Preprint

Reconfigurable Scan Networks: Modeling, Verification, and Optimal Pattern Generation

Rafal Baranowski, Michael A. Kochte, and Hans-Joachim Wunderlich

ACM Transactions on Design Automation of Electronic Systems (TODAES), Volume 20 Issue 2, Article No. 30, February 2015

url: dl.acm.org/citation.cfm?id=2699863

This is the author's "personal copy" of the final, accepted version of the paper published by ACM.

ACM COPYRIGHT NOTICE

© 2015 ACM. Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Efficient access to on-chip instrumentation is a key requirement for post-silicon validation, test, debug, bringup, and diagnosis. Reconfigurable scan networks, as proposed by e.g. IEEE P1687 and IEEE Std 1149.1-2013, emerge as an effective and affordable means to cope with the increasing complexity of on-chip infrastructure.

Reconfigurable scan networks are often hierarchical and may have complex structural and functional dependencies. Common approaches for scan verification based on static structural analysis and functional simulation are not sufficient to ensure correct operation of these types of architectures. To access an instrument in a reconfigurable scan network, a scan-in bit sequence must be generated according to the current state and structure of the network. Due to sequential and combinational dependencies, the access pattern generation process (pattern retargeting) poses a complex decision and optimization problem.

This article presents the first generalized formal model that considers structural and functional dependencies of reconfigurable scan networks and is directly applicable to P1687-based and 1149.1-2013-based scan architectures. This model enables efficient formal verification of complex scan networks, as well as automatic generation of access patterns. The proposed pattern generation method supports concurrent access to multiple target scan registers (access merging) and generates short scan-in sequences.

Categories and Subject Descriptors: B.7.2 [Integrated Circuits]: Design Aids

General Terms: Algorithms, Verification, Performance

Additional Key Words and Phrases: Access pattern generation, design for debug & diagnosis, DFT verification, JTAG, JTAG, pattern retargeting, P1687, reconfigurable scan network, 1149.1-2013

ACM Reference Format:
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION
To assure short time to market and high system dependability, a significant fraction of nanoelectronic systems is devoted to embedded instrumentation that facilitates test, post-silicon validation, debug, and bringup. On-chip instrumentation is also used during operation in the field for power-up initialization, system monitoring, reprogramming, error management, and repair [Stollon 2011; Larsson and Sibin 2012; Rearick and Volz 2006].

This work was supported by the German Research Foundation (DFG) under grant WU 245/17-1 (ACCESS). This article is an extended version of [Baranowski et al. 2012] and [Baranowski et al. 2013].

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© YYYY ACM 1084-4309/YYY/01-ARTA $15.00
DOI: http://dx.doi.org/10.1145/0000000.0000000
Embedded instrumentation is most often accessed using scan techniques via the four-wire Test Access Port (TAP) defined by IEEE Std 1149.1-1990 (a.k.a. JTAG) [1149.1 2013]. The 1149.1 TAP has become a de facto standard for efficient, low-cost access to on-chip instruments [Rearick et al. 2005; Larsson and Sibin 2012]. Embedded instruments are interfaced with the 1149.1 circuitry using scan registers called Test Data Registers (TDR). To access an instrument via 1149.1 TAP, an instruction word enabling the corresponding TDR is first loaded into the 1149.1 Instruction Register (IR). Similarly, in scan networks based on IEEE 1500 wrappers, the content of the Wrapper Instruction Register (WIR) defines which registers (scan chains) are currently accessible.

The flexibility of conventional 1149.1-based (1500-based) scan architectures is limited: Only the IR (WIR) may configure the scan path through which scan data are shifted, i.e., a single register defines which instruments (registers) are currently accessible. When the number of instruments interfaced with such architectures is high, either the access time or the area and routing overhead is high [Rearick et al. 2005; Larsson and Sibin 2012].

To allow flexible access to individual instruments and reduce performance and area overhead, custom scan architectures can be used, which are here collectively referred to as Reconfigurable Scan Networks (RSN). In RSNs, the path through which scan data are shifted (scan path or access mode) is determined by the state of many registers distributed arbitrarily over the network. In principle, conventional 1149.1-based and 1500-based scan networks are RSNs, but their reconfigurability is limited to a single register. Examples of RSNs with distributed configuration include the recent IEEE Std 1149.1-2013, which proposes segment selectors and TDRs with hierarchically excludable segments [1149.1 2013]. The ongoing effort IEEE P1687, also known as IJTAG (Internal JTAG), allows RSNs with nearly arbitrary, user-defined structure and functionality [Rearick et al. 2005; Eklow and Bennetts 2006; Larsson and Sibin 2012]. Such RSNs emerge as an effective means to access the instrumentation of complex Systems-on-a-Chip (SoC).

An example of an RSN compliant with IEEE P1687 is given in Figure 1. Scan data are shifted from the primary scan-input, through a subset of scan registers called scan segments, to the primary scan-output. The scan path (access mode) is configured by bits $a$ and $b$ of Segment 1. Segments 2 and 3 can be used e.g. as an interface to on-chip instruments.

The time to access a scan segment in an RSN is proportional to the length of the scan path. Access time can be significantly reduced by choosing access modes in which irrelevant segments are bypassed. Given the target pattern for the target scan segment, the process of computing the required scan-in sequence (scan data) is called access pattern generation, or pattern retargeting in IEEE P1687. For instance in Figure 1, given a pattern for Segment 2, access pattern generation is the search for a scan-in sequence that first sets $ab = 01$ or $ab = 10$ and then applies the target pattern to Segment 2.

The high performance and flexibility offered by RSNs comes at a price: As the number of possible access modes can be exponential in the RSN size, existing verification algorithms for traditional scan networks cannot efficiently handle them. General-purpose formal verification tools also face scalability issues in deeply sequential circuits such as RSNs. For similar reasons, designing an access pattern generation algorithm that works for arbitrary RSNs and guarantees minimal access time is challenging.

This article presents a novel modeling method based on temporal abstraction for complex reconfigurable scan networks. It is applicable to a wide range of RSN architectures, including—but not limited to—structures compliant with IEEE P1687. Compared with cycle-accurate models, the proposed RSN model has a significantly reduced
sequential depth. It enables efficient formal verification and access pattern generation for arbitrary reconfigurable scan networks.

The next section reviews the state of the art and discusses the contributions of this work. Section 3 introduces the structure and terminology of RSNs. Section 4 describes our RSN modeling method. The application of this model to formal verification is discussed in Section 5. Section 6 presents the access pattern generation procedure, followed by experimental results in Section 7. A conclusion is given in Section 8.

2. RELATED WORK AND CONTRIBUTIONS

2.1. Verification

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. 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 [Fisher 2002]. The structure of the IEEE 1149.1 circuitry, including the TAP controller and TDR connectivity, can be verified by logic tracing [Melocco et al. 2003].

The functionality of the 1149.1 circuitry can be validated by simulation using automatically generated stimuli [Bruce Jr et al. 1996]. Similarly, the functionality of IEEE 1500 wrappers can be validated by coverage-driven, constrained-random simulation [Diamantidis et al. 2005]. The stimuli are chosen in such a way as to maximize coverage of behavioral rules [Benso et al. 2008]. 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 exists, such that the scan network functions as a shift register [Eichelberger and Williams 1977]. 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 [Kamepalli et al. 2006]. The functionality of the 1149.1 circuitry can be verified by symbolic simulation [Bryant 1990; Singh et al. 1997] or four-valued logic simulation using preconditioning and checking sequences [Dahbura et al. 1989; Melocco et al. 2003].

2.2. Access Pattern Generation

In reconfigurable scan networks, access to a scan register may be realized in many ways, using different scan-in sequences. Possible solutions greatly differ in the number of bits that need to be shifted in. The goal of optimal access pattern generation is to find the shortest scan-in sequence that implements the access to a set of target scan segments.
Recently, algorithms for optimal construction of scan hierarchies compliant with IEEE Std 1149.1-2013 and IEEE P1687 were proposed in [Ghani Zadegan et al. 2011], and methods for access time analysis were developed in [Ghani Zadegan et al. 2012; Larsson and Ghani Zadegan 2012]. These and similar contributions are based on scan networks with reconfigurability limited to Segment Insertion Bits (SIB). As shown in Figure 2a, a SIB is a simple network component that either bypasses or connects a lower-level scan segment (or a scan network) to the higher-level scan chain, depending on the state of the one-bit configuration register (CR). In IEEE Std 1149.1-2013, this bypassing concept is realized with so called segment selectors and excludable segments. In such architectures, optimal access pattern generation is a straightforward task: The scan-in sequence required to access any scan segment is easily found by examining the current state of all SIBs. All SIBs that enclose the target scan segment must be opened, and all remaining SIBs must be closed to optimize the access time.

2.3. Motivation

While the existing verification techniques efficiently handle simple scan chains and the IEEE 1149.1 circuitry, the verification of arbitrary reconfigurable scan networks poses a much more difficult problem. The number of access modes that need to be verified may be exponential in the size of the RSN: with \( n \) register bits defining the configuration, the number of distinct access modes may reach \( 2^n \). In core-based design, scan networks may be composed of third-party modules (IP cores), 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.

