Journal of Electronic Testing (2022) 38:395–417
https://doi.org/10.1007/s10836-022-06020-z

Automated Design Error Debugging of Digital VLSI Circuits

Mohammed Moness[1] · Lamya Gaber[1] · Aziza I. Hussein[2] · Hanafy M. Ali[1]

Received: 10 January 2022 / Accepted: 26 July 2022 / Published online: 31 August 2022

Abstract
As the complexity and scope of VLSI designs continue to grow, fault detection processes in the pre-silicon stage have become crucial to guaranteeing reliability in IC design. Most fault detection algorithms can be solved by transforming them into a satisfiability (SAT) problem decipherable by SAT solvers. However, SAT solvers consume significant computational time, as a result of the search space explosion problem. This ever- increasing amount of data can be handled via machine learning techniques known as deep learning algorithms. In this paper, we propose a new approach utilizing deep learning for fault detection (FD) of combinational and sequential circuits in a type of stuck-at-faults. The goal of the proposed semi-supervised FD model is to avoid the search space explosion problem by taking advantage of unsupervised and supervised learning processes. First, the unsupervised learning process attempts to extract underlying concepts of data using Deep sparse autoencoder. Then, the supervised process tends to describe rules of classification that are applied to the reduced features for detecting different stuck-at faults within circuits. The FD model proposes good performance in terms of running time about 187 × compared to other FD algorithm based on SAT solvers. In addition, it is compared to common classical machine learning models such as Decision Tree (DT), Random Forest (RF) and Gradient Boosting (GB) classifiers, in terms of validation accuracy. The results show a maximum validation accuracy of the feature extraction process at 99.93%, using Deep sparse autoencoder for combinational circuits. For sequential circuits, stacked sparse autoencoder presents 99.95% as average validation accuracy. The fault detection process delivers around 99.6% maximum validation accuracy for combinational circuits from ISCAS'85 and 99.8% for sequential circuits from ISCAS'89 benchmarks. Moreover, the proposed FD model has achieved a running time of about 1.7x, compared to DT classifier and around 1.6x, compared to RF classifier and GB machine learning classifiers, in terms of validation accuracy in detecting faults occurred in eight different digital circuits. Furthermore, the proposed model outperforms other FD models, based on Radial Basis Function Network (RBFN), achieving 97.8% maximum validation accuracy.

Keywords Fault diagnosis · Deep learning · Neural networks · Autoencoder

1 Introduction

Given rapid downscaling of integration, there have been ever increasing challenges to circuit designers [1]. Figure 1 summarizes the main phases of the IC design flow with verification, debugging and correction processes. IC design is a highly complicated task as it needs a full understanding of the IC restrictions, specifications and all the required EDA tools. It starts with writing a form of its specifications passed through number of complex design steps to achieve the desired final chip after fabricating billions of transistors onto one piece of a semiconductor die, having a very small size (no larger than a fingernail). The main challenges in the design process are functional verification, debugging and auto-correction processes. Functional debugging aims

Responsible Editor: V. D. Agrawal

✉ Lamya Gaber
lamya.gaber@mu.edu.eg

Mohammed Moness
m.moness@mu.edu.eg

Aziza I. Hussein
azibrahim@effatuniversity.edu.sa

Hanafy M. Ali
hmali@mu.edu.eg

[1] Computers and Systems Eng. Dept, Minia University, Minia, Egypt

[2] Electrical and Computer Eng. Dept, Effat University, Jeddah, Saudi Arabia

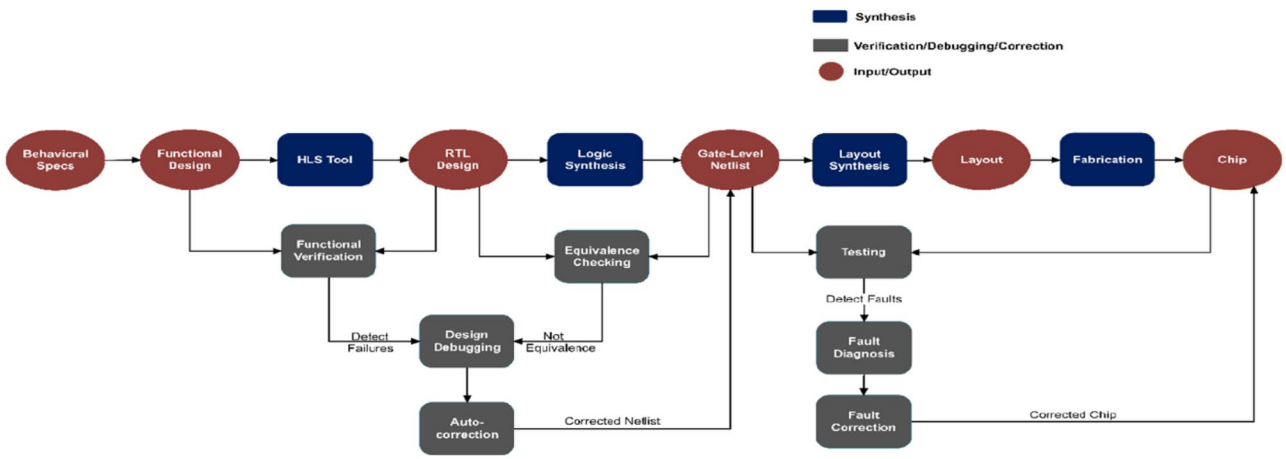


Fig. 1 Overview of IC Design Flow

to find root-cause of a functional failure. This increase in size and scope of bugs has made functional debugging one of the largest bottlenecks in the design cycle. Therefore, it points to a crucial requirement for more scalable and innovative debugging solutions.

Consequently, the nature of faults in digital circuits has been more complex and unpredictable, and automatic fault diagnosis of complex systems has become crucial to maintain low production costs with changing technologies. This process attempts to enhance system performance, avoid negative results of logic faults and determine abnormal functioning from data that may be corrupted due to unpredictable events. Following detection of abnormal functioning, reasons for failures can be located and identified. Therefore, many researchers have investigated differing methods for verifying, debugging and correcting digital systems by trading-off accuracy, speed and human interpretations. The main difference between each process is explained in the following definitions:

Definition 1: Verification is the process of searching for discrepancies between two levels of circuit abstraction in the pre-silicon design.

Definition 2: Debugging is the process following failed verification intended for diagnosing and detecting potential bug locations in an erroneous circuit. Therefore, it is also often termed Fault Localization.

Definition 3: Correction is the phase responsible for modifying components causing errors discovered by debuggers, so it can rectify the desired circuit to behave in its intended manner.

Debugging process is a crucial process in digital design which consumes 60% of the total formal verification time in digital design [2]. This process is required for determining all potential faults in digital VLSI circuits. Many traditional methods are primarily based on satisfiability (SAT) problem that convert the whole process into SAT instance, followed by an attempt to analyze this instance for verifying, debugging and even correcting the digital system [3–6]. For detecting logic faults in the case of erroneous digital circuits, most of these approaches are based on detecting different subsets that give valuable meaning to designers, such as minimal correction subsets (MCSes) and minimal unsatisfiable subsets (MUSes).

There are many techniques based on machine learning algorithms that have been proposed for fault detection processes [7, 8]. Machine learning (ML) techniques are primarily based on a large amount of historical data available for all such applications. It transforms this data into meaningful information used for future analytics and predictive solutions. Therefore, the primary requirement for establishment of a good ML engine is an adequate amount of reliable data. The success of ML models can be proven by quick solutions. The second reason behind the upthrust of ML is deep learning which introduces successive solutions with the availability of high-speed hardware and graphic processing units for expediting the required computation. With the development of smart manufacturing, fault diagnosis becomes essential to ensuring the reliability and safety of industrial systems. Despite most approaches based on ML are focused on locating design faults at a post-silicon stage, detecting faults in a pre-silicon stage is more important to make sure of all components in digital circuits before going deeper into the design flow [9].

In this paper, we illustrate a new model for fault detection (FD) approach based on Autoencoders for detecting stuck-at-faults, in both combinational and sequential circuits. The conducted research introduces the following contributions that are more promising than SAT:

- The search space explosion problem in FD approaches based on formal methods can be solved using counterexample resulting from failed verification passed to semi-supervised learning model for classifying stuck-at-faults, which might be causes for failing verification. In this way, multiple calls of SAT solvers can be avoided.
- Instead of reducing SAT instances by SAT encoding algorithms for detecting faults using SAT solvers, Deep Sparse Auto-Encoders, as unsupervised model, can be implemented on test patterns directly to extract robust latent features. Therefore, conjunction between multiple clauses can be dispensed.
- Exact debuggers using Max-SAT approach for detecting the main reason of faults existed in circuit is replaced by ATALANTA tool for generating multiple patterns for each stuck-at-fault with high fault coverage. Therefore, dataset can be helpful to avoid repetitive generation of multiple unsatisfiable cores caused by single fault.

The rest of our paper is organized as follows: Sect. 2 and Sect. 3 give a brief description of fault detection background and approaches based on machine learning and SAT problem, respectively. Section 4 illustrates the proposed model. Section 5 shows the experimental results. Finally, Sect. 6 is the conclusion.

2 Background

2.1 Fault Classes

Faults are physical defects that may cause a failure in logic circuits or systems. They are described by: its nature, value and duration. The nature of any fault may be a logical or non-logical fault. A logical fault occurs if the logic function of a component or a signal is changed to some other function. Otherwise, it is defined as non-logical fault such as malfunction of clock signals or a power failure, parametric fault and delay fault. The value of a fault may be fixed or varying erroneous logical values. Also, the duration of a fault may be permanent (known as solid faults) or temporary.

