC design, Burlington, MA: Elsevier Newnes, 2005.

[26] E. Bin et al, "Hardware and Software, Verification and Testing", Second International Haifa Verification Conference, HVC 2006, Haifa, Israel, October 23–26, 2006, Revised Selected Papers., 1 ed., Berlin, Germany: Springer Berlin Heidelberg, 2007.

[27] S. Friedrichs, M. Fugger and C. Lenzen, "Metastability-Containing Circuits," *IEEE transactions on computers,* vol. 8, no. 67, p. 1167–1183, 2018.

[28] T. Chaney and C. Molnar, "Anomalous Behavior of Synchronizer and Arbiter Circuits.," *IEEE transactions on computers,* vol. 4, no. 22, p. 421–422, 1973.

[29] Z. Al-Tarawneh;, "On Metastability Resilience of Multi-Gate MOSFETs," *Jordan Journal of Electrical Engineering,* vol. 1, no. 2, p. 83–92, 2015.

[30] J. Liu et al., "Clock domain crossing aware sequential clock gating," *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition. EDA Consortium,* p. 1–6, 2015.

[31] Y. Feng, Z. Zhou, D. Tong and X. Cheng, "Clock domain crossing fault model and coverage metric for validation of SoC design," in *Design, Automation and Test in Europe Conference*, Nice, France, 2007.

[32] Atrenta Inc: S. Sarwary, S. Verma, "Critical clock-domain crossing bugs," *EDN,* vol. 7, no. 53, p. 55, 2008.

[33] G. Vrinda, K. Mohammad and J. Mohandas, "Towards Improving Clock Domain Crossing Verification for SoCs," *Journal of electrical and electronics engineering,* vol. 2, no. 14, p. 19–24, 2021.

[34] E. Clarke et al, Model Checking, Cambridhe, Massachusetts: The MIT Press, 1999.

[35] C. Pixley, J. Yuan and A. Aziz, Constraint-Based Verification, New York: Springer, US, 2006.

[36] ActiveState, "ActiveState front page," [Online]. Available: https://www.activestate.com/. [Accessed 5 May 2022].

[37] M. Hassenzahl and N. Tractinsky, "User experience - a research agenda," *Behaviour & information technology,* vol. 2, no. 25, p. 91–97, 2006.

[38] A. Kahng, J. Lienig, I. L. Markov and J. Hu, VLSI Physical Design: From Graph Partitioning to Timing Closure, 1 ed., Dordrecht, Netherlands: Springer Netherlands, 2011.

[39] Intel, "Intel 's stance on Side Channel Vulnerabilities," [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/mds.html. [Accessed 15th March 2022].

[40] M. Kasim, G. Vrinda and M. Jebin, "Methodology for Detecting Glitch on Clock, Reset and CDC path," in *Proceedings of the Fifth International Conference on Communication and Electronics Systems*, 2020.

[41] Amazon, "Current Price of High-Speed Storage," Amazon, [Online]. Available: https://www.amazon.com/s?k=nvme+ssd&crid=14R5GUOXBH50P&sprefix=nvme+ssd%2Caps%2C175&ref=nb_sb_noss_1. [Accessed 21st April 2022].

[42] V. Gupta, M. Kasim and M. Jebin, "Towards Improving Clock Domain Crossing Verification for SoCs," *Journal of electrical and electronics engineering,* vol. 2, no. 14, p. 19–24, 2021.

[43] "Centos wiki," [Online]. Available: https://wiki.centos.org/FAQ/General#What_is_the_support_.27.27end_of_life.27.27_for_each_CentOS_release.3F. [Accessed 15 3 2022].

[44] M. Massoumi and A. Sagahyroon, "ASIC verification: Integrating formal verification with HDL-based courses," *Computer applications in engineering education,* vol. 2, no. 18, p. 269–276, 2010.

APPENDIX A: STRUCTURED FEATURE CHART

		weight
Methodology & Flow	Comprehensiveness of methodology	5
	Methodology & flow documentation	4
	Integration with current flow	3

Quality of Results	Accuracy of results (minimal false violations)	4
	Static CDC	4
	Dynamic CDC	4
	Static RDC	3
	RDC Protocol	3
	Lint	4
	Clarity of results (actionable information density)	4
	Static CDC	4
	Dynamic CDC	4
	Static RDC	3
	RDC Protocol	3
	Lint	4
	CDC path analysis	5
	Complex synchronization schemes	4
	Custom synchronization support	3
	Detailed scheme reporting	4
	Found all issues found by competitor	5
	Found new issues not found by competitor	5
	Constraint verification	5
	Protocol verification accuracy	5
	Hierarchical abstraction accuracy	4
	Multimodal analysis accuracy	4
	UPF DVFS analysis quality of results	4
	Information density generated by default rule sets	4

Performance	Performance (run time)	3
	Memory footprint	2
	Time to analysis complete (including post-run application of status flow)	3
User Experience	Ease of use	5
	Ease of setup	3
	Accurate clock recognition	5
	Accurate reset recognition	5
	Constraint review & debug	3
	Reset grouping	3
	Clock grouping	3
	Usability of status flow	4
	Dynamic simulation integration (CDC-FX)	3
	Setup portability between Lint, CDC and RDC	4
	Ease of debug	5
	Viewing and filtering of results	4
	Schematic viewing	4
	Schematic links to RTL source code	5
	Ease of adding constraints	3
	Comparison utility for design changes	4
	Integration between CDC steps (i.e. structural, protocol, etc.)	4
	Waveform viewing of dynamic results	5
	Coverage viewing of dynamic results	4
	Language support	5
	Constraint support	4
	Incremental/Differential Analysis	4
	Setup & Maintenance of hierarchical flow	4
	Report format diversity	4
Support	Documentation	5
	Customer Support	5
	Issue responsiveness	5

Tool Integration	Testplan coverage generation	4
	Built-in FPGA library support	1
	Same UI experience across related tools	3
	Common status flow across related tools	3
	Common compiler across related tools	3