A Codeword-based Compaction Technique for On-Chip Generated Debug Data Using Two-Stage Artificial Neural Networks

Sebastian Huhn*† Marcel Merten* Stephan Eggersglüß‡ Rolf Drechsler*†

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

†Cyber-Physical Systems, DFKI GmbH
28359 Bremen, Germany

‡Mentor Graphics Tessent®
21079 Hamburg, Germany
Stephan_Eggersgluess@mentor.com

***Abstract*—The steadily increasing complexity of state-of-the-art designs requires enhanced capabilities for post-silicon test and debug to meet the demands concerning quality as well as diagnosis. Several sophisticated techniques have been proposed in the past to address these new challenges. However, these techniques heavily enlarge the on-chip generated data that have to be stored in rarely available and highly expensive memory and, especially, to be transferred via a low-speed interface. Thus, applying data compression is a highly beneficial objective to reduce the dedicated on-chip memory and the required transfer time. This work proposes a novel compression technique, which significantly reduces the volume of on-chip generated debug data, by orchestrating a deliberately parameterized two-stage neural network. More precisely, a codeword-based compression procedure is realized, which can be directly implemented in hardware and, hence, be seamlessly integrated in an existing test/debug infrastructure. So far, it has not been possible to realize such a compression by classical (algorithmic) approaches allocating only reasonable hardware resources. First experiments already show that the proposed scheme achieves similar results as off-chip codeword-based approach being applied for incoming data.**

I. Introduction

Different breakthroughs in the field of electronic design automation have enabled an enormous increase of the complexity of *Integrated Circuits* (ICs). Modern ICs often realize a *System-on-a-Chip* (SoC), which typically contains several nested components. Due to the limited external controllability and observability of a state-of-the-art of internal signals of these components, a new challenge concerning post-silicon validation arises. Generally, this challenging circumstance is typically addressed by introducing a dedicated *Test Access Mechanism* (TAM) into the SoC. The IEEE Std. 1149.1 (JTAG) specifies such a TAM, which is widely disseminated in modern designs to provide access to the boundary pins of the embedded components. In particular, various debugging scenarios require more comprehensive mechanism to succeed.

Several enhancements [1]–[3] have been proposed in the past, which further enhance the standardized base function of JTAG following the intent of *Design-for-Debug* (DfD). These enhanced capabilities allow to accomplish a certain level of quality and diagnosis abilities. However, these functions strongly increase the resulting volume of on-chip generated data, e.g., emerged by tracing specific internal signals or by capturing certain functional registers. Due to the typical low-speed transfer via TAMs, transferring large data sets is a time-consuming and, hence, expensive task. Furthermore, the available size of dedicated memory for debugging is strictly limited on-chip.

To tackle this challenge of increasing data volume, compression techniques are applied for post-silicon debug. Typically, a trade-off exists between the desired compression ratio, the introduced hardware overhead and the loss of information (induced by lossy data compression). In this field, codeword-based approaches have proven themselves to be well-working for compressing incoming data [4]. However, these data have to be priorly processed off-chip, which is typically done by a retargeting procedure running on a workstation. Due to the strictly limited on-chip resources, codeword-based techniques have not been applicable for outgoing data as generated by the enhanced debug functions [1]–[3].

This work proposes a novel codeword-based compression technique, which enables to compress on-chip generated debug data, and which can be seamlessly integrated into an existing TAM. More precisely, the technique orchestrates a fast two-stage *Artificial Neural Network* (ANN), which significantly reduces the volume of generated debug data in-place and, thus, improve the diagnosis capabilities even more. First experiments are conducted on a large trace data set, which has been determined by observing randomly selected signals of a state-of-the-art open-source microprocessor implementation [5], and prove that the proposed scheme works in principle.

The remainder of this work is structured as follows: In Section II briefly introduces the preliminaries. The proposed two-stage ANN scheme is described in Section III and experimentally evaluated in Section IV. Finally, some conclusion are drawn and a outlook to future work is given in Section V.

II. Preliminaries

Within the last decades, the JTAG interface has been continuously improved and been adopted to address the upcoming challenges of highly complex SoC designs. For instance, the authors of [1] realize well-known software debugging mechanisms like break points. In work [2] a scheme is proposed to transfer debug data via the embedded TAM and a enhanced built-in self-test and physical layer test capabilities are presented in [6]. Beside this, commercially representative state-of-the-art micro-controllers are also equipped with certain

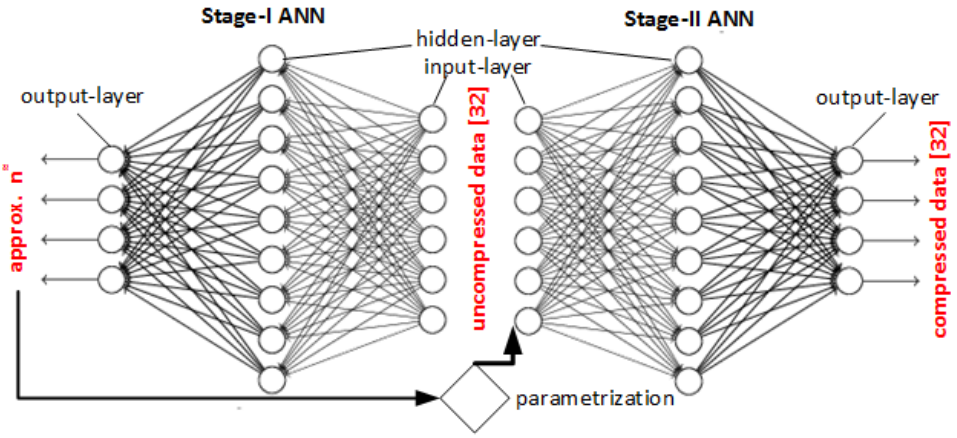

Figure 1: Proposed scheme of two-staged ANN

debug mechanism like *ARM CoreSight*™ [7] to provide a powerful debug architecture. However, these improvements also affect a strong increase in the on-chip generated debug data, which requires both, strictly limited dedicated memory and time consuming data transfer. To address these shortcomings, compression techniques have to be taken into consideration for these applications. Particularly while designing on-chip compression techniques, specific requirements (meaning hardware constraints) exist, which are discussed in work [4] and it is shown, for instance, in [8] that a codeword-based technique works well for the intended application. Such a compression technique utilizes an embedded dictionary holding N codewords $c_1, ..., c_N$ such that codeword c_i can be individually configured in $\mathcal{D}$ with specific (uncompressed) dataword u_i. Thus, the dictionary $\mathcal{D}$ realizes a mapping function Ψ with $\Psi(c_1) \rightarrow u_i$, i.e., every single codeword c_i is projected to a data word u_i. The *compressor* condenses a data stream consisting of m datawords $u_1, ..., u_m$ to a suitable sequence of n codewords $c_1, ..., c_n$ with respect to the given $\mathcal{D}$. In contrast to this, the *decompressor* restores the original data stream $u_1, ..., u_m$ equivalently by expanding every codeword $c_1, ..., c_n$.

To solve a computationally hard task within an application holding limited resources –as given here by the compressor– a common technique is to invoke an approximation function, which typically requires less resources compared to an *exact* implementation. For instance, an ANN acts as an approximation function and holds two main advantages compared to other approximation function like (classical) regression methods: At first, an ANN can cope with non-linear relationships, which prevail in the application and, secondly, the ANN can be implemented by a little among of a-priori known data. Such an ANN consists of multiple neurons (nodes), which are clustered in different hierarchical layers: The input-, output- and a variable number of hidden-layers. All links (edges) between neurons hold a certain weight, which is individually adjusted in a training-phase - here based on a supervised learning procedure, i.e., every training sample is labeled with the expected value.

III. Scheme of Two-Stage Neural Networks

A two-stage ANN scheme is orchestrated to implement the required compressor, which is order of magnitudes harder to realize compared to the decompressor (as described and implemented in [9]). Thus, this proposed scheme implements an approximation function, which works well for data with non-linearly relationships as clearly given by the given application.

In Figure 1 the overall flow is shown, which is designed as follows: The Stage-I ANN calculates the lower boundary of the approximated number of codewords $n^{\approx}$, which have to be at least used to compress the given data. Currently, the proposed scheme holds 32 nodes in the input-layer, i.e., 32 bits are simultaneously compressed and, especially, the determined $n^{\approx}$ emitted by the output-layer. Internally, a multi-class loss layer is applied to compute the single gradient value. For the remainder, three different classes are distinguished: a) exact matches $n^{\approx} = n$, b) non-critical overestimation $n^{\approx} > n$ and c) critical underestimation $n^{\approx} < n$. In fact, c) has to be avoided, otherwise the Stage-II processing can be impacted adversely. Finally, the Stage-II ANN determines the actual sequence of codewords with respect to the implicitly encoded dictionary $\mathcal{D}$, whereas it also processes the determined $n^{\approx}$. Evaluating $n^{\approx}$, beneficially affects the actual compression, which is due to the fact that optimal parameters can be selected for the adoption of this ANN. To the end, the codewords $c_1, .., c_n$ –representing the compressed data– are emitted at the output-layer's nodes.

IV. Experimental Evaluation

This work focuses on the evaluation of Stage-I, which determines n –the number of required codewords (with respect to the given dictionary $\mathcal{D}$, which is assumed to be static) for an arbitrarily given uncompressed data stream, which is mostly due to the page limitation.

All experiments were executed on an *Intel i5-6200U 2.8 GHz* with *8 GB* system memory within a *C++* software-environment, which uses the *dlib 19.7* [10] as a library to implement ANNs. The realized ANN can be transformed to a digital hardware implementation by following the proposed scheme of [11]. The considered training as well as validation data sets have been generated by a randomly investigated set of 32 signals of a state-of-the-art open-source microprocessor implementation [5]. These random data are characterized by a very high entropy, so that the lower bound for compression ratio should be determined [12]. In particular, the ANN is trained with a set of $64K$ samples, i.e., a training rate of 1.5×10^{-5}

is achieved, and the following experimental evaluation is done on basis of a disjoint validation set holding $64K$ samples as well. The labels –representing n– for the training data set are calculated by applying the technique of [4] and hold an average size of 14.6.

The experiments have been conducted for two fully-connected Stage-I ANN_2 holding 2 hidden-layers and ANN_4 holding 4 hidden-layers, respectively. The training phase requires 7559 sec. (20286 sec.) and the validation phase requires <1 sec. (<1 sec.) for ANN_2 (ANN_4). The classification ratios in % for matches : overestimations : underestimations are distributed as follows 68.9 : 15.5 : 15.6 (73.5 : 12.0 : 14.5) for ANN_2 (ANN_4) In fact, the validation time is negligible for both ANN types, the training time of ANN_4 is 2.7 times higher due to the increased number of hidden-layers, and accordingly the larger network size. However, introducing two additional layer increases the performance, i.e., higher ratio of matches and, thus, lower ratio of over- as well as underestimations.

In case of such an overestimation, the compression ratio is reduced. More precisely, the Stage-II ANN does not reach the optimum (highest possible compression ratio), which is due to the incorrect paramitrization by Stage-I. Although the data content is not affected at all. In contrast to this, an occurring underestimation can lead to a (partial) data corruption. This partial corruption is completely negligible for different debug applications as long as a certain boundary is not exceeded, e.g., some corrupted signals in a very small among of the overall debug traces does not affect the application. In case, however, the corruption has to be completely avoided, a safety margin can be added to the Stage-I prediction value or a recalculation of the compressed data can be induced, if a certain level of confidence is undershot.

Figure 2 presents the final results as a heat-map for ANN_4 while holding the single validation sample at the x-axis, the (relative) distance $n^{\approx} - n$ to the expected value on the y-axis and the (absolute) expected value n on the z-axis used for colorizing. For the ANN_4, the data show clearly that the classification works very well for $\approx 88\%$ of all investigated data, which have not been a-priori known: The standard error of the calculation is just about 0.04. Even in case of an underestimation – as occurring in 14.5% of the validations – the standard error is 0.06, whose adverse impact on the final compression ratio is manageable.