In digital VLSI circuits, there are different classes of faults or bugs which are divided into three main types: design faults, verification faults, manufacturing faults and electrical faults. Design faults are that occurred as functional bugs or electrical bugs. The main reason for design faults is interference by a designer in a synthesis phase in order to reach a specific level of system optimization. Also, they may be caused by automated synthesis tools with software bugs. These bugs occur in gate-level implementation or RTL implementation of IC design. In this paper, we focus on detecting logical faults as they may occur around 98% before tap-out and about 2% after tap-out. Tables 1 illustrates different types of faults in IC design.

The proposed algorithm is focused on detecting stuck-at-faults that can be mapped to gate replacement faults in logical circuits as proposed in [10], proving set of corollaries for describing mapping from stuck-at-faults into design fault model domain. The following definitions denotes types of design faults:

Definition 1 (Gate replacement error). It defines a design error which can be rectified by replacing the gate g_i with another gate g_j.

Definition 2 (Extra/missing inverter error). It describes a design error which can be corrected by removing or adding an inverter at some input of a gate at some fanout branch.

The method of mapping Stuck-at faults into gate-replacement faults is concluded from the following proof in [10]:

Table 1 Different Types of Faults in IC design

Fault Type	Functional Faults	Verification Faults	Manufacturing Faults	Circuit or Electrical Faults
Occurred in	- Design implementation (RTL or gate -level implementation)	- Behavioral specification (testbenches or assertions) Also, called as *testbench bugs*	- IC manufacturing phase Classified into Gross area defects and spot defects	- Pre-or post-silicon verification - operating region such as frequency, voltage, and temperature
Caused by	- Change in specification -Human factor - Automated synthesis tools with software bugs	- Incorrect transformation of a behavioral specification to a verification code (testbench or assertions)	- Out of tolerance steps (macro level variations) - Scratches from wafer mishandling (global faults) - Missing patterns or extra patterns (spot bugs)	- Undesired interaction between a design and an electrical state such as: Cross-talk, Power -supply noise, Thermal effects and Process variations

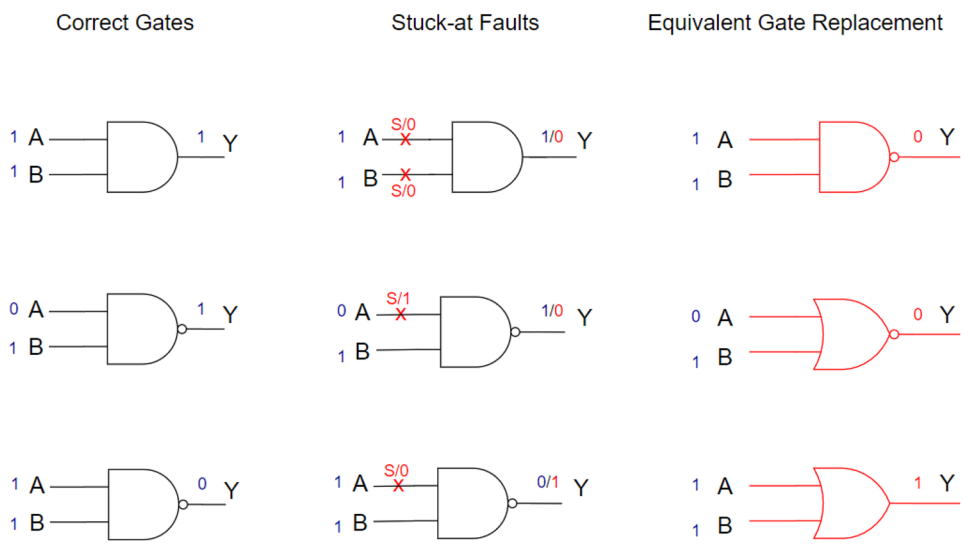


Fig. 2 Examples of Mapping Stuck at faults into gate replacement errors

Theorem 1: *For detecting design errors in the implementation at any gate in a given circuit, it is sufficient to apply a pair of test patterns which detect the stuck-at one and stuck-at zero at one of the gate inputs.*

Therefore, the following set of corollaries describes the mapping process from stuck-at fault model to the design error domain. Also, Fig. 2 gives simple examples of these corollaries.

- Missing/extra inverter at the gate output mapped from s/1 and s/0 faults on two or more gate inputs.
- Replacement faults: AND→OR, OR→NAND, NAND→NOR and NOR→AND results by s/1 faults at one or more gate inputs.
- Replacement faults: AND→NOR, OR→AND, NAND→OR and NOR→NAND mapped from s/0 faults are one or more gate inputs.

2.2 Logical Design Debugging

Design debugging or diagnosis is considered an internal process for improving the design cycle, manufacture yield and shorten the time-to-market window. It defines as the process of finding all sets of fault locations or suspects in the buggy design and correcting the design faults for satisfying given specification. Figure 3 illustrates a design flow of guarantying correctness before going deeper into manufacturing steps. Generally, design debugging is followed by a failed verification for detecting bugs to make them not able to find

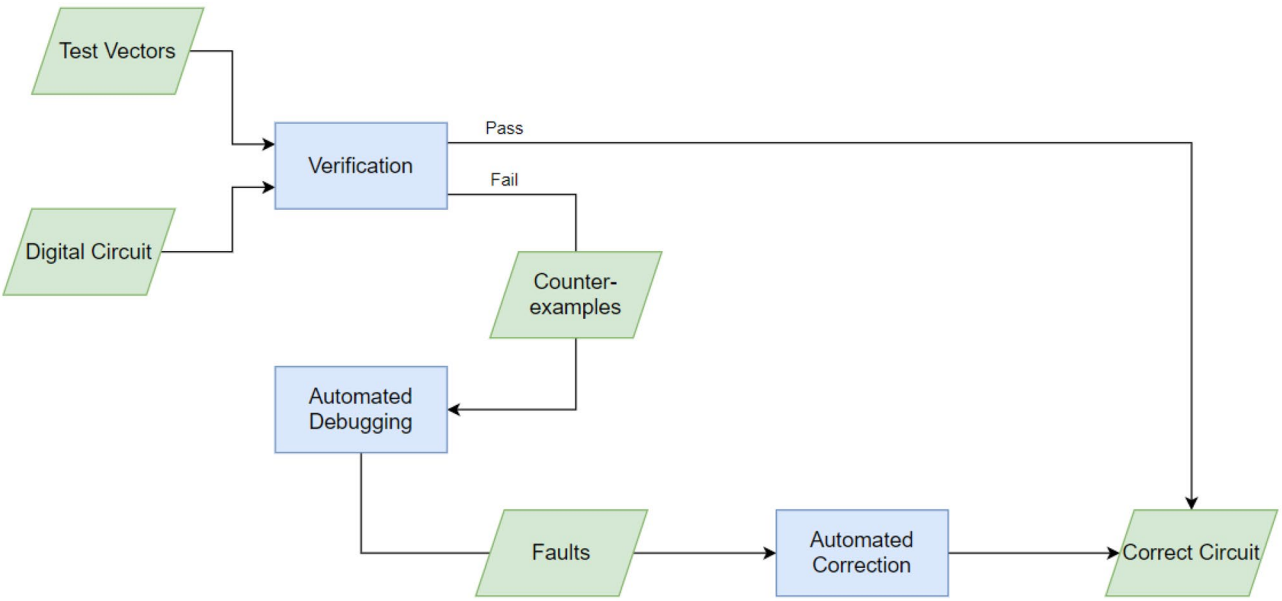


Fig. 3 Design Flow of verification, debugging and correction phases

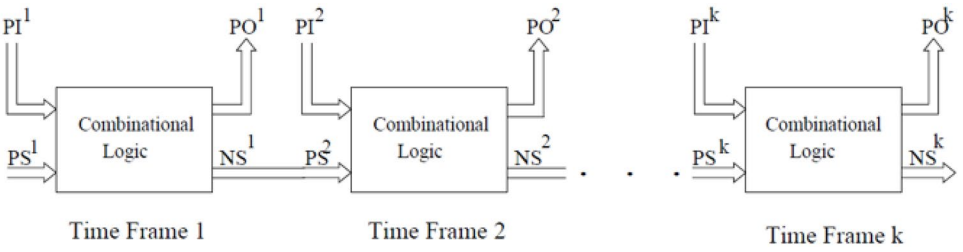


Fig. 4 Iterative Logic Array Model [11]

their ways in the field. Debugging techniques involve taking a counter-example that triggering failures then return a set of locations in the buggy design that might be responsible for the observed faults.

In general, there are two types of Fault diagnosis or debugging which are Design error diagnosis and Fault Diagnosis. The design flow is separated into two groups: the first group is HDL specification, RTL synthesis and Logic synthesis. The second group is generating physical design and Chip. In the early first design stages, malfunctions might exist as a result of specification changes bugs in automated tools or the human factor. Therefore, logic corrections identify the possible corrections in the erroneous netlist to match a specification. In the second latter stages, the fabricated chip might fail testing, so the fault diagnosis should take place given the faulty chip and a netlist. It injects faults into a correct netlist until the netlist emulated the behavior of the faulty chip.

For sequential circuits, debugging process of sequential circuits is similar to that of combinational circuits except that their behavior must be modelled for a finite number of clock cycles. Therefore, the most common approach for modelling sequential circuits is to use the time frame expansion technique or the iterative logic array (ILA) representation. These methods connect the current state and the text state together. Therefore, the sequential circuit is transformed into a new circuit called “unfolded combinational circuit”. Then, it can be debugged like any other combinational circuit. Figure 4 illustrates the iterative logic array model for any combinational logic. Our proposed model focuses on the properties of unfolded combinational circuit of sequential circuits.

2.3 Fault Detection Based on AI

In general, fault diagnosis can be categorized into model-based, signal-based, knowledge-based (also called data-driven) and hybrid/active approaches [12]. Data-driven fault diagnosis that is implemented based on machine learning models (such as support vector machine (SVM), neural network (NN) and fuzzy logic) is more common because of the data analytical methods in them. On the other hand, knowledge-based diagnosis methods are different from other classes as it requires employment of a large volume of historical data available where other classes of fault diagnosis methods have to utilize real-time data. The schematic diagram of knowledge-based fault diagnosis is illustrated in Fig. 5. A variety of artificial intelligence techniques is applied on historical data, then the knowledge base can be extracted which represents the dependency of the variables of the system. Then fault diagnosis decision is determined by checking the consistency between the observed behavior of the system and the knowledge base with the aid of a classifier. Extracting knowledge base can be either qualitative or quantitative in nature.

