Reconfigurable Scan Networks: Formal Verification, Access Optimization, and Protection

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Rafał Baranowski

aus Gliwice / Polen

Hauptberichter:	Prof. Dr. rer. nat. H.-J. Wunderlich
Mitberichter:	Prof. Dr.-Ing. W. Kunz
Tag der mündlichen Prüfung:	7. Januar 2014

Institut für Technische Informatik der Universität Stuttgart

2014

Dedicated to my wife Ela and my parents.

Acknowledgements

This work would not have been possible without the professional support of my supervisor and colleagues. I would like to thank Prof. Hans-Joachim Wunderlich for his advice, ideas, and feedback that greatly contributed to this thesis. I am very grateful to Michael Kochte who involved me in this project, was always willing to share his experience, and provided me with continuous support and encouragement. Many thanks also go to Christian Zöllin for sharing his enthusiasm and expertise in IEEE Std. P1687, and to Alejandro Cook for proofreading this manuscript.

I was able to find the time to pursue this project thanks to competent colleagues with whom I shared the teaching responsibilities at the University of Stuttgart. It was a great pleasure and fun to work with Melanie Elm, Laura Rodríguez Gómez, Michael Kochte, Stefan Holst, Christian Zöllin, and Alejandro Cook. Many thanks also go to Mirjam Breitling, Helmut Häfner, Lothar Hellmeier, and Wolfgang Moser for their excellent administrative and technical assistance.

I thank my wife, Elżbieta Strachocińska, for her caring support and for joining me in Stuttgart despite expected professional difficulties. I also thank my parents and my mother-in-law for their continuous encouragement and faith in the success of this undertaking. Thanks also go to Jakub Cichoszewski for his help during my early days in Stuttgart.

Finally, I would like to thank all other colleagues and students with whom I had the pleasure to cooperate with on various projects during my stay at the University of Stuttgart, especially: Andrew Mofid Boktor, Claus Braun, Francesco Cervellera, Atefe Dalirsani, Nadereh Hatami, Michael Imhof, Anusha Kakarala, Chang Liu, Abdullah Mumtaz, Eric Schneider, Marcel Schaal, Alexander Schöll, Dominik Ull, and Marcus Wagner.

Stuttgart, November 2013

Rafał Baranowski

Contents

Abbreviations

ATE	Automated Test Equipment
ATPG	Automatic Test Pattern Generator
BDD	Binary Decision Diagram
BMC	Bounded Model Checking
CAM	CSU-Accurate Model
CSU	Capture-Shift-Update
CTL	Computational Tree Logic
DfT	Design-for-Test
EDA	Electronic Design Automation
FSM	Finite State Machine
IJTAG	Internal JTAG
IP	Intellectual Property
JTAG	Joint Test Action Group
LTL	Linear Temporal Logic
MC	Model Checking
PBSAT	Pseudo-Boolean SATisfiability
QBF	Quantified Boolean Formula
RSN	Reconfigurable Scan Network
RTL	Register-Transfer Level
SAT	Boolean SATisfiability
SoC	System-on-a-Chip
TAP	Test Access Port
VLSI	Very Large Scale Integration

Notation

- Sets:
  $\varnothing$ – empty set,
  $\mathbb{B}$ – set of Boolean values,
  $\mathbb{N}^+$ – set of positive natural numbers,
  $\mathbb{N}$ – set of natural numbers with 0, $\mathbb{N} \equiv \mathbb{N}^+ \cup \{0\}$,
  $\mathbb{Z}$ – set of integer numbers,
  $|\cdot|$ – cardinality of a set,
  $\mathcal{P}(\cdot)$ – power set.
- Set operators:
  $\cup$ – union,
  $\cap$ – intersection,
  $\setminus$ – difference,
  $\equiv$ – equivalence.
- Boolean operators:
  $\neg$ – negation,
  $\wedge$ – conjunction,
  $\vee$ – disjunction,
  $\oplus$ – exclusive disjunction,
  $\Rightarrow$ – implication,
  $\Leftrightarrow$ – equivalence.
- Boolean values, *true* and *false*, are denoted by 0 and 1, respectively. ($\mathbb{B} \equiv \{0, 1\}$.)
- A Boolean function is a mapping $f : \mathbb{B}^n \to \mathbb{B}$, where $n \in \mathbb{N}^+$.
- A vector of Boolean variables or Boolean functions is accented with an arrow, e.g. $\vec{a} = [a_0, a_1, a_2, \ldots, a_n]$.
- A *multi-output* Boolean function $\vec{f} : \mathbb{B}^n \to \mathbb{B}^m$, where $n, m \in \mathbb{N}^+$, is a vector of

Boolean functions $\vec{f} = [f_0, f_1, f_2, \ldots, f_m]$, such that $f_i : \mathbb{B}^n \rightarrow \mathbb{B}$ for each $i \in \mathbb{N}$, $i \leq m$.

- Characteristic functions of sets are denoted by the symbol Ω, e.g., $\Omega_Y : X \rightarrow \mathbb{B}$ denotes a characteristic function of set Y such that $Y \subseteq X$.

Summary

To facilitate smooth VLSI development and improve chip dependability, VLSI designs incorporate instrumentation for post-silicon validation and debug, volume test and diagnosis, as well as in-field system maintenance. Examples of on-chip instruments include embedded logic analyzers, trace buffers, test and debug controllers, assertion checkers, and physical sensors, to name just a few. Since the amount of embedded instrumentation in system-on-a-chip designs increases at an exponential rate, scalable mechanisms for instrument access become indispensable.

Reconfigurable scan architectures emerge as a suitable mechanism for access to on-chip instruments. Such structures integrate embedded instrumentation into a common scan network together with configuration registers that determine how data are transported through the network. For test purposes, the design of regular reconfigurable scan networks is covered by IEEE Std. 1149.1-2013 (Joint Test Action Group, JTAG) and IEEE Std. 1500 (Standard for Embedded Core Test, SECT). For general-purpose instrumentation, the ongoing standardization effort IEEE P1687 (Internal JTAG, IJTAG) allows user-defined scan architectures with arbitrary access control.

The flexibility of reconfigurable scan networks poses a serious challenge: The deep sequential behavior, limited serial interface, and complex access dependencies are beyond the capabilities of state-of-the-art verification methods. This thesis contributes a novel modeling method for formal verification of reconfigurable scan architectures. The proposed model is based on a temporal abstraction which is both *sound* and *complete* for a wide array of scan networks. Experimental results show that this abstraction improves the scalability of model checking algorithms tremendously.

The access to instruments in complex reconfigurable scan networks requires specialized algorithms for pattern generation. This problem is addressed with formal techniques that leverage the temporal abstraction to generate valid access patterns with low access time. This work presents the first method applicable to *pattern retargeting* and *access merging* in complex reconfigurable architectures compliant with IEEE Std. P1687.

Embedded instrumentation is an integral system component that remains functional throughout the lifetime of a chip. To prevent harmful activities, such as tampering with safety-critical systems, and reduce the risk of intellectual property infringement, the access to embedded instrumentation requires protection. This thesis provides a novel, scalable protection for general reconfigurable scan networks. The proposed method allows fine-grained control over the access to individual instruments at low hardware cost and without the need to redesign the scan architecture.

Zusammenfassung

Um eine reibungslose Chipentwicklung zu ermöglichen und die Verlässlichkeit von VLSI-Schaltkreisen zu steigern, werden Chipentwürfe um spezielle Instrumente für Post-Silicon-Validierung und Debug, Produktionstest und Diagnose, sowie für Systembetrieb und Instandhaltung erweitert. Diese Chip-interne Infrastruktur umfasst unter anderem eingebettete Logik-Analyser, Beobachtungsspeicher (*trace buffers*), Test- und Debugsteuereinheiten, Assertion-Checkers und Sensoren. Da die Menge der Instrumente in modernen Chipentwürfen exponentiell steigt, sind skalierbare Zugriffsmechanismen für diese Infrastruktur unerlässlich.

Rekonfigurierbare Scan-Netze bilden einen geeigneten Zugriffsmechanismus für die On-Chip-Infrastruktur. Sie integrieren die eingebetteten Instrumente und Konfigurationsregister in ein gemeinsames Netz, in dem der Datenfluss von den Konfigurationsregistern bestimmt wird. Für Testzwecke wird der Entwurf von regulären rekonfigurierbaren Scan-Netzen im IEEE Std. 1149.1-2013 (Joint Test Action Group, JTAG) sowie IEEE Std. 1500 (Standard for Embedded Core Test, SECT) festgelegt. Im Hinblick auf allgemeine Instrumentalisierung, erlaubt die laufende Normierung IEEE P1687 (Internal JTAG, IJTAG) benutzerdefinierte Scan-Architekturen mit beliebiger Zugriffsansteuerung.

Die Flexibilität von rekonfigurierbaren Scan-Netzen stellt eine große Herausforderung dar: Die erhebliche sequenzielle Tiefe, die begrenzte serielle Schnittstelle und die komplexen sequenziellen und kombinatorischen Abhängigkeiten solcher Strukturen übersteigen die Leistungsfähigkeit heutiger Algorithmen zur formalen Hardwareverifikation. Diese Arbeit trägt eine neue Modellierungsmethode zur Lösung des Problems bei. Die Modellierung basiert auf einer temporalen Abstraktion, die für ein breites Spektrum an Scan-Netzen sowohl korrekt (*sound*) als auch vollständig (*complete*) ist. Die experimentellen Ergebnisse bestätigen, dass die Skalierbarkeit von Model-Checking-Verfahren durch diese Abstraktion drastisch gesteigert wird.

Effizienter Zugriff auf rekonfigurierbare Scan-Netze fordert spezielle Algorithmen zur Zugriffsmustergenerierung. Dieses Problem wird durch einen formalen Ansatz gelöst, der mittels der temporalen Abstraktion gültige Zugriffsmuster mit reduzierten Zugriffszeiten generiert. Diese Arbeit präsentiert erstmalig eine Methode, die sich zur automatisierten Zugriffsmustergenerierung in komplexen rekonfigurierbaren Scan-Netzen nach IEEE P1687 (*pattern retargeting* und *access merging*) eignet.

Die On-Chip-Instrumente sind wesentliche Systemkomponenten, welche die ganze Systemlebensdauer hindurch funktionsfähig bleiben. Der Zugriff auf eingebettete Instrumente muss z. B. zum Schutz geistigen Eigentums und zur Absicherung gegen Sabotage beschränkt werden. Diese Arbeit liefert eine kostengünstige Zugriffssicherung für rekonfigurierbare Scan-Netze. Sie erlaubt eine detaillierte Kontrolle von Zugriffen auf einzelne Instrumente, ohne dass der Netzentwurf angepasst werden muss.

1. Introduction

Since over 40 years, the complexity of integrated circuits has been increasing at an exponential rate [ITRS12], fulfilling the prophecy of Gordon E. Moore [Moore65, Moore75]. This unprecedented pace of development has given rise to a wide range of new applications and markets, making electronic devices ubiquitous in nearly all branches of industry and in everyday life. To facilitate this growth, design methodologies and verification techniques need to be constantly revised to deal with a plethora of dependability issues that stem from the high integration density and system complexity.

As the complexity of VLSI designs grows, it becomes extremely challenging to verify and validate the design so as to eliminate or reduce the number of design errors that reach the silicon [Kropf99]. Today's System-on-a-Chip (SoC) projects need to allocate as much as 75% of human resources to the verification process [ITRS12]. Due to the extreme scaling of transistor sizes, VLSI chips become increasingly prone to manufacturing defects, process variations, and adverse effects that manifest themselves during in-field operation, such as aging mechanisms and soft errors [Borkar05, Baumann05]. VLSI designs with reliability constraints must therefore deal with the decay of silicon reliability to guarantee that the specification is met throughout the lifetime of a chip [Borkar05].

To facilitate smooth VLSI development and improve product dependability, VLSI designs incorporate on-chip instrumentation that makes the process of production ramp-up more tractable and facilitates in-field system maintenance. This embedded instrumentation includes, for instance, debug structures for post-silicon validation, as well as components that enable on-line system monitoring, reconfiguration, diagnosis, and repair [Abramovici08, Stollon11]. Scan networks, traditionally employed as a test access mechanism and recently extended with configurability features, emerge as a scalable and cost-effective access mechanism for such instrumentation [Rearick05, Stollon11].

As any other design feature, the scan networks are themselves prone to design errors which may compromise system reliability, security, or availability. Existing tools for formal verification are not robust enough to deal with the high complexity of advanced scan architectures [Baranowski12]. Moreover, the improved accessibility of on-chip instrumentation contradicts security and safety requirements for chip internals [Tehranipoor11, Baranowski13c]. This calls for efficient techniques for security improvement, as well as robust verification techniques to prove relevant properties of advanced scan networks, including functional correctness, safety and security.

This chapter reviews the purpose of on-chip instrumentation in the context of VLSI product life-cycle. Then, a brief introduction to reconfigurable scan networks is given, followed with a discussion of verification challenges and security issues. The chapter is concluded with an overview of this thesis.

1.1. VLSI Circuit Instrumentation

The life-cycle of VLSI circuits comprises four stages [Wang10]: *Circuit design* (1) involves design specification, implementation, verification, and pre-silicon validation. *Production ramp-up* (2) deals with correcting design errors that were overlooked in pre-silicon validation and targets yield improvement. It starts with the production of the first silicon (initial *tape-out*) and may require several design revisions and silicon *re-spins*. After the design is sufficiently validated and yield is acceptable, *volume production* (3) begins. The defect-free chips that are sold and operate *in the field* are subject to *maintenance* (4). On-chip instrumentation is used throughout the lifetime of a chip, and is especially important in production ramp-up, as explained below.

Post-Silicon Validation and Debug

As soon as the first silicon is delivered, the prototype chips are subject to post-silicon validation and debug. The chips are validated at-speed in scenarios that were too hard or impossible to tackle in pre-silicon validation due to uncertainties and high simulation cost. Very often, post-silicon validation identifies corner-case problems which were missed in pre-silicon verification due to low activation probability, inherent design indeterminism (multiple clock domains, asynchronous communication), signal integrity issues (cross-talk, power-droop, noise), process variations, or thermal stress [Abramovici08].

On average, just one third of VLSI designs are fully functional in the first silicon, and almost one third requires more than three re-spins [Foster11]. If the first silicon does not meet its specification, the root cause must be accurately diagnosed to facilitate rapid design revision. The diagnosis of complex silicon devices in advanced technology nodes is a hard task: Even if a chip fails in validation experiments (an error is detected), it may produce a “no trouble found” outcome when tested on an Automated Test Equipment (ATE) due to different operating conditions [Abramovici08]. The aim of post-silicon debug is to make the observed error reproducible and quickly pinpoint its root-cause, be it a logical design bug or a more sophisticated signal integrity issue. To this end, the chip is equipped with a range of on-chip instruments that improve signal observability and controllability. These include *scan chains* to control and capture the logic state of the system, observation scan chains for non-intrusive at-speed state dump, embedded logic analyzers and trace buffers to track the events at internal system nodes, and various controllers for signal masking and clock control [Wang10]. Sophisticated techniques make use of reconfigurable logic that is dynamically programmed to suit various debug tasks such as assertion checking, detection of events, identification of transactions, or even circuit repair [Abramovici08]. Advanced microprocessor architectures are also equipped with specialized instruction recorders to restore the microarchitectural state [Mitra10].

Volume Production and Test

After all detected design errors are fixed and the production yield is satisfactory, volume production begins. Each produced chip undergoes a series of ATE-based tests to screen out defective devices: on the wafer (*wafer test*), after packaging (*package test*) and at the customer’s site (*acceptance tests*) [Wang06]. The goal of the volume test is to guarantee high product quality and prevent that defective chips be shipped to the customer. To identify systematic defects and facilitate yield learning and yield ramp, defective chips are subject to *volume diagnosis* [Holst09].

To support volume test and diagnosis, VLSI circuits are equipped with Design-for-Test (DfT) instruments that improve design testability and reduce test cost [Bushnell00]. Scan chains are used to load test patterns to the sequential elements of a circuit and capture the test responses [Eichelberger77]. Individual components of a circuit are enclosed in *test wrappers* that facilitate hierarchical testing of core-based designs [Zorian98]. Structures for test pattern decompression and response compaction reduce the test data volume and test time [Wang10]. For the test of analog and mixed-signal components, multiplexed analog buses are used [Bushnell00].

Maintenance

Chips that pass the production tests are delivered to the customer. During operation in the field, the devices are exposed to various stress factors, including high ambient temperature, mechanical stress, electromagnetic interference, particle strikes, and aging processes. Due to shrinking feature sizes, the devices become increasingly susceptible to such stress factors, which may result in their temporary or permanent malfunction [Borkar05, Baumann05].

To guarantee reliability throughout the lifetime of a chip, various methods for in-field monitoring, error correction, test, diagnosis and repair are adopted. Such techniques are often supported with on-chip instrumentation. For instance, on-chip assertion checkers are used to detect circuit malfunction during regular system operation [Abramovici08]. Aging-induced circuit degradation is monitored with specialized sensors such as silicon odometers [Keane10], workload monitors [Baranowski13a], or stability checkers [Agarwal07] that are distributed over the chip. In-field fault detection is implemented with Built-in Self-Test (BIST) controllers that apply random or deterministic test patterns to logic and memory components and evaluate the test response [Wang06]. Defective memory cells are repaired in the field using so called infrastructure IP cores [Zorian02]. On-chip infrastructure must also provide for system maintenance, including in-field reprogramming and reconfiguration [Bonnett99], as well as in-field error detection and fault management [Jutman11].

1.2. Examples of On-Chip Instruments

Embedded instrumentation is commonly used for the test and characterization of high speed system components that are beyond the capabilities of Automated Test Equipment (ATE). On-chip instruments circumvent the need for specialized and costly labo-

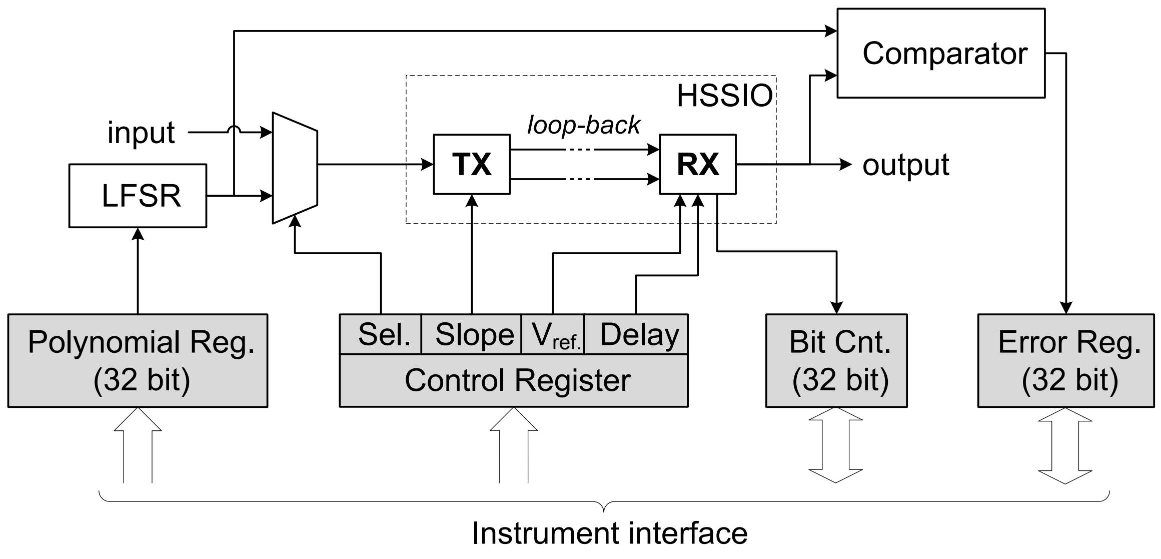


Figure 1.1.: Embedded instrumentation for the test and characterization of High-Speed Serial I/O (after [Rearick06])

ratory equipment and facilitate in-field test and calibration.

For example, an instrument for the test and characterization of High-Speed Serial I/O (HSSIO) is given in Figure 1.1 after [Rearick06]. In the test/characterization mode, the output of the HSSIO transmitter (TX) is fed to the receiver (RX). A Linear Feedback Shift Register (LFSR) generates a pseudo-random sequence of bits which is fed to the transmitter and compared at the output of the receiver. The configuration and status of the instrument is stored in dedicated registers that form the instrument's interface. The configuration includes the settings for the loop-back mode, channel parameters such as output slope of the transmitter, reference voltage and sample delay of the receiver, and the primitive polynomial of the LFSR. The status includes the number of transmitted bits and the number of detected errors.

Debug instrumentation has become an unquestionable necessity in modern microprocessor designs [Stollon11]. They facilitate event detection, trace storage, configurable breakpoints, and run-time access to embedded memories, among others. A simplified version of the generic on-chip debug system presented in [Stollon11] is illustrated in Figure 1.2: The debug instrumentation includes a set of registers that constitute the debug interface. These registers store, for instance, the current debug status, provide the current value of the instruction pointer, configure triggers and breakpoints, and facilitate bidirectional access to the on-chip memories and caches.

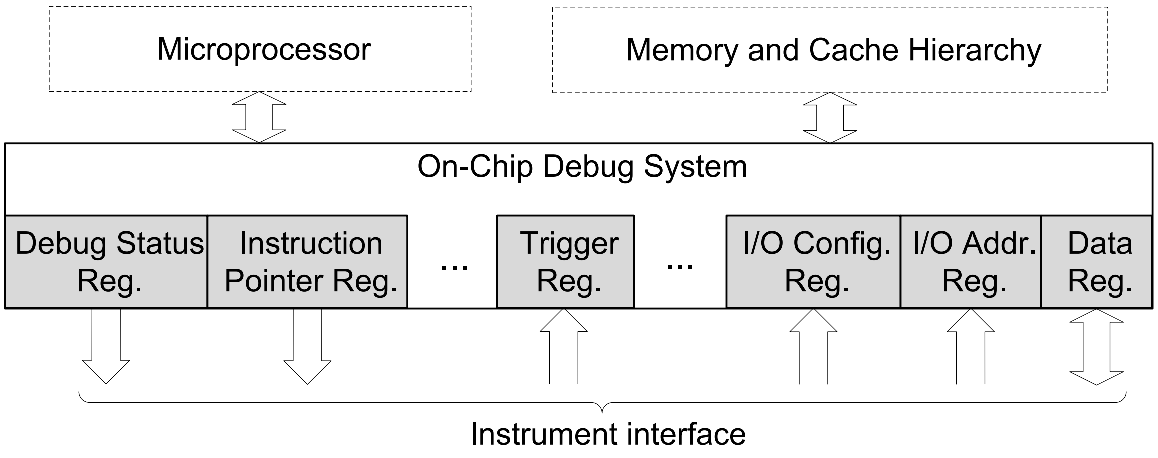


Figure 1.2.: Embedded instrumentation for microprocessor debug (after [Stollon11])

1.3. Cost-Effective Access to Embedded Instrumentation

The most widely used interface for accessing on-chip instrumentation is the 4-wire Test Access Port (TAP) defined by IEEE Std. 1149.1 (Joint Test Action Group, JTAG). Formally, JTAG targets the test of interconnects on printed circuit assemblies [JTA01]. Over the years, however, the TAP interface has become a *de facto* standard for efficient, low-cost access to on-chip instrumentation widely used for structural logic test, system monitoring, reprogramming, and debug [Rearick05, Ley09, Stollon11]. Recently, an extension to JTAG in form of IEEE Std. 1149.7 has been proposed to reduce the TAP pin count to two wires, improve its bandwidth and reduce power consumption [Ley09].

The IEEE 1149.1 circuitry consists of a TAP interface, a TAP controller, an Instruction Register `IR`, a bypass register $\mathtt{DR}_0$, a set of optional user Data Registers $\mathtt{DR}_1 \ldots \mathtt{DR}_n$, and a scan multiplexer, as shown in Figure 1.3. The TAP comprises four mandatory signals: Test Data Input (TDI), Test Data Output (TDO), Test Mode Select (TMS), and Test Clock (TCK). Both `IR` and `DR`s are shift registers, the access to which is controlled by the TMS signal and the TAP controller. The content of `IR` is called *instruction*: It determines the addressing of the scan multiplexer and chooses one of the `DR`s. The TAP controller implements a finite state machine which performs a Capture-Shift-Update (CSU) operation on a chosen shift register: During the *capture* phase, the state of the chosen shift register is loaded in parallel, e.g. from an attached instrument. During the *shift* phase, the input data from TDI (*scan data*) are shifted through the chosen shift register down to the TDO. During the *update* phase, the newly shifted scan data

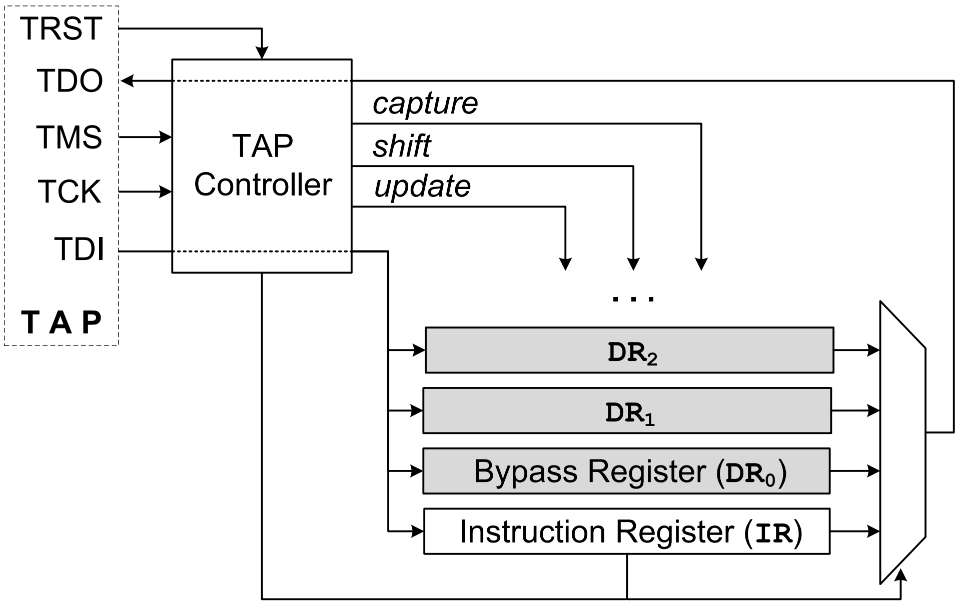


Figure 1.3.: IEEE 1149.1/JTAG circuitry (simplified)

are stored in the shadow registers (if any) of the chosen shift register.

The data registers (DR) in Figure 1.3 are either shift registers composed of *boundary scan cells* that isolate the system logic from physical pins to facilitate interconnect test, or user-defined shift registers of arbitrary purpose [JTA01]. The support for user-defined data registers has been the enabling factor for the wide adoption of JTAG TAP for structural logic test and access to on-chip instrumentation.

In structural logic test, the sequential elements of a circuit (flip-flops or latches) are replaced with *scan cells* to improve testability [Eichelberger77]. In regular system operation (system mode), the scan cells operate as regular sequential elements. In test mode, the scan cells form a shift register called *scan chain* which is used to supply test patterns to the sequential elements and capture the test response. Multiple scan chains can be interfaced with the JTAG infrastructure as data registers (DRs).

Due to the support for user data registers, the JTAG TAP is often exploited for cost-effective access to on-chip instrumentation [Rearick05]. To this end, the interface of an instrument (cf. examples in Section 1.2, p. 12) is equipped with a shift register for bidirectional, scan-based communication, as shown in Figure 1.4. This shift register is connected to the JTAG circuitry as a DR.

Each DR in the JTAG circuitry is activated by a unique instruction that must be written

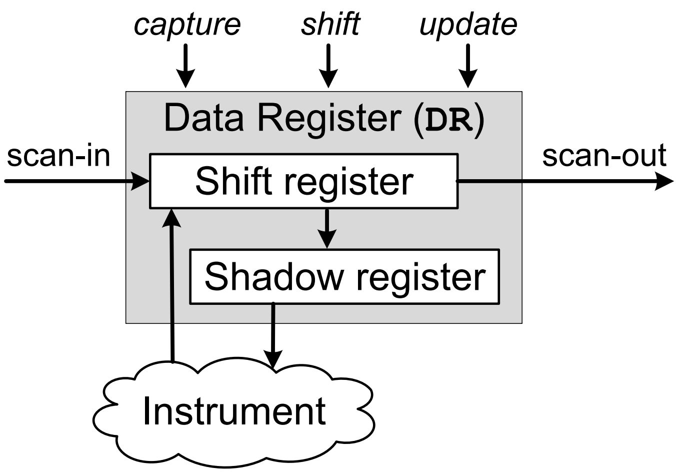


Figure 1.4.: Interfacing an on-chip instrument as a JTAG data register

to the instruction register (IR) to properly set the address of the scan multiplexer. To access a given DR, two capture-shift-update operations are performed: one to set the corresponding instruction in the IR, and one more to access the DR itself.

Multiple scan chains or instruments can be connected to the JTAG circuitry as either:

- Single DR which integrates many scan chains or instruments into a single, long shift register. This scheme is beneficial when instruments are often accessed together (concurrently).
- Multiple DRs, requiring individual instructions (configurations of the IR) to address them. This scheme works well when instruments are often accessed individually, in a sequential manner.

In practice, depending on the application, a mixture of concurrent and individual accesses is required. The access time is proportional to the length of the shift register that is addressed by the JTAG scan multiplexer (cf. Figure 1.3) and includes the overhead of IR reconfiguration. To minimize the access time, only relevant instruments should be accessed, which calls for custom instructions for each required combination of accessed instruments. The number of such combinations can be exponential in the number of instruments. This causes not only high area overhead for the decode logic, scan multiplexers and control signal wiring, but also increases the access time due to the IR access overhead.

The scalability of scan infrastructures becomes crucial for efficient access to on-chip instrumentation in complex SoC designs. Scan architectures based exclusively on JTAG

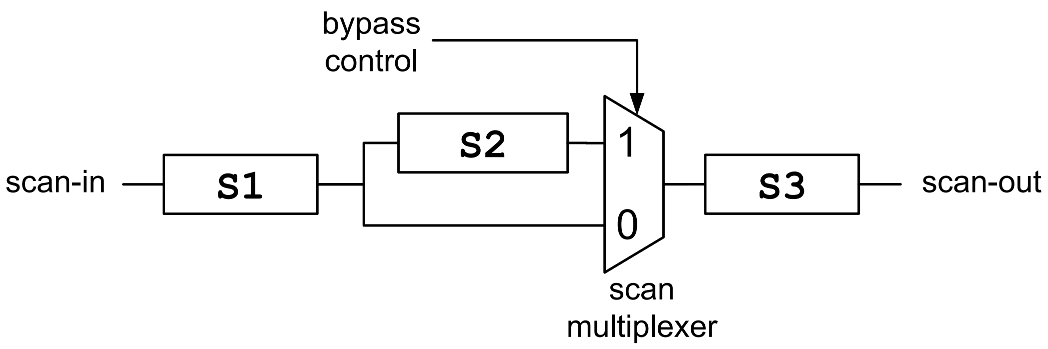


Figure 1.5.: Bypassing principle in reconfigurable scan architectures: Segment S2 can be excluded from the scan chain to improve the access time to S1 and S3.

do not scale well with the number of instruments [Larsson12, Ghani Zadegan12a]. To improve access flexibility and reduce the access time, various *bypass*-based scan architectures have been proposed. In such architectures, chains of scan cells called *scan segments* can be excluded from the scan chain if they need not be accessed. A scan bypass is implemented with a multiplexer, as shown at an example in Figure 1.5. The address input of the multiplexer is driven either by primary inputs, some bits of the JTAG instruction register, or by the content of some scan cells in the scan chain itself.

Various bypass-based scan architectures have been proposed in the past for the reduction of test application time and test data volume [Narayanan93, Kapur99, Samaranayake02, Samaranayake03, Arslan04, Quasem04, Xiang08, Chakraborty11], test compression and power reduction [Bhattacharya03], improved core isolation [Nadeau-Dostie09], and concurrent testing of SoC cores [Zorian98, Marinissen98, Whetsel99, Koranne03, Larsson03, Sehgal04, Benabdenbi00]. In more advanced architectures such as [Chakraborty11] or [Ghani Zadegan12a], the control of the bypass multiplexers is generated inside the scan network itself.

Reconfigurable Scan Networks (RSNs) are the most advanced scan architectures which integrate on-chip instruments together with configuration registers into a common scan network, as shown in Figure 1.6. Scan data are shifted from the primary scan-in, through a subset of instruments and configuration registers, down to the primary scan-out. The chosen *scan path*, i.e., the subset of accessible instruments and configuration registers depends on the state of the configuration registers themselves. Such a structure can be regarded as a JTAG data register (DR) with *variable length*. This kind of advanced RSNs emerge as a scalable option for the access to on-chip instrumentation,

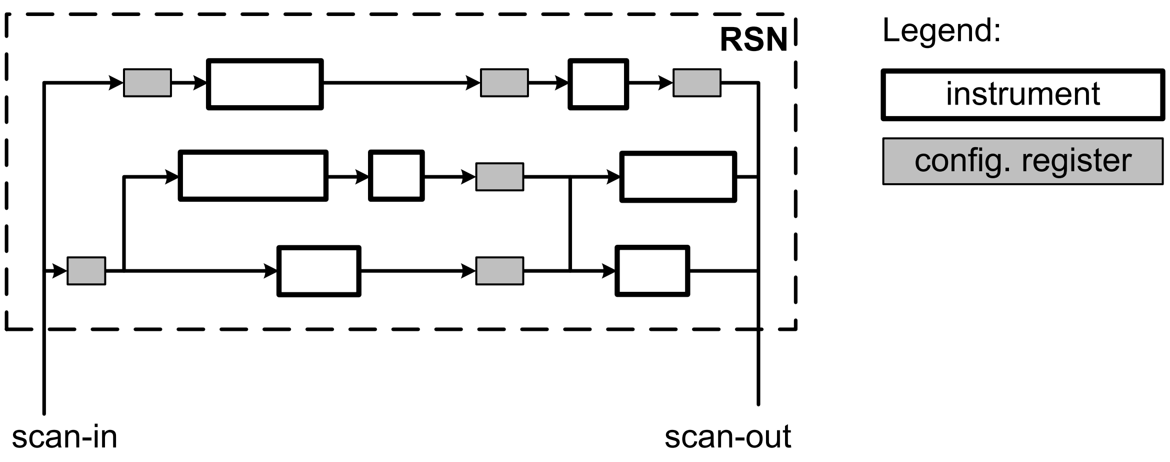


Figure 1.6.: Example of a Reconfigurable Scan Network (RSN) that integrates on-chip instruments with configuration registers

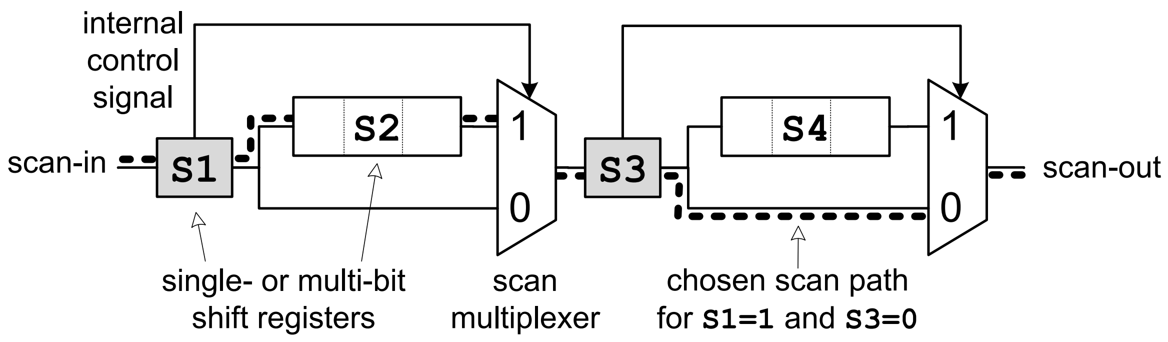

internal control signal
scan-in
S1
S2
1
0
S3
S4
1
0
scan-out
single- or multi-bit shift registers
scan multiplexer
chosen scan path for S1=1 and S3=0

Figure 1.7.: Detailed example of a Reconfigurable Scan Network (RSN)

offering a flexible, low-latency, and low-cost access [Rearick05, Abramovici06, Stollon11, Ghani Zadegan12a].

Figure 1.7 presents a more detailed example of an RSN that integrates two instruments `S2` and `S4` together with two configuration registers `S1` and `S3` using two scan multiplexers. The shadow registers of `S1` and `S3` drive the address ports of the scan multiplexers. The content of `S1` (`S3`) specifies if `S2` (`S4`) is connected to the scan chain or bypassed.

The widespread use of bypass-based and reconfigurable scan networks resulted in multiple standardization efforts in this area. IEEE Std. 1500 (Standard for Embedded Core Test, SECT) specifies configurable wrappers with a well defined, scan-based interface for embedded core test [Zorian05]. The novel IEEE Std. 1149.1-2013 (JTAG-2013) defines *excludable* and *selectable* scan cells for efficient access to systems with power-gated components [JTA13]. Several standards have been ratified recently for scan-based access to on-chip instrumentation, e.g. for configuring programmable devices (IEEE Std. 1532) or access to debug instrumentation (IEEE Std. 5001, Nexus) [Vermeulen08]. The emerging IEEE Std. P1687 (*Standard for Access and Control of Instrumentation Embedded within a Semiconductor Device*), also known as IJTAG for *internal* JTAG, targets advanced scan-based interfacing and reuse of arbitrary on-chip instrumentation [Rearick05, Eklow06, Stollon11, Ghani Zadegan12b]. Unlike existing architectures and standards for reconfigurable scan-based access such as JTAG or SECT, IJTAG does not restrict the network architecture to regular structures and allows flexible RSNs with arbitrary control signals and distributed configuration.

1.4. Verification of Scan Infrastructure

Scan infrastructure must provide reliable access to embedded instrumentation throughout the life-cycle of a VLSI design, from production ramp-up to system maintenance. The malfunction of post-silicon debug infrastructure, for instance, would make the first silicon useless and could have a disastrous impact on the time to market. The operability of on-chip instrumentation is also mandatory for in-field maintenance: The instruments for system monitoring, built-in self test, system diagnosis, reconfiguration, and repair must function correctly throughout the lifetime of the chip.

Today's SoCs already take advantage of the flexibility offered by simple reconfigurable scan networks [Rearick05]. For emerging scan architectures, scalable verification techniques become mandatory to meet the requirements on system reliability, safety and security, as well as to capture infrastructure bugs early and facilitate smooth production ramp-up.

A recent industrial survey on VLSI circuit design and verification reveals that, for an average project, the verification effort outweighs the design effort, while formal verification techniques are increasingly being adopted to cope with the growing design complexity [Foster11]. Unlike simulation based methods which can verify design correctness only for a very limited subset of possible system executions, formal verification is able to *guarantee* design correctness by proving that the implementation meets the specification for *all* possible executions [Kropf99]. This characteristic of formal methods is key to the verification of safety and security properties.

While efficient formal verification techniques exist for combinational and some classes of sequential circuits, reconfigurable scan networks pose a serious challenge. RSNs have a very limited serial interface and exhibit deeply sequential behavior. Due to the high sequential depth, existing formal verification methods are ineffective for proving RSN properties, as indicated by the case study in Section B.3.

1.5. Access Security

Good observability and controllability of chip internals is requisite for low time to market, high product quality, as well as system reliability and maintainability. In many applications domains, however, the accessibility of the on-chip instrumentation clashes with safety and security requirements [Tehranipoor11, Da Rolt12].

The security of scan-based access is crucial for prevention of criminal activities such as sabotage, unlicensed usage, or intellectual property (IP) theft. An attacker may exploit the scan infrastructure to gain access to protected data (secret key or IP), alter the system state by fault injection, or perform illegal operations. Successful attacks on the JTAG interface are reported for pirating satellite TV services or circumvent mechanisms for Data Rights Management (DRM) [Tehranipoor11]. On-chip instruments may potentially expose sensitive data that are otherwise protected in the system. For instance, even if the registers holding private data are not scannable, the scan infrastructure is still prone to side-channel attacks on cryptographic cores [Yang04].

Different levels of infrastructure accessibility are required during product development, volume production, and in-field operation. In production ramp-up, volume test and diagnosis, high observability and controllability is key to low time to market and high product quality. However, during in-field operation and maintenance, the accessibility of chip internals must be restricted due to security and safety reasons, e.g. to prevent IP theft or tampering. Moreover, different accessibility levels may be required depending on the eligibility of the user. In automotive applications, for instance, full access is mandatory during manufacturing and assembly test, while only limited access is allowed during operation and maintenance in a workshop to prevent unauthorized chip tuning.

It is a well known fact that absolute security is impossible, and given enough time and financial resources, any security mechanism can be broken [Tehranipoor11]. Practical solutions for scan-based access strike a trade-off between security, flexibility, and cost. Typical protection methods include access authorization, scan data encryption with stream ciphers, and scan chain obfuscation [Rosenfeld10]. The goal of these approaches is to assure that only users who know a shared secret (e.g. encryption key, or obfuscation principle) can access the scan infrastructure. If the shared secret is known to the attacker, full access becomes possible which is unacceptable in safety critical applications.

To guarantee inaccessibility of protected registers, the physical interface or parts of scan infrastructure can be made permanently unusable after they are not needed anymore. Such blocking schemes are implemented for instance using One Time Programmable (OTP) on-chip memory cells called fuses [Ebrard09]. Fuse-based protection prevents any access to protected structures at the cost of reduced flexibility and accessibility: By blowing an on-chip fuse, some instructions of the JTAG TAP controller or chosen scan chains can be permanently disabled [Sourgen92].

In flexible RSN architectures, the access to each on-chip instrument can be realized in many ways, using different configurations of the network. Even if some configurations are permanently blocked for security reasons, some other configurations may potentially give access to sensitive instruments or allow side-channel attacks. The increasing complexity of RSNs calls for efficient and scalable methods that guarantee the security of on-chip instrumentation.

1.6. Overview and Contributions

This chapter introduced two design automation challenges posed by reconfigurable scan networks: the difficulties of formal design verification and the need for secure access to on-chip instrumentation. While mature techniques exist for regular scan architectures with no or limited configurability, the high complexity and deep sequential nature of RSNs call for novel modeling methods and algorithms that can handle current and future RSN designs in a scalable way.

The content of this thesis is structured as follows:

Chapter 2 – *Formal Foundation* – introduces the basics of formal methods and notation used in this thesis. A discussion of circuit models and abstractions is followed by formal design specification methods. The problems of model checking, Boolean satisfiability (SAT) solving, and pseudo-Boolean optimization are introduced.

Chapter 3 – *State of the Art* – discusses the existing verification methods for scan infrastructures as well as techniques for optimal and secure access. State-of-the-art formal verification algorithms for sequential circuits are reviewed. The scalability and limits of existing methods is discussed in the context of reconfigurable scan networks.

Chapter 4 – *Scan Network Modeling* – presents an efficient method for RSN modeling that exploits temporal abstraction to reduce modeling complexity. This novel modeling method greatly reduces the effort of algorithms for formal verification and secure access generation, as discussed in the following chapters.

Chapter 5 – *Formal Verification* – describes an application of the abstract model to verification of safety properties in RSNs, including accessibility and security. State-of-the-art SAT-based model checking techniques are extended to handle the verification of RSNs. Experimental results show that the proposed method can efficiently handle large and complex RSNs, and is far more scalable than state-of-the-art, general-purpose model checking tools.

Chapter 6 – *Access Optimization* – addresses an efficient pattern generation method for access to on-chip instrumentation. The problem of access time reduction is mapped to a pseudo-Boolean optimization problem leveraging the model from Chapter 4.

Chapter 7 – *Access Port Protection* – presents a novel method for RSN protection. To prohibit the access to protected instruments, the access port of the RSN is equipped with a *sequence filter*. The algorithm for sequence filter construction guarantees that all input sequences that could expose the content of protected instruments are rejected before any change to RSN's configuration is made.