V. Conclusions

This paper presents a novel scheme orchestrating a two-stage ANN to implement a (codeword-based) compressor, which can be introduced to modern SoC designs. To the end, the proposed approach allows to compress on-chip generated debug data to, finally, reduce the size of required memory and accelerate the download of on-chip generated data. Future work will focus on the further evaluation of Stage-II and, additionally, on improvements of Stage-I such that the ratio of underestimations is reduced. In particular, additional boosting techniques will be evaluated for improving the prediction quality of Stage-I ANN to, finally, address candidates, which lead to underestimations.

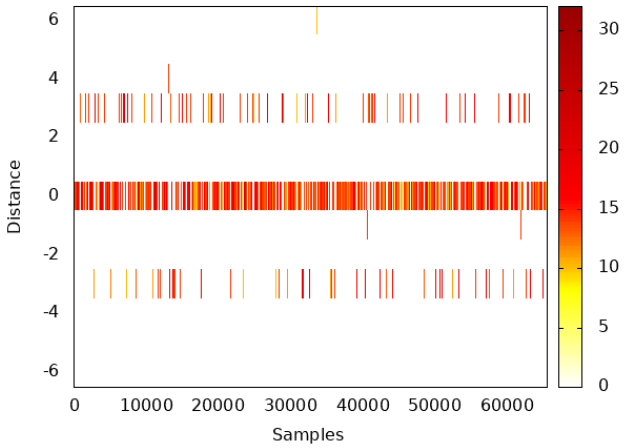

Figure 2: Stage-I validation heat-map for ANN_4

VI. Acknowledgment

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.

References

[1] Y. Liu, W. h. Wu, X. f. Zhou, and D. Zhou, "A novel on-chip debug system with quick all-registers scan chain based on JTAG," in *International Conference on Solid-State and Integrated Circuit Technology*, 2006, pp. 1941–1943.

[2] X. Liu and Q. Xu, "On reusing test access mechanisms for debug data transfer in SoC post-silicon validation," in *IEEE Asian Test Symp.*, 2008, pp. 303–308.

[3] L. van de Logt, F. van der Heyden, and T. Waayers, "An extension to JTAG for at-speed debug on a system," in *International Test Conference*, vol. 2, 2003, pp. 123–130 Vol.2.

[4] S. Huhn, S. Eggersglüß, and R. Drechsler, "VecTHOR: Low-cost compression architecture for IEEE 1149-compliant TAP controllers," in *IEEE European Test Symp.*, 2016, pp. 1–6.

[5] D. Lampret, "OpenRISC-1000 SoC," 2003, http://opencores.org/project,jtag.

[6] G. Jian-min and L. De-lin, "A functional enhancement methodology to JTAG controller in complex SoC," in *International Conference on Computer Science Education*, 2009, pp. 1128–1131.

[7] A. Limited, "CoreSight™ components technical reference manual," 2009.

[8] A. Wurtenberger, C. Tautermann, and S. Hellebrand, "Data compression for multiple scan chains using dictionaries with corrections," in *International Test Conference*, 2004, pp. 926–935.

[9] S. Huhn, S. Eggersglüß, K. Chakrabarty, and R. Drechsler, "Optimization of retargeting for IEEE 1149.1 TAP controllers with embedded compression," in *Design, Automation and Test in Europe*, 2017, pp. 578–583.

[10] D. E. King, "Dlib-ml: A machine learning toolkit," *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.

[11] H. Faiedh, Z. Gafsi, K. Torki, and K. Besbes, "Digital hardware implementation of a neural network used for classification," in *International Conference on Microelectronics*, 2004, pp. 551–554.

[12] K. Balakrishnan and N. Touba, "Relationship between entropy and test data compression," *IEEE Transaction on CAD of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 386–395, 2007.