2.4 Autoencoders

Since 2006, Deep learning becomes a crucial domain in machine learning. Autoencoder is a special type of feedforward neural network for the task of representation learning. The goal of autoencoders is to find the compressed representation of the input called "code" or "latent-space-representation" that can

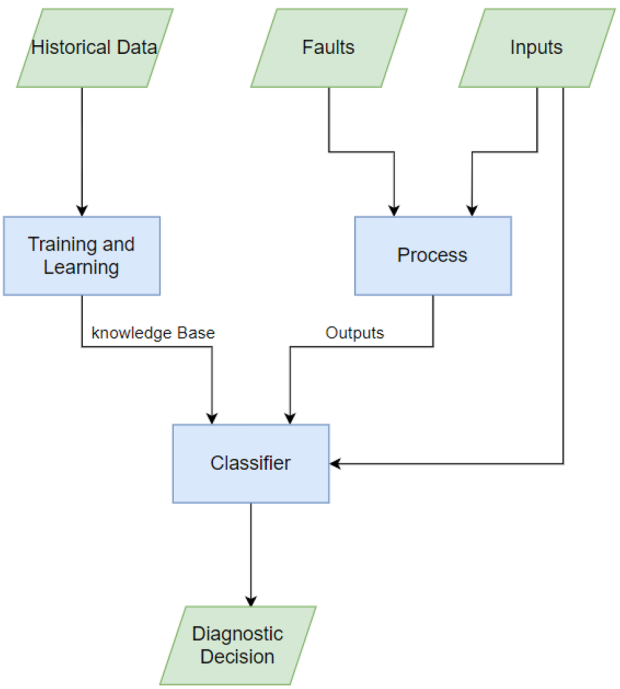


Fig. 5 Schematic diagram of knowledge-based fault diagnosis

be used to reconstruct the output of autoencoders correctly. Therefore, it is simply used for performing dimensionally feature reduction from a higher dimension to a lower dimension. Autoencoders are considered unsupervised learning algorithm, as their outputs are simply reconstructed data from their inputs or features. it consists of three main parts: encoder, code, decoder which are a fully connected feedforward neural network (ANNs). Code is a single layer with a size of nodes representing the dimensionality of our choice. So, code size is a hyperparameter that we determined before training the autoencoder. Figure 6 illustrates the visual description of the autoencoder. Both encoder and decoder have a similar fully-connected ANN structure. And the main goal is to produce output which is same of the input. Therefore, the only requirement is that the input and output dimensions are typically the same. Mainly, autoencoders have four following hyperparameters that should be set before training.

- *Code size* is the number of nodes in the middle layer. The smaller size of the code layer, the more compression we can get.
- *Number of layers in the encoder and decoder:*The more layers exist in the encoder, the deeper autoencoder can be formed.
- *Number of nodes per layer:* There are many types of autoencoders such as stacked autoencoder where layers are stacked one after another.
- *Loss function:* There are two types of loss function which set according to a type of input data.

For binary inputs (like our case), the loss function is the *cross-entropy* described in Eq. 1 (more precisely: sum of Bernoulli cross entropies). Note that x_k is the input data, $\hat{x}_k$ is the reconstructed data and k is the number of samples.

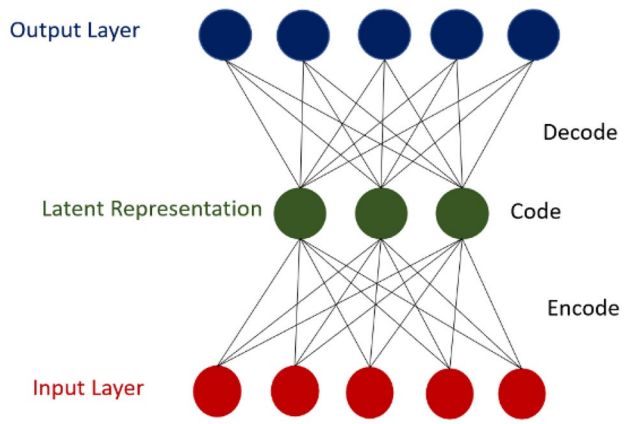


Fig. 6 A simple architecture of autoencoder

$$L(x,\hat{x}) = \sum_k (x_k \log(\hat{x}_k) + (1 - x_k)\log(1 - \hat{x}_k)) \quad (1)$$

For real-valued input, the loss function is sum of squared differences (squared Euclidean distance) and the output should be a linear activation function as follows:

$$L(x,\hat{x}) = \frac{1}{2}\sum_k (\hat{x}_k - x_k)^2 \quad (2)$$

The training of autoencoders is Not different from ANNs where parameter gradients are obtained by backpropagating the gradient like a regular network. The architecture of autoencoders can be handled to form powerful autoencoder by rising nodes per layer, code size and number of layers. Therefore, autoencoder can learn complex codings by increasing these hyperparameters and avoiding overfitting at the same time. Therefore, it should be important to balance between its sensitivity to the inputs good enough to build an accurate reconstructed output and its insensitivity to the input good enough to avoid memorizing and overfitting the training data. Therefore, the loss function of the model $L(x,\hat{x})$ can be defined by two parts: one term for satisfying the sensitivity to the input and the other term for avoiding memorization/overfitting which is called regularizer.

A couple of important properties of dimensionality reduction using autoencoders:

– Data-specific:
  autoencoders are capable of compressing data that is similar to training data. Therefore, autoencoders are Not as standard data compression (like gzip) as the features learned by autoencoders are specific to the given training data.
– Lossy:
  autoencoders are not the way for lossless compression as the output of autoencoders will be close to the input but not the same (it is degraded representation).
– Unsupervised or self-supervised:
  as the training process of autoencoders does Not need any explicit labels for the input data, autoencoder is considered an unsupervised learning technique. Also, they can be called self-supervised because they can produce their labels from the training data.

The main goal of autoencoders is to extract the meaningful features of raw signals and reconstructing them again at the output layer and avoid copying from input layer to hidden layer. Therefore, there are many types of autoencoders to guarantee this property such as sparse autoencoder (SAE) [13], deep autoencoder [14], denoising AE [15] and contractive AEs [16]. There are many types of autoencoders. Table 2 illustrates the main differences between two used autoencoders in our model: sparse AE (SAE) and Deep AE.

Table 2 Description of Sparse and Deep Autoencoders

Type of AE	Definition	Advantages	Drawbacks
Sparse Autoencoder (SAE)	- It has hidden nodes greater than input nodes -Sparsity is obtained by additional terms in the loss function during the training process (either by comparison between the probability distribution and low desired value or by zeroing all but the strongest hidden unit activations manually)	- Preventing overfitting by applying sparsity penalty on the hidden layer in addition to the reconstructed fault - Preventing autoencoder to use all of the hidden nodes - Forcing a lessened number of hidden nodes	- The individual activated nodes should be data dependent - Different inputs will activate different nodes through the network
Deep Autoencoder	- It consists of two identical deep belief networks for encoding and decoding -It uses unsupervised layer by layer pre-training for this model - Layers are the building blocks of deep-belief networks	- The final layer of encoding is fast and compact - It can be used for datasets with real-valued data	- Overfitting may be occurred as a result of high parameters other than input data - Lower learning rate lead training data being a nuance

3 State of the Art for Fault Debugging

Many researches have been devoted to diagnosis and detect faults in digital systems. Most advanced methods are based on SAT solvers or machine learning or deep learning. The following two subsections discuss different approaches for fault detection based on machine learning or deep learning and fault detection and localization based on SAT solvers.

3.1 Fault Detection Algorithms based on AI

The first famous system of data-driven fault diagnosis was revealed in 1980s using expert system [6]. It depends on a set of rules from the past prior experience learned by experts. In [7], authors examined the recent benefits of SVM for fault diagnosis process. In [8], fuzzy genetic algorithms were developed for detecting failures in aircraft automatically.

On the other hand, deep learning becomes an essential topic in machine learning field since 2006. Many researchers have focused on getting benefits of different types of models based on deep learning models for improving accuracy and time consumed of fault detection process. In [9], authors used recurrent neural network and dynamic Bayesian modelling for detecting faults in induction motors. Also, stacked autoencoder (AE) was studied in [10] for fault classification of the induction motor. In addition, unsupervised two-layer neural network using the sparse filtering method was proposed in [11] for fault diagnosis. In [12], authors proposed health state identification method for the fault diagnosis of rotary machinery based on the stacked denoising autoencoder. A deep belief network (DBN) was developed in [13] for intelligent fault diagnosis based on autoencoder.

Recently, feature extraction can be achieved using many algorithms of artificial intelligence. In [14] support vector machine (SVM) is utilized for detecting faults. SVM is one the most common ML algorithms that can be used for both regression and classification (but it is widely used in classification). It attempts to find a hyperplane in an N-dimensional space that can classify the data points where N is the number of features. But the main drawback of this method is nonlinearity, local minimum and sample size problems. In [15], fuzzy C-means clustering algorithm have been utilized for dividing the fault pattern space into small sub-spaces. In [16], fault classification was implemented using a global two-layer backpropagation. Also, authors in [17, 18] have proposed the fault diagnosis approach using the multiclass relevance vector machine and random forest. But the main drawbacks in this method is a long time consumed for achieving good results.

