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

Marcel Merten* Sebastian Huhn*† Rolf Drechsler*†

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

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

***Abstract*—The steadily increasing complexity of state-of-the-art designs requires enhanced capabilities for post-silicon test and debug. By this, the demands concerning the quality as well as the diagnostic capability are met. 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 has 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 into an existing test/debug infrastructure. So far, it has not been possible to realize such compression by classical (algorithmic) approaches allocating only reasonable hardware resources. The experimental evaluation validates that the proposed scheme achieves similar results as the off-chip codeword-based approach being applied for incoming test 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 internal signals of these components, a new challenge concerning post-silicon validation arises. This challenging circumstance is typically addressed by introducing a dedicated *Test Access Mechanism* (TAM) into the SoC. The IEEE Std. 1149.1 (JTAG) [1] specifies such a TAM, which is widely disseminated in modern designs to provide access to the boundary pins of the embedded components. In addition to JTAG, a further IEEE Std. 1687 (IJTAG) [2] has been proposed for creating even large and complex on-chip test networks that, among others, take advantage of reconfigurable scan paths. By this, a new dimension of possibilities is opened up to, for instance, introduce several debug and diagnosis instruments on-chip.

Various debugging scenarios require more comprehensive mechanisms to succeed. Several enhancements [3]–[5] have been proposed in the past, which further enhance the standardized base function of JTAG/IJTAG following the intent of *Design-for-Debug* (DfD). These enhanced measures allow accomplishing a certain level of quality and diagnostic capabilities. However, these mechanisms strongly increase the resulting volume of on-chip-generated data, e.g., by tracing specific internal signals or 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 the dedicated debugging memory is strictly limited on-chip.

978-1-6654-1609-2/21/$31.00 ©2021 IEEE

Compression techniques are applied for post-silicon debug to tackle the challenge of increasing data volume. Typically, a trade-off exists between the desired compression ratio, the introduced hardware overhead, and the information loss (induced by lossy data compression algorithms). In this field, codeword-based approaches have proven themselves to be well-working for compressing incoming data [6], [7]. 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 [3]–[5].

This work proposes a novel codeword-based compactor that enables the compression of on-chip generated debug data while being seamlessly integrated into any existing TAM. The technique orchestrates a fast two-stage *Artificial Neural Network* (ANN), which significantly reduces the volume of generated debug data in place and, thus, improves the diagnostic capabilities even more.

Various experiments are conducted on a large trace data set, which has been determined by (a) observing randomly selected signals of a state-of-the-art open-source microprocessor implementation [8] and by (b) randomly generated data of functional registers. The results clearly prove that the proposed ANN scheme allows reducing the resulting data volume significantly by up to 30%.

The remainder of this work is structured as follows: Section II briefly introduces the preliminaries of this work. Furthermore, Section III describes the two-stage ANN scheme in detail, and the experimental evaluation is presented in Section IV. Finally, some conclusions are drawn and an outlook to future work is given in Section V.

II. Preliminaries

This section introduces different on-chip data compression techniques and gives a brief overview of ANNs as approximation functions, which is important for comprehending this work.

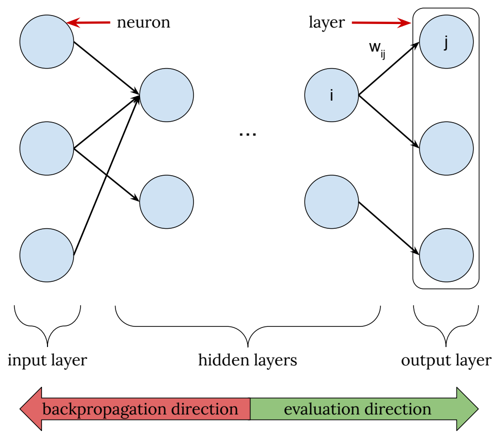


Figure 1: Scheme of an ANN

A. *Compression*

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 [3] realize well-known software debugging mechanisms like breakpoints. In work [4], a scheme is proposed to transfer debug data via the embedded TAM and an enhanced built-in self-test and physical layer test capabilities are presented in [9]. Besides this, commercially representative state-of-the-art micro-controllers are also equipped with certain debug mechanisms like *ARM CoreSight*™ [10] to provide a powerful debug architecture. However, these improvements substantially increase in the on-chip generated debug data requiring 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.

When designing on-chip compression techniques, specific requirements have to be taken into account, meaning hardware constraints, as explained in [6]. Further works, like [11], clearly demonstrate that a codeword-based technique works well for the intended application of this work. Generally, a codeword-based 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 *compactor* condenses a data stream consisting of m datawords $u_1, ..., u_m$ to a suitable sequence of n codewords $c_1, ..., c_n$ for the given $\mathcal{D}$. In contrast to this, the *decompressor* restores the original data stream $u_1, ..., u_m$ equivalently by expanding every codeword $c_1, ..., c_n$.

B. *Artificial Neural Networks*

A common strategy to solve a computationally intensive task in environments with limited resources is about invoking an approximation function. Such an approximation function requires substantially fewer resources compared to an algorithm that solves the task *exactly*. A prominent candidate for an approximation function is an ANN. Such an ANN is modeled as a set of layers, whereby each consists of neurons, as exemplary given in Figure 1. The layers are ordered in the direction of the evaluation, beginning with the input layer, followed by an arbitrary number of hidden layers, and ending with the output layer. Typically, the number of input nodes corresponds to the input data size, while the number of output nodes corresponds to the number of classes to be predicted. Every neuron i can have a connection with a connection-weight w_{ij} to any other neuron j. During the evaluation, the layers will be evaluated in the predefined order, whereby the input of each neuron j is $\sum_i (w_{ij} * o_i)$ where o_i is the output of the predecessor neuron i. A neuron's input is passed to an activation function ϕ to calculate the output of the neuron. In order to learn non-linear relationships, a non-linear activation function can be applied to each of the neurons.

For the ANN training, the calculated outputs are used to determine the error using a loss function and the expected output. An essential loss function is the cross-entropy orchestrating the *Mean Squared Error* (MSE). The multiclass cross-entropy is defined in Equation (1), whereby n is the number of inputs, o_i is the output of the ANN of output-neuron i, and $\overline{o_i}$ is the expected output of output-neuron i. The loss is used to evaluate the ANN's ability to solve the classification problem.

$$\sum^{n} (\sum_{i=1}^{c} (\frac{1}{n} * \overline{o_i} * -ln(o_i))) \tag{1}$$

A commonly used method to determine error-minimizing adjustments of the connecting weights is the gradient descent. Thereby, an error gradient is computed for the output layer as specified in Equation (2). Afterward, the error gradient is backpropagated through the ANN.

$$gradient_i = (o_i - \overline{o_i}) \tag{2}$$

In comparison to other approximation functions like (classical) regression methods, ANNs hold two major advantages as follows: At first, an ANN can cope with non-linear relationships, which prevail in the application and, secondly, the ANN can be implemented by a small a-priori known data set. Such an ANN consists of multiple neurons (nodes) clustered in different hierarchical layers: the input layer, the output layer, and a variable number of hidden layers. All links (edges) between neurons hold a certain weight, which is individually adjusted during the training. Here, the training is assumed to be a supervised learning procedure, i.e., every training sample is labeled with the expected value. In fact, different types of ANNs have been frequently used in various applications and, hence, form an established machine learning concept to solve even highly complex classification problems.

III. Scheme of Two-Stage Architecture

This work proposes a two-stage ANN scheme realizing the compactor of a codeword-based compression technique. In the end, this compactor significantly reduces on-chip generated debug data significantly enhancing the debug capabilities of state-of-the-art SoCs. Generally, the proposed scheme implements an approximation function, which works well for data with even non-linear relationships, as clearly given by the application.

The ANNs of the proposed scheme are meant to be trained off-chip in a classical software-based fashion while orchestrating a newly developed and application-specific loss function.

TABLE I: Necessary codewords

Codeword-type	uncompressed codeword	compressed codeword
SBI	1	1
SBI	0	0
ER	/	/

After the training's completion, the learned ANN parameters are extracted from the software model and the corresponding ANN structure is meant to be implemented in hardware on-chip. Since the training is not performed on-chip, the training capabilities are not implemented later on-chip, which saves hardware resources.

Figure 2a presents the integration of the compactor into the usual debug data evaluation flow driven by a workstation. The preparation of the compactor is realized in software by training the ANN. The fully trained compactor then performs the codeword-based compression purely on-chip. Figure 2b shows the overall scheme of a two-staged ANN. At first, Stage-I evaluates the uncompressed data to determine the set of parameters for Stage-II. Finally, Stage-II performs the compression of the uncompressed data using the specific parameters. It has to be noted that the uncompressed data are divided into compressed sequences (comp-seqs) to meet the input layer size constraints, particularly when compressing large uncompressed data streams. Thus, the proposed compression scheme processes a comp-seq at a time.

Due to the nature of a codeword-based compression technique, the choice of codewords stored in the dictionary affects the resulting compression efficacy. Consequently, it is of utmost importance to select the codewords deliberately. In order to ensure the completeness of codeword-based compression techniques, so-called *Single-Bit-Injections* (SBIs) are orchestrated covering the single bit replacements '0' and '1', respectively [6]. Even though no compression is achieved by an SBI locally, this principle ensures that arbitrary bit sequences, meaning uncompressed data, can be compressed. For that reason, each dictionary $\mathcal{D}$ holds two special codewords for representing the SBIs. A third special codeword is required when an ANN is used to realize the codeword-based compression technique called *Empty Replacement* (ER). The special codewords are summarized in Table I.

A. Stage-I

The Stage-I ANN calculates the lower bound of the approximated number of codewords $n^{\approx}$. More precisely, $n^{\approx}$ describes how many codewords have to be at least used while compressing the given data sequence. This metric reflects certain characteristics of the data sequence in combination with the considered dictionary. For each value $n^{\approx}$, as predicted by Stage-I, an individual set of parameters (with different weights) is used during the off-chip training of Stage-II. By this, the Stage-II is well aligned for different kinds of data.

The input layer size of Stage-I equals the comp-seq's size. The size of the output layer determining $n^{\approx}$ equals the maximum number of codewords. Consequently, the output layer's size equals the input layer size when only SBIs are considered. Internally, a multi-class loss layer is applied to compute the single gradient value for Stage-I.

For the remainder of this work, three different classes are being distinguished:

a) exact matches $n^{\approx} = n$
b) non-critical overestimations $n^{\approx} > n$
c) critical underestimations $n^{\approx} < n$

In fact, c) has to be avoided since, otherwise, the Stage-II can be impacted adversely. Thus, an optionally adapted gradient's calculation is implemented. For each class i in Stage-I, the gradient is adapted as follows, whereby in case c) a constant punishment-value (cpv) is added:

$$gradient_i = \begin{cases} (o_i - 1) & \text{if a)} \\ (o_i) & \text{if b)} \\ (o_i + cpv) & \text{if c)} \end{cases} \tag{3}$$

B. Stage-II

The Stage-II ANN determines the actual sequence of codewords with respect to the implicitly encoded dictionary $\mathcal{D}$. The weights of the Stage-II ANN are selected considering $n^{\approx}$. The evaluation of $n^{\approx}$ beneficially affects the compression efficacy since the 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.

In order to further improve the resulting compression efficacy, five enhancements have been developed, which are discussed in the remainder of this section.

a) Application-specific Loss Function: The first improvement concerns an application-specific loss function that is being used during the off-chip training phase of the ANN. This function computes the loss only by using the specified $\mathcal{D}$ instead of learning by computing the loss with respect to expected results, as shown in Figure 3. The codewords are rated by the compression ratio (c_{ratio}), defining the expected result of a neuron with respect to the proportion of the compression for all fitting codewords. Given N the set of appropriate codewords, e_i the i-th dictionary entry with $e_i \in N$ and $\beta(e_i)$ the compression of e_i, the computation of the c-ratio is shown in Equation (4). Here, $\beta(e_i)$ is set to a fixed value β_{SBI} (for each SBI), yielding a higher reward for SBIs than inappropriate codewords.

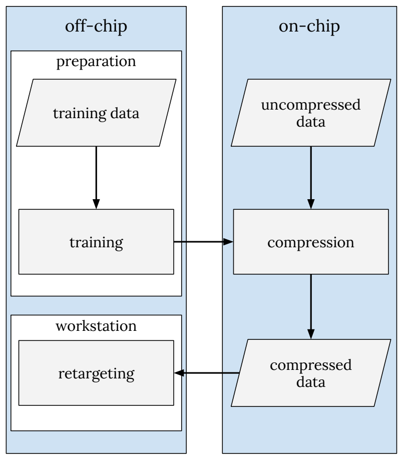
$$c_{ratio} = \frac{\beta(e_i)}{\sum_{e_x \in N}(\beta(e_x) + \beta_{SBI})} \tag{4}$$

b) Reward Boosting: Multi-objective optimization is used that initially differentiates in two cases and if a codeword does not fit, any non-zero output forms an error as follows:

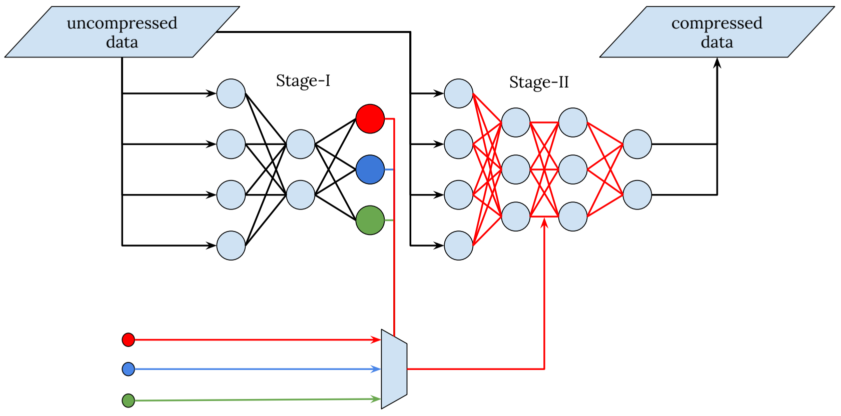
$$gradient_i = \begin{cases} (o_i) & \text{if codeword does not fit} \\ (o_i - c_{ratio}) & \text{if codeword does fit} \end{cases} \tag{5}$$

This equation considers the achieved compression of a codeword for minimizing the size of the compressed data.

If a dictionary $\mathcal{D}$ provides multiple possible replacements for one uncompressed data sequence, a reward boosting increases the compression. The idea is about adapting the gradient to prefer codewords that compress the input data instead of introducing SBIs. If the Stage-II choose an SBI for a certain bit position, possible non-SBI replacements are rewarded, as shown in Equation (6),



(a) Training and evaluation procedure



(b) Two-stage topology

Figure 2: Proposed scheme of the two-stage ANN

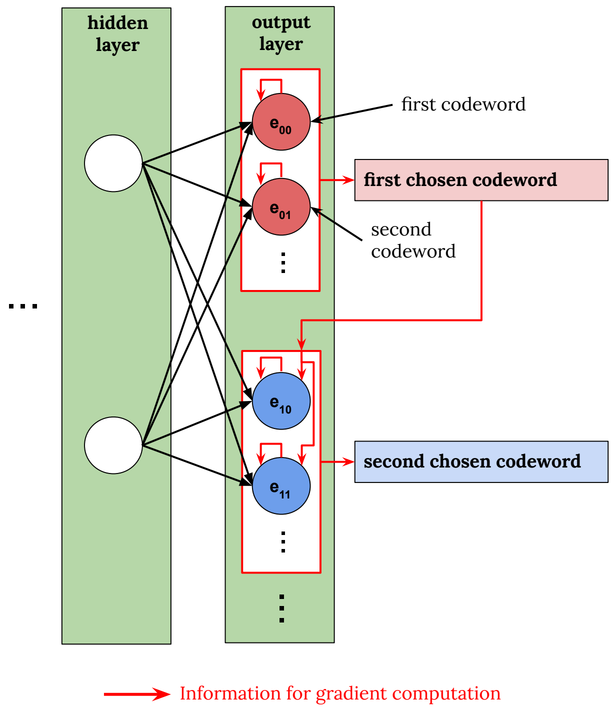


Figure 3: Stage-II - gradient's calculation structure

$$gradient_i = (gradient_i - c_{ratio}) \tag{6}$$

c) Punishment Boosting: A specialized punishment boosting technique has been developed to minimize the output of inappropriate codewords. Codewords that are not suitable, but hold higher prediction values than the matching codewords, are treated with an additional punishment depending on the compression β, as shown in Equation (7).

$$gradient_i = \begin{cases} (gradient_i + \frac{1}{\beta(e_x)}) & \text{if } \beta(e_x) > 0 \\ (gradient_i + \beta_{SBI}) & \text{else} \end{cases} \tag{7}$$

d) Pivot-Split: An (optional) pivot split technique is implemented, which splits one input in two equally sized input chunks. This allows to halve the input neurons and, by this, decreases the resulting classification problem's complexity. Furthermore, an optional split was implemented that divides one input into two equally sized inputs. Otherwise, the maximal compression is decreased if the input is too small.

e) Switching Activity: The last improvement concerns the *Switching Activity* (SWA) of the uncompressed data. The SWA provides further information about the characteristics of the data sequence, which can affect the codewords' usability. Consequently, the SWA is incorporated as a parametrization of Stage-II in addition to Stage-I.

IV. Experimental Evaluation

This section describes the experimental evaluation of the proposed codeword-based compression technique orchestrating a two-stage neural network and discusses the obtained results concerning the superiority of the proposed enhancements of the basic ANNs' structure.

All experiments have been conducted on an *AMD Ryzen 7 2700X* with *16 GB* system memory within a *C++* software environment using the *dlib 19.7* [12] library for ANNs. The proposed ANNs scheme is meant to be transformed to a hardware implementation by following the technique [13]. During the last years, a lot of research work has been spent on the hardware realization of ANNs since they have proven themselves as fast and accurate classification instruments [14]. In particular, off-chip learning approaches benefit from the more accurate training on a software basis [15]. In this work, the resulting hardware overhead has been considered as an essential factor during the design phase of the proposed scheme and, hence, only small ANN topologies have been invoked. The different kinds of Stage-I ANNs, as introduced later, consist of 84 to 128 hidden neurons per hidden layer. The ANN of Stage-II has one hidden layer with 240 hidden nodes and 240 output neurons, and 16 input neurons.

The considered training and validation data sets have been generated by a randomly investigated set of 16 signals of a state-of-the-art open-source microprocessor implementation of an *OpenRISC 1000 SoC* [8] while executing functional verifi-

TABLE II: Stage-I results

ANN / values	ANN_{I-1}	ANN_{I-2}	ANN_{I-3}
training phase	22.3 s	42.3 s	22.7 s
validation phase	<1 s	<1 s	<1 s
matches	90.1 %	84.8 %	71.0 %
overestimations	5.6 %	7.4 %	28.1 %
underestimations	4.3 %	7.8 %	0.9 %

TABLE III: Stage-II results

values	Huffman [18]
compression ratio	28.2%
correct bits	100.0%
tradeoff	28.2%

cation benchmarks from the *Dhrystone* benchmark suite [16]. These random data are characterized by a very high entropy such that the lower bound for compression ratio should be determined [17]. Furthermore, an equally sized set of on-chip generated debug data has been evaluated. The ANN is trained with a set of 768 samples, i.e., a training rate of 1.5×10^{-5} is achieved, and the following experimental evaluation is done on the basis of a disjoint validation set holding 768 samples. The labels – representing $n^{\approx}$ – for the training data set are calculated by applying the technique of work [18].

The comp-seq size is set to 16 bits and, hence, the input- and output layer sizes of Stage-I are 16 as well. The ANN$_{I-1}$ holds one fully-connected hidden layer with 84 hidden neurons, and ANN$_{I-2}$ holding three fully-connected hidden layers with 84, 128 and 84 hidden neurons. ANN$_{I-3}$ has the same topology as ANN$_{I-1}$ but uses the adapted gradient's calculation. The respective results for the random data are shown in Table II.

The validation time is negligible for all ANN types. The training time of ANN$_{I-2}$ is 1.9 times higher than of ANN$_{I-1}$ due to the increased number of hidden layers and, consequently, the larger network size. It has to be noted that the introduction of two additional layers does not increase the accuracy. In case of an overestimation, the compression ratio is reduced since the Stage-II ANN misses the optimum (meaning highest possible compression ratio) due to the incorrect parameterization by Stage-I. Even though the data content will not be affected in such a case. In return, an occurring underestimation can potentially lead to (partial) data corruption. This partial corruption is completely negligible for different debug applications as long as a certain bound is not exceeded, e.g., some corrupted signals in the overall debug traces do 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.

ANN$_{I-1}$ and ANN$_{I-2}$ perform a *classical* calculation of the gradient. Accordingly, over- and underestimations are equally distributed. Using the adapted gradient's calculation in ANN$_{I-3}$ impacts the results by drastically reducing the number of underestimations. However, the adapted gradient's calculation leads to significantly more overestimations meaning that the overall accuracy is decreased. As mentioned, the compression ratio is reduced when overestimations occur.

Figure 4 presents the final results as a heat map for ANN$_{I-1}$ while the single validation sample is given at the x-axis, the (relative) distance $n^{\approx} - n$ to the expected value at the y-axis, and the (absolute) expected value n at the z-axis, as used for dyeing. For ANN$_{I-1}$, the obtained results prove

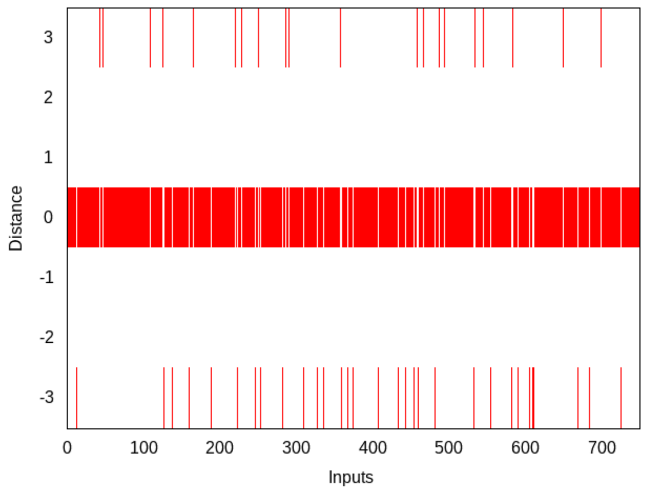


Figure 4: Stage-I validation heat map for ANN$_{I-1}$

that the classification works excellent for approx. 96% of all investigated data sets, which have not been a-priori known: The standard error of the calculation is just about 0.04, and all missclassifications miss the expected value by 3. Even in case of an underestimation – as occurring in 4.3% of the validations – the standard error is 0.11, whose adverse impact on the final compression ratio is manageable.

In order to have a baseline result, the ANN$_{II-1}$ is used for codeword-compression without the adapted loss function and without the approximation of Stage-I. Afterward, it has been evaluated using ANN_{I-1} for parametrization with all combinations of the described optional features for Stage-II. Even though all results exceed the baseline, the best predictions were achieved with a combination of reward boosting (ANN$_{II-2}$) and using reward boosting with SWA-supported parametrization (ANN$_{II-3}$). Thereby the total number of parameter-sets is three by only using Stage-I for parametrization and 13 with an additional parametrization by the SWA.

The results are evaluated considering different ratios as follows: At first, the compression ratio describes the percentage of saved bits of the compressed data compared to the uncompressed data. Secondly, the value correct bits measures the number of matching bits by comparing the original uncompressed data with the uncompressed data after compression. Finally, a tradeoff of both values shows the effective compression ratio considering the correct bits and is computed by $(compressionrate) - (correctbits) - 1$. The reference values are calculated using the word-level Huffman Encoding approach of [18], as presented in Table III. The results emphazise the great potential compression with more complex off-chip compaction techniques.

Table IV presents the classification ratios in $\%$ for compression, correct bits, and the tradeoff for the random data. The deviation of the training time of the slightly modified setups increases with the number of trained sets of parameters. However, ANN$_{II-1}$ has the least correct bits and the lowest compression efficacy. In contrast to ANN$_{II-3}$, ANN$_{II-2}$ achieves a higher compression but less correct bits. The most appropriate configuration depends on the intended use case and can be adapted to the designer's choice.

Since highly entropic random data does not necessarily correspond to regular on-chip generated trace/debug data, the ANNs are also evaluated on a further data set, which holds

TABLE IV: Stage-II results

ANN / values	ANN_{II-1}	ANN_{II-2}	ANN_{II-3}
training phase	45.1 s	295.1 s	814.7 s
validation phase	< 1 s	< 1 s	< 1 s
compression ratio	35.1%	44.7%	27.3%
correct bits	57.3%	86,4%	91.5%
tradeoff	-7,6%	31.1%	18.8%
parameter-sets	1	3	13

TABLE V: Stage-II results on debug data

ANN / Values	ANN_{II-1}	ANN_{II-2}	ANN_{II-3}
training phase	42.9 s	284.1 s	839.2 s
validation phase	< 1 s	< 1 s	< 1 s
compression ratio	37.8%	41.9%	30,9%
correct bits	59.5%	87.9%	94.5%
tradeoff	-2.7%	29.8%	25.4%
parameter-sets	1	3	13

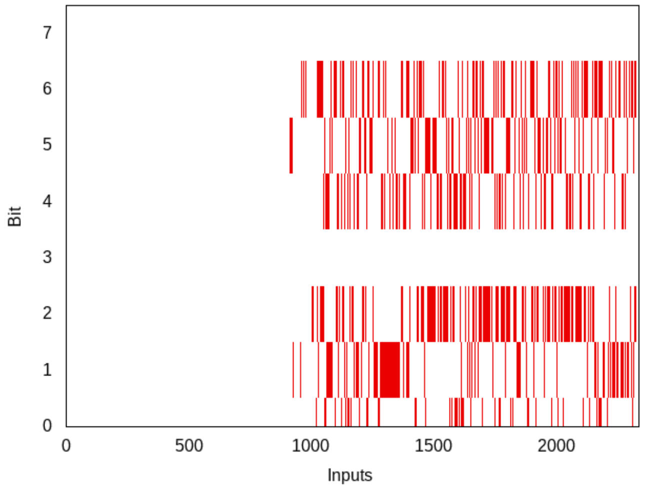


Figure 5: Stage-II validation heat map for ANN$_{II-2}$

less entropy and has been omitted by tracing single functional registers. These data are more continuous than the initial considered random data. Consequently, the compression efficacy increases for this less-entropic data, as presented in Table V.

Figure 5 (6) presents the final results as a heat map for ANN$_{II-2}$ (ANN$_{II-3}$) while given the single validation sample at the x-axis, the bits of the predicted uncompressed data at the y-axis, and the bit errors at the z-axis. The red marks indicate a bit error after decompressing. Bits that exceed the expected number of bits in the uncompressed data can be ignored because the size of the uncompressed sequence is a-priori known. For the ANN$_{II-2}$, the data show that many bit errors are equally distributed in the bits 0-2 and 4-6 bit position ranges. ANN$_{II-3}$ shows fewer bit errors and many of them are caused by underestimations of the uncompressed data length.

V. Conclusions

This paper presented a novel scheme orchestrating a two-stage ANN to implement a (codeword-based) compactor, which can be introduced to modern SoC designs. To the end, the proposed approach allowed to compress random data as well as on-chip generated debug data to, finally, reduce the size of required memory and accelerate the transfer of on-chip generated data by up to 30%, which was competitive to the word-level Huffman Encoding approach. Future work will focus on the orchestration of neuro-evolutionary algorithms

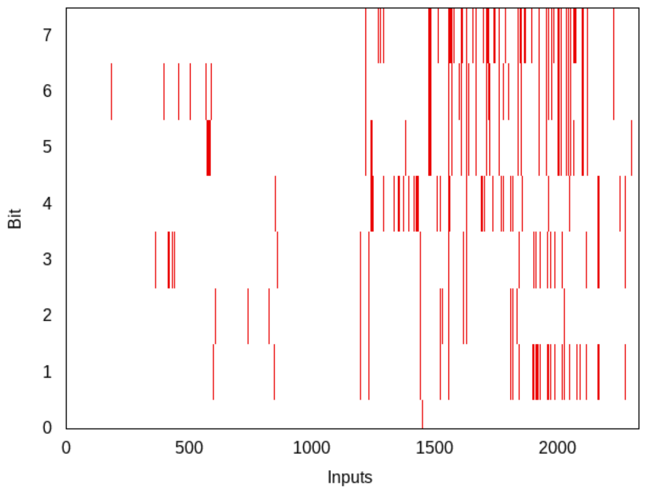


Figure 6: Stage-II validation heat map for ANN$_{II-3}$

on top of the proposed ANN scheme to further improve the resulting correctness of the compressed data such that the trade-off is being increased even more.

VI. Acknowledgements

This work was supported by the AI initiative of the Free Hanseatic City of Bremen.

References

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

[2] "IEEE standard for access and control of instrumentation embedded within a semiconductor device," *IEEE Std. 1687-2014 - Redline*, pp. 1–283, 2014.

[3] 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.

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

[5] 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.

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

[7] 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.

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

[9] 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.

[10] A. Limited, "CoreSight^{TM} components technical reference manual," 2009.

[11] 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.

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

[13] 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.

[14] N. Botros and M. Abdul-Aziz, "Hardware implementation of an artificial neural network," in *IEEE International Conference on Neural Networks*, 1993, pp. 1252–1257 vol.3.

[15] M. Forssell, "Hardware implementation of artificial neural networks," Technical report, Carnegie Mellon University, Department of Electrical and Computer Engineering, Tech. Rep., 2014.

[16] R. P. Weicker, "Dhrystone: A synthetic systems programming benchmark," *Commun. ACM*, vol. 27, no. 10, p. 1013–1030, Oct. 1984. [Online]. Available: https://doi.org/10.1145/358274.358283

[17] 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.

[18] K. Ilambharathi, G. Manik, N. Sadagopan, and B. Sivaselvan, "Domain specific hierarchical Huffman encoding," *ArXiv*, vol. abs/1307.0920, 2013.