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

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

ACM Transactions on Design Automation of Electronic Systems (TODAES) Vol. 20(2)
February 2015

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

Abstract: 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.

Preprint

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

RAFAL BARANOWSKI, MICHAEL A. KOCHTE, and HANS-JOACHIM WUNDERLICH,
University of Stuttgart

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, IJTAG, JTAG, pattern retargeting, P1687, reconfigurable scan network, 1149.1-2013

ACM Reference Format:
Rafal Baranowski, Michael A. Kochte, and Hans-Joachim Wunderlich. 2014. Reconfigurable Scan Networks: Modeling, Verification, and Optimal Pattern Generation. *ACM Trans. Des. Autom. Electron. Syst.* V, N, Article A (January YYYY), 26 pages.
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].
Authors' address: Rafal Baranowski, Michael A. Kochte, and Hans-Joachim Wunderlich, Universität Stuttgart, Institut für Technische Informatik, Pfaffenwaldring 47, D-70569 Stuttgart, Germany.

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

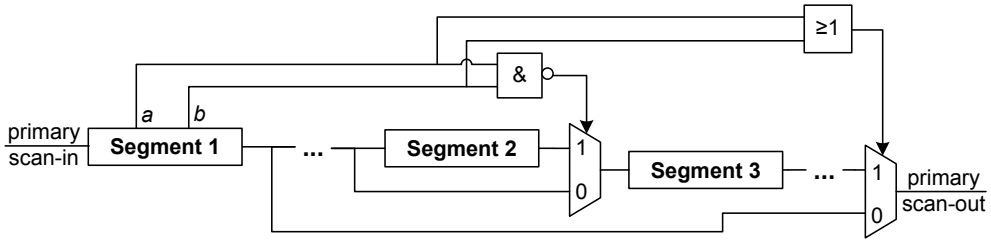

Fig. 1. Example of a reconfigurable scan network

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.

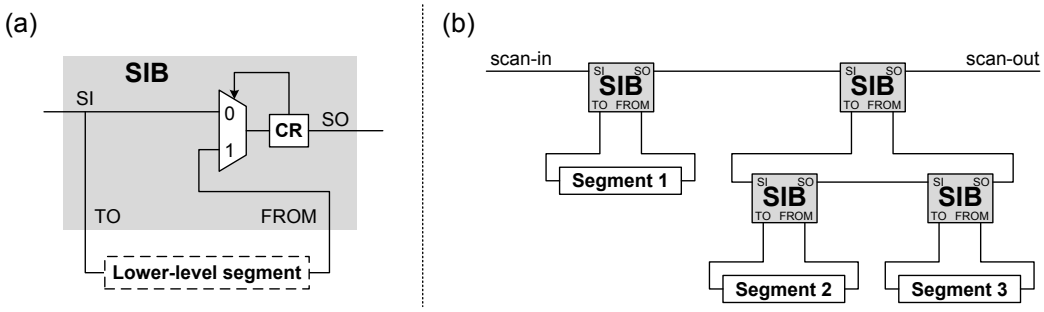

Fig. 2. (a) Segment Insertion Bit (SIB) and (b) a hierarchical RSN based on SIBs

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 $\texttt{S1} = \texttt{S3} = 1$.

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.

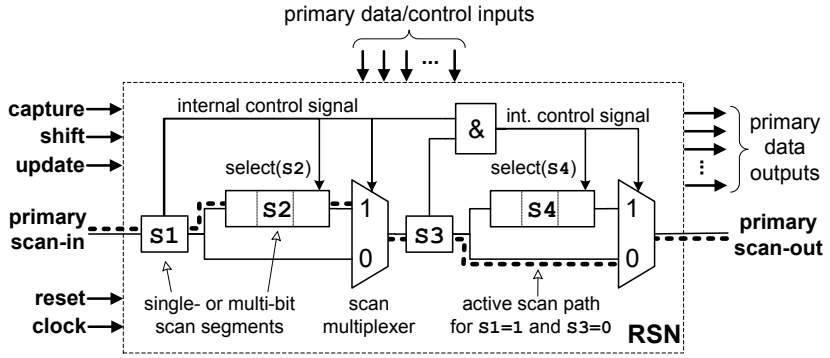

Fig. 3. Example of a reconfigurable scan network and its terminology

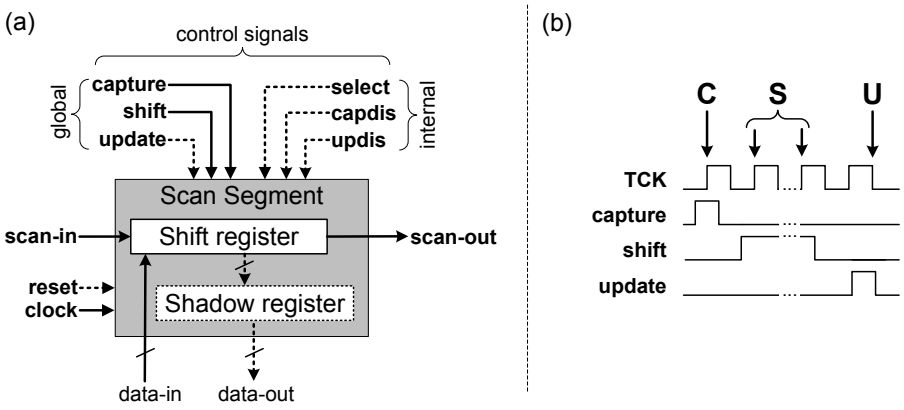

Fig. 4. (a) Scan segment; (b) Capture, Shift, Update (CSU) operation

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. S2 and S4 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. S1 and S3 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 $\texttt{S1} = 1$ 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. (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 towards 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 $M = \{S, I, C, \texttt{Active}\}$ 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 $\texttt{Active}$. 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 $\texttt{Active} : C \times S \to \{0, 1, X\}$ assigns each state element $s \in S$ in scan configuration $c \in C$ a value denoted by $\texttt{Active}(c, s)$.

Each scan configuration $c \in C$ is also treated as a function $c : S \cup I \to \{0, 1, X\}$ 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 $\texttt{Select}(c, s)$, $\texttt{Updis}(c, s)$ and $\texttt{Capdis}(c, s)$ 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 $\texttt{Active}(c, s)$ is defined as follows:

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

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

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 $\texttt{Active}$ 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 $V : C \to \{0, 1\}$ 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:

$$V(c) := \bigwedge_{s \in S} v(c, s), \tag{2}$$

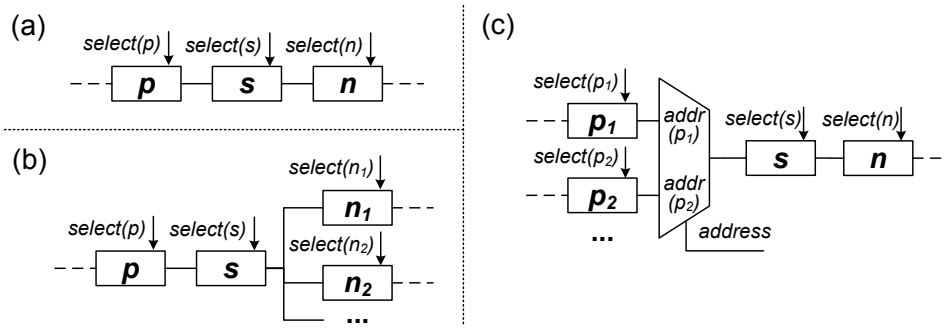

Fig. 5. (a) Chained, (b) branching and (c) multiplexed scan substructures

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 $p \in \texttt{pred}(s)$ and a single successor $n \in \texttt{succ}(s)$ (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:

$$v(c,s) := (\texttt{Select}(c,s) = 1) \Rightarrow [(\texttt{Select}(c,p) = 1) \wedge (\texttt{Select}(c,n) = 1)]. \tag{3}$$

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:

$$(\texttt{Select}(c,s) = 1) \Rightarrow \bigvee_{n \in \texttt{succ}(s)} (\texttt{Select}(c,n) = 1). \tag{4}$$

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

$$\forall_{n_k, n_l \in \texttt{succ}(s), n_k \neq n_l} : [(\texttt{Select}(c,n_k) = 1) \Rightarrow (\texttt{Select}(c,n_l) \neq 1)]. \tag{5}$$

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):

$$v(c,s) := [(\texttt{Select}(c,s) = 1) \Rightarrow (\texttt{Select}(c,p) = 1)] \wedge (4) \wedge (5). \tag{6}$$

This assures that in case of a branching scan path (fanout>1) only one branch is active, i.e. there are not 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:

$$(\texttt{Select}(c,s) = 1) \Rightarrow \bigvee_{p \in \texttt{pred}(s)} (\texttt{Select}(c,p) = 1). \tag{7}$$

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

$$\forall_{p \in \texttt{pred}(s)} : [(\texttt{Select}(c,p) = 1) \Rightarrow (\texttt{Address}(c,s) = addr(p))], \tag{8}$$

where $addr(p)$ 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:

$$v(c,s) := [(\texttt{Select}(c,s) = 1) \Rightarrow (\texttt{Select}(c,n) = 1)] \wedge (7) \wedge (8). \tag{9}$$

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: $v(c,s) := (4) \wedge (5) \wedge (7) \wedge (8)$.

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

$$\texttt{Active}(c,s) := \begin{cases} 0 & \text{if } \texttt{Select}(c,s) = 0, \\ 1 & \text{if } (\texttt{Select}(c,s) = 1) \wedge V(c), \\ X & \text{otherwise.} \end{cases} \tag{10}$$

The $\texttt{Select}$ predicates are obtained by traversing the input cones of the corresponding *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 $\texttt{Select}(c, s_b) = 0$. 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 $M = \{S, I, C, \texttt{Active}\}$ is defined as a set $T \subseteq C \times C$ with the following characteristic function:

$$T(c_1, c_2) := \bigwedge_{s \in S} [\texttt{Stable}(c_1, s) \Rightarrow (c_2(s) = c_1(s))] \wedge [\texttt{Unknown}(c_1, s) \Rightarrow (c_2(s) = X)], \tag{11}$$

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

$$\texttt{Stable}(c,s) := (\texttt{Active}(c,s) = 0) \vee (\texttt{Updis}(c,s) = 1), \tag{12}$$

$$\begin{aligned} \texttt{Unknown}(c,s) := \; & [(\texttt{Active}(c,s) = X) \wedge (\texttt{Updis}(c,s) \neq 1)] \vee \\ & [(\texttt{Active}(c,s) = 1) \wedge (\texttt{Updis}(c,s) = X)]. \end{aligned} \tag{13}$$

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 *updis* signal is active in scan configuration c_1, the state of s must not differ in the consecutive scan

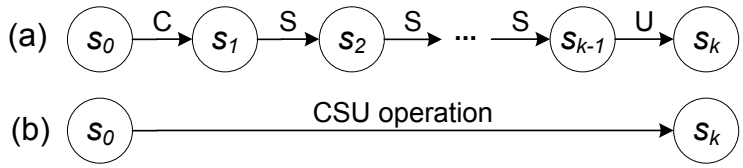

Fig. 6. State transitions during one CSU operation in (a) a cycle-accurate RSN model and (b) a CSU-accurate model (CAM)

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 $\texttt{Active}(c_1, s) = 1$ and $\texttt{Updis}(c_1, s) = 0$.

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:

$$\varphi^n := I \wedge \left[\bigwedge_{i=0}^{n-1} T_i\right] \wedge (\neg P^n), \tag{14}$$

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 $M = \{S, I, C, \texttt{Active}\}$, proving the accessibility of a scan segment $s \in S$ is equivalent to disproving the following LTL formula in the CAM:

$$\boldsymbol{G}\neg[(\texttt{Active}(s) = 1) \wedge (\texttt{Updis}(s) = 0) \wedge (\texttt{Capdis}(s) = 0)]. \tag{15}$$

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:

$$\begin{aligned}\texttt{Check}(s, c_0, n) := \; & \mathbf{1}(c_0) \wedge \bigwedge_{i=1\ldots n} T(c_{i-1}, c_i) \wedge \\ & \bigvee_{i=0\ldots n} \left[(\texttt{Active}(s) = 1) \wedge (\texttt{Updis}(s) = 0) \wedge (\texttt{Capdis}(s) = 0)\right],\end{aligned} \tag{16}$$

where $\mathbf{1}(c_0)$ 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 $\texttt{Check}(s, c_0, n)$ 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 $\boldsymbol{G}V$ 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 $\texttt{Active}(\texttt{c},\texttt{s}) := \texttt{Select}(\texttt{c},\texttt{s})$ 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 $\boldsymbol{G}V$ 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 $M = \{S, I, C, \texttt{Active}\}$ 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:

$$\mathbf{1}(c_0) \wedge \neg V(c_0), \tag{17}$$

where $\mathbf{1}(c_0)$ 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:

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

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.

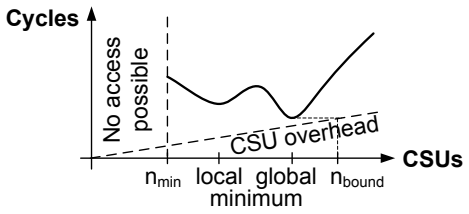

Fig. 7. Example of a minimal access time curve

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 $M = \{S, I, C, \texttt{Active}\}$ 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:

$$(c_n = c_t) \wedge \forall_{i=1\ldots n} ((c_{i-1}, c_i) \in T) \tag{19}$$

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

$$\texttt{Cycles}(c_0, \ldots, c_n) := D \cdot n + \sum_{i=0}^{n-1} \sum_{s \in S} (\texttt{Active}(c_i, s) = 1), \tag{20}$$

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 $\texttt{Cycles}(c_0, c_1, \ldots, c_n = c_t)$ 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 $n_{\texttt{min}}$ 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 $1, 2, 3 \ldots$ CSU operations. Let $\texttt{Cycles}_n$ denote the access time of the pattern with n CSU operations, and set $n_{\texttt{bound}} := \lceil \texttt{Cycles}_n / D \rceil$. The access time of the pattern with n CSUs is the *global minimum* if there does not exist any other solution with up to $n_{\texttt{bound}}$ CSUs with lower access time.

In practice, due to limited computational resources, the search for all solutions with up to $n_{\mathtt{bound}}$ 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 n CSU operations the target segment $s \in S$ is read in at least one of the *intermediate* scan configurations, i.e., $\bigvee_{i=0\ldots n-1}[(\mathtt{Active}(c_i, s) = 1) \wedge (\mathtt{Capdis}(c_i, s) = 0)]$ is true. 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 leads to the minimal value of a pseudo-Boolean cost function A among all assignments that satisfy a Boolean formula Ψ. The pseudo-Boolean cost function has the form:

$$A(x_1, x_2, \ldots, x_k) = a_0 + \sum_{i=1}^{k} a_i \cdot x_i, \tag{21}$$

where $a_0, a_1, \ldots, a_k \in \mathbb{Z}$ and $x_i \in \{0, 1\}$. Pseudo-Boolean optimization can be performed, for instance, by incremental SAT solving techniques with pseudo-Boolean constraint translation to SAT [Eén and Sörensson 2006], or speculative model enumeration techniques [Gebser et al. 2011].

According to condition (19), an access (c_0, c_t) is implemented by a sequence of scan configurations $c_0, c_1, \ldots, c_n$ exactly when the following Boolean formula is satisfied:

$$\mathtt{Access}(c_0, c_t, n) := \mathbf{1}(c_0) \wedge \bigwedge_{i=1\ldots n} T(c_{i-1}, c_i) \wedge \bigwedge_{s \in S} (c_n(s) = c_t(s)), \tag{22}$$

where $\mathbf{1}(c_0)$ 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 $\mathtt{Access}(c_0, c_t, n)$, given by (22), is subject to pseudo-Boolean optimization with the cost function $\mathtt{Cycles}(c_0, c_1, \ldots, c_n)$, 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

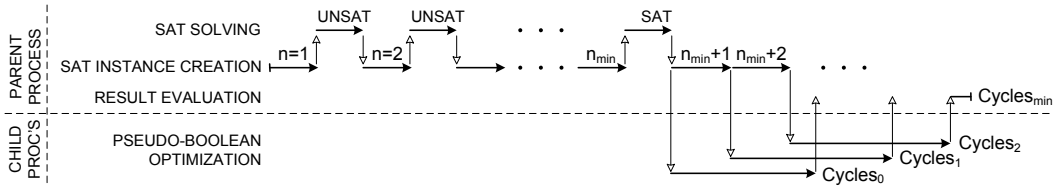

Fig. 8. Parallel execution of the pattern generation procedure (horizontal lines represent processing; vertical lines represent inter-process communication)

$s \in S$ belongs to the active scan path if $\texttt{Active}(c_{i-1}, s) = 1$. 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 $\texttt{Cycles}_n$ 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 $\texttt{Cycles}_{n+1}$. If the cost of the new solution is higher than the previous one, i.e. when $\texttt{Cycles}_{n+1} > \texttt{Cycles}_n$, the pattern generation procedure terminates. Otherwise, the number of CSU operations is increased and the procedure is repeated until the user specified bound $n_{\texttt{max}}$ 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 $n_{\texttt{opt}} > n_t + 1$ 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 *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 $n_{\texttt{min}}$ that is required to implement the access. After $n_{\texttt{min}}$ 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 $n_{\texttt{min}}$ 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 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 ($\texttt{AM} = 0$), in which only the configuration registers (`C`) can be accessed, or data access mode ($\texttt{AM} = 1$). 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).

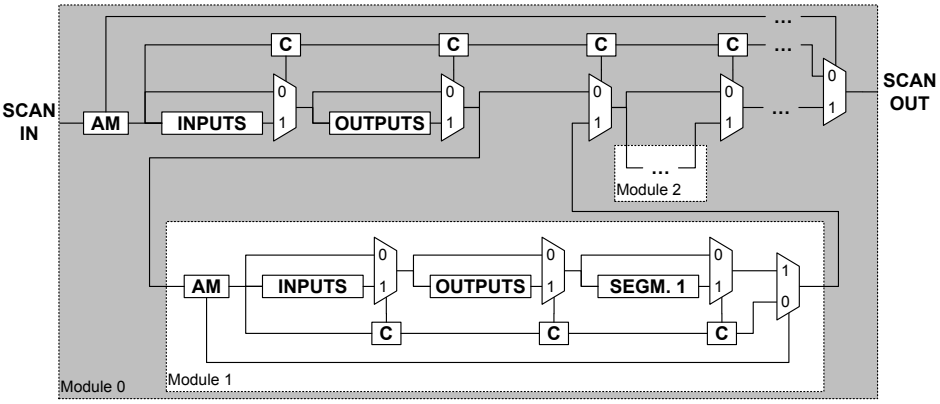

Fig. 9. MUX-based scan architecture for the p34392 benchmark (note: *select* signals of scan segments are omitted for better readability)

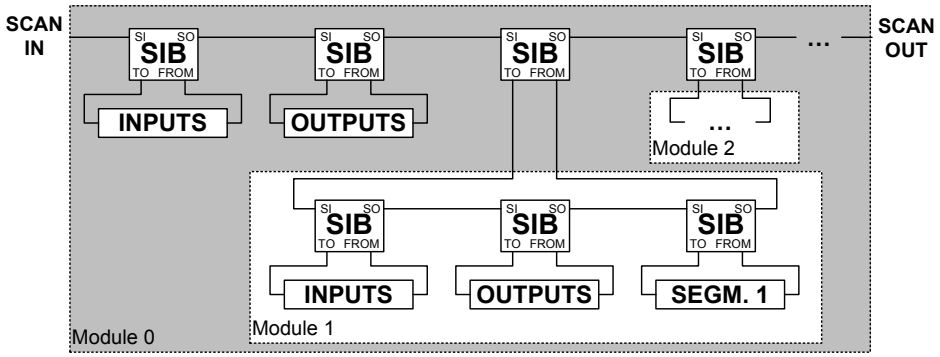

Fig. 10. SIB-based scan architecture for the p34392 benchmark (note: *select* signals of scan segments are omitted for better readability)

Table I. Characteristics of the benchmark scan networks

Design	MUX-based Architecture			SIB-based Architecture		
	Num. MUX	Total #scan segm.	Total #scan bits	Num. SIB	Total #scan segm.	Total #scan bits
u226	59	99	1 475	50	90	1 466
d281	67	117	3 880	59	109	3 872
d695	178	335	8 407	168	325	8 397
h953	63	109	5 649	55	101	5 641
g1023	94	159	5 400	80	145	5 386
f2126	45	81	15 834	41	77	15 830
q12710	30	51	26 188	25	47	26 183
p22810	311	565	30 139	283	537	30 111
p34392	142	245	23 261	123	226	23 242
p93791	653	1241	98 637	621	1209	98 605
t512505	191	319	77 037	160	288	77 006
a586710	47	79	41 682	40	72	41 675

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

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

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

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

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_{solve}^{max} is the maximum solve time for the check of a single scan segment (iteration), whereas t_{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 $\texttt{Active}(\texttt{c},\texttt{s}) := \texttt{Select}(\texttt{c},\texttt{s})$ 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_{total}^{avg} and t_{total}^{max} 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

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

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

Design	No optimization			Optimization effort $2s$		Optimization effort $20s$	
	$n_{\min}$ avg / max	t_{avg} [s]	`cycles` [cycles]	$n_t - n_{\min}$ avg / max	`reduction` avg / max	$n_t - n_{\min}$ avg / max	`reduction` avg / max
u226	5.6 / 7	0.03	705	0.4 / 3	1.54 / 6.8x	0.4 / 2	1.54 / 6.8x
d281	5.7 / 7	0.03	1 718	0.8 / 2	1.90 / 13.4x	0.8 / 3	1.91 / 13.7x
d695	6.0 / 7	0.09	3 569	0.5 / 4	1.78 / 11.2x	0.7 / 4	1.84 / 11.2x
h953	5.7 / 7	0.03	2 776	0.9 / 3	1.91 / 16.1x	0.9 / 3	1.91 / 16.1x
g1023	5.9 / 7	0.04	2 482	0.5 / 2	1.89 / 10.7x	0.6 / 2	1.93 / 10.7x
f2126	5.6 / 7	0.02	9 327	0.8 / 3	1.78 / 12.1x	0.9 / 3	1.79 / 12.1x
q12710	5.7 / 7	0.01	17 769	0.8 / 3	1.78 / 12.3x	0.8 / 3	1.78 / 12.3x
p22810	6.0 / 10	0.17	12 335	0.5 / 4	1.65 / 33.3x	0.5 / 3	1.75 / 33.7x
p34392	6.9 / 10	0.09	14 633	0.7 / 3	2.02 / 49.2x	0.8 / 4	2.16 / 49.2x
p93791	6.0 / 9	0.38	21 073	0.8 / 4	1.84 / 28.2x	0.9 / 4	1.99 / 28.2x
t512505	5.7 / 7	0.09	22 146	0.5 / 3	2.31 / 87.8x	0.5 / 3	2.39 / 87.8x
a586710	6.3 / 10	0.02	36 417	1.2 / 6	2.19 / 74.1x	1.3 / 5	2.26 / 74.1x

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`)

Design	No optimization			Opt. eff. $2s$	Opt. eff. $20s$
	$n_{\texttt{min}}$ avg / max	$t_{\texttt{avg}}$ [s]	`cycles` [cycles]	`reduction` avg / max	`reduction` avg / max
u226	2.6 / 3	0.01	879	1.09 / 1.81x	1.09 / 1.81x
d281	2.7 / 3	0.02	2 039	1.13 / 1.81x	1.13 / 1.81x
d695	2.7 / 3	0.04	4 294	1.14 / 1.61x	*1.15* / 1.61x
h953	2.7 / 3	0.01	3 110	1.16 / 1.69x	1.16 / 1.69x
g1023	2.7 / 3	0.02	2 507	1.17 / 1.62x	1.17 / 1.62x
f2126	2.5 / 3	0.01	9 662	1.11 / 1.72x	1.11 / 1.72x
q12710	2.5 / 3	0.01	16 550	1.09 / 1.68x	1.09 / 1.68x
p22810	2.8 / 4	0.08	12 009	1.06 / 1.25x	*1.12* / *1.50*x
p34392	3.1 / 4	0.04	13 122	1.16 / 1.45x	*1.17* / *1.66*x
p93791	2.9 / 4	0.19	36 278	1.09 / 1.24x	*1.14* / *1.48*x
t512505	2.7 / 3	0.04	35 275	1.13 / 1.57x	*1.18* / *1.74*x
a586710	2.8 / 4	0.01	24 618	1.13 / 1.85x	1.13 / 1.85x

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

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 ($n_{\texttt{min}}$). 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

Design	Cycle-accurate MC			CAM-based BMC		Speedup	
	t_{avg} [s]	t_{max} [s]	aborts [%]	t_{avg} [s]	t_{max} [s]	avg	max
u226	112.1	488.0	0%	0.01	0.01	12 808x	48 799x
d281	> 1427.7	> 3600.0	25%	0.01	0.01	> 163 166x	> 721 049x
d695	> 3600.0	> 3600.0	100%	0.02	0.03	> 171 644x	> 180 267x
h953	> 1969.3	> 3600.0	35%	0.01	0.01	> 225 066x	> 720 834x
g1023	> 2431.0	> 3600.0	60%	0.01	0.02	> 231 519x	> 360 919x
f2126	> 362.1	> 3600.0	5%	0.01	0.01	> 45 261x	> 360 521x
q12710	53.4	264.1	0%	0.01	0.01	8 208x	26 411x
p22810	> 3600.0	> 3600.0	100%	0.04	0.06	> 86 860x	> 120 154x
p34392	> 3166.3	> 3600.0	80%	0.02	0.03	> 147 269x	> 360 469x
p93791	> 3600.0	> 3600.0	100%	0.10	0.18	> 36 789x	> 51 502x
t512505	> 2989.7	> 3600.0	55%	0.02	0.03	> 135 893x	> 180 286x
a586710	> 594.3	> 3600.0	5%	0.01	0.01	> 69 914x	> 621 978x

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

1149.1 2013. IEEE Standard for Test Access Port and Boundary-Scan Architecture 1149.1-2013. (2013).

R. Baranowski. 2014. *Reconfigurable Scan Networks: Formal Verification, Access Optimization, and Protection*. Ph.D. Dissertation. University of Stuttgart. http://elib.uni-stuttgart.de/opus/volltexte/2014/8982

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

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

A. Benso, S. Di Carlo, P. Prinetto, and Y. Zorian. 2008. IEEE Standard 1500 Compliance Verification for Embedded Cores. *IEEE Trans. VLSI Syst.* 16, 4 (2008), 397–407.

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

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

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

A.T. Dahbura, M.U. Uyar, and Chi W.Y. 1989. An Optimal Test Sequence for the JTAG/IEEE P1149.1 Test Access Port Controller. In *Proc. IEEE International Test Conference (ITC)*. 55–62. DOI:http://dx.doi.org/10.1109/TEST.1989.82277

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

E.B. Eichelberger and TW Williams. 1977. A Logic Design Structure for LSI Testability. In *Proceedings of the Design Automation Conference (DAC)*. 462–468.

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

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

R.L. Fisher. 2002. 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.

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

M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. 2007. clasp : A Conflict-Driven Answer Set Solver. In *Logic Programming and Nonmonotonic Reasoning*. LNCS, Vol. 4483. Springer, 260–265.

F. Ghani Zadegan, U. Ingelsson, G. Carlsson, and E. Larsson. 2011. Design Automation for IEEE P1687. In *Proceedings of the Design, Automation Test in Europe Conference (DATE)*. 1412–1417.

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

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

S.C. Kleene. 1950. *Introduction to Metamathematics*. D. Van Nostrand, Princeton, NJ.

E. Larsson and F. Ghani Zadegan. 2012. Accessing Embedded DfT Instruments with IEEE P1687. In *Proceedings of the IEEE Asian Test Symposium (ATS)*. 71–76.

E. Larsson and K. Sibin. 2012. Fault management in an IEEE P1687 (IJTAG) environment. In *IEEE International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*. 7.

E.J. Marinissen, V. Iyengar, and K. Chakrabarty. 2002. A Set of Benchmarks for Modular Testing of SOCs. In *Proceedings of the IEEE International Test Conference (ITC)*. 519–528.

K.L. McMillan. 2003. Interpolation and SAT-Based Model Checking. In *Computer Aided Verification*. Lecture Notes in Computer Science, Vol. 2725. Springer, 1–13.

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

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

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

J. Remmers, M. Villalba, and R. Fisette. 2004. Hierarchical DFT Methodology - A Case Study. In *Proceedings of the IEEE International Test Conference (ITC)*. 847–856.

M. Sheeran, S. Singh, and G. Stålmarck. 2000. Checking Safety Properties Using Induction and a SAT-Solver. In *Formal Methods in Computer-Aided Design*. Lecture Notes in Computer Science, Vol. 1954. Springer, 127–144.

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

N. Stollon. 2011. *On-Chip Instrumentation: Design and Debug for Systems on Chip*. Springer US.