In [19] authors proposed a deep transfer learning (DTL) for fault diagnosis using a three-layer sparse auto-encoder for extracting features of raw data and implements the maximum mean discrepancy (MMD) for minimizing the variance penalty between features in training data and test data. Fault diagnosis task is considered a supervised learning problem as the goal is to correctly find which fault class each fault belongs to. Autoencoders are especially beneficial in extracting nonlinear features for unsupervised learning from various types of data [20–22]. LSTM is suitable for complex sequential problems as it is more efficient to learn the long-term dependencies of unknown lengths of time series data using nonlinear gating functions [22, 23]. In [24] a prediction of circuit complexity can be performed using recurrent neural network (RNNs) models. In this method, RNN takes the number of variables and the number of min-terms as inputs. Then it produces the number of nodes. This procedure can learn from the Boolean function of a circuit for 2 to 14 variables with an overall fault of less than 1%. The benefit

of this approach is that a single NN can be used for a wide range of variables.

In [25], a combination of autoencoder and long short-term memory (LSTM) is introduced for detecting rare fault events and classifying different types of faults, respectively. The autoencoder is utilized with offline normal data as anomaly detection. Then the predicted faulty data detected by autoencoder are passed into the LSTM network to identify the types of faults. Therefore, this method exploits the power of autoencoder in strong low-dimensional nonlinear representations for detecting rare events and the strength of LSTM in time series learning ability for the fault diagnosis. In this approach, the proposed network begins with a sequence input layer of the multivariate time series samples. Then autoencoder analyzes the time series data using the concept of anomaly detection to detect rare events. Then, once the autoencoder detects a fault, the dependencies between various time steps of sequential data can be learned by the LSTM network to identify the types of faults.

3.2 Fault Detection Approaches Based on Satisfiability

One of the most fundamental problems in computer science is the propositional satisfiability problem (SAT) that can be solved using one of SAT solvers [17–19]. The advances on SAT solvers [19, 20] lead to give tremendous solutions on many EDA applications such as hardware and software verification, test pattern generation and combinatorics by expressing them in SAT problem. Generally, SAT problem is an NP-complete problem [21] that checkable certificate in any context can be considered as attempting to find a satisfying assignment of a propositional formula.

Digital circuits can be formulated as SAT instances which are propositional logic built using true (1), false (0), variables, negations, conjunctions and disjunctions. A model of a formula is an assignment of Boolean values to its variable performing the formula to 1. Therefore, a model is known as satisfying assignment. Many forms can be used to express SAT formulas. A satisfiable formula is conjunction normal form (CNF) which is a conjunction of disjunctions of variables or their negated. Every conjunction is called a clause and each variable or its negated is called a literal. Every CNF formula is satisfiable if and only if every clause has at least one literal mapped to one '1'.

Verification of digital circuits can be performed by combining test patterns with their golden outputs and the equivalent SAT instance then calling SAT solver. If the formula is satisfiable, it means that the designed digital circuit has no logic fault. Otherwise, the combined formula is unsatisfiable where we want to find a proof for falsifiability. There are many sound and complete proof systems for propositional logics such as resolution proof which is a standard choice for formulae in CNF (it has only a single inference rule).

Some researches focused on giving IC designers more precise information about logic faults by resolving the maximum satisfiability (MAX-SAT) problems which is the optimization version of SAT problem. This is considered the most crucial field for recognizing valuable diagnostic information about faults in digital VLSI circuits. In MAX-SAT approaches, three main subsets of clauses are generated in case of unsatisifiability equation which are Maximum Satisfiability Subsets (MSS), Minimal Correction subsets (MCS) and Minimal Unsatisfiable Subsets (MUS). These subsets are required for giving detailed information about the location of faults and even sometimes possible corrections [3, 4, 22]. For example, if we have a following unsatisfiable equation.

$$\varphi = (a) \vee (\neg a) \vee (\neg a \wedge b) \vee (\neg b) \tag{3}$$

The following definitions explain the difference between all subsets which are used in SAT-based Fault detection and diagnosis.

Definition 3. (Maximal Satisfiable Subset) A subset **S** of an unsatisfiable CNF formula $\boldsymbol{\varphi}$ is an **MSS** if $\mathbf{S} \subseteq \varphi$ is satisfiable subset where $\forall_{c_i \in (\varphi \setminus S)}$, $(\boldsymbol{S} \bigcup \{c_i\})$ is unsatisfiable.

Definition 4. (Minimal Correction Subset) A subset C of an unsatisfiable CNF formula φ is a correction subset if $(\varphi - C)$ is a satisfiable subset. A correction subset can be considered MCS if $C_{new} = \varnothing$, where $C_{new} \subset C$ and $(\varphi - C)$ forms an MSS.

Definition 5. (Minimal Unsatisfiable Subset) A subset U of an unsatisfiable CNF formula φ is an MUS if $U \subseteq \varphi$ is unsatisfiable subset where,$\forall_{c_i \in U}$, $U - \{c_i\}$ is satisfiable.

Figure 7 describes an example of finding **MSSes**, *MCSes* of a CNF formula φ (Note: C_i indicates a clause i).

In SAT-based approaches, MCSes and MUSes have been used in functional debugging and diagnosing corresponding. This can be done by mapping every clause in

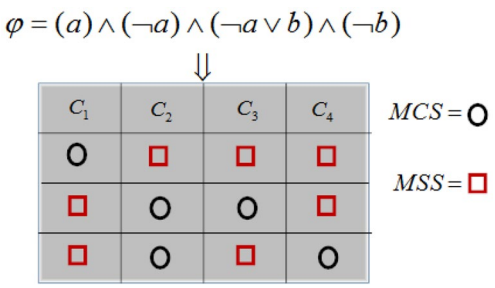


Fig. 7 Example of finding MSSes, MCSes

subsets to the corresponding faulty component in a given circuit. Most of approaches based on MCSes and MUSes follow these steps with its own improvement procedures:

- Every gate in the circuit is transformed to the equivalent CNF equation according to number of inputs. In [6], we show the equivalent CNF equation for every logic gate.
- The equivalent CNF equation of a given circuit is combined to form conjunction of clauses.
- Test patterns with correct outputs are combined to the equivalent CNF formula of a given circuit. The following equation is generated

$$\varphi = I.O_c.CNF(C_f) \tag{4}$$

- The final CNF equation is considered to be unsatisfiable, so number of MCSes and MUSes should be extracted to find all possible faulty components.

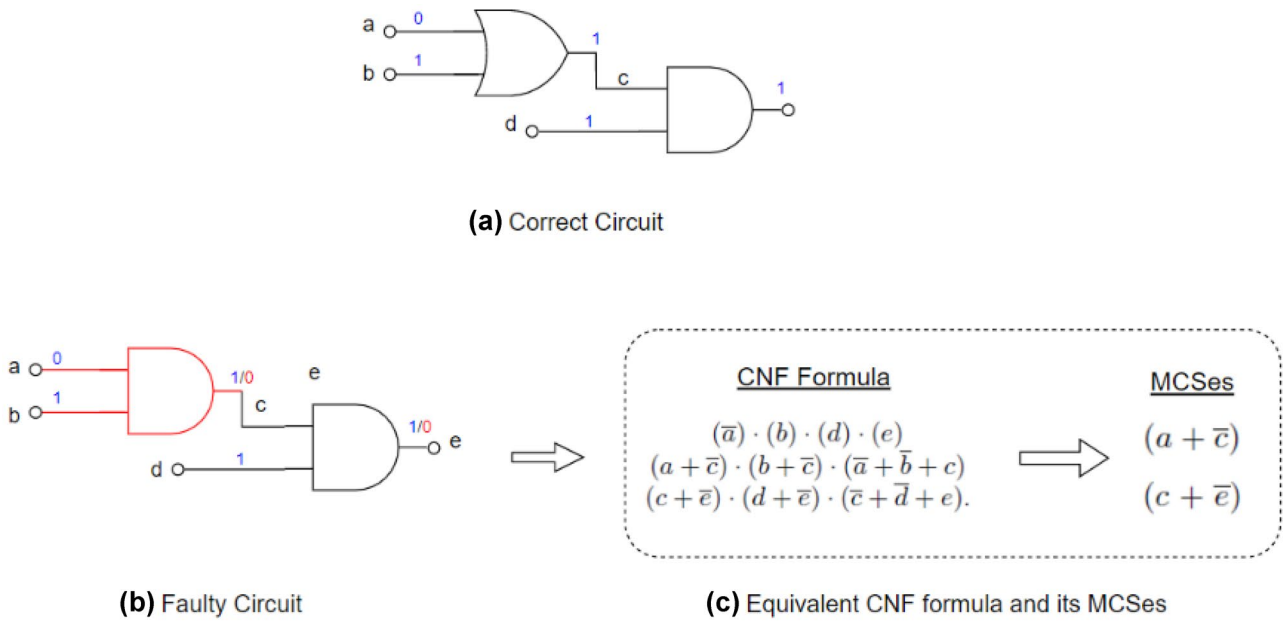
Figure 8 shows an example of the correct circuit (a) and faulty circuit with gate replacement (b), the equivalent CNF formula of faulty circuit and the corresponding minimal correction subsets (MCSes) (d).

Therefore, many developed techniques have been proposed for MUSs detection [23–27]. Also, many approaches have improved and proposed new representations and applications using this concept such as: debugging declarative specifications [28, 29], infeasibility-based maximum satisfiability problem (MAX-SAT) and detecting minimal strongly unsatisfiable sets (MSUS) for reactive specification systems [30]. In Table 3, a brief illustration of the most common approaches for fault detection process based on SAT in MCSes or MUSes.

As the neural network has increased scope for solving various problems. In [33], a simple neural network architecture (NeuroSAT) is proposed to perform a discrete search after end-to-end training for avoiding hard-coded search procedures. NeuroSAT is a novel GNN that is trained as a classifier to predict satisfiability on a dataset of random SAT problems using a single bit of supervision (indicating whether or not the problem is satisfiable). NeuroSAT can solve SAT problems that are more difficult than those used in the training stage using hundreds of thousands of iterations of message passing.