An example of a design bug caused by erroneous integration of network components is explained using Figure 1: Assume that the OR gate is by mistake replaced with an AND gate. Clearly, in this case there does not exist any assignment to bits \( a \) and \( b \) such that Segment 2 belongs to the scan path—there is a combinational dependency that cannot be satisfied. Such problems may arise in P1687 architectures when legacy or third-party components are integrated into a system-level RSN. To prevent such design bugs, architectural design rules can be established, but this may lead to overly restrictive architectures and reduce design flexibility. To detect the design bug caused by replacing the OR gate in Figure 1, a combinational Automatic Test Pattern Generator (ATPG) 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 to control signals that puts the target segment on the active 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 pattern generation for RSNs are hard decision problems. Deep state space exploration may be required to verify design correctness or find an access sequence to embedded instruments. While state-of-the-art ATPG and model checking algorithms can handle sequential depths of several dozens of clock cycles, access to a reconfigurable scan network may require justification over hundreds of thousands cycles. This is for instance the case in hierarchical RSNs with wide scan registers: to access a scan register at the lowest hierarchy level, scan data may need to be flushed through the RSN many times setting registers at higher hierarchy levels. The sequential nature of scan registers is the main reason for the high sequential depth of cycle-accurate RSN models.

2.4. Contributions

This article presents an abstraction-based RSN modeling method that is applicable to complex reconfigurable scan architectures, including arbitrary P1687 structures. Compared with conventional cycle-accurate models, the abstract model has a significantly reduced sequential depth. It enables efficient formal verification and pattern generation for most complex RSNs with distributed configuration and control signals driven by arbitrary combinational logic. The presented modeling is a generalization of our recent modeling techniques from [Baranowski et al. 2012] and [Baranowski et al. 2013], extended with handling of unknown values.

Our unified model is applicable to both formal verification and optimal access pattern generation. As an example, we describe the use of the model in bounded model checking to formally prove certain properties of the RSN, such as observability and controllability of scan registers. The abstract model is also used to generate access patterns with reduced access time. We present a mapping of access pattern generation to a pseudo-Boolean optimization problem and propose a fast, parallelized pattern generation procedure. The optimization method can handle large RSNs and is more general than algorithms that are developed for only specific reconfigurable architectures, such as [Ghani Zadegan et al. 2012] for the SIB architecture. Our experimental results show that such an optimization method is crucial for complex RSNs, reducing the access time by up to a factor 88.

3. RSN STRUCTURE AND TERMINOLOGY

Reconfigurable scan networks can be viewed as scan registers with configurable scan path and variable length. In this paper, we follow a general definition of RSNs at Register-Transfer Level (RTL) and abstract from gate-level implementation details. The modeling method presented in the next section allows arbitrary architectures that follow the rules described below. The correspondence of our RSN definition to existing standards is discussed at the end of this section.

Figure 3 presents a simple RSN example and explains the basic terminology. The one-bit scan segments S1 and S3 control the access to two multi-bit segments S2 and S4, respectively. The scan-in data are shifted through segments S2 and S4 only if the previous access assured that S1 = S3 = 1.

RSNs are sequential circuits composed of registers or latches, multiplexers, and combinational logic. An RSN has a global clock and reset 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 Std 1149.1 [1149.1 2013]. If the RSN is accessed through a 1149.1-compliant Test Access Port (TAP), the entire RSN is connected to the 1149.1 circuitry as a Test Data Register (TDR) and the global control inputs are driven by the TAP controller. Optionally, an RSN may have primary data input and output ports for communication with on-chip instruments, as well as primary control input ports for network configuration.
3.1. 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 primarily to communicate with on-chip instrumentation (cf. $S_2$ and $S_4$ in Figure 3). A scan segment is essentially a shift register composed of one or more scan cells sharing a set of control signals. A scan segment is optionally equipped with a shadow register that can be loaded in parallel from the shift register and set to an initial state with the global reset. This structure may be used to communicate with instruments bidirectionally or to drive internal control signals (cf. $S_1$ and $S_3$ in Figure 3). Figure 4a 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:

- During a capture operation, the shift register is loaded with data from the data-in port (e.g. from an attached instrument) or remains stable if no such port exists. The shadow register (if it exists) is stable during the capture operation.
- 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. The shadow register (if it exists) is stable during the shift operation.
During an update operation, the shadow register (if it exists) is loaded with data from the shift register. These data are then available at the data-out port (e.g. to an attached instrument or internal control signal). If the shadow register does not exist, the update operation has no effect on the segment. After the update operation, the state of the shift register is assumed unknown (can be any value).

When none of the scan operations nor a reset is performed, both shift registers and shadow registers remain stable. To inhibit the scan operations, a scan segment may possess up to three optional control ports:

- Select port (select) specifies if the capture, shift, and update operation is performed on the segment.
- 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. Only scan segments that contain a shadow register may include this port.

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.

Scan segments are chained via scan-out and scan-in ports, either directly, via buffers or inverters, or through scan multiplexers of any width. A scan multiplexer controls the path through which data are shifted in an RSN. For instance, the two two-input scan multiplexers in Figure 3 allow to bypass scan segments S2 and S4. The control signal of a scan multiplexer is called address and specifies the selected scan input.

A non-circular sequence of chained scan segments is referred to as scan chain. A scan path is scan chain starting at a primary scan-input and ending at a primary scan-output. We allow arbitrary scan architectures composed of any combination of scan segments and scan multiplexers except for circular architectures.

### 3.2. Control and Data Signals

The control ports of scan segments (select, capdis, updis) and multiplexers (address) are driven by signals that are collectively referred to as internal control signals. Internal control signals may only change in consequence of the update operation, and must have a defined reset state. They can be driven by arbitrary combinational logic blocks that take their inputs from shadow registers of scan segments distributed over the RSN, as well as from primary control inputs. (The shadow register is mandatory for scan segments driving internal control signals, cf. Figure 4). For instance, the select port of scan segment S2 in Figure 3 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 (shadow registers themselves are not explicitly shown in the figure).

The data-in ports of scan segments may be driven by arbitrary combinational logic blocks driven by shadow registers and primary data inputs. The data-out ports of scan segments may drive the primary data outputs of an RSN, either directly or through arbitrary combinational logic. The primary data inputs and outputs are used for bidirectional communication with instruments.

### 3.3. 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 3, if $S1 = 1$ and $S3 = 0$, 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. (In fact, the state of shadow registers and primary control inputs alone is sufficient to determine how data are shifted through the RSN.) 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 toward the primary scan-output, and the shadow registers of all scan segments that do not take part in the access (i.e., do not belong to the active scan path) are stable. For instance, the control signals driving the select ports of $S2$ and $S4$ in Figure 3 ensure that these segments are selected if and only if they are chosen by the multiplexers and hence form the active scan path. This guarantees that if $S2$ or $S4$ does not belong to the active scan path, the data held by its shadow register are not lost during the update operation.

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 4b. 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. These data are shifted out of the network during the shift phase at each 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 optional shadow registers of scan segments on the active scan path (if no shadow register is present in a scan segment, the update operation has no effect on the segment). 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 3). 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.

Reconfigurable scan networks, as defined above, constitute a superset of IEEE 1149.1 scan architectures: We allow the multiplexed scan network composed of an IR, bypass and boundary scan registers, and multiple TDRs. In addition to excludable and selectable TDR segments defined in IEEE Std 1149.1-2013 [1149.1 2013], we allow arbitrary signals generated internally in the scan network to control the capture, shift, and update operations of individual scan segments. Our definition of RSNs is also a superset of structures defined in a recent revision of IEEE P1687, as we do not pose any structural constraints on the composition of control signals: we allow for arbitrary control of scan segments generated by combinational blocks that take their input from any scan registers distributed over the network. For the sake of brevity, the IEEE P1687 data registers are not explicitly represented in this work, but can be handled in a straightforward way as non-scannable registers. This extension is given in [Baranowski 2014]. Moreover, the presented RSN definition includes serial scan architectures composed of IEEE 1500-based wrappers. The restriction to serial architectures results from the fact that the RSN is assumed to have a single primary scan input and output, and a single active scan path. In principle, both the definition and the modeling method can be extended to support multiple primary scan inputs and outputs, as well as multiple (parallel) active scan paths. However, this would require...
a different approach to access pattern generation, which is beyond the scope of this article.

4. CSU-ACCURATE RSN MODEL (CAM)

In the following, we describe a formal way of modeling RSNs by temporal abstraction. The model can be easily derived from any structural description of an RSN: either a gate- or RT-level netlist, or high-level representations, e.g. in Instrument Connectivity Language (ICL) defined by IEEE P1687.

The state of scan segments and control signals is modeled in three-valued logic with three symbols \( \{0, 1, X\} \) to represent logic value 0, logic value 1, and an unknown (X) value, respectively. The unknown value is used to model partially specified initial scan configurations (the state of uninitialized registers), and the high-impedance (unknown) state of tri-state logic gates. The interpretation of logic operators over three-valued variables follows Kleene’s strongest regular three-valued logic [Kleene 1950].

**Definition 4.1.** The Capture-shift-update-Accurate Model (CAM) of an RSN is a tuple \( M = \{S, I, C, \text{Active}\} \) that consists of a set of state elements \( S \), a set of external control inputs \( I \), a set of scan configurations \( C \subseteq \{0, 1, X\}^{S \cup I} \), and a predicate Active. Each state element \( s \in S \) corresponds uniquely to a one-bit scan register in the network. Each scan configuration \( c \in C \) defines the state of all scan elements in \( S \) and all external control inputs in \( I \). The predicate Active: \( C \times S \rightarrow \{0, 1, X\} \) assigns each state element \( s \in S \) in scan configuration \( c \in C \) a value denoted by Active\((c, s)\).

Each scan configuration \( c \in C \) is also treated as a function \( c : S \cup I \rightarrow \{0, 1, X\} \) that maps each element \( e \in S \cup I \) to its state denoted as \( c(e) \). If \( s \in S \) is part of a scan segment that contains a shadow register, \( c(s) \) corresponds to the state of the shadow register. Otherwise, if \( s \) is part of a segment with no shadow register, \( c(s) \) corresponds to the state of the shift register upon the last update or reset operation.

By Select\((c, s)\), Updis\((c, s)\) and Capdis\((c, s)\) we denote the state of the select, updis and capdis control signals of scan segment \( s \in S \) in scan configuration \( c \in C \), respectively. Predicate Active\((c, s)\) is defined as follows:

\[
\text{Active}(c, s) := \begin{cases} 
0 & \text{if Select}(c, s) = 0, \\
1 & \text{if Select}(c, s) = 1 \text{ and } c \text{ is valid}, \\
X & \text{otherwise.}
\end{cases}
\]

Thus, an element \( s \) belongs to the active scan path exactly when Active\((c, s)\) = 1.

Please note that the CAM is defined over one-bit scan registers instead of scan segments only for the sake of simplicity. In a practical implementation, the CAM can be optimized by grouping scan registers with same control signals together and deriving only one set of control signals and one Active predicate per group.

In the following, we describe the construction of predicates (Section 4.1) and we define the CSU-accurate transition relation of the CAM (Section 4.2).

4.1. Valid Scan Configuration

To construct the predicates, we need to distinguish valid scan configurations from invalid ones (cf. Section 3). To this end, we construct a predicate \( V : C \rightarrow \{0, 1\} \) that evaluates to 1 if and only if the scan configuration is valid, i.e. when there exists a well formed scan path and all off-path scan segments are deselected. We construct the predicate \( V \) piecewise as a conjunction of the form:

\[
V(c) := \bigwedge_{s \in S} v(c, s),
\]
where $v(c, s)$ is a predicate that evaluates to true exactly when the local scan configuration of the scan segment $s$ is valid in $c \in C$, as explained below.

For a scan segment $s$ with a single predecessor $p \in \text{pred}(s)$ and a single successor $n \in \text{succ}(s)$ (cf. Figure 5a), it is required that both $p$ and $n$ be selected if $s$ is selected, such that scan data are not lost. Thus:

$$v(c, s) := (\text{Select}(c, s) = 1) \Rightarrow [(\text{Select}(c, p) = 1) \land (\text{Select}(c, n) = 1)].$$

(3)

For a scan segment $s$ with a single predecessor $p$ and multiple successors (cf. Figure 5b), a valid scan configuration requires that exactly one successor of $s$ is selected if $s$ is selected. Formula (4) specifies that at least one successor of $s$ is selected if $s$ is selected:

$$(\text{Select}(c, s) = 1) \Rightarrow \bigvee_{n \in \text{succ}(s)} (\text{Select}(c, n) = 1).$$

(4)

Formula (5) ensures that at most one successor of $s$ can be selected at any time:

$$\forall_{n_k, n_l \in \text{succ}(s), n_k \neq n_l} : [(\text{Select}(c, n_k) = 1) \Rightarrow (\text{Select}(c, n_l) \neq 1)].$$

(5)

The following formula $v(c, s)$ states the requirement for a valid scan configuration for a segment $s$ with one predecessor $p$ and multiple successors using formulas (4) and (5):

$$v(c, s) := [(\text{Select}(c, s) = 1) \Rightarrow (\text{Select}(c, p) = 1)] \land (4) \land (5).$$

(6)

This assures that in case of a branching scan path (fanout $> 1$) only one branch is active, i.e. there are no multiple selected successors.

For a scan segment $s$ with a single successor $n$ and multiple predecessors selected by a multiplexer (cf. Figure 5c), a valid scan configuration requires that exactly one predecessor of $s$ is selected if $s$ is selected. Formula (7) below states that at least one predecessor must be selected if $s$ is selected:

$$(\text{Select}(c, s) = 1) \Rightarrow \bigvee_{p \in \text{pred}(s)} (\text{Select}(c, p) = 1).$$

(7)

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

$$\forall_{p \in \text{pred}(s)} : [(\text{Select}(c, p) = 1) \Rightarrow (\text{Address}(c, s) = \text{addr}(p))],$$

(8)

where $\text{addr}(p)$ is the multiplexer address for $p$ (i.e. a constant, cf. Figure 5c).

The requirement for a valid scan configuration for segment $s$ with one successor $n$ and multiple predecessors is captured by the following formula:

$$v(c, s) := [(\text{Select}(c, s) = 1) \Rightarrow (\text{Select}(c, n) = 1)] \land (7) \land (8).$$

(9)
This assures that in case of a multiplexed scan path the active path is correctly routed. In case of a node with multiple predecessors and multiple successors, the following formula captures the condition for a valid scan configuration:

\[ v(c, s) := (4) \land (5) \land (7) \land (8). \]

Since \( V(c) = \bigwedge_{s \in S} v(c, s) \) holds exactly when \( c \) is valid, predicate \( \text{Active} \) can be derived as:

\[
\text{Active}(c, s) := \begin{cases} 
0 & \text{if } \text{Select}(c, s) = 0, \\
1 & \text{if } (\text{Select}(c, s) = 1) \land V(c), \\
X & \text{otherwise.}
\end{cases}
\] (10)

The \( \text{Select} \) predicates are obtained by traversing the input cones of the corresponding \( \text{select} \) signals in the RSN netlist.

The presented modeling requires that the active scan path includes only scan segments, multiplexers, buffers and inverters. This requirement is also posed by IEEE P1687 to prevent that scan data are altered while shifting. Arbitrary logic on scan paths—e.g. test compression structures—are handled in our modeling by black-boxing: We define that a scan configuration is invalid if such structures belong to the active scan path. To this end, for each scan segment \( s_b \) of a black-boxed component, the predicate \( V(c) \) is extended with the constraint \( \text{Select}(c, s_b) = 0 \). In access pattern generation (Section 6), this assures that scan data are never shifted through the black-boxed components. In formal verification (Section 5), such constraints constitute an assumption that the black-boxed components never belong to the active scan path.

A similar technique can be applied to handle RSN modules with unknown structure or functionality (e.g. protected third-party IP): If such a module behaves as a fixed-length scan chain, it can be represented in our modeling as a simple scan segment with the specified length. If the length of the module cannot be determined, it must never belong to the active scan path. This is achieved with additional constraints in predicate \( V \), as explained above.

### 4.2. Transition Relation of the CSU-Accurate Model (CAM)

The CAM transition relation models the effect of a CSU operation which we consider atomic. A CSU operation 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. We describe this behavior with a transition relation, as defined below.

**Definition 4.2.** The transition relation of a CAM \( M = \{S, I, C, \text{Active}\} \) is defined as a set \( T \subseteq C \times C \) with the following characteristic function:

\[
T(c_1, c_2) := \bigwedge_{s \in S} [\text{Stable}(c_1, s) \Rightarrow (c_2(s) = c_1(s))] \land [\text{Unknown}(c_1, s) \Rightarrow (c_2(s) = X)],
\] (11)

where \( c_1, c_2 \in C \) while \( \text{Stable}() \) and \( \text{Unknown}() \) are Boolean functions defined as follows:

\[
\text{Stable}(c, s) := (\text{Active}(c, s) = 0) \lor (\text{Updis}(c, s) = 1),
\] (12)

\[
\text{Unknown}(c, s) := [(\text{Active}(c, s) = X) \land (\text{Updis}(c, s) = 1)] \lor [(\text{Active}(c, s) = 1) \land (\text{Updis}(c, s) = X)].
\] (13)

The transition relation \( T \) includes all pairs of scan configurations \((c_1, c_2)\), such that \( c_2 \) can be reached from \( c_1 \) within one CSU operation.

The characteristic function of the transition relation defines the requirement for state changes: If an element \( s \) does not belong to the active scan path or its \( \text{updis} \) signal is active in scan configuration \( c_1 \), the state 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 $s$ is assumed unknown in the consecutive scan configuration $c_2$. As a consequence, the state of $s$ may change freely only when $s$ is selected in a valid scan configuration $c_1$, i.e. when $\text{Active}(c_1, s) = 1$ and $\text{Updis}(c_1, s) = 0$.

5. FORMAL VERIFICATION

With appropriate design rules, the verification of certain properties of simple reconfigurable scan networks may not be required. For instance, the SIB-based architecture proposed in [Ghani Zadegan et al. 2011] consist of hierarchically connected SIBs and scan segments, and no combinational logic is allowed for control signals (cf. Figure 2). In this RSN architecture, the accessibility of a scan chain (e.g. a scan segment or a chain of scan segments and SIBs) requires that the following recursive rule is fulfilled: (1) the parent SIB of the scan chain (SIB to which the chain is connected to) works correctly, (2) the parent SIB is properly connected to the higher level chain, (3) the higher level chain is also accessible. While the first condition is easily checked by exhaustive simulation of the SIB, the second condition can be enforced with a structural design rule.

In contrast, the verification of arbitrary RSN architectures with control signals driven by combinational logic poses a much more difficult problem. Due to the high sequential depth, existing formal verification algorithms are ineffective in proving RSN properties such as accessibility, as shown in Section 7.6. The CSU-Accurate Model (CAM) is used to improve the scalability of existing model checking methods and enable formal verification of complex RSNs. In principle, the CAM can be used within any formal verification technique that models the circuit with a transition relation, e.g. in a symbolic or SAT-based model checker.

In the following, we study the implications of CSU-accurate modeling in formal verification and discuss its limitations. As an example, we show an application of the CAM to prove accessibility properties using Bounded Model Checking (BMC). Finally, we define a class of robust RSNs, explain their advantages, and show how to prove the robustness property.

5.1. Implications of CSU-Accurate Modeling

Intuitively, the CAM can be viewed as an abstract FSM that exactly models the RSN state but abstracts its temporal behavior. One state transition (clock cycle) in the abstract FSM corresponds to a full CSU operation, i.e. multiple clock cycles in the cycle-accurate RSN model. An example is given in Figure 6, where $k$ state transitions during one CSU operation (a) are combined into a single transition in the CAM (b).

CSU-accurate modeling is sound (for a proof please refer to [Baranowski 2014]): A property that holds in the CAM is guaranteed to hold in the cycle-accurate model, under the assumption that all internal control signals (e.g. the address of a scan multiplexer) are stable during the capture and shift phases of a CSU operation. This assumption holds trivially for all control signals generated internally to the RSN, as they
are driven by shadow registers of scan segments, and has to be ensured for external control inputs (if any). If an external control input is unstable during the capture or shift phase, the active scan path may change during the shift operation and scan data may be lost. Therefore, the stability of external control inputs during the capture and shift phases must be proven in the system model. The existence of shadow registers in scan segments driving internal control signals is assured by structural analysis of the netlist or enforced with design rules, as in IEEE P1687.

CSU-accurate modeling is pessimistic: According to the CAM transition relation (cf. formula (11)), the content of all scan segments is assumed undefined in an invalid scan configuration, although it may be well defined in the cycle-accurate model. Thus, the CAM may produce spurious counterexamples to a property, even if the property holds in the cycle-accurate model. However, for scan networks in which invalid scan configurations are not reachable, a property that is disproved in the CAM is guaranteed to be false in the cycle-accurate model. In Section 5.4, we show how to prove that only valid scan configurations are reachable, in which case the CSU-accurate abstraction is complete, i.e., causes no spurious counterexamples.

5.2. 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 a finite-state automaton $F$ for all bounded executions of $F$ (sequences of consecutive states in $F$ of bounded length) rooted in one of the initial states of $F$ [Biere et al. 2003]. The method is very efficient in detecting design bugs and significantly outperforms BDD-based symbolic model checkers. Extensions of BMC to unbounded LTL model checking include, for instance, techniques based on state space interpolation [McMillan 2003] and induction [Sheeran et al. 2000].

Given a temporal logic property $P$ of a finite-state automaton $F$, bounded model checking consists in searching for an execution sequence (counterexample) that refutes $P$ within a certain number of transitions (steps). The property is disproved (counterexample is found) if, for a certain number of steps $n \in \mathbb{N}^+$, the following Boolean formula is satisfiable:

$$\varphi^n := I \land \bigwedge_{i=0}^{n-1} T_i \land \neg P^n,$$

where $I$ is the characteristic function of the initial states of $F$, $T_i$ is the characteristic function of the transition relation of $F$ in $i$-th step, and $P^n$ is a propositional representation of the temporal property $P$ for $n$ steps. The formula is typically transformed to conjunctive normal form (CNF) and its satisfiability is checked with a SAT solver.

In the following, we show the application of BMC for proving accessibility of the RSN using the CAM. For the details on translating general LTL formulas to bounded propositional formulas please refer to [Biere et al. 2003].

5.3. Accessibility Proof

To assure that a scan segment can be both read from and written to, it is necessary to prove that it is observable and controllable. A necessary requirement is that there exists a scan path from a primary scan input, through the segment, down to a primary scan output. To determine if a structural connection exists, a static connectivity check can be used [Remmers et al. 2004]. For complex scan architectures with arbitrary control signals, the necessary and sufficient requirement is the justification of control signals over one or multiple CSU operations.
We define that a scan segment is accessible in an initial scan configuration (or a set of initial scan configurations) if and only if there exists a scan-in sequence that puts the scan segment on the active scan path while the corresponding update and capture disable signals are inactive. Given the CAM of an RSN $M = \{S, I, C, \text{Active}\}$, proving the accessibility of a scan segment $s \in S$ is equivalent to disproving the following LTL formula in the CAM:

$$G \neg [(\text{Active}(s) = 1) \land (\text{Updis}(s) = 0) \land (\text{Capdis}(s) = 0)].$$

The LTL formula states that for all 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 through the corresponding updis and capdis signals. Note that our definition of accessibility refers to a certain (possibly partially specified) initial scan configuration. It is not guaranteed that a scan segment which is accessible in the initial scan configuration is accessible in all reachable scan configurations.

We disprove this property and hence prove accessibility of the scan segments with the bounded model checking approach. To this end, the LTL property is translated into a bounded propositional formula over $n$ CSU operations. The proof of accessibility within $n$ CSU operations reduces to checking the satisfiability of the following Boolean formula:

$$\text{Check}(s, c_0, n) := 1(c_0) \land \bigwedge_{i=1 \ldots n} T(c_{i-1}, c_i) \land \bigvee_{i=0 \ldots n} [(\text{Active}(s) = 1) \land (\text{Updis}(s) = 0) \land (\text{Capdis}(s) = 0)],$$

where $1(c_0)$ is the characteristic function of an initial scan configuration $c_0 \in C$ and for $0 < i \leq n$, $c_i$ represents the scan configuration in the $i$-th time step (after the $i$-th CSU operation). The formula $\text{Check}(s, c_0, n)$ is satisfiable if and only if the scan segment $s$ is accessible within $n$ CSU operations.

The accessibility proof is an iterative procedure that checks the satisfiability of formula (16) for an increasing number of CSU operation ($n = 1, 2, \ldots$) until the formula is satisfiable, or until a predefined bound for the allowed number of CSU operations is reached. To improve SAT solving performance, incremental solving techniques are employed: the Boolean formula (SAT instance) generated for $n$ CSU operations is extended with additional clauses for the characteristic function of the transition relation and reused in iteration $n + 1$.

5.4. Verification of Robustness

We define that a reconfigurable scan network is robust if all scan configurations that are reachable from the initial configuration are valid, i.e., the selected scan segments always form an active scan path regardless of the input data sequence. More formally, an RSN is robust if and only if the LTL property $G V$ holds in the CAM, i.e., the Boolean validity predicate $V$ (cf. Section 4.1) is always true.

The main advantage of robust RSNs compared with non-robust networks lies in a lower verification effort: as shown in Section 7.4, many design bugs affect the robustness property and hence can be efficiently detected just by checking robustness. Furthermore, for robust RSNs, the CSU-accurate model is complete: a property that holds in the RSN will also hold in the CAM, i.e., the CAM abstraction cannot cause spurious counterexamples (cf. Section 5.1). The reason is that the CAM fully captures the effects that a CSU operation causes in the cycle-accurate model (for a proof, please refer to [Baranowski 2014]). Moreover, as all reachable scan configurations are valid, the CAM can be simplified by removing the validity predicate $V$ from the predicates,
simply defining that $\text{Active}(c,s) := \text{Select}(c,s)$ for all $c \in C$ and $s \in S$. This significantly simplifies the CAM and makes formal verification and pattern generation even more efficient.

The robustness property $GV$ can be proven in the CAM using any unbounded LTL model-checking method. In the following, as an example, we show the application of a SAT-based inductive technique [Sheeran et al. 2000] to prove this property. Although the inductive technique is incomplete as discussed below, it is very efficient. For a more detailed discussion of robustness and for verification techniques for robust RSNs please refer to [Baranowski 2014].

The robustness property can be proven on a CAM $M = \{S, I, C, \text{Active}\}$ with transition relation $T$ by showing that:

1. $V$ holds in the initial scan configuration $c_0 \in C$ of $M$, i.e., $V(c_0)$ is true, and
2. $V$ is an invariant of the transition relation $T$, i.e. $\forall_{(c_1,c_2) \in T} [V(c_1) \Rightarrow V(c_2)]$.

If both conditions hold, the network is robust, as the initial scan configuration is valid, and the validity is preserved by any CSU operation.

The two conditions for robustness can be formulated as a satisfiability problem and solved with a SAT solver. The first condition holds if all initial states are valid, which is exactly when the following Boolean formula is unsatisfiable:

$$1(c_0) \land \neg V(c_0), \quad (17)$$

where $1(c_0)$ is the characteristic function of the initial configuration $c_0 \in C$. The validity is an invariant of the transition relation $T$ (second condition holds) if and only if the following formula is unsatisfiable:

$$V(c_i) \land T(c_i,c_{i+1}) \land \neg V(c_{i+1}). \quad (18)$$

If both formula (17) and (18) are unsatisfiable, the RSN is robust. Otherwise, the satisfying assignment from the SAT solver provides a counterexample with a transition from a valid scan configuration into an invalid one.

Note that the first requirement of robustness is a necessary condition, while the second is not, i.e., the RSN may be robust even though $V$ is not an invariant of the transition relation. This is the case if all valid scan configurations that lead to invalid configurations are not reachable from the initial scan configuration. As a consequence, this method proves a stronger requirement and may pessimistically classify a robust network as non-robust. The advantage of this method lies in its efficiency, as it reduces to a Boolean satisfiability problem instead of unbounded LTL model checking. As our experimental results show (Section 7.3), this technique can rapidly prove robustness even for large RSN architectures and hence can be used as a quick preprocessing step before using a full-featured model checker.

6. ACCESS PATTERN GENERATION

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

In the following, we formulate the problem of computing minimal (shortest) access patterns. As the search for the global minimum may be prohibitively expensive in large RSNs, we propose an affordable pattern generation procedure which supports arbitrary RSN architectures that are modeled with the CAM. The proposed method is applicable to access merging, i.e. generation of efficient scan-in sequences that access multiple scan elements during one or multiple CSU operations.
6.1. Problem Formulation

We search for the sequence of bits that must be shifted into the RSN during one or multiple CSU operations to reach a certain target scan configuration with minimal access time. We specify a scan access by its initial scan configuration \( c_0 \in C \) and target scan configuration \( c_t \in C \). We denote the access by \( (c_0, c_t) \).

Given is the CAM of an RSN \( M = \{S, I, C, Active\} \) with transition relation \( T \), and a scan access \( (c_0, c_t) \). Access pattern generation is the computation of a sequence of \( n \in \mathbb{N}^+ \) consecutive scan configurations \( c_1, c_2, \ldots, c_n \) such that the following conditions hold:

\[
(c_n = c_t) \land \forall i=1...n \ ( (c_{i-1}, c_i) \in T )
\]

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

\[
\text{Cycles}(c_0, \ldots, c_n) := D \cdot n + \sum_{i=0}^{n-1} \sum_{s \in S} (\text{Active}(c_i, s) = 1),
\]

where \( D \) is a constant that amounts to the number of cycles required to perform the capture and update operation. Note that \( n \) is the number of CSU operations required for the optimal solution, which is a priori unknown.

Condition (19) is satisfied exactly when \( c_0, c_1, \ldots, c_n \) is a valid sequence of consecutive scan configurations, such that the last configuration equals the target scan configuration. Access time \( \text{Cycles}(c_0, c_1, \ldots, c_n = c_t) \) given by formula (20) amounts to the time required to perform capture and update cycles \((D \cdot n)\) plus the number of required shift cycles in each scan configuration, except for the target scan configuration \( c_t \). Constant \( D \) amounts to at least 4 cycles if the RSN is accessed through a 1149.1 TAP due to the overhead of the TAP controller, or more if pause cycles are required. The number of required shift cycles, i.e. the scan-in sequence length, equals the number of predicates that evaluate to 1, since each predicate corresponds to a one-bit scan register, and the predicate is true if the corresponding scan register is part of the active scan path.

The search for the access sequence with the globally minimal access time is a hard problem: The global minimum is not necessarily found for the minimal number \( n_{\text{min}} \) of CSU operations required to perform the access, i.e., to satisfy formula (19). Often, the access time can be reduced by allowing additional CSU operations (see Figure 7).

Note that a CSU operation always incurs an access overhead of \( D \) cycles, and hence an access pattern with \( n \) CSU operations takes at least \( n \cdot D \) cycles (this time is depicted by the line “CSU overhead” in Figure 7). Therefore, an access pattern with the globally minimal access time can be found with an iterative procedure: Compute shortest access patterns with 1, 2, 3 ... CSU operations. Let \( \text{Cycles}_n \) denote the access time of the pattern with \( n \) CSU operations, and set \( n_{\text{bound}} := \lceil \text{Cycles}_n / D \rceil \). The access time of the pattern with \( n \) CSUs is the global minimum if there does not exist any other solution with up to \( n_{\text{bound}} \) CSUs with lower access time.
In practice, due to limited computational resources, the search for all solutions with up to \( n_{\text{bound}} \) CSU operations is often impractical. In contrast, the search for the first local minimum (cf. Figure 7) is more tractable. Moreover, our experiments show that the search beyond the first local minimum seldom leads to further access time reduction.

The following section explains merging of concurrent read and write accesses to multiple scan segments. Section 6.3 describes how we generate an access sequence with the minimal access time for a given (fixed) number of CSU operations. In Section 6.4 we present an affordable pattern generation procedure.

6.2. Access Merging

The challenge of access merging is to find the optimal order of multiple accesses to scan segments that results in a minimal scan-in sequence. The target scan segments must have their target values in the final scan configuration \( c_t \), but the order in which the merged accesses are performed is not specified. It is therefore sufficient to specify the concurrent access to multiple scan segments by its initial and target scan configurations \((c_0, c_t)\).

Specifying read accesses in this way restricts them to the last CSU operation. To improve merging flexibility, a read access may be specified by ensuring that during the concurrent access to multiple scan segments by its initial and target scan configurations \((c_0, c_t)\).

Condition (19) is extended with such a disjunction for each read access.

6.3. Mapping to Pseudo-Boolean Optimization

A pseudo-Boolean optimization problem is to find an assignment to the Boolean variables \((x_1, x_2, \ldots, x_k)\) that results in a scan segment that implements a valid scan access, such that the following Boolean formula is satisfied:

\[
\text{Access}(c_0, c_t, n) := 1(c_0) \land \bigwedge_{i=1\ldots n} T(c_{i-1}, c_i) \land \bigwedge_{s \in S} (c_n(s) = c_t(s)),
\]

where \(1(c_0)\) is the characteristic function of the initial configuration \(c_0\) and for \(0 < i \leq n\), \(c_i\) represents the scan configuration in the \(i\)-th time step (after the \(i\)-th CSU operation). This formula is transformed into a conjunctive normal form (CNF) or a set of clauses. If the formula is satisfiable, there exists a sequence of scan configurations \(c_0, c_1, \ldots, c_n\) that describes a valid scan access, such that \(c_n = c_t\). Otherwise, if the formula is unsatisfiable, no scan access with \(n\) CSU operations exists.

The formula \(\text{Access}(c_0, c_t, n)\), given by (22), is subject to pseudo-Boolean optimization with the cost function \(\text{Cycles}(c_0, c_1, \ldots, c_n)\), given by (20). The satisfying assignment (optimization solution) provides the state of all scan segments in scan configurations \(c_1 \ldots c_{n-1}\). The scan-in sequence that implements the scan access is derived from the satisfying assignment: The \(i\)-th CSU operation is fully specified by a pair of scan configurations \(c_{i-1}\) and \(c_i\). Configuration \(c_{i-1}\) specifies the active scan path. An element
\[s \in S \text{ belongs to the active scan path if } \text{Active}(c_{i-1}, s) = 1.\]

Configuration \(c_i\) specifies the content of scan segments and so provides the scan-in sequence for the \(i\)-th CSU operation. The resulting scan-in sequence is guaranteed to have the minimal access time among all solutions with \(n\) CSU operations.

### 6.4. Pattern Generation Procedure

Our pattern generation procedure is based on a heuristic that finds a local access time minimum (cf. Figure 7): we search for access sequences with increasing number of CSU operations as long as allowing more CSU operations provides a reduction of access time.

Let \(\text{Cycles}_n\) be the value of the cost function (20) 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 \(\text{Cycles}_{n+1}\). If the cost of the new solution is higher than the previous one, i.e. when \(\text{Cycles}_{n+1} > \text{Cycles}_n\), the pattern generation procedure terminates. Otherwise, the number of CSU operations is increased and the procedure is repeated until the user specified bound \(n_{\text{max}}\) is reached.

Let \(n_t\) be the number of CSU operations at which the pattern generation procedure terminates. The procedure guarantees that the final solution has the minimal access time among all solutions with \(n \leq n_t + 1\) CSU operations. There may exist a global minimum with lower access time that requires \(n_{\text{opt}} > n_t + 1\) CSU operations. However, experimental results show that the first local minimum is often the global minimum and increasing the number of CSU operations beyond \(n_t + 1\) rarely provides better results.

### 6.5. Implementation

The pattern generation procedure is implemented using the \textit{clasp} toolkit [Gebser et al. 2007], which includes a Boolean SAT solver and a pseudo-Boolean optimization engine. As the SAT solver is generally faster than the pseudo-Boolean optimizer, we use it initially to find the minimal number of CSU operations \(n_{\text{min}}\) that is required to implement the access. After \(n_{\text{min}}\) is found, pseudo-Boolean optimization is performed for increasing number of CSU operations, as described in Section 6.4.

Our framework exploits parallelism in the pattern generation procedure: After \(n_{\text{min}}\) is found, pseudo-Boolean optimization for \(n \geq n_{\text{min}}\) 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 8 illustrates the parallel execution of the pattern generation procedure.
7. EVALUATION

The proposed modeling approach is evaluated on several RSN benchmarks in two use cases: design verification and access pattern generation. To exploit the parallelism of the pattern generation procedure, the experiments are run on an Intel Xeon CPU with 12 cores operating at 3.33 GHz.

The results presented in the following sections are validated by cycle-accurate simulation in a commercial logic simulator. For this purpose, the RSN models are automatically translated to hardware Verilog models. The generated patterns are used as stimuli for the primary scan input of a network. During simulation, assertions verify that the scan access is performed correctly.

7.1. Benchmark Circuits

We evaluate our approach on two hierarchical RSN architectures: one implemented with multiplexers and the other implemented with Segment Insertion Bits (SIBs). The RSNs are synthesized for ITC’02 benchmarks, which are in widespread use for evaluation of test scheduling methods [Marinissen et al. 2002]. Benchmark hierarchies are reflected in the RSN architectures: Each module is assigned a dedicated RSN with one scan input and one scan output. The scan network of a module includes scan segments that represent the module’s boundary and internal scan chains, as well as the RSNs of constituent submodules.

The **MUX-based** architecture supports two access modes: configuration access and data access. Configuration access allows to reconfigure the scan chain by attaching or detaching internal scan segments or submodules. Figure 9 shows the MUX-based architecture for the top-level part of the p34392 benchmark. The scan chain of each module starts with a one-bit configuration register \( AM \) that sets the configuration mode \( (AM = 0) \), in which only the configuration registers \( (C) \) can be accessed, or data access mode \( (AM = 1) \). Once configured, this architecture allows faster access compared to the SIB-based scheme, as fewer control registers are present on the active scan path in the data access mode.

The **SIB-based** scan architecture implements hierarchical scan bypasses with SIBs. A SIB consists of a one-bit configuration register and a scan multiplexer that either bypasses or connects the lower-level scan segment (or a scan network) to the higher-level scan chain, depending on the content of the configuration register. The scan chain of a single module is composed of several SIBs, as in [Ghani Zadegan et al. 2011]. SIBs provide configurable access to the scan segments of the core, its submodules, as well as its inputs and outputs. Figure 10 shows such a scan architecture for the top-level part of the p34392 benchmark.

The initial scan configuration (after reset) is \( 0 \) for all control scan segments (e.g. \( C \) and \( AM \) in Figure 9) and unknown \( (X) \) for all data segments (e.g. **INPUTS** and **OUTPUTS** in Figure 9). If required, both MUX- and SIB-based architectures can be extended with **pipelining registers** at chosen scan multiplexers to prevent long combinational paths through multiple multiplexers, as in [Ghani Zadegan et al. 2012].

Table I presents the characteristics of our benchmark RSNs. For the MUX-based benchmarks, the number of multiplexers is given in the second column, the total number of scan segments (including configuration segments) in the third column, and the total number of scan flip-flops in the fourth column. The characteristics of the SIB-based benchmarks are listed in the last three columns of this table. In the MUX-based architecture, the number of scan segments is higher due to the additional segments \( AM \) that set the access mode for each module (cf. Figure 9).
Fig. 9. MUX-based scan architecture for the p34392 benchmark (note: select signals of scan segments are omitted for better readability)

Fig. 10. SIB-based scan architecture for the p34392 benchmark (note: select signals of scan segments are omitted for better readability)

Table I. Characteristics of the benchmark scan networks

<table>
<thead>
<tr>
<th>Design</th>
<th>MUX-based Architecture</th>
<th>SIB-based Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Num. Total Total</td>
<td>Num. Total Total</td>
</tr>
<tr>
<td></td>
<td>MUX #scan segm. bits</td>
<td>SIB #scan segm. bits</td>
</tr>
<tr>
<td>u226</td>
<td>59 99 1 475</td>
<td>50 90 1 466</td>
</tr>
<tr>
<td>d281</td>
<td>67 117 3 880</td>
<td>59 109 3 872</td>
</tr>
<tr>
<td>d695</td>
<td>178 335 8 407</td>
<td>168 325 8 397</td>
</tr>
<tr>
<td>h953</td>
<td>63 109 5 649</td>
<td>55 101 5 641</td>
</tr>
<tr>
<td>g1023</td>
<td>94 159 5 400</td>
<td>80 145 5 386</td>
</tr>
<tr>
<td>f2126</td>
<td>45 81 15 834</td>
<td>41 77 15 830</td>
</tr>
<tr>
<td>q12710</td>
<td>30 51 26 188</td>
<td>25 47 26 183</td>
</tr>
<tr>
<td>p22810</td>
<td>311 565 30 139</td>
<td>283 537 30 111</td>
</tr>
<tr>
<td>p34392</td>
<td>142 245 23 261</td>
<td>123 226 23 242</td>
</tr>
<tr>
<td>p53791</td>
<td>653 1241 98 637</td>
<td>621 1209 98 605</td>
</tr>
<tr>
<td>t512505</td>
<td>191 319 77 037</td>
<td>160 288 77 006</td>
</tr>
<tr>
<td>a586710</td>
<td>47 79 41 682</td>
<td>40 72 41 675</td>
</tr>
</tbody>
</table>

Table II. Accessibility verification effort for the MUX-based scan architecture

<table>
<thead>
<tr>
<th>Design</th>
<th>Access len. avg / max</th>
<th>Clauses max</th>
<th>Conflicts avg / max</th>
<th>( t_{\text{solve}}^{\text{max}} [s] )</th>
<th>( t_{\text{total}} [s] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>u226</td>
<td>3.5 / 5</td>
<td>20,617</td>
<td>1.6 / 8</td>
<td>0.02</td>
<td>0.9</td>
</tr>
<tr>
<td>d281</td>
<td>3.7 / 5</td>
<td>23,797</td>
<td>1.7 / 9</td>
<td>0.02</td>
<td>1.4</td>
</tr>
<tr>
<td>d695</td>
<td>3.9 / 5</td>
<td>65,189</td>
<td>1.9 / 10</td>
<td>0.06</td>
<td>11.8</td>
</tr>
<tr>
<td>h953</td>
<td>3.6 / 5</td>
<td>22,293</td>
<td>2.4 / 17</td>
<td>0.03</td>
<td>1.2</td>
</tr>
<tr>
<td>g1023</td>
<td>3.6 / 5</td>
<td>32,917</td>
<td>2.0 / 27</td>
<td>0.03</td>
<td>2.6</td>
</tr>
<tr>
<td>f2126</td>
<td>3.7 / 5</td>
<td>16,213</td>
<td>2.2 / 16</td>
<td>0.02</td>
<td>0.7</td>
</tr>
<tr>
<td>q12710</td>
<td>3.8 / 5</td>
<td>10,573</td>
<td>2.4 / 8</td>
<td>0.01</td>
<td>0.3</td>
</tr>
<tr>
<td>p22810</td>
<td>3.9 / 7</td>
<td>150,787</td>
<td>5.6 / 88</td>
<td>0.15</td>
<td>36.5</td>
</tr>
<tr>
<td>p34392</td>
<td>4.4 / 7</td>
<td>67,367</td>
<td>9.1 / 107</td>
<td>0.07</td>
<td>8.1</td>
</tr>
<tr>
<td>p93791</td>
<td>4.1 / 7</td>
<td>322,891</td>
<td>6.5 / 223</td>
<td>0.57</td>
<td>187.1</td>
</tr>
<tr>
<td>t512505</td>
<td>3.6 / 5</td>
<td>66,465</td>
<td>3.8 / 38</td>
<td>0.06</td>
<td>10.3</td>
</tr>
<tr>
<td>a586710</td>
<td>4.0 / 7</td>
<td>22,105</td>
<td>4.1 / 41</td>
<td>0.02</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Table III. Accessibility verification effort for the SIB-based scan architecture

<table>
<thead>
<tr>
<th>Design</th>
<th>Access len. avg / max</th>
<th>Clauses max</th>
<th>( t_{\text{solve}}^{\text{max}} [s] )</th>
<th>( t_{\text{total}} [s] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>u226</td>
<td>2.3 / 3</td>
<td>10,431</td>
<td>0.02</td>
<td>0.4</td>
</tr>
<tr>
<td>d281</td>
<td>2.4 / 3</td>
<td>12,482</td>
<td>0.01</td>
<td>0.7</td>
</tr>
<tr>
<td>d695</td>
<td>2.5 / 3</td>
<td>36,574</td>
<td>0.04</td>
<td>6.0</td>
</tr>
<tr>
<td>h953</td>
<td>2.3 / 3</td>
<td>11,594</td>
<td>0.01</td>
<td>0.6</td>
</tr>
<tr>
<td>g1023</td>
<td>2.3 / 3</td>
<td>16,826</td>
<td>0.02</td>
<td>1.2</td>
</tr>
<tr>
<td>f2126</td>
<td>2.4 / 3</td>
<td>8,698</td>
<td>0.01</td>
<td>0.2</td>
</tr>
<tr>
<td>q12710</td>
<td>2.4 / 3</td>
<td>5,368</td>
<td>0.01</td>
<td>0.1</td>
</tr>
<tr>
<td>p22810</td>
<td>2.5 / 4</td>
<td>77,376</td>
<td>0.05</td>
<td>17.2</td>
</tr>
<tr>
<td>p34392</td>
<td>2.7 / 4</td>
<td>33,028</td>
<td>0.03</td>
<td>3.5</td>
</tr>
<tr>
<td>p93791</td>
<td>2.5 / 4</td>
<td>172,082</td>
<td>0.14</td>
<td>94.5</td>
</tr>
<tr>
<td>t512505</td>
<td>2.3 / 3</td>
<td>33,685</td>
<td>0.03</td>
<td>4.8</td>
</tr>
<tr>
<td>a586710</td>
<td>2.5 / 4</td>
<td>10,521</td>
<td>0.01</td>
<td>0.3</td>
</tr>
</tbody>
</table>

7.2. Verification of Accessibility

The integrity of the benchmark scan architectures is verified using bounded model checking, as described in Section 5.3. We formally prove that all scan segments are observable and controllable.

Table II and III present the results of network verification for the MUX- and SIB-based architectures, respectively. Column “Access length” gives the average and maximal number of CSU operations to access a single scan segment. Under “Clauses” we give the maximum number of clauses contained in the SAT instance after the minimal number of CSU operations is found. Column \( t_{\text{solve}}^{\text{max}} \) is the maximum solve time for the check of a single scan segment (iteration), whereas \( t_{\text{total}} \) is the total design verification time.

Although the size of SAT instances grows up to about 323,000 clauses, the maximum solve time for a single iteration is just 0.6 s in the worst case, and about 150 ms on average for the largest RSN (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 most of the RSNs, the total validation time is below 10 s, and raises to about 3 minutes for the largest RSN.

The verification of SIB-based benchmarks does not require the solver to backtrack since a solution is found by direct implications. Contrary to the SIB-based design, the MUX-based architecture may cause temporal conflicts and backtracking if the solver takes a wrong decision on the access order to configuration registers. The average and
maximum number of times the solver needs to backtrack is given in Table II under “Conflicts”.

7.3. Verification of Robustness

Our benchmark circuits are proven robust using the technique presented in Section 5.4. The verification is successful for all considered benchmarks, i.e., we are able to formally prove that the scan configuration remains valid regardless of the applied input sequence. For the largest benchmark p93791, the proof for the MUX-based architecture takes up to 84 s, and up to 81 s for the SIB-based architecture.

As the benchmark circuits are robust, we can simplify their CAMs by removing the validity predicate \( V \) from Active predicates, i.e., we can define that \( \text{Active}(c,s) := \text{Select}(c,s) \) for all scan configurations \( c \in C \) and scan segments \( s \in S \) (cf. Section 5.4). This simplification leads to a slightly lower verification effort: compared with the results from the previous section, the verification of accessibility in the simplified CAM leads to about 8% reduction in the number of clauses, and 4% reduction of the SAT solver runtime on average. The effort of robustness verification is amortized when the simplified model is used for several consecutive verification or pattern generation jobs.

7.4. Debugging Faulty Designs

We apply the bounded model checking method from Section 5.3 to verify the accessibility of faulty RSNs. We mutate the benchmarks from Section 7.1 to model possible design bugs. Due to space limitations, we consider only four types of MUX-based RSNs (d281, g1023, p22810, and p93791) and three types of design bugs:

*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 scan inputs of a random scan multiplexer are swapped.

We assume that if a scan segment cannot be accessed within at most 30 CSU operations, the segment is inaccessible (i.e., the bound of model checking is set to 30 time steps). Note that this bound is significantly more than the worst case access length for fault-free MUX-based benchmarks, which is 7 (cf. Table II).

For each benchmark and each bug type, a hundred of random faulty RSNs is considered. Table IV presents the verification results: Column “Found” gives the ratio of faulty RSNs in which the bug is found, i.e., at least one scan segment is not accessible within 30 CSU operations. Column “Inaccessible” gives the average ratio of scan segments that are found inaccessible in a faulty RSN. Columns \( t_{\text{avg}} \) and \( t_{\text{max}} \) total give the average and maximum verification time, respectively.

The average verification effort for a faulty RSN is similar to the verification time of its fault-free counterpart (cf. Table II). The maximum verification time is an order of magnitude more than the average time, and is below 31 minutes 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 99% 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.

Since design bugs often result in the possibility to put the network into an invalid scan configuration, they are very likely to violate the robustness property. Indeed, using the approach from Section 5.4, we successfully prove that all the considered faulty RSNs are not robust and obtain counterexamples that can help localize the bugs. The
Table IV. Accessibility verification for faulty MUX-based RSNs

<table>
<thead>
<tr>
<th>Design</th>
<th>Bug type</th>
<th>Found [%]</th>
<th>Inaccessible [%]</th>
<th>$t_{avg}^{total}$ [s]</th>
<th>$t_{max}^{total}$ [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>d281</td>
<td>path</td>
<td>100%</td>
<td>26.7%</td>
<td>2.2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>control</td>
<td>2%</td>
<td>3.5%</td>
<td>1.2</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>mux</td>
<td>100%</td>
<td>22.2%</td>
<td>2.3</td>
<td>9</td>
</tr>
<tr>
<td>g1023</td>
<td>path</td>
<td>100%</td>
<td>13.3%</td>
<td>2.8</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>control</td>
<td>5%</td>
<td>1.9%</td>
<td>1.8</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>mux</td>
<td>100%</td>
<td>20.0%</td>
<td>4.6</td>
<td>18</td>
</tr>
<tr>
<td>p22810</td>
<td>path</td>
<td>100%</td>
<td>7.6%</td>
<td>29.3</td>
<td>251</td>
</tr>
<tr>
<td></td>
<td>control</td>
<td>2%</td>
<td>1.0%</td>
<td>21.7</td>
<td>267</td>
</tr>
<tr>
<td></td>
<td>mux</td>
<td>100%</td>
<td>8.0%</td>
<td>39.0</td>
<td>363</td>
</tr>
<tr>
<td>p93791</td>
<td>path</td>
<td>100%</td>
<td>7.9%</td>
<td>139.4</td>
<td>152</td>
</tr>
<tr>
<td></td>
<td>control</td>
<td>1%</td>
<td>1.0%</td>
<td>109.5</td>
<td>1301</td>
</tr>
<tr>
<td></td>
<td>mux</td>
<td>100%</td>
<td>5.9%</td>
<td>172.1</td>
<td>1859</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Design</th>
<th>No optimization</th>
<th>Optimization effort $2s$</th>
<th>Optimization effort $20s$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$n_{min}$ avg / max</td>
<td>$t_{avg}$ cycles</td>
<td>$n_{min}$ avg / max</td>
</tr>
<tr>
<td>u226</td>
<td>5.6 / 7 0.03</td>
<td>705</td>
<td>0.4 / 3 1.54 / 6.8x</td>
</tr>
<tr>
<td>d281</td>
<td>5.7 / 7 0.03</td>
<td>1718</td>
<td>0.8 / 2 1.90 / 13.4x</td>
</tr>
<tr>
<td>d695</td>
<td>6.9 / 7 0.09</td>
<td>3569</td>
<td>0.5 / 4 1.78 / 11.2x</td>
</tr>
<tr>
<td>h933</td>
<td>5.7 / 7 0.03</td>
<td>2778</td>
<td>0.9 / 3 1.31 / 18.1x</td>
</tr>
<tr>
<td>g1023</td>
<td>5.9 / 7 0.04</td>
<td>2482</td>
<td>0.5 / 2 1.89 / 10.7x</td>
</tr>
<tr>
<td>f2126</td>
<td>5.6 / 7 0.02</td>
<td>9327</td>
<td>0.8 / 3 1.78 / 12.1x</td>
</tr>
<tr>
<td>q12710</td>
<td>5.7 / 7 0.01</td>
<td>1718</td>
<td>0.8 / 3 1.78 / 12.3x</td>
</tr>
<tr>
<td>p22810</td>
<td>6.0 / 10 0.17</td>
<td>12335</td>
<td>0.5 / 4 1.65 / 33.3x</td>
</tr>
<tr>
<td>p34392</td>
<td>6.9 / 10 0.09</td>
<td>14633</td>
<td>0.7 / 3 2.02 / 49.2x</td>
</tr>
<tr>
<td>p93791</td>
<td>6.0 / 9 0.03</td>
<td>21073</td>
<td>0.8 / 4 1.84 / 28.2x</td>
</tr>
<tr>
<td>t512505</td>
<td>5.7 / 7 0.09</td>
<td>22146</td>
<td>0.5 / 3 2.31 / 87.8x</td>
</tr>
<tr>
<td>a586710</td>
<td>6.3 / 10 0.02</td>
<td>36417</td>
<td>1.2 / 6 2.19 / 74.1x</td>
</tr>
</tbody>
</table>

worst case robustness verification effort in the faulty RSNs is below 2 minutes. Therefore, many RSN design bugs can be found efficiently by just checking the robustness property.

7.5. Access Pattern Generation

To evaluate the pattern generation procedure from Section 6, we perform 1000 experiments per benchmark RSN. In each experiment, we search for the shortest scan-in sequence that merges read or write accesses to 10 randomly chosen scan segments. Optimization is performed with up to 6 additional CSU operations, executed in 6 parallel jobs (cf. Figure 8).

Column “No optimization” in Tables V and VI presents the results of pattern generation without optimization. A SAT solver is used to iteratively check the satisfiability of instances with increasing number of CSU operations until a solution is found. For the 1000 experiments, column $n_{min}$ gives the average and maximal number of CSU operations that are required to implement an access. Column $t_{avg}$ gives the average pattern generation time per access. The average access time of the unoptimized patterns is given in column cycles in clock cycles.

Access time reduction is evaluated in two series of experiments, limiting the maximal effort of the pattern generation procedure to 2 and 20 s per access. Table V and VI give the average and maximal access time reduction (column reduction) w.r.t. the unoptimized solution obtained with the SAT solver. The average and maximal number
Table VI. Access time reduction (reduction) for the SIB-based scan architecture w.r.t. unoptimized solution (cycles)

<table>
<thead>
<tr>
<th>Design</th>
<th>No optimization</th>
<th>Opt. eff. 2s</th>
<th>Opt. eff. 20s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(n_{\text{min}}/\text{max})</td>
<td>(\text{avg} / \text{max})</td>
<td>(\text{cycles} / \text{cycles})</td>
</tr>
<tr>
<td>u226</td>
<td>2.6 / 3</td>
<td>0.01</td>
<td>879</td>
</tr>
<tr>
<td>d281</td>
<td>2.7 / 3</td>
<td>0.02</td>
<td>2 039</td>
</tr>
<tr>
<td>d695</td>
<td>2.7 / 3</td>
<td>0.04</td>
<td>4 294</td>
</tr>
<tr>
<td>h953</td>
<td>2.7 / 3</td>
<td>0.01</td>
<td>4 110</td>
</tr>
<tr>
<td>g1023</td>
<td>2.7 / 3</td>
<td>0.02</td>
<td>2 507</td>
</tr>
<tr>
<td>f2126</td>
<td>2.5 / 3</td>
<td>0.01</td>
<td>9 662</td>
</tr>
<tr>
<td>q12710</td>
<td>2.5 / 3</td>
<td>0.01</td>
<td>15 550</td>
</tr>
<tr>
<td>p22810</td>
<td>2.8 / 4</td>
<td>0.08</td>
<td>12 009</td>
</tr>
<tr>
<td>p34392</td>
<td>3.1 / 4</td>
<td>0.04</td>
<td>13 122</td>
</tr>
<tr>
<td>p93791</td>
<td>2.9 / 4</td>
<td>0.19</td>
<td>36 278</td>
</tr>
<tr>
<td>t512505</td>
<td>2.7 / 3</td>
<td>0.04</td>
<td>35 275</td>
</tr>
<tr>
<td>a586710</td>
<td>2.8 / 4</td>
<td>0.01</td>
<td>24 618</td>
</tr>
</tbody>
</table>

of additional CSU operations that are required to obtain the best solution is given in column \(n_t - n_{\text{min}}\).

The proposed method significantly reduces the access time for the MUX-based benchmarks (Table V): For almost all circuits, a maximal access time reduction of over 10x is achieved. For the t512505 benchmark, the access time is reduced by up to 88x. Compared to results obtained with a SAT solver in [Baranowski et al. 2012], the proposed method achieves up to 230x access time reduction (not presented in the table). This shows that access optimization is crucial to prevent solutions with prohibitively high access time or data volume. The proposed method also reduces unnecessary access overhead: for most of the benchmarks, the average access time over the 1000 experiments 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.

For the SIB-based architecture, efficient scheduling techniques exist for access time minimization [Larsson and Ghani Zadegan 2012]. In this architecture optimal pattern generation reduces to a simple decision problem. In the following, our pattern generation procedure is evaluated for this architecture only for the sake of completeness. The results show that the access time for SIB-based benchmarks is reduced by up to a factor 1.8 w.r.t. the unoptimized solution (Table VI). In contrast to the MUX-based architecture, the local minimum is always found for the minimal number of CSU operations that is required to implement the access \((n_{\text{min}})\). The local minimum is usually found within 2 s of optimization. Extending the effort to 20 s achieves only a minimal access time reduction for larger benchmarks (italic in Table VI).

The results presented in Table V and VI are obtained with the pattern generation procedure of Section 6 that terminates as soon as a local minimum is found. We check if the access time can be improved if we allow more CSU operations. To this end, we computed shortest access sequences with 6 additional CSU operations over \(n_t\). Despite the additional effort, the resulting access times are exactly the same as those obtained in the proposed algorithm. For all the examined circuits, the proposed algorithm provides the best achievable solution among all solutions with at most 6 additional CSU operations.

7.6. Performance Analysis

In the following, we compare the performance of our CAM-based BMC approach (cf. Section 5.2) to the performance of a state-of-the-art commercial model checker. The commercial tool uses a cycle-accurate RT-level RSN model augmented with constraints which ensure that each access follows the CSU pattern (single capture cycle followed
Table VII. Performance comparison of a cycle-accurate model checker and the proposed CAM-based BMC

<table>
<thead>
<tr>
<th>Design</th>
<th>Cycle-accurate MC</th>
<th>CAM-based BMC</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$t_{\text{avg}}$ [s]</td>
<td>$t_{\text{max}}$ [s]</td>
<td>aborts [%]</td>
</tr>
<tr>
<td>u226</td>
<td>112.1</td>
<td>488.0</td>
<td>0%</td>
</tr>
<tr>
<td>d281</td>
<td>&gt; 1427.7</td>
<td>&gt; 3600.0</td>
<td>25%</td>
</tr>
<tr>
<td>d695</td>
<td>&gt; 3600.0</td>
<td>&gt; 3600.0</td>
<td>100%</td>
</tr>
<tr>
<td>b953</td>
<td>&gt; 1969.3</td>
<td>&gt; 3600.0</td>
<td>35%</td>
</tr>
<tr>
<td>g1023</td>
<td>&gt; 2431.0</td>
<td>&gt; 3600.0</td>
<td>60%</td>
</tr>
<tr>
<td>f32126</td>
<td>&gt; 362.1</td>
<td>&gt; 3600.0</td>
<td>5%</td>
</tr>
<tr>
<td>q12710</td>
<td>53.4</td>
<td>264.1</td>
<td>0%</td>
</tr>
<tr>
<td>p22810</td>
<td>&gt; 3600.0</td>
<td>&gt; 3600.0</td>
<td>100%</td>
</tr>
<tr>
<td>p34392</td>
<td>&gt; 3166.3</td>
<td>&gt; 3600.0</td>
<td>80%</td>
</tr>
<tr>
<td>p93791</td>
<td>&gt; 3600.0</td>
<td>&gt; 3600.0</td>
<td>100%</td>
</tr>
<tr>
<td>t512505</td>
<td>&gt; 2899.7</td>
<td>&gt; 3600.0</td>
<td>55%</td>
</tr>
<tr>
<td>a586710</td>
<td>&gt; 594.3</td>
<td>&gt; 3600.0</td>
<td>5%</td>
</tr>
</tbody>
</table>

by zero or more shift cycles followed by single update cycle). To make the job of the commercial model checker easier, the cycle-accurate model is manually simplified: the length of all scan segments is reduced to one-bit. Please note that without this simplification, the commercial tool exceeds the time limit of one hour in the vast majority of experiments.

For each MUX-based benchmark, we conduct 20 experiments. In each experiment, we verify the accessibility of 10 randomly chosen scan segments (cf. Section 5.3). Table VII shows the average and maximal time that is required to verify accessibility of 10 scan segments using the commercial model checker (columns 2 and 3) and CAM-based BMC (columns 5 and 6). The solving time of the commercial tool varies widely: The average proof time is 53.4 s up to over an hour. The commercial tool often exceeds the time limit of one hour; for three benchmarks none of the experiments is successful—the abort rate is given in column 4. In contrast, CAM-based BMC is successful for all benchmarks in all the experiments and exhibits much more stable runtimes below 0.18 s. This result clearly shows that the proposed CSU-accurate abstraction provides a great performance improvement over cycle-accurate models.

8. CONCLUSION

Reconfigurable scan networks allow scalable access to on-chip infrastructure. The design complexity due to hierarchies and IP reuse requires novel EDA tools for scan network verification, pattern generation, and access optimization. In this work, we propose a model that abstracts the temporal behavior of complex scan networks and thus allows efficient formal verification and optimization of access sequences. Our modeling approach supports unknown values and is applicable to a wide range of configurable architectures. Based on this model, we present an efficient method for verification of accessibility and robustness. We also propose an access pattern generation method that supports merging of multiple concurrent accesses and provides the optimal access time for a given bound on the number of scan operations. The results show that even for complex reconfigurable scan architectures the proposed method leads to significant reduction of access time by up to 88x with low computational effort.

ACKNOWLEDGMENTS

The authors thank Christian Zöllin for many insightful discussions about IEEE P1687.

REFERENCES