Chapter 8 – *Conclusions* – recapitulates the contributions of this thesis and indicates future research directions that may benefit from this work.

2. Formal Foundation

This chapter provides the basic definitions and concepts that this thesis builds upon. After introduction of the formal notation and modeling of combinational and sequential circuits, fundamental concepts of formal verification and formal specification are discussed. This chapter also introduces the problems of Boolean satisfiability and pseudo-Boolean optimization, which constitute the foundation of the algorithms presented in Chapter 5 and 6.

2.1. Circuit Models

Digital designs can be divided into combinational and sequential circuits, depending on the existence of state-holding elements such as feedback loops, flip-flops, and latches. Combinational circuits are memory-less: The output of a combinational circuit depends exclusively on the current input pattern. In contrast, sequential circuits contain state-holding elements, and their output depends on the history of input patterns.

This section reviews the basic modeling techniques for combinational and sequential circuits. It distinguishes the structural and behavioral view, and introduces the basic models that are used throughout this thesis.

2.1.1. Combinational Circuits

From a structural point of view, combinational circuits are composed of building blocks that implement certain Boolean or arithmetic functions. At high abstraction levels, the building blocks realize complex Boolean or arithmetic functions, e.g. addition, multiplication, code-word generation, etc. The high-level description is refined in a manual or automated *synthesis* process into a lower-level representation that is closer to the actual hardware implementation.

In this thesis, combinational circuits are modeled structurally at *gate-level*. A gate-level circuit representation is a composition of primitive components called *logic gates* that realize basic Boolean functions such as conjunction (AND), disjunction (OR), negation (NOT), and the like. Unless otherwise noted, a zero-delay model is assumed, i.e. the timing of the actual hardware implementation is neglected.

Functional Representation

From a functional point of view, a combinational circuit C with $n \in \mathbb{N}^+$ inputs and $m \in \mathbb{N}^+$ outputs implements a multi-output Boolean function $\vec{f}_C : \mathbb{B}^n \to \mathbb{B}^m$. The Boolean function $\vec{f}_C$ constitutes the *behavioral model* of the circuit C: For each input assignment $\vec{i} = [i_1, i_2, \ldots, i_n]$, the function $\vec{f}_C$ provides the output assignment $\vec{o} = [o_1, o_2, \ldots, o_m] = \vec{f}_C(\vec{i})$.

The function $\vec{f}_C$ must be represented in a way suitable for design automation algorithms: High-level representations are beneficial for high performance simulation. In logic synthesis and equivalence checking, And-Inverter Graphs (AIG) are often used to represent combinational circuits [Mishchenko06]. In formal verification, various kinds of decision diagrams are applied [Kropf99].

Relational Representation

In formal verification, combinational circuits are often modeled in a relational way:

Definition 1. (Defining Relation) Let C be a combinational circuit with $n \in \mathbb{N}^+$ inputs and $m \in \mathbb{N}^+$ outputs that realizes a Boolean function $\vec{f}_C : \mathbb{B}^n \to \mathbb{B}^m$. The defining relation of C is a set:

$$\mathcal{R}_C := \{(\vec{i}, \vec{o}) \mid \vec{i} \in \mathbb{B}^n \wedge \vec{f}_C(\vec{i}) = \vec{o}\,\}. \tag{2.1}$$

Remark 1. The defining relation $\mathcal{R}_C$ includes all pairs of input and output assignments that are “allowed” in the circuit. Defining relations are often represented with *characteristic functions*. A general definition of characteristic functions is given below:

Definition 2. (Characteristic Function) Given a set X and its subset $Y \subseteq X$, the characteristic function of the subset Y, denoted as $\Omega_Y : X \to \mathbb{B}$, is defined as follows:

$$\Omega_Y(x) := \begin{cases} 1 & \text{if } x \in Y, \\ 0 & \text{otherwise}. \end{cases} \tag{2.2}$$

Remark 2. The characteristic function $\Omega_Y(\cdot)$ evaluates to *true* for all elements of Y, and to *false* for all elements in $X \setminus Y$. Given a combinational circuit C with the defining relation $\mathcal{R}_C$, the characteristic function of $\mathcal{R}_C$, denoted as $\Omega_R : \mathbb{B}^n \times \mathbb{B}^m \to \mathbb{B}$, evaluates to *true* only for pairs of input and output assignments that are allowed in the circuit.

Characteristic functions of defining relations for combinational designs are directly derived from the circuit structure by Tseitin transformation [Tseitin83, Biere09]. The Tseitin transformation leads to a representation in Conjunctive Normal Form (CNF). The size of the resulting CNF formula, i.e. the number of variables and clauses, scales linearly with the circuit size. This representation is favorable for various algorithm based on satisfiability solving (see Section 2.4).

2.1.2. Sequential Circuits

In contrast to combinational circuits, sequential circuits contain state-holding elements such as feedback loops, latches or flip-flops. In this thesis, the state-holding elements of a sequential circuits are assumed to be triggered simultaneously by the rising or falling transition of a single clock signal. Arbitrary sequential circuits with complex clocking schemes can be mapped to this single-clock, single-transition representation [Kropf99].

Sequential circuits that fulfill the single-clock, single-transition property can be represented structurally with the Huffman model [McCluskey86]. The Huffman model is composed of an array of memory elements and two combinational circuits that implement a state transition function $\vec{\delta}$ and an output function $\vec{\lambda}$, as shown in Figure 2.1. The input, output, and the memory elements are represented with signal vectors: $\vec{x} = [x_0, x_1, \ldots, x_n]$, $\vec{y} = [y_0, y_1, \ldots, y_m]$, $\vec{s} = [s_0, s_1, \ldots, s_k]$, respectively.

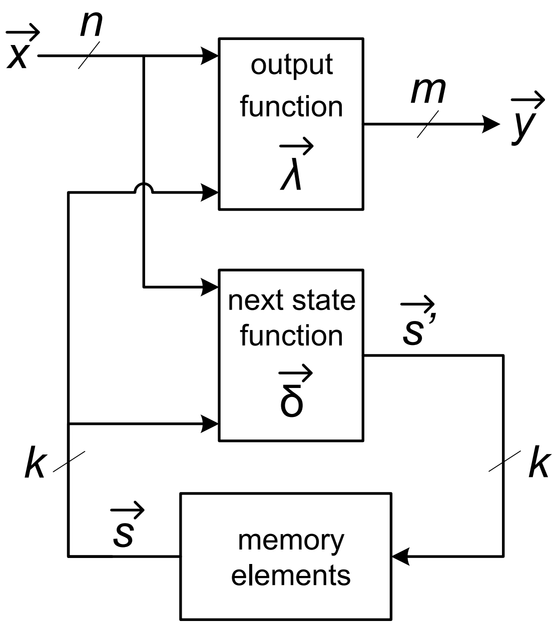


Figure 2.1.: Huffman model of a sequential circuit

Finite State Machine Model

A sequential circuit that follows the Huffman model can be modeled functionally as a *Finite State Machine* (FSM). In the FSM abstraction, the inputs, outputs, and memory elements of a sequential circuit are modeled as Boolean variables, while the state of the circuit is modeled as an assignment to these variables. The two combinational logic blocks of the Huffman model are represented by Boolean functions.

Definition 3. (Finite State Machine) A finite, deterministic state machine $\mathcal{M}$ (Mealy automaton) is a 6-tuple $\mathcal{M} = (\mathbb{B}^k, \mathbb{B}^n, \mathbb{B}^m, \vec{\delta}, \vec{\lambda}, \vec{s^0})$, where $k, n, m \in \mathbb{N}^+$, $\mathbb{B}^k$ is the internal state space of the FSM, $\mathbb{B}^n$ is the input pattern space, and $\mathbb{B}^m$ is the output pattern space. $\vec{s^0} \in \mathbb{B}^k$ is the initial state, while $\vec{\delta} : \mathbb{B}^k \times \mathbb{B}^n \to \mathbb{B}^k$ is the state transition function that maps the current internal state $\vec{s} \in \mathbb{B}^k$ and the input assignment $\vec{i} \in \mathbb{B}^n$ into the next state denoted by $\vec{\delta}(\vec{s}, \vec{i})$. $\vec{\lambda} : \mathbb{B}^k \times \mathbb{B}^n \to \mathbb{B}^m$ is the output function that maps the current internal state $\vec{s} \in \mathbb{B}^k$ and the input assignment $\vec{i} \in \mathbb{B}^n$ into the output assignment denoted by $\vec{\lambda}(\vec{s}, \vec{i})$.

Remark 3. The FSM abstraction applies directly to the structural Huffman model of a sequential circuit with k memory elements, n inputs, and m outputs, as shown in Figure 2.1. In every clock cycle, the state of the FSM changes according to the formula $\vec{s} := \vec{\delta}(\vec{s}, \vec{x})$. The FSM output is defined by the output function: $\vec{y} := \vec{\lambda}(\vec{s}, \vec{x})$.

Kripke Structure

In formal verification, sequential circuits are often modeled as *Kripke structures* [Kripke63, Clarke99]. A Kripke structure is a graph with nodes representing the circuit states, and edges representing state transitions. Compared to the FSM model, a Kripke structure does not explicitly model the input assignment nor the output function of the circuit.

Definition 4. (Kripke Structure) Let A be a set of atomic propositions. A Kripke structure $\mathcal{K}$ over A is a 4-tuple $\mathcal{K} := (S, I, T, L)$, where S is a finite set of states, $I \subseteq S$ is a set of initial states, $T \subseteq S \times S$ is a transition relation, and $L : S \to \mathcal{P}(A)$ is a state labeling function. The transition relation T is *total*, i.e. $\forall_{s \in S}\ \exists_{s' \in S}\ (s, s') \in T$. The labeling function L maps each state $s \in S$ to the set of atomic propositions that hold in this state, denoted by $L(s)$. $T(s, s')$ denotes that $(s, s') \in T$, and $I(s)$ denotes that $s \in I$.

The temporal behavior of a Kripke structure is described with *execution paths* (or simply *paths*), as defined below:

Definition 5. (Execution Path) A path π in a Kripke structure $\mathcal{K} = (S, I, T, L)$ is defined as a sequence of states $\pi =< s_0, s_1, s_2, \ldots >$ such that $\forall_{i \geq 0}\ (s_i, s_{i+1}) \in T$. The i-th element of the path π is denoted by $\pi(i) := s_i$. The suffix of the path π that starts with the i-th element is denoted by $\pi^i :=< s_i, s_{i+1}, s_{i+2}, \ldots >$. The path π is initialized if and only if it satisfies $\pi(0) \in I$. For a bounded path $\pi_b =< s_0, s_1, s_2, \ldots, s_n >$, path length is defined as $|\pi_b| := n$.

The computational complexity of formal verification algorithms depends on various characteristics of the Kripke structure, such as the total number of states, the *diameter*, or the *recurrence diameter*, as defined below (after [Biere09]).

Definition 6. (Diameter) The diameter $d(\mathcal{K})$ of a Kripke structure $\mathcal{K} = (S, I, T, L)$ with the set of initialized paths Π is defined as the length of the longest path among all shortest paths between pairs of reachable states in $\mathcal{K}$:

$$d(\mathcal{K}) := \max\big\{\ k \mid a, b \in S,\ k = \min\{\ l \mid \pi \in \Pi,\ \pi(0) = a,\ \pi(l) = b\}\ \big\}. \qquad (2.3)$$

The diameter of a Kripke structure is also called state graph *eccentricity*.

Definition 7. (Recurrence Diameter) The recurrence diameter $d_r(\mathcal{K})$ of a Kripke structure $\mathcal{K} = (S, I, T, L)$ with the set of initialized paths Π is defined as the length of the longest simple path in $\mathcal{K}$ (i.e., longest path with unique states, or equivalently, longest *loop-free* path):

$$d_r(\mathcal{K}) := \max \left\{ k \mid \pi \in \Pi, \bigforall_{0<i\leq k,\ 0<j\leq k,\ i\neq j} \pi(i) \neq \pi(j) \right\}. \tag{2.4}$$

2.2. Model Checking and Formal Specification

Model checking is an automated *formal verification* technique for proving properties of finite state concurrent systems. Unlike simulation, model checking is able to exhaustively cover the state space of the system and hence *guarantee* its correctness with respect to given specification [Clarke99].

Model checking requires an adequate *model* of the system implementation and its *formal specification*. The implementation is usually given as an FSM model or a Kripke structure [Clarke99]. Depending on the form of formal specification, model checking techniques can be classified into:

- *Equivalence checking*, where the formal specification constitutes a golden model of the system and its equivalence with the implementation is analyzed.
- *Property checking*, where the formal specification is a (possibly incomplete) set of properties, the validity of which is checked in the implementation.

2.2.1. Linear Temporal Logic

The most common formalisms for property specification are Linear Temporal Logic (LTL) and Computational Tree Logic (CTL) [Kropf99]. This thesis uses a subset of LTL which is developed in [Pnueli77]. In the following, the simplified semantics of LTL is presented after [Clarke99].

Let A be a set of atomic propositions. The syntax of LTL formulas over A is defined as follows:

- If $p \in A$, then p is an LTL formula,

- If g and h are LTL formulas, then $\neg g$, $g \wedge h$, and $g \vee h$ are LTL formulas.
- If g is an LTL formula, then $\boldsymbol{X}\ g$, $\boldsymbol{G}\ g$, and $\boldsymbol{F}\ g$ are LTL formulas.

The characters $\boldsymbol{X}$, $\boldsymbol{G}$, and $\boldsymbol{F}$ denote temporal operators which are read as "next", "always", and "eventually", respectively. The semantics of an LTL formula f is defined over a Kripke structure $\mathcal{K} = (S, I, T, L)$ with the set of initialized execution paths Π. Given a path $\pi \in \Pi$, the notation $\pi \models f$ means that f holds along the path π. Assuming that $p \in A$ and g, h are LTL formulas, the satisfaction operator $\models$ is defined recursively as follows:

$$\pi \models p \quad \Leftrightarrow \quad p \in L(\pi(0)) \tag{2.5}$$

$$\pi \models \neg g \quad \Leftrightarrow \quad \pi \not\models g \tag{2.6}$$

$$\pi \models g \wedge h \quad \Leftrightarrow \quad \pi \models g \wedge \pi \models h \tag{2.7}$$

$$\pi \models g \vee h \quad \Leftrightarrow \quad \pi \models g \vee \pi \models h \tag{2.8}$$

$$\pi \models \boldsymbol{X}\ g \quad \Leftrightarrow \quad \pi^1 \models g \tag{2.9}$$

$$\pi \models \boldsymbol{G}\ g \quad \Leftrightarrow \quad \forall_{i \geq 0}\ \pi(i) \models g \tag{2.10}$$

$$\pi \models \boldsymbol{F}\ g \quad \Leftrightarrow \quad \exists_{i \geq 0}\ \pi(i) \models g \tag{2.11}$$

An LTL formula f is satisfied by the Kripke model $\mathcal{K}$, denoted as $\mathcal{K} \models f$, if and only if all paths $\pi \in \Pi$ satisfy f. The task of LTL model checking is to determine whether $\mathcal{K} \models f$. If the property does not hold, the model checker produces a *counterexample*, i.e. an execution path allowed in $\mathcal{K}$ that contradicts the LTL formula. State-of-the-art model checking algorithms are reviewed in Section 3.2.

This thesis focuses on model checking of simple *reachability* properties (also called *safety* properties) of the form $\boldsymbol{G}\ p$ (or equivalently, $\neg \boldsymbol{F}\ \neg p$), where p is an atomic proposition, e.g. a Boolean function defined over the state of the model. More complex specifications, including *liveness* and *fairness* properties, can be efficiently transformed into reachability properties and handled with algorithms for reachability checking, as shown in [Biere02].

2.3. Model Abstraction

To improve the tractability and performance of model checking algorithms, irrelevant details of the circuit implementation should be removed from the circuit model. This technique is known as *model abstraction*. Two types of abstraction techniques are distinguished in this thesis:

- *Structural abstraction* reduces the level of modeling detail or removes those model elements that are irrelevant to a certain property. Examples of structural abstraction techniques include *cone of influence reduction* and *data abstraction*, as defined in [Clarke99].
- *Temporal abstraction* aims to reduce the model complexity by a simplification of its temporal behavior. The RSN modeling technique developed in Section 4.4 is an example of such an abstraction.

Abstraction techniques are used to prove properties that would be computationally too expensive to prove in the concrete model. Ideally, model checking should lead to the same results in the concrete model and its abstraction. In practice, however, the applicability of an abstraction is usually restricted to a certain class of properties, for which the abstraction is *sound* and *complete*, as defined below.

Definition 8. (Abstraction Soundness) An abstraction $\mathcal{M}'$ of a model $\mathcal{M}$ is sound with respect to a set of properties P if and only if every property $p \in P$ that holds in the abstract model $\mathcal{M}'$ also holds in $\mathcal{M}$.

Definition 9. (Abstraction Completeness) An abstraction $\mathcal{M}'$ of a model $\mathcal{M}$ is complete with respect to a set of properties P if and only if every property $p \in P$ that holds in the concrete model $\mathcal{M}$ also holds in $\mathcal{M}'$.

If $\mathcal{M}'$ is an *incomplete* abstraction of $\mathcal{M}$, model checking of $\mathcal{M}'$ may result in *spurious counterexamples*. A spurious counterexample is an execution path that refutes the given property in $\mathcal{M}'$ but is inconsistent with $\mathcal{M}$, and hence is not a valid counterexample to the property.

2.4. Boolean Satisfiability

Boolean SATisfiability (SAT) is a decision problem that asks whether a given Boolean formula can be satisfied. Formally, the SAT problem for a given formula representing a Boolean function $f : \mathbb{B}^n \to \mathbb{B}$ consists in searching for an assignment $a \in \mathbb{B}^n$, such that $f(a) = 1$. The Boolean formula subject to satisfiability analysis is called *SAT instance*. The assignment that satisfies the SAT instance is called *satisfying assignment*. A formula (SAT instance) for which there exists a satisfying assignment is called *satisfiable*. If no such assignment exists, the formula is called *unsatisfiable*.

The majority of SAT solving algorithms are developed for instances in Conjunctive Normal Form (CNF). A CNF formula has the following form:

$$(l_{1,1} \lor l_{1,2} \lor l_{1,3} \lor \ldots) \land (l_{2,1} \lor l_{2,2} \lor l_{2,3} \lor \ldots) \land (l_{3,1} \lor l_{3,2} \lor l_{3,3} \lor \ldots) \land \ldots \tag{2.12}$$

where each $l_{i,j}$ is a *literal,* i.e. a Boolean variable v or its negation $\neg v$, while disjunctions of literals are called *clauses*.

The SAT problem for instances in CNF is NP-complete, hence, unless P = NP, there exists no algorithm that efficiently handles arbitrary instances in CNF [Garey79]. The recent developments in SAT technology, however, have paved the way for wide adoption of SAT solvers in various design automation, test generation, and verification tasks. The state-of-the-art SAT solvers most often rely on a search-based procedure known as the DPLL algorithm [Davis62]. An overview of state-of-the-art SAT algorithms is found in [Biere09].

2.4.1. Incremental SAT Solving

State-of-the-art SAT solvers leverage learning techniques to speedup the search for satisfying assignments. During the search process, sets of assignments that do not satisfy the formula are identified, enlarged, and stored in form of *learned clauses*. The *learned clauses* are added to the SAT instance to prune the search space [Biere09].

In many applications including formal verification, satisfiability solving is used in an iterative manner: The SAT solver is invoked multiple times with slightly modified SAT instances. Ideally, the SAT instance should be reused in consecutive iterations to avoid the costly process of instance parsing, reduction, and clause learning. However, the modification of a SAT instance in a learning-based SAT solver is hard: Every invocation of the SAT solver extends the instance with learned clauses which hold only for the actual instance. To avoid the effort of book-keeping and removal of out-of-date learned clauses, a technique known as *incremental SAT solving* is employed [Eén03].

An incremental SAT solver checks the satisfiability of a formula of the form $F \wedge A$, where:

- F is a formula in CNF,
- $A = a_0 \wedge a_1 \wedge a_2 \wedge \ldots$ is a conjunction of *assumptions*, where a_i are unit clauses (literals).

To iteratively check the satisfiability of F for distinct assumptions in A, the incremental SAT solver can be restarted multiple times without any performance loss due to clause removal [Eén03].

Incremental SAT solving is often used to iteratively check the satisfiability of an instance with and without a subset of its clauses. For example, assume that the satisfiability of two formulas is checked: F and $F \wedge (x \vee y \vee z)$. To this end, the satisfiability of another formula $F' := F \wedge (a \vee x \vee y \vee z)$ may be checked, where a is a *selector variable* which is used as an *assumption*. The formula F is satisfiable if and only if $F' \wedge (a)$ is satisfiable. Similarly, the formula $F \wedge (x \vee y \vee z)$ is satisfiable if and only if $F' \wedge (\neg a)$ is satisfiable. As an incremental SAT solver can reuse the instance F' together with the learned clauses, the check for the satisfiability of $F' \wedge (a)$ and $F' \wedge (\neg a)$ is performed efficiently by two consecutive invocations of the solver.

2.5. Pseudo-Boolean Satisfiability and Optimization

Pseudo-Boolean Satisfiability (PBSAT) is an extension of the SAT problem: While a SAT instance is a propositional logic formula in CNF, the instance of a PBSAT problem is a conjunction of *pseudo-Boolean constraints*, as defined below:

Definition 10. (Pseudo-Boolean Constraint) A pseudo-Boolean constraint (PB-constraint) is an inequality of the form $C_0 l_0 + C_1 l_1 + C_2 l_2 + \ldots + C_{n-1} l_{n-1} \geq C_n$, where $C_i \in \mathbb{Z}$ are integer coefficients and l_i are literals. A coefficients C_i is weighted with value 1 if the corresponding literal l_i is *true*, and with 0 otherwise. The PB-constraint is satisfied if and only if the weighted sum of coefficients $C_0 \ldots C_{n-1}$ is larger than or equal to C_n. If all integer coefficients are set to 1, the PB-constraint becomes a standard clause.

The PBSAT problem asks whether a conjunction of PB-constraints is satisfiable. Formally, given a conjunction of PB-constraints representing a Boolean function $f : \mathbb{B}^n \to \mathbb{B}$, the PBSAT problem consists in searching for an assignment $a \in \mathbb{B}^n$, such that $f(a) = 1$. The PBSAT problem can be solved using a SAT solver by PB-constraint translation to Boolean clauses [Eén06].

Pseudo-Boolean Optimization (PBO) is an extension of the PBSAT problem, where the solution must both satisfy a set of PB-constraints and minimize a given cost function. Formally, given a conjunction of PB-constraints representing a Boolean function $f : \mathbb{B}^n \to \mathbb{B}$ and a cost function $h : \mathbb{B}^n \to \mathbb{Z}$ defined over the set of assignments to f, the PBO problem consists in searching for an *optimal assignment* $a \in \mathbb{B}^n$, such that $f(a) = 1$ and $\forall_{a' \in \mathbb{B}^n} \big[[f(a') = 1] \Rightarrow [h(a') \geq h(a)] \big]$. Pseudo-Boolean optimization can be performed, for instance, by iterative SAT solving with PB-constraints translated to clauses [Eén06], or with methods based on speculative model enumeration [Gebser11].

3. State of the Art

This chapter discusses the state of the art in verification, access scheduling, and access protection for scan infrastructures. Existing verification methods are introduced and their applicability to reconfigurable scan networks is discussed. For scalable formal verification, abstraction techniques are reviewed. Finally, recent techniques for access pattern generation and infrastructure protection are presented.

3.1. Validation and Verification of Scan Networks

Since on-chip instrumentation is key to rapid production ramp-up and high product quality, its access mechanism must be thoroughly verified to avoid costly design bugs and prevent in-field dependability issues. In the following, the state-of-the-art techniques for the validation and verification of scan infrastructures are reviewed.

Scan networks are essentially sequential circuits and hence may suffer from timing violations [Wu98]. As the critical paths in scan networks are short compared to system logic, the most common timing issue is *hold-time violation*. *Setup-time violation* may still occur due to long signal propagation between distant scan registers. To verify the timing closure of *shift* and *capture* operations in a JTAG circuitry, Static Timing Analysis (STA) is used [Remmers04]. Timing analysis for deep-submicron technologies requires consideration of statistical delay models [Blaauw08].

Design rules, either imposed by a standard or recommended as good design practice, are usually verified by structural analysis: Multiple drivers, broken scan chains, and loop-backs can be found by structural traversal of the network [Fisher02]. The structure of the IEEE 1149.1 (JTAG) circuitry, including the TAP controller and the connectivity of data registers, can be verified by logic tracing [Melocco03].

The functionality of JTAG circuitry including the TAP controller can be validated by simulation using automatically generated stimuli [Bruce Jr96]. Similarly, the functionality of IEEE 1500 wrappers can be validated by coverage-driven, constrained-random simulation [Diamantidis05]. The stimuli are chosen in such a way as to maximize coverage of the behavioral rules [Benso08]. Such simulation based techniques can verify that the scan infrastructure works correctly in predefined scenarios, but cannot guarantee the absence of design errors in general.

Accessibility of scan registers requires that a primary input sensitizing condition (called scan state) exists, such that the scan network functions as a shift register [Eichelberger77]. The accessibility and connectivity of Level Sensitive Scan Design (LSSD) can be verified with expert systems [Horstmann84,Papaspyridis88]. Certain properties of scan infrastructures, such as the functionality of a reset signal or the equivalence of two scan network models, can be verified by a reduction to combinational equivalence checking [Kamepalli06]. The functionality of the JTAG circuitry can be verified by symbolic simulation [Bryant90, Singh97] or four-valued logic simulation using preconditioning and checking sequences [Dahbura89, Melocco03].

While the existing verification techniques efficiently handle simple scan chains, the verification of reconfigurable scan networks poses a much more difficult problem. Control signals for scan registers may be generated by combinational logic driven by other scan registers in the same or different hierarchy levels. In core-based design, scan networks may be composed of third-party modules, the behavior of which may not be fully disclosed. As a consequence, certain configurations may be illegal or contradictory, causing integration issues, such as exclusive or limited access to certain scan registers. An exhaustive search may be required to find a valid access sequence or to prove inaccessibility [Baranowski12].

An example is given in Figure 3.1, where the access to scan segment 2 is controlled by bits a and b of scan segment 1. Such a structure is compliant with IEEE Std. P1687 and can result e.g. from erroneous integration of design modules. Clearly, there exists no assignment to bits a and b such that segment 2 is part of the chosen scan path—there exists a *combinational dependency* that cannot be satisfied. Combinational Automatic Test Pattern Generation (ATPG) [Bushnell00] can be used to prove segment 2 inaccessible because the dependency is of combinational nature. However, such dependencies can also be sequential: Even if there exists an assignment that puts the target segment on the chosen scan path, this assignment may be not reachable from the initial state of the network. Due to such *sequential dependencies*, both the verification of and access

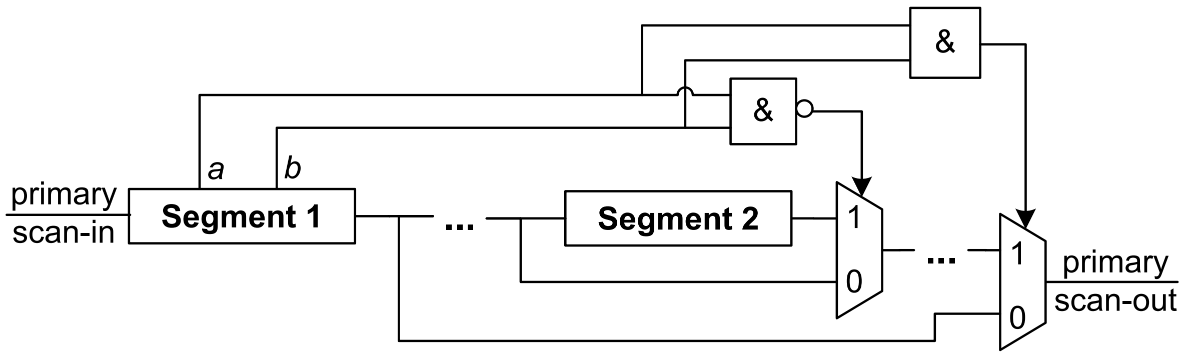


Figure 3.1.: Example of a reconfigurable scan network with conflicting access conditions

pattern generation for RSNs is an NP-hard decision problem which is similar to sequential stuck-at fault ATPG. While state-of-the-art sequential ATPG algorithms can handle sequential depths of several dozens of clock cycles, an access to a reconfigurable scan network may require justification over hundreds of thousands cycles. Moreover, for verification of more complex properties, e.g. unreachability of illegal scan configurations or safety and security related requirements, a dedicated formal verification technique is required. The first scalable method for formal verification of RSNs is developed in [Baranowski12] and discussed in Chapter 5.

3.2. Model Checking

As reconfigurable scan networks are essentially sequential circuits, existing model checking methods can potentially be used to verify them. This section discusses the most promising model checking techniques and analyzes their scalability for complex RSNs.

Given an FSM model and a temporal logic formula expressing its desired property, model checking consists in an exhaustive search over the set of reachable states to check if the property always holds in the model [Clarke99]. If the property does not hold, model checking returns a counterexample, i.e., an execution path that refutes the property.

The early CTL model checking method by Clarke et al. [Clarke86] traverses an explicit representation of the state space of an FSM (Kripke structure) and labels each state with the satisfied temporal formulas until a fixed point is reached. This algorithm has polynomial complexity in the size of the Kripke structure, which can be exponential in the number of sequential elements [Clarke99]. For this reason, explicit model checkers suffer from scalability issues in hardware verification and are primarily used to verify process interactions and communication protocols, as in [Holzmann97].

As an FSM with a hundred of state elements has potentially $2^{100} \approx 10^{30}$ reachable states, designs with a few dozens of sequential elements may be already too complex to handle with explicit model checking techniques. McMillan et al. in [McMillan93] propose a method for symbolic representation of states and symbolic state space traversal, called *symbolic model checking*. Instead of the explicit state graph representation such as Kripke structure, symbolic model checking uses characteristic functions to represent state sets and the FSM's transition relation. Symbolic traversal of the state space consists in manipulating the characteristic functions: The set of reachable states is found by calculating the fixed point of the transitive image for the set of initial states. To speed-up the process of quantification and fixed point calculation, the characteristic functions are represented with canonical Binary Decision Diagrams (BDD) [Bryant86].

Both the calculation of the set of reachable states (symbolic reachability) and LTL model checking in general are PSPACE complete problems [Sistla85,Prasad05]. While symbolic model checking has significantly improved scalability over explicit model checking techniques, it still faces scalability issues when the number of sequential elements exceeds a few hundred [Biere09]. This is mainly due to quantifier elimination required for the image calculation, which often leads to BDD blow-up.

Recent research in model checking concentrates on scalability improvement by using efficient heuristics to solve partial problems, e.g. property falsification. Most promising techniques map such problems to Boolean satisfiability in order to leverage the recent progress in SAT technology [Prasad05]. Such SAT-based model checking techniques are summarized in the following sections.

3.2.1. Bounded Model Checking

Bounded model checking (BMC) is a successful formal verification technique based on SAT procedures. The goal of BMC is to check whether a given temporal logic property holds in all initialized, bounded execution paths of an FSM [Biere99]. BMC is very efficient at early detection of design bugs. In this application, it significantly outperforms BDD-based symbolic model checkers [Biere03]. However, the maximal bound that can be examined is limited by the memory and runtime capacity of the SAT solver. As only bounded execution paths are considered, BMC cannot prove LTL properties such as $\boldsymbol{G}\ p$ (p always holds) or $\boldsymbol{F}\ p$ (p eventually holds).

Formally, given a property expressed with an LTL formula f, a finite state machine $\mathcal{M}$, and a bound $k \in \mathbb{N}$, *bounded model checking* consists in proving that every initialized execution path π in $\mathcal{M}$, such that $|\pi| \leq k$, satisfies f, written as $\pi \models f$. The property is disproved if there exists a path π_c of length $n \leq k$ such that $\pi_c \not\models f$. If such a path exists, it constitutes a counterexample to the property.

A BMC instance is encoded as a SAT instance by unrolling the transition relation of $\mathcal{M}$ for consecutive *time steps* (clock cycles). In each time step, the state elements and input/output ports of $\mathcal{M}$ are modeled with a distinct set of Boolean variables. For simple LTL properties of the form $\boldsymbol{G}\ p$, where p is a Boolean function defined over the state elements and input/output ports, the SAT instance is formed as follows:

$$\varphi(k) := \Omega_I(V_0) \wedge \left[\bigwedge_{n=0}^{k-1} \Omega_T(V_n, V_{n+1})\right] \wedge \left[\bigvee_{n=0}^{k} \neg p(V_n)\right], \tag{3.1}$$

where V_i is the set of state and input/output variables in the i-th time step, while Ω_I and Ω_T are the characteristic functions of the set of initial states and the transition relation of $\mathcal{M}$, respectively. Note that function Ω_I is applied to the set of variables of the initial time step, while Ω_T and p are applied to the variable sets of the consecutive time steps.

The formula $\varphi(k)$ is satisfiable if and only if there exists a counterexample to property $\boldsymbol{G}\ p$ of length k or less. $\varphi(k)$ is typically transformed to conjunctive normal form (CNF) and its satisfiability is checked using a conventional SAT solver. Such SAT instances can also be constructed for more complex LTL formulas, including liveness properties, as shown in [Biere99, Biere03]. SAT encodings that are linear in the bound of BMC are developed for LTL properties in [Biere06].

BMC techniques are well established and enjoy wide adoption in the industry [Prasad05, Biere09]. However, the size of SAT instances grows linearly with the bound of BMC. To find realistic design bugs in RSNs, prohibitively high bounds may be required due long scan paths and complex sequential dependencies. To deal with the high sequential depth of RSNs, a novel temporal abstraction is developed in Chapter 4 and used for bounded model checking in Chapter 5.

3.2.2. Completeness

A *complete* verification method always terminates with a definite verification response, i.e. either proves or refutes a property. BMC is not complete for LTL as it can only examine execution paths of bounded lengths. However, if the bound is sufficiently high—for instance, if it equals the total number of FSM states and the property has the form $\boldsymbol{G}\ p$—BMC can exhaustively cover all possible execution paths and hence either guarantee that the property *always* holds or provide a counterexample [Biere09].

The bound that allows to verify properties in the unbounded sense is called *completeness threshold*. Biere et al. in [Biere03] show that the completeness threshold for unnested LTL properties like $\boldsymbol{G}\ p$ can be no higher than the model diameter (Definition 6, p. 27). As the calculation of the exact diameter requires satisfiability solving of Quantified Boolean Formulas (QBF), it is not easier than symbolic model checking itself, since both are PSPACE-complete problems [Biere09].

Several methods for calculating an overapproximation of the model diameter have been proposed. Clarke et al. use Büchi automata constructed for a specific property [Clarke04]. Biere et al. develop SAT procedures that calculate the recurrence diameter (Definition 7, p. 28) [Biere03]. However, the recurrence diameter can be *arbitrarily* larger than the model diameter itself [Biere09]. Baumgartner et al. develop a diameter approximation method based on structural circuit analysis [Baumgartner02]. They show, for instance, that a memory with n rows has a diameter of n since any state is reachable within n clock cycles required for n write operations, regardless of the memory word length.

Completeness by Induction

To prove a property of the form $\boldsymbol{G}\ p$, where p is a Boolean formula defined over the state and input/output assignment of an FSM, it is necessary to show that p holds for all reachable states in the FSM. A sufficient but not necessary condition is that p holds in the initial state and is preserved by the FSM's transition relation. This condition is encoded as two separate SAT instances (after [Sheeran00]):

$$\varphi_{\text{initial}} := \Omega_I(V_i) \wedge \neg p(V_i), \tag{3.2}$$

$$\varphi_{\text{induction}} := p(V_i) \wedge \Omega_T(V_i, V_j) \wedge \neg p(V_j), \tag{3.3}$$

where V_i, V_j are two sets of state variables, while Ω_I and Ω_T are the characteristic functions of the set of initial states and the transition relation of the FSM, respectively.

The formula φ_{initial} is unsatisfiable if and only if p holds in the initial state. The formula $\varphi_{\text{induction}}$ is unsatisfiable if and only if p is preserved by the transition relation. If both formulas are unsatisfiable, p is an *inductive invariant* of the transition relation, and hence $\boldsymbol{G}\ p$ holds.

This simple verification technique is not complete since $\boldsymbol{G}\ p$ may hold even if p is not an inductive invariant of the transition relation. Completeness can be achieved with k-induction proposed in [Sheeran00] and later improved with incremental SAT solving techniques in [Eén03]. In this technique, the transition relation is unrolled several times for consecutive time frames, and the SAT instance is extended with constraints to force that the states in all time frames are unique (i.e., the execution path is loop-free). The size of SAT instances required to achieve completeness of k-induction is polynomial in the recurrence diameter [Biere09]. As the recurrence diameter may be as large as the size of the state space (which is the case e.g. for a memory element), k-induction is often ineffective for practical verification tasks.

An extension of the inductive verification technique for the class of *interval properties* is presented in [Nguyen08, Nguyen11]. Interval properties form a subset of LTL that most often occurs in practical formal specifications. Such properties have the form $\boldsymbol{G}\ (a \Rightarrow c)$, where the *antecedent* a and *consequent* c are LTL formulas which may include nested *next* operators ($\boldsymbol{X}$) but no other temporal operators ($\boldsymbol{G}$, $\boldsymbol{F}$, and $\boldsymbol{U}$) are allowed. This subset of LTL is subject to *Interval Property Checking* (IPC) which is similar to bounded model checking with two exceptions:

- The transition relation is unrolled for the length of the interval property, i.e., for the maximal number of nested $\boldsymbol{X}$ operators in a and c.
- The initial state is constrained only by the antecedent a, i.e., the set of reachable states is overapproximated by a.

To avoid spurious counterexamples due to the overapproximation of reachable states, the antecedent a is strengthened with *reachability invariants* that are generated in an automated or manual way [Nguyen08].

Completeness by Interpolation

Interval property checking overapproximates the set of reachable states implicitly with the property's antecedent, while k-induction implicitly tightens the set of reachable states by unrolling the transition relation. In contrast, *interpolation*-based techniques calculate an overapproximation of the set of reachable states explicitly and reduce it iteratively.

The first interpolation-based model checking method was proposed by McMillan in [McMillan03a]. The core of the method is similar to bounded model checking: A SAT-solver is used to check if a property holds within a certain bound. If it does hold, i.e., if the SAT instance is unsatisfiable, a *proof of unsatisfiability* is derived from the SAT instance. This proof is used to overapproximate the set of reachable states using Craig interpolation [Craig57]. Intuitively, a property of the form $\boldsymbol{G}\ p$ holds if and only if the following SAT instance is unsatisfiable:

$$\Omega_R(V_i) \wedge \Omega_T(V_i, V_j) \wedge \neg p(V_j), \tag{3.4}$$

where Ω_R is the characteristic function of the overapproximation of reachable states. The algorithm iteratively checks if the property holds within an increasing bound: In each iteration, a tighter overapproximation of reachable states is calculated. This technique is complete for LTL properties of the form $\boldsymbol{G}\ p$ and the size of SAT instances is linear in the model diameter [McMillan03a]. Interpolation-based techniques are currently one of the most efficient model checking approaches [Biere09].

3.3. Model Abstractions

Model abstraction is a widely applied technique for verification of large and complex hardware designs. Abstraction-based verification strives to remove all model constraints that are irrelevant w.r.t. the verified property. Model abstractions for hardware verification are formalized in [Melham87, Giunchiglia92].

Melham in [Melham87] distinguishes structural, behavioral, data, and temporal abstractions. Structural abstractions simplify the structure of a concrete model but preserve its full behavioral characteristics. For instance, a gate-level representation of a combinational circuit can be modeled as an abstract logic unit that performs the same function. Behavioral abstractions neglect irrelevant behavioral characteristics of the concrete model. Data abstractions map the data types of a concrete model (e.g. real-valued analog signals) to more abstract types (e.g. Boolean variables). In this thesis, these three abstraction types are collectively referred to as *structural abstractions*. In contrast, *temporal abstractions* reduce the temporal granularity of the model, e.g. by using a cycle-accurate timing model for combinational logic instead of a more accurate gate-delay model.

Structural abstractions are often realized as state space abstractions. A state space abstraction consists in a mapping of states of the concrete model to compound states in the abstract model. Clarke et al. in [Clarke94] show a method for automatic generation of state space abstractions and describe a technique for symbolic execution over the abstract state space. Alternatively, the concrete state space can be mapped to an abstract state space using property preserving transformations [Loiseaux95]. Bruns et al. in [Bruns99] formalize incomplete state spaces using partial Kripke structures with 3-valued atomic propositions, where the third value is used to express uncertainty whether the proposition holds in a given state or not. This work also defines a completeness preorder on partial Kripke structures and develops a model checking algorithm for them.

Many techniques for abstraction-based formal verification start with a coarse abstraction which is sound but not necessarily complete, and refine it iteratively in an automated way until it is strong enough to prove or refute a given property. An early technique for such *automatic abstraction refinement* was based on replacing complex predicates with auxiliary Boolean variables [Saïdi99]. Clarke et al. in [Clarke03] propose a *counterexample-guided abstraction refinement*: They simulate each counterexample obtained with an abstract model in the concrete model to verify its consistency. If a counterexample is spurious (inconsistent with the concrete model), the abstract model is refined by splitting compound (abstract) states that caused the inconsistency. An alternative solution called *proof-based abstraction* is proposed by McMillan and Amla in [McMillan03b]. In this technique, model abstractions are constructed from proofs of unsatisfiability derived by a SAT solver.

The techniques discussed above are generally structural abstractions, as they do not explicitly reduce the temporal granularity of the concrete model. The majority of temporal abstractions found in the literature simplify the timing of combinational logic, for instance, by reducing gate-level timing to cycle-accurate timing, as in [Jain95]. The cycle-accurate timing is often abstracted further, e.g. for the verification of high-level models or communication protocols, as in [Urdahl12] for efficient handling of component interactions in complex SoC designs. An interesting example of a dedicated temporal abstraction for microprocessor verification is developed in [Windley95] within a theorem proving system: The cycle-accurate timing at microarchitectural level is reduced to instruction-accurate timing at architectural level. For scan infrastructures, the first domain-specific temporal abstraction is developed in [Baranowski12] and discussed in Chapter 4.

3.4. Access Scheduling

In scan networks, *access pattern generation* or *access scheduling* is the process of calculating a scan-in sequence (scan data) that implements an access to a specified *target* scan register (instrument) by writing and/or reading its content. The usual objective is to optimize the access time, i.e., minimize the total number of scan operations including *capture*, *shift*, and *update* cycles (see Section 1.3, p. 14). As scan infrastructure has been traditionally used for test pattern delivery and transfer of test responses, the majority of access generation algorithms targets optimal test scheduling. This section reviews state-of-the-art techniques for test scheduling, followed with a discussion of the first attempts at general purpose access generation for reconfigurable scan networks.

In scan networks compliant with IEEE Std. 1149.1 and 1500, the generation of scan sequences is straightforward: To connect the required data register (DR) to the scan chain, the instruction register (IR in IEEE 1149.1 or WIR in IEEE 1500) is loaded with a predefined instruction word [JTA01, Zorian05]. The access scheduling for concurrent testing of system components (cores) poses a challenge only due to resource conflicts and power constraints. Various access scheduling methods that target test time minimization under resource and power constraints have been proposed over the last two decades. Chou et al. in [Chou97] model structural dependencies with resource graphs and map the problem of test time optimization to covering table minimization.

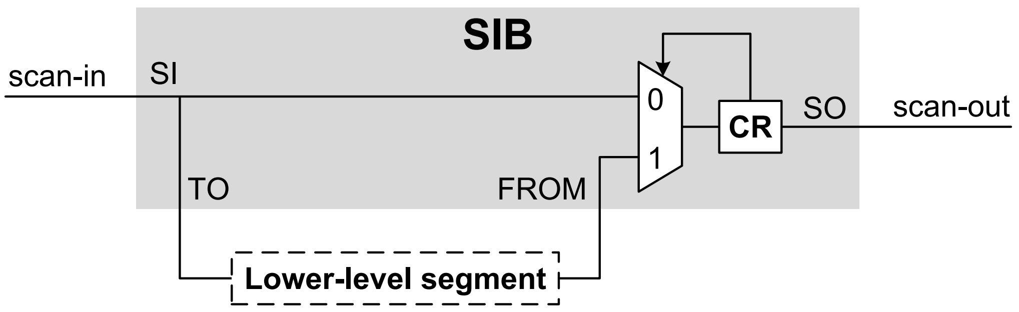


Figure 3.2.: Segment Insertion Bit (SIB)

Other researchers find the optimal test schedule using techniques based on mixed-integer linear programming [Chakrabarty00], simulated annealing [Zou03], genetic algorithms [Chattopadhyay03], or rectangle packing algorithms [Iyengar03], among others. More recent methods are based on co-optimization of the access schedule with the scan network architecture [Koranne03, Larsson06, Ghani Zadegan11a].

Recently, the access to on-chip instruments in reconfigurable scan networks gained attention. Zadegan et al. in [Ghani Zadegan11b] presents an algorithm inspired by Huffman encoding for optimal construction of RSNs. This method results in optimal access time under the assumption that the access frequencies to individual instruments are constant and known at design time. A method for calculating the average access time for concurrent and sequential schedules is given in [Ghani Zadegan12a, Larsson12].

The work in [Ghani Zadegan11b, Ghani Zadegan12a, Larsson12] is limited to regular, hierarchical RSNs constructed using Segment Insertion Bits (SIB). An SIB is composed of a 2-input multiplexer and a 1-bit configuration register (CR), as shown in Figure 3.2. The address of the multiplexer is driven by the shadow register of CR. When the content of CR is 1, the lower-level scan segment (e.g. a chain of instruments and other SIBs) is attached to the scan path (the SIB is *open*). Otherwise, when CR is set to 0, the scan path comprises only the 1-bit configuration register while the lower-level segment is bypassed (SIB is *closed*).

Figure 3.3 shows an exemplary SIB-based reconfigurable scan network with a 3-level hierarchy. For such regular RSNs, access sequence generation is a straightforward task: The scan sequence required to access any scan register is found by examining the current state of SIBs. SIBs above the target scan register in the hierarchy must be opened (set to 1 by performing a CSU operation), and all remaining SIBs should be

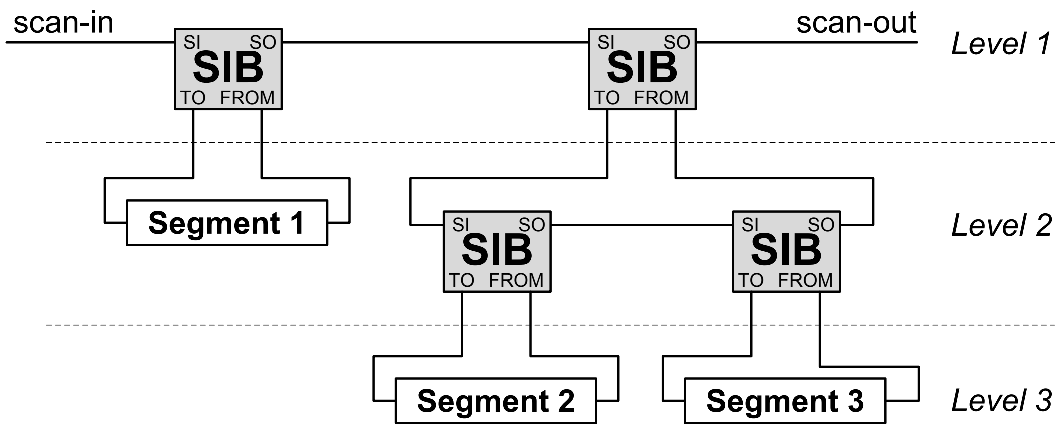


Figure 3.3.: An example of a SIB-based reconfigurable scan network

closed (set to 0) to reduce the access time. While this trivial access scheduling algorithm can provide minimal access time in SIB-based architectures, it cannot be applied to arbitrary RSNs, where the multiplexer control may be generated by arbitrary combinational logic blocks driven by multiple scan registers distributed over the network (cf. Figure 3.1, p. 37). The first algorithm that performs access time optimization and can handle more general RSNs is developed in [Baranowski13b] and discussed in Chapter 6.

3.5. Infrastructure Security

The accessibility of on-chip infrastructure contradicts security and safety requirements for chip internals [Tehranipoor11]. An attacker may exploit the scan infrastructure to gain access to protected data (secret key or IP), alter the system state to perform illegal operations, or conduct side-channel attacks, e.g. on cryptographic cores [Yang04]. In the following, state-of-the-art techniques for scan access authorization and scan sequence encryption are briefly introduced, followed with a discussion of access restriction methods.

Authentication and Authorization

The majority of existing techniques for securing scan infrastructure is based on authentication: The user (e.g. a tester or a service person) gains permission to access the scan network only after proving its identity, e.g. by presenting a *key*. The simplest authorization schemes assume a static key that is known only to entitled users. To gain access, the key must be either applied to dedicated primary inputs [Hely04], embedded at constant [Lee06,Agarwal11] or variable [Dworak13] positions in scan data (scan-in sequence), or written to a dedicated data register in a JTAG circuitry [Lee07,Pierce13] or an IEEE 1500 wrapper [Chiu12]. If a wrong key is used, the protected instruments are inaccessible [Dworak13], the scan-in and scan-out sequences are internally replaced with constant or pseudo-random values [Lee06,Agarwal11,Chiu12], the order of scan registers is dynamically changed in an unpredictable fashion [Hely04,Lee07], or the JTAG *update* operation is blocked [Pierce13] to disrupt any unauthorized access.

Stronger authorization schemes are based on challenge-response protocols [Buskey06, Clark10, Rosenfeld10, Park12, Das13, Pierce13]: The chip generates a random or pseudo-random *challenge* value and expects the user to provide the expected *response* value based on a *shared secret*. This shared secret is never transferred in plaintext (unencrypted) during the authorization process. The *response* is calculated from the *challenge* using various cryptographic algorithms, e.g. elliptic curve arithmetic [Buskey06, Das13] or hash functions [Clark10, Rosenfeld10]. More advanced scheme require mutual authentication based on three-entity protocols that require certification authorities and authentication servers [Park10,Park12,Das13].

Scan Sequence Encryption

To prevent that sensitive data are revealed by scan infrastructure, the scan sequences can be protected by encryption. On-chip stream ciphers are used to decrypt the scan-in sequence and encrypt the scan-out sequence at the JTAG TAP interface [Rosenfeld10]. This encryption scheme effectively prevents sniffing and spoofing of secret data at the TAP level. However, inside the chip, data are still shifted in plaintext. This can be exploited by an attacker to expose the unencrypted scan sequence using side-channel attacks, e.g. through mission logic [Rosenfeld10]. To prevent this type of attacks, the encryption circuitry can be distributed over the chip to locally decrypt scan inputs and encrypt scan outputs of individual scan network components [Rosenfeld11]. However, if many components require protection, this scheme becomes unwieldy and incurs high hardware overhead.

Access Restriction

The majority of approaches discussed so far assure that only authenticated users can access the scan infrastructure. However, if the authentication key or shared secret is leaked, full access becomes possible which is unacceptable in safety critical applications.

To prevent that sensitive data stored in scan registers be leaked via scan infrastructure, *mirror* registers can be used. Once sensitive data are written to a scan register, a mirror register replaces the original register in the scan network until the infrastructure is reset. This specialized approach is applied to *key registers* of cryptographic cores in [Yang06].

To avoid the need for authentication, the physical interface or parts of scan infrastructure are permanently deactivated using One Time Programmable (OTP) memory cells called *fuses* [Ebrard09]. By blowing an on-chip fuse, some instructions of a JTAG TAP controller or chosen scan chains can be permanently disabled [Sourgen92]. Most often, the fuses are blown after manufacturing test to prevent that scan chains are used for side-channel attacks on cryptographic cores or theft of intellectual property [Tehranipoor11]. Such fuse-based protection is widely adopted in microprocessors, e.g. in i.MX31 (Freescale) [Tehranipoor11] or MPS430 (Texas Instruments) [Clark10]. Alternatively, to guarantee inaccessibility of the entire scan infrastructure, the JTAG TAP can be completely removed after manufacturing test with a wafer saw [Kömmerling99]. This radical approach results in high security but makes the scan infrastructure completely unusable.

The existing techniques for access restriction enable only coarse-grained access management: The partial or full deactivation of scan infrastructure after manufacturing test is not acceptable in modern SoC designs, as the access to instrumentation must be provided throughout the lifetime of a chip (cf. Section 1.1, p. 10). Moreover, existing techniques require thorough consideration early in the design process and are therefore difficult or impossible to apply in core-based design flows. The first method for fine-grained access management of reconfigurable scan networks applicable to core-based designs is developed in [Baranowski13c] and discussed in Chapter 7.

3.6. Conclusions

Reconfigurable scan networks, as proposed by IEEE Std. 1149.1-2013 and the upcoming IEEE Std. P1687, emerge as an effective means to access the instrumentation of complex SoCs. The high performance and flexibility offered by RSNs, however, comes at a price: To assure high dependability and rapid development, novel methods are required to deal with the numerous challenges posed by RSNs.

The existing methods for the verification of regular scan infrastructures cannot be directly applied to arbitrary RSNs. Meanwhile, general-purpose formal verification methods such as model checking face scalability issues in deeply sequential circuits such as RSNs. To improve the scalability of existing verification techniques, a novel RSN modeling method based on temporal abstraction is developed in Chapter 4 and applied to model checking in Chapter 5.

While specialized algorithms exist for access scheduling in bypass-based scan networks, these methods are restricted to regular architectures with simple access dependencies. Irregular scan architectures with complex control signals require novel algorithms for the generation of access patterns and access time optimization. This goal is approached with a method based on pseudo-Boolean SAT solving developed in Chapter 6 that leverages the formal RSN model from Chapter 4.

The existing techniques for securing on-chip scan infrastructures focus on protecting individual scan chains or the entire access port. While such techniques can be directly applied to secure RSNs as a whole, restricting the access to individual instruments is costly due to high area overhead and may be impossible in core-based designs. A novel method for fine-grained access management in RSNs is developed in Chapter 7.

4. Scan Network Modeling

This chapter defines the terminology and structure of reconfigurable scan networks and introduces a novel RSN modeling method. The functional behavior of RSNs is explained using a cycle-accurate representation at Register-Transfer Level (RTL). Next, a method for constructing a temporal RSN abstraction is presented, followed with a discussion of its applications and limitations. The temporal abstraction defined in this chapter is the basis of formal verification methods in Chapter 5 and access pattern generation in Chapter 6 and 7.

4.1. Specification Languages

Structural and functional models of scan infrastructure serve various purposes in the chip development process. Scan network models are used:

- in verification and validation,
- during synthesis—to generate the actual (low-level) hardware implementation of the scan network using automated synthesis tools,
- in the integration step—to merge multiple scan networks into a global, chip-level network and connect them with system components and physical pins,
- for test and maintenance—to generate access patterns, i.e. scan-in sequences that implement the access to target scan chains or instruments.

Scan networks are usually described using dedicated, high-level specification languages which are standardized to enable design reuse and assure compatibility with commercial design automation tools. The following standards define common description languages for scan infrastructure:

- IEEE Std. 1149.1 [JTA01] defines a simple scan architecture and the Boundary

Scan Description Language (BSDL) for describing its implementation details. A specification in BSDL provides the count and lengths of data registers, the length of the instruction register, the encoding of instructions, as well as the type and mapping of boundary scan cells.

- IEEE Std. 1500 [SEC05] establishes a wrapper-based scan architecture and defines the Core Test Language (CTL). CTL is used to specify, among other parameters, the connectivity of wrappers, the number of scan chains, and user defined instructions.
- IEEE Std. P1687 [Stollon11, Ghani Zadegan12b] allows flexible, user-defined scan architectures and proposes an Instrument Connectivity Language (ICL) to describe them.

Languages such as BSDL and CTL are dedicated for specific scan architectures defined in their respective standards. In contrast, ICL is an expressive language that facilitates the design of almost arbitrary, user-defined scan networks. An ICL design is a netlist of scan registers, data latches, multiplexers, and combinational logic blocks.

A scan network specification in BSDL, CTL, or ICL is translated in a straightforward way to a structural, cycle-accurate hardware model. For the sake of generality, the following section describes RSNs as usual hardware structures at Register-Transfer Level (RTL). The scan architectures considered in this thesis and defined in the following section comprise a superset of RSNs allowed by IEEE 1149.1 and IEEE P1687: In addition to excludable and selectable scan registers defined in 1149.1-2013 [JTA13], arbitrary signals generated internally to the scan network are allowed to control the capture, shift, and update operations of individual scan registers. The presented definition of RSNs is also a superset of structures defined in P1687. A recent revision of this standard proposal enforces structural constraints which ensure that, for instance, scan multiplexers cannot disconnect their controlling registers from the scan chain. In contrast, the RSN model presented below imposes no such structural constraints: Control signals can be generated by combinational logic blocks that take their inputs from arbitrary registers distributed over the RSN and its primary inputs.

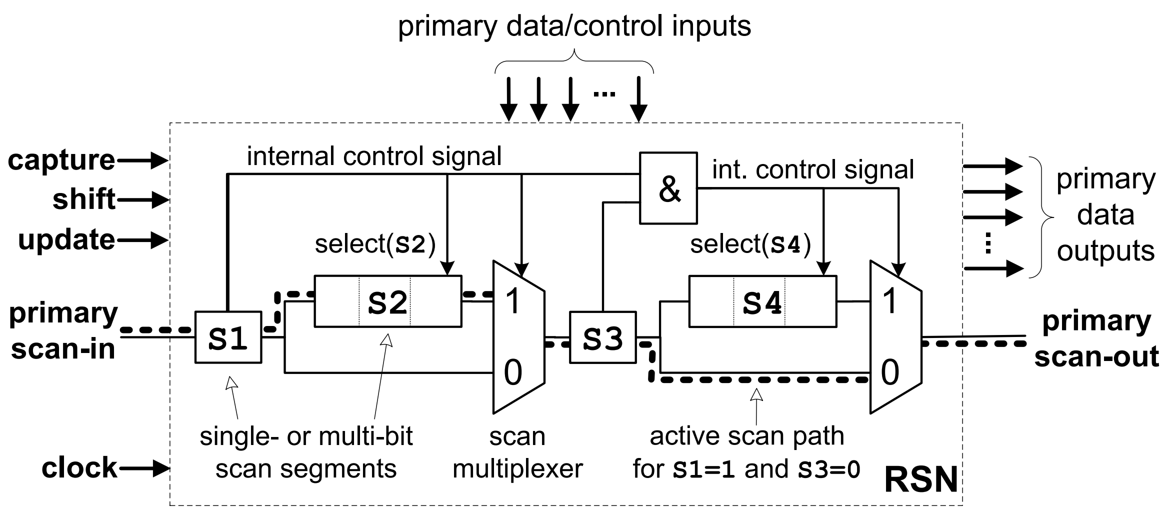


Figure 4.1.: Example of a reconfigurable scan network and its terminology

4.2. Structural Modeling

Reconfigurable scan networks are sequential circuits composed of scan registers, latches, multiplexers, and combinational logic. An RSN has a global *clock* input port, a *primary scan-input* and *-output*, as well as three *global control* inputs that activate the three *scan operations*: *capture*, *shift*, and *update*, as defined by IEEE 1149.1 [JTA01]. Optionally, an RSN may have *primary data* input and output ports for communication with instrumentation, as well as *primary control* input ports for network configuration. Figure 4.1 presents an RSN example and explains the basic terminology.

Scan Segments and Scan Paths

The basic building block of an RSN is a *scan segment* with a *scan-in* and a *scan-out* port. Scan segments are used to communicate with on-chip instrumentation or drive internal control signals. A scan segment is essentially a *shift register* (scan chain) composed of one or more *scan cells* sharing a set of control signals. A scan segment is optionally equipped with a *shadow register* that is loaded in parallel from the shift register. Figure 4.2 presents a block diagram of a scan segment with optional elements marked by a dashed line.

A scan segment supports up to three scan operations which are activated by the *global control* signals—*capture*, *shift*, and *update* (cf. Figure 4.1 and 4.2):

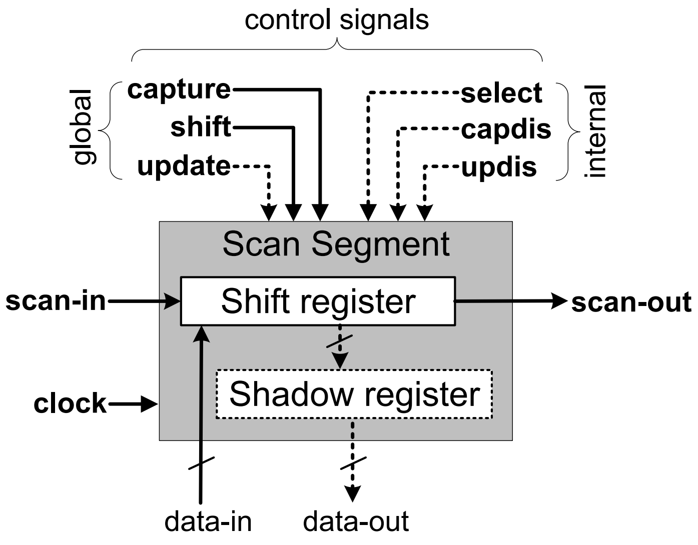


Figure 4.2.: Scan segment block diagram

- During a *capture* operation, the shift register is loaded with data from the *data-in* port.
- During a *shift* operation, data are shifted from the segment's scan-input, through its register bits, down to the scan-output of the segment.
- During an *update* operation, the optional shadow register is loaded with data from the shift register. Note that the shadow register is stable during the shift operation.

To inhibit the scan operations, a scan segment may possess up to three optional control ports:

- Select port (*select*) specifies if the scan segment is enabled for capture, shift, and update operation.
- Capture disable port (*capdis*) invalidates the capture operation on the scan segment, regardless of the *select* port state.
- Update disable port (*updis*) invalidates the update operation, regardless of the *select* port state.

The functionality of the *capdis* and *updis* ports of a scan segment may be implemented by gating the global control signals *capture* and *update*, respectively, at the segment's boundary. The *select* port may be implemented by local clock gating.

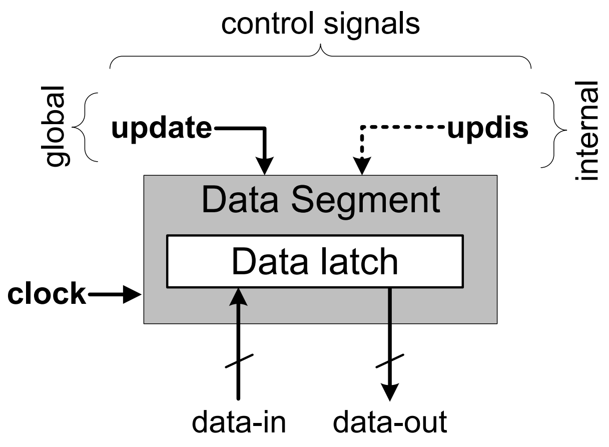


Figure 4.3.: Data segment block diagram

Scan segments are chained via scan-out and scan-in ports. A *scan path* is a non-circular sequence of scan segments starting at a primary scan-input and ending at a primary scan-output. The consecutive scan segments forming a scan path are connected either directly, via buffers or inverters, or through *scan multiplexers*. A scan multiplexer controls the path through which data are shifted in an RSN. For instance, the two scan multiplexers in Figure 4.1 allow to bypass scan segments S2 and S4. The control signal of a scan multiplexer is called *address* and specifies the selected scan input.

Data Segments

Apart from scan segments, an RSN may contain *data segments* for auxiliary data storage, communication with on-chip instruments, and generation of internal control signals. A data segment consists of a single- or multi-bit *data latch* which loads data from the *data-in* port, as shown in Figure 4.3. The data latch is transparent during an *update* operation unless the optional control signal *updis* is active.

Control and Data Signals

The control ports of scan segments (*select*, *capdis*, *updis*), data segments (*updis*), and multiplexers (*address*) are driven by signals that are collectively referred to as *internal control signals*. Internal control signals can be driven by arbitrary combinational logic blocks that take their inputs from any shadow registers (cf. Figure 4.2) and data latches (cf. Figure 4.3) distributed over an RSN, as well as from any primary control inputs. For instance, the *select* port of scan segment S2 in Figure 4.1 is driven directly by the shadow register of S1, while the *select* of S4 is generated by a logic gate driven by the shadow registers of S1 and S3. All internal control signals must be stable whenever the global *update* signal is active and the *clock* signal is low. This requirement is satisfied using latches.

The *data-in* ports of both scan and data segments may also be driven by arbitrary combinational logic blocks driven by shadow registers, data latches, and primary data inputs. The *data-out* ports of scan and data segments may drive the *primary data outputs* of an RSN, either directly or through arbitrary combinational logic.

4.3. Scan Network Operation

Scan data are shifted in an RSN from the primary scan-input, through an *active* scan path, down to the primary scan-output. The flow of the active scan path depends on the logic state of the RSN itself: The *select* signals of all scan segments on the active scan path are asserted, and all on-path multiplexers select the on-path inputs. For instance, in Figure 4.1 (p. 53), if $\texttt{S1} = 1$ and $\texttt{S3} = 0$, the active scan path goes through S1, S2, and S3, while S4 is bypassed.

A *scan configuration* of an RSN is the logic state of its sequential elements and primary data/control inputs. The scan configuration determines which scan segments in the network are currently accessible. A scan configuration is *valid* if and only if: (i) an active scan path exists and (ii) scan segments that do not belong to the active scan path are deselected. This ensures that the scan data are delivered to the target scan segments, the captured data are shifted towards the primary scan-output, and all scan segments that do not take part in the access (i.e., do not belong to the active scan path) are stable.

The operation of an RSN is synchronized with the global clock signal. The basic access to the scan network is an atomic (inseparable) operation that consists of three phases: *Capture*, *Shift*, and *Update* (CSU). Each phase is activated by its respective global control signal, as shown in Figure 4.4. During the *capture* phase at the rising clock edge, the scan segments on the active scan path are loaded with data from their *data-in* ports. This data are shifted out of the network during the *shift* phase at each

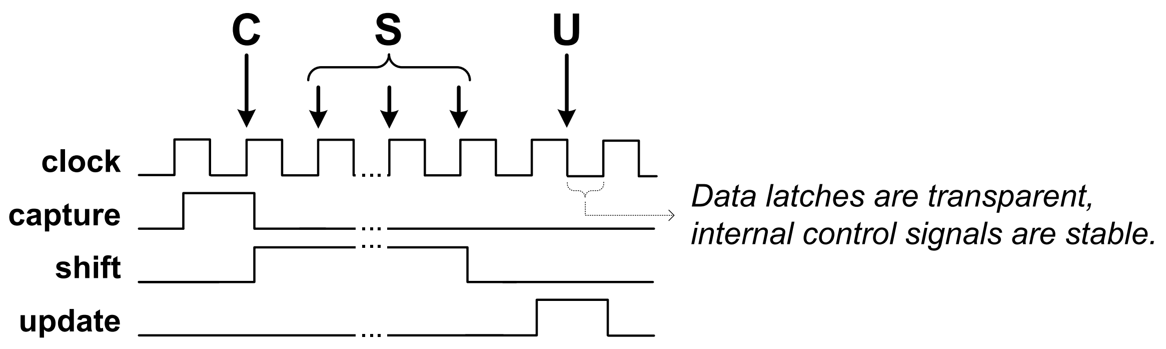

C
S
U
clock
capture
shift
update
Data latches are transparent, internal control signals are stable.

Figure 4.4.: Capture, Shift, Update (CSU) operation

rising clock edge, while new data are shifted in. Finally, during the *update* phase at the falling clock edge, the shifted-in data are latched in the shadow registers of scan segments on the active scan path. While the *clock* signal is low and the *update* signal is asserted, the latches of enabled data segments are transparent and all internal control signals are stable. Note that the *capture* and *update* phases of a CSU operation require a constant number of clock cycles to complete. In contrast, the *shift* phase may take any number of cycles (zero or more) and usually lasts as long as is necessary to shift through the full active scan path.

A read or write access to a scan register in the network requires that the accessed register is part of an active scan path (cf. Figure 4.1, p. 53). A *scan access* is a sequence of CSU operations required to reconfigure the scan network and access the target registers. *Access time* is the number of clock cycles that are required to perform the scan access, including the update and capture cycles of each CSU.

4.4. Temporal Abstraction

A cycle-accurate behavioral representation of a reconfigurable scan network is easily derived from a structural RSN model. A scan network can be modeled as an FSM, as shown in Figure 4.5: On the input side, the FSM has a single *scan-input*, three *global control inputs* (*capture*, *shift*, and *update*), and optional *primary data/control inputs*. On the output side, the FSM has a single *scan-output* and optional *primary data outputs*. The primary data inputs and outputs are used for communication with on-chip instrumentation which is not part of the RSN itself. The primary data outputs may expose the state of shadow registers of scan segments, latches of data segments,

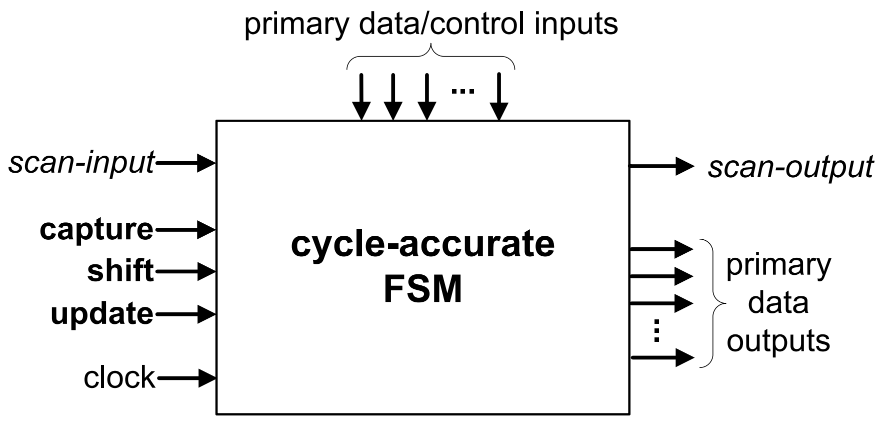


Figure 4.5.: Cycle-accurate FSM representation of a reconfigurable scan network

and any internal data or control signals.

The operation of an FSM representing a reconfigurable scan networks is constrained in the following way:

1. The three global control signals—*capture*, *shift*, and *update*—always follow the pattern of a CSU operation, as defined by the IEEE Std. 1149.1 and shown in Figure 4.4 (p. 57): Initially, *capture* is active for exactly one clock cycle. Next, *shift* is active for zero or more cycles, followed with an *update* operation of exactly one cycle.
2. The shadow registers and data latches internal to the RSN may load new data only during the *update* phase. In consequence, since the primary data outputs of the RSN are generated by combinational logic blocks driven only by those two types of sequential elements, their state is stable in the *capture* and *shift* phase.

As the global control inputs are constrained to a predefined pattern and the primary data outputs may change only when the *update* signal is active, the FSM model can be simplified by applying the following temporal abstraction: Instead of modeling the effect of each *capture*, *shift*, and *update* cycle individually, the full CSU operation is treated as an atomic operation that changes the state of shadow registers and data latches. This modeling technique is called *CSU-accurate* abstraction, and the resulting model is a *CSU-Accurate Model* (CAM).

Intuitively, a CSU-accurate model can be viewed as an FSM with an *abstract clock*. One state transition (cycle of the abstract clock) in the CSU-accurate FSM corresponds to

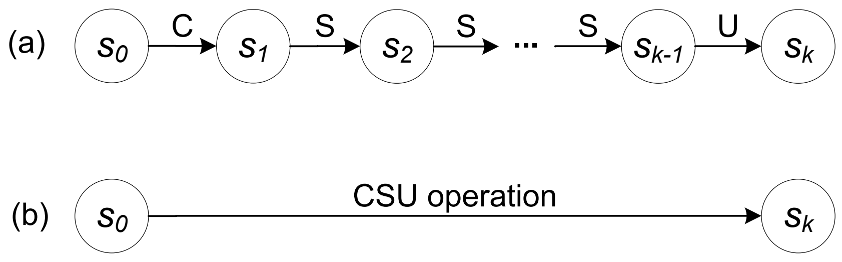


Figure 4.6.: State transitions during one CSU operation in (a) a cycle-accurate RSN model and (b) a CSU-accurate model (CAM)

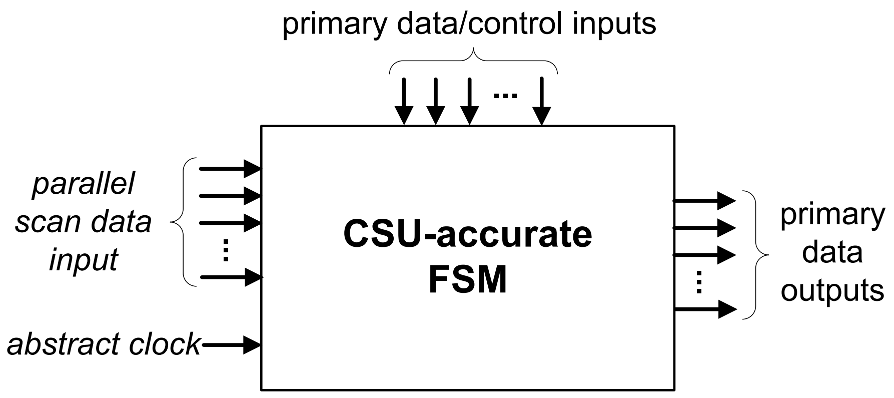


Figure 4.7.: CSU-accurate FSM representation of a reconfigurable scan network

a full CSU operation, i.e., multiple clock cycles in the cycle-accurate RSN model. An example is given in Figure 4.6 where k cycles of a CSU operation are combined into a single transition in the CAM.

The block diagram of a CSU-accurate FSM is depicted in Figure 4.7. Compared with the cycle-accurate FSM model from Figure 4.5, the CSU-accurate FSM has no inputs that control the three phases of a CSU operation, and neither a *scan-input* nor a *scan-output*. Instead of representing the scan data as a sequence of bits at the *scan-input*, the scan data of a CSU operation are modeled as a bit vector. Scan data for each scan segment are provided to the CSU-accurate FSM via the *parallel scan data* input. In each state transition (cycle of the abstract clock that models a CSU operation), the parallel scan data are transferred to the corresponding scan segment if and only if this scan segment is selected and belongs to the active scan path.

4.4.1. CSU-Accurate Model

In the following, the CSU-accurate model is defined formally. The state of sequential elements and control signals is modeled in 3-valued logic with three symbols $\{0, 1, X\}$ that represent logic value 0, logic value 1, and an unknown value, respectively. The unknown value (X) is used to model partially specified initial scan configurations (the state of uninitialized registers), and the high-impedance state of tri-state logic gates. The interpretation of logic operators over 3-valued variables follows Kleene's strongest regular 3-valued logic [Kleene50].

Definition 11. (CSU-Accurate Model, CAM) The CSU-accurate model of an RSN is a tuple $\mathcal{M} = \{S, H, D, I, V, C, c_0, \texttt{Select}, \texttt{Updis}, \texttt{Capdis}, \texttt{DataIn}, \texttt{Active}\}$ that consists of:

- S: the set of scan segments in the RSN.
- H: the set of 1-bit shadow registers that form scan segments in S. The correspondence between scan segments and shadow registers is captured by a surjective function $\mathcal{S} : H \to S$ that maps each shadow register $h \in H$ to its corresponding scan segment $s \in S$ denoted as $\mathcal{S}(h)$.
- D: the set of 1-bit data latches that form data segments in the RSN.
- I: the set of primary data/control inputs of the RSN.
- V: the set of 3-valued variables corresponding to the elements from $H \cup D \cup I$ with a mapping defined by a bijective function $\mathcal{V} : H \cup D \cup I \to V$.
- $C := \{0, 1, X\}^{|H \cup D \cup I|}$: the set of scan configurations. Each scan configuration $c \in C$ defines the state of shadow registers, data latches, and primary data/control inputs. Each scan configuration $c \in C$ is a *valuation* of variables in V which is also treated as a function $c : H \cup D \cup I \to \{0, 1, X\}$ that assigns each element $e \in H \cup D \cup I$ a 3-valued state denoted as $c(e)$.
- $c_0 \in C$: the initial scan configuration (reset state).
- $\texttt{Select} : C \times S \to \{0, 1, X\}$: the function that defines the state of the *select* control port of each scan segment $s \in S$ in each scan configuration $c \in C$, denoted as $\texttt{Select}(c, s)$.
- $\texttt{Updis} : C \times (S \cup D) \to \{0, 1, X\}$: the function that defines the state of the

updis control port of each element $e \in (S \cup D)$ in each scan configuration $c \in C$, denoted as `Updis`(c, e).

- `Capdis` $: C \times S \rightarrow \{0, 1, X\}$: the function that defines the state of the *capdis* control port of each scan segment $s \in S$ in each scan configuration $c \in C$, denoted as `Capdis`(c, s).
- `DataIn` $: C \times D \rightarrow \{0, 1, X\}$: the function that defines the state of the *data-in* port of each data latch $d \in D$ in each scan configuration $c \in C$, denoted as `DataIn`(c, d).
- `Active` $: C \times S \rightarrow \{0, 1, X\}$: the function that determines the active scan path. For each scan segment $s \in S$ and each scan configuration $c \in C$ this function is defined as follows:

$$\texttt{Active}(c, s) := \begin{cases} 0 & \text{if } s \text{ does not belong to the active scan path in } c, \\ 1 & \text{if } s \text{ belongs to the active scan path in } c, \\ X & \text{if it is not known whether } s \text{ is on the active scan path in } c. \end{cases}$$

A CSU-accurate model can be easily derived from any structural description of an RSN: either from a gate- or RT-level netlist, or from a high-level representation, e.g. in Instrument Connectivity Language (ICL) defined by IEEE P1687:

- The sets S, H, D, I are found by inspecting the netlist components and ports.
- The initial scan configuration c_0 is defined by the reset state of the RSN. The state of uninitialized sequential elements is assumed unknown (X) and the state of primary data/control inputs is unconstrained in c_0.
- The functions `Select`, `Updis`, `Capdis`, and `DataIn` are obtained by traversing the input cones of the corresponding control/data ports of scan segments and data segments in the netlist.

The followings two sections describe the construction of the `Active` function and define the transition relation of the CSU-accurate model.

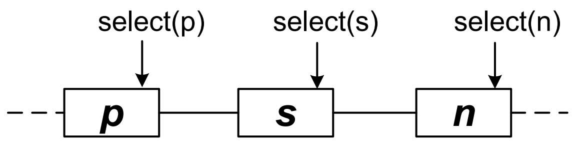


Figure 4.8.: Chained scan structure

4.4.2. Valid Scan Configurations

The function `Active` determines if a scan segment belongs to the active scan path and is constructed as follows:

$$\texttt{Active}(c, s) := \begin{cases} 0 & \text{if } \texttt{Select}(c, s) = 0, \\ 1 & \text{if } (\texttt{Select}(c, s) = 1) \wedge \texttt{Valid}(c), \\ X & \text{otherwise.} \end{cases} \tag{4.1}$$

where $s \in S$, $c \in C$, and $\texttt{Valid} : C \to \mathbb{B}$ is a *validity predicate* that evaluates to 1 if and only if a scan configuration is valid, i.e., when there exists a well formed scan path and all off-path scan segments are deselected (cf. Section 4.3, p. 56). The validity predicate is constructed piecewise as a conjunction of the form:

$$\texttt{Valid}(c) = \bigwedge_{s \in S} v(c, s), \tag{4.2}$$

where $v : C \times S \to \mathbb{B}$ is a *local validity predicate* that evaluates to *true* if and only if the local scan configuration of a scan segment is valid, as explained below.

Validity of Chained Scan Structures

Given a scan segment $s \in S$, let $\texttt{pred}(s)$ and $\texttt{succ}(s)$ denote the set of its predecessor and successor scan segments, respectively, connected either directly, through buffers or inverters, or via scan multiplexers. For a scan segment s with a single predecessor $p \in \texttt{pred}(s)$ and a single successor $n \in \texttt{succ}(s)$ (cf. Figure 4.8), it is required that both p and n be selected if s is selected, such that scan data are not lost. Thus:

$$v(c, s) := (\texttt{Select}(c, s) = 1) \Rightarrow [(\texttt{Select}(c, p) = 1) \wedge (\texttt{Select}(c, n) = 1)] \tag{4.3}$$

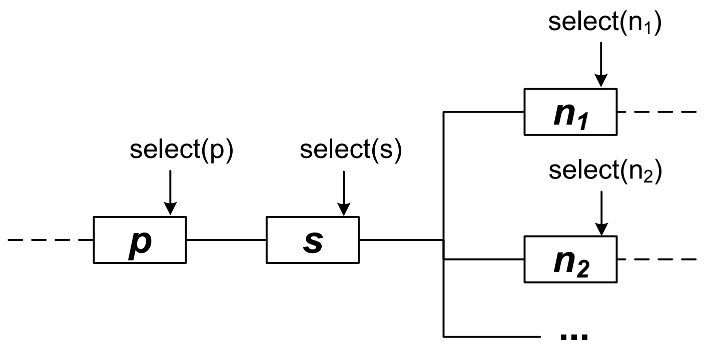


Figure 4.9.: Branching scan structure (fanout)

Validity of Branching Scan Structures

For a scan segment s with a single predecessor p and multiple successors (cf. Figure 4.9), a valid scan configuration requires that *exactly* one successor of s is selected if s is selected. Formula (4.4) specifies that *at least* one successor of s is selected if s is selected:

$$(\texttt{Select}(c, s) = 1) \Rightarrow \bigvee_{n \in \texttt{succ}(s)} (\texttt{Select}(c, n) = 1) \tag{4.4}$$

Formula (4.5) ensures that *at most* one successor of s is selected at any time:

$$\bigforall_{n_k, n_l \in \texttt{succ}(s), n_k \neq n_l} [(\texttt{Select}(c, n_k) = 1) \Rightarrow (\texttt{Select}(c, n_l) \neq 1)] \tag{4.5}$$

The following formula states the requirement for a valid scan configuration for a segment s with one predecessor p and multiple successors using (4.4) and (4.5):

$$v(c, s) := [(\texttt{Select}(c, s) = 1) \Rightarrow (\texttt{Select}(c, p) = 1)] \wedge (4.4) \wedge (4.5) \tag{4.6}$$

This assures that in case of a branching scan path (fanout>1) only one branch is active, i.e., there exists at most one selected successor.

Validity of Multiplexed Scan Structures

For a scan segment s with a single successor n and multiple predecessors selected by a multiplexer (cf. Figure 4.10), a valid scan configuration requires that if s is selected, then *exactly* one predecessor of s is selected and the addressing is correct. Formula

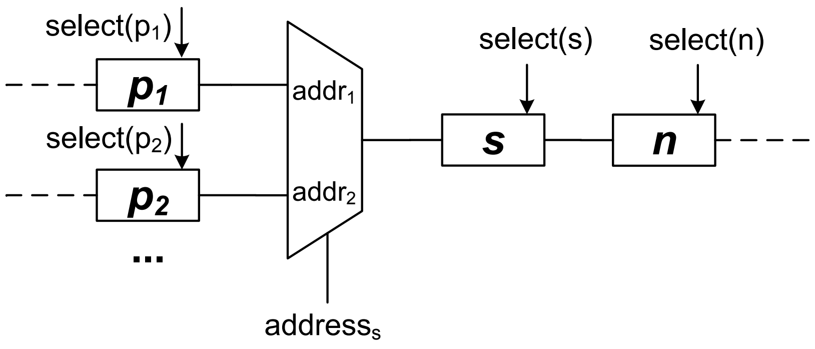


Figure 4.10.: Multiplexed scan structure (scan path convergence)

(4.7) below states that *at least* one predecessor must be selected if s is selected:

$$(\texttt{Select}(c,s)=1) \Rightarrow \bigvee_{p\in\texttt{pred}(s)} (\texttt{Select}(c,p)=1) \tag{4.7}$$

If a predecessor of s is selected, the *address* of the multiplexer must be correctly set:

$$\bigforall_{p\in\texttt{pred}(s)} [(\texttt{Select}(c,p)=1) \Rightarrow (\texttt{address}_s(c)=\texttt{addr}_p)] \tag{4.8}$$

where $\texttt{address}_s(c)$ denotes the state of the address port in scan configuration c, and $\texttt{addr}_p$ is a constant that denotes the address of the multiplexer input connected to segment p. Note that (4.8) implicitly assures that at most one predecessor of s is active. The requirement for a valid scan configuration of segment s with one successor n and multiple predecessors is given below using (4.7) and (4.8):

$$v(c,s) := [(\texttt{Select}(c,s)=1) \Rightarrow (\texttt{Select}(c,n)=1)] \wedge (4.7) \wedge (4.8) \tag{4.9}$$

This assures that in case of a multiplexed scan path, the active path is correctly routed.

In case of a node s with multiple predecessors and multiple successors, the following formula captures the condition for a valid scan configuration:

$$v(c,s) := (4.4) \wedge (4.5) \wedge (4.7) \wedge (4.8). \tag{4.10}$$

4.4.3. Transition Relation

CSU operations are considered atomic in a CSU-accurate model. A CSU operation loads data segments with new values from their respective *data-in* ports, and it may arbitrarily change the state of all scan segments on the active scan path since any data may be shifted into those segments from the primary scan-input. This behavior is captured with the transition relation, as defined below.

Definition 12. (CSU-accurate transition relation) The transition relation of a CAM $\mathcal{M} = \{S, H, D, I, V, C, c_0, \texttt{Select}, \texttt{Updis}, \texttt{Capdis}, \texttt{DataIn}, \texttt{Active}\}$ is defined as a set $T \subseteq C \times C$ that includes all pairs of scan configurations (c_1, c_2) such that $c_2 \in C$ can be reached from $c_1 \in C$ within one CSU operation. The characteristic function of the transition relation is defined as follows:

$$\begin{aligned} T(c_1, c_2) := &\bigwedge_{h \in H} \big[\texttt{Stable}(c_1, \mathcal{S}(h)) \Rightarrow \big(c_2(h) = c_1(h)\big)\big] \wedge \\ &\bigwedge_{h \in H} \big[\texttt{Unknown}(c_1, \mathcal{S}(h)) \Rightarrow \big(c_2(h) = X\big)\big] \wedge \\ &\bigwedge_{d \in D} \big[(\texttt{Updis}(c_1, d) = 0) \Rightarrow \big(c_2(d) = \texttt{DataIn}^*(c_1, c_2, d)\big)\big] \wedge \\ &\bigwedge_{d \in D} \big[(\texttt{Updis}(c_1, d) = 1) \Rightarrow \big(c_2(d) = c_1(d)\big)\big] \wedge \\ &\bigwedge_{d \in D} \big[(\texttt{Updis}(c_1, d) = X) \Rightarrow \big(c_2(d) = X\big)\big], \end{aligned} \tag{4.11}$$

where predicate $\texttt{Stable} : C \times S \to \mathbb{B}$ is defined as:

$$\texttt{Stable}(c, s) := \big[(\texttt{Active}(c, s) = 0) \vee (\texttt{Updis}(c, s) = 1)\big], \tag{4.12}$$

predicate $\texttt{Unknown} : C \times S \to \mathbb{B}$ is defined as:

$$\begin{aligned} \texttt{Unknown}(c, s) := &\big[(\texttt{Active}(c, s) = X) \wedge (\texttt{Updis}(c, s) \neq 1)\big] \vee \\ &\big[(\texttt{Active}(c, s) = 1) \wedge (\texttt{Updis}(c, s) = X)\big], \end{aligned} \tag{4.13}$$

and function $\texttt{DataIn}^*(c_i, c_j, d)$ is derived from $\texttt{DataIn}(c, d)$ by substituting variables in c with:

- variables from c_i for primary data/control inputs from I,
- variables from c_j for shadow registers from H and data latches from D.

The characteristic function of a transition relation defines the requirements for state changes: If a scan segment s in scan configuration c_1 does not belong to the active scan path or its *updis* signal is active, the state of all shadow registers of s must not differ in the consecutive scan configuration c_2. Additionally, if the scan configuration c_1 is invalid or the activation condition of s is unknown, the state of all shadow registers of s is assumed unknown in c_2. As a consequence, the state of a shadow register h may change freely only when $\mathcal{S}(h)$ is selected in a valid scan configuration c_1, i.e., when $\texttt{Active}(c_1, \mathcal{S}(h)) = 1$ and $\texttt{Updis}(c_1, \mathcal{S}(h)) = 0$.

The state of a data segment is constrained by the state of its corresponding *data-in* port. The latch of an enabled data segment is transparent at the end of the *update* phase—after scan segments on the active scan path have already registered new values, and when other enabled data segments are also transparent (cf. Figure 4.4, p. 57). Therefore, a data segment receives values from both the current scan configuration c_1 (from primary inputs) and the next scan configuration c_2 (from shadow registers and other data latches).

Remark 4. By construction, the transition relation of a CSU-accurate model includes a transition between two valid scan configurations if an only if this transition is possible in the cycle-accurate RSN model. Therefore, for transitions to *valid* scan configurations, the CSU-accurate abstraction is *exact*. However, for transitions to *invalid* scan configurations, this model pessimistically assumes unknown values (X) for all scan and data segments that potentially become enabled.

4.4.4. Implications of CSU-Accurate Modeling

The CSU-accurate modeling is based on the following assumptions:

1. A CSU operation is an atomic operation that consists of exactly one *capture* cycle at the beginning, optional *shift* cycles in between, and exactly one *update* cycle at the end.
2. The impact of individual scan operations is neglected and only state transitions caused by full CSU operations are considered. Therefore, the state of primary scan-inputs, primary scan-outputs, and shift registers is not modeled.
3. Primary data/control inputs of the RSN are stable throughout a CSU operation.

4. Internal control signals are driven by combinational logic blocks that may take their inputs only from external control inputs, data latches, and shadow registers distributed over the network. These signals are stable during the *capture* and *shift* phase, and they are also equipped with latches that assure their stability during the *update* phase (cf. Section 4.2, p. 55).
5. An active scan path may only pass through scan segments, buffers, inverters, and scan multiplexers.
6. In invalid scan configurations, the state of shadow registers in selected scan segments is assumed unknown (X).

Assumption 1 is justified in reconfigurable scan networks accessed through a JTAG TAP, since the IEEE 1149.1-compliant TAP controller requires that scan operations be performed in the defined order. This assumption can be relaxed and CSU-accurate modeling can be easily extended to support user-defined access mechanisms with any order of *capture*, *shift*, and *update* phases. However, such extensions are application specific and hence are beyond the scope of this thesis.

As the CAM does not explicitly represent individual *shift* cycles and only models state transitions caused by full CSU operations, it does not allow to reason about the state of shift registers, primary scan-inputs and primary scan-outputs (Assumption 2).

Assumption 3 states that the primary data/control inputs of the RSN are stable during a CSU as the temporal granularity of the CAM allows to model signal states only before and after a CSU operation. Assumption 4 defines which elements drive internal control signals and effectively states that these signals are also stable throughout a CSU. Without these two assumptions—for instance by allowing that internal control signals are driven by shift registers—the active scan path could dynamically change during the *shift* phase. This situation is best avoided, as such RSN designs are highly unpredictable and difficult to verify. Note that Assumption 4 is actually a requirement imposed on RSNs by IEEE P1687.

Assumption 5 states that the active scan path cannot pass through any logic components that could change the scan data, except for inverting it. Consequently, CSU-accurate modeling cannot be applied to generate access to test compression logic on the scan path. Such structures must be excluded from the RSN model or treated as black-boxes that should never belong to the active scan path. Note that IEEE P1687 allows scan paths composed of scan segments, scan multiplexers, and inverters only.

Due to Assumption 6, CSU-accurate modeling is pessimistic: Recall that in an invalid scan configuration $c \in C$, it holds that $\forall_{s \in S} \texttt{Active}(c, s) = X$. Therefore, according to the CAM transition relation (cf. Definition 12), the content of all potentially selected shadow registers is assumed undefined in invalid scan configurations although it may be well defined in the cycle-accurate model. For applications in access pattern generation, as discussed in Chapter 6, this pessimism is irrelevant since invalid scan configurations should be avoided to assure reliable access. For applications in formal verification, it may compromise abstraction completeness and lead to spurious counterexamples. Completeness of the CSU-accurate modeling is discussed in more detail in Section 5.3.

5. Formal Verification

In simple reconfigurable scan networks, the verification of properties such as accessibility may not be required if appropriate design rules are observed. For instance, the SIB-based RSNs proposed in [Ghani Zadegan11b] are very regular: They consist of hierarchically connected SIBs and scan segments, and their internal control signals are driven directly by 1-bit shadow registers (cf. Figure 3.3, p. 46). In this type of scan architectures, the accessibility of a scan chain—i.e., a single scan segment or a chain of scan segments and SIBs—requires that the following recursive condition is fulfilled:

1. The parent SIB of the scan chain (the SIB to which the chain is connected to) works correctly.
2. The parent SIB is properly connected to its higher level scan chain.
3. The higher level scan chain is also accessible, i.e., it fulfills conditions 1, 2 and 3.

While the first condition is easily checked by exhaustive simulation of the SIB, the second condition can be enforced with a simple structural design rule. In contrast, the verification of irregular RSNs with control signals driven by arbitrary combinational logic blocks poses a much more difficult problem. Due to the high sequential depth of RSNs, existing model checking algorithms are often ineffective in proving even simple properties such as accessibility, as is shown in Section 5.4 (p. 85).

This chapter discusses the applicability of the CSU-accurate model defined in Chapter 4 to formal verification of complex RSNs. Section 5.1 presents a CSU-accurate bounded model checking technique and its application to the verification of accessibility. Section 5.2 defines a class of *robust* RSNs, discusses their properties, and presents complete methods to verify them. The soundness and completeness of CSU-accurate modeling for both robust and non-robust RSNs is discussed in Section 5.3. Experimental results for verification of large RSN designs are summarized in Section 5.4 and presented in detail in Appendix B.

5.1. CSU-Accurate Bounded Model Checking

Bounded model checking (BMC) is a successful formal verification technique based on propositional decision procedures (SAT). The goal of BMC is to check whether a given temporal logic formula holds in an FSM for all initialized executions paths with a given (bounded) length. The basics of bounded model checking and state-of-the-art BMC methods are discussed in Section 3.2.1. These methods are directly applied to the verification of RSNs represented with CSU-accurate models, as explained below.

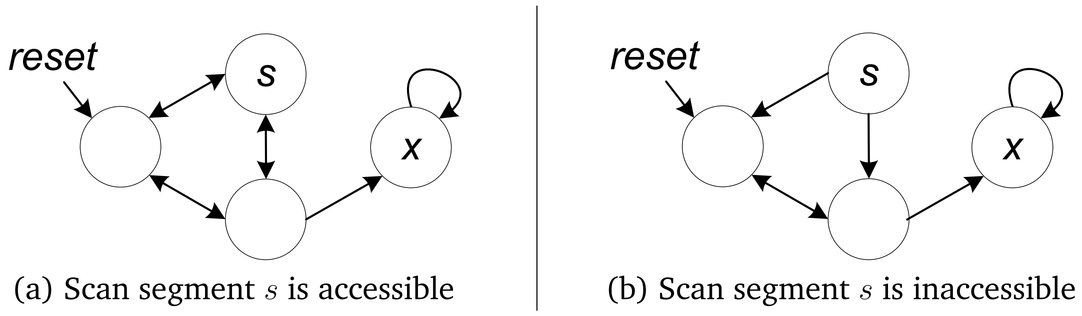
Bounded model checking is mapped to a satisfiability problem by unrolling the model's transition relation (see Section 3.2.1, p. 38). Let $\mathcal{M} = \{S, H, D, I, V, C, c_0, \texttt{Select}, \texttt{Updis}, \texttt{Capdis}, \texttt{DataIn}, \texttt{Active}\}$ be the CSU-accurate model of an RSN, and let T be the transition relation of $\mathcal{M}$. Given an LTL property P defined over variables in V, a SAT instance is composed by unrolling the transition relation T and the property P. Each unrolled instance of T corresponds to a CSU operation, and each time step corresponds to a scan configuration. If the SAT instance is satisfiable, the satisfying assignment constitutes a counterexample to P and provides the valuation of variables in V for each time step. The scan data that cause the violation of P can be easily derived from the counterexample (see Section 6.2, p. 94).

For instance, for a simple LTL property of the form $\boldsymbol{G}\ p$, where p is a Boolean function defined over V, the SAT instance is formed as follows:

$$\varphi(k) := \Omega_I(V_0) \wedge \left[\bigwedge_{n=0}^{k-1} \Omega_T(V_n, V_{n+1})\right] \wedge \left[\bigvee_{n=0}^{k} \neg p(V_n)\right], \tag{5.1}$$

where V_i is the set of variables in the i-th time step, such that each element in V_i corresponds to exactly one element in V ($\forall_{0 \le i \le k}\ |V_i| = |V|$), while Ω_I and Ω_T are the characteristic functions of the initial scan configuration c_0 and the transition relation T, respectively. $\varphi(k)$ is satisfiable if and only if there exists a counterexample to the property $\boldsymbol{G}\ p$ with k or less CSU operations.

In the following, the application of bounded model checking to proving RSN accessibility is shown. For the details on BMC for general LTL properties please refer to [Biere03].



(a) Scan segment s is accessible

(b) Scan segment s is inaccessible

Figure 5.1.: Examples of CSU-accurate state diagrams (scan segment s is accessible in scan configurations annotated with s)

5.1.1. Application: Accessibility Proof

To assure that a scan segment is accessible, it is necessary to prove that it is observable and controllable. A necessary requirement is that there exists a scan path that goes from the primary scan-input, through the segment, down to the primary scan-output of the network. To determine if such a scan path exists, a static connectivity check can be used [Remmers04]. For complex scan architectures with arbitrary control signals, the necessary and sufficient requirement is a justification of control signals over one or multiple CSU operations to put the target scan segment on the active scan path. In the following, the search for such a justification is mapped to bounded model checking.

A scan segment is defined *accessible* in a given initial scan configuration (or a set of initial scan configurations) if and only if there exists an access pattern that puts the scan segment on the active scan path while the corresponding *updis* and *capdis* ports of the segment are inactive.

Figure 5.1 presents two CSU-accurate state diagrams of an exemplary RSN with a scan segment s. The scan segment s belongs to the active scan path and is enabled for access in scan configurations annotated with s. Clearly, the scan configuration s is reachable and hence the target segment is accessible in the RSN of Figure 5.1a. In contrast, this scan configuration is unreachable and therefore the target is inaccessible in Figure 5.1b. Note that the definition of accessibility refers to a certain (possibly partially specified) initial scan configuration. This definition does not require that scan segments are accessible from all reachable scan configurations (e.g. from state x in Figure 5.1a).

Given the CSU-accurate model of an RSN, proving the accessibility of a scan segment

$s \in S$ is equivalent to refuting the following LTL formula in the CAM:

$$\begin{aligned} A_s := \boldsymbol{G}\, \neg[(\texttt{Active}(s) = 1) \wedge (\texttt{Updis}(s) = 0) \wedge (\texttt{Capdis}(s) = 0)] = \\ \neg \boldsymbol{F}\, [(\texttt{Active}(s) = 1) \wedge (\texttt{Updis}(s) = 0) \wedge (\texttt{Capdis}(s) = 0)]. \end{aligned} \tag{5.2}$$

The LTL formula A_s states that for all execution paths (sequences of scan configurations) in the CAM, the scan segment s does *not* belong to the active scan path or the access to it is disabled by the corresponding *updis* and *capdis* signals.

To check the accessibility of scan segment s, the LTL formula A_s is subject to bounded model checking. To this end, A_s is translated into a propositional formula by unrolling over k CSU operations together with the transition relation T of the CSU-accurate RSN model:

$$\begin{aligned} \texttt{Accessible}(s, c_0, k) := \ \Omega_I(V_0) \wedge \left[\bigwedge_{n=0}^{k-1} \Omega_T(V_n, V_{n+1})\right] \wedge \\ \left[\bigvee_{n=0}^{k} [(\texttt{Active}(V_n, s) = 1) \wedge (\texttt{Updis}(V_n, s) = 0) \wedge (\texttt{Capdis}(V_n, s) = 0)]\right]. \end{aligned} \tag{5.3}$$

The formula $\texttt{Accessible}(s, c_0, k)$ is satisfiable if and only if the scan segment s is accessible within k CSU operations. The accessibility proof is an iterative procedure that checks the satisfiability of formula (5.3) for an increasing number of CSU operation ($k = 1, 2, \ldots$) until the formula is satisfiable, or until a user-defined bound for the number of CSU operations is reached. The truth of A_s is proven by BMC if the SAT instance is unsatisfiable for a sufficiently high bound, as is discussed in Section 5.2.4.

5.1.2. Completeness by Induction

Since bounded model checking can only check execution paths of bounded length, it cannot be used to prove LTL properties of the form $\boldsymbol{G}\, p$, where p is a Boolean formula defined over V. To prove such formulas, it is necessary to show that p holds in all reachable scan configurations of $\mathcal{M}$.

A sufficient but not necessary condition is that p holds in the initial scan configuration and is an *inductive invariant* of the CAM transition relation, as discussed in Section 3.2.2 (p. 40). This condition holds if the following SAT instances are *unsatisfiable*:

$$\varphi_{\text{init}} := \Omega_I(V_i) \wedge \neg p(V_i), \tag{5.4}$$

$$\varphi_{\text{induct}} := p(V_i) \wedge \Omega_T(V_i, V_j) \wedge \neg p(V_j), \tag{5.5}$$

where V_i and V_j are two sets of variables, such that each element in V_i and V_j corresponds to exactly one element in V and $|V_i| = |V_j| = |V|$. Ω_I and Ω_T are the characteristic functions of the initial scan configuration c_0 and the transition relation T, respectively.

Formula φ_{init} is unsatisfiable if and only if p holds in the initial scan configuration. Formula φ_{induct} is unsatisfiable if and only if p is preserved by the transition relation. If both formulas are unsatisfiable, it follows that $\boldsymbol{G}\ p$ holds.

This simple induction method is sufficient to prove many properties of practical interest, including the *robustness* property discussed in Section 5.2. However, this technique is incomplete: If φ_{induct} is satisfiable and φ_{init} is not, the verification result is unknown. Completeness can be achieved with k-induction [Sheeran00] by unrolling the transition relation, but this technique may result in complex SAT instances of polynomial size in the recurrence diameter (see Section 3.2.2, p. 40). In CSU-accurate models, the recurrence diameter—i.e., the longest execution path without repeated scan configurations—grows exponentially in the length of scan segments: If $\mathcal{M}$ contains a scan segment with n shadow registers, an execution path with 2^n distinct scan configurations is possible, which makes k-induction unwieldy for RSNs.

Alternatively, the CSU-accurate BMC can be extended in a straightforward way for the verification of *interval properties*, as in [Nguyen08]. Completeness of CSU-accurate BMC can be also achieved by *interpolation* techniques, as in [McMillan03a, Biere09]. For the class of *robust* RSNs, BMC completeness threshold can be found by structural analysis of the network, as is discussed in Section 5.2.4.

5.1.3. Implementation

CSU-accurate bounded model checking consists in testing the satisfiability of formulas over 3-valued variables (e.g. instance (5.3), p. 72). To check the satisfiability of such formulas using a conventional SAT solver, they are translated to propositional formulas in conjunctive normal form (CNF), as explained below.

Each 3-valued variable $v \in V$ of the CAM is represented in the SAT instance by a pair of Boolean variables (v_0, v_1), as in [Eggersglüss07]. The encoding is as follows:

- $[(v_0, v_1) = (0, 0)] \Leftrightarrow [v = X]$,
- $[(v_0, v_1) = (0, 1)] \Leftrightarrow [v = 0]$,
- $[(v_0, v_1) = (1, 0)] \Leftrightarrow [v = 1]$.

Note that the state $(1, 1)$ has no 3-valued interpretation and hence is forbidden. To prevent solutions with forbidden assignments, the clause $(\neg v_0 \vee \neg v_1)$ is added to the SAT instance for each variable $v \in V$.

The 3-valued functions defined in the CAM such as `Select` or `Active` are derived from the circuit structure (see Section 4.4.1, p. 60) and transformed into CNF by applying the Tseitin transformation [Tseitin83, Biere09]. The interpretation of logic operators follows the Kleene's strongest regular 3-valued logic [Kleene50]. After encoding, a negation of a 3-valued variable x represented by a pair of Boolean variables (x_0, x_1) corresponds to swapping the variables in the pair, i.e., $\neg x$ is represented by a pair (x_1, x_0). A conjunction of 3-valued variables $x \wedge y$ is represented by a pair of Boolean variables (z_0, z_1) such that $z_0 := x_0 \wedge y_0$ and $z_1 := x_1 \vee y_1$. The encoding of any other 3-valued logic operator is easily derived from the encoding of $\wedge$ and $\neg$.

To improve performance of the iterative BMC procedure, incremental SAT solving techniques are employed: The SAT instance for the k-th iteration (i.e., for k CSU operations) is reused in iteration $k + 1$ together with learned clauses from the previous iterations. In iteration $k + 1$, the SAT instance from the k-th iteration is extended by unrolling the transition relation for one more time step, and by addition of clauses that specify property violation in one of the $k + 1$ time steps. The old clauses that describe property violation in k time steps are deactivated using selector variables, as discussed in Section 2.4.1 (p. 31).

5.2. Verification of Robust Scan Networks

This section defines the class of *robust* reconfigurable scan networks, discusses their properties, and develops efficient verification techniques for this type of structures. Robustness is defined formally in Section 5.2.1. Section 5.2.2 presents an efficient method for the verification of RSN robustness using the CSU-accurate model. Section 5.2.3 introduces a structural method for diameter approximation in robust RSNs. In Section 5.2.4, the RSN diameter is used to find a tight completeness threshold for bounded model checking.

5.2.1. Robustness Definition and Properties

Two robustness properties are distinguished in this thesis: weak robustness and strong robustness. Below, these properties are defined formally and explained at an example.

Let $\mathcal{M}$ be the CSU-accurate model of an RSN, $c_0 \in C$ the initial scan configuration of $\mathcal{M}$, T the transition relation of $\mathcal{M}$, and `Valid` the validity predicate, as defined in Section 4.4.2 (p. 62).

Definition 13. (Weak Robustness) The RSN represented by $\mathcal{M}$ is *weakly robust* if the initial scan configuration c_0 is valid ($\texttt{Valid}(c_0) = 1$) and the predicate `Valid` is globally true in all initialized execution paths of $\mathcal{M}$ (i.e., the LTL property $\boldsymbol{G}$ `Valid` holds in $\mathcal{M}$).

Definition 14. (Strong Robustness) The RSN represented by $\mathcal{M}$ is *strongly robust* if the initial scan configuration c_0 is valid ($\texttt{Valid}(c_0) = 1$) and the predicate `Valid` is an inductive invariant of the transition relation T, i.e., the following condition holds:

$$\bigforall_{(c_1, c_2) \in T} \texttt{Valid}(c_1) \Rightarrow \texttt{Valid}(c_2). \tag{5.6}$$

Intuitively, an RSN is *weakly robust* if and only if all reachable scan configurations are valid. An RSN is *strongly robust* if and only if no invalid scan configuration can be reached from a valid scan configuration.

Examples of CSU-accurate state diagrams for each type of RSNs are given in Figure 5.2. Invalid scan configurations are annotated with $\neg V$. The RSN represented by Figure 5.2a is not robust since there exists an invalid scan configuration that is reachable from the initial state. This invalid scan configuration is absent in Figure 5.2b and hence this RSN is weakly robust. The RSN from Figure 5.2c is both weakly and strongly robust since no invalid scan configuration is reachable from the set of valid scan configurations.

Remark 5. The definition of weak robustness allows that for some *unreachable* scan

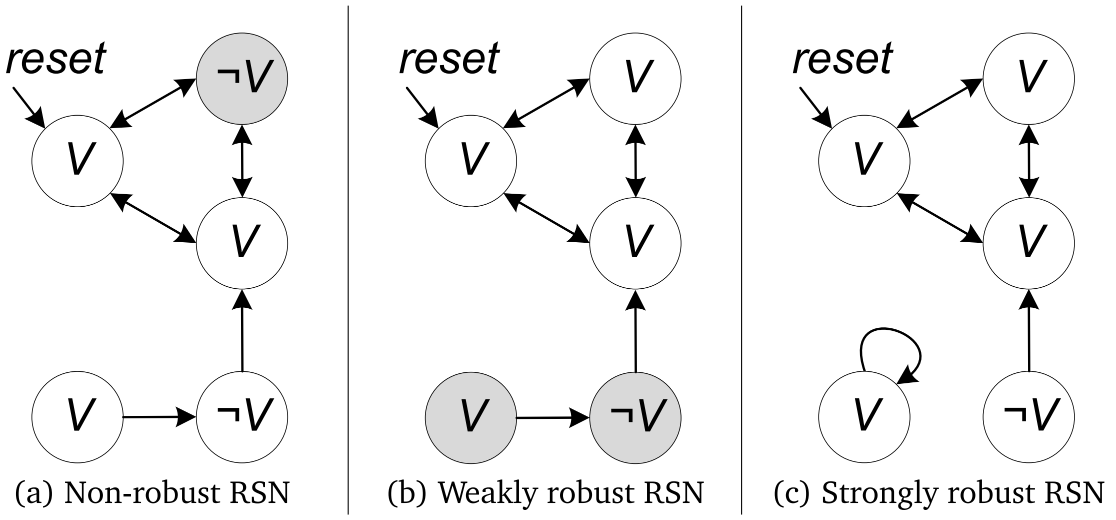


Figure 5.2.: Examples of CSU-accurate state diagrams (invalid scan configurations are annotated with $\neg V$)

configuration $c_\text{v} \in C$ such that `Valid`$(c_\text{v}) = 1$, there exists $c_\text{nv} \in C$ such that $(c_\text{v}, c_\text{nv}) \in T$ and `Valid`$(c_\text{nv}) = 0$. Thus, a weakly robust RSN is not necessarily strongly robust.

Lemma 1. The class of weakly robust RSNs includes the class of strongly robust RSNs.

Proof. For every initialized execution path $\pi =< c_0, c_1, c_2, \ldots >$ of a *strongly robust* $\mathcal{M}$ the following statement holds according to Definition 14: If the initial scan configuration c_0 is valid (`Valid`$(c_0) = 1$) then the validity of consecutive scan configurations is preserved by each transition, i.e., $\forall_{i \geq 0}$ `Valid`(c_i). Therefore, the LTL property $\boldsymbol{G}$ `Valid` holds in the strongly robust $\mathcal{M}$ and hence $\mathcal{M}$ is also *weakly robust*. □

Weakly robust RSNs (and hence also strongly robust RSNs according to Lemma 1) have the following properties:

1. The selected scan segments always form an active scan path regardless of the scan data applied at the primary scan-input. Therefore, erroneous scan data can never break the active scan path of a robust RSN. This property is beneficial, for instance, for observability in post-silicon debug: An internal fault that alters scan data is less likely to affect the integrity of the scan path.
2. The CSU-accurate model of a weakly robust RSN does not produce spurious counterexamples as stated by the following theorem:

Theorem 1. (Completeness of a Weakly Robust CAM) The CSU-accurate model $\mathcal{M}$ of a weakly robust RSN is *complete* with respect to the class of LTL properties that can be expressed in terms of variables in $\mathcal{M}$ (cf. Definition 9, p. 30).

Proof. According to Definition 13 (p. 75), the LTL property $\boldsymbol{G}\ \mathtt{Valid}$ holds in a weakly robust RSN and hence only transitions between valid scan configurations are possible. For such transitions, the CSU-accurate abstraction *exactly* models CSU-accurate behavior, as stated by Remark 4 (p. 66). Thus, the CAM of a weakly robust RSN does not cause spurious counterexamples and hence is complete. □

3. As all reachable scan configurations are valid, the CAM of a weakly robust RSN can be simplified by removing the predicate `Valid` from the definition of the `Active` function: For all $c \in C$ and $s \in S$ it holds that $\mathtt{Active}(\mathtt{c}, \mathtt{s}) = \mathtt{Select}(\mathtt{c}, \mathtt{s})$. This simplification of the CAM improves performance of CSU-accurate formal verification, as well as access pattern generation discussed in Chapter 6.

5.2.2. Verification of Robustness

Weak robustness can be verified using any unbounded LTL model checking method which can prove the property $\boldsymbol{G}\ \mathtt{Valid}$ of the CAM. However, as LTL model checking is PSPACE-complete in general [Sistla85], such methods are computationally expensive and may fail to verify robustness of large RSN designs. In contrast, the verification of *strong robustness* is an NP-complete problem that can be mapped to SAT-based induction, as discussed in Section 5.1.2 (p. 72), and solved using efficient SAT solvers.

Let $\mathcal{M}$ be the CSU-accurate model of an RSN, $c_0 \in C$ the initial scan configuration of $\mathcal{M}$, T the transition relation, V the set of variables in $\mathcal{M}$, and `Valid` the validity predicate. According to Definition 14 (p. 75), strong robustness requires that the initial scan configuration c_0 be valid and that `Valid` be an inductive invariant of T. These two conditions are encoded as separate SAT instances:

$$\varphi_{\text{init}} := \Omega_I(V_i) \land \neg\mathtt{Valid}(V_i), \tag{5.7}$$

$$\varphi_{\text{induct}} := \mathtt{Valid}(V_i) \land \Omega_T(V_i, V_j) \land \neg\mathtt{Valid}(V_j), \tag{5.8}$$

where V_i and V_j are two sets of variables, such that each element in V_i and V_j corresponds to exactly one element in V and $|V_i| = |V_j| = |V|$. Ω_I and Ω_T are the characteristic functions of the initial scan configuration c_0 and the transition relation T, respectively.

An RSN is strongly robust if and only if both φ_{init} and φ_{induct} are unsatisfiable. Otherwise, the satisfying assignment to φ_{induct} describes a transition from a valid scan configuration into an invalid one, and hence constitutes a counterexample to strong robustness.

5.2.3. Model Diameter

Bounded model checking becomes complete if the bound of examined execution paths is sufficiently high (cf. Section 3.2.2, p. 40). For LTL properties of the form $\boldsymbol{G}\,p$, the BMC completeness threshold is often approximated with the model diameter (cf. Definition 6, p. 27). For a circuit with n sequential elements, the upper bound of the diameter is 2^n. In practice, this overapproximation is much too high to achieve completeness of BMC. As discussed below, for weakly robust reconfigurable scan networks without circular dependencies, a much tighter approximation is easily found by structural analysis of the RSN.

Dependency Graph

In the following, the structural dependencies of a robust RSN are captured in form of a *dependency graph*. The dependency graph is later used to derive an overapproximation of the model diameter. For the sake of brevity, it is assumed that the RSN contains no data segments. The dependency graph can handle data segments in the same way as scan segments and hence the extension is straightforward.

Definition 15. (Dependency Graph) Let $\mathcal{M}$ be a CSU-accurate model of a weakly robust RSN with the set of scan segments S and the functions `Select`, `Updis`, and `Capdis` (cf. Definition 11, p. 60). The dependency graph of $\mathcal{M}$ is a directed graph $G = (S, E)$ with a set of nodes S equivalent to the set of scan segments in $\mathcal{M}$, and a set of edges $E \subseteq S \times S$ that represents the dependencies: An edge between two scan segments $s_1, s_2 \in S$ exists in G, written $(s_1, s_2) \in E$, if and only if any shadow register of s_1 resides in the combinational input cone of any control signal of s_2, i.e., if function $\texttt{Select}(s_2)$, $\texttt{Updis}(s_2)$ or $\texttt{Capdis}(s_2)$ depends on any shadow register of s_1.

For *acyclic* dependency graphs, *levelization* is defined as follows:

Definition 16. (Levelization of a Dependency Graph) Let $G = (S, E)$ be the dependency graph of a CSU-accurate model $\mathcal{M}$. If G is acyclic, the *levelization* of G is a function $l_G : S \to \mathbb{N}$ that maps each scan segment to its corresponding *level*. For a scan segment $s \in S$, $l_G(s)$ is defined recursively as follows:

$$l_G(s) := \begin{cases} 0 & \text{if } \forall_{(s_i,s_j)\in E}\ s_j \neq s, \\ 1 + \max\{\ l_G(s_i) \mid (s_i, s) \in E\ \} & \text{otherwise.} \end{cases} \tag{5.9}$$

The set of scan segments at the n-th level of graph G is denoted by G^n:

$$G^{\,n} := \{\ s \in S \mid l_G(s) = n\ \}. \tag{5.10}$$

Diameter Calculation

For a weakly robust RSN with an acyclic dependency graph, a tight overapproximation of the CSU-accurate model diameter is derived directly from the dependency graph. Note that the diameter of a CAM is expressed in terms of the number of CSU operations instead of clock cycles.

Theorem 2. (CSU-Accurate Model Diameter) Let $\mathcal{M}$ be a CSU-accurate model of a weakly robust RSN with an acyclic dependency graph G and levelization l_G with $k \in \mathbb{N}^+$ levels. The diameter of $\mathcal{M}$, written $d(\mathcal{M})$, fulfills the following inequality:

$$d(\mathcal{M}) \leq |G^{\,0}| \cdot \prod_{i=1}^{k-1} \left(|G^{\,i}| + 1\right). \tag{5.11}$$

Proof. This theorem is proven by induction on the number of levels. The diameter of a CSU-accurate model is understood as the maximal number of CSU operations that are required to reach an arbitrary scan configuration from any reachable scan configuration. If G has just one level ($k = 1$), all scan segments in the network are independent of each other. In this case, a scan segment is put on the active scan path by justification of the primary data/control inputs of the RSN. Therefore, just one CSU operation is required to access each scan segment, and hence the diameter is at most the number of scan segments, i.e. $|G^{\,0}|$. It follows that (5.11) holds for $k = 1$.

For $k > 1$ and $0 \leq i < k$, let $d_{\,i}(\mathcal{M})$ denote the diameter of a partial dependency graph containing only scan segments at levels $0 \ldots i$. The partial diameter $d_{\,i}(\mathcal{M})$ is understood as the maximal number of CSU operations that are required to reach an arbitrary reachable state of scan segments at levels $0 \ldots i$ from any reachable state. Note that $d_{\,k-1}(\mathcal{M})$ is the diameter of $\mathcal{M}$, i.e., $d_{\,k-1}(\mathcal{M}) = d(\mathcal{M})$. To reach an arbitrary state of the scan segments at the i-th level, each scan segment at this level must be accessed at most once. A distinct state of scan segments at levels $0 \ldots (i-1)$ may be needed to justify the access to each scan segment at the i-th level, requiring at most $d_{\,i-1}(\mathcal{M})$ CSU operations per access to each scan segment in $G^{\,i}$. Thus, in the worst case, a total of $|G^{\,i}| \cdot d_{\,i-1}(\mathcal{M})$ CSU operations is required to access all scan segments at the i-th level, and another $d_{\,i-1}(\mathcal{M})$ CSU operations to reach an arbitrary state at lower levels. Therefore, for $0 < i < k$:

$$d_{\,i}(\mathcal{M}) \leq \big(|G^{\,i}| + 1\big) \cdot d_{\,i-1}(\mathcal{M}). \tag{5.12}$$

According to the proof for $k = 1$, it also holds that:

$$d_{\,0}(\mathcal{M}) \leq |G^{\,0}|. \tag{5.13}$$

From (5.12) and (5.13) it follows that $|G^{\,0}| \cdot \prod\limits_{i=1}^{k-1} \big(|G^{\,i}| + 1\big)$ is an overapproximation of $d_{\,k-1}(\mathcal{M}) = d(\mathcal{M})$. □

Corollary 1. If all scan segments in $G^{\,0}$ are always accessible (i.e., their *select* and *capdis/updis* signals are independent from primary control/data inputs), it follows that:

$$d(\mathcal{M}) \leq \prod_{i=1}^{k-1} \big(|G^{\,i}| + 1\big). \tag{5.14}$$

(Note that for $k = 1$ the empty product yields 1.)

Example

Figure 5.3 presents the structure of an exemplary reconfigurable scan network. The lines connecting scan segments and multiplexers denote scan paths. Scan segments that drive internal control signals, i.e. `S1`, `S3`, `S4` and `S7`, are assumed 1-bit long. (The length of scan segments, however, has no impact on the CAM diameter.) The *select* ports of scan segments, as well as the *address* ports of multiplexers are driven

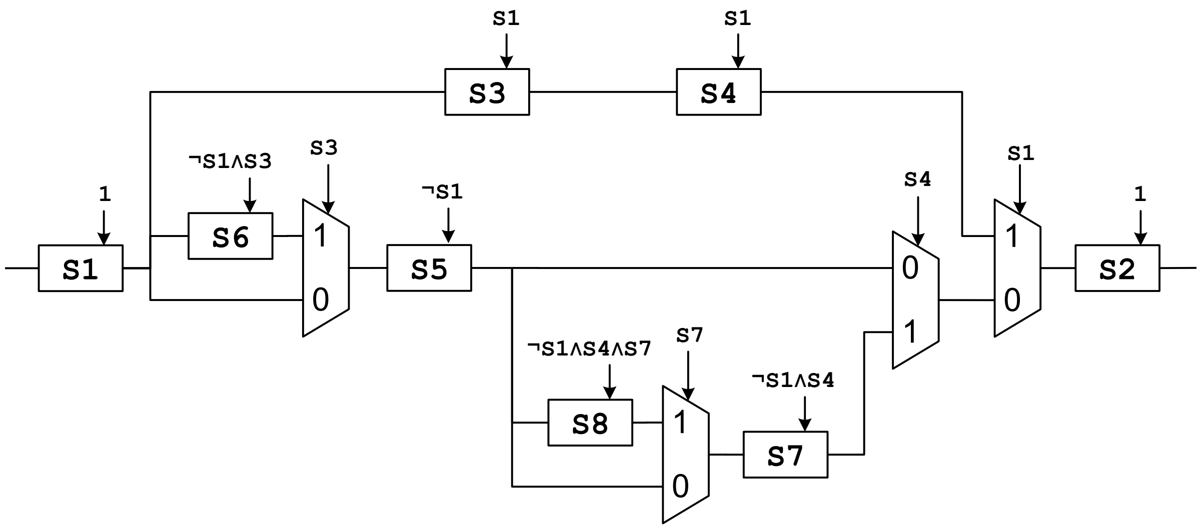


Figure 5.3.: Example of a *strongly robust* reconfigurable scan network

by combinational logic blocks that take inputs from the shadow registers of S1, S3, S4 and S7. For the sake of readability, the combinational logic is omitted in Figure 5.3. Instead, internal control signals for *select* and *address* ports are annotated with their corresponding logic functions.

The RSN from Figure 5.3 is *strongly robust* since there exists no invalid scan configuration: Each scan segment belongs to the active scan path (is accessible) if and only if the scan segment is selected (its *select* signal is 1). This is assured by the structure of the network and can be verified with the approach from Section 5.2.2.

The dependency graph of the exemplary RSN is presented in Figure 5.4. According to Definition 15, an edge from node a to b exists in the dependency graph if and only if any control signal of the scan segment corresponding to b depends on the content of any shadow register of scan segment a. The dependency graph is acyclic and has four levels which are indicated at the top of Figure 5.4.

The scan segments on the lowest level (S1 and S2) always belong to the active scan path as their *select* ports are tied to logic 1 and the RSN is robust. Hence, according to Corollary 1 (p. 80), the overapproximation of the CAM diameter is calculated by multiplying the incremented cardinalities of levels 1, 2, and 3, which yields: $4 \cdot 3 \cdot 2 = 24$. This diameter overapproximation can be used as a BMC completeness threshold: If CSU-accurate BMC does not produce any counterexample to an LTL property of the form $\boldsymbol{G}\, p$ within a bound of 24 CSU operations (i.e., if the SAT instance with 25 time steps is unsatisfiable), the property is guaranteed to hold in the RSN.

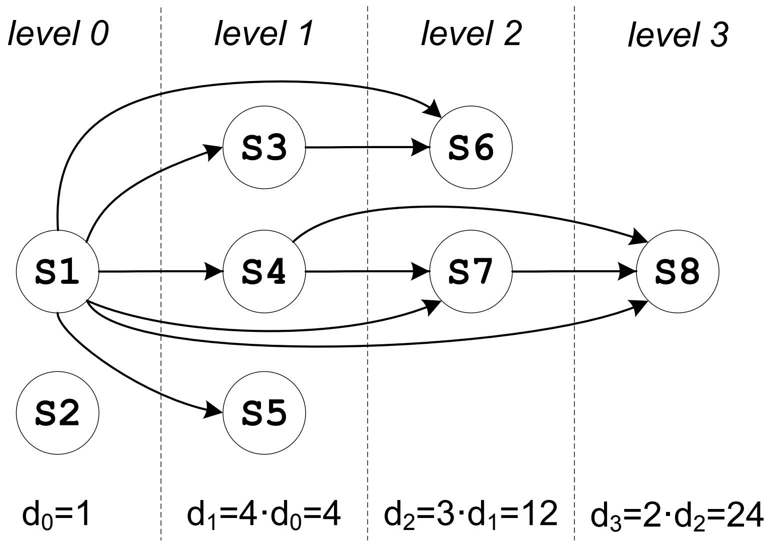


Figure 5.4.: Dependency graph for the RSN from Figure 5.3

5.2.4. Completeness Threshold

As discussed in Section 3.2.2 (p. 40), an overapproximation of the model diameter is a completeness threshold for bounded model checking of LTL properties of the form $\boldsymbol{G}\ p$, where p is a proposition over the model's variables. For a specific property, this threshold can be further tightened by examining the property's *cone of influence*.

Definition 17. (Cone of Influence) Let $\mathcal{M}$ be a CSU-accurate model of an RSN with the set of scan segments S and dependency graph $G = (S, E)$, and let $P := \boldsymbol{G}\ p$ be an LTL property where p is a Boolean function that depends on a subset of scan segments $S' \subseteq S$. The *cone of influence* of P is a dependency graph $G_P := (S_P \subseteq S, E_P \subseteq E)$, such that:

$$S_P := \texttt{cone}(S'), \tag{5.15}$$

$$E_P := \{\ (s_i, s_j) \in E \mid s_i, s_j \in S_P\ \}, \tag{5.16}$$

where for any $A \subseteq S$, $\texttt{cone}(A)$ is defined inductively as follows:

$$\texttt{cone}^0(A) := A, \tag{5.17}$$

$$\texttt{cone}^i(A) := \{\ s \in S \mid \exists_{s_j \in A}\ (s, s_j) \in E\ \} \cup \texttt{cone}^{i-1}(A), \tag{5.18}$$

$$\texttt{cone}(A) := \texttt{cone}^\infty(A). \tag{5.19}$$

Theorem 3. (CSU-Accurate Completeness Threshold for $\boldsymbol{G}\, p$) Let $\mathcal{M}$ be a CSU-accurate model of a weakly robust RSN, $P := \boldsymbol{G}\, p$ be an LTL property with an acyclic cone of influence G_P and levelization l_P with $k \in \mathbb{N}^+$ levels. The completeness threshold of bounded model checking for property P in $\mathcal{M}$, written $\texttt{ct}(\mathcal{M}, P)$, fulfills:

$$\texttt{ct}(\mathcal{M}, P) \leq |G_P^{\,0}| \cdot \prod_{i=1}^{k-1} \big(|G_P^{\,i}| + 1\big). \tag{5.20}$$

Proof. This theorem is proven by leveraging Theorem 2 (p. 79). Due to the definition of the cone of influence G_P, the scan segments in S_P can be set to an arbitrary reachable state regardless of the content of scan segments in $S \setminus S_P$. According to Theorem 2, the diameter of G_P, denoted by d_P, fulfills $d_P \leq |G_P^{\,0}| \cdot \prod_{i=1}^{k-1} \big(|G_P^{\,i}| + 1\big) = \hat{d}_P$. The overapproximation of the diameter denoted by $\hat{d}_P$ gives the maximal number of CSU operations that are required to reach an arbitrary state of scan segments in S_P from any reachable scan configuration. Therefore, if P does not hold, the shortest counterexample can have at most $\hat{d}_P$ CSU operations. It follows that $\hat{d}_P$ is an overapproximation of the completeness threshold for the property P, i.e., $\texttt{ct}(\mathcal{M}, P) \leq \hat{d}_P$. □

Corollary 2. As in Corollary 1 (p. 80), if all scan segments in $G_P^{\,0}$ are always accessible, it follows that:

$$\texttt{ct}(\mathcal{M}, P) \leq \prod_{i=1}^{k-1} \big(|G_P^{\,i}| + 1\big). \tag{5.21}$$

Example

In the following, an LTL property $P := \boldsymbol{G}\, (\texttt{Select(S8)} = 0)$ is checked for the RSN from Figure 5.3 (p. 81) using the CSU-accurate BMC approach and leveraging the completeness threshold. The property P states that scan segment `S8` never belongs to the active scan path and hence is inaccessible. The cone of influence of P, as shown in Figure 5.5, is a subset of the dependency graph from Figure 5.4 which includes only scan segments that the property refers to (i.e., `S8`), and their transitive input cone (`S7`, `S4`, and `S1`).

The cone of influence of P has four levels with exactly one scan segment per level. According to Corollary 2, the completeness threshold for property P is hence at most

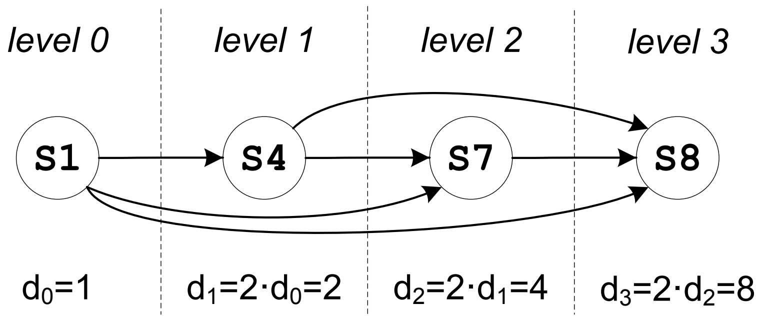


Figure 5.5.: Cone of influence for proving inaccessibility of scan segment S8 in the RSN from Figure 5.3 (p. 81)

$2 \cdot 2 \cdot 2 = 8$. Therefore, S8 is guaranteed to be inaccessible if CSU-accurate BMC does not find any counterexample to P within a bound of 8 CSU operations (i.e., the SAT instance with 9 time steps is unsatisfiable).

5.3. Model Soundness and Completeness

An abstraction is *sound* if and only if every property that holds in the abstract model also holds in the concrete model (cf. Definition 8, p. 30). The CSU-accurate model abstracts the temporal behavior of an RSN but still *exactly* models state transitions between valid scan configurations (under the assumption of stable internal control signals which is discussed below; cf. Remark 4, p. 66). For state transitions to invalid scan configurations, CAM pessimistically assumes that the state of potentially selected scan and data segments becomes unknown/X (cf. Assumption 6, page 67). In Kleene's strongest regular 3-valued logic, well-formed formulas are *monotonic*: If the input to a formula becomes less specified (i.e., some of its input variables are set to X), the output of the formula is either stable or also becomes less specified (some of its outputs become unknown/X) [Kleene50]. Therefore, a 3-valued formula that is satisfied for all reachable states of the CAM (i.e., an invariant property of the CAM) is also guaranteed to hold of all reachable states of the concrete RSN model. Thus, the pessimism of the CSU-accurate model does not compromise the soundness of this abstraction.

As cycle-accurate execution of a CSU operation is abstracted in the CAM into a single state transition, the CAM implicitly assumes that all internal control signals are stable throughout the CSU operation (cf. Assumption 3 and 4, page 66). This assumption holds trivially for all control signals generated internally to the RSN since these signals change only after the *update* phase. For primary data/control signals, however, this assumption may not hold. In this case, for instance, if an *address* port of a scan multiplexer is driven by a primary control input which is unstable during the *shift* phase, scan data may be lost, which is not modeled by the CAM. Thus, the CAM is *sound* only if primary data/control inputs are guaranteed stable during the *capture* and *shift* phases. Otherwise, the CAM is *not* sound, i.e., a property may be false in the RSN although it holds in the CAM. To guarantee CAM soundness, the stability of primary data/control inputs can be either ensured by design or must be proven in the cycle-accurate RSN/system model.

As mentioned above and discussed in Section 4.4.4 (p. 66), the CAM is pessimistic: According to the CAM transition relation, the content of potentially selected scan segments is assumed undefined in invalid scan configuration although it may be well defined in the cycle-accurate model. Thus, the CAM is *not* complete in general and may produce spurious counterexamples to a property even if the property holds in the cycle-accurate model. However, if invalid scan configurations are unreachable, the CAM is complete (this is the case for all robust RSNs, cf. Theorem 1, p. 77).

5.4. Experimental Evaluation

The proposed CSU-accurate verification approach is evaluated on SIB-based, MUX-based and flat scan architectures described in detail in Appendix A (p. 141). The experimental setup and detailed results are covered in Appendix B (p. 147). This section presents a brief summary.

Figure 5.6 presents the verification effort for the three types of scan architectures. For all considered benchmarks, the *strong robustness* property is successfully proven using the technique presented in Section 5.2.2 (p. 77). For the SIB- and MUX-based architectures, the verification of robustness takes up to 100 s in the worst case. For flat scan architectures, robustness verification effort is below 1 s.

The *accessibility* of the benchmarks is verified with the approach discussed in Section 5.1.1 (p. 71). It is formally proven that all scan segments of considered benchmark are both controllable and observable. For a majority of the RSNs, the total verification time is below 10 s, and it raises up to 200 s for the largest MUX-based benchmark.

Figure 5.7 presents the average and maximal number of CSU operations required to access a scan segment in SIB- and MUX-based architectures. Due to more complex sequential dependencies, the MUX-based architecture requires on average about one CSU operation more than the SIB-based architecture, and up to three more CSUs in the worst case. As the number of required CSU operations corresponds to the number of time steps in bounded model checking, this result explains the slightly higher verification effort required for MUX-based RSNs.

The verification of robustness and accessibility in MUX-based benchmarks with random design bugs is covered in Appendix B (Section B.1, p. 147, and Section B.2, p. 148). Interestingly, while the accessibility of all scan segments is preserved for some random design errors, the verification of robustness discovers all the injected bugs. On average, robustness verification requires slightly less effort in the erroneous designs compared with fault-free benchmarks, while the maximal verification effort is below two minutes.

As shown in Section B.2 (p. 148), the worst case BMC completeness threshold for accessibility verification in MUX-based architectures is 64. This threshold is low enough to *prove* inaccessibility of scan segments, e.g. for security verification. In the faulty designs, the verification with a threshold of 64 CSU operations requires up to two hours in the worst case.

Figure 5.8 compares the performance of the proposed *CSU-accurate* bounded model checking method with a *cycle-accurate* model checking tool. In each experiment, the accessibility of a random scan segment in the largest MUX-based benchmark (p93791) is proven. The cycle-accurate model checker exceeds a time limit of one hour in two experiments, and the solving time changes by over an order of magnitude in different experiments. In contrast, the proposed approach is successful in all the experiments, exhibits much more stable run-times, and is faster by at least two orders of magnitude. This result clearly shows that the proposed CSU-accurate abstraction provides a great performance improvement over cycle-accurate models.

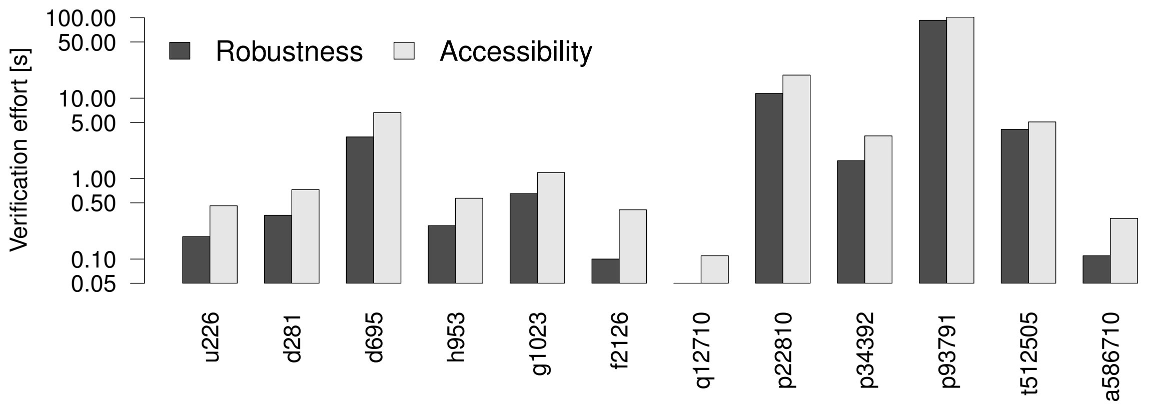
5.5. Summary

CSU-accurate modeling is directly applicable to formal verification of complex reconfigurable scan networks. This temporal abstraction significantly improves the performance of model checking algorithms which otherwise face scalability issues in cycle-accurate RSN models. Under minor assumptions about the stability of primary data/-control inputs of an RSN, the CSU-accurate model is *sound*. Moreover, for the class of *robust* scan networks, CSU-accurate models are *complete*.

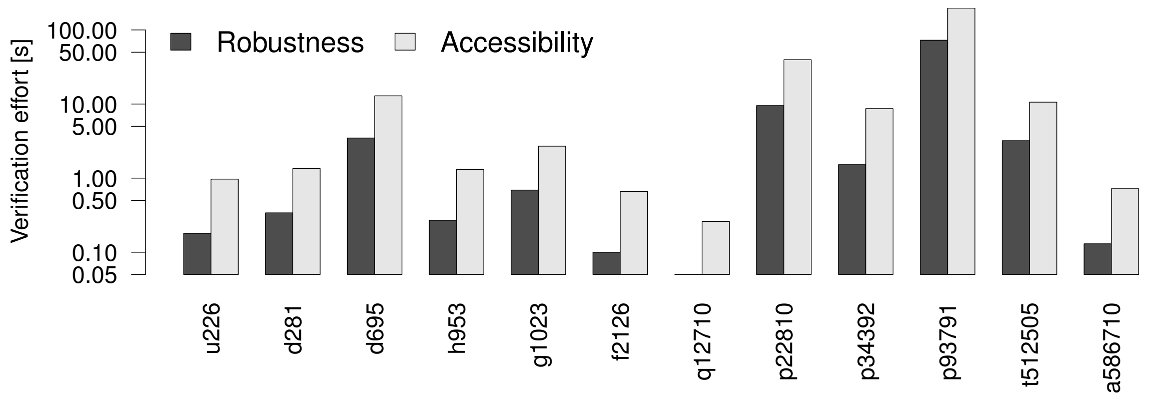
While the diameter of cycle-accurate scan network models may be very large, the diameter of CSU-accurate abstractions is significantly lower and does not depend on the length of scan segments. For the class of *robust* scan networks with acyclic dependency graphs, a tight overapproximation of the diameter is easily found by structural analysis of the RSN. This diameter overapproximation is used as a completeness threshold in bounded model checking experiments to prove properties in the *unbounded* sense. To further improve verification performance, the completeness threshold is tightened by a structural analysis of the cone of influence of a given property.

Experiments show that the CSU-accurate BMC technique efficiently handles even large and complex scan networks. The completeness threshold is small enough to prove inaccessibility of scan segments even in the largest benchmarks. The robustness property is very beneficial both due to the possibility of calculating a tight completeness threshold, and also due to the high probability of catching design bugs just by checking robustness.

(a)



(b)



(c)

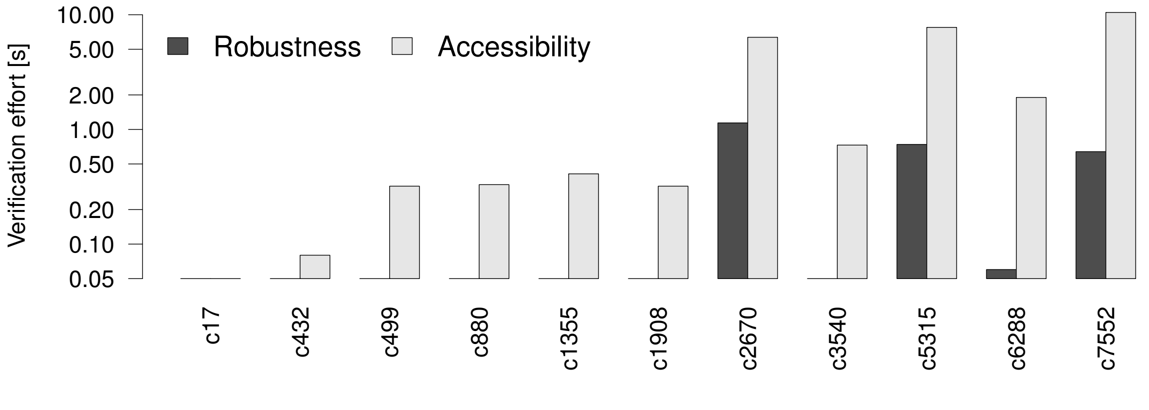


Figure 5.6.: Robustness and accessibility verification effort in (a) SIB-based, (b) MUX-based, and (c) flat scan architecture

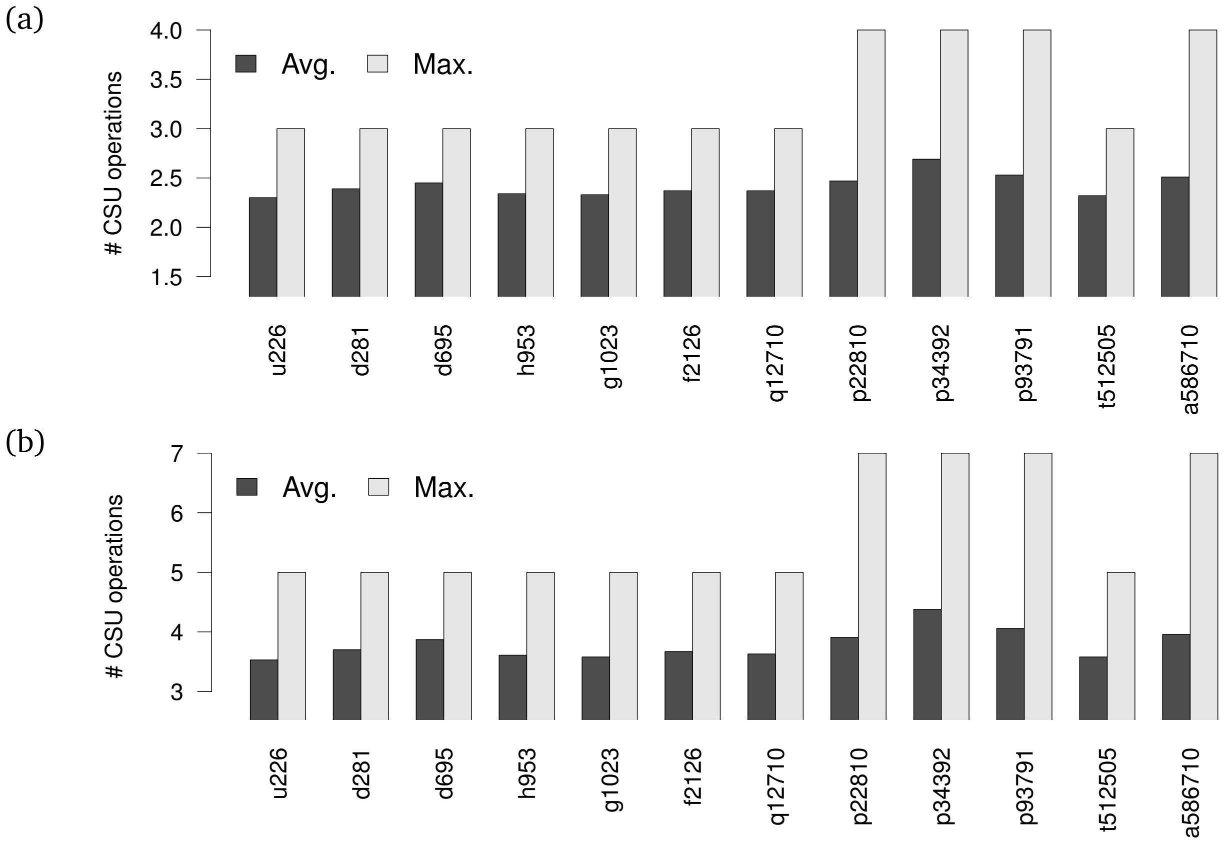


Figure 5.7.: Average and maximal number of CSU operations required to access a scan segment in (a) SIB-based and (b) MUX-based scan architecture

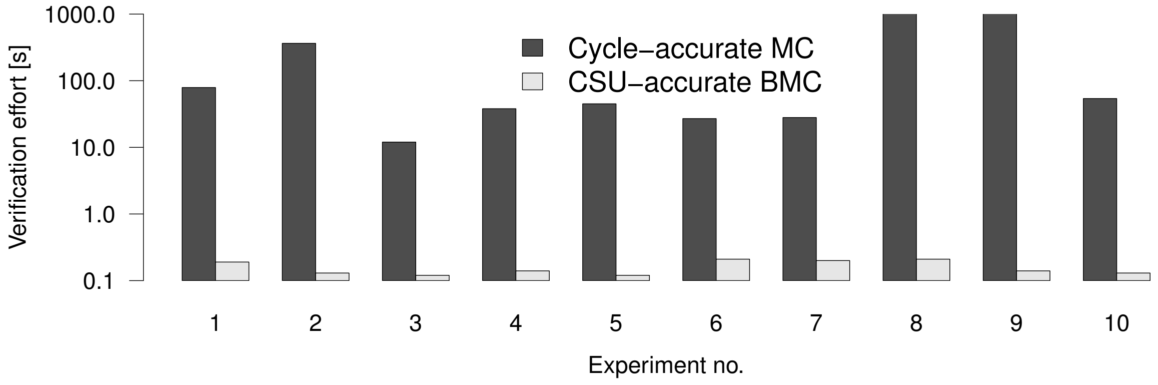


Figure 5.8.: Performance comparison of a cycle-accurate model checking tool and the proposed CSU-accurate BMC algorithm

6. Access Optimization

An access to a reconfigurable scan network may require several CSU operations to put the target segment on the active scan path. The process of computing the required scan-in sequence (scan data) is called *access pattern generation*, or *pattern retargeting* in IEEE Std. P1687.

Figure 6.1 shows an example of a simple RSN. The shadow registers of the 1-bit scan segments S1 and S3 control the access to two multi-bit scan segments S2 and S4, respectively. Segment S2 (S4) belongs to the active scan path only if S1 (S3) is set to 1. In the initial scan configuration, it is assumed that $\texttt{S1} = 1$ and $\texttt{S3} = 0$, hence S4 is bypassed.

Table 6.1 shows two examples of access patterns for the scan segment S4: In access A1, the first CSU operation sets S3 to 1 to put the target S4 on the active scan path, and the second CSU operation accesses S4. The *access time* in clock cycles amounts to the number of bits that need to be shifted (length of the active scan path) plus 2 cycles per CSU operation required for the *capture* and *update* phases. Thus, the total access time of A1 with two CSU operations amounts to $8 + 2 \cdot |\texttt{S2}| + |\texttt{S4}|$.

In reconfigurable scan networks, an access to a scan segment may be realized in many ways, using different access patterns. Possible solutions may greatly differ in the access time. For instance, the access time of A1 in Table 6.1 can be reduced by bypassing S2 in the second CSU operation, as shown for access A2.

Access pattern generation for SIB-based architectures as in [Ghani Zadegan11b] is straightforward: All SIBs that enclose the target scan segment must be opened, and all remaining SIBs must be closed to optimize the access time. While this trivial access generation algorithm provides minimal access time in SIB-based architectures, it cannot be applied to general RSNs, where internal control signals may be generated by arbitrary combinational logic blocks driven by multiple scan segments distributed over the network.

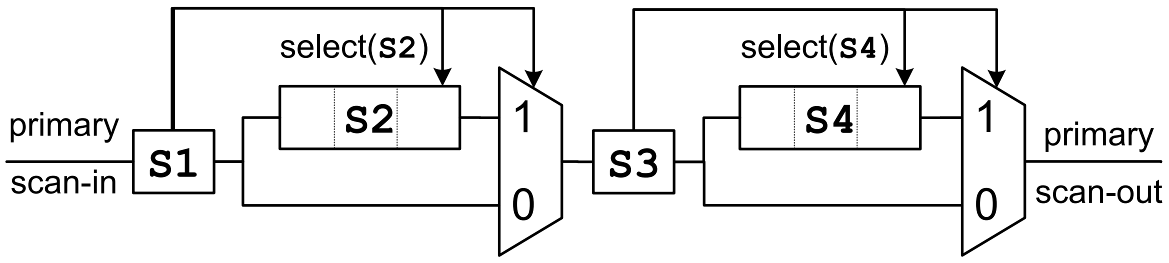


Figure 6.1.: Example of a reconfigurable scan network

		S1	S2	S3	S4	access time
A1	initial state	1	X	0	X	
	CSU no. 1	W1	ACCESS	W1	BYPASS	2+|S2|+2
	CSU no. 2	W1	ACCESS	W0	ACCESS	2+|S2|+|S4|+2
A2	initial state	1	X	0	X	
	CSU no. 1	W0	ACCESS	W1	BYPASS	2+|S2|+2
	CSU no. 2	W1	BYPASS	W0	ACCESS	2+|S4|+2

Table 6.1.: Access patterns to scan segment S4 in the RSN from Figure 6.1

In general reconfigurable scan networks, the search for an access pattern can be mapped to bounded model checking (cf. Section 5.1.1, p. 71) and solved with a SAT solver. This approach, however, does not allow for access time optimization, which poses a much harder problem and requires a dedicated algorithm.

The problem of access time optimization is formulated formally in Section 6.1 and mapped to pseudo-Boolean optimization in Section 6.2 leveraging the CSU-accurate model from Chapter 4 (p. 51). Based on this mapping, an efficient access optimization procedure is developed in Section 6.3. The performance of the proposed algorithm is studied in Section 6.4 and Appendix C.

6.1. Problem Formulation

Optimal *access pattern generation* (or optimal *pattern retargeting*) is a search for the shortest sequence of bits that need to be shifted into the RSN during one or multiple CSU operations to reach a certain target scan configuration with minimal access time.

Let $\mathcal{M}$ be the CAM of an RSN with the set of scan segments S, the `Active` and `Capdis` functions, and the transition relation T and let (c_0, c_t) specify an access with initial scan configuration $c_0 \in C$ and target scan configuration $c_t \in C$. Optimal access pattern generation is the computation of an execution path π with length $|\pi| \in \mathbb{N}$, such that the following condition holds:

$$\big(\pi(0) = c_0\big) \wedge \Bigg(\underset{i=1\ldots|\pi|}{\forall} \big(\pi(i-1), \pi(i)\big) \in T \Bigg) \wedge \big(\pi(|\pi|) = c_t\big) \tag{6.1}$$

and the path π minimizes the *access time* (number of required clock cycles) expressed with the following pseudo-Boolean cost function:

$$\texttt{Cycles}(\pi) := D \cdot |\pi| + \sum_{i=0}^{|\pi|-1} \sum_{s \in S} \Big[|s| \cdot \big[\texttt{Active}\big(\pi(i), s\big) = 1\big]\Big], \tag{6.2}$$

where $D \in \mathbb{N}$ is a constant that amounts to the number of cycles required to perform the *capture* and *update* phase of a CSU operation, and $|s|$ denotes the length of a scan segment $s \in S$. If the RSN is accessed through a JTAG TAP, the constant D amounts to at least 4 cycles due to the overhead of the TAP controller [JTA13], or more if pause cycles are required.

Condition (6.1) is satisfied if and only if the first element of π equals the initial scan configuration c_0, π is a valid execution path in $\mathcal{M}$, and the last element of π equals the target scan configuration c_t. The access time given by formula (6.2) amounts to the total number of clock cycles for the *capture* and *update* phases ($D \cdot n$) plus the total number of shift cycles. Since the `Active` function for a scan segment $s \in S$ evaluates to 1 if s is part of the active scan path (cf. Definition 11, p. 60), the number of required shift cycles (scan-in sequence length) equals the number of `Active` functions that are 1 weighted with the length of the corresponding scan segments. Note that the length of the execution path that minimizes the cost function (the number of CSU operations $|\pi|$ required for the optimal solution) is *a priori* unknown.

Access Merging

Access merging is the generation of access patterns for concurrent read and write operations on multiple target scan segments. The challenge of access merging is to find the optimal access order which results in minimal access time.

For multiple write operations, it is sufficient to specify the values for target scan segments by constraining the target scan configuration c_t. This form of access specification does not restrict the access order and gives room for access time optimization. However, specifying read accesses in this way restricts them to the last CSU operation, although the individual read operations may occur in any intermediate scan configuration between c_0 and c_t. To enable access order optimization for concurrent read operations, condition (6.1) is extended as follows:

$$\begin{gathered}\big(\pi(0) = c_0\big) \wedge \left(\forall_{i=1\ldots|\pi|} \big(\pi(i-1), \pi(i)\big) \in T \right) \wedge \big(\pi(|\pi|) = c_t\big) \wedge \\ \left(\forall_{s \in S_R} \; \exists_{i=0\ldots|\pi|-1} \Big[[\texttt{Active}\big(\pi(i), s\big) = 1] \wedge [\texttt{Capdis}\big(\pi(i), s\big) = 0] \Big] \right), \end{gathered} \tag{6.3}$$

where $S_R \subseteq S$ is the set of read scan segments. This condition requires that each read scan segment is accessible in at least one intermediate scan configuration, and the target scan configuration c_t is finally reached.

6.2. Mapping to Pseudo-Boolean Optimization

This section presents a mapping of access time minimization to pseudo-Boolean optimization for a given (fixed) number of CSU operations. The pseudo-Boolean optimization problem is defined in Section 2.5 (p. 32). The search for the optimal number of CSU operations is addressed in the next section.

Let $\mathcal{M}$ be the CSU-accurate model of an RSN with the set of variables V, set of scan configurations C, and transition relation T. According to condition (6.1), an access $(c_0 \in C, c_t \in C)$ can be implemented within n CSU operations if and only if the following Boolean formula is satisfiable:

$$\texttt{Access}(c_0, c_t, n) := \Omega_0(V_0) \wedge \left[\bigwedge_{i=1\ldots n} \Omega_T(V_{i-1}, V_i) \right] \wedge \Omega_t(V_n), \tag{6.4}$$

where for $0 \leq i \leq n$, V_i denotes the set of variables for the i-th time step (scan configuration), such that each element in V_i corresponds to exactly one element in V ($\forall_{0 \leq i \leq n} \; |V_i| = |V|$), Ω_0 and Ω_t are the characteristic functions of scan configurations c_0 and c_t, respectively, and Ω_T is the characteristic function of the transition relation T.

Formula (6.4) is subject to pseudo-Boolean optimization with the following pseudo-Boolean cost function derived from (6.2):

$$\texttt{Cycles}(n) := D \cdot n + \sum_{i=0}^{n-1} \sum_{s \in S} |s| \cdot \big[\texttt{Active}(V_i, s) = 1\big]. \tag{6.5}$$

The *optimal assignment* that satisfies (6.4) and minimizes the cost function (6.5) is found with a pseudo-Boolean SAT solver. The assignment to variables in each set V_i defines the i-th scan configuration and is denoted by c_i. The consecutive scan configurations fully specify the scan-in sequence that implements the access (c_0, c_t). The length of this sequence is guaranteed to be minimal among all solutions with n CSU operations.

For the i-th CSU operation, the required scan-in sequence is derived from scan configurations c_{i-1} and c_i as follows:

- Configuration c_{i-1} specifies the active scan path: The order of scan segments on the active scan path is found by traversing the structural RSN model from the primary scan-input, through the selected scan segments, down to the primary scan-output. Scan segment $s \in S$ belongs to the active scan path during the i-th CSU operation if $\texttt{Active}(c_{i-1}, s) = 1$.
- Configuration c_i specifies the content of scan segments: The scan-in sequence for the i-th CSU operation is constructed by concatenating the data held by scan segments in configuration c_i in the order of the active scan path. For each inverter on the active scan path, all bits following the inverter's position are inverted in the scan-in sequence.

6.3. Pattern Generation Procedure

The challenge of access pattern generation consists in finding the optimal number of CSU operations required to perform an access with minimal access time. Let $n_{\texttt{min}}$ be the minimal number of CSU operations required to satisfy formula (6.4) and minimize (6.5). Often, the access time can be reduced by allowing *additional* CSU operations, as depicted in Figure 6.2.

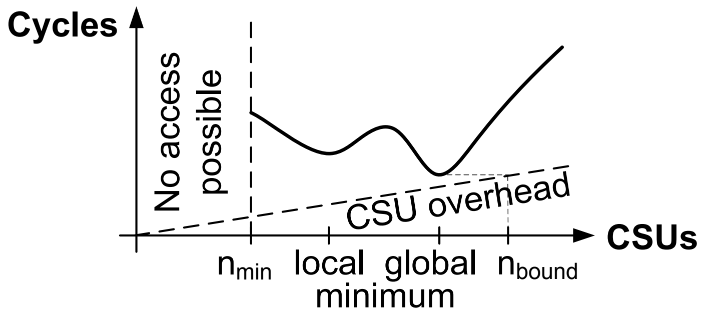


Figure 6.2.: Example of a minimal access time curve

Theorem 4. (Minimal Access Time) Let $D \in \mathbb{N}^+$ be the number of cycles required for the *capture* and *update* phases of a CSU operation (*CSU overhead*), and let $\texttt{Cycles}_i$ denote the access time with i CSU operations. A solution with $n \in \mathbb{N}$ CSU operations is the global minimum if the access time of all solutions with up to n_{bound} CSU operations is not less than $\texttt{Cycles}_n$, where:

$$n_{\text{bound}} := \lceil \texttt{Cycles}_n / D \rceil. \tag{6.6}$$

Proof. Due to the overhead of D cycles, a solution with i CSU operations requires at least $D \cdot i$ cycles (cf. formula (6.5) and curve “CSU overhead” in Figure 6.2). Thus, for every solution with n CSU operations, there exists an access pattern with n_{bound} CSU operations, for which the CSU overhead equals or exceeds $\texttt{Cycles}_n$, i.e.:

$$n_{\text{bound}} \cdot D \geq \texttt{Cycles}_n. \tag{6.7}$$

For all solutions with more than n_{bound} CSU operations, the access time is higher than $\texttt{Cycles}_n$. Therefore, it is sufficient to check the solutions with up to n_{bound} CSUs to decide whether the solution with n CSUs is the global minimum. According to (6.7), the minimal n_{bound} equals $\lceil \texttt{Cycles}_n / D \rceil$. □

According to Theorem 4, the global minimum can be found with an iterative procedure: Compute the shortest access patterns with $n = 1, 2, 3 \ldots$ CSU operations. For every pattern with n CSU operations, calculate n_{bound}. Terminate when an access pattern with n CSU operations is the shortest among all solutions with up to n_{bound} CSUs.

In practice, due to limited computational resources, the search for shortest solutions with up to n_{bound} CSU operations is often impossible. In contrast, the search for the first local minimum (cf. Figure 6.2) is more tractable. Below, an iterative pattern generation procedure is described that increases the number of allowed CSU operations as long as it leads to access time reduction.

Let $\texttt{Cycles}_n$ be the value of the cost function (6.5) after optimization with n CSU operations. Potentially, a solution with lower access time can be found if more CSU operations are allowed. The SAT instance is extended to $n+1$ CSU operations to find the value of the cost function $\texttt{Cycles}_{n+1}$. If the cost of the new solution is higher than the previous one, i.e. when $\texttt{Cycles}_{n+1} > \texttt{Cycles}_n$, a local minimum is found for n CSU operations and the pattern generation procedure terminates. Otherwise, the number of CSU operations is increased and the procedure is repeated until a local minimum is found or a user specified bound is reached.

Let n_t be the number of CSU operations for which the pattern generation procedure terminates at a local minimum. The procedure guarantees that the final solution has the minimal access time among all solutions with $n \leq n_t + 1$ CSU operations. Although there may exist a global minimum with lower access time that requires $n_{\texttt{opt}} > n_t + 1$ CSU operations, experimental results show that increasing the number of CSU operations beyond $n_t + 1$ rarely provides better results and leads to high solve times.

6.3.1. Implementation

As satisfiability solving is generally faster than pseudo-Boolean optimization, a SAT solver is used to find the minimal number of CSU operations that is required to perform the access ($n_{\texttt{min}}$). The iterative search for $n_{\texttt{min}}$ leverages incremental SAT solving techniques and follows the implementation of bounded model checking presented in Section 5.1.3 (p. 73).

After $n_{\texttt{min}}$ is found, pseudo-Boolean optimization for $n \geq n_{\texttt{min}}$ CSU operations is performed in parallel: A *parent* process is responsible for the generation of SAT instances with growing number of CSU operations. The optimization of each instance is performed in a parallel child process. For retrieval of optimal assignments, inter-process communication is implemented using POSIX *pipes*. Figure 6.3 illustrates the parallel execution of the pattern generation procedure.

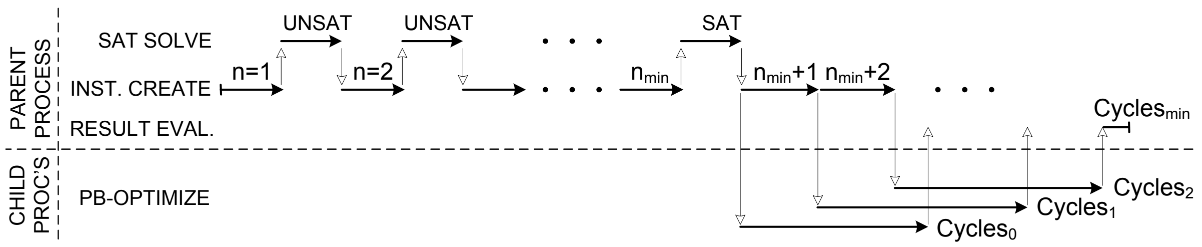


Figure 6.3.: Parallel execution of the pattern generation procedure

6.4. Experimental Evaluation

The proposed access optimization technique is evaluated in one thousand random experiments per benchmark circuit. In each experiment, the shortest pattern that *merges* read and write accesses to 10 randomly chosen scan segments is searched for. This section provides a brief summary of the achieved access time reduction. The benchmark circuits are discussed in Appendix A (p. 141). The experimental setup and detailed optimization results are found in Appendix C (p. 157).

Two series of experiments are performed: with the optimization effort limited to 2 and 20 s per access. Figure 6.4 presents the optimized *average* access time w.r.t. a pure SAT-based solution. Within 2 s of optimization effort, the average access time is nearly halved for most of the MUX-based RSNs. The access to SIB-based architectures can be optimized with a much simpler algorithm (cf. discussion on page 91). Nevertheless, compared with the SAT-based solution, the proposed optimization procedure reduces the access time to about 75% for the majority of SIB-based RSNs. In the flat scan architecture, the access time is halved in the best case. If the optimization procedure is allowed 20 s of time, the access time is reduced further by up to 7% for larger benchmarks.

Figure 6.5 presents the optimized access time for *outliers*, i.e., for access patterns with highest optimization potential. The access optimization procedure is allowed 2 s of time. The access time of outliers is reduced down to 50% in the SIB-based architecture, and to 30% in the flat scan architecture. For the majority of MUX-based benchmarks, the access time of outliers is reduced to 10%, and for the t512505 benchmark the optimized access time is below 1%. This shows that access optimization is crucial to prevent solutions with prohibitive access time and scan data volume.

6.5. Summary

Reconfigurable scan networks allow flexible and scalable access to on-chip infrastructure. However, the high complexity of RSNs arising from IP reuse and deep hierarchies necessitates the development of novel EDA tools for access scheduling and access time reduction.

This chapter maps the access time optimization problem to pseudo-Boolean optimization and solves it with efficient pseudo-Boolean SAT solvers. This novel method is applicable to a wide range of reconfigurable architectures and to *merging* of multiple concurrent scan accesses. For a given bound on the number of CSU operations, the proposed technique guarantees that the generated patterns have the *minimal* length. The experiments demonstrate that even for complex reconfigurable scan architectures the proposed method leads to significant reduction of access time by over 100x with low computational effort. The reduction of access time leads to a proportional reduction in scan data volume.

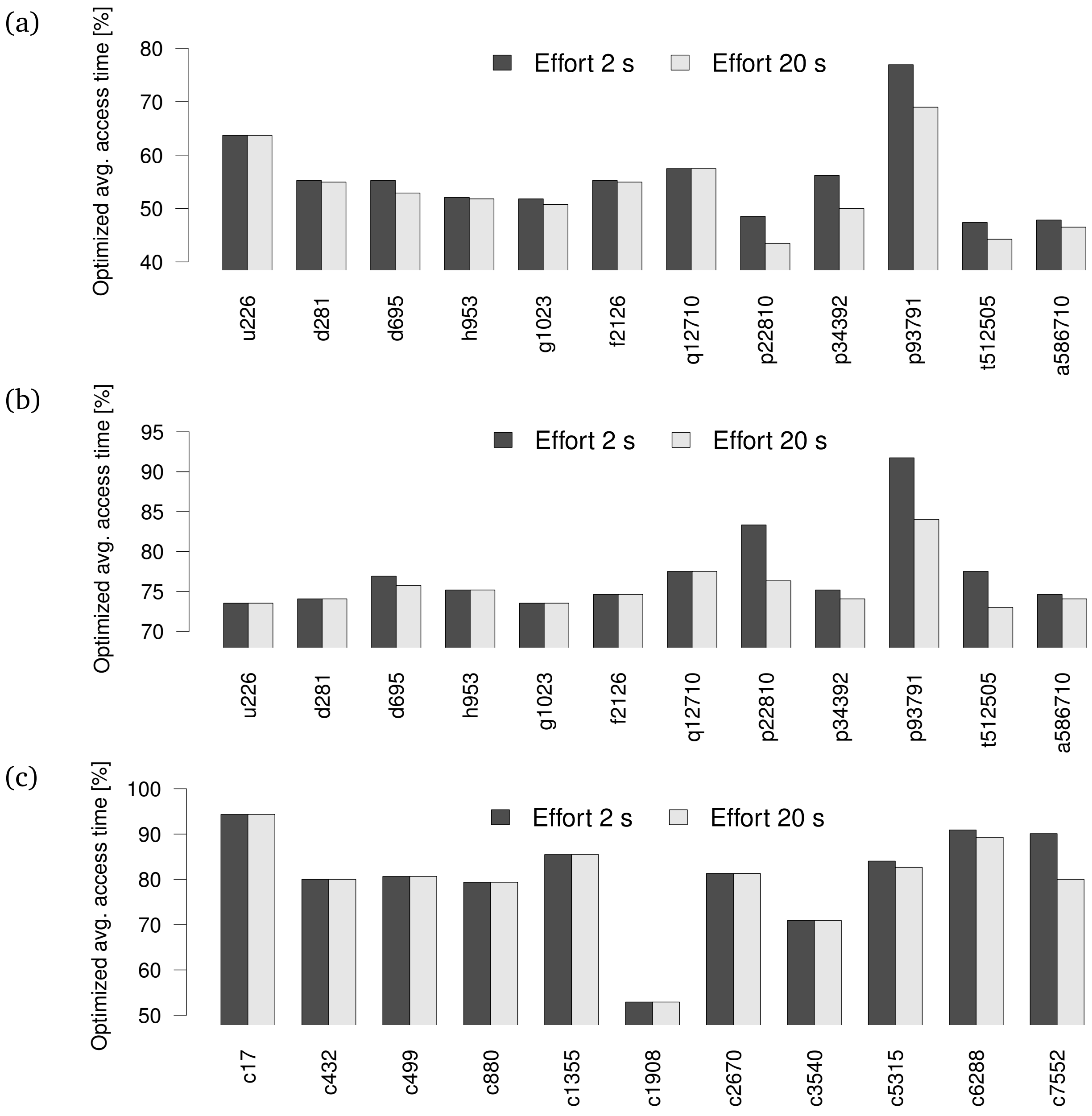
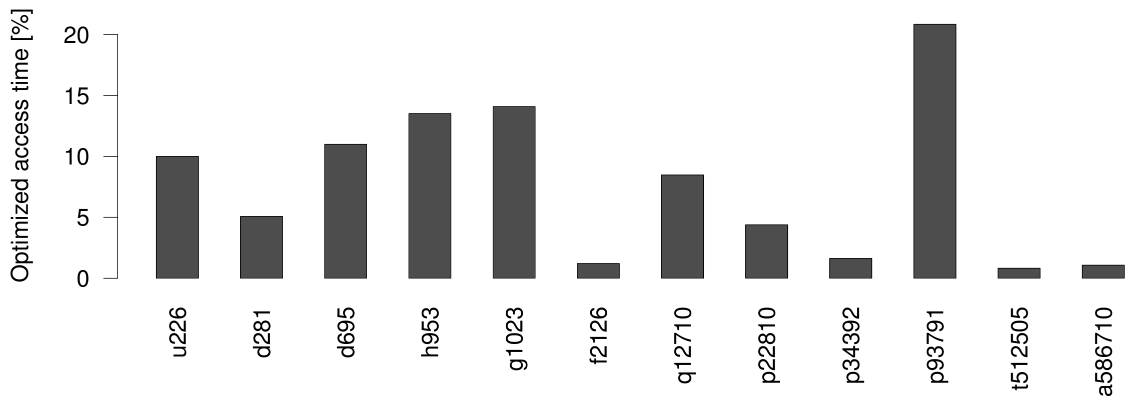
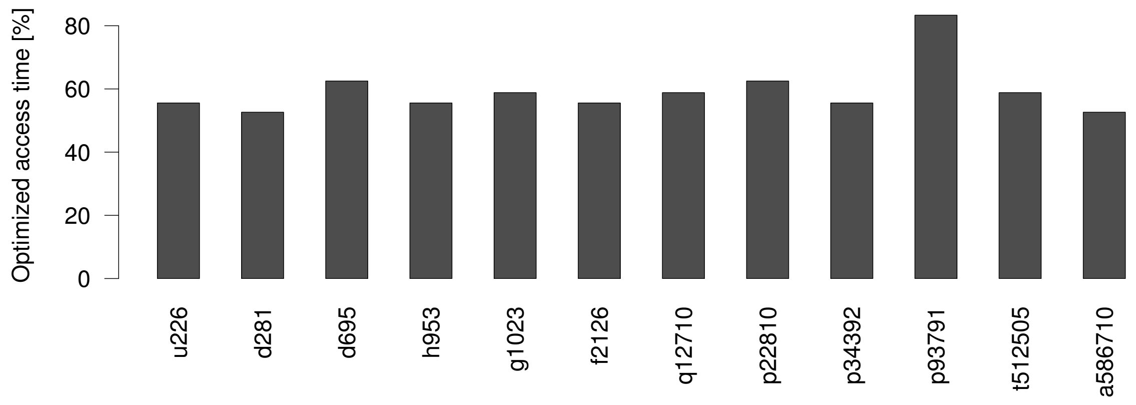


Figure 6.4.: Optimized average access time in (a) MUX-based, (b) SIB-based, and (c) flat scan architecture

(a)


(b)


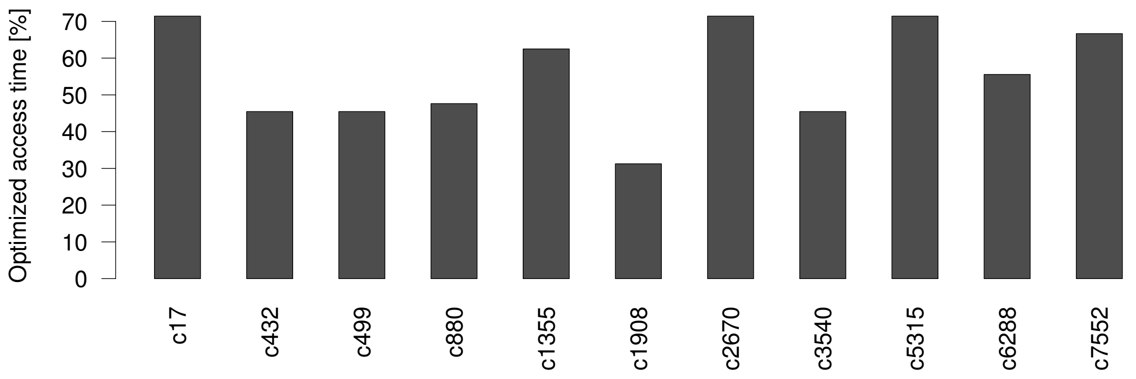
(c)


Figure 6.5.: Optimization of outliers in (a) MUX-based, (b) SIB-based, and (c) flat scan architecture

7. Access Port Protection

The accessibility of embedded instrumentation offered by reconfigurable scan networks poses a serious security threat. Protection against unauthorized access is crucial to security and safety of chip internals (see discussion in Section 1.5, p. 20).

The IEEE 1149.1 Test Access Port (TAP) can be protected using authorization mechanisms, scan data encryption, and access restriction techniques (see Section 3.5, p. 46). Such techniques can be directly applied to protect RSNs which are integrated as JTAG data registers (cf. Section 1.3, p. 14). With this approach, however, an RSN is protected as a whole, and fine-grained security control over individual scan segments is impossible.

State-of-the-art authorization mechanisms and access restriction techniques can be extended in a straightforward way to protect chosen RSN components. For instance, the *select* signals of protected segments can be gated by an authorization controller or an on-chip fuse. This solution, however, requires modification of the RSN design, needs additional global wires for security control, and requires consideration at early design stages.

This chapter presents a novel protection method that offers scalable, multi-level access management for RSNs. This protection technique is based on *sequence filters* that require no modification of the RSN and need no global wiring. The next section presents an example of access management in RSNs and provides an overview of the proposed protection method.

7.1. Access Management Overview

Figure 7.1 presents an example of a simple RSN. The one-bit scan segments `S1` and `S3` control the access to two multi-bit scan segments `S2` and `S4`, respectively. In the initial

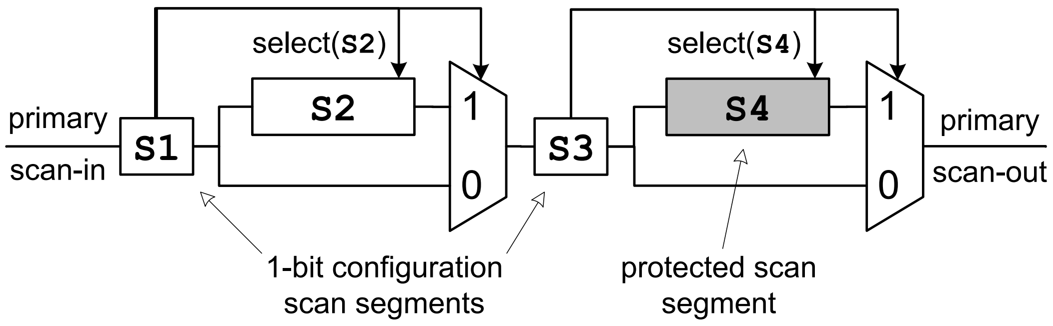


Figure 7.1.: Example of a reconfigurable scan network with a *protected* scan segment S4

scan configuration, it is assumed that S1 = S3 = 0, hence both S2 and S4 are initially bypassed. The access to scan segment S2 is allowed, while S4 is *protected*, i.e., S4 must not be accessible from the primary scan-input.

An access to the RSN is called *restricted* if it does not put any protected scan segment on the active scan path (formal definition is given in Section 7.2). For instance in Figure 7.1, restricted access to the target scan segment S2 must ensure that S4 is bypassed at all times. To this end, S3 must always be loaded with 0.

The aim of the proposed security management method is to prevent access to protected scan segments by allowing restricted accesses only. This goal is achieved with a *sequence filter* that is placed between the TAP and the scan network, as shown in Figure 7.2. The filter observes the sequence of scan operations (*capture*, *shift*, *update*) and the scan data at the TDI port to decide whether the access pattern is allowed or forbidden. If the scan operations do not expose any protected scan segment, the filter does not interfere with the access. Otherwise, the filter inhibits the *update* operation and so prevents all RSN registers from latching any data that could expose or give access to any protected scan segment.

Figure 7.3 presents an example for *multi-level* access management in a System-on-a-Chip (SoC) design using two sequence filters, F1 and F2. Filter F1 restricts the external accessibility of debug instrumentation, e.g. for IP protection. F2 blocks the internal accessibility of embedded test instruments at the TAP of “Core 1”, e.g. due to safety requirements for in-field operation. Still, full internal accessibility is preserved for debugging purposes via the internal TAP of “Core 2”.

The proposed protection method requires only a minor extension of the TAP, which

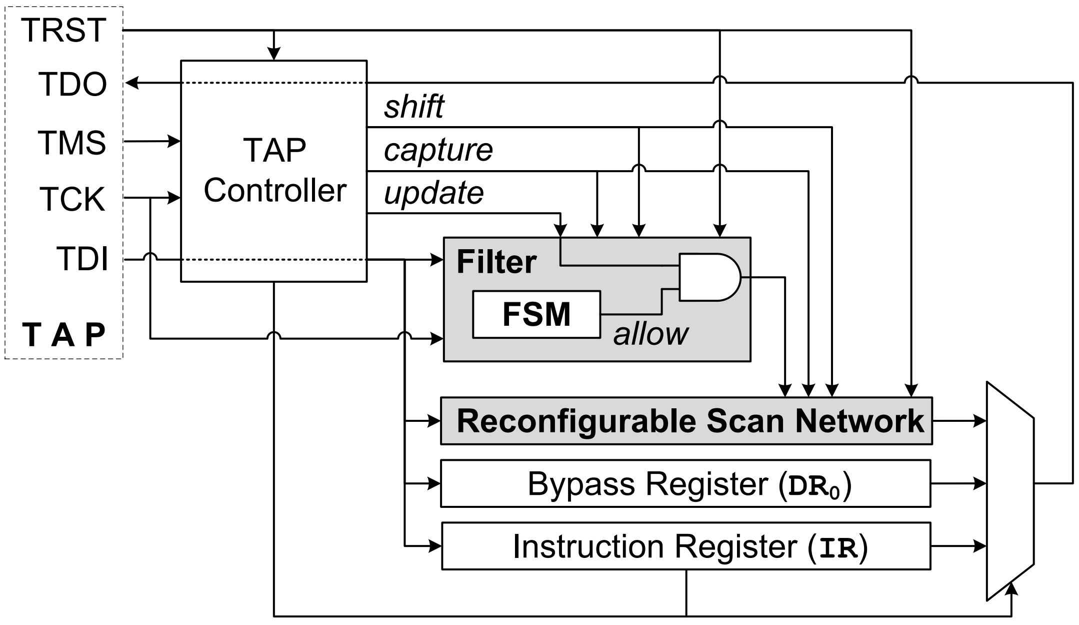


Figure 7.2.: Access protection using a *sequence filter* at the Test Access Port (TAP)

changes neither the internal nor the external TAP interface. In particular, sequence filters require no modification of the RSN architecture and no global signals for security control. This makes this approach well-suited for 3D integrated circuits and core-based designs with hard macro IPs.

A sequence filter can be activated by a single fuse, e.g. after manufacturing test. Moreover, this approach can be combined with authorization mechanisms such as [Buskey06, Clark10] to provide logical security of individual scan segments without the need to redesign the scan network. Sequence filters can also be used to allow individual (exclusive) access to a set of instruments, and still block simultaneous (concurrent) access to them, e.g. to prevent that sensitive data are shifted through exposed or untrusted instruments.

An overview of the proposed method is presented in Figure 7.4. The restricted access patterns are generated in an automated way for a given set of target and protected scan segments, which is discussed in Section 7.2. The restricted patterns are fed to the filter construction algorithm presented in Section 7.3. The area overhead of the proposed filter-based protection is evaluated in Section 7.4 and Appendix D.

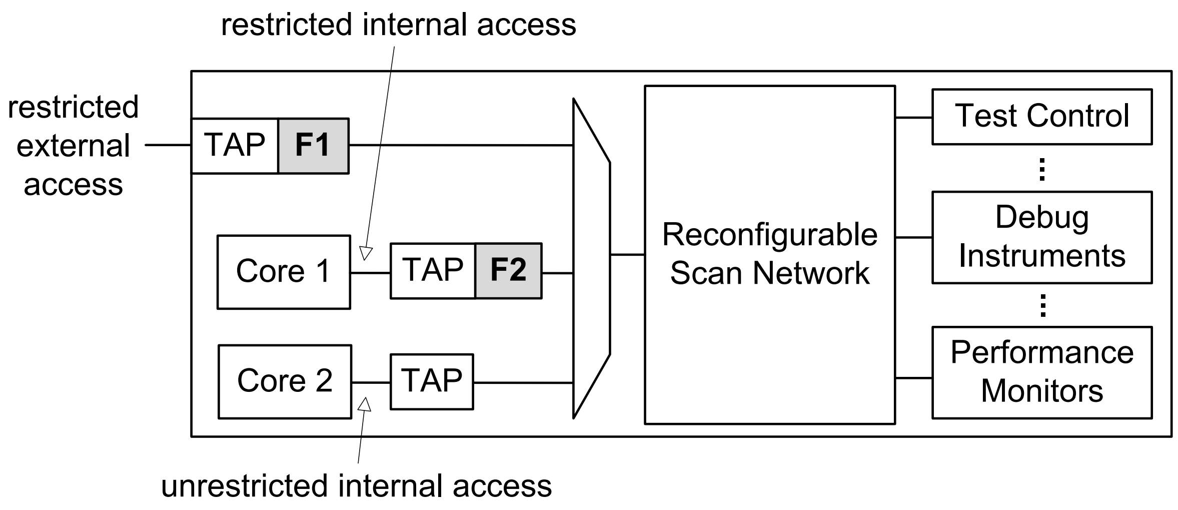

restricted internal access
restricted external access
TAP
F1
Core 1
TAP
F2
Core 2
TAP
unrestricted internal access
Reconfigurable Scan Network
Test Control
Debug Instruments
Performance Monitors

Figure 7.3.: Example of a multi-level access protection based on *sequence filters* (F1, F2)

7.2. Generation of Restricted Access Patterns

In the following, the restricted access patterns are defined formally based on the CSU-accurate model from Section 4.4.1 (p. 60).

Definition 18. (Restricted Access Pattern) Let $\mathcal{M}$ be the CSU-accurate model of an RSN with the set of scan segments S, set of scan configurations C, and initial (reset) scan configuration $c_0 \in C$. Given a set of *protected* segments $S_P \subset S$ and a set of initial scan configurations $I \subseteq C$ such that $c_0 \in I$, an access pattern for *target* scan segments in $S \setminus S_P$ is *restricted* if it fulfills all of the following conditions:

- The target segments are properly accessed for all initial scan configurations in I.
- During the access, no protected scan segment from S_P belongs to the active scan path (the scan data do not pass through any protected scan segment).
- For all initial scan configurations in I, the scan configuration after the access belongs to set I.

Remark 6. Since the final scan configuration after a restricted access belongs to the set of initial scan configurations I, it follows that any concatenation of restricted accesses is also a restricted access.

Restricted access patterns with minimal access time are generated in an automated way using the procedure presented in Section 6.3 with a modified SAT instance: For

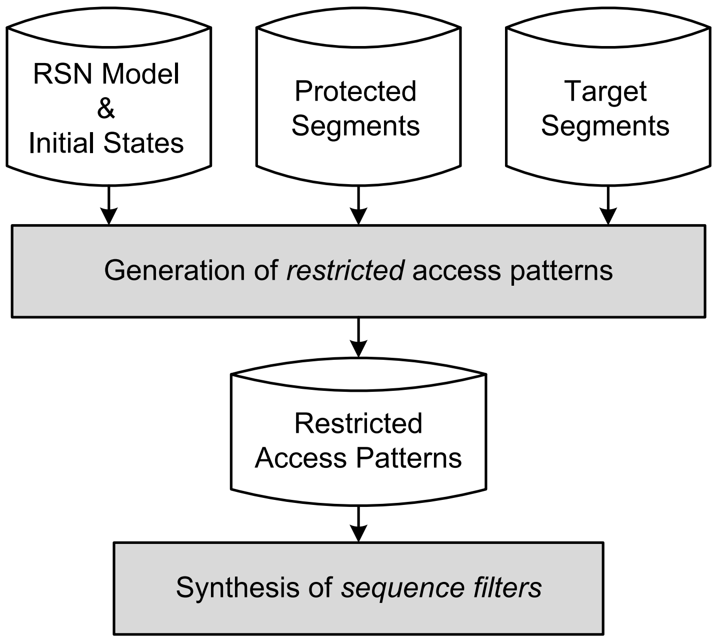

RSN Model & Initial States
Protected Segments
Target Segments
Generation of *restricted* access patterns
Restricted Access Patterns
Synthesis of *sequence filters*

Figure 7.4.: Overview of the proposed method

n CSU operations, the SAT instance representing a restricted access is constructed as follows:

$$\begin{aligned}\texttt{Access}(n) := \Omega_I(V_0) \wedge \left[\bigwedge_{i=1\ldots n} \Omega_T(V_{i-1}, V_i)\right] \wedge \Omega_I(V_n) \wedge \\ \left[\bigwedge_{i=0\ldots n} \bigwedge_{s \in S_P} [\texttt{Active}(V_i, s) = 0]\right] \wedge \Omega_R(V_0, V_1, \ldots, V_n),\end{aligned} \tag{7.1}$$

where for $0 \leq i \leq n$, V_i denotes the set of variables for the i-th scan configuration, Ω_R represents access constraints for the target scan segments in the final and/or intermediate scan configurations, while Ω_I and Ω_T are the characteristic functions of the set of initial scan configurations I and the transition relation of $\mathcal{M}$, respectively. This instance is satisfiable if and only if there exists a restricted access with n CSU operations such that target scan segments are properly accessed (Ω_R is satisfied), protected scan segments in S_P never belong to the active scan path (their content is never altered nor exposed), and the initial scan configuration I is restored.

The restricted access generation method poses two requirements on the RSN: In the initial (reset) scan configuration, no protected scan segment may belong to the active scan path, and there must exist a way to bypass all protected scan segments while accessing target scan segments. If the access to a target segment requires that any protected scan segment be modified or exposed, the protected segment needs to be extended with a configurable bypass that is initially active, e.g. a Segment Insertion Bit (SIB) [Stollon11].

Restricted Access Example

In the RSN from Figure 7.1 (p. 104), the access to scan segment S2 is allowed, while segment S4 is protected. Assume that I is defined as the set of all scan configurations in which $\texttt{S1} = \texttt{S3} = 0$. According to Definition 18, a restricted access to S2 must guarantee that:

- S2 is accessed for all initial scan configuration satisfying $\texttt{S1} = \texttt{S3} = 0$ (regardless of the content of S2 and S4).
- S4 is never part of the active scan path.
- After the access, the initial scan configuration is restored, i.e. $\texttt{S1} = \texttt{S3} = 0$.

A possible restricted access pattern for segment S2 consists of two CSU operations with the following scan data (leftmost bit is shifted first): *01* and *0X0*, where X stands for the target value of S2. The first CSU operation puts segment S2 on the active scan path by setting S1 to 1. In the second CSU, S2 is accessed and the initial state of S1 is restored. During the two CSU operations, the protected segment S4 is bypassed. After the access, the final scan configuration satisfies $\texttt{S1} = \texttt{S3} = 0$.

7.3. Synthesis of Sequence Filters

A sequence filter consists of a Finite State Machine (FSM) that receives the scan data input (TDI) of the TAP, as well as the *capture*, *shift*, and *update* control signals driven by the TAP controller (cf. Figure 7.2, p. 105). The state diagram of the filter's FSM is constructed directly from a set of user-defined restricted access patterns, as described in the later part of this section. The FSM tracks scan operations at the TAP and generates a single output *allow* which controls the *update* operation in the RSN: As long as the sequence of scan operations matches any allowed restricted access, the *allow* signal is active and the access is applied to the RSN without any delay. Otherwise, *allow* is deactivated and the FSM enters a *trap* state. In the trap state, no further reconfiguration of the RSN is allowed, and hence no access to protected scan segments is possible.

The state of the filter's FSM must be synchronized with the scan configuration of the protected RSN: The reset signal must reliably put both the RSN and the sequence filter to their initial states. If the RSN is accessed through another TAP (e.g. via an internal interface), the sequence filter is put into the *trap* state. This assures that no forbidden access can take place when the sequence filter is not synchronized.

To guarantee security in presence of soft errors and hardware defects, the FSM can be designed fail-safe [Nicolaidis89]: In presence of faults, the FSM's output *allow* must be either correct or inactive (0). For instance, to protect against single faults, the FSM is duplicated and the *allow* outputs are used as a dual-rail encoded signal, or conjoined with an AND gate.

7.3.1. State Diagram Construction

Procedure 1 presents the state diagram construction algorithm for sequence filters. The input to the procedure is a set of sequences (strings) representing restricted accesses patterns that the filter should allow (`sequenceSet`). The input sequences are composed of five scan operations denoted as follows:

- *0*: shift of bit 0,
- *1*: shift of bit 1,
- *X*: shift of an unconstrained (*don't care*) bit,
- *C*: capture,
- *U*: update.

For instance, a restricted access consisting of two CSU operations with scan data *01* and *0X0* is represented by the following sequence: *C01UC0X0U*. Note that a single sequence represents 2^k restricted access patterns, where k is the number of unconstrained data bits (*X*) in the sequence.

The diagram construction algorithm starts with the creation of an "initial" state (`initialState`) that corresponds to the set of initial scan configurations, and a "trap"

Algorithm 1 Sequence filter construction

```
Input: sequenceSet
Output: state diagram
 1: Create initialState, trapState.
 2: Annotate initialState with all sequences from sequenceSet.
 3: currentStateSet $\leftarrow$ {initialState}
 4: $n \leftarrow 0$
 5: while currentStateSet $\neq \varnothing$ do
 6:     nextStateSet $\leftarrow \varnothing$
 7:     for all state $\in$ currentStateSet do
 8:         for all sequence $\in$ annotations of state do
 9:             transition $\leftarrow$ sequence$[n]$
10:             if transition $= U$ and length(sequence) $= n + 1$ then
11:                 Add transition from state to initialState.
12:             else
13:                 Create newState and annotate it with sequence.
14:                 Add transition from state to newState.
15:                 Add newState to nextStateSet.
16:             end if
17:         end for
18:     end for
19:     Replace overlapping transitions from states in currentStateSet.
20:     Add escape transitions from states in currentStateSet to trapState.
21:     Merge equivalent states in nextStateSet.
22:     currentStateSet $\leftarrow$ nextStateSet
23:     $n \leftarrow n + 1$
24: end while
25: Collapse state sequences with equivalent outbound transitions.
```

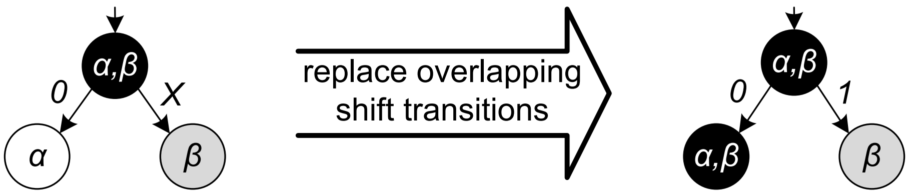


Figure 7.5.: Example of a state diagram before and after replacement of overlapping shift transitions. Annotations α and β denote two sequences.

state (`trapState`) that is reached upon detection of any forbidden scan operation (line 1 in Procedure 1). Each state in the state diagram is annotated with the sequences that put the FSM into this state. State transitions are conditioned either by a single scan operation (i.e. an element from the set {*0*, *1*, *X*, *C*, *U*}), or a disjunction of scan operations (e.g. *C* or *U*, denoted as *C,U*). All states are stable as long as no scan operation takes place.

The construction algorithm is a stepwise procedure (lines 5 to 24): In the first step, the first scan operation of each sequence is processed (i.e., the capture operations). In the n-th step, another level of states is added to the state diagram based on the n-th scan operations of the the provided sequences (lines 13 to 15). The current scan operation in each sequence is assigned a new successor state (`newState`) with an incoming transition from the respective state in `currentStateSet`. Since any concatenation of restricted accesses is also a restricted access (cf. Remark 6, p. 106), the last *update* operation in each sequence corresponds to a transition to the initial state (line 11). The procedure terminates when all sequences are completely processed.

In each step, after the successor states are found, overlapping shift transitions of each current state are replaced (line 19): If a state has both an outbound *X* transition and an outbound *0* (*1*) transition, the *X* transition is replaced with a *1* (*0*) transition, and the annotations of both successors are updated accordingly. An example is given in Figure 7.5.

After execution of line 19, if all scan operations are allowed in a state from `currentStateSet`, this state has either 3 or 4 outbound transitions, conditioned by either *C*, *U*, and *X* or *C*, *U*, *0*, and *1*. If some operations are *forbidden* (not allowed by any provided sequence), the sequence filter must detect them and prevent any further reconfiguration of the network. To this end, an *escape transition* pointing to the `trapState` is added for the forbidden operations (line 20). Once a forbidden operation is encountered, the filter is stuck in the `trapState`. In this state, the update operation is inhibited until the sequence filter and the scan network are reset.

7.3.2. State Merging and Sequence Collapsing

To reduce the size of the state diagram, redundancies are removed by *merging* equivalent states (line 21 in Procedure 1) and *collapsing* sequences of states with equivalent outbound transitions (line 25), as described below.

Each pair of successor states in `nextStateSet` is *merged* into a single state if it fulfills one of the following conditions:

- The two states have identical annotations (belong to the same sequences).
- The inbound transitions of the two states have the same condition, and their predecessors have the same annotations.

A state that results from merging of two states receives all annotations of its constituent states.

The resulting state diagram often includes long sequences of consecutive shift operations with constant or unconstrained (*X*) bits (see example in Figure 7.7a). Typically, long sequences of *X* operations represent unconstrained data for scan segments that do not control the active scan path. Such sequences are *collapsed* into a single state, and a counter is used to keep track of their length, as shown in Figure 7.6. During a transition to a collapsed state, the counter is set to the number of states that were removed due to collapsing (via the *value* signal; by asserting the *load* signal). The counter is decremented upon detection of every shift transition (via the *decrement* input) and asserts its *wait* output as long as its value is larger than zero. The FSM leaves the collapsed state as soon as the *wait* signal is deasserted or a forbidden operation (*C* or *U*) is detected. Just a single counter is required regardless of how many sequences are collapsed.

Figure 7.7 presents an example for collapsing the sequence *XXX1*. The states b, c, d in Figure 7.7a are collapsed into a single state m in Figure 7.7b. During the transition to the collapsed state m, the counter is set to 2. The counter is decremented upon every shift operation (*X*). The final state e is reached as soon as the *wait* signal is deasserted and the final scan operation is correct (*1*). Otherwise, the trap state is reached.

The final state diagram can be further optimized to allow repeated accesses to a set

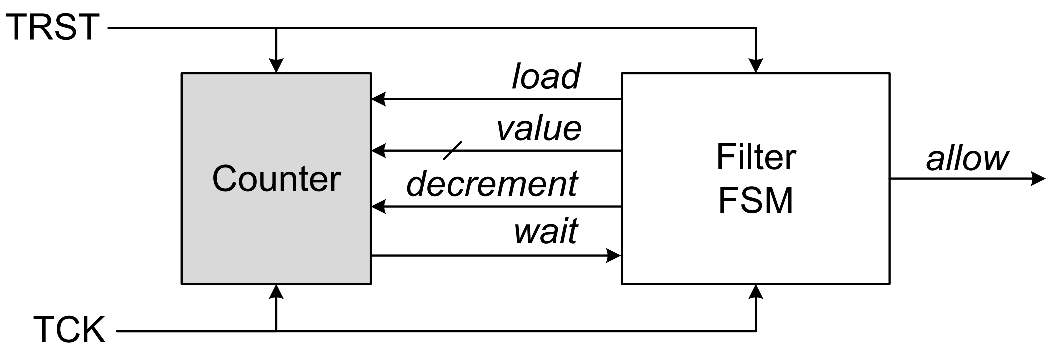


Figure 7.6.: Sequence filter augmented with a counter for collapsed states

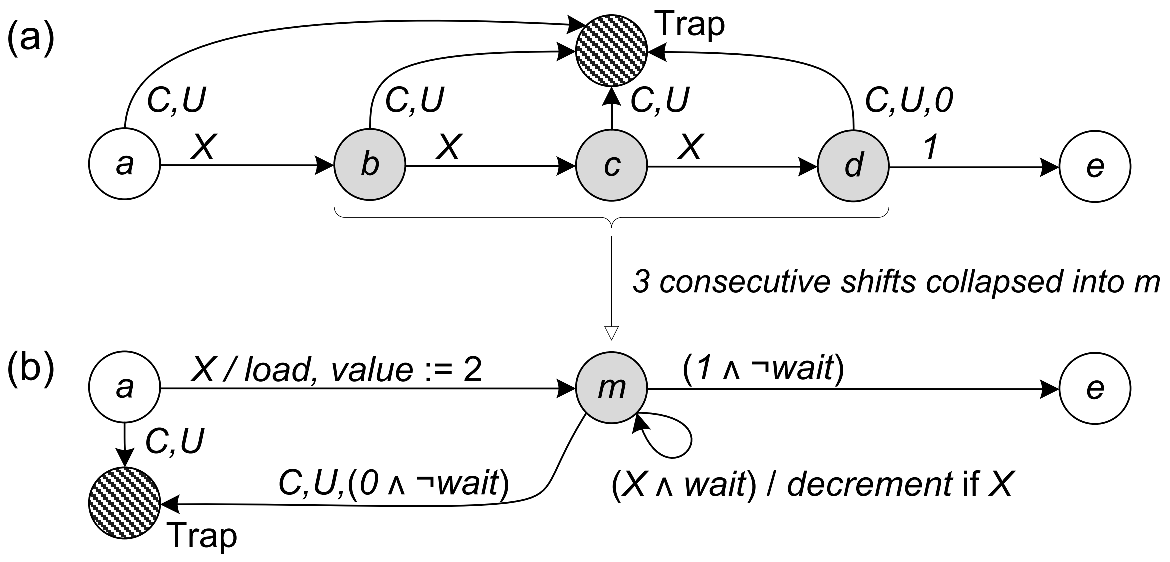


Figure 7.7.: Example for sequence collapsing with (a) a state diagram and (b) its collapsed equivalent

of target scan segments with little or no hardware overhead. This is crucial to apply many patterns to a set of scan segments with no access time penalty for scan path reconfiguration. To this end, just two repeated accesses must be reflected in the input sequence, such that the first access does not modify the active scan path. The resulting state diagram is then extended with a loop transition for the first access, which enables an unlimited number of repeated accesses. This is explained at an example in the following section.

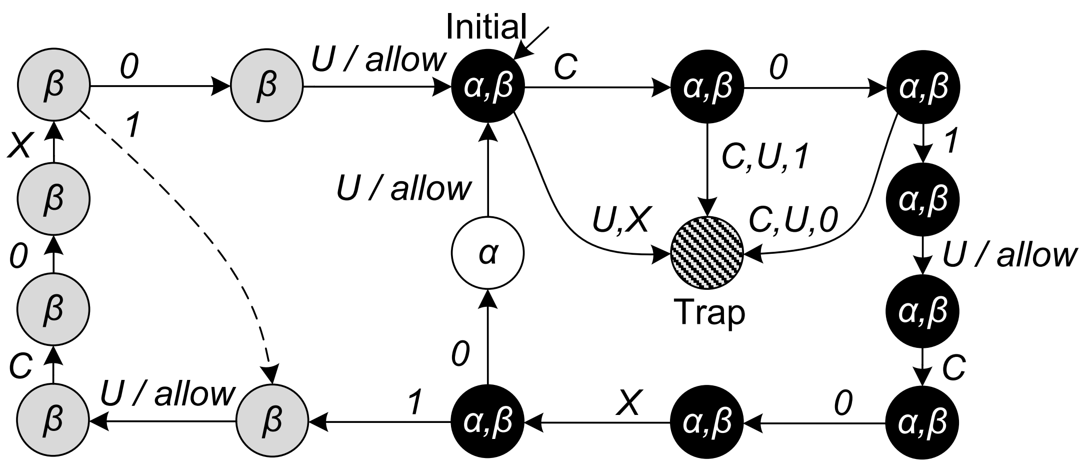


Figure 7.8.: The state diagram of a sequence filter allowing two sequences: α: *C01UC0X0U*, and β: *C01UC0X1UC0X0U*

7.3.3. Sequence Filter Example

In the following, sequence filter construction is illustrated at an example of the RSN from Figure 7.1 (p. 104). The filter is constructed for two restricted accesses patterns characterized by the following sequences α and β:

- α: *C01UC0X0U*, which accesses S2 once (as in Section 7.2, p. 106),
- β: *C01UC0X1UC0X0U*, which accesses S2 twice.

Such sequences are found in an automated way using the approach presented in Section 7.2 (p. 106).

Figure 7.8 presents the state diagram produced for the sequences α and β by Procedure 1 (p. 110). The annotations of states are denoted inside the state symbols (α and β). For the sake of clarity, the escape transitions to the trap state are shown only for the first three states.

The filter tracks the scan operations and the scan data at the TAP. As long as the sequence matches either α or β, the update operations are allowed. Otherwise, the trap state is reached, in which no further reconfiguration of the network is possible, and hence the protected segment S4 is inaccessible.

The filter can be extended with a single loop transition to allow repeated accesses to S2 without the need to reconfigure S1. This transition is dashed in Figure 7.8.

7.4. Experimental Evaluation

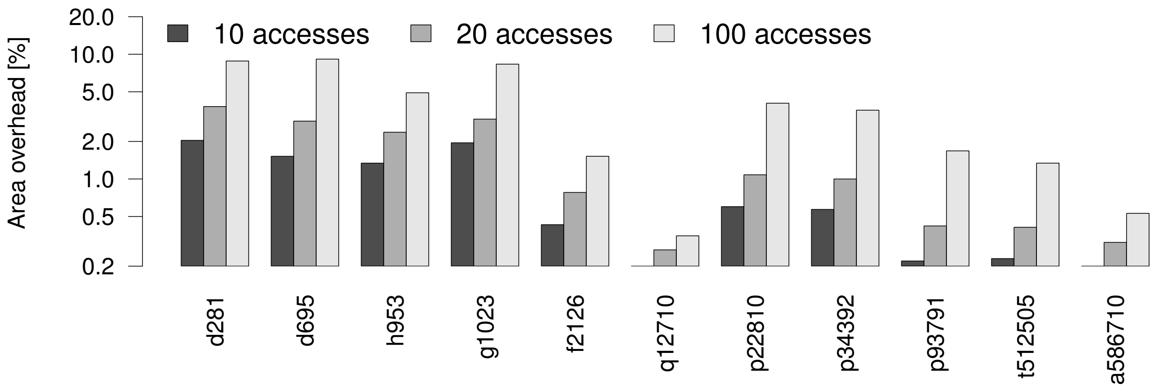
The proposed protection method is evaluated on SIB- and MUX-based scan architectures presented in Appendix A (p. 141). Restricted accesses are generated for random samples of target scan segments. Except for scan segments that configure the active scan path, all remaining scan segments of a benchmark RSN are considered protected. This section provides a brief summary of the area overhead required for the protection w.r.t. RSN area without system logic. The experimental setup and detailed results are found in Appendix D (p. 161).

Figure 7.9 presents the area overhead of sequence filters constructed for 10, 20, and 100 restricted accesses patterns. Each pattern implements the shortest restricted access to a single target scan segment. The area overhead is below 2.7% for 10 patterns, 4.3% for 20 patterns, and rises up to 10.6% for 100 patterns. Note that some of the RSNs, e.g. f2126, q12710, and a586710, include less than a hundred scan segments (see Appendix A). For these benchmarks, even if access to a high fraction or all of their scan segments is allowed, the area overhead is below 1.7%.

The size of a sequence filter is proportional to the number of states in the filter's state diagram. State merging and sequence collapsing (see Section 7.3.2, p. 112) considerably reduce the area overhead. Figure 7.10 shows the cumulative length of 100 restricted access patterns ("sequence bits") and the corresponding number of filter's states after state merging and sequence collapsing ("FSM states"). These techniques reduce the size of the state diagram by a factor of 2 at least, and by over 2 orders of magnitude for two benchmarks: q12710 and a586710.

In the second series of experiments, sequence filters are constructed for the concurrent access to 100 random scan segments realized by 1, 5, 10 and 20 restricted access patterns. Figure 7.11 shows area overhead of the resulting filters. For 20 accesses à 5 segments ("20 à 5"), area overhead of the resulting filters is close to the area for individual accesses ("100 à 1"). However, if the access to all 100 segments is realized with a *single* access pattern ("1 à 100"), the cost is reduced by a factor of 3 to 16 compared with the cost of individual accesses. If the segments are often accessed together, concurrent access has two benefits: The access times are lower, and the resulting sequence filters are smaller.

(a)



(b)

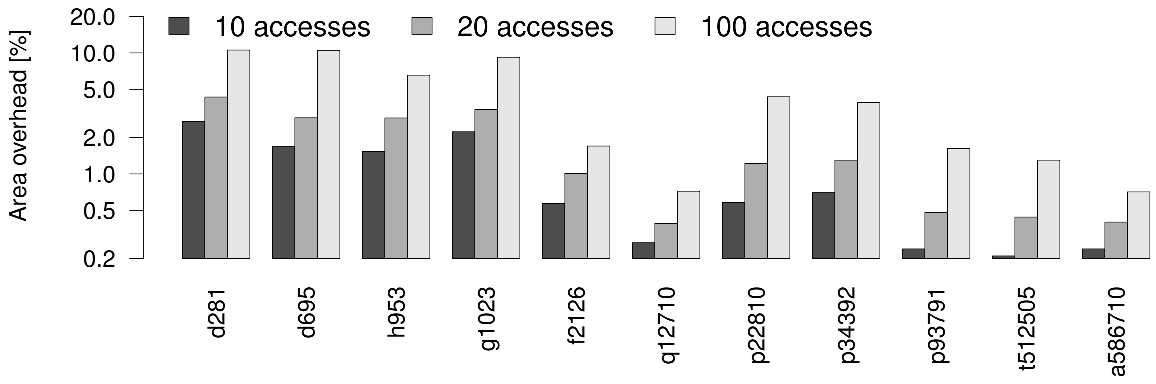


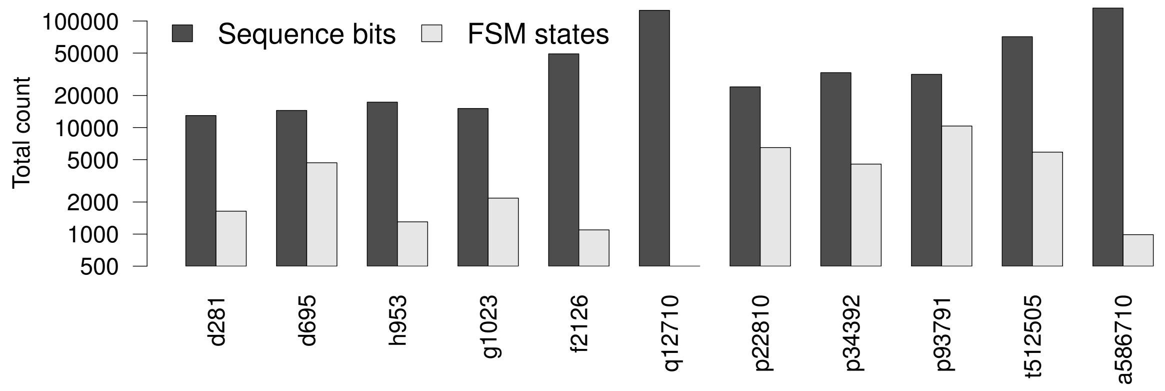
Figure 7.9.: Area overhead of sequence filters w.r.t. RSN area for (a) SIB-based and (b) MUX-based scan architecture

7.5. Summary

To guarantee secure chip development and safe in-field system operation, embedded instrumentation requires protection against unauthorized access. While state-of-the-art techniques provide effective protection for JTAG circuitry, reconfigurable scan networks call for dedicated access management methods with fine-grained control over the security of their constituent scan segments.

The proposed access management technique secures reconfigurable scan networks at the Test Access Port (TAP) and facilitates fine-grained control over the access to individual scan segments. The TAP is extended with a sequence filter that permits only a set of access patterns that are defined at design time. If required, the filter can be enabled by a single fuse, e.g. after manufacturing test, or disabled by an authorization controller. This approach is directly applicable to scan networks compliant with IEEE Std. 1149.1-2013 (JTAG) and P1687 (IJTAG).

(a)



(b)

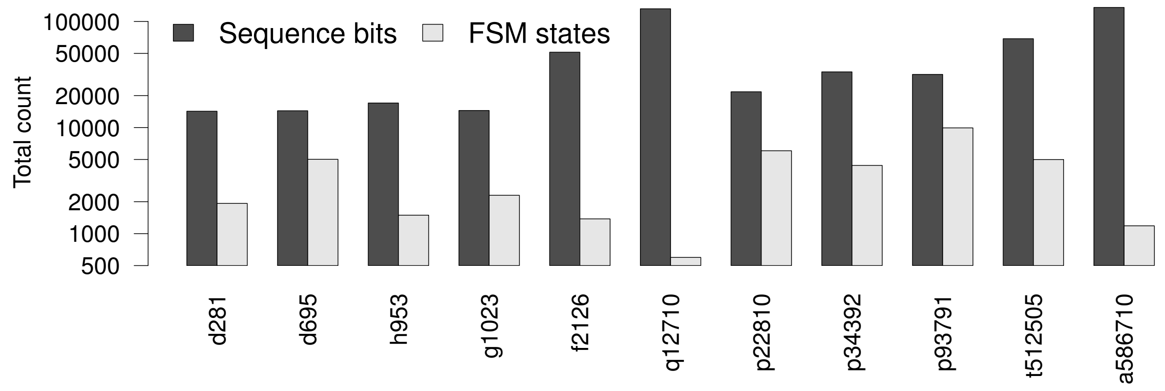
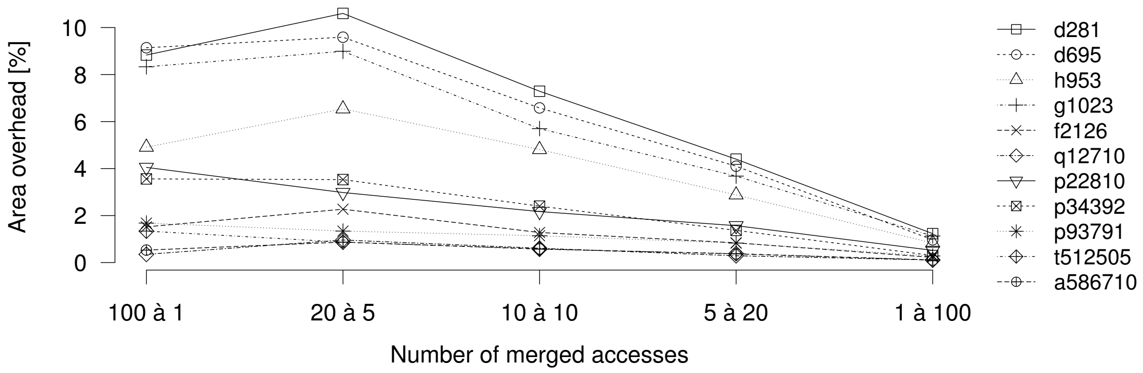


Figure 7.10.: Comparison of the total sequence length (in bits) and the number of FSM states after state merging and sequence collapsing for 100 restricted access patterns in (a) SIB-based and (b) MUX-based scan architecture

The sequence filters do not affect the access time and do not require any modification of the RSN design. Since no additional global wiring is required, this protection technique is well-suited for core-based designs and 3D integrated circuits. Experimental results show that on average, to assure security of designs with over 10,000 scan cells and retain the accessibility of 100 scan segments, the proposed approach increases the area of scan infrastructure by less than 5%, which is marginal with respect to the total chip area.

(a)



(b)

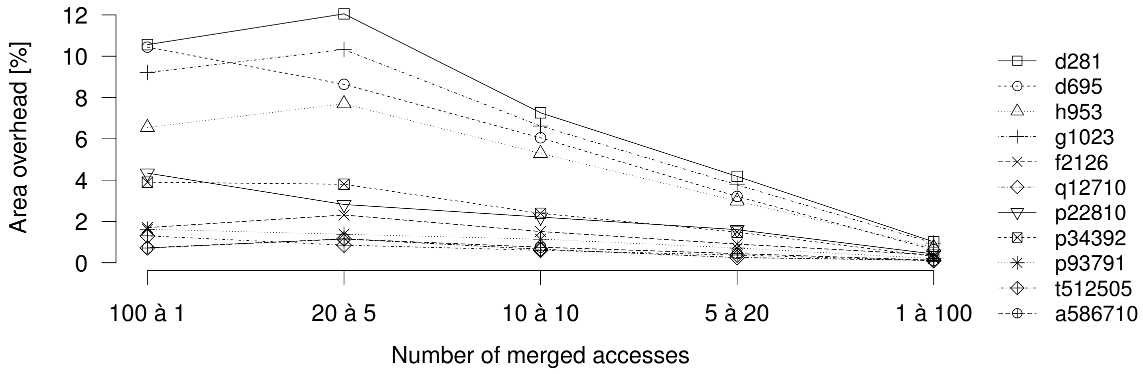


Figure 7.11.: Reduction of sequence filter overhead by merging the access to 100 scan segments in (a) SIB-based, and (b) MUX-based scan architecture

8. Conclusions

The amount of embedded instrumentation in system-on-a-chip designs increases at an exponential rate. Such structures serve various purposes throughout the life-cycle of VLSI circuits, e.g. in post-silicon validation and debug, production test and diagnosis, as well as during in-field test and maintenance. Reliable access mechanisms for embedded instruments are therefore key to rapid chip development and secure system maintenance.

Reconfigurable scan networks defined by IEEE Std. P1687 emerge as a scalable and cost-effective access medium for on-chip instrumentation. However, due to complex combinational and sequential dependencies, such reconfigurable architectures are beyond the capabilities of state-of-the-art algorithms for formal verification and access scheduling.

This thesis contributes a novel CSU-accurate modeling method based on temporal abstraction. The proposed abstraction improves the scalability of model checking algorithms in verification of complex reconfigurable scan networks. A time step in the abstract model corresponds to a full CSU operation that spans multiple clock cycles required for capturing, shifting, and updating scan data. Under minor assumptions on the stability of external signals, the CSU-accurate abstraction is *sound*, i.e., the properties of the abstraction are guaranteed to hold in the concrete RSN implementation. Experimental results show that the CSU-accurate model reduces the formal verification effort tremendously.

The investigation of *robust* scan architectures shows that this class of RSNs has many advantageous properties, such as improved verifiability and reduced vulnerability to defects. Robust scan networks can be verified using bounded model checking techniques with a tractable completeness threshold. Moreover, the CSU-accurate model is proven *complete* in the class of robust RSNs, i.e., it does not produce spurious counterexamples. The verification of the robustness property itself uncovers design bugs with high probability and is efficiently mapped to Boolean satisfiability.

For the generation of low-latency access patterns (*pattern retargeting* in the terminology of IEEE Std. P1687), this thesis develops the first automated algorithm that can handle complex reconfigurable scan networks. This method leverages existing techniques for pseudo-Boolean optimization to perform access time minimization and *merging* of concurrent accesses to multiple instruments. Experimental results show that this method effectively reduces the reconfiguration overhead and prevents solutions with prohibitive access time.

The accessibility offered by reconfigurable scan networks contradicts security and safety requirements for embedded instrumentation. Since RSNs have distributed configuration and integrate a high number of instruments, state-of-the-art techniques for scan access protection are either ineffective or offer only coarse-grained security control. This thesis presents a novel access protection method which requires only a *local* extension of the access port. The protected access port allows a user-defined set of access patterns and prevents the access to protected instrumentation. This approach provides fine-grained access management with low area overhead and can be combined with existing fuse- and authorization-based protection schemes.

8.1. Future Research Directions

While scan networks support system test and diagnosis, they are themselves vulnerable to defects. The test and diagnosis of complex reconfigurable scan networks seem to be the most important direction for future research. Test of RSNs is challenging due to high sequential depth, complex access dependencies, and the interdependence between the RSN, on-chip instrumentation, and mission logic. Faults in the scan network can affect the configuration of the active scan path, leading to diagnostic difficulties which are further exacerbated by limited observability.

The CSU-accurate abstraction can be extended to facilitate test pattern generation for accurate fault models, e.g. at gate-level. While low-level fault activation and propagation conditions must be modeled at a cycle-accurate level, pattern delivery and response readout can be handled efficiently with the CSU-accurate abstraction. Therefore, the challenge consists in combining models at different abstraction levels into a single SAT instance.

For post-silicon debug and production ramp-up, the accessibility of the scan infrastructure is crucial to locate design bugs and defects. A single bug or defect may affect the access to a large fraction of on-chip instruments, making system debug and diagnosis difficult or impossible. Nevertheless, even if the scan infrastructure is partially defective, the remaining accessibility can still be utilized for chip diagnosis. Diagnostic algorithms that narrow down the faulty RSN region or determine the remaining functionality constitute a challenging yet interesting research field. The CSU-accurate access generation methods can potentially be extended for the generation of robust access patterns that bypass an identified faulty region.

The CSU-accurate modeling can be also extended to support the verification of interactions between system logic and the RSN. In particular, to formally prove security of the protection method developed in this thesis, sequence filters can be modeled in a CSU-accurate way.

Bibliography

[Abramovici06] M. Abramovici, P. Bradley, K. N. Dwarakanath, P. Levin, G. Memmi, and D. Miller. A Reconfigurable Design-for-Debug Infrastructure for SoCs. In *Proc. Design Automation Conference (DAC)*, pages 7–12. 2006.

[Abramovici08] M. Abramovici. In-System Silicon Validation and Debug. *IEEE Design & Test of Computers*, 25(3):216–223, 2008.

[Agarwal07] M. Agarwal, B. Paul, M. Zhang, and S. Mitra. Circuit Failure Prediction and Its Application to Transistor Aging. In *Proc. IEEE VLSI Test Symposium (VTS)*, pages 277–286. 2007.

[Agarwal11] K. Agarwal. Secure Scan Design, June 2011. US Patent App. 7,966,535.

[Arslan04] B. Arslan and A. Orailoglu. Test Cost Reduction Through A Reconfigurable Scan Architecture. In *Proc. IEEE International Test Conference (ITC)*, pages 945–952. 2004.

[Baranowski12] R. Baranowski, M. A. Kochte, and H.-J. Wunderlich. Modeling, Verification and Pattern Generation for Reconfigurable Scan Networks. In *Proc. IEEE International Test Conference (ITC)*. 2012. Paper 8.2.

[Baranowski13a] R. Baranowski, A. Cook, M. E. Imhof, C. Liu, and H.-J. Wunderlich. Synthesis of Workload Monitors for On-Line Stress Prediction. In *Proc. IEEE Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, pages 137–142. 2013.

[Baranowski13b] R. Baranowski, M. A. Kochte, and H.-J. Wunderlich. Scan Pattern Retargeting and Merging with Reduced Access Time. In *Proc. IEEE European Test Symposium (ETS)*, pages 39–45. 2013.

[Baranowski13c] R. Baranowski, M. A. Kochte, and H.-J. Wunderlich. Securing Access to Reconfigurable Scan Networks. In *Proc. IEEE Asian Test Symposium (ATS)*. 2013.

[Baumann05] R. C. Baumann. Radiation-Induced Soft Errors in Advanced Semiconductor Technologies. *IEEE Trans. on Device and Materials Reliability*, 5(3):305–316, 2005.

[Baumgartner02] J. Baumgartner, A. Kuehlmann, and J. Abraham. Property Checking via Structural Analysis. In E. Brinksma and K. Larsen, editors, *Computer Aided Verification (CAV)*, volume 2404 of *Lecture Notes in Computer Science (LNCS)*, pages 151–165. Springer, 2002. ISBN 978-3-540-43997-4.

[Benabdenbi00] M. Benabdenbi and W. Maroufi. CAS-BUS: A Scalable and Reconfigurable Test Access Mechanisms for Systems on a Chip. In *Proc. Design, Automation and Test in Europe (DATE)*, pages 141–145. 2000.

[Benso08] A. Benso, S. Di Carlo, P. Prinetto, and Y. Zorian. IEEE Standard 1500 Compliance Verification for Embedded Cores. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 16(4):397–407, 2008.

[Bhattacharya03] B. B. Bhattacharya, S. C. Seth, and S. Zhang. Double-Tree Scan: A Novel Low-Power Scan-Path Architecture. In *Proc. IEEE International Test Conference (ITC)*, pages 470–479. 2003.

[Biere99] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In W. R. Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 of *Lecture Notes in Computer Science (LNCS)*, pages 193–207. Springer, 1999. ISBN 978-3-540-65703-3.

[Biere02] A. Biere, C. Artho, and V. Schuppan. Liveness Checking as Safety Checking. *Electronic Notes in Theoretical Computer Science*, 66(2):160–177, 2002.

[Biere03] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded Model Checking. *Advances in Computers*, 58:117–148, 2003.

[Biere06] A. Biere, K. Heljanko, T. A. Junttila, T. Latvala, and V. Schuppan. Linear Encodings of Bounded LTL Model Checking. *Logical Methods in Computer Science*, 2:1–64, 2006.

[Biere09] A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, Feb. 2009. ISBN 978-1-58603-929-5.

[Blaauw08] D. Blaauw, K. Chopra, A. Srivastava, and L. Scheffer. Statistical Timing Analysis: From Basic Principles to State of the Art. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 27(4):589–607, 2008.

[Bonnett99] D. Bonnett. Design for In-System Programming. In *Proc. IEEE International Test Conference (ITC)*, pages 252–259. 1999.

[Borkar05] S. Borkar. Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation. *IEEE Micro*, 25(6):10–16, 2005.

[Brglez85] F. Brglez. A Neutral Netlist of 10 Combinational Benchmark Circuits. In *IEEE Proc. International Symposium on Circuits and Systems (ISCAS)*, pages 695–698. 1985.

[Bruce Jr96] W. C. Bruce Jr, J. E. Drufke Jr, C. O. Eluwa, and J. M. Hudson. Method for Testing a Test Architecture within a Circuit, May 1996. US Patent App. 5,517,637.

[Bruns99] G. Bruns and P. Godefroid. Model Checking Partial State Spaces with 3-Valued Temporal Logics. In N. Halbwachs and D. Peled, editors, *Computer Aided Verification (CAV)*, volume 1633 of *Lecture Notes in Computer Science (LNCS)*, pages 274–287. Springer, 1999.

[Bryant86] R. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. on Computers*, C-35(8):677–691, 1986.

[Bryant90] R. E. Bryant. Symbolic Simulation – Techniques and Applications. In *Proc. ACM/IEEE Design Automation Conference (DAC)*, pages 517–521. 1990.

[Bushnell00] M. Bushnell and V. D. Agrawal. *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*. Springer, 2000. ISBN 978-0-7923-7991-1.

[Buskey06] R. Buskey and B. Frosik. Protected JTAG. In *Proc. IEEE International Conference on Parallel Processing Workshops (ICCPW)*, pages 405–414. 2006.

[Chakrabarty00] K. Chakrabarty. Test Scheduling for Core-Based Systems Using Mixed-Integer Linear Programming. *IEEE Trans on. Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 19(10):1163–1174, 2000.

[Chakraborty11] T. Chakraborty, C.-H. Chiang, S. Goyal, M. Portolan, and B. G. Van Treuren. Apparatus and Method for Controlling Dynamic Modification of a Scan Path, May 2011. US Patent App. 7,954,022.

[Chattopadhyay03] S. Chattopadhyay and K. Reddy. Genetic Algorithm based Test Scheduling and Test Access Mechanism Design for System-on-Chips. In *Proc. IEEE International Conference on VLSI Design (VLSI)*, pages 341–346. 2003.

[Chiu12] G.-M. Chiu and J.-M. Li. A Secure Test Wrapper Design Against Internal and Boundary Scan Attacks for Embedded Cores. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 20(1):126–134, Jan. 2012.

[Chou97] R. Chou, K. Saluja, and V. Agrawal. Scheduling Tests for VLSI Systems under Power Constraints. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 5(2):175–185, 1997.

[Clark10] C. Clark. Anti-Tamper JTAG TAP Design Enables DRM to JTAG Registers and P1687 On-Chip Instruments. In *Proc. IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 19–24. 2010.

[Clarke86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic

Specifications. *ACM Trans. on Programming Languages and Systems*, 8(2):244–263, April 1986.

[Clarke94] E. M. Clarke, O. Grumberg, and D. E. Long. Model Checking and Abstraction. *ACM Trans. on Programming Languages and Systems*, 16(5):1512–1542, Sept. 1994.

[Clarke99] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT press, 1999.

[Clarke03] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-Guided Abstraction Refinement for Symbolic Model Checking. *Journal of the ACM (JACM)*, 50(5):752–794, September 2003.

[Clarke04] E. Clarke, D. Kroening, J. Ouaknine, and O. Strichman. Completeness and Complexity of Bounded Model Checking. In B. Steffen and G. Levi, editors, *Verification, Model Checking, and Abstract Interpretation*, volume 2937 of *Lecture Notes in Computer Science (LNCS)*, pages 85–96. Springer, 2004. ISBN 978-3-540-20803-7.

[Craig57] W. Craig. Linear Reasoning. A New Form of the Herbrand-Gentzen Theorem. *The Journal of Symbolic Logic*, 22(3):250–268, 1957.

[Da Rolt12] J. Da Rolt, G. Di Natale, M.-L. Flottes, and B. Rouzeyre. Are Advanced DfT Structures Sufficient for Preventing Scan-Attacks? In *Proc. IEEE VLSI Test Symposium (VTS)*, pages 246–251. 2012.

[Dahbura89] A. Dahbura, M. Uyar, and C. W.Yau. An Optimal Test Sequence for the JTAG/IEEE P1149.1 Test Access Port Controller. In *Proc. IEEE International Test Conference (ITC)*, pages 55–62. 1989.

[Das13] A. Das, J. Rolt, S. Ghosh, S. Seys, S. Dupuis, G. Natale, M.-L. Flottes, B. Rouzeyre, and I. Verbauwhede. Secure JTAG Implementation Using Schnorr Protocol. *Journal of Electronic Testing (JETTA)*, 29(2):193–209, 2013.

[Davis62] M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem-Proving. *Communications of the ACM*, 5(7):394–

397, 1962.

[Diamantidis05] I. Diamantidis, T. Oikonomou, and S. Diamantidis. Towards an IEEE P1500 Verification Infrastructure: A Comprehensive Approach. In *Proc. IEEE International Workshop on Infrastructure IP (IIP)*, pages 25–30. 2005.

[Dworak13] J. Dworak, A. Crouch, J. Potter, A. Zygmontowicz, and M. Thornton. Don't Forget to Lock your SIB: Hiding Instruments using P1687. In *Proc. IEEE International Test Conference (ITC)*. 2013. Paper 6.2.

[Ebrard09] E. Ebrard, B. Allard, P. Candelier, and P. Waltz. Review of Fuse and Antifuse Solutions for Advanced Standard CMOS Technologies. *Microelectronics Journal*, 40(12):1755–1765, 2009.

[Eén03] N. Eén and N. Sörensson. Temporal Induction by Incremental SAT Solving. *Electronic Notes in Theoretical Computer Science*, 89(4):543–560, 2003.

[Eén04] N. Eén and N. Sörensson. An Extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *Theory and Applications of Satisfiability Testing (SAT)*, volume 2919 of *Lecture Notes in Computer Science (LNCS)*, pages 502–518. Springer, 2004. ISBN 978-3-540-20851-8.

[Eén06] N. Eén and N. Sörensson. Translating Pseudo-Boolean Constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

[Eggersglüss07] S. Eggersglüss, G. Fey, R. Drechsler, A. Glowatz, F. Hapke, and J. Schloeffel. Combining Multi-Valued Logics in SAT-based ATPG for Path Delay Faults. In *Proc. IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMCODE)*, pages 181–187. 2007.

[Eichelberger77] E. B. Eichelberger and T. W. Williams. A Logic Design Structure for LSI Testability. In *Proc. ACM/IEEE Design Automation Conference (DAC)*, pages 462–468. 1977.

[Eklow06] B. Eklow and B. Bennetts. New Techniques for Accessing Embedded Instrumentation: IEEE P1687 (IJTAG). In *Proc. IEEE European Test Symposium (ETS)*, pages 253–254. 2006.

[Fisher02] R. Fisher. Method and Apparatus to Check the Integrity of Scan Chain Connectivity by Traversing the Test Logic of the Device, Nov. 2002. US Patent App. 10/300,513.

[Foster11] H. Foster. Challenges of Design and Verification in the SoC Era. In *Design and Verification Conference and Exhibition*. 2011.

[Garey79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., 1979. ISBN 978-0716710455.

[Gebser11] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. Multi-Criteria Optimization in Answer Set Programming. In *Technical Communications of the International Conference on Logic Programming (ICLP)*, volume 11 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1–10. 2011.

[Ghani Zadegan11a] F. Ghani Zadegan, U. Ingelsson, G. Asani, G. Carlsson, and E. Larsson. Test Scheduling in an IEEE P1687 Environment with Resource and Power Constraints. In *Proc IEEE Asian Test Symposium (ATS)*, pages 525–531. 2011.

[Ghani Zadegan11b] F. Ghani Zadegan, U. Ingelsson, G. Carlsson, and E. Larsson. Design Automation for IEEE P1687. In *Proc. Design, Automation Test in Europe Conference (DATE)*, pages 1412–1417. 2011.

[Ghani Zadegan12a] F. Ghani Zadegan, U. Ingelsson, G. Carlsson, and E. Larsson. Access Time Analysis for IEEE P1687. *IEEE Trans. on Computers*, 61(10):1459–1472, October 2012.

[Ghani Zadegan12b] F. Ghani Zadegan, U. Ingelsson, E. Larsson, and G. Carlsson. Reusing and Retargeting On-Chip Instrument Access Procedures in IEEE P1687. *IEEE Design & Test of Computers*, 29(2):79–88, 2012.

[Giunchiglia92] F. Giunchiglia and T. Walsh. A Theory of Abstraction. *Artificial Intelligence*, 57(2):323–389, 1992.

[Hely04] D. Hely, M. L. Flottes, F. Bancel, B. Rouzeyre, N. Berard, and M. Renovell. Scan Design and Secure Chip [Secure IC Testing]. In *Proc. IEEE On-Line Testing Symposium (IOLTS)*, pages 219–224. 2004.

[Holst09] S. Holst and H.-J. Wunderlich. Adaptive Debug and Diagnosis without Fault Dictionaries. *Journal of Electronic Testing (JETTA)*, 25(4-5):259–268, 2009.

[Holzmann97] G. Holzmann. The Model Checker SPIN. *IEEE Trans. on Software Engineering*, 23(5):279–295, 1997.

[Horstmann84] P. Horstmann and E. Stabler. Computer Aided Design (CAD) Using Logic Programming. In *Proc. Design Automation Conference (DAC)*, pages 144–151. 1984.

[ITRS12] ITRS. International Technology Roadmap for Semiconductors, 2012. http://www.itrs.net/Links/2012ITRS/Home2012.htm.

[Iyengar03] V. Iyengar, K. Chakrabarty, and E. Marinissen. Test Access Mechanism Optimization, Test Scheduling, and Tester Data Volume Reduction for System-on-Chip. *IEEE Trans. on Computers*, 52(12):1619–1632, 2003.

[Jain95] S. Jain, R. E. Bryant, and A. Jain. Automatic Clock Abstraction from Sequential Circuits. In *Proc. Design Automation Conference (DAC)*, pages 707–711. 1995.

[JTA01] IEEE Standard Test Access Port and Boundary-Scan Architecture 1149.1-2001, 2001. Test Technology Technical Committee of the IEEE Computer Society, USA.

[JTA13] IEEE Standard for Test Access Port and Boundary-Scan Architecture 1149.1-2013, 2013. Test Technology Technical Committee of the IEEE Computer Society, USA.

[Jutman11] A. Jutman, S. Devadze, and J. Aleksejev. Invited Paper: System-wide Fault Management Based on IEEE P1687 IJTAG. In *Proc. IEEE International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 1–4. 2011.

[Kamepalli06] H. B. Kamepalli, P. Sanjeevarao, and C.-J. Park. Scan Chain Verification Using Symbolic Simulation, May 2006. US Patent App. 7,055,118.

[Kapur99] R. Kapur, D. Martin, and T. W. Williams. Dynamic Scan Chains and Test Pattern Generation Methodologies Therefor, Dec. 1999. US Patent App. 09/469,729.

[Keane10] J. Keane, X. Wang, D. Persaud, and C. Kim. An All-In-One Silicon Odometer for Separately Monitoring HCI, BTI, and TDDB. *IEEE Journal of Solid-State Circuits*, 45(4):817–829, 2010.

[Kleene50] S. Kleene. *Introduction to Metamathematics*. D. Van Nostrand, Princeton, NJ, 1950.

[Kömmerling99] O. Kömmerling and M. G. Kuhn. Design Principles for Tamper-Resistant Smartcard Processors. In *Proc. USENIX Workshop on Smartcard Technology (WOST)*, pages 9–20. USENIX Association, 1999.

[Koranne03] S. Koranne. Design of Reconfigurable Access Wrappers for Embedded Core Based SoC Test. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 11(5):955–960, 2003.

[Kripke63] S. Kripke. Semantical Considerations on Modal Logic. *Acta Philosophica Fennica*, 16:83–94, 1963.

[Kropf99] T. Kropf. *Introduction to Formal Hardware Verification*. Springer, 1999. ISBN 978-3-540-65445-2.

[Larsson03] E. Larsson and Z. Peng. A Reconfigurable Power-Conscious Core Wrapper and its Application to SOC Test Scheduling. In *Proc. IEEE International Test Conference (ITC)*, pages 1135–1144. 2003.

[Larsson06] E. Larsson and H. Fujiwara. System-on-Chip Test Scheduling with Reconfigurable Core Wrappers. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 14(3):305–309, 2006.

[Larsson12] E. Larsson and F. Ghani Zadegan. Accessing Embedded DfT Instruments with IEEE P1687. In *Proc. IEEE Asian Test Symposium*

(ATS), pages 71–76. 2012.

[Lee06] J. Lee, M. Tehranipoor, and J. Plusquellic. A Low-Cost Solution for Protecting IPs Against Scan-Based Side-Channel Attacks. In *Proc. IEEE VLSI Test Symposium (VTS)*, pages 94–99. 2006.

[Lee07] J. Lee, M. Tehranipoor, C. Patel, and J. Plusquellic. Securing Designs against Scan-Based Side-Channel Attacks. *IEEE Trans. on Dependable and Secure Computing*, 4(4):325–336, Oct.-Dec. 2007.

[Ley09] A. Ley. Doing More with Less—An IEEE 1149.7 Embedded Tutorial: Standard for Reduced-Pin and Enhanced-Functionality Test Access Port and Boundary-Scan Architecture. In *Proc. IEEE International Test Conference (ITC)*. 2009. Paper ET3.1.

[Loiseaux95] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, S. Bensalem, and D. Probst. Property Preserving Abstractions for the Verification of Concurrent Systems. *Formal Methods in System Design*, 6(1):11–44, 1995.

[Marinissen98] E. J. Marinissen, R. G. J. Arendsen, G. Bos, H. Dingemanse, M. Lousberg, and C. Wouters. A Structured and Scalable Mechanism for Test Access to Embedded Reusable Cores. In *Proc. IEEE International Test Conference (ITC)*, pages 284–293. 1998.

[Marinissen02] E. Marinissen, V. Iyengar, and K. Chakrabarty. A Set of Benchmarks for Modular Testing of SOCs. In *Proc. IEEE International Test Conference (ITC)*, pages 519–528. 2002.

[McCluskey86] E. J. McCluskey. *Logic Design Principles with Emphasis on Testable Semicustom Circuits*. Prentice-Hall, 1986. ISBN 0-13-539784-7.

[McMillan93] K. L. McMillan. Symbolic Model Checking. pages 25–60. Springer US, 1993. ISBN 978-1-4613-6399-6.

[McMillan03a] K. McMillan. Interpolation and SAT-Based Model Checking. In J. Hunt, Warren A. and F. Somenzi, editors, *Computer Aided Verification (CAV)*, volume 2725 of *Lecture Notes in Computer Science (LNCS)*, pages 1–13. Springer, 2003. ISBN 978-3-540-40524-5.

[McMillan03b] K. L. McMillan and N. Amla. Automatic Abstraction without Counterexamples. In H. Garavel and J. Hatcliff, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2619 of *Lecture Notes in Computer Science (LNCS)*, pages 2–17. Springer, 2003. ISBN 978-3-540-00898-9.

[Melham87] T. F. Melham. Abstraction Mechanisms for Hardware Verification. In G. Birtwistle and P. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*, volume 35 of *The Kluwer International Series in Engineering and Computer Science*, pages 267–291. Springer US, 1987. ISBN 978-1-4612-9197-8.

[Melocco03] K. Melocco, H. Arora, P. Setlak, G. Kunselman, and S. Mardhani. A Comprehensive Approach to Assessing and Analyzing 1149.1 Test Logic. In *Proc. IEEE International Test Conference (ITC)*, pages 358–367. 2003.

[Mishchenko06] A. Mishchenko, S. Chatterjee, and R. Brayton. DAG-Aware AIG Rewriting A Fresh Look at Combinational Logic Synthesis. In *Proc. ACM Design Automation Conference (DAC)*, pages 532–535. 2006.

[Mitra10] S. Mitra, S. Seshia, and N. Nicolici. Post-Silicon Validation Opportunities, Challenges and Recent Advances. In *Proc. ACM/IEEE Design Automation Conference (DAC)*, pages 12–17. 2010.

[Moore65] G. E. Moore. Cramming More Components onto Integrated Circuits. *Electronics*, 38(8), April 1965.

[Moore75] G. E. Moore. Progress in Digital Integrated Electronics. In *Proc. IEEE International Electron Devices Meeting*, volume 21, pages 11–13. 1975.

[N45] Nangate 45nm Open Cell Library v1.3, http://www.nangate.com. Accessed: Oct. 16, 2013.

[Nadeau-Dostie09] B. Nadeau-Dostie, S. Adham, and R. Abbott. Improved Core Isolation and Access for Hierarchical Embedded Test. *IEEE Design & Test of Computers*, 26(1):18 –25, 2009.

[Narayanan93] S. Narayanan and M. A. Breuer. Reconfigurable Scan Chains: A Novel Approach To Reduce Test Application Time. In *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 710–715. 1993.

[Nguyen08] M. Nguyen, M. Thalmaier, M. Wedler, J. Bormann, D. Stoffel, and W. Kunz. Unbounded Protocol Compliance Verification Using Interval Property Checking With Invariants. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 27(11):2068–2082, 2008.

[Nguyen11] M. Nguyen, M. Wedler, D. Stoffel, and W. Kunz. Formal Hardware/Software Co-Verification by Interval Property Checking with Abstraction. In *Proc. ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 510–515. 2011.

[Nicolaidis89] M. Nicolaidis, S. Noraz, and B. Courtois. A Generalized Theory of Fail-Safe Systems. In *International Symposium on Fault-Tolerant Computing (FTCS), Digest of Papers*, pages 398–406. 1989.

[Papaspyridis88] A. Papaspyridis. A PROLOG-based Connectivity Verification Tool. In *Proc. ACM/IEEE Design Automation Conference (DAC)*, pages 523–527. 1988.

[Park10] K. Park, S. Yoo, T. Kim, and J. Kim. JTAG Security System Based on Credentials. *Journal of Electronic Testing (JETTA)*, 26:549–557, 2010.

[Park12] K.-Y. Park, S.-G. Yoo, and J. Kim. Debug Port Protection Mechanism for Secure Embedded Devices. *IEEE Journal of Semiconductor Technology and Science*, 12(2):240–253, 2012.

[Pierce13] L. Pierce and S. Tragoudas. Enhanced Secure Architecture for Joint Action Test Group Systems. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 21(7):1342–1345, 2013.

[Pnueli77] A. Pnueli. The Temporal Logic of Programs. In *Proc. IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 46–57. 1977.

[Prasad05] M. R. Prasad, A. Biere, and A. Gupta. A Survey of Recent Advances in SAT-Based Formal Verification. *International Journal on Software Tools for Technology Transfer*, 7(2):156–173, 2005.

[Quasem04] M. S. Quasem and S. K. Gupta. Designing Reconfigurable Multiple Scan Chains for Systems-on-Chip. In *Proc. IEEE VLSI Test Symposium (VTS)*, pages 367–376. 2004.

[Rearick05] J. Rearick, B. Eklow, K. Posse, A. Crouch, and B. Bennetts. IJTAG (Internal JTAG): A Step Toward a DFT Standard. In *Proc. IEEE International Test Conference (ITC)*. 2005. Paper 32.4.

[Rearick06] J. Rearick and A. Volz. A Case Study of Using IEEE P1687 (IJTAG) for High-Speed Serial I/O Characterization and Testing. In *Proc. IEEE International Test Conference (ITC)*. 2006. Paper 10.2.

[Remmers04] J. Remmers, M. Villalba, and R. Fisette. Hierarchical DFT Methodology—A Case Study. In *Proc. IEEE International Test Conference (ITC)*, pages 847–856. 2004.

[Rosenfeld10] K. Rosenfeld and R. Karri. Attacks and Defenses for JTAG. *IEEE Design & Test of Computers*, 27(1):36–47, 2010.

[Rosenfeld11] K. Rosenfeld and R. Karri. Security-Aware SoC Test Access Mechanisms. In *Proc. IEEE VLSI Test Symposium (VTS)*, pages 100–104. 2011.

[Saïdi99] H. Saïdi and N. Shankar. Abstract and Model Check while You Prove. In N. Halbwachs and D. Peled, editors, *Computer Aided Verification (CAV)*, volume 1633 of *Lecture Notes in Computer Science (LNCS)*, pages 443–454. Springer, 1999. ISBN 978-3-540-66202-0.

[Samaranayake02] S. Samaranayake, N. Sitchinava, R. Kapur, M. Amin, and T. Williams. Dynamic Scan: Driving Down the Cost of Test. *IEEE Trans. on Computers*, 35(10):63–68, 2002.

[Samaranayake03] S. Samaranayake, E. Gizdarski, N. Sitchinava, F. Neuveux, R. Kapur, and T. W. Williams. A Reconfigurable Shared Scan-in Architecture. In *Proc. IEEE VLSI Test Symposium (VTS)*, pages 9–14. 2003.

[SEC05] IEEE Standard for Embedded Core Test 1500-2005, 2005. Test Technology Technical Committee of the IEEE Computer Society, USA.

[Sehgal04] A. Sehgal, S. K. Goel, E. J. Marinissen, and K. Chakrabarty. IEEE P1500-Compliant Test Wrapper Design for Hierarchical Cores. In *Proc. IEEE International Test Conference (ITC)*, pages 1203–1212. 2004.

[Sheeran00] M. Sheeran, S. Singh, and G. Stålmarck. Checking Safety Properties Using Induction and a SAT-Solver. In J. Hunt, Warren A. and S. D. Johnson, editors, *Formal Methods in Computer-Aided Design (FMCAD)*, volume 1954 of *Lecture Notes in Computer Science (LNCS)*, pages 127–144. Springer, 2000. ISBN 978-3-540-41219-9.

[Singh97] H. Singh, G. Patankar, and J. Beausang. A Symbolic Simulation-Based ANSI/IEEE Std 1149.1 Compliance Checker and BSDL Generator. In *Proc. IEEE International Test Conference (ITC)*, pages 256–264. 1997.

[Sistla85] A. P. Sistla and E. M. Clarke. The Complexity of Propositional Linear Temporal Logics. *Journal of the ACM (JACM)*, 32(3):733–749, July 1985.

[Sourgen92] L. Sourgen. Security Locks for Integrated Circuit, May 1992. US Patent App. 5101121 A.

[Stollon11] N. Stollon. *On-Chip Instrumentation: Design and Debug for Systems on Chip*. Springer US, 2011. ISBN 978-1-4419-7563-8.

[Tehranipoor11] M. Tehranipoor and C. Wang. *Introduction to Hardware Security and Trust*. Springer, 2011. ISBN 978-1-4419-8080-9.

[Tseitin83] G. S. Tseitin. On the Complexity of Derivation in Propositional Calculus. In *Automation of Reasoning*, pages 466–483. Springer, 1983.

[Urdahl12] J. Urdahl, D. Stoffel, M. Wedler, and W. Kunz. System Verification of Concurrent RTL Modules by Compositional Path Predicate Abstraction. In *Proc. ACM Design Automation Conference*

(DAC), pages 334–343. 2012.

[Vermeulen08] B. Vermeulen, N. Stollon, R. Kuhnis, G. Swoboda, and J. Rearick. Overview of Debug Standardization Activities. *IEEE Design & Test of Computers*, 25(3):258–267, 2008.

[Wang06] L.-T. Wang, C.-W. Wu, and X. Wen. *VLSI Test Principles and Architectures: Design for Testability*. Elsevier, 2006. ISBN 9780080474793.

[Wang10] L.-T. Wang, C. E. Stroud, and N. A. Touba. *System-on-Chip Test Architectures: Nanometer Design for Testability*. Morgan Kaufmann, 2010.

[Whetsel99] L. Whetsel. Addressable Test Ports An Approach to Testing Embedded Cores. In *Proc. IEEE International Test Conference (ITC)*, pages 1055–1064. 1999.

[Windley95] P. Windley. Formal Modeling and Verification of Microprocessors. *IEEE Trans. on Computers*, 44(1):54–72, 1995.

[Wu98] Y. Wu. Diagnosis of Scan Chain Failures. In *Proc. IEEE Defect and Fault Tolerance in VLSI Systems (DFTS)*, pages 217–222. 1998.

[Xiang08] D. Xiang, Y. Zhao, K. Chakrabarty, and H. Fujiwara. A Reconfigurable Scan Architecture With Weighted Scan-Enable Signals for Deterministic BIST. *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems (TCAD)*, 27(6):999–1012, 2008.

[Yang04] B. Yang, K. Wu, and R. Karri. Scan Based Side Channel Attack on Dedicated Hardware Implementations of Data Encryption Standard. In *Proc. IEEE International Test Conference (ITC)*, pages 339–344. 2004.

[Yang06] B. Yang, K. Wu, and R. Karri. Secure Scan: A Design-for-Test Architecture for Crypto Chips. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 25(10):2287–2293, 2006.

[Zorian98] Y. Zorian, E. Marinissen, and S. Dey. Testing Embedded-Core Based System Chips. In *Proc. IEEE International Test Conference*

(ITC), pages 130–143. 1998.

[Zorian02] Y. Zorian. Embedded Memory Test and Repair: Infrastructure IP for SOC Yield. In *Proc. IEEE International Test Conference (ITC)*, pages 340–349. 2002.

[Zorian05] Y. Zorian and A. Yessayan. IEEE 1500 Utilization in SOC Design and Test. In *Proc. IEEE International Test Conference (ITC)*. 2005. Paper 23.2.

[Zou03] W. Zou, S. Reddy, I. Pomeranz, and Y. Huang. SOC Test Scheduling Using Simulated Annealing. In *Proc. IEEE VLSI Test Symposium (VTS)*, pages 325–330. 2003.

Appendices

A. Benchmark Scan Networks

The reconfigurable scan networks considered in this thesis are derived from benchmark suites for test scheduling (ITC'02) and test pattern generation (ISCAS'85):

- The ITC'02 [Marinissen02] suite describes a set of hierarchical systems composed of multiple modules (cores) with a defined number of inputs, outputs, internal scan chains, and submodules (constituent cores). This suite is used to construct hierarchical RSN architectures.
- The ISCAS'85 [Brglez85] suite is a set of combinational circuits with up to a few thousand gates. Based on this suite, complex control signals are developed for flat RSN architectures.

Three benchmark RSN architectures are developed:

- *SIB-based* architecture with hierarchies derived from ITC'02 benchmarks. Each module of an ITC'02 benchmark is assigned a dedicated RSN with one scan input and one scan output. The scan network for each module includes scan segments for boundary and internal scan chains, as well as the scan networks of the constituent submodules. The access to scan segments and submodules is controlled by SIBs (cf. Section 3.4 and Figure 3.2, p. 45).
- *MUX-based* architecture constructed from ITC'02 benchmarks as above, with two modes of operation for data and configuration access.
- *Flat* architecture with control signals derived from ISCAS'85 circuits. This architecture has little practical significance, but it serves as a valuable benchmark with complex combinational dependencies for scalability evaluation of verification algorithms.

To the best knowledge of the author, this is the first benchmark suite for reconfigurable scan architectures.

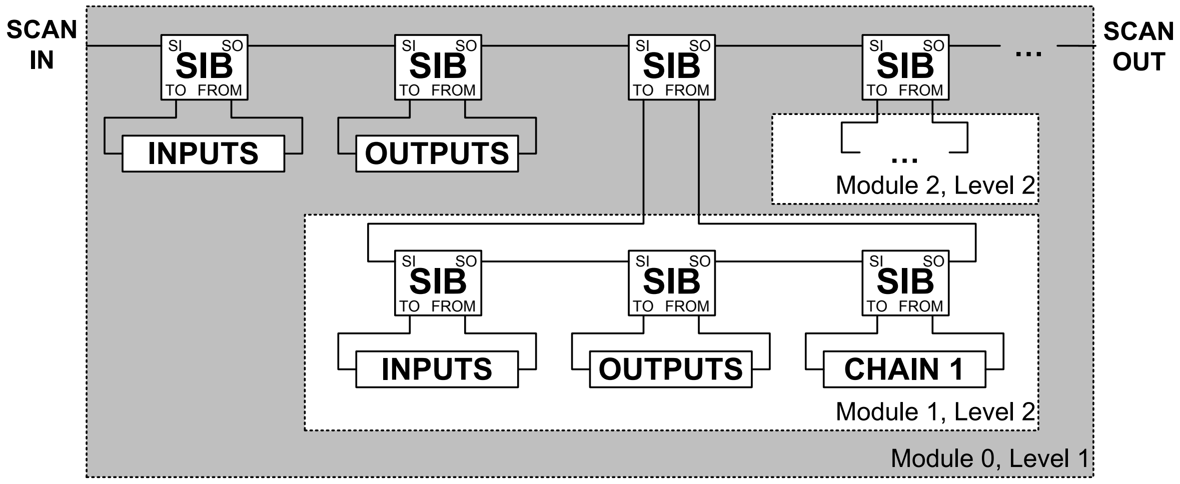


Figure A.1.: SIB-based scan architecture for the p34392 benchmark

A.1. SIB-Based Architecture

The *SIB-based* scan architecture follows the design from [Ghani Zadegan11b]. Segment insertion bits (cf. Figure 3.2, p. 45) are used as hierarchical gateways to the ITC'02 cores, their submodules, as well as input and output boundary registers and individual scan chains. Figure A.1 shows the SIB-based architecture for the top-level part of the p34392 benchmark. The initial scan configuration is 0 for all SIBs (SIBs are closed) and unknown (X) for all data scan segments (e.g. `INPUTS`, `OUTPUTS`).

Table A.1 shows the characteristics of the SIB-based benchmarks. Column "# Levels" gives the number of hierarchy levels in the corresponding ITC'02 benchmark. The number of scan segments ("# Scan segments") includes the 1-bit scan segments that comprise the SIBs. The last column gives the total benchmark area for the Nangate 45 nm library [N45].

A.2. MUX-Based Architecture

The MUX-based architecture supports two access modes: configuration access and data access. The *configuration access* mode allows to reconfigure the active scan path by attaching or detaching internal scan segments or submodules. The *data access* mode provides access only to those scan segments that were chosen in the configuration mode.

Design	# Levels	# SIBs	# Scan segments	# Scan cells	Area $[\mu m^2]$
u226	2	50	90	1 466	22 313
d281	2	59	109	3 872	58 747
d695	2	168	325	8 397	126 777
h953	2	55	101	5 641	85 133
g1023	2	80	145	5 386	81 396
f2126	2	41	77	15 830	239 902
q12710	2	25	47	26 183	397 483
p22810	3	283	537	30 111	453 537
p34392	3	123	226	23 242	352 290
p93791	3	621	1 209	98 605	1 486 289
t512505	2	160	288	77 006	1 167 569
a586710	3	40	72	41 675	634 087

Table A.1.: Characteristics of the SIB-based benchmark architectures

Figure A.2 shows the MUX-based architecture for the top-level part of the p34392 benchmark. The scan chain of each module starts with a 1-bit configuration scan segment `AM` that sets either the *configuration access mode* (`AM` $= 0$), in which only the configuration scan segments `C` can be accessed, or the *data access* mode (`AM` $= 1$). Once configured, this architecture allows fast data access since the control scan segments `C` are not present on the active scan path in the *data access* mode. The initial scan configuration (after reset) is 0 for all control scan segments (`C` and `AM`) and unknown (X) for all data scan segments (e.g. `INPUTS` and `OUTPUTS`).

Table A.2 presents the characteristics of the MUX-based benchmarks. Column "# Levels" gives the number of hierarchy levels in the corresponding ITC'02 benchmark. The number of multiplexers ("# Muxes") is equivalent to the number of control scan segments (`C` and `AM`). The last column gives the total benchmark area for the Nangate 45 nm library [N45].

A.3. Flat Architecture

The *flat* scan architecture is a daisy-chain of 1-bit configuration scan segments, followed with a chain of 32-bit data scan segments that can be individually bypassed by scan multiplexers. Figure A.3 shows the flat architecture for the c17 benchmark from

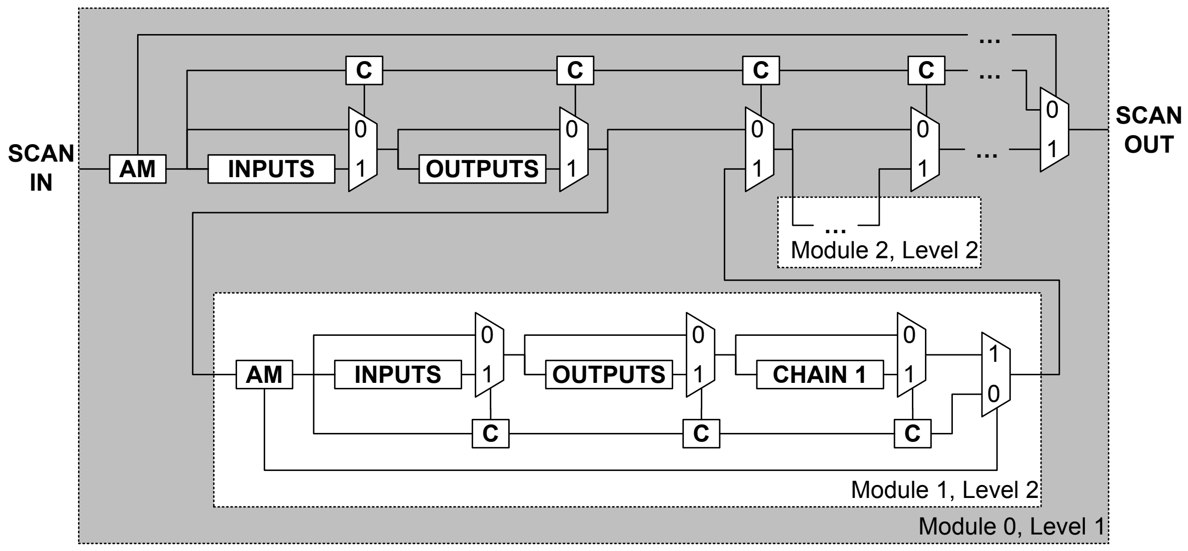


Figure A.2.: MUX-based scan architecture for the p34392 benchmark

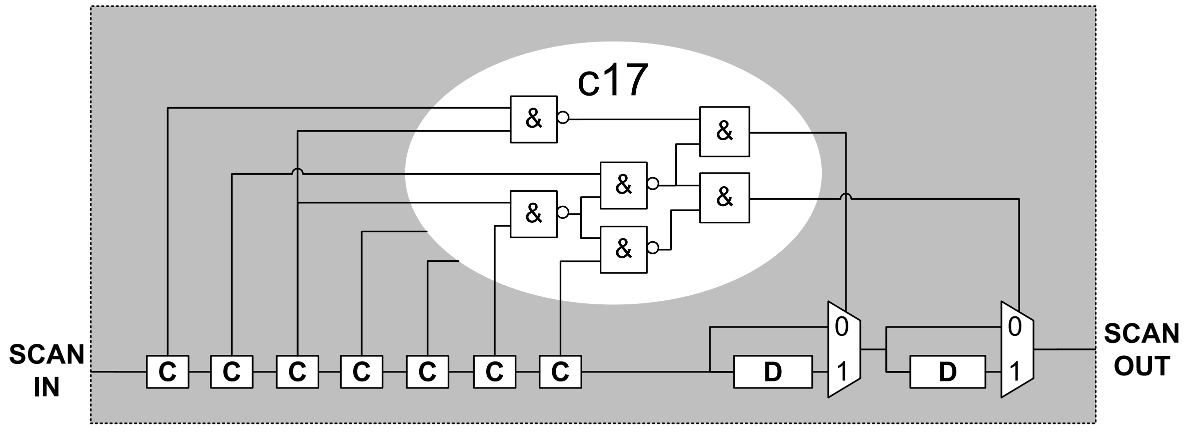


Figure A.3.: Flat scan architecture for the c17 benchmark

the ISCAS'85 suite. Each configuration scan segment C drives a primary input of c17, and the access to each data scan segment D is controlled by a primary output of c17. A data scan segment belongs to the active scan path if the corresponding output of c17 is 1 and is bypassed otherwise. The initial scan configuration is 0 for all control scan segments C and unknown (X) for all data scan segments D.

Table A.3 presents the characteristics of the flat benchmarks. Column "# Logic gates" gives the number of gates in the control logic block (corresponding ISCAS'85 benchmark). The number of multiplexers ("# Muxes") is equivalent to the number of data scan segments (D). The last Clim gives the total benchmark area for the Nangate 45 nm library [N45].

Design	# Levels	# Muxes	# Scan segments	# Scan cells	Area $[\mu m^2]$
u226	2	59	99	1 475	22 557
d281	2	67	117	3 880	58 979
d695	2	178	335	8 407	127 007
h953	2	63	109	5 649	85 349
g1023	2	94	159	5 400	81 727
f2126	2	45	81	15 834	240 021
q12710	2	30	51	26 188	397 592
p22810	3	311	565	30 139	454 107
p34392	3	142	245	23 261	352 699
p93791	3	653	1 241	98 637	1 486 772
t512505	2	191	319	77 037	1 168 310
a586710	3	47	79	41 682	634 258

Table A.2.: Characteristics of the MUX-based benchmark architectures

Design	# Muxes	# Logic gates	# Scan segments	# Scan cells	Area $[\mu m^2]$
c17	2	6	7	69	1 038
c432	7	160	43	260	3 950
c499	32	202	73	1 065	16 098
c880	26	383	86	892	13 473
c1355	32	546	73	1 065	16 097
c1908	25	880	58	833	12 633
c2670	140	1 269	373	4 713	70 720
c3540	22	1 669	72	754	11 731
c5315	123	2 307	301	4 114	62 276
c6288	32	2 416	64	1 056	17 313
c7552	108	3 513	315	3 663	55 774

Table A.3.: Characteristics of the flat benchmark architectures

B. Results: Verification

The CSU-accurate verification approach developed in Chapter 5 (p. 69) is evaluated on RSN benchmarks from Appendix A (p. 141). Section B.1 presents the results for robustness verification of both fault-free and faulty benchmark circuits. The accessibility of scan segments is verified in Section B.2. Section B.3 compares the performance of the proposed verification method with a cycle-accurate model checking tool.

The experiments are run on an Intel Core2 CPU operating at 2.83 GHz. As a SAT solver, MiniSat [Eén04] is used.

B.1. Verification of Robustness

The strong robustness property is verified using the technique presented in Section 5.2.2 (p. 77). All considered benchmarks are successfully proven strongly robust.

Detailed SAT solver statistics for proving strong robustness are presented in Table B.1 for the SIB-based, MUX-based, and flat RSN architectures. Columns "Variables" and "Clauses" give the number of variables and clauses contained in the inductive SAT instance φ_{induct} (cf. formula (5.8), p. 77). The average and maximal number of times that the SAT solver backtracks while checking the satisfiability of φ_{induct} is given under "Conflicts". Columns $t_{\text{solve}}^{\text{init}}$ and $t_{\text{solve}}^{\text{induct}}$ give the SAT solving time for proving φ_{init} and φ_{induct} unsatisfiable, respectively. The total verification time amounts to $t_{\text{solve}}^{\text{init}} + t_{\text{solve}}^{\text{induct}}$.

For the SIB-based architecture, the verification of robustness takes from 50 ms (q12710) up to 93 s for the largest benchmark (p93791). For MUX-based designs, the worst case verification time is 72 s for p93791. Benchmarks with the flat scan architecture have simpler sequential dependencies and, hence, the worst case verification effort is below 1 s.

Debugging Faulty Designs

In the following, the MUX-based benchmarks from Section A.2 (p. 142) are randomly mutated to model possible design bugs. Three types of design bugs are considered:

- *Path bug*: The successors of two random scan elements (scan segments or scan multiplexers) are swapped.
- *Control bug*: The *address* control signals of two random scan multiplexers are swapped.
- *Mux bug*: The inputs of a random 2-input scan multiplexer are swapped.

For each benchmark and each bug type, a hundred random faulty RSNs is generated. The CSU-accurate bounded model checking method from Section 5.1 (p. 70) proves that *none* of the faulty RSNs is strongly robust. For each faulty RSN, the generated counterexample shows the cause of robustness violation, and thus can be used to locate the bug. Since all the random bugs are detected, it appears that design errors are very likely to violate the robustness property.

Table B.2 presents the detailed verification results for the faulty MUX-based benchmarks. Columns “Clauses” and “Conflicts” give the number of clauses in the BMC SAT instances and the number of times that the solver backtracks. The length of counterexamples (number of CSU operations required to refute the robustness property by reaching an invalid scan configuration) is given in column “Depth”. Each column lists the average and maximal values for the random faulty instances of each benchmark. $t_{\text{solve}}^{\text{avg}}$ and $t_{\text{solve}}^{\text{max}}$ give the average and maximal effort required to detect a bug.

The average verification effort for a faulty RSN is smaller than the verification time of its fault-free counterpart (cf. Table B.1). The maximal verification effort is an order of magnitude more than the average effort, and is below 2 minutes in the worst case. All counterexamples are found within a depth of 2 CSU operations.