Also, the same neural network proposed in [33] can be used to find proofs for unsatisfiability, calling NeuroUNSAT which is trained on different unsatisfiable problems and contains unsat cores. NeuroUnSAT learns to detect these unsat cores from NeuroSAT's activations. NeuroSAT model encodes a SAT instance as an undirected graph where clauses and literals are represented by nodes. Clauses and literals have vector space embeddings. And edges are connections between clauses and their literals and connections between each pair of complementary literals. Then the network attempts to refine a vector space embedding for each node using message passing along edges of the graph. But the main problem of NeuroSAT is still vastly less reliable than traditional SAT solvers.

The main problem of SAT-based Fault detection is multiple SAT solver calls. In SAT-based methods, the unsatisfiable equation shown in Eq. (4) are generated where I



(a) Correct Circuit

(b) Faulty Circuit

(c) Equivalent CNF formula and its MCSes

Fig. 8 Example of Correct circuit, Erroneous circuit, Equivalent CNF and MCSes

Table 3 Review of some Approaches of Fault Detection based on SAT domain

Method	Description
Destructive Approach [9, 31]	- It is iteratively excluding irrelative clauses on infeasibility of a given CNF instance - The best extraction of MUS requires O(m) where m is the number of clauses in SAT instance - Extensive consuming time for finding MUSes, especially with large digital circuits
CAMUS Approach [26]	- Computing all minimal correction subsets (MCSes) - Searching for MUS into the complete set of MCSes - No Extra calls for SAT solvers for finding MUS
ExcludeMUS Routine [9]	- Based on the previous method - Enhancing MCSes enumeration for improving MUS generation - Can handle a large amount of MCSes - Some parts of codes are implemented on multicores for eliminating the negative impact of large sized instances (Complex Circuits)
Shrink Routine [32]	- A recursive method of computing MUSes directly from the input formula - Critical constraints are the main core of this process - No need to compute MCSes - The number of calling SAT solver is reduced then the destructive algorithm to be $O(\|\varphi\| - \|crits\|)$ - Still repetitive process of satisfiability check is the hardest part
Improved Shrink routine [9]	- It is based on the previous method for avoiding time and cost of computing the complete set of MCSes - It tries to reduce the number of SAT solver calls for computing MUS - It performs two processes: classification and reduction processes consequently for reducing the number of calls of SAT solvers - It speeds up the computation using GPUs algorithms

represents test patterns and O represents the corresponding correct responses and $CNF(C_f)$ represents the equivalent equation of a given erroneous circuit in a conjunction normal form. Therefore, the search space is increased by growing number of lines and gates. In general, SAT-based method attempts to find the specific conjunction of clauses that causes unsatisfiability as shown in Fig. 6. Therefore, the search space is huge to find the clauses related to the error inside large number of clauses. Our proposed solution to this problem is to take advantages of deep learning model by the following:

1. Instead of encoding every gate to its equivalent CNF equation, the fault list file can represent all possible faults.
2. Instead of injecting debug circuits like multiplexers on every line for checking faults. Our model can learn from fault masks as labels for every test pattern to find all equivalent faults.
3. Instead of solving the CNF equation (which is in large size after injecting debug circuits) by one of SAT solvers, our model can learn the connections between lines from the data set which contains test patterns with corresponding correct outputs and the corresponding fault mask.

4 The Proposed Implementation

This section illustrates the proposed model for fault detection of digital circuits. In this section, we propose a new method for detecting all potential faults in a circuit for solving the problem of:

– Tracking the effect of all test patterns and their corresponding correct outputs on all lines on the circuit for finding the location of faults or the root cause of errors.
– Solve the problem of search space process which make the fault detection and diagnosing more complex for large-sized digital systems.

In our application, our problem is handled as a matrix (X) of a dimension N*M where N represents the number of samples and M represents the set of primary inputs and primary outputs of a given circuit. Each x_{ij} corresponds to a Boolean value of (j) input in sample (i). The proposed architecture consists of two main phases a dimensionality reduction phase and a classification phase. The proposed model mainly contains two main sequential processes: unsupervised feature extraction model and semi-supervised model. The following subsections discuss each step in details.

4.1 Dataset Description

In our application, the input data or features are test inputs and their golden responses for each digital circuit. So, the whole model depends on the amount of training data and how much it is unique in giving the network the ability to learn different features of digital circuits. Therefore, "ATALANTA" tool [34] is utilized in this phase for preparing data that represents digital circuit in bench format. ATALANTA is an automatic test pattern generator for stuck-at-faults which employs FAN algorithm for test pattern generation and parallel pattern single fault propagation technique for fault simulation. Also, it can produce the fault mask for every pattern for the consequent processes. Therefore, specifying/identifying/detecting the whole digital circuits using conjunction normal form of circuits CNF(c) can be avoided to reduce complexity specifically for large-sized circuit. This can be achieved by defining features as all inputs and output pins of digital circuits with the complete test patterns which can detect all stuck-at-faults as features of our model. Then, our model learns from these data to detect any fault occurred in the digital circuit, by knowing only the inputs and output of digital circuits. We collect data of best test paths using ATALANTA tool which is shown in Fig. 9 at the output file in the second and third column as the best test inputs and golden responses of a given circuit ("C17" in this example). Using these types of data, we can cover all input paradigms and avoiding unnecessary repetitions.

4.2 Unsupervised Feature Dimension Reduction

This process is the first phase of our proposed model and the goal of this phase is to extract a new feature dimension representation which is more accurate in multi-class fault diagnosis. For large-sized digital circuit, extracting the main reasons for possible faults is considered a challenging process which needs a high accuracy for avoiding new faults or escaping dominant faults. In proposed semi-supervised model, autoencoder is proposed to learn the lower dimensional representation for test patterns and their corresponding correct responses which can be used in creating a model for faults classification. Therefore, this procedure can give the following advantages:

- With the complexity of line connections, stuck-at fault detection requires best representation for inputs and outputs for detection. Therefore, AE is used for reduce the complexity of values with high accuracy for reconstructions.
- A higher validation accuracy means that AE can reconstruct test patterns better by finding the important input values and observing stuck-at faults. Therefore, our model can extract the starting points of all potential stuck-at faults (the root cause of faults).

As we attempt to detect faults on a given circuit without implicitly injecting the structure of a circuit due to its large size, we use test patterns with its correct responses and their fault mask for forming a model that can approximate the connections between lines which are faulty lines due to single stuck-at fault. Therefore, we use autoencoders as the first phase of our model to learn a lower-dimensional representation (encoding) for a higher-dimensional data. This phase can capture the most important part of the Boolean values of all primary inputs and their primary outputs which can be used in the next classification phase. In other words, the model attempt to extract the important primary inputs and outputs and remove all other inputs or outputs that are not important in detecting a specific fault mask. This can be useful due to the following reasons:

- The used data set are test patterns for primary inputs and corresponding correct outputs for primary outputs corresponding to a specific fault mask.
- In our proposed model, we used the trained encoder layers and bottleneck layer of the autoencoder for the next classification phase. Therefore, we need to train the proposed autoencoders to find the appropriate weights on encoder layers and code layer for reducing the unimportant features (noisy data).

So, the validation accuracy of the autoencoder means that the autoencoder can extract the most important part of test patterns and faulty-free responses of a circuit for detecting all possible stuck-at fault. These parts are used in the next phase of classification. Therefore, the autoencoder

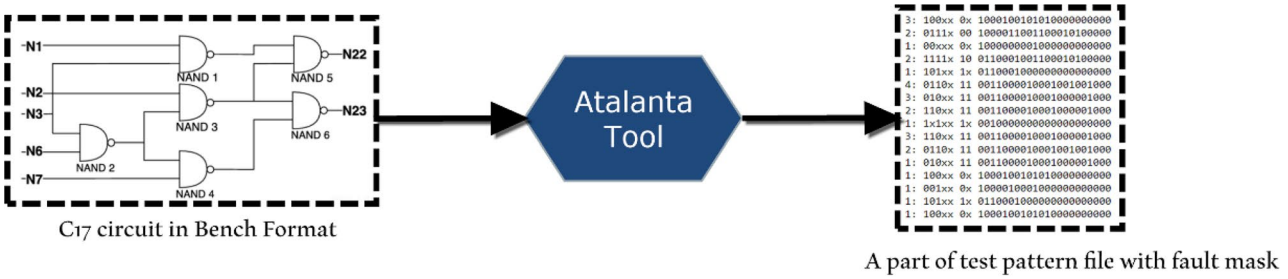


Fig. 9 An example of preparing data of C17 circuit

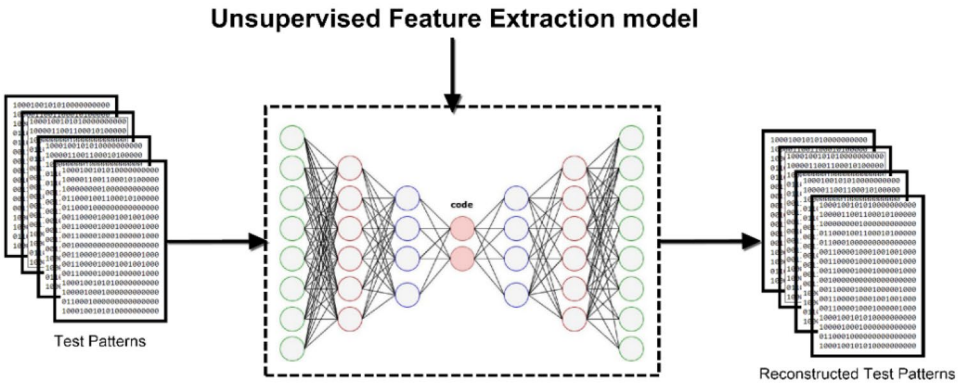


Fig. 10 ETPs Extraction Process

stage can shrink the search space of faults inside a circuit by reducing the starting points of all possible paths.

