Optimization of Retargeting for IEEE 1149.1 TAP Controllers with Embedded Compression

Sebastian Huhn*† Stephan Eggersglüß*† Krishnendu Chakrabarty‡ Rolf Drechsler*†

*University of Bremen, Germany
{huhn,segg,drechsle}@informatik.uni-bremen.de

†Cyber-Physical Systems
DFKI GmbH
28359 Bremen, Germany

‡Duke University
Durham, NC 27708, USA
krish@ee.duke.edu

***Abstract*—We present a formal optimization technique that enables retargeting for codeword-based IEEE 1149.1-compliant TAP controllers. The proposed method addresses the problem of high test data volume and *Test Application Time* (TAT) for a system-on-chip design during board or in-field testing, as well as during debugging. This procedure determines an optimal set of codewords with respect to given hardware constraints, e.g., embedded dictionary size and the interface to the Test Data Register in the IEEE 1149.1 Std. A complete traversal of the spanned search space is possible through the use of formal methods. An optimal set of codewords can be determined, which is directly utilized for retargeting. The proposed method is evaluated using test data with high-entropy, which is known to be the least amenable to compression, as well as input data for debugging and *Functional Verification* (FV) test data. Our results show a compression ratio improvement of more than 30% and a reduction in TAT up to 20% compared to previous techniques.**

I. Introduction

System-on-Chip (SoC) *Integrated Circuits* (ICs) are now widely used in the semiconductor industry. SoC designs inevitably lead to higher test complexity and the problem of reduced test access to internal logic blocks. An important aspect to be considered in the SoC design phase is the accessibility of nested sub-modules. Hence, test access mechanisms are embedded into the design. The IEEE 1149.1 Std. specifies such a *Test Access Port* (TAP), which is commonly used within industrial designs. This standardized TAP provides capabilities to transfer test data to the *Circuit-under-Test* (CuT). However, a problem in using the IEEE 1149.1 Std. for testing or debugging is that it provides only a serial interface for data transfer, leading to a high data transfer time. In addition, the high *Test Data Volume* (TDV) and debug data volume require the use of a considerable amount of tester memory, which is often not feasible in a system environment in the field or for rapid post-silicon debug using IEEE 1149.1.

Test compression techniques have been proposed in the literature for reducing TDV. These methods embed dedicated hardware on-chip to enable the transfer of compressed stimuli from the tester and on-the-fly decompression [1]–[6]. For example, *Embedded Deterministic Test* (EDT) [1] achieves very high compression for test data derived using *Automatic Test Pattern Generation* (ATPG) tools. However, either the test data have to contain a large number of unspecified values for EDT to be effective, or the generation of test cubes using ATPG must be integrated in the test compression procedure [7].

Beside this, other hardware modules have been developed with the aim to compress scan test data by taking advantage of static compression techniques [2], e.g., *Run-Length Encoding* (RLE)-based methodology [4], dictionary-based techniques [3] or even combined techniques within hybrid compression modules [5], [6]. The dictionary has a major impact on the achievable compression ratio, which is defined as the fraction of the test data after compression compared to the test data before compression. The work in [8] proposes an approach to generate a suitable dictionary for test data containing a large proportion of unspecified values. However, this technique does not work for fully-specified data. Other techniques from the field of software-based compression exist, e.g., the Huffman encoding, which is applied in [9] for test data compression. In general, these dictionary-based approaches are unable to incorporate any hardware constraints, e.g., the available dictionary size or other specific properties of the codewords. Due to this fact, the work in [10] proposes a domain-specific word-based Huffman technique which addresses some of these shortcomings. More complex techniques such as the *Lempel-Ziv* (LZ) algorithm [11] has been adapted for implementation in hardware. These modules are capable of handling a large volume of incoming data. However, the additional area overhead as well as the adverse impact on timing behavior have to be considered.

To tackle these limitations and to reduce the overall TDV, VecTHOR has been proposed in [12] as a compression-based architecture for standardized IEEE 1149.1-compliant TAP controllers. In particular, a scheme has been developed to take advantage of a codeword-based compression technique, which is dynamically configurable and therefore applicable even on high-entropy test data. Furthermore, the extension requires only a slight overhead in hardware and no additional pins at the chip level. For utilizing this compression technique, existing test data have to be retargeted off-chip only once. Typically, this is done by a retargeting framework, which determines a configuration, processes the test data and generates a compressed test data stream. This stream is transferred for each test run to the CuT embedding the compression-based TAP controller. Subsequently, the compressed stream is decompressed on-chip such that the original test data are restored.

The VecTHOR approach achieves a promising TDV reduction when it is applied to test data as presented in [12]. However, the approach is not able to tap its full potential. The effectiveness strongly depends on the selection of the codewords. In [12], a simple heuristic is used to determine suitable codewords. Due to the large number of search parameters to consider, a greedy algorithm is not likely to find optimal codewords. In particular, this shortcoming can be observed when the approach is applied on high-entropy test data.

This work proposes a new formal optimization-based technique to determine a set of optimal codewords. The problem of finding a set of effective codewords for a given test data stream is modeled as a *Pseudo-Boolean Optimization* (PBO) problem for which solvers exist that are able to compute appropriate solutions in reasonable time. The generated codewords can then be used to dynamically configure a compression-based TAP architecture to increase the compression ratio and even decrease the number of required test cycles.

Experiments are conducted on random test data, on debugging data of a JPEG encoder, as well as on commercially representative FV test data for a state-of-the-art softcore microprocessor. Random test data have characteristically a high-entropy. Thus, this data typically determine the lower

bound on the compression ratio [13]. Both results show that the new optimization-based approach improves the compression results significantly. A TDV reduction by up to 37.1% can be achieved for high-entropy data. In particular, the *Test Application Time* (TAT) overhead of previous techniques is completely eliminated. In fact, the TAT for real FV test data can be even reduced by up to 20% compared to the time an uncompressed data transfer consumes.