B.2. Verification of Accessibility

The observability and controllability of scan segments is verified using the technique discussed in Section 5.1.1 (p. 71). For each scan segment s of an RSN, it is shown that the property $\boldsymbol{G}\ [\neg\texttt{Active}(s)]$ does not hold in the CAM, i.e., s is accessible. In addition, it is also verified that the initial state of remaining scan segments can be restored after accessing the target scan segment.

Table B.3 presents the detailed verification effort for SIB-based, MUX-based and flat scan architectures. Column “Clauses” gives the number of clauses contained in the SAT instance after a counterexample is found. The number of times the SAT solver needs to backtrack is listed under “Conflicts”. Column “Depth” shows the number of CSU operations that are required to access a target scan segment and restore the initial state of the remaining scan segments. The average and maximal numbers refer to the accessibility proofs for individual scan segments. $t_{\text{solve}}^{\text{max}}$ is the maximal solve time for a single scan segment, whereas $t_{\text{solve}}^{\text{total}}$ is the total verification time of a benchmark.

Although the size of SAT instances grows up to about 323,000 clauses, the maximal solve time for accessibility verification of a single scan segment is 0.5 s in the worst case, and just 0.16 s on average for the largest RSN (MUX-based p93791). This is due to the fact that the majority of clauses describes signal propagation with just two literals, which is efficiently handled by state-of-the-art SAT solvers. For the majority of the RSNs, the total verification time is below 10 s, and it raises up to 200 s for the largest RSN.

The verification of SIB-based architectures does not require the solver to backtrack since a solution is found by direct implications as soon as a sufficient number of CSU operations is allowed in the SAT instance. In contrast, the MUX-based architecture may cause temporal conflicts and backtracking if the solver takes a wrong decision on the access order to configuration registers. This also explains the increased effort for the verification of the MUX-based RSNs, which is roughly doubled in comparison to the verification time for SIB-based designs.

As already shown in the previous section, all considered benchmark circuits are robust. Therefore, the benchmark CAMs can be simplified by removing the predicate `Valid` from the definition of the `Active` function (cf. Section 5.2.1, p. 75). This simplification leads to a slightly lower verification effort: Accessibility verification in the simplified CAM leads to about 8% reduction in the number of clauses, and 4% reduction of the SAT solver runtime on average.

Debugging Faulty Designs

In the following, the accessibility verification method is applied to faulty MUX-based RSNs with random design bugs. The faulty designs are generated as described in the previous section.

A scan segment s is inaccessible if the LTL property $A_s := \boldsymbol{G}\ [\neg\texttt{Active}(s)]$ holds in the CSU-accurate model. Let $\texttt{ct}(A_s)$ be the completeness threshold for A_s. The property A_s is proven by showing that there exists no counterexample to A_s with length $\texttt{ct}(A_s)$. The completeness threshold $\texttt{ct}(A_s)$ is calculated using the method from Section 5.2.4 (p. 82): In the MUX-based architecture, for any target scan segment s, the cone of influence of A_s always includes only one node per level. Let $n \in \mathbb{N}^+$ be the hierarchy level of the module that contains s in the ITC'02 benchmark (cf. Figure A.2, p. 144). According to Corollary 2 (p. 83):

- If s is a configuration scan segment `AM`, then:

$$\texttt{ct}(A_s) := 2^{2n-2} \tag{B.1}$$

- If s is a configuration scan segment `C`, then:

$$\texttt{ct}(A_s) := 2^{2n-1} \tag{B.2}$$

- Otherwise, if s is not a configuration scan segment (e.g. `INPUTS` or `OUTPUTS` in Figure A.2), then:

$$\texttt{ct}(A_s) := 2^{2n} \tag{B.3}$$

Thus, as the deepest benchmark has 3 hierarchy levels, the maximal completeness threshold for property A_s is $2^6 = 64$.

The completeness threshold of 64 time steps is used as the BMC bound in all verification experiments regardless of the actual hierarchy level of the target scan segment. Table B.4 presents the verification results: Column "Found" gives the ratio of faulty RSNs in which the bug is found, i.e., it is proven that at least one scan segment is inaccessible. Column "Inaccessible" gives the average ratio of scan segments that are found inaccessible in a faulty RSN. $t_{\text{solve}}^{\text{avg}}$ and $t_{\text{solve}}^{\text{max}}$ give the average and maximal total verification time per faulty RSN, respectively.

The average verification effort for a faulty RSN is a few times higher than the verification time of its fault-free counterpart due to the higher bound (cf. Table B.3, p. 154). The average verification time for the largest benchmark is below 400 s. The maximal verification time is below 2 hours in the worst case. The *path bugs* and *mux bus* are always found (each faulty RSN has at least one inaccessible scan segment), whereas up to 98% of *control bugs* do not affect the accessibility of scan segments. This means that it is often possible to find access sequences to all scan segments in the RSN, even if control signals of two random scan multiplexers are swapped. Nevertheless, all the random bugs are found by checking the robustness property, as shown in the previous section.

B.3. Performance Analysis

In the following, the performance of the CSU-accurate BMC technique from Section 5.1 (p. 70) is compared to the performance of a general purpose, *cycle-accurate* model checker.

A state-of-the-art, commercial model checker is applied to the verification of accessibility in a cycle-accurate hardware model of the largest benchmark (p93791) with MUX-based architecture. In each experiment, the following property is subject to verification: $P := \boldsymbol{G}\ [\neg \texttt{select}(\texttt{s})]$, where s is a random scan segment. Since this property does not specify if the selected segment s belongs to an active scan path, P is weaker than the property $\boldsymbol{G}\ [\neg \texttt{Active}(\texttt{s})]$ that can be checked in the CAM. (Function $\texttt{Active}(s)$ is not part of the cycle-accurate network model and is nontrivial to specify in the general-purpose tool.)

Table B.5 shows the time that is required to find a counterexample to property P with the cycle-accurate model checker (second column) and the proposed approach (third column). The cycle-accurate model checker exceeds the time limit of 1 hour in two experiments. In the remaining experiments, the solving time varies widely from 12 up to 364 s. In contrast, the proposed approach is successful in all the experiments and exhibits much more stable run-times below 0.21 s. This result clearly shows that the proposed CSU-accurate abstraction provides a great performance improvement over cycle-accurate models.

Design	Arch.	Variables	Clauses	Conflicts	$t_{\text{solve}}^{\text{init}}$ [s]	$t_{\text{solve}}^{\text{induct}}$ [s]
u226	SIB	2600	5622	1289	0.00	0.19
d281	SIB	3121	6645	1877	0.00	0.35
d695	SIB	9191	19545	4554	0.01	3.29
h953	SIB	2897	6179	1576	0.00	0.26
g1023	SIB	4195	8950	2339	0.00	0.65
f2126	SIB	2181	4623	965	0.00	0.10
q12710	SIB	1341	2884	658	0.00	0.05
p22810	SIB	15325	32605	8884	0.05	11.40
p34392	SIB	6518	13902	3568	0.01	1.66
p93791	SIB	34185	72633	18955	0.22	92.70
t512505	SIB	8386	17931	4783	0.06	4.03
a586710	SIB	2074	4442	1084	0.00	0.11
u226	MUX	2756	5195	1352	0.02	0.16
d281	MUX	3220	6098	1734	0.04	0.30
d695	MUX	9023	17111	4502	0.18	3.30
h953	MUX	3008	5688	1585	0.03	0.24
g1023	MUX	4411	8370	2321	0.08	0.61
f2126	MUX	2214	4209	1018	0.01	0.09
q12710	MUX	1419	2682	768	0.00	0.04
p22810	MUX	15352	29054	8225	0.54	8.98
p34392	MUX	6755	12795	3497	0.01	1.51
p93791	MUX	33318	63006	17725	0.60	72.00
t512505	MUX	8872	16787	4662	0.26	2.94
a586710	MUX	2200	4177	1124	0.01	0.12
c17	FLAT	123	317	12	0.00	0.00
c432	FLAT	861	1446	40	0.01	0.00
c499	FLAT	2163	4206	383	0.01	0.02
c880	FLAT	2019	3368	335	0.02	0.02
c1355	FLAT	2435	4513	407	0.02	0.03
c1908	FLAT	2853	4790	213	0.02	0.01
c2670	FLAT	8715	11962	2144	0.53	0.61
c3540	FLAT	4405	6479	298	0.01	0.01
c5315	FLAT	9929	13898	1594	0.33	0.41
c6288	FLAT	6121	9821	786	0.02	0.04
c7552	FLAT	12037	15427	2001	0.29	0.35

Table B.1.: Robustness verification effort

Design	Bug type	Clauses avg / max	Conflicts avg / max	Depth avg / max	$t_{\text{solve}}^{\text{avg}}$ [s]	$t_{\text{solve}}^{\text{max}}$ [s]
u226	path	7808 / 10369	338 / 679	0.97 / 1	0.12	0.2
d281	path	9259 / 11914	259 / 457	1.00 / 1	0.13	0.2
d695	path	25482 / 32989	401 / 694	1.00 / 1	0.46	0.7
h953	path	8248 / 8533	285 / 575	0.96 / 1	0.12	0.2
g1023	path	13031 / 16458	543 / 889	0.99 / 1	0.24	0.4
f2126	path	6139 / 8003	146 / 291	0.99 / 1	0.07	0.1
q12710	path	3845 / 3948	93 / 251	1.00 / 1	0.04	0.1
p22810	path	42526 / 55032	955 / 9617	1.00 / 2	1.31	13.8
p34392	path	17335 / 27144	578 / 3516	1.15 / 2	0.43	1.8
p93791	path	91082 / 118357	1509 / 16579	1.05 / 2	6.11	78.7
t512505	path	25505 / 33359	904 / 1818	0.99 / 1	0.68	1.4
a586710	path	5836 / 9236	255 / 1186	1.06 / 2	0.09	0.2
u226	control	7501 / 10110	373 / 1708	0.94 / 2	0.14	0.4
d281	control	8923 / 11526	362 / 2042	1.01 / 2	0.16	0.4
d695	control	25732 / 33013	628 / 4816	1.01 / 2	0.66	3.3
h953	control	8352 / 10857	386 / 800	0.99 / 1	0.15	0.3
g1023	control	12344 / 16778	480 / 1031	0.97 / 1	0.26	0.5
f2126	control	6307 / 8183	203 / 2023	0.99 / 2	0.09	0.4
q12710	control	3817 / 5310	127 / 923	1.00 / 2	0.05	0.1
p22810	control	42901 / 59812	1743 / 10149	1.09 / 2	2.43	15.0
p34392	control	20170 / 27690	1685 / 4310	1.53 / 2	0.91	2.1
p93791	control	94212 / 134023	3690 / 18330	1.17 / 2	15.42	107.0
t512505	control	25105 / 33139	980 / 2243	0.94 / 1	0.84	2.6
a586710	control	6292 / 9557	622 / 1788	1.36 / 2	0.14	0.3
u226	mux	7013 / 9889	303 / 915	0.80 / 1	0.14	0.2
d281	mux	8401 / 11389	279 / 1110	0.87 / 1	0.17	0.4
d695	mux	23878 / 25214	621 / 1586	0.95 / 1	0.76	1.6
h953	mux	7844 / 11160	374 / 819	0.88 / 1	0.16	0.2
g1023	mux	10909 / 16557	454 / 1155	0.75 / 1	0.28	0.7
f2126	mux	5480 / 8121	169 / 530	0.83 / 1	0.10	0.2
q12710	mux	3405 / 5170	88 / 411	0.77 / 1	0.04	0.1
p22810	mux	40673 / 54955	1528 / 10092	0.98 / 2	2.32	13.3
p34392	mux	17349 / 28780	1183 / 4627	1.26 / 2	0.82	2.9
p93791	mux	89575 / 118381	2211 / 20957	1.03 / 2	10.21	97.4
t512505	mux	23002 / 33546	802 / 2329	0.81 / 1	0.93	3.1
a586710	mux	5226 / 8934	297 / 1483	0.93 / 2	0.11	0.3

Table B.2.: Robustness verification effort for MUX-based RSNs with random design bugs

Design	Arch.	Clauses avg / max	Conflicts avg / max	Depth avg / max	$t_{\text{solve}}^{\text{max}}$ [s]	$t_{\text{solve}}^{\text{total}}$ [s]
u226	SIB	8506 / 10431	0 / 0	2.30 / 3	0.01	0.46
d281	SIB	10459 / 12482	0 / 0	2.39 / 3	0.02	0.73
d695	SIB	31269 / 36574	0 / 0	2.45 / 3	0.04	6.63
h953	SIB	9565 / 11594	0 / 0	2.34 / 3	0.01	0.57
g1023	SIB	13822 / 16826	0 / 0	2.33 / 3	0.02	1.19
f2126	SIB	7240 / 8698	0 / 0	2.37 / 3	0.01	0.41
q12710	SIB	4471 / 5368	0 / 0	2.37 / 3	0.01	0.11
p22810	SIB	52553 / 77376	0 / 0	2.47 / 4	0.06	19.36
p34392	SIB	23954 / 33028	0 / 0	2.69 / 4	0.03	3.40
p93791	SIB	119056 / 172082	0 / 0	2.53 / 4	0.15	101.83
t512505	SIB	27653 / 33685	0 / 0	2.32 / 3	0.04	5.07
a586710	SIB	7231 / 10521	0 / 0	2.51 / 4	0.01	0.32
u226	MUX	15379 / 20617	1.6 / 8	3.53 / 5	0.02	0.97
d281	MUX	18467 / 23797	1.7 / 9	3.70 / 5	0.03	1.35
d695	MUX	52488 / 65189	1.9 / 10	3.87 / 5	0.06	12.90
h953	MUX	16934 / 22293	2.4 / 17	3.61 / 5	0.02	1.31
g1023	MUX	24855 / 32917	2.0 / 27	3.58 / 5	0.04	2.70
f2126	MUX	12484 / 16213	2.2 / 16	3.67 / 5	0.02	0.66
q12710	MUX	8072 / 10573	2.4 / 8	3.63 / 5	0.01	0.26
p22810	MUX	91012 / 150787	5.1 / 86	3.91 / 7	0.19	39.62
p34392	MUX	44745 / 67367	8.9 / 93	4.38 / 7	0.08	8.67
p93791	MUX	201127 / 322891	6.4 / 222	4.06 / 7	0.53	197.39
t512505	MUX	50169 / 66465	3.8 / 38	3.58 / 5	0.06	10.60
a586710	MUX	13506 / 22105	4.1 / 41	3.96 / 7	0.02	0.72
c17	FLAT	372 / 504	0 / 0	1.29 / 2	0.00	0.00
c432	FLAT	3811 / 5403	0.5 / 9	1.16 / 2	0.01	0.08
c499	FLAT	9740 / 12102	0 / 0	1.44 / 2	0.01	0.32
c880	FLAT	8216 / 11186	2.4 / 27	1.24 / 2	0.01	0.33
c1355	FLAT	11574 / 14358	1.1 / 4	1.44 / 2	0.02	0.41
c1908	FLAT	14337 / 18183	4.0 / 36	1.38 / 2	0.01	0.32
c2670	FLAT	31772 / 42480	21.2 / 159	1.29 / 2	0.04	6.36
c3540	FLAT	22913 / 31510	6.1 / 109	1.19 / 2	0.02	0.73
c5315	FLAT	43937 / 59037	22.4 / 203	1.26 / 2	0.08	7.76
c6288	FLAT	39725 / 47760	200.1 / 1670	1.50 / 2	0.12	1.90
c7552	FLAT	54705 / 76078	36.3 / 629	1.18 / 2	0.12	10.47

Table B.3.: Accessibility verification effort

Design	Bug type	Found [%]	Inaccessible [%]	$t_{\text{solve}}^{\text{avg}}$ [s]	$t_{\text{solve}}^{\text{max}}$ [s]
u226	path	100%	36.6%	5.2	23
d281	path	100%	23.0%	4.4	8
d695	path	100%	16.5%	26.5	35
h953	path	100%	16.6%	3.1	26
g1023	path	100%	11.6%	4.8	59
f2126	path	100%	33.4%	3.0	12
q12710	path	100%	22.7%	0.8	1
p22810	path	100%	8.9%	65.5	1096
p34392	path	100%	28.1%	28.3	50
p93791	path	100%	7.5%	217.7	269
t512505	path	100%	10.0%	18.1	320
a586710	path	100%	34.8%	3.0	11
u226	control	9%	2.6%	1.2	25
d281	control	7%	1.8%	1.4	40
d695	control	2%	1.2%	11.0	389
h953	control	6%	0.7%	0.8	4
g1023	control	2%	1.0%	2.4	83
f2126	control	13%	5.7%	1.3	21
q12710	control	18%	8.8%	0.7	7
p22810	control	2%	0.1%	21.2	46
p34392	control	4%	0.7%	5.2	43
p93791	control	3%	2.0%	221.3	5829
t512505	control	4%	2.7%	17.2	378
a586710	control	12%	5.4%	1.3	18
u226	mux	100%	27.5%	5.7	28
d281	mux	100%	20.8%	5.8	37
d695	mux	100%	12.3%	43.3	298
h953	mux	100%	16.8%	4.5	33
g1023	mux	100%	25.9%	20.4	97
f2126	mux	100%	30.3%	4.9	19
q12710	mux	100%	30.7%	1.4	7
p22810	mux	100%	10.9%	124.9	1100
p34392	mux	100%	19.4%	38.2	187
p93791	mux	100%	6.0%	380.8	6956
t512505	mux	100%	19.5%	70.4	618
a586710	mux	100%	34.3%	4.5	16

Table B.4.: Accessibility verification effort for faulty MUX-based RSNs

Exp. No.	Cycle-Accurate MC	CSU-Accurate BMC	Speedup
1	79 s	0.19 s	415x
2	364 s	0.13 s	2,800x
3	12 s	0.12 s	100x
4	38 s	0.14 s	271x
5	45 s	0.12 s	375x
6	27 s	0.21 s	129x
7	28 s	0.20 s	140x
8	>1 h	0.21 s	>17,000x
9	>1 h	0.14 s	>25,000x
10	54 s	0.13 s	415x

Table B.5.: Performance comparison of a cycle-accurate model checker and the proposed CSU-accurate BMC

C. Results: Access Optimization

The access optimization technique developed in Chapter 6 (p. 91) is evaluated on MUX-based, SIB-based and flat scan architectures from Appendix A (p. 141). The implementation of the pattern generation procedure is based on the *clasp* toolkit [Gebser11] which includes a Boolean SAT solver and a pseudo-Boolean optimization engine. The optimization of access time is allowed up to three additional CSU operations over the minimal number of CSUs required for an access. The pattern generation procedure is executed in four parallel processes (cf. Figure 6.3, p. 98) on an Intel Core2 CPU with four cores operating at 2.83 GHz.

The efficiency of the pattern generation procedure is evaluated in 1000 experiments per benchmark RSN. In each experiment, the shortest pattern that *merges* read or write accesses to 10 randomly chosen scan segments is searched for. It is assumed that the *update* and *capture* phases of a CSU operation take one cycle each ($D := 2$ in formula (6.5), p. 95). The resulting access time improvement is analyzed w.r.t. a pure SAT-based solution.

The generated access patterns are validated by cycle-accurate simulation. For this purpose, the RSN models are automatically translated to hardware Verilog models. The generated access patterns are used as stimuli for the network's primary scan-input. During simulation, assertions verify that the access is performed correctly.

C.1. MUX-based Architecture

Table C.1 presents the pattern generation statistics for the MUX-based scan architecture: Column "No optimization" gives the results of SAT-based pattern generation for the minimal number of CSU operations (*without* optimization). For the 1000 experiments, column $n_{\mathtt{min}}$ gives the average and maximal number of CSUs that are required to implement an access to 10 random scan segments. Column $t^{\mathtt{avg}}_{\mathtt{solve}}$ gives the average pattern generation effort per access. The average *unoptimized* access time in clock cycles is given in column `cycles`.

Access time reduction of the proposed pattern generation procedure is evaluated in two series of experiments, limiting the optimization effort to 2 and 20 s per access. The corresponding columns "Opt. effort" in Table C.1 give the average and maximal *access time reduction* (column `reduction`) w.r.t. the unoptimized, SAT-based solution. The average number of additional CSU operations (in addition to $n_{\mathtt{min}}$) that are required to obtain the local minimum is given in column `depth`.

For the majority of benchmarks, an access time reduction of over 10x is achieved for at least one pattern. For the t512505 benchmark, the access time is reduced by up to 121x. This shows that access optimization is crucial to prevent solutions with prohibitive access time or data volume. The proposed method also reduces unnecessary access overhead: For most of the RSNs, the average access time is nearly halved within 2 s of computational time. Note that the reduction of access time leads to a proportional reduction in scan data volume.

The results presented in Table C.1 are obtained with the pattern generation procedure that terminates as soon as a local access time minimum is found. Potentially, a shorter access may be found if more CSU operations are allowed. Further experiments with up to 6 additional CSU operations over $n_{\mathtt{min}}$, however, do not result in any further access time improvement.

C.2. SIB-based Architecture

Access time optimization in SIB-based architectures reduces to a simple decision problem, as discussed in the introduction to Chapter 6 (p. 91). Although the proposed pattern generation procedure is not required in this case, it is evaluated for SIB-based benchmarks for the sake of completeness.

The results of access pattern generation for SIB-based architectures are presented in Table C.2 (contents are analogous to Table C.1). The proposed access optimization procedure reduces the access time by a factor of up to 1.9 w.r.t. the unoptimized solution obtained with the SAT solver. In contrast to MUX-based architectures, the local minimum is always found for the minimal number of CSU operations that is

	No optimization			Opt. effort 2 s		Opt. effort 20 s	
Design	$n_{\mathtt{min}}$ avg / max	$t^{\mathtt{avg}}_{\mathtt{solve}}$ [s]	`cycles` [cycles]	`depth` avg	`reduction` avg / max	`depth` avg	`reduction` avg / max
u226	5.9 / 7	0.02	922	0.42	1.57 / 10.0x	0.46	1.57 / 10.0x
d281	5.8 / 7	0.02	2342	0.72	1.81 / 19.7x	0.81	1.82 / 19.7x
d695	5.9 / 7	0.07	2752	0.58	1.81 / 9.1x	0.68	1.89 / 9.1x
h953	5.8 / 7	0.02	3240	0.83	1.92 / 7.4x	0.93	1.93 / 7.4x
g1023	5.7 / 7	0.03	2293	0.60	1.93 / 7.1x	0.65	1.97 / 7.1x
f2126	5.7 / 7	0.01	9574	0.88	1.81 / 82.5x	0.92	1.82 / 82.5x
q12710	5.7 / 7	0.01	17134	0.86	1.74 / 11.8x	0.86	1.74 / 11.8x
p22810	6.0 / 9	0.14	8686	0.60	2.06 / 22.8x	0.66	2.30 / 34.7x
p34392	6.9 / 10	0.06	13368	0.62	1.78 / 61.3x	0.74	2.00 / 62.2x
p93791	6.7 / 10	0.51	41790	0.30	1.30 / 4.8x	0.35	1.45 / 10.1x
t512505	5.6 / 7	0.07	28422	0.49	2.11 / 121.3x	0.48	2.26 / 121.3x
a586710	6.5 / 10	0.02	28672	1.21	2.09 / 93.0x	1.28	2.15 / 93.0x

Table C.1.: Access time reduction (`reduction`) for the MUX-based scan architecture w.r.t. unoptimized solution (`cycles`)

required to implement the access (`depth` is zero). The best solution is usually found within 2 s of optimization time, and extending the effort to 20 s achieves only a minor access time reduction for larger benchmarks.

C.3. Flat Architecture

Table C.3 presents the optimization results for the flat scan architecture (contents are analogous to Table C.1). Up to 3 CSU operations are required to perform an unoptimized access to 10 random scan segments. The optimal solution is always found for the minimal number of CSU operations (`depth` is zero). Optimization with an effort of 2 s improves the access time by a factor of up to 3.2x. With an effort of 20 s, the average access time is slightly improved for largest benchmarks (c5315, c6288, c7552).

	No optimization			Opt. effort 2 s		Opt. effort 20 s	
Design	$n_{\texttt{min}}$ avg / max	$t_{\texttt{solve}}^{\texttt{avg}}$ [s]	`cycles` [cycles]	`depth` avg	`reduction` avg / max	`depth` avg	`reduction` avg / max
u226	2.6 / 3	0.01	1129	0	1.36 / 1.8x	0	1.36 / 1.8x
d281	2.6 / 3	0.01	2711	0	1.35 / 1.9x	0	1.35 / 1.9x
d695	2.6 / 3	0.02	5287	0	1.30 / 1.6x	0	1.32 / 1.6x
h953	2.7 / 3	0.01	3765	0	1.33 / 1.8x	0	1.33 / 1.8x
g1023	2.7 / 3	0.01	2825	0	1.36 / 1.7x	0	1.36 / 1.7x
f2126	2.6 / 3	0.01	11522	0	1.34 / 1.8x	0	1.34 / 1.8x
q12710	2.6 / 3	0.00	20293	0	1.29 / 1.7x	0	1.29 / 1.7x
p22810	2.8 / 4	0.04	14596	0	1.20 / 1.6x	0	1.31 / 1.7x
p34392	3.2 / 4	0.02	14706	0	1.33 / 1.8x	0	1.35 / 1.8x
p93791	2.9 / 4	0.12	37423	0	1.09 / 1.2x	0	1.19 / 1.5x
t512505	2.6 / 3	0.02	44779	0	1.29 / 1.7x	0	1.37 / 1.7x
a586710	2.8 / 4	0.01	30538	0	1.34 / 1.9x	0	1.35 / 1.9x

Table C.2.: Access time reduction (`reduction`) for the SIB-based scan architecture w.r.t. unoptimized solution (`cycles`)

	No optimization			Opt. effort 2 s		Opt. effort 20 s	
Design	$n_{\texttt{min}}$ avg / max	$t_{\texttt{solve}}^{\texttt{avg}}$ [s]	`cycles` [cycles]	`depth` avg	`reduction` avg / max	`depth` avg	`reduction` avg / max
c17	1.6 / 2	0.00	79	0	1.06 / 1.4x	0	1.06 / 1.4x
c432	1.5 / 2	0.00	270	0	1.25 / 2.2x	0	1.25 / 2.2x
c499	1.9 / 2	0.01	828	0	1.24 / 2.2x	0	1.24 / 2.2x
c880	1.8 / 2	0.01	752	0	1.26 / 2.1x	0	1.26 / 2.1x
c1355	1.9 / 2	0.01	779	0	1.17 / 1.6x	0	1.17 / 1.6x
c1908	1.9 / 2	0.01	1036	0	1.89 / 3.2x	0	1.89 / 3.2x
c2670	1.9 / 3	1.76	3602	0	1.23 / 1.4x	0	1.23 / 1.4x
c3540	1.9 / 3	0.01	710	0	1.41 / 2.2x	0	1.41 / 2.2x
c5315	1.9 / 3	0.03	3733	0	1.19 / 1.4x	0	1.21 / 1.7x
c6288	2.0 / 2	0.33	691	0	1.10 / 1.8x	0	1.12 / 2.4x
c7552	1.8 / 2	0.04	3860	0	1.11 / 1.5x	0	1.25 / 1.7x

Table C.3.: Access time reduction (`reduction`) for the flat scan architecture w.r.t. unoptimized solution (`cycles`)

D. Results: Access Protection

The cost of the filter-based protection method from Chapter 7 (p. 103) is evaluated on MUX-based, SIB-based and flat scan architectures presented in Appendix A (p. 141). Optimal restricted accesses are generated with the approach presented in Chapter 6 (p. 91) extended with the required constraints, as explained in Section 7.2 (p. 106). The sequence filters are constructed according to the algorithm from Section 7.3.1 (p. 109). The resulting state diagrams are transformed automatically into Verilog hardware models and synthesized for the Nangate 45nm open cell library [N45] with area optimization goal.

The area overhead is calculated w.r.t. the area of the scan network, which includes no system logic. The actual area overhead w.r.t. the full chip is expected to be much lower. The resulting operating frequency of all evaluated filters is above 300 MHz, which is significantly more than the usual JTAG clock speed (10 to 100 MHz).

Restricted accesses are generated for random samples of target scan segments. Except for scan segments that configure the active scan path, all remaining scan segments are considered protected. The results discussed in the following sections, including the area overhead and the number of FSM states, represent average values acquired from the evaluation of 10 filters built for different random samples of target segments. The standard deviation of the area overhead is below 3% in the experiments with SIB- and MUX-based scan architectures, and below 7% for flat architectures.

D.1. Individual Accesses

For each benchmark RSN, sequence filters are constructed for 10, 20, and 100 restricted accesses patterns. Each pattern realizes the shortest access to a single target scan segment, and the remaining segments are considered protected. This is relevant for low-latency access to individual segments.

Table D.1 presents the properties of the sequence filters for individual accesses. The design name, scan architecture type (SIB, MUX, or flat), and RSN area are given in the first three columns. Columns 4-7 give the properties of sequence filters that allow 10 restricted accesses: the cumulative length of the sequences, the number of states in the state diagram, and the filter's area overhead w.r.t. the RSN area (column 3). The properties of filters allowing 20 and 100 accesses are given in columns 8-11 and 12-15, respectively.

The filter size depends on the number of allowed accesses: In SIB- and MUX-based architectures, the area overhead ranges from 0.2 to 2.7% for 10 individual accesses (column 7). For 20 accesses, the area is 0.3 to 4.3% (column 11), and for 100 accesses it rises up to 10.6% (column 15). In most cases, the increase in area overhead is less than the increase in the number of allowed accesses. Note that twelve of the SIB- and MUX-based RSNs include about a hundred or less scan segments (cf. Appendix A). Even if access to a high fraction or all of their scan segments is allowed, the area overhead is below 1.7% for f2126, q12710, and a586710 (column 15 in Table D.1).

The size of a sequence filter is proportional to the number of states in the filter's state diagram, which, in turn, is proportional to the length of allowed sequences. Thanks to state merging and sequence collapsing (see Section 7.3.2, p. 112), the number of states in the state diagram is significantly less than the cumulative sequence length: Merging and collapsing reduces the number of states by a factor of 1.9 for 10 accesses to d695-MUX, and by up to 312x for 100 accesses to q12710-SIB.

Since the flat scan architecture has little practical significance (cf. Appendix A), the results for flat benchmarks are presented only for the sake of completeness. The absolute area of sequence filters for the flat RSNs is close to the area required for other benchmark types. However, the relative protection cost is high, with up to 25% for 10 restricted accesses already. This is caused primarily by the small size of the flat benchmarks and the very high number of scan segments that configure the active scan path (greater than the number of data scan segments, cf. Figure A.3, p. 144). Moreover, the restricted accesses patterns often differ at early bit positions (bits that are shifted first), which reduces the efficiency of state merging and collapsing (cf. Section 7.3.2, p. 112).

D.2. Concurrent Accesses

In the second series of experiments, sequence filters are constructed for the concurrent access to 100 random scan segments realized by 1, 5, 10 and 20 restricted access patterns. The concurrent access is efficient if the target segments are usually accessed together.

Table D.2 shows the properties of sequence filters for the concurrent access. In columns 3-5, the filters for a *single* concurrent access to 100 segments are described, including the sequence length, the number of states in the filter's state diagram, and the area overhead. The properties of filters for 5 accesses à 20 segments, 10 accesses à 10 segments, and 20 accesses à 5 segments are given in the consecutive columns.

For 20 accesses à 5 segments, the area overhead (column 14) is close to the area for individual accesses (cf. column 15 in Table D.1). However, if the access to all 100 segments is realized with a *single* access pattern, the cost (column 5) is reduced by a factor of 3 to 16 compared with the cost of individual accesses. This is explained by the lower length of the concurrent sequences compared with the cumulative length of individual sequences. Thus, if the segments are often accessed together, concurrent access has two benefits: The access time is lower, and the resulting sequence filter is smaller.

Benchmark			10 Restricted Accesses				20 Restricted Accesses				100 Restricted Accesses			
		Area	Length	States	Area		Length	States	Area		Length	States	Area	
Design *(1)*	Arch. *(2)*	[μm^2] *(3)*	[cycles] *(4)*	[#] *(5)*	[μm^2] *(6)*	[+%] *(7)*	[cycles] *(8)*	[#] *(9)*	[μm^2] *(10)*	[+%] *(11)*	[cycles] *(12)*	[#] *(13)*	[μm^2] *(14)*	[+%] *(15)*
d281	SIB	58 747	1 265	394	1 196	+2.04%	2 603	667	2 238	+3.81%	12 969	1 641	5 187	+8.83%
d695	SIB	126 777	1 433	733	1 926	+1.52%	2 937	1 369	3 683	+2.91%	14 473	4 670	11 581	+9.14%
h953	SIB	85 133	1 790	368	1 139	+1.34%	3 565	587	2 019	+2.37%	17 317	1 303	4 177	+4.91%
g1023	SIB	81 396	1 520	495	1 584	+1.95%	2 956	849	2 460	+3.02%	15 103	2 177	6 782	+8.33%
f2126	SIB	239 902	5 521	342	1 037	+0.43%	10 060	570	1 880	+0.78%	49 192	1 095	3 635	+1.52%
q12710	SIB	397 483	12 068	178	726	+0.18%	26 087	265	1 055	+0.27%	125 892	404	1 379	+0.35%
p22810	SIB	453 537	2 436	1 009	2 718	+0.60%	4 993	1 800	4 876	+1.08%	24 096	6 491	18 361	+4.05%
p34392	SIB	352 290	3 699	711	2 002	+0.57%	7 069	1 307	3 526	+1.00%	32 747	4 541	12 527	+3.56%
p93791	SIB	1 486 289	3 104	1 336	3 199	+0.22%	6 109	2 537	6 279	+0.42%	31 582	10 339	25 004	+1.68%
t512505	SIB	1 167 569	8 063	1 058	2 629	+0.23%	16 295	1 884	4 798	+0.41%	71 239	5 877	15 672	+1.34%
a586710	SIB	634 087	14 723	319	1 111	+0.18%	28 189	533	1 964	+0.31%	132 255	988	3 333	+0.53%
d281	MUX	58 979	1 411	459	1 603	+2.72%	2 597	810	2 548	+4.32%	14 243	1 927	6 225	+10.56%
d695	MUX	127 007	1 434	760	2 131	+1.68%	2 875	1 358	3 700	+2.91%	14 384	5 016	13 253	+10.44%
h953	MUX	85 349	1 596	349	1 309	+1.53%	3 287	620	2 475	+2.90%	17 030	1 493	5 591	+6.55%
g1023	MUX	81 727	1 295	501	1 826	+2.23%	2 712	851	2 778	+3.40%	14 473	2 299	7 527	+9.21%
f2126	MUX	240 021	5 009	402	1 368	+0.57%	9 899	701	2 423	+1.01%	51 279	1 377	4 072	+1.70%
q12710	MUX	397 592	13 242	255	1 068	+0.27%	27 175	388	1 531	+0.39%	131 592	597	2 881	+0.72%
p22810	MUX	454 107	2 258	882	2 621	+0.58%	4 488	1 625	5 540	+1.22%	21 730	6 039	19 707	+4.34%
p34392	MUX	352 699	3 075	767	2 460	+0.70%	6 216	1 379	4 601	+1.30%	33 475	4 394	13 746	+3.90%
p93791	MUX	1 486 772	3 075	1 302	3 531	+0.24%	6 060	2 392	7 141	+0.48%	31 677	9 922	24 108	+1.62%
t512505	MUX	1 168 310	7 161	832	2 419	+0.21%	14 138	1 520	5 179	+0.44%	68 719	4 994	15 230	+1.30%
a586710	MUX	634 258	13 027	401	1 523	+0.24%	27 447	643	2 551	+0.40%	135 388	1 185	4 493	+0.71%
c432	FLAT	3 950	1 100	360	973	+24.64%	2 200	423	1 174	+29.73%	11 000	439	1 075	+27.22%
c499	FLAT	16 098	1 200	524	936	+5.82%	2 400	891	1 606	+9.98%	12 000	1 888	3 203	+19.90%
c880	FLAT	13 473	3 476	813	1 712	+12.71%	6 912	1 289	2 666	+19.79%	33 959	2 023	3 147	+23.36%
c1355	FLAT	16 097	1 200	666	1 179	+7.33%	2 400	1 144	2 080	+12.93%	12 000	2 437	3 769	+23.41%
c1908	FLAT	12 633	2 239	411	1 372	+10.86%	4 537	624	2 138	+16.93%	22 665	949	2 138	+16.93%
c2670	FLAT	70 720	16 527	3 286	5 179	+7.32%	32 757	6 003	9 098	+12.87%	163 896	20 287	34 936	+49.40%
c3540	FLAT	11 731	4 046	545	1 954	+16.66%	7 961	800	2 901	+24.74%	39 364	1 125	3 379	+28.80%
c5315	FLAT	62 276	23 375	1 891	3 400	+5.46%	47 515	3 369	5 962	+9.57%	235 185	7 753	11 820	+18.98%
c6288	FLAT	17 313	1 039	489	1 787	+10.33%	2 078	790	2 784	+16.08%	10 289	1 541	6 970	+40.26%
c7552	FLAT	55 774	25 840	2 029	4 891	+8.77%	50 892	3 326	7 484	+13.42%	258 776	10 259	16 836	+30.19%

Table D.1.: Hardware overhead of the sequence filters w.r.t. RSN area for individual accesses

Benchmark		1 Access to 100 Segments			5 Accesses à 20 Segments			10 Accesses à 10 Segments			20 Accesses à 5 Segments		
		Length	States	Area	Length	States	Area	Length	States	Area	Length	States	Area
Design	Arch.	[cycles]	[#]	[+%]	[cycles]	[#]	[+%]	[cycles]	[#]	[+%]	[cycles]	[#]	[+%]
(1)	*(2)*	*(3)*	*(4)*	*(5)*	*(6)*	*(7)*	*(8)*	*(9)*	*(10)*	*(11)*	*(12)*	*(13)*	*(14)*
d281	SIB	3 945	181	+1.23%	7 570	688	+4.40%	8 998	1 149	+7.29%	9 157	1 849	+10.60%
d695	SIB	5 618	447	+0.97%	6 868	1 685	+4.10%	8 295	2 857	+6.58%	9 762	4 406	+9.59%
h953	SIB	5 712	171	+0.83%	12 430	659	+2.88%	13 470	1 082	+4.81%	13 756	1 594	+6.54%
g1023	SIB	5 488	246	+1.13%	9 066	862	+3.68%	9 746	1 344	+5.70%	10 453	2 031	+8.99%
f2126	SIB	15 883	129	+0.22%	44 454	510	+0.84%	43 597	875	+1.28%	47 264	1 476	+2.27%
q12710	SIB	26 220	82	+0.11%	124 930	292	+0.29%	122 825	595	+0.61%	124 576	946	+0.96%
p22810	SIB	12 956	752	+0.52%	14 526	2 397	+1.57%	15 967	3 463	+2.17%	16 235	4 704	+2.98%
p34392	SIB	22 667	448	+0.30%	24 524	1 503	+1.37%	25 337	2 597	+2.40%	27 549	4 091	+3.53%
p93791	SIB	18 518	1 518	+0.28%	21 580	5 274	+0.83%	24 026	7 460	+1.14%	25 908	9 357	+1.34%
t512505	SIB	58 833	455	+0.11%	57 912	1 345	+0.38%	61 220	2 149	+0.57%	62 592	3 390	+0.87%
a586710	SIB	41 748	144	+0.11%	130 674	552	+0.37%	127 817	972	+0.58%	126 344	1 526	+0.87%
d281	MUX	3 970	164	+1.01%	7 932	722	+4.18%	9 272	1 272	+7.26%	9 882	2 023	+12.05%
d695	MUX	5 538	349	+0.63%	6 696	1 533	+3.21%	8 073	2 690	+6.05%	9 633	4 249	+8.64%
h953	MUX	5 748	170	+0.74%	12 819	746	+2.99%	13 549	1 267	+5.29%	14 565	1 879	+7.70%
g1023	MUX	5 542	250	+0.95%	9 445	983	+3.77%	9 977	1 504	+6.62%	10 254	2 222	+10.32%
f2126	MUX	15 907	122	+0.38%	45 334	543	+0.90%	44 997	985	+1.51%	46 573	1 696	+2.31%
q12710	MUX	26 237	82	+0.11%	125 138	296	+0.25%	122 744	703	+0.65%	126 538	1 204	+1.15%
p22810	MUX	15 721	688	+0.42%	15 769	2 282	+1.60%	14 651	3 250	+2.21%	16 345	4 421	+2.82%
p34392	MUX	24 240	449	+0.30%	22 881	1 678	+1.48%	25 716	2 835	+2.39%	26 145	4 324	+3.80%
p93791	MUX	31 630	1 502	+0.20%	25 471	4 812	+0.71%	24 889	6 872	+1.14%	25 344	8 730	+1.38%
t512505	MUX	59 633	480	+0.10%	62 409	1 423	+0.37%	62 376	2 131	+0.60%	63 729	3 234	+0.85%
a586710	MUX	77 856	156	+0.12%	133 801	722	+0.44%	129 003	1 266	+0.75%	133 670	1 955	+1.15%
c432	FLAT	302	78	+8.70%	1 510	78	+8.53%	3 020	78	+8.62%	4 760	781	+51.67%
c499	FLAT	1 112	88	+2.59%	3 640	384	+5.81%	4 080	745	+11.13%	4 960	1 445	+17.15%
c880	FLAT	990	127	+3.86%	4 319	604	+12.08%	6 408	1 169	+21.73%	9 790	2 223	+36.79%
c1355	FLAT	1 112	88	+2.57%	3 640	417	+6.28%	4 080	823	+11.41%	4 960	1 619	+21.33%
c1908	FLAT	936	73	+2.87%	3 838	310	+9.53%	4 787	612	+18.50%	6 816	1 183	+31.58%
c2670	FLAT	5 012	630	+1.88%	10 469	2 033	+5.75%	18 383	4 031	+9.32%	36 130	8 344	+13.95%
c3540	FLAT	919	160	+5.61%	4 366	592	+20.77%	6 235	1 078	+38.60%	10 281	1 833	+60.24%
c5315	FLAT	4 798	544	+2.61%	17 317	1 949	+6.78%	30 234	3 576	+11.25%	58 456	6 593	+20.77%
c6288	FLAT	1 160	105	+2.64%	3 803	329	+7.70%	4 191	640	+14.46%	4 945	1 244	+32.39%
c7552	FLAT	3 780	421	+1.68%	14 864	2 074	+9.94%	30 942	4 107	+16.23%	61 993	7 728	+29.92%

Table D.2.: Hardware overhead of the sequence filters w.r.t. RSN area for concurrent accesses

E. Curriculum Vitae of the Author

Rafał Baranowski received his MS degree in Electronics and Telecommunications from the Silesian University of Technology, Poland, in 2007. In 2008 he joined the Institute of Computer Architecture and Computer Engineering (*Institut für Technische Informatik*, ITI) at the University of Stuttgart, Germany.

The author has been working for the last six years as a research and teaching assistant under the supervision of Prof. Dr. rer. nat. habil. H.-J. Wunderlich. He was involved in the research projects "RM-BIST: Reliability Monitoring and Managing Built-In Self Test", "OASIS: Online Failure Prediction for Microelectronic Circuits Using Aging Signatures", and "DFG Forschergruppe 460: Development of Concepts and Methods for Reliability Evaluation of Mechatronic Systems in Early Development Phases" supported by the German Research Foundation (DFG), as well as "VIGONI: Combining Fault Tolerance and Offline Test Strategies for Nanoscaled Electronics" supported by the German Academic Exchange Service (DAAD).

The author supported several graduate courses including "Advanced Processor Architectures", "Elements of High-Performance RISC Processors — Design and Synthesis Lab", "Design and Test of Systems-on-a-Chip", "Hardware-based Fault Tolerance", and the undergraduate course "Hardware Lab". He also supervised students in several seminars, master's theses, and undergraduate projects.

The author's research interests include formal hardware verification and hardware security.

Index

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

Rafał Baranowski