By increasing size of most digital circuits, the number of test patterns are increasing exponentially. Therefore, multiple types of autoencoder can be used in this phase as an unsupervised learning algorithm to accomplish feature dimension reduction. Instead of passing the all-Boolean values for all inputs and outputs of a circuit, a meaningful feature representation of test patterns can be extracted using autoencoders. In our application, features are both test patterns and their corresponding faulty-free outputs for every digital circuit. We have used different types of autoencoders to find the best one for this application: normal AE, stacked sparse autoencoder (SSAE) and Deep Sparse autoencoder. Therefore, the meaningful inputs and outputs can be extracted from all test patterns for helping model to detect faults quickly and solve the search space explosion problem occurred in traditional fault detection methods especially faults occurred in large-sized digital circuits.

Autoencoder networks have been built to learn how to transform Boolean values of inputs and outputs into a compressed essential pattern by minimizing the reconstruction fault which measures the difference between the original input patterns and reconstructed patterns. Therefore, the bottleneck shown in Fig. 10 is the key attribute of this network for avoiding simple memorizing input data.

In our model, we utilize stacked sparse autoencoder (SSAE) which is one of classical variants of the traditional AE to improve the performance. It consists of several stacked autoencoder (SAE) layers and its goal is to avoid reduction of hidden nodes in hidden layers by activating a small number of neurons. This introduction of bottleneck without reduction can be achieved by imposing a sparsity constraint on the hidden units and constructing loss function by penalizing activation within hidden layers. In other words, it learns the relatively sparse features by penalizing the hidden unit biases.

Informally, if the output of neuron close to 1, the neuron will be "active". Otherwise, the neuron will be "inactive" (In case of using sigmoid activation function). In SAE, most of neural in the hidden layer is inactive if the average activation of the hidden unit ρ_k (k = 1, …, s) (shown in Eq. 5) is close to zero where the dataset denotes to x_i, n is number of samples, s_f is the nonlinear function in the encoder network.

$$\rho_k = \frac{1}{n}\sum_{i=1}^{n}[s_f(b_k + W_{ik}x_i)] \tag{5}$$

There are two ways for imposing a sparsity constraint for penalizing activation within hidden layers that are L1-Regulaization and Kullback–Leibler (KL) Divergence. Generally, each method depends on measuring average activations of hidden neurons and adding some penalty term to loss function for controlling excessive activations.

In L1-Regulaization, the absolute value of the activation vector in the hidden layer for observation training batch is added to loss function $L(x_i, \hat{x}_i)$. This value is scaled by tuning parameter λ as the following equation:

$$C(\theta) = \sum_{i=1}^{n} L(x_i, \hat{x}_i) + \lambda \sum_{i=1}^{n} |s_f(b_k + W_{ik}x_i)| \tag{6}$$

where $L(x_i, \hat{x}_i)$ is loss function between input values x and the reconstruction ones $\hat{x}$, m is the dimension of each input sample and θ is the parameter set $\{W, b, W', d\}$.

In KL-divergence, to penalize ρ_k in Eq. 5, the sparse penalty term is expressed by deviating from a predefined sparse parameter ρ. These deviations can be measured by Eq. 7 which attempts to hold the following property: $KL(\rho||\rho_k) = 0 at \rho = \rho_k$.

$$KL(\rho||\rho_k) = \rho log\frac{\rho}{\rho_k} + (1-\rho) log\frac{1-\rho}{1-\rho_k} \tag{7}$$

In order to prevent overfitting, a regularization item is added to cost function $C(\theta)$ as shown in Eq. 8. The penalty coefficients of the sparse item are β and γ, respectively.

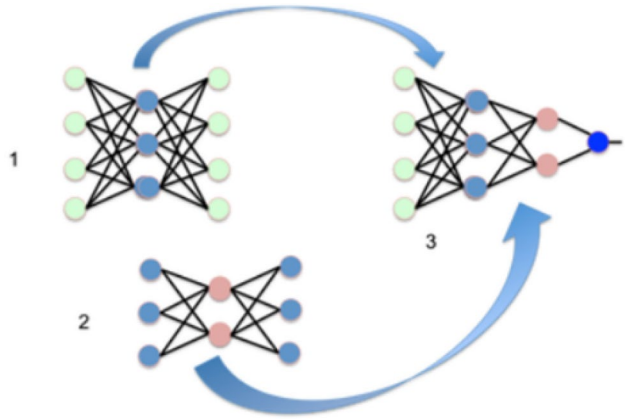


Fig. 11 Deep Autoencoder

$$C(\theta) = \sum_{i=1}^{n} L\left(x_i, \hat{x}_i\right) + \beta KL(\rho||\rho_k) + \frac{\gamma}{2} \sum_{i=1}^{m} \sum_{k=1}^{s} (W_{ik}^2 + W'^2_{ki}) \tag{8}$$

Therefore, minimal extracted features can be utilized for enhancing the performance of the proposed model and balancing between validation accuracy and consumed time.

Using Stacked Sparse Autoencoder (SSAE), we have used three SAE1 takes the features vector S of the matrix X of range M, and gave it to the encoder, in the bottleneck layer. Therefore, a new latent space $LS1$ of range$K$, where $k < M$ is generated. According to this latent space, the decoder attempts to reconstruct the input S as close as possible at the output of the decoder for$S \approx S^{'}$. Then, the output ($S^{'}$) of SAE1 becomes the input to the next SAE2 for more reduction of features. And the same steps are followed to generate a latent space ($LS2$) and decoder tries to reconstruct $S^{'}$ at the output of decoder $\left(S^{prime}\right)$ where $S^{prime} \approx S^{prime}$. Then, the last SAE3 is used using the output of SAE2 as inputs of SAE3 for more reduction. The last latent features space vector ($LS3$) is generated using the bottleneck of the third sparse autoencoder. The final architecture settings of autoencoders and each autoencoder have been evaluated by calculating the reconstruction error loss between the input of the encoder and the output of the decoder for each SAEi. We have used the binary-crossentropy loss function (Eq. 1). Every autoencoder is separately trained as unsupervised stage by minimizing its reconstruction errors. When all layers are trained, the network can be passed to supervised stage (stage 3 in Fig. 11). The concatenation between three generated features [$LS1, LS2, LS3$] from each SAE is performed in one feature vector ($LS4$) which is used to train the classifiers.

4.3 The Complete Semi-supervised FD Model

After building Deep autoencoder that can learn a concise representation of test patterns, the learned encoder layers is connected to a classifier layer for building the full complete model (as shown in Fig. 12), for learning how to classify between multiple faults occurred in this current used circuit. Therefore, a classifier is considered the last layer in our proposed FD model where the number of classes in this layer equals to the number of faults recognized in the first phase. This process is an alternative approach than searching for unsatisfiable subsets (*MUSes or MCSes)* in falsified constrains used in SAT fault detection methods. After this, we have tested the model using unseen test patterns and stuck-at-faults for improving accuracy. In this phase, we have used

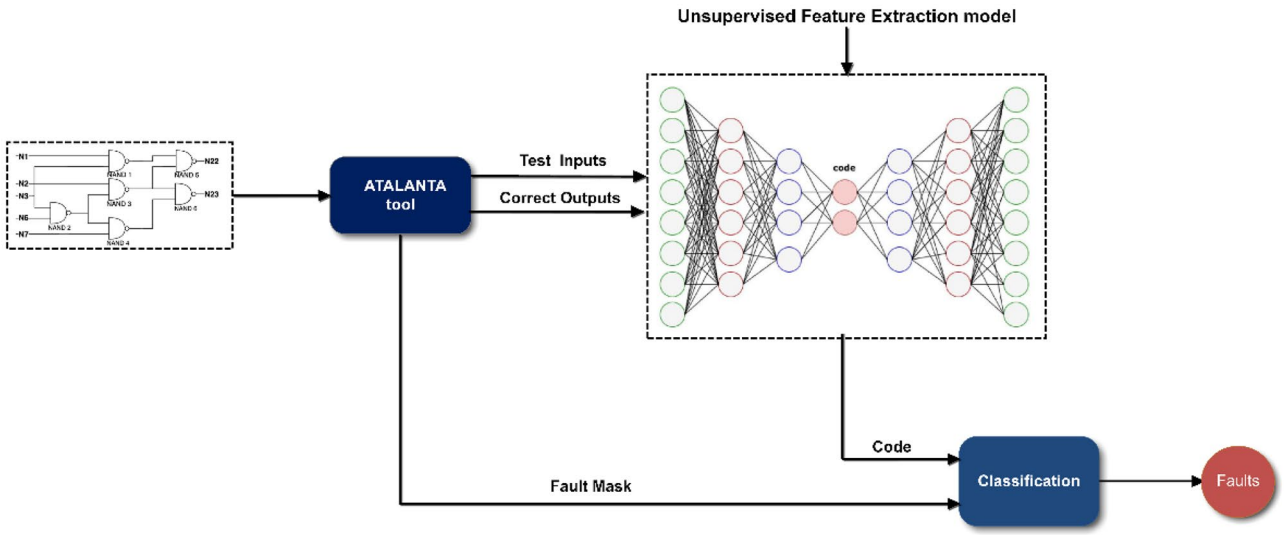


Fig. 12 The proposed semi-supervised FD model

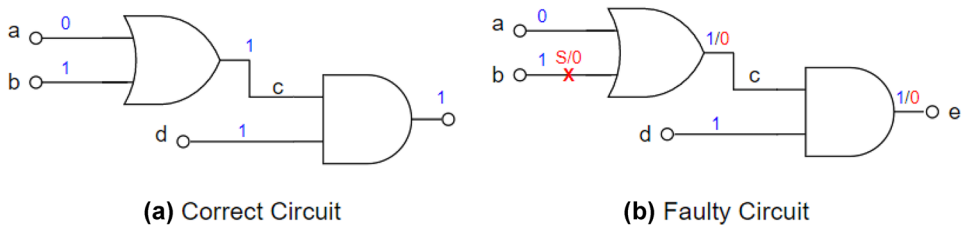


Fig. 13 Simple example of correct Circuit **a** and faulty circuit **b**

three different classifiers to predict the fault lines. The following steps are following:

- The concatenated training layers of encoders are combined with binary classifier with sigmoid activation as the output layer and use the binary cross-entropy as loss function. The labels of this supervised learning are the fault mask.
- Also, three different classifiers in machine learning models are used which are Decision Tree (DT) and Random Forest (RF) and Gradient Boosting (GB) classifiers.

4.4 Example

A Data Set Generation

In this section, we explain the proposed model step by step applying on a simple circuit called "cExample". This model requires circuits in bench format so our example called "cExample.bench" which contains two gates (2-input OR gate and 2-input AND gate) as shown in Fig. 13. We assume that we have stuck-at zero in line (b).

In this phase, we attempt to form a model which can generate all possible stuck-at faults for each test patterns. Therefore, ATALANTA is used to generate our data set which are test patterns and Fault mask of each pattern. According to this data, our model attempts to follow two main phases: *unsupervised learning phase* for extracting the important main features for classifying faults using Autoencoders. Then, *Semi-supervised learning phase* for classifying faults. Classification as we previously mentioned is performed as binary classification which can detect faulty lines and faulty-free lines.

First of all, we generate fault list file called "cExmaple.flt" of a given circuit (see Fig. 12) by assuming all possible stuck-at faults. In ATALANTA tool, we can generate Fault List (FL) using the following line:

```
atalanta.exe -F cExample.flt cExample.bench
```

In Fig.14, Stuck at faults in fault list can be written as follows: line-name/0 or line-name/1. For example b/0 means line "b" has stuck-at zero. Then, fault mask file is generated using ATALANTA tool for each test patterns. Therefore, the size of Fault mask is matrix (N * L) where N equals to

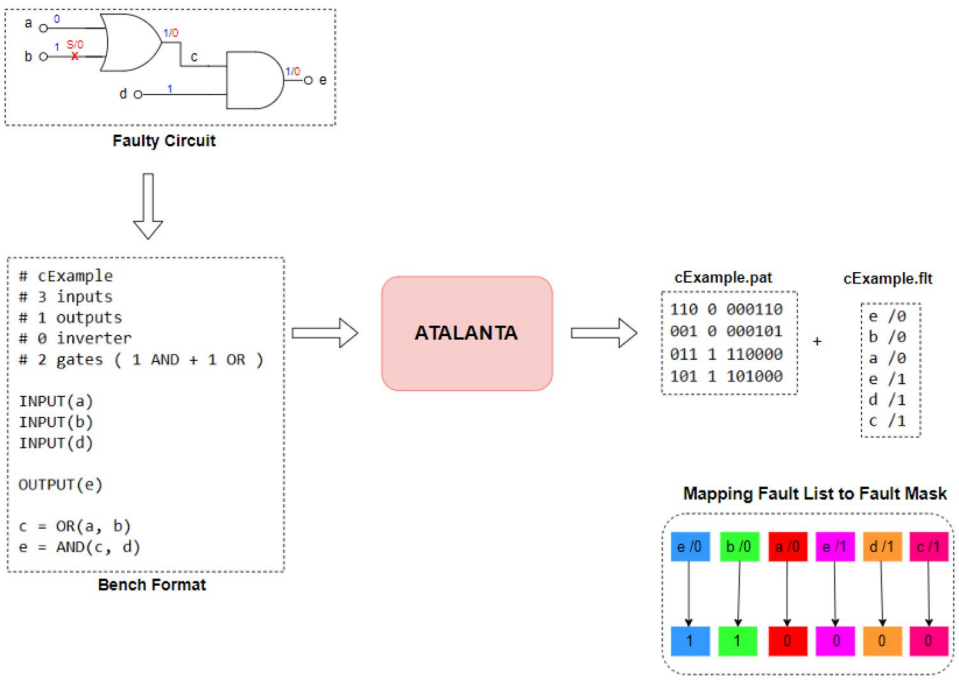


Fig. 14 Representation of Data set for a simple given example in Fig. 13

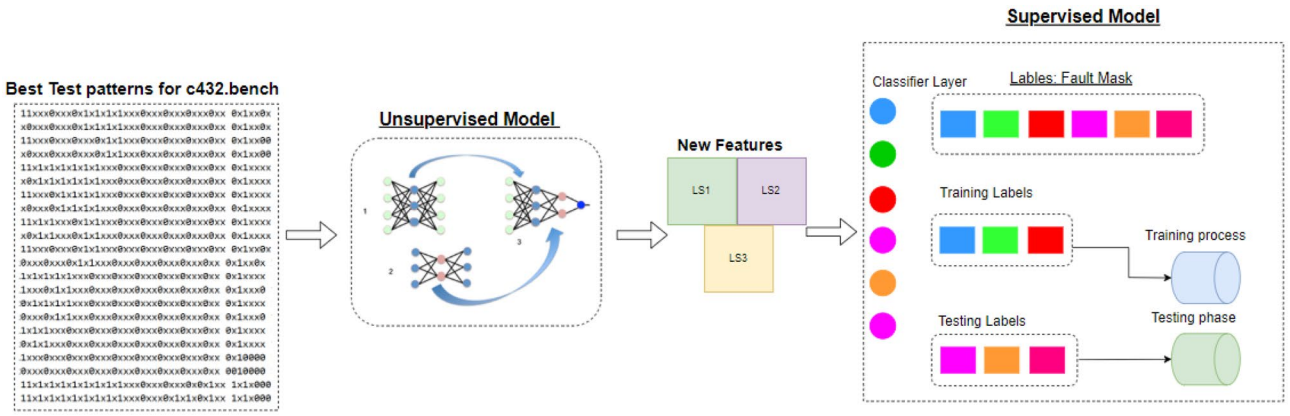


Fig. 15 visual explanation of unsupervised phase and supervised phase

number of test patterns and F is number of possible faults in fault list. In our example, Fault list assumes to be (e/0, b/0, a/0, e/1, d/1, c/1). So, fault mask may be [110000] which means that both line "e" and "b" are fault lines and all other lines are not faulty lines. The following command can generate all test patterns with fault mask on the same file called "cExample.pat" in this example.

```
-F cExample.flt -t cExample.pat -W 4 cExample.bench
```

Or the following command if we have a specific fault list called "cExample.flt".

```
-f cExample.flt -t cExample.pat -W 4 cExample.bench
```

In order to generate as more as possible data points for reducing loss errors and avoid underfitting problems during learning. We can generate multiple test patterns for each single fault type which can give our model the accurate connections between fault lines and test patterns without injecting the implicit structures of a circuit. This is can be achieved by adding "-D #N" to the previous command, where #N is number of test patterns.

```
-F cExample.flt -t cExample.pat -W 4 cExample.bench
```

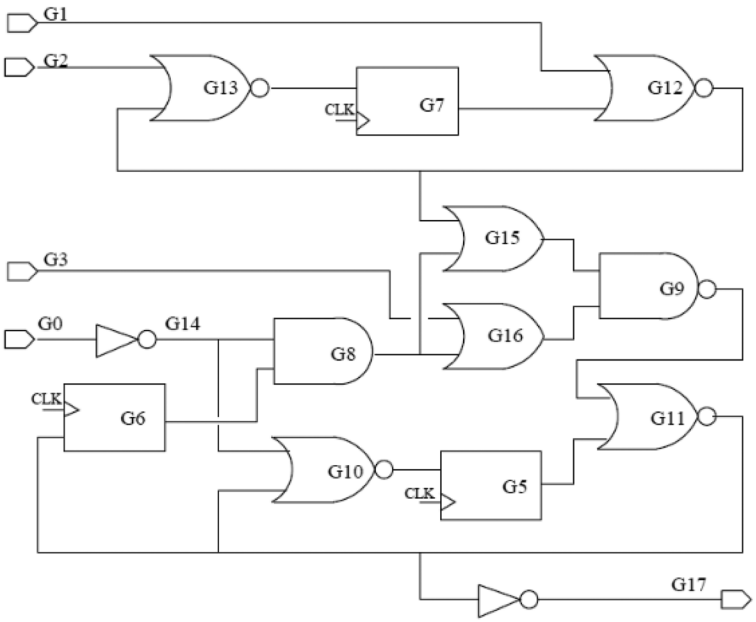
A Semi-supervised Learning Model

In this step, we create an unsupervised model using autoencoders for dimensionality Reduction of our features. The main problem of using deep learning for fault detection using data driven methods as the complexity of approximating a function of fault classification where multiple inputs and outputs. The data set of this phase are test patterns which are generated at previous step. This step is required for extracting new feature space for effective fault detection. As we explained in Sect. 4.2, we utilize multiple stacked sparse autoencoder and deep autoencoders for balancing between the quality of the new features by detecting the validation performance of Autoencoder and reducing the consumed time for fault detection. In Fig. 15, unsupervised model is created using test patterns with faulty-free responses for extracting new rich features which can be used in the consequent classifier. Every Stacked Autoencoder is sparse for imposing a sparsity constraint on the hidden units and introducing bottleneck without reduction.

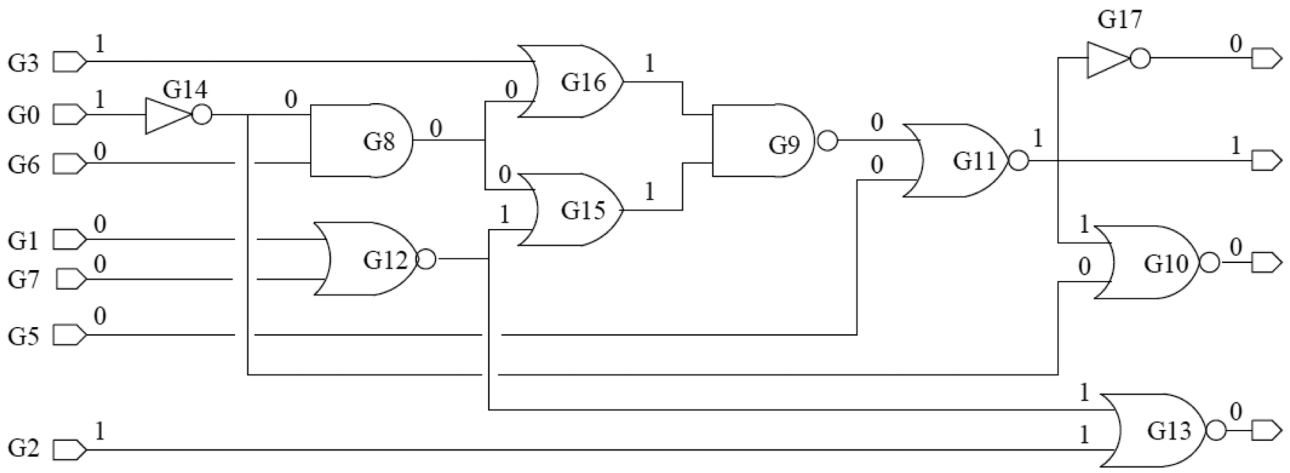
Then, the trained encoder layers for each SSAE are combined together with a binary classifier for the next phase of supervised learning. In this phase, labels are fault mask which can be mapped to the fault lines in fault list for approximating fault detection process. The binary classification can detect every possible line in fault list as faulty line if its value is 1. Otherwise, it is Not a faulty line. These mapping from fault mask to fault list make the model not only can detect the specific faulty line but also it can approximate the connections between the root-cause line and other lines in the circuit that are not the main causes of faults. This proof is concluded according to our assumption of using test patterns for every single stuck-at faults that have multiple negative impact on other lines in the circuit. Using this method, we can avoid the searching problem of finding the minimal correction subsets or minimal unsatisfiable subsets in SAT-based Fault detection which can be an efficient way in large-size circuits. In Fig. 13, we illustrate unsupervised model which is trained using test patterns and their correcting responses for c432.bench circuit which is 27-channel interrupt controller.

Therefore, classification rules in supervised process for feature reduction are measured by the following steps:

- The Data set used on our model are (X,Y) where X matrix represents features of our model which are number of test patterns with corresponding correct output and Y matrix represents Labels which are the corresponding possible fault mask for every test pattern
- The Fault mask can be mapped to the fault list which contains all possible fault lines on a given circuit (see Fig. 12).
- Our main goal is to learn the connection between faulty lines which lead to the faulty outputs.
- Dimensionality reduction of test patterns with golden responses is applied on the best test vectors for given circuit.
- The validation accuracy of Autoencoder is measured to find the best type for dimensionality reduction with best sparsity constraint.
- The trained encoder layers and bottleneck layer is combined with classifier layer which is the last layer on our model.
- The sigmoid function is used in the last layer as an activation layer. And the number of neurons in this layer equals to all possible faults in circuit.
- For training model, optimizer is chosen to be "adam" and loss function is "binary cross entropy" which is explained in page 3 Eq. (1)



(a) The circuit s27 in ISCAS'89 benchmark



(b) The circuit s27 as combinational circuit.

Fig. 16 Example of sequential circuit as combinational circuit

Table 4 Number of inputs and outputs, faults, test patterns and fault coverage of 11 combinational circuit [38]

Circuit	#gates	Sum of #Inputs and #outputs	#Faults	# Test Vectors	Fault Coverage%
c17	6	7	22	54	100
c432	160	43	524	8950	98.8
c499	202	73	758	13,066	95.5
c880	383	86	942	15,357	100
c1355	546	73	1574	25,824	96.8
c1908	880	58	1879	34,216	99.2
c2670	1193	373	2747	39,725	94.1
c3540	1669	72	3428	55,752	95.9
c5315	2307	301	5350	88,978	98.7
c6288	2406	64	7744	114,101	80.8
c7552	3512	315	7550	134,093	94.1

5 Experimental Results

This section shows the experimental results of a proposed algorithm for detecting stuck-at-0 and stuck-at-1 faults in digital circuits. In order to verify the performance of our model, some classic models of machine learning and deep learning were compared with our model such as Decision Tree, Random Forest and Gradient Boosting classifiers in machine learning model and Radial Basis Function (RBF) networks.

The proposed algorithm is used on the ISCAS'85 benchmark [35] for combinational circuits and ISCAS'89 benchmark [36] for sequential circuits. The proposed FD model is implemented using python 3.7.9, keras API [37], the most used deep learning framework built on top of TensorFlow and an open source python library for developing deep learning models. It was executed on intel core i7 10,750 working at 2.60 GHz with 16 GB system memory. Also, some of the training processes is executed on NVIDIA GeForce GTX 1650 Ti using cuDNN toolkit.

The sequential circuit is transformed into a new circuit called "unfolded combinational circuit". Figure 16 illustrates s27 treated as combinational circuits in (b). Therefore, the total ports of sequential circuits are changed, for example s13207.1 from 214 to be 1490 port. Using this method, we avoid the search space explosion for finding faults in sequential circuits.

5.1 Data Preprocessing

From ISCAS'85 and ISCAS'89, every circuit has been passed to ATALANTA software which generates multiple test patterns for each fault. We have selected around 20/50 test vectors for each fault in combinational digital circuit and one vector for each fault in sequential circuits. These features and labels of our proposed algorithm are split into 70% for training data and 20% for testing data. In Table 4, the parameters of digital circuits in terms of number of inputs, number of outputs, number of faults, number of test vectors and fault coverage are illustrated.

5.2 Feature Reduction and Classifier

5.2.1 Extracting Features Using Different AE

In this section, Different models of Autoencoders for feature reduction, which is the second phase of our algorithm, are implemented on both combinational and sequential circuits. Table 5 illustrates four different stacked autoencoder with different sizes of neurons to find the best model for every circuit. From this table, we can find that maximum validation accuracy for dimensionality reduction of test patterns in "c5315" with SAE1 which contains three layers (200,100,50).

Table 5 Different Stacked Autoencoders with different sizes of neurons

CNF Type	SAE1		SAE2		SAE3		SAE4	
	#N	VA%	#N	VA%	#N	VA%	#N	VA%
c17	5,3,2	71.4	6,5,3	69.74	5,4,2	68.06	4,2,1	62.18
c432	(40,30,20)	99%	(30,20,10)	97.5%	(20,10,5)	92.52%	(25,15,10)	96.19%
c499	(60,30,20)	98.91%	(50,30,20)	98.50%	(40,30,20)	98.4%	(40,15,10)	95.78%
c880	(70,50,30)	99.56%	(70,40,20)	98.9%	(50,30,20)	98.6%	(30,20,10)	95.78%
c1355	(60,30,20)	98.1%	(70,50,20)	98.87	(70,30,10)	95.16	50,20,10	94.58
c1908	(50,30,20)	98.8%	(40,20,10)	94.98	(50,30,10)	95.36	50,40,15	96.4
c2670	(200,150,50)	99.94	(200,100,20)	99.82	(150,50,10)	99.55	100,30,10	99.26
c3540	(50,30,20)	98.2%	(60,40,20)	97.99	(60,50,20)	96.55	50,40,15	95.72
c5315	**(200,100,50)**	**99.93**	**(150,50,20)**	**99.79**	(100,50,20)	**99.7%**	**130,30,10**	**99.08**
c6288	(55,35,20)	99.13	(60,30,15)	98.23	(50,30,20)	97.3	55,25,5	96.09

Table 6 Number of hidden neurons and validation accuracy of feature extraction with sparsity constraints [38]

Circuit	# hidden neurons	VA% using simple AE	VA% using Deep AE	Sparsity Constraints
c432	30,20,10	95.4	97.5	10e-6
c499	50,30,20	98.2	98.50	10e-9
c880	50,30,20	98.3	98.6	10e-6
c1355	70,50,20	97.44	98.87	10e-9
c1908	50,30,20	97.8	98.8	10e-6
c2670	200,150,50	**99.9**	99.94	10e-9
c3540	50,30,20	97.3	98.2	10e-6
c5315	100,50,20	98.9	99.7	10e-9
c6288	50,30,20	97.9	99.3	10e-6
c7552	200,100,20	99.5	99.6	10e-9

For combinational circuits, Table 6 shows that the maximum validation accuracy of simple AE equals to 98.9% for "c5315". On the other hand, Deep AE with 10e-9 sparsity constraint outperforms a simple autoencoder in extracting features in terms of validation accuracy around 99.7% for the same circuit. In terms of average validation accuracy, Deep AE achieves around 96.4% where simple AE accomplishes 94.95% using unseen dataset.

Figure 17 visualizes accuracy and loss accuracy during 100 epochs of SSAE model for c2760 circuit. It shows that model can find a good balance between underfitting and overfitting to find the best latent information of any new test patterns that can be efficiently reconstructed to the original data.

The previous results are based on choosing the best value of sparsity constraint (explained in section II.C), for deep and stacked sparse AEs. The effect of sparsity constraint of SSAE for the reconstruction phase is illustrated in Fig. 18 where three values are used (ρ1 = 10e-6, ρ2 = 10e-5, ρ3 = 10e-6) implementing on ten different combinational

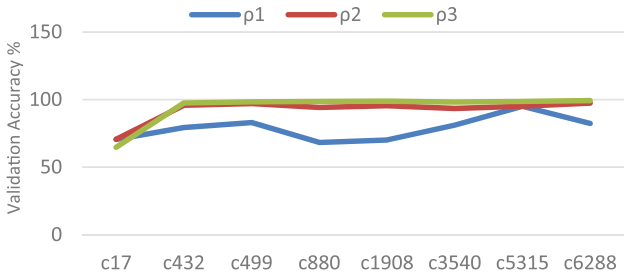


Fig. 18 The effect of sparsity constraints on the accuracy of SSAