A Neural-Network-Based Fault Classifier

Rodríguez Gómez, Laura; Wunderlich, Hans-Joachim

Proceedings of the 25th IEEE Asian Test Symposium (ATS'16) Hiroshima, Japan, 21-24 November 2016

doi: http://dx.doi.org/10.1109/ATS.2016.46

Abstract: In order to reduce the number of defective parts and increase yield, especially in early stages of production, systematic defects must be identified and corrected as soon as possible. This paper presents a technique to move defect classification to the earliest phase of volume testing without any special diagnostic test patterns. A neural-network-based fault classifier is described, which is able to raise a warning, if the frequency of certain defect mechanisms increases. Only in this case more sophisticated diagnostic patterns or the even more expensive physical failure analysis have to be applied. The fault classification method presented here is able to extract underlying fault types with high confidence by identifying relevant features from the circuit topology and from logic simulation.

Preprint

A Neural-Network-Based Fault Classifier

Laura Rodríguez Gómez and Hans-Joachim Wunderlich

Institute of Computer Architecture and Computer Engineering, University of Stuttgart
Pfaffenwaldring 47, D-70569, Germany, Email: {laura.rodriguez, wu}@informatik.uni-stuttgart.de

***Abstract*—In order to reduce the number of defective parts and increase yield, especially in early stages of production, systematic defects must be identified and corrected as soon as possible. This paper presents a technique to move defect classification to the earliest phase of volume testing without any special diagnostic test patterns. A neural-network-based fault classifier is described, which is able to raise a warning, if the frequency of certain defect mechanisms increases. Only in this case more sophisticated diagnostic patterns or the even more expensive physical failure analysis have to be applied. The fault classification method presented here is able to extract underlying fault types with high confidence by identifying relevant features from the circuit topology and from logic simulation.**

***Index Terms*—Neural networks, machine learning, fault classification, diagnosis**

I. Introduction

Manufacturing monitoring includes an early identification of systematic defects in order to correct them as soon as possible and to ensure or increase yield [1]. Detecting the problem requires the generation of high quality test sets, while identifying and correcting the problem requires sophisticated logic diagnosis, physical analysis, process monitoring and yield learning. Standard logic diagnosis algorithms can work on production test data and are able to locate the fault sites to a large extent [2][3][4]. Yet the identification of the underlying defect type is often only possible if additional tests with high resolution are applied in a second test and are complemented by physical failure analysis (PFA) [5].

PFA may benefit from any approach that can correctly identify susceptible weak structures and locations [6]. Still, the lack of accurate logic models of faulty behaviours caused by defects complicates this analysis. The activation and detection of the fault depend on physical and topological parameters of the design, some of which cannot be predicted [7]. Along with non-modelled non-functional interactions during testing, the increased occurrence of intermittent faults aggravates the problem[8]. This leads to difficulties in finding a match between test outcomes and defect mechanisms [9] and exceeds the capabilities of most logic diagnosis techniques.

The lack of predictable rules to relate the test outcome to a defect mechanism is the perfect scenario for machine learning approaches. Machine learning refers to a variety of algorithms used in contexts where the solution cannot be programmed in an if-then-else fashion, i.e., with fixed rules. Such algorithms are able to infer a structure in a given data set. In particular, neural networks [10] have been successfully applied in different domains, such as speech or image recognition, with great success.

We present an approach to find a match between the observed faulty behaviour during production test and certain fault models with the help of neural networks. The traditional test flow (figure I-a) is complemented with the classifier. The classification technique works on top of a logic diagnosis algorithm for fault location, and does not require a second pass to test failing chips with diagnostic patterns. The modified flow is depicted in figure I-b. With information from test and logic diagnosis, the classifier is able to identify the fault type. Based on previous experience, a threshold T is set which is considered to be the minimum bound for a defect to be considered systematic. Whenever T is surpassed, a warning is issued that informs of a systematic defect. Further analysis is important in chips affected by a systematic defect, since a confirmation of this diagnosis would allow to correct the problem immediately. Early classification allows a prioritization of a second diagnostic pass and PFA utilization. Given that classification using an ANN is an extremely fast and cheap operation, a manufacturer could avoid time consuming second pass diagnosis or even PFA for defects which are outliers and rather dedicate the resources to confirm the presence of a systematic defect, which can be corrected. The approach is validated with permanent and intermittent faults which include crosstalk, complex bridging functions, delay faults and can easily be extended.

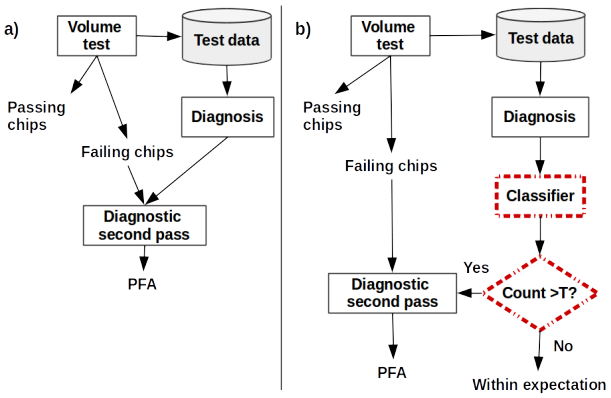

Fig. 1. a) Traditional test flow; b) Modified test flow

The remainder of the paper is organized as follows: section II describes the available techniques for test and diagnosis that contribute to locating or characterizing the fault. Section III gives an insight into neural networks, while IV explains how the classification takes place. Section V summarizes the experimental results.

II. Test, diagnosis and defect mechanism identification

The primary goal of test is to detect any possible defect in the chip, and a high success has been achieved up to now with relatively simple fault models [9]. For diagnosis, more sophisticated defect mechanisms introduced by newer technologies require more accurate modeling. In [11], the authors allow the user to describe a "user-defined fault model", or UDFM, which may be generated taking into account the layout of the considered library cell. Each of the cells in the standard library is characterized, and the detecting patterns are stored for each defect. This information is fed to the test generator, which generates high-efficiency test sets.

More general fault models are the conditional line flip model[12] or pattern-dependent faults [13], which allow considering timing, indeterminism, or layout neighbourhoods. These rather complex fault models are useful for high quality test pattern generation and high defect coverage, but there is limited progress to deduct a fault model from a test outcome. Techniques such as presented in [2] or [14] perform model-independent logic diagnosis. The logic-level representation of the circuit provides the diagnosis approaches with enough information to find the fault location. Unfortunately, it is insufficient to provide insight into the underlying problem.

Many techniques have been proposed which take advantage of layout information both in test pattern generation and diagnosis. Methods such as [4] extract from the layout the defective location's neighborhood, and infer from the logic values the function which activates the fault. However, the authors assume deterministic behaviour of the faults in given neighbourhood conditions, and do not handle intermittent faults. They use both failing and passing pattern information, and patterns which are not in the test are given a *don't care* value, possibly leading to deviations from the original activation function. Based on [4], [15] uses machine learning to discard candidates with inconsistent activation functions and thus enhance diagnosis resolution. Such layout-aware diagnostic approaches provide a good first approximation to understand the underlying defect. When a big population is available, they can be successfully combined with yield learning [16]. Other approaches assume a fault model and locate the problematic signal or gate based on this. Examples are [17], where the authors locate bridges in a design and classify their type. While such techniques are successful, the analysis is limited to bridging faults.

As nanometer technology evolves, many factors influencing circuit behaviour must be accounted for, including variations. It is possible to predict the performance of a unit by statistically analysing high volume test results. Researchers have developed approaches to predict the test outcome of a circuit based on the results of previous tests [18]. Machine learning has also been used to separate critical from non-critical faults [19][8]. Although these approaches rule out systems affected by noise, they do not provide information about the critical defect in the actual faulty chips.

This paper presents an approach to point out the fault model to which the observed behaviour may belong based on volume test outcomes and standard logic diagnosis. This will avoid the need of waiting for PFA results for every faulty chip, while it provides more detailed information and speeds up PFA. To the best of the authors' knowledge, no attempt has been made so far to relate test outcomes to permanent and intermittent fault manifestations by means of neural networks.

III. Neural Networks

An artificial neural network (ANN) is an information processing structure, often considered a universal function approximator. It can detect data trends and structures too complex to be detected by human experts or even by other computing techniques. ANNs are configured (or *learn*) to solve a certain problem. The term *supervised learning* refers to algorithms which find a mapping between a set of inputs called *features* and the provided output values. *Classification* refers to the mapping of certain patterns of features into a certain given category. In our work, we assume a (correctly) labelled learning set is available from previous production, i.e., a large set of feature vectors and the corresponding class or *label* are available a priori. The ANN learns from (or is *trained* with) this set.

This section introduces ANNs in the context of supervised learning for classification. More specifically, it describes the type of ANNs deployed in this work using the Keras [20] library, which is based on Theano [21]. For readers interested in more details we refer to [10].

A. Structure and feed-forwarding

ANNs are formed by a number of processing elements referred to as *neurons*. A neuron is connected by means of directed edges to other neurons, and those edges are annotated with *weights*.

Neurons are organized in *layers*. Each layer l is connected to layers $l-1$ and $l+1$. In a fully connected ANN like the one used in this work, this means every cell in layer l has all $l-1$ cells as predecessors, and all $l+1$ cells as successors. There are three types of layers, namely input, hidden and output layer. The input layer has one cell per feature, and its *activation* is just the input vector X. Hidden layers calculate their activation as

$A_l = \Phi(A_{l-1} * W_l^T)$

where A_{l-1} is the activation of the predecessor layer and W_l is the weight matrix, of size $|l-1| * |l|$, as it is formed by all weight vectors associated to edges between layers $l-1$ and l. Φ is a differentiable non-linear function, which allows the network to approximate non-linear functions. One of the most common choices for Φ is the hyperbolic tangent, used in this work.

For classification problems, the output layer has as many nodes as the number of classes, C. The activation of the last layer, which is a vector of real numbers of length C, is converted into a probability vector by using the softmax function:

$P(faultclass = \mathrm{c}|X) = e^{F(X)[c]} / \sum_{k \in C} e^{F(X)[k]}$

and then class c for which the corresponding probability is the highest, is taken as the solution. The process of calculating the output given a certain input vector X is known as *feed-forwarding*.

B. Learning

ANNs learn because they can adapt the weights associated to their neurons. Given that ANNs learn by example, they must be provided with a so-called *tranining set*. A training set TrS consists of a large set of pairs of the form $(features, label)$. An objective function Q is defined which the learning algorithm will try to minimize. In this work we have considered the mean squared error between the labels in the training set and the feedforward value calculated by the ANN given the corresponding feature vectors. *Backpropagation* learning [22] iteratively adapts the weights using an optimizer function that tries to minimize the difference between labels and feedforward values for the training set. The algorithm finishes when a maximum number of iterations or a maximum error bound have been reached.

A possible optimizer is gradient descent, which for every iteration adapts weights by differentiating the cost function: $w := w - \alpha \nabla Q(w)$ where w is the weight, Q is the cost function and α is the *learning rate*. Keras implements stochastic gradient descent, which means the weights are modified after evaluating a *minibatch*: a subset of the training set, and not the complete population. In succeeding iterations, data is shuffled to avoid cycles. Classic stochastic gradient descent can diverge in some cases. To avoid it,the learning rate can include momentum, i.e., the learning method has a "memory" to prevent oscillations in the direction of the steps. Keras implements Adadelta [23], which includes a dynamic learning rate for each parameter. This is particularly useful since gradients in different cells may have several orders of magnitude between them, complicating learning with traditional approaches.

Neural networks may infer a too complex function which fits closely the training data set but extrapolates poorly. This problem is known as *overfitting*, and one technique to avoid it is weight decay, which penalizes weights with very large absolute values. Weight decay can be included in the optimization function to ensure good learning properties. Another problem is *underfitting*, which can be detected because the accuracy for both TrS and TeS is low. This is an indication that the inferred function is too simple, i.e., the configuration of the ANN may not be expressive enough.

IV. Defect Classification with Neural Networks

The proposed method takes advantage of knowledge acquired during past production. As a first step, the ANN is trained with information gathered from past production. The trained ANN, ready to classify new data, is then integrated into the fault classifier. The block diagram in figure 2 shows the inputs to be provided as well as the basic structure and output. Test provides the failing test patterns, including input and observed output. Diagnosis provides a candidate location, to which we will refer as *victim line*. If the diagnosis tool returns more than one candidate, the analysis can be performed for all returned candidates.

The defect classifier is divided in two blocks. The first block, or logic simulator, gathers the features. The second block contains the trained ANN which receives the features from the logic collector as input and outputs the classification.

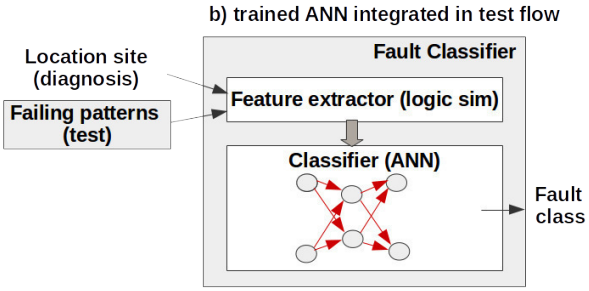

Fig. 2. Proposed defect classifier.

Keras allows to parametrize the ANNs to a large extent. The experimental results in section V report the accuracy of classification depending on the topology of the ANN. In the following subsections we describe the chosen features and the output encoding.

A. Features

Along with the neural network structure choice and proper training, feature selection is a crucial step which determines the success of the approach. We do not apply feature learning, but select input values representative of the classes we want to differentiate. The approach is not intended to increase the costs of the traditional flow. The features used below can be easily obtained by logic simulation. The fault models covered at the moment include interconnect and gate faults:

- crosstalk induced delay: a victim line is pulled down/up because of switching activity in the opposite direction neighbourhood.
- dominant-and(-or) bridge: due to an unwanted connection, two lines are connected. The victim line changes its value to the AND (OR) of its value and the aggressor's.
- byzantine bridge: either of the two signals connected by mistake may flip if they have opposite values.
- slow-to-rise and slow-to-fall gates: a gate exhibits long delays for transitions to high or to low, respectively.

We now select information obtainable from the circuit topology and logic simulation that can represent our fault models. From the simulation of the subset of failing patterns $T_f \subset T$ we can obtain the following values:

1) Ratio of failing patterns with victim line 0: If for all T_f the victim line is 0 in the fault-free case, then we have some strong evidence that the victim line may be driven by a dominant-or bridge. Let $T_f^0 \subset T_f$ be the subset of failing patterns which set the victim line to 0, and set $vl@0 = |T_f^0|/|T_f|$. Then $vl@0$ is a representative feature of a dominant-or fault.

2) Ratio of failing patterns with victim line 1: In exactly the same way we set $vl@1 = |T_f^1|/|T_f|$, which is the ratio of all failing patterns where the victim line is 1 in the fault-free case among all the failing patterns. It is an indication of a dominant-and bridge.

3) Ratio of falling transitions at victim line: This feature is calculated as $fallVL = |T_{fall}|/|T_f|$, where $T_{fall} = t_i \in T_f | vl(t_i) = 0 \& vl(t_{i-1}) = 1$, i.e., T_{fall} is the number of

failing patterns at which the victim line had value 0, and a transition from the previous cycle took place. This feature is a strong indicator for slow-to-fall faults.

4) Ratio of rising transitions at victim line: Analogously, we define $riseVL = |T_{rise}/|T_f|$, where $T_{rise} = t_i \in T_f | vl(t_i) = 1 \& vl(t_{i-1}) = 0$. $riseVL$ also quantifies evidence of a slow-to-rise fault.

5) Driving gate inputs: The driving gate and the patterns applied at its input help characterize some given faults. Features are added to include information such as switching at the input of the gate, even if it causes no switching at the output. This gives a hint towards possible temporal glitches in the signal that may derive in an error if a timing fault is present. For gate g driven by inputs I, we gather six additional numbers from logic simulation:

$$\begin{aligned}
T_{c0} &= |\{t_i \in T_f | g(t_i) = g(t_{i-1}) = 0 \& I(t_i) ! = I(t_{i-1})\}| \\
T_{c1} &= |\{t_i \in T_f | g(t_i) = g(t_{i-1}) = 1 \& I(t_i) ! = I(t_{i-1})\}| \\
T_{s0} &= |\{t_i \in T_f | g(t_i) = g(t_{i-1}) = 0 \& I(t_i) = I(t_{i-1})\}| \\
T_{s1} &= |\{t_i \in T_f | g(t_i) = g(t_{i-1}) = 1 \& I(t_i) = I(t_{i-1})\}| \\
T_{tf} &= |\{t_i \in T_f | g(t_i) = 1 \& g(t_{i-1}) = 0 \& I(t_i) ! = I(t_{i-1})\}| \\
T_{tr} &= |\{t_i \in T_f | g(t_i) = 0 \& g(t_{i-1}) = 1 \& I(t_i) ! = I(t_{i-1})\}|,
\end{aligned}$$

where $T_{c0}(T_{c1})$ is the number of patterns for which at least one input of the gate changed but the output remained at 0 (1), T_{s0} (T_{s1}) is the number of patterns for which inputs and output remained stable and the output had value 0 (1) and T_{tf} (T_{tr}) is the number of patterns for which a transition at the inputs caused a falling (rising) transition at the output. Based on these values, we extract the following features:

$$\frac{T_{c0}}{T_f}, \frac{T_{c1}}{T_f}, \frac{T_{s0}}{T_f}, \frac{T_{s1}}{T_f}, \frac{T_{falling}}{T_f}, \frac{T_{rising}}{T_f} \quad (1)$$

6) Ratio of unexplained patterns: We define $T_f^u \subset T_f$ as those failing patterns which cannot be explained by flipping the victim line. We define the feature $unexplained = |T_f^u|/|T_f|$, and if $unexplained > 0$, there must be at least one more culprit in the circuit. This reflects the situation that a bridging fault affects two lines in an arbitrary way.

7) Maximum ratio of transitions in the neighborhood: We extract the physical neighbourhood of the victim line from layout information. Logic simulation determines for each neighbour $v \in N_h$ the number of patterns $t_i \in T_f$ for which they switch their value by applying t_i:

$$max\_trans = \frac{max_{v \in N_h}\{t_i \in T_f | v(t_i) \neq v(t_{i-1})\}}{|T_f|}. \quad (2)$$

Note, that $t_i \in T_f$, but not necessarily $t_{i-1} \in T_f$. The feature max_trans helps to point out crosstalk faults.

8) Average ratio of neighbours with different value than the victim line: The feature

$$avg\_dist = \frac{\sum_{v \in N_h} \sum_{t \in T_f} \{v | v(t) \neq vl(t)\}}{|N_h||T_f|} \quad (3)$$

computes the average of the ratio of patterns in which the neighbours in N_h have a different value than the victim line vl with different value. It is included to help classify bridges.

The values of all thirteen features are integrated into a vector X. None of the features allows by itself the classification of the faults.

The features are expected to have noisy behaviour. Given that feature extraction is based solely on logic simulation, which does not capture timing behaviour accurately, this is the typical diagnosis limitation: it is not possible to know the defect behaviour beforehand, which requires diagnosis methods that can work with less-than-perfect matches.

B. Output Encoding

The problem at hand is a multiclass classification with possible outcomes: slow-to-rise, slow-to-fall, crosstalk, dominant-AND, dominant-OR, byzantine bridge. Outputs are encoded as vectors of length c, where c is the number of classes. The class with the highest probability is taken as the result provided by the network. For the training set, the labels are known and the expected values are set to 0 except for the one corresponding to the class, which should be 1. The expected outputs for the elements in the training set are set to $[1, 0, 0, 0, 0, 0]$ for slow-to-rise faults, $[0, 1, 0, 0, 0, 0]$ for slow-to-fall faults, and so on.

V. Experimental Validation

To validate the presented approach, faults have been injected in random locations into a set of five of the ITC'99 circuits and five bigger circuits kindly provided by NXP. Under full-scan test assumption, test patterns were generated with a commercial tool targeting transition faults with n-detect. 200 random faults have been injected per fault type and circuit. The injection was performed in random locations. Each fault was simulated and features were collected for all of them. The test environment has been simulated with a timing-accurate simulation framework [24]. The classifier, on the other hand, integrates an in-house logic simulator which extracts the feature values and feeds them to the ANN. The gathered data is divided into training and test sets.

The ANNs deployed in this work are fully connected. The number of layers and number of cells per layer are indicated for each network in the experimental results. The maximum number of learning iterations was set to 15000. 65% of the generated data are used as training set. The remaining 35% fault injections are used as test set. Configurations with over- or underfitting for at least one circuit are discarded. The classification results presented are exclusively based on the test set. Accuracy is calculated as the percentage of faults correctly classified.

A. ANN trained with product knowledge

The first batch of experiments consisted of training a network for each circuit, and then testing using the test set of the same circuit. Experimental results show that the best results were obtained with up to four layers and from eight to thirteen cells per layer. The first ten data rows of table I show the configuration which obtained the best result for every circuit. Column *accuracy* shows the average accuracy for all fault models. The last six columns show the accuracy for each of the six fault models considered.

The accuracy of the tool is limited by the indeterminism introduced by timing. For instance, in case of a slow gate which has a stable value in logic simulation but switches

TABLE I
CLASSIFICATION RESULTS FOR ANN TRAINED WITH DATA OF THE DIAGNOSED DESIGN

Optimal configuration per circuit									
circuit	layers	cells	accuracy	or bridge	and bridge	byzantine	slow to rise	slow to fall	crosstalk
b18	1	13	83.52%	79.31%	83.93%	84.62%	75.51%	81.54%	93.85%
b19	2	12	78.51%	66.67%	66.67%	92.18%	77.08%	64.62%	96.92%
b20	2	10	83.61%	71.19%	87.93%	84.37%	79.59%	81.54%	95.38%
b21	3	12	87.22%	86.44%	85.96%	87.69%	87.75%	75.38%	100%
b22	3	12	87.32%	87.93%	94.44%	87.5%	87.75%	72.31%	95.38%
p35k	3	10	84.84%	78.33%	81.67%	89.23%	87.75%	79.69%	92.31%
p45k	3	8	85.67%	88.14%	84.21%	85.93%	85.71%	81.54%	87.69%
p78k	3	11	92.66%	96.77%	96.77%	96.92%	81.63%	83.08%	98.46%
p141k	2	13	90.98%	98.39%	91.80%	93.75%	83.67%	80%	96.92%
p330k	1	13	90.88%	88.14%	91.67%	92.19%	87.75%	89.23%	95.38%
Identical configuration per circuit									
circuit	layers	cells	accuracy	or bridge	and bridge	byzantine	slow to rise	slow to fall	crosstalk
b18	2	9	85.47%	79.31%	92.85%	87.69%	75.51%	78.46%	96.92%
b19	2	9	77.61%	57.89%	63.89%	92.19%	81.25%	67.69%	95.38%
b20	2	9	81.39%	67.79%	87.93%	89.06%	75.51%	78.46%	87.69%
b21	2	9	85.83%	89.83%	84.21%	86.15%	83.67%	70.76%	100%
b22	2	9	83.94%	82.75%	96.29%	82.81%	79.59%	67.69%	95.38%
p35k	2	9	84.02%	71.67%	83.33%	89.23%	89.79%	78.13%	92.31%
p45k	2	9	84.11%	91.52%	63.15%	87.5%	87.75%	72.31%	89.23%
p78k	2	9	89.94%	95.16%	93.54%	96.92%	81.63%	72.31%	98.46%
p141k	2	9	91.26%	98.39%	96.72%	98.44%	75.51%	81.53%	93.84%
p330k	2	9	90.61%	91.53%	90%	90.63%	85.71%	89.23%	95.38%

shortly -and slower than it should-, causing a problem that logic simulation cannot model. In particular, for a very deep circuit, gates near the output may have glitches with more frequency, caused by differences in the arrival times of their inputs. If this situation arises too often in the test set, it will lead the tool to interpret the values as a bridging faults. If such faults are also included in the training set, it will cause lower accuracy levels, as for circuit $b19$. However, regardless of this, the method can indeed work with this typical diagnosis limitation: the overall accuracy reaches almost 80% in the worst case and over 90% in the best cases.

Although optimal configuration for the circuits differ, such an optimal choice can only be met with some knowledge about the circuit and available data. As a starting point, however, a global optimal configuration can be chosen which works acceptably for all circuits. In this case, a neural network with two hidden layers and nine cells per layer renders accuracy results as shown in the last ten rows of table I. Despite not achieving optimal configuration results, the accuracy still suffices in volume test as a first estimation of systematic defects.

Training and classification were performed on a 64-bit x86 at 3301 MHz. Both operations depend on the size of the ANN and the topology. In our experiments, training takes under $90s$ for any ANN topology, while prediction for 400 experiments can be performed in $1s$.

B. Classification of intermittent faults

Due to physical parameters, a fault may be activated intermittently, for instance, depending on temperature or power droop. We have generated a test set with intermittent faults, i.e., the setup remains as described before but in the timing simulation, when the activation condition is met we randomly decide if the fault is injected or not. Table II shows the results of the classification accuracy for this kind of faults, and confirms that out method is robust w.r.t. indeterminism.

C. ANN reuse for new product

The results in the previous sections are the best possible scenario: data from the circuit under test is available and labelled to train the network. But the approach is also intended for the first stages of production, when there may be no data available. Gathering the data for one design can take a long time, so it is interesting to survey the possibility of reusing data from other products. We show in table III the average accuracy for each ANN trained with data from the circuit indicated in column *circuit* when data from all other nine circuits is classified. The ANN has been configured with two layers and nine cells per layer, the "best global configuration" chosen in the previous section. These are not the best results obtained, but it is reasonable to assume that without prior experience with the circuit, a "generally good" configuration is to be taken.

VI. CONCLUSION

We presented a method to distinguish between crosstalk induced delay, slow to rise/fall gates and bridges in interconnects from production test data. By using neural networks we are able to distinguish among different types of faults without changing the test set. Moreover, we show that the method could be used with data from another design at very early stages of production, when there is no available data, and it is robust with faults which exhibit an intermittent behaviour. If more complex fault models have to be classified as well, the set of features could be enhanced.

TABLE II
AVERAGE ACCURACY OF ANNS CLASSIFYING INTERMITTENT FAULTS.

Circuit	accuracy	or bridge	and bridge	byzantine	slow to rise	slow to fall	crosstalk
b18	83.51%	74.60%	83.05%	87.69%	87.69%	83.07%	84.62%
b19	82.49%	61.02%	76.27%	89.06%	89.23%	81.54%	95.38%
b20	86.41%	78.33%	82.14%	92.31%	81.54%	91.07%	95.45%
b21	87.59%	83.60%	79.31%	90.77%	93.84%	81.53%	95.38%
b22	83.51%	82.46%	84.75%	90.77%	67.69%	76.92%	98.46%
p35k	83.69%	80.70%	77.19%	86.15%	84.61%	85.93%	92.30%
p45k	85.49%	84.74%	84.37%	93.85%	80.00%	76.92%	92.65%
p78k	89.29%	81.96%	93.55%	96.92%	84.62%	80.00%	98.46%
p141k	86.77%	83.33%	79.66%	89.23%	89.23%	85.94%	92.31%
p330k	86.70%	77.97%	84.21%	88.89%	89.23%	87.69%	92.31%

TABLE III
AVERAGE ACCURACY OF ANNS CLASSIFYING DATA FROM OTHER DESIGNS. ANN CONFIGURED WITH 2 LAYERS AND 9 CELLS PER LAYER.

Circuit	accuracy	or bridge	and bridge	byzantine	slow to rise	slow to fall	crosstalk
b18	83.71%	78.27%	90.86%	87.72%	73.61%	72.25%	98.63%
b19	83.33%	74.20%	73.05%	91.54%	85.03%	76.01%	98.11%
b20	84.02%	75.06%	83.48%	89.13%	87.92%	73.96%	95.04%
b21	85.04%	84.23%	81.47%	90.67%	82.93%	74.13%	95.56%
b22	85.21%	78.42%	86.89%	89.82%	82.02%	75.16%	97.78%
p35k	80.93%	70.68%	78.55%	88.94%	82.01%	72.99%	91.28%
p45k	80.32%	90.59%	63.26%	91.54%	79.75%	64.54%	90.59%
p78k	77.85%	84.11%	83.31%	82.20%	76.79%	66.44%	76.75%
p141k	83.65%	93.21%	89.65%	98.29%	73.85%	73.87%	92.13%
p330k	84.12%	92.29%	89.17%	87.91%	73.16%	69.51%	91.45%

BIBLIOGRAPHY

[1] R. Aitken, "Yield learning perspectives", *IEEE Design & Test of Computers*, vol. 29, no. 1, pp. 59–62, Feb 2012.

[2] S. Holst and H.-J. Wunderlich, "Adaptive debug and diagnosis without fault dictionaries", *Journal of Electronic Testing*, vol. 25, no. 4-5, pp. 259–268, 2009.

[3] Y. Benabboud, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, L. Bouzaida, and I. Izaute, "A fault-simulation-based approach for logic diagnosis", in *4th Int'l Conf. on Design Technology of Integrated Systems in Nanoscale Era, 2009. (DTIS'09)*, April 2009, pp. 216–222.

[4] R. Desineni, O. Poku, and R. Blanton, "A logic diagnosis methodology for improved localization and extraction of accurate defect behavior", in *Proc. IEEE Int'l Test Conference, 2006. (ITC '06)*, Oct 2006, pp. 1–10.

[5] S.-Y. Liu, Y.-C. Hou, C.-C. Chang, and J.-C. Lin, "Sige profile inspection by using dual beam fib system in physical failure analysis", in *20th IEEE Int'l Symposium on the Physical and Failure Analysis of Integrated Circuits, 2013 (IPFA'13)*,, July 2013, pp. 490–492.

[6] C. Hora, R. Segers, S. Eichenberger, and M. Lousberg, "An effective diagnosis method to support yield improvement", in *Proc. Int'l Test Conference, 2002 (ITC'02)*, 2002, pp. 260–269.

[7] A. Ivanov, S. Rafiq, M. Renovell, F. Azais, and Y. Bertrand, "On the detectability of cmos floating gate transistor faults", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 1, pp. 116–128, Jan 2001.

[8] L. Rodríguez Gómez, A. Cook, T. Indlekofer, S. Hellebrand, and H.-J. Wunderlich, "Adaptive Bayesian Diagnosis of Intermittent Faults", *Journal of Electronic Testing: Theory and Applications (JETTA)*, vol. 30, no. 5, pp. 527–540, 2014.

[9] W. Maly, A. Gattiker, T. Zanon, T. Vogels, R. Blanton, and T. Storey, "Deformations of ic structure in test and yield learning", in *Proc. IEEE Int'l Test Conference (ITC'03)*, vol. 1, Sept 2003, pp. 856–865.

[10] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[11] F. Hapke, W. Redemund, A. Glowatz, J. Rajski, M. Reese, M. Hustava, M. Keim, J. Schloeffel, and A. Fast, "Cell-aware test", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 9, pp. 1396–1409, Sept 2014.

[12] H.-J. Wunderlich, Ed., *Models in Hardware Testing*. Springer-Verlag Heidelberg, 2010, vol. 43.

[13] R. Blanton and J. Hayes, "The input pattern fault model and its application", in *Proc. European Design and Test Conference, 1997 (EDTC'97)*, Mar 1997, p. 628.

[14] A. Cook, M. Elm, H. Wunderlich, and U. Abelein, "Structural In-Field Diagnosis for Random Logic Circuits", in *Proc. European Test Symposium (ETS'11)*, Trondheim, Norway, may 2011, pp. 111 –116.

[15] Y. Xue, O. Poku, X. Li, and R. Blanton, "Padre: Physically-aware diagnostic resolution enhancement", in *Proc. IEEE Int'l Test Conference (ITC'13)*, Sept 2013, pp. 1–10.

[16] R. Blanton, W. Tam, X. Yu, J. Nelson, and O. Poku, "Yield learning through physically aware diagnosis of ic-failure populations", *IEEE Design & Test of Computers*, vol. 29, no. 1, pp. 36–47, Feb 2012.

[17] Y. Benabboud, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, L. Bouzaida, and I. Izaute, "Comprehensive bridging fault diagnosis based on the slat paradigm", in *4th Int'l Conf. on Design Technology of Integrated Systems in Nanoscale Era, 2009. (DTIS'09)*, April 2009, pp. 216–222.

[18] R. Madge, B. Benware, M. Ward, and R. Daasch, "The value of statistical testing for quality, yield and test cost improvement", in *Proc. of IEEE Int'l Test Conference(ITC'05)*, Nov 2005, pp. 10 pp.–332.

[19] J. De Kleer, "Diagnosing multiple persistent and intermittent faults", in *Proc. of the 21st Int'l Jont Conf. on Artifical Intelligencei(IJCAI'09)*, ser. IJCAI'09. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2009, pp. 733–738.

[20] F. Chollet, "Keras", https://github.com/fchollet/keras, 2015.

[21] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio, "Theano: new features and speed improvements", Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[22] P. J. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences", Ph.D. dissertation, Harvard University, 1974, department of Applied Mathematics.

[23] M. D. Zeiler, "ADADELTA: an adaptive learning rate method", *CoRR*, vol. abs/1212.5701, 2012.

[24] S. Holst, E. Schneider, and H.-J. Wunderlich, "Scan Test Power Simulation on GPGPUs", in *Proceedings of the 21st IEEE Asian Test Symposium (ATS'12)*. IEEE Computer Society, 2012, pp. 155–160.