The structure of this paper is as follows: Section II briefly presents the formal methods being applied and the previously proposed compression technique. Subsequently, Section III draws the formal model used for the problem instance. Experimental results are shown in Section IV. Finally, Section V summarizes the paper and discusses possible future work.

II. Background

The *Boolean Satisfiability* (SAT) problem asks the question whether a satisfying solution for a given Boolean function exists. These functions can be represented by using a *Conjunctive Normal Form* (CNF). A CNF Φ is a conjunction of clauses, whereby, such a clause ω is a disjunction of literals and a literal represents a Boolean variable ν in its positive x or negative form $\bar{x}$. This Boolean function $\Phi : \{0,1\}^n \rightarrow \{0,1\}$ is classified as *satisfiable* (sat) if an assignment of all variables exists such that $\Phi = 1$ holds. Otherwise, it is classified as *unsatisfiable* (unsat) [14]. In fact, such a SAT problem can be used to model several (research) questions. Generally, solving these functions is a hard computational task, hence, a lot of research work has been spent on developing powerful solving algorithms (SAT solvers) to address this challenging problem.

Example 1. *Let* $\Phi = (x_1 + \bar{x}_2 + x_3) \cdot (\bar{x}_1 + x_2) \cdot (x_2 + \bar{x}_3)$. *Consequently,* $x_1 = 1,\ x_2 = 1$ *and* $x_3 = 0$ *is a satisfying variable assignment.*

The *Pseudo-Boolean* (PB) SAT problem allows for an integration of weights. The PB-SAT instance $\Phi : \{0,1\}^n \rightarrow \{0,1\}$ consists of conjugated constraints $\sum_{i=1}^{n-1} c_i \cdot \widehat{x}_i \geq c_n$ using $c_1, \ldots, c_n \in \mathbb{Z}$ as weights and $\widehat{x}_i$ as positive or negative literals. Additionally, the PBO problem extends the PB-SAT problem such that an objective function $\mathcal{F}$ can be integrated in order to assess the quality of the determined solution. By this, it is possible not only to give an arbitrary solution as regular SAT solvers do, but to determine the optimal solution with respect to $\mathcal{F}$. PBO-based or similar optimization-based procedures have already been successfully applied in the testing domain, e.g., in [15], [16]. Although dedicated solving algorithms exist for these kind of problems, many of these algorithms use SAT solving techniques internally. The PBO problem is one of these problems. Here, the (PB-)SAT instance, i.e., the Boolean formula in CNF or the PB constraints, respectively, is extended with an objective function $\mathcal{F}$.[1] Typically, the objective function $\mathcal{F}$ is given as a linear sum:

$$\mathcal{F}(x_1, \ldots, x_k) = \sum_{i=1}^{k} m_i \cdot \widehat{x}_i \text{ with } m_1, \ldots, m_k \in \mathbb{Z}$$

Basically, the result of $\mathcal{F}$ is the arithmetic sum of all constants m_i associated to a literal $\widehat{x}_i$, which evaluates to true under a given assignment. Usually, a PBO solver utilizes the minimization as a solving target, i.e., it returns the solution which minimizes $\mathcal{F}$.

Example 2. *Let* $\Phi = (3x_1 + 4\bar{x_2} + x_3 \geq 3) \wedge (3\bar{x}_1 + 4x_2 \geq 2) \wedge (4x_2 + \bar{x}_3 \geq 4)$ *and* $\mathcal{F} = 1x_1 + 1x_2 + 1x_3$. *In this case, the solution* $x_1 = 1,\ x_2 = 1$ *and* $x_3 = 0$ *satisfies the given PB-SAT instance and, at the same time, minimizes the outcome of the objective function* $\mathcal{F}$ *(*$\mathcal{F} = 2$*). In contrast, the solution* $x_1 = 1,\ x_2 = 1$ *and* $x_3 = 1$ *also satisfies the instance, but results in* $\mathcal{F} = 3$*, which is higher than the previous solution.*

Modern PBO solvers also support multiple objective functions $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_n$. Here, priorities are used. First, $\mathcal{F}_1$ is used as objective function. Afterwards, the solution is improved concerning $\mathcal{F}_2, \ldots, \mathcal{F}_n$. However, it is not possible to decrease the result of the objective function with a higher priority.

Another important aspect to be briefly introduced concerns the accessibility for test or debug purposes of a SoC. A dedicated TAM is embedded into the design to ensure accessibility by providing a suitable communication channel to transfer test or debug data to nested sub-modules, e.g., by embedding the standardized and commonly used IEEE 1149.1 (JTAG) [17] into the SoC.

An extended TAP controller has been developed in [12], which combines the TAM with a dynamically configurable codeword-based compression architecture providing RLE capabilities. Instead of transferring the original test data directly to the SoC using legacy JTAG, this data are preprocessed by a suitable retargeting framework off-chip. This framework generates the compressed test data consisting of a *Compressed Data Word* (CDW) sequence to be transferred to the extended TAP controller. After transmission, the CDW sequence is expanded by the *Dynamic Decompressing Unit* (DDU) on-chip, i.e., the original test data chunks namely the (corresponding) *Uncompressed Data Words* (UDWs) are restored into a certain *Test Data Register* (TDR). This DDU includes a dictionary storing the codewords, which can be configured before the actual data transfer starts. The instruction set of the TAP controller is enhanced by additional instructions *compr_preload* and *compr*, which are used to dynamically configure (load codewords) and apply the decompression. To perform this, a configuration $\mathcal{C}$ consisting of the instruction code *compr_preload* as well as a set of codewords has to be determined.

It is necessary to incorporate certain hardware constraints such as the number of codewords. The parameter *Chunk Size* (CS) is used to control the trade-off between compression capability and hardware overhead. A good trade-off is achieved by $CS = 3$ leading to the following $\sum_{i=0}^{3} 2^i = 15$ binary encodings:

- ∅, ‘0‘, ‘1‘, ‘00‘, ‘01‘, ‘10‘, ‘11‘
- ‘000‘, ‘001‘, ‘010‘, ‘011‘, ‘100‘, ‘101‘, ‘110‘, ‘111‘

These listed CDWs have to be configured in the DDU realizing a mapping function $\Psi(\text{CDW}^c) \rightarrow \text{UDW}^u$ such that $0 \leq c \leq CS$ holds for the length c of the CDW. The length u of the UDW is determined by the synthesis parameter of the connected TDR, which is fixed to $u = \{1, 4, 8\}$ in [12]. During configuration of the DDU, the images of Ψ become overwritten. An exemplary implementation of such a mapping function Ψ is shown in Table I. In this example, the CDWs ‘∅‘, ‘0‘ and ‘1‘ hold exposed functions: ‘∅‘ encodes the RLE, i.e., at the current time t, the last successfully transferred CDW (at time of $t-1$) is repeated. The remaining two are mapped to namely *Single Bit Injections* (SBIs) ensuring that all possible incoming data can be processed.

Generally, the approach is intended to be used as follows:

1) Preprocess existing test data off-chip by retargeting framework. Generates a) compressed test data $\mathcal{D}$ and b) suitable configuration $\mathcal{C}$ for DDU,
2) Load instruction *compr_preload* & configuration data $\mathcal{C}$,
3) Load instruction *compr* & compressed data $\mathcal{D}$.

Table I and II show a simplified example to demonstrate the importance of careful codeword selection. Two different configurations $\mathcal{C}^1$ and $\mathcal{C}^2$ are presented in Table I. Here, the mapping between codewords (column *CDW*) and uncompressed

[1] Since a Boolean formula in CNF can be easily transformed into PB constraints and modern PBO solvers typically accept CNFs as input, we use the notion of CNF in this paper if possible.

TABLE I: Example configurations $\mathcal{C}^1$ and $\mathcal{C}^2$ for mapping function Ψ using $CS = 3$

No.	CDW	UDW @ $\mathcal{C}^1$	UDW @ $\mathcal{C}^2$
1	$\varnothing$	CDW@$t-1$	CDW@$t-1$
2	0	0	0
3	1	1	1
4	00	1111	01011010
5	01	0101	0110
6	10	0110	0001
7	11	00000000	10010110
8	000	01010101	1100
9	001	1010	1010
10	010	0000	0000
11	011	10101010	10101010
12	100	1000	1000
13	101	1001	1001
14	110	0001	0001
15	111	11111111	11111111

TABLE II: Application on example data using configuration $\mathcal{C}^1$ and $\mathcal{C}^2$

	Byte index	0							1							2							
	Bit index	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
Config.	Incoming data	0	1	0	1	1	0	1	0	0	1	1	0	0	0	0	1	1	0	0	1	0	1	1	0
$\mathcal{C}_1$	Compressed data $\mathcal{D}^1$	01				001				10				110				101				10			
Config.	Incoming data	0	1	0	1	1	0	1	0	0	1	1	0	0	0	0	1	1	0	0	1	0	1	1	0
$\mathcal{C}_2$	Compressed data $\mathcal{D}^2$	00								01				10				11							

data words (column *UDW*) is given. As shown in Table II, $\mathcal{I}$ contains 24 bits and is compressed to a sequence of 6 CDWs by using $\mathcal{C}^1$ and to a sequence of 4 CDWs by using $\mathcal{C}^2$. These CDW sequences contain 15 bits overall ($\mathcal{C}^1$) and 8 bits ($\mathcal{C}^2$), i.e., a TDV reduction about 37.5% ($\mathcal{C}^1$) and about 66.7% ($\mathcal{C}^2$) is achieved, respectively. Obviously, the selected $\mathcal{C}$ has a strong impact on the compression ratio. Finding a good configuration is a challenging task due to mutual dependencies and codeword overlaps in the test data stream.

III. Optimization-based Retargeting

TAP controllers with embedded compression offer a powerful mechanism for TDV and TAT reduction. However, the problem of codeword selection for a test data stream, particularly for high-entropy data, is yet to be solved satisfactorily. This section describes the new proposed technique that addresses this shortcoming. A retargeting procedure using formal optimization-based techniques to determine an optimal configuration as well as the best sequence of replacements is presented. The proposed technique is applied on the parameters of the codeword-based TAP controller, which were used in [12]. However, the approach is easily adaptable to other parameters.

A. General Idea

The main idea is to formulate the retargeting problem as a formal optimization problem. The proposed retargeting flow is shown in Figure 1. The regular uncompressed test data $\mathcal{I}$ are given as input. An optimization problem consisting of SAT and PB constraints, respectively, and an optimization function are formulated. Subsequently, a PBO solver is called to solve the problem instance. The result is a satisfying model, whose data are extracted and directly used to determine an optimal configuration $\mathcal{C}$ as well as the targeted compressed test data $\mathcal{D}$. Since the TAP controller is dynamically configurable, the calculated configuration $\mathcal{C}$ can be loaded into the controller to decompress the compressed test data $\mathcal{D}$ on-chip, which restores the original uncompressed test data $\mathcal{I}$. This technique allows a TDV reduction of the data, which have to be serially transferred into the TAP controller. This procedure can also be used for partitioned data streams due to the dynamic configuration.

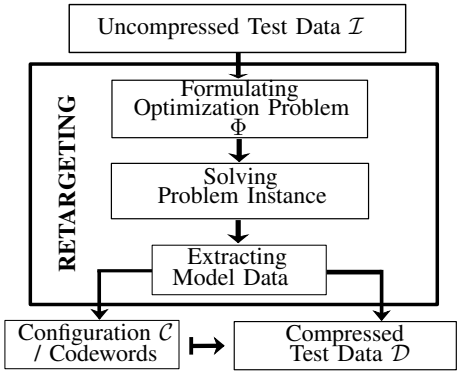

Fig. 1: Proposed Retargeting Flow

The hardware constraints are considered by the following assumptions:

1) A circuit design that embeds a codeword-based TAP controller, which includes a DDU with a dictionary consisting of $n = \sum_{i=2}^{CS} 2^i$ dynamically configurable entries.
2) Three dictionary entries are statically included, i.e., $\varnothing$, '0', '1'. These encode the RLE capability and both codewords with a length of 1 for SBI modeling. Therefore, only $n-3$ codeword entries are dynamically configured by the proposed retargeting procedure.
3) Let $u = \{1, 4, 8\}$ be the UDW length supported by the TDR interface. Then, $2^1 + 2^4 + 2^8$ possible Boolean permutations exist. Each permutation is a candidate for being included as a codeword in the DDU.
4) The incoming test data sequence $\mathcal{I}$ contains only fully-specified values, i.e., '0' and '1'.

Lemma 1. *Given the mapping function Ψ and configuration $\mathcal{C}$ that holds two CDWs for the SBIs cdw_i and cdw_j, such that $\Psi(cdw_i) = 0$ and $\Psi(cdw_j) = 1$. Then, every possible sequence of incoming data $\mathcal{I}$ can be represented by a sequence of CDWs, i.e., the compressed test data $\mathcal{D}$. At least, this can be achieved by simply using the cdw_i or cdw_j successively.*

Lemma 2. *A compressed test data $\mathcal{D}$ and a configuration $\mathcal{C}$ are given. Then, the original test data $\mathcal{I}$ are restored if the mapping function Ψ (using $\mathcal{C}$) is applied on $\mathcal{D}$.*

Based on these basic conditions, a formal PBO model has to be built, which allows the automatic computation of valid configurations with optimal codeword selection for effective TDV and TAT reduction.

B. Generating PBO Instance

The PBO instance to develop consists of two parts: Φ, $\mathcal{F}$. The formulation of the constraints Φ spans the solution space such that each solution to the set of constraints is a valid configuration and vice versa. The optimization function $\mathcal{F}$ is then used to rate each solution in terms of their costs. By this, the search is guided towards the most beneficial solution.

First, the meaning of the variables of the problem instance is described. Since each possible segment in the data stream $\mathcal{I}$ has to be covered by a codeword, a variable ν_i^{uC} is used to determine the status of each of these segments. The parameter i denotes the offset of the segment in $\mathcal{I}$. This has to be done for each considered UDW length u. An example is shown in Table III. Here, all possible segments of a 8 bit data stream are listed. For $u = 8$, only one segment is possible, $u = 4$ leads to 5 possible segments and for $u = 1$, 8 segments are possible. In the following, if a variable ν_i^{uC} is assigned to 1, the corresponding segment of length u is meant to be covered by a codeword, i.e., it is *active*.

Next, all possible UDW permutations have to be modeled, which is important for the determination of the codeword images

that are included in the dictionary/configuration. Therefore, two variables $\nu_{\text{UDW}}^{2\text{CDW}}$ and $\nu_{\text{UDW}}^{3\text{CDW}}$ are assigned to each possible UDW permutation to keep track of the active images to be configured. One variable is not sufficient, since codewords of different lengths, i.e., length of 2 and 3 bits, have to be distinguished for the hardware requirements. Hereby, the codewords of length 1 are encoded statically since these model SBIs. A total number of $2 \cdot (2^4 + 2^8)$ variables are needed. For example, $\nu_{0001}^{2\text{CDW}} = 1$ means that the UDW '0001' will be replaced by a codeword of length 2. The specific codeword is not relevant for the calculation.

Constraints have to be generated such that the solving algorithm is able to consistently assign these variables. Overall, the problem instance including these constraints is defined by:

$$\Phi = \Phi_{\text{ME}} \wedge \Phi_{\text{uC}} \wedge \Phi_{\text{RET}} \wedge \Phi_{\#\text{CDW}}$$

Φ_{uC} realizes that all possible segments are covered to ensure the completeness, Φ_{ME} ensures that all bits of the original test data $\mathcal{I}$ are covered exactly once, $\Phi_{\#\text{CDW}}$ guarantees that the maximum number of dictionary entries is not surpassed and, finally, Φ_{RET} implements the retargeting itself.

1) Maximum number of CDWs: Hardware restrictions allow only a limited number of dictionary entries. This means that all solutions, which configure more than the allowed number, have to be excluded from the solution space. In our case, $2^2 = 4$ different entries are allowed for codewords of length 2 and $2^3 = 8$ different entries are allowed for codewords of length 3. This can be modeled as weighted constraints, i.e.,

$$\Phi_{\#\text{CDW}} = (\sum_{i=1}^{272} c \cdot x_i^{2\text{CDW}} \leq 4) \wedge (\sum_{i=1}^{272} c \cdot x_i^{3\text{CDW}} \leq 8)$$

with an equal weight $c = 1$ and x_i the positive literal representing the variable $\nu_{\text{UDW}}^{2\text{CDW}}$ and $\nu_{\text{UDW}}^{3\text{CDW}}$, respectively, for all $2^4 + 2^8 = 272$ permutations. Each *active* UDW variable, i.e., $\nu_{\text{UDW}}^{2\text{CDW}} = 1$ or $\nu_{\text{UDW}}^{3\text{CDW}} = 1$, increases the sum by 1 and the current model is only valid, if the overall weight is lower than the threshold.

Due to the modeling of these capacity limits by the weighted constraints $\Phi_{\#\text{CDW}}$, the CDWs themselves have not to be modeled explicitly. This completely avoids further huge overhead in the number of Boolean variables and clauses.

2) Equivalence: One important property between uncompressed test data $\mathcal{I}$ and compressed test data $\mathcal{D}$ is the equivalence, which has to hold after $\mathcal{D}$ is decompressed on-chip (Lemma 2). To achieve this, it is required that all bits in $\mathcal{I}$ are covered by exactly one replacement: Uncovered as well as multiple covered bits destroy the necessary equivalence. Consider the small example in Table III. It has to be ensured that each bit position is covered by a segment (active). This can be achieved by adding one clause for each bit position as follows: Given the segment variables $\nu_1^{uC}, \ldots, \nu_s^{uC}$ covering bit position i, the corresponding clause would be:

$$(x_1^{uC} \vee \cdots \vee x_s^{uC})$$

This clause is unsatisfied (and consequently the complete problem instance) when none of these segment variables is activated, i.e., the segment is not covered at all. The conjunction of these clauses is represented by Φ_{uC}. The corresponding constraints for the example in Table III are shown in Equation 1.

Furthermore, it has to be guaranteed that from the set of all segments, which cover the same bit position, only one segment is considered active. Since the mutual exclusive segments are known, implications are formulated as follows: Given a segment variable ν^{uC} and the corresponding segment variables $\nu_1^{uC}, \ldots, \nu_t^{uC}$ of conflicting segments, the following clauses have to be added:

$$(\overline{x}^{uC} \vee \overline{x}_1^{uC}) \wedge \cdots \wedge (\overline{x}^{uC} \vee \overline{x}_t^{uC})$$

These clauses are unsatisfied when more than one segment variables are active from a conflicting segment set. This has to be formulated for each assigned segment variable. The conjunction of these clauses are denoted by Φ_{ME}.

Table IV shows the set of conflicting segment variables for the example. Equation 2 shows the corresponding clauses for the mutual exclusion for the segment variable ν_0^{4C}.

3) Retargeting: Eventually, the constraints Φ_{RET} establish a link between the incoming test data $\mathcal{I}$ and the possible UDW permutations, i.e., the dictionary entries. Given is a segment of length u represented by the Boolean variable ν_i^{uC}, hence, the segment covers a bitfield between position i and $(i-1)+u$. Since the specific assignment in $\mathcal{I}$ in this segment is known, the specific UDW w is also known, which is able to cover this segment. Therefore, the segment can only be active, if w is an entry in the dictionary, i.e., the variable $\nu_w^{2\text{CDW}}$ or $\nu_w^{3\text{CDW}}$ is assigned with 1. Since only a limited number of UDW variables can be assigned with 1 (due to $\Phi_{\#\text{CDW}}$), finding the most beneficial solution is the task of the solving algorithm.

In order to encode these implications, the incoming test data $\mathcal{I}$ have to be analyzed during the problem instance generation. For each segment variable ν_i^{uC}, the binary assignment between position i and $(i-1)+u$ as well as the the corresponding specific UDW variables $\nu_{\text{UDW}}^{2\text{CDW}}$ and $\nu_{\text{UDW}}^{3\text{CDW}}$ have to be determined. The clause

$$(\overline{x}_i^{uC} \vee x_{\text{UDW}}^{2\text{CDW}} \vee x_{\text{UDW}}^{3\text{CDW}})$$

is added to Φ_{RET} in order to make sure that if the segment variable is active, either $x_{\text{UDW}}^{2\text{CDW}}$ or $x_{\text{UDW}}^{3\text{CDW}}$ is assigned with 1, i.e., a dictionary entry exists for this segment. Since single bit segments are also encoded for each bit position, it is ensured that each incoming test data stream can be processed as stated in Lemma 1.

For instance, consider the segment variable ν_0^{4C}. The binary assignment in $\mathcal{I}$ at position $[0:3]$ is assumed to be '0011'. Then, the corresponding UDW variables are $x_{0011}^{2\text{CDW}}$ and $x_{0011}^{3\text{CDW}}$.

4) Criteria for Quality: The constraints described so far restrict the solution space of the problem instance in a way that all valid configurations are part of the solution space and invalid configurations are no solutions. However, an optimization function is strictly necessary to ensure the effectiveness of the approach. If the problem is formulated as a decision problem, each valid configuration can potentially be chosen. Most likely,

TABLE III: Exemplary mapping of UDW segments

Byte index	0							
Bit index	0	1	2	3	4	5	6	7
8C	ν_0^{8C}							
4C	ν_0^{4C}				ν_4^{4C}			
4C		ν_1^{4C}						
4C			ν_2^{4C}					
4C				ν_3^{4C}				
1C	ν_0^{1C}	ν_1^{1C}	ν_2^{1C}	ν_3^{1C}	ν_4^{1C}	ν_5^{1C}	ν_6^{1C}	ν_7^{1C}

Equation 1 Φ_{uC} to ensure complete coverage of $\mathcal{I}$

$$\begin{aligned}\Phi_{\text{uC}} = &(x_0^{1C} \vee x_0^{4C} \vee x_0^{8C}) \wedge (x_7^{1C} \vee x_4^{4C} \vee x_0^{8C}) \\ &\wedge (x_1^{1C} \vee x_0^{4C} \vee x_1^{4C} \vee x_0^{8C}) \wedge (x_2^{1C} \vee x_0^{4C} \vee x_1^{4C} \vee x_2^{4C} \vee x_0^{8C}) \\ &\wedge (x_3^{1C} \vee x_0^{4C} \vee x_1^{4C} \vee x_2^{4C} \vee x_3^{4C} \vee x_0^{8C}) \\ &\wedge (x_4^{1C} \vee x_1^{4C} \vee x_2^{4C} \vee x_3^{4C} \vee x_4^{4C} \vee x_0^{8C}) \\ &\wedge (x_5^{1C} \vee x_2^{4C} \vee x_3^{4C} \vee x_4^{4C} \vee x_0^{8C}) \wedge (x_6^{1C} \vee x_3^{4C} \vee x_4^{4C} \vee x_0^{8C})\end{aligned}$$

TABLE IV: Mutual exclusions of segments (w/o single bit segments)

No.	Segment var.	Position [start:end]	Conflicting Segment var.
1	ν_0^{4C}	0:3	$\nu_1^{4C}, \nu_2^{4C}, \nu_3^{4C}, \nu_0^{8C}$
2	ν_1^{4C}	1:4	$\nu_1^{4C}, \nu_2^{4C}, \nu_3^{4C}, \nu_0^{8C}$
3	ν_2^{4C}	2:5	$\nu_1^{4C}, \nu_2^{4C}, \nu_3^{4C}, \nu_0^{8C}$
4	ν_3^{4C}	3:6	$\nu_1^{4C}, \nu_2^{4C}, \nu_3^{4C}, \nu_0^{8C}$
5	ν_4^{4C}	4:7	$\nu_1^{4C}, \nu_2^{4C}, \nu_3^{4C}, \nu_0^{8C}$
6	ν_0^{8C}	0:7	all others

Equation 2 Φ_{ME} to model mutal exclusion $\mathcal{I}$, $ME = \nu_0^{4C}$

$$\begin{aligned}\Phi_{\mathrm{ME}} =&(\bar{x}_0^{4C} \vee \bar{x}_1^{4C}) \wedge (\bar{x}_0^{4C} \vee \bar{x}_2^{4C}) \wedge (\bar{x}_0^{4C} \vee \bar{x}_3^{4C}) \\ &\wedge(\bar{x}_0^{4C} \vee \bar{x}_0^{8C}) \wedge (\bar{x}_0^{4C} \vee \bar{x}_0^{1C}) \wedge (\bar{x}_0^{4C} \vee \bar{x}_1^{1C}) \\ &\wedge(\bar{x}_0^{4C} \vee \bar{x}_2^{1C}) \wedge (\bar{x}_0^{4C} \vee \bar{x}_3^{1C})\end{aligned}$$

a solving algorithm would choose a solution, in which each bit position is covered by a single bit codeword, since this is a very easy solution to find.

Therefore, the quality of a configuration has to be encoded to guide the solving algorithm to find a cost-effective solution. The quality of a solution could be directly associated with the active segments in the data stream. However, in order to calculate the length of the compressed data stream accurately, three so called cost-variables have to be associated with each segment. Implications on the active segments and used codewords can be used to determine the specific length of the compressed stream. However, preliminary experiments have shown that the reduction of active SBIs is most important for increasing the compression ratio. For improving the run time, the optimization function was only used based on the knowledge of the segment length, i.e., using existing variables. The difference in compression ratio compared to the very accurate solution is only marginal.

The optimization function $\mathcal{F}$ is formulated over all segment variables $\nu_1^{uC}, \ldots, \nu_n^{uC}$. The weight of the variables depends on the number of bits the segment covers. Obviously, the more bits a segment covers, the better the solution and, consequently, the smaller the weight.[2] The result of $\mathcal{F}$ for each solution is the sum of all weights, whose segment variables are active.

$$\mathcal{F}(\Phi_{\mathrm{COST}}) = \sum_{i=1}^{n} m_i \cdot x_i^u \begin{cases} m_i = 4 & \text{if } u \equiv 1\mathrm{C} \\ m_i = 2 & \text{if } u \equiv 4\mathrm{C} \\ m_i = 1 & \text{if } u \equiv 8\mathrm{C} \end{cases}$$

Beside this main optimization function $\mathcal{F}$, a secondary optimization function $\mathcal{F}_2$ is utilized and considered automatically during the optimization process. This function is used to determine the length of the CDW to be used for an active UDW, i.e., whether a codeword of length 2 or 3 is used. The main idea is that UDWs which occur more often are encoded by shorter codewords, i.e., of length 2. During instance generation, a counter $c_w^{\mathcal{D}}$ is used for each possible UDW w to keep track of the number of occurrences in $\mathcal{D}$. $\mathcal{F}_2$ is built over all variables $\nu_w^{k\mathrm{CDW}}$ with $k = \{2, 3\}$.

$$\mathcal{F}_2(\Phi_{\mathrm{COST}}) = \sum_{j=1}^{272} 2 \cdot c_{w_j}^{\mathcal{D}} \cdot x_{w_j}^{2\mathrm{CDW}} + 3 \cdot c_{w_j}^{\mathcal{D}} \cdot x_{w_j}^{3\mathrm{CDW}}$$

Since $\mathcal{F}_2$ has a lower priority, the result of $\mathcal{F}$ cannot be decreased, but the length of the CDW used for each active UDW is determined, i.e., 2 or 3, such that the costs are minimized. The run time overhead of using $\mathcal{F}_2$ is negligible. As an alternative, this can be also achieved in a post-processing step.

After the solution has been found, a configuration and the corresponding sequence of CDWs can be directly extracted from the variable assignment.

[2]Please note that the solving algorithms typically perform minimization.

IV. Experimental Results

This section describes the experimental evaluation of the optimization-based retargeting approach. In particular, the final results of the developed retargeting technique is clearly distinguished against other existing techniques as well as the standardized JTAG protocol itself. Different test cases were considered for the experiments:

1) Random test data with sizes from 2048 (RTDR_2048) to 8192 (RTDR_8192) bytes (generated by a pseudo-random number generator based on Mersenne Twister).
2) Commercially representative FV test cases of the *MiBench* benchmark suite [18], which were cross-compiled for a state-of-the-art softcore microprocessor by using an optimized library for embedded systems.
3) *Golden* signature data for a JPEG encoder circuit given as input debug data following the technique of [19].

Two testbenches are utilized to simulate the test data transfer to a *Circuit-Under-Test* (CuT) for validation of the obtained results: $\mathrm{TB_{LEG}}$ implements a reference TAP controller and $\mathrm{TB_{COMPR}}$ embeds a TAP controller using a codeword-based compression technique [12], both fully compliant with IEEE 1149.1 Std. [17]. A TDR models the interface between the TAP controller and a functional core logic block and, therefore, operates as the data sink.

The $\mathrm{TB_{LEG}}$ is applied for each test case on the incoming test data $\mathcal{I}$ to determine the content of the TDR after the transmission has been completed, i.e., the golden test results. Subsequently, the compressed test data $\mathcal{D}_i$ are generated by applying the different retargeting techniques on $\mathcal{I}$. Finally, $\mathcal{D}_i$ is transferred by $\mathrm{TB_{COMPR}}$ - this TDR and the golden one determined by $\mathrm{TB_{LEG}}$ must be the same to pass the validation.

All retargeting procedures were executed on an *Intel Xeon E3-1270v3 3.5 GHz* processor with *32 GB* system memory. The implemented retargeting framework is written in *C++* and *clasp 3.1.4* is used as PBO solver [20].

The results concerning the TDV reduction and the influence on the TAT are shown in Table V and VI. These show the overall **Run time** for the test vector retargeting in CPU minutes, the **size** of TDI in bit and the achieved **TDV reduction** in % compared to legacy TAP. Column **#variables** gives the total no. of allocated variables, **#data-cycles** the number of test cycles within the data path and, finally, the achieved **TAT reduction** in % compared to legacy TAP. All required configuration steps are considered in the measured data, i.e., they are included within the numbers of data cycles and data bits.

Several experiments were conducted to show the superiority of the proposed approach: **leg**, the IEEE 1149.1 protocol without any compression at all, the **Huffman** retargeting procedure using a domain-specific Huffman algorithm, i.e., the existing hardware constraints concerning the TDR interface are considered internally and **heur** invokes the greedy retargeting procedure proposed in [12]. Finally, both techniques proposed in this work: **opt**, a retargeting procedure using formal techniques with exhaustive traversal and **opt-lim**, a modified version of **opt** limiting the computational effort to reduce the resulting run time significantly.

The results show that the proposed *opt* technique is able to improve both quality criteria – namely TDV and TAT reduction – for all test cases compared to the legacy TAP as well as to previous techniques [10], [12] significantly. Compared to the legacy TAP, the TDV can be reduced for high-entropy random data in average by 36.4% and up to 37.1%. Previous approaches achieved only an average reduction of 21.1% (Huffman) and of 26.7% [12]. The application on the debugging data or on FV data confirms these results. Here, the proposed technique is able to reduce the TDV by 47.6%. It is also shown that a

TABLE V: Benchmarks: Processing random test data & debug data considering TDV

No.	test name	run time [min]				size [bit]					data reduction [%]			
		Huffman	heur [12]	opt	opt-lim	leg	Huffman	heur [12]	opt	opt-lim	Huffman	heur [12]	opt	opt-lim
1	RTDR_2048	0.05	0.19	97.59	0.76	16384	12973	12109	10444	10739	20.8	26.1	36.3	34.5
2	RTDR_4096	0.10	0.63	190.89	14.85	32768	26005	23607	20626	21282	20.6	28.0	37.1	35.1
3	RTDR_8192	0.22	2.16	492.08	29.47	65536	51119	48529	42068	43057	22.0	26.0	35.8	34.3
4	SHA	0.06	0.10	217.10	37.49	46944	34861	28154	23271	23271	25.7	40.1	50.4	50.4
5	MATH	0.09	0.15	314.28	68.42	67616	57470	48217	33876	35220	15.0	28.7	49.9	47.9
6	DIJKSTRA	0.06	0.10	183.78	38.76	49441	34056	29172	24621	25165	31.2	40.1	50.2	49.1
7	FFT	0.08	0.14	356.07	72.11	66880	48856	39743	34034	34310	26.9	40.6	49.1	48.7
8	DEBUG_JPEG	< 0.01	0.02	30.86	2.13	5632	4134	4069	3470	3470	26.6	27.8	38.4	38.4

TABLE VI: Benchmarks: Processing random test data & debug data considering TAT

No.	test name	#Boolean variables	#data-cycles					TAT reduction [%]			
		opt/opt-lim	leg	Huffman	heur [12]	opt	opt-lim	Huffman	heur [12]	opt	opt-lim
1	RTDR_2048	58930	16389	20603	18013	16082	16378	-25.7	-9.9	1.9	1.0
2	RTDR_4096	117203	32773	41442	35236	31809	32591	-26.5	-7.5	2.9	0.5
3	RTDR_8192	233997	65568	81951	71847	64420	64494	-25.0	-9.6	1.7	1.5
4	SHA	204388	46949	51921	47090	37561	37561	-10.6	-0.3	20.0	20.0
5	MATH	286954	67621	84851	74253	57152	57264	-25.5	-9.8	15.4	15.3
6	DIJKSTRA	222101	49446	55949	49493	40533	41210	-13.2	-0.1	18.0	16.7
7	FFT	289219	66885	79393	67027	55345	55733	-18.7	-0.2	17.3	16.7
8	DEBUG_JPEG	21449	5637	6871	6125	5412	5412	-27.0	-13.2	4.0	4.0

large run time benefit can be achieved if the resources of the solving algorithm are limited (*opt-lim*), which decreases the compression ratio only slightly.

Additionally, the shortcoming of technique [12] concerning TAT reduction is successfully eliminated. Approximately 26% more data cycles are needed using the Huffman-based approach. The work in [12] needs approximately 9% more data cycles on high-entropy random data, which was due to the fact that the dictionary entries were not well chosen by [12]. Consequently, a large number of SBIs had to be used for the retargeting. This causes a large overhead in TAT. The proposed approach does not suffer this circumstance due to the use of formal optimization techniques. It is able to even reduce the data cycles by 2.2% on average and up to 2.9% for random high-entropy data and by 15% on average and up to 20.0% for the debug or FV data.

V. Conclusions & Future Work

This paper proposed an optimization-based approach for retargeting incoming test data using a codeword-based compression architecture for IEEE 1149.1-compliant TAP controllers. The problem of finding optimal codewords is modeled as a Pseudo-Boolean Optimization problem and formal solving techniques are leveraged to find beneficial codewords. Experiments have shown that the TDV as well as TAT reduction is significantly improved compared to other existing techniques by using the proposed formal techniques. In fact, this technique reduces the TDV for FV data by up to 50.4%. The TDV is even reduced for high-entropy test and debug data by up to 37.1%. Furthermore, the TAT is also reduced by up to 20.0% compared to the TAT of a legacy transfer while processing FV data.

Future work will focus on other metrics for the quality criteria, e.g., a multilevel optimization procedure may improve the results even more. For instance, the RLE encoding could be such a second-level optimization target. Furthermore, partitioning techniques for very large test data will be developed.

VI. Acknowledgment

The authors thank S. Deutsch for providing the debug data and for the interesting discussions. This work was supported by the University of Bremen's graduate school SyDe, funded by the German Excellence Initiative, by the subproject P01 'Predictive function' of the Collaborative Research Center SFB1232, funded by the German Research Foundation, by the Institutional Strategy of the University of Bremen, funded by the German Excellence Initiative and by the German Research Foundation under contract number EG 290/5-1 as well as by a Research Award from the Alexander von Humboldt Foundation, Germany.

References

[1] J. Rajski, J. Tyszer, M. Kassab, and N. Mukherjee, "Embedded deterministic test," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 5, pp. 776–792, 2004.

[2] V. Iyengar, K. Chakrabarty, and B. Murray, "Deterministic built-in pattern generation for sequential circuits," *Journal of Electronic Testing*, vol. 15, no. 1-2, pp. 97–114, 1999.

[3] L. Li and K. Chakrabarty, "Test data compression using dictionaries with fixed-length indices - SoC testing," in *VLSI Test Symp.*, 2003, pp. 219–224.

[4] A. Jas and N. Touba, "Test vector decompression via cyclical scan chains and its application to testing core-based designs," in *Int'l Test Conf.*, 1998, pp. 458–464.

[5] A. Wurtenberger, C. S. Tautermann, and S. Hellebrand, "A hybrid coding strategy for optimized test data compression," in *Int'l Test Conf.*, vol. 1, 2003, pp. 451–459.

[6] T. Kim, S. Chun, Y. Kim, M. H. Yang, and S. Kang, "An effective hybrid test data compression method using scan chain compaction and dictionary-based scheme," in *IEEE Asian Test Symp.*, 2008, pp. 151–156.

[7] S. Mitra and K. S. Kim, "XPAND: an efficient test stimulus compression technique," *IEEE Trans. on Comp.*, vol. 55, no. 2, pp. 163–173, 2006.

[8] K. Basu and P. Mishra, "Test data compression using efficient bitmask and dictionary selection methods," *Int'l Conf. on VLSI Design*, vol. 18, no. 9, pp. 1277–1286, 2010.

[9] A. Jas, J. Ghosh-Dastidar, and N. Touba, "Scan vector compression/decompression using statistical coding," in *VLSI Test Symp.*, 1999, pp. 114–120.

[10] K. Ilambharathi, G. S. N. V. V. Manik, N. Sadagopan, and B. Sivaselvan, "Domain specific hierarchical Huffman encoding," *Cornell University Library*, vol. abs/1307.0920, 2013.

[11] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. on Information Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[12] S. Huhn, S. Eggersglüß, and R. Drechsler, "VecTHOR: Low-cost compression architecture for IEEE 1149-compliant TAP controllers," in *IEEE European Test Symp.*, 2016.

[13] K. Balakrishnan and N. Touba, "Relationship between entropy and test data compression," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 386–395, 2007.

[14] A. Biere, M. Heule, H. Maaren, and T. Walsh, *Handbook of Satisfiability*, ser. Frontiers in AI and Applications. IOS Press, 2009, vol. 185.

[15] S. Eggersglüß, R. Wille, and R. Drechsler, "Improved SAT-based ATPG: More constraints, better compaction," in *Int'l Conf. on CAD*, 2013, pp. 85–90.

[16] M. Sauer, B. Becker, and I. Polian, "PHAETON: A SAT-based framework for timing-aware path sensitization," *IEEE Trans. on Comp.*, vol. PP, no. 99, pp. 1–1, 2015.

[17] "IEEE standard for test access port and boundary-scan architecture - redline," *IEEE Std 1149.1-2013 (Revision of IEEE Std 1149.1-2001) - Redline*, pp. 1–899, 2013.

[18] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *IEEE Int. Workshop on Workload Characterization, 2001.*, Dec 2001, pp. 3–14.

[19] S. Deutsch and K. Chakrabarty, "Massive signal tracing using on-chip DRAM for in-system silicon debug," in *Int'l Test Conf.*, 2014, pp. 1–10.

[20] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub, "Conflict-driven answer set solving," in *Int'l Joint Conf. on AI*, 2007, pp. 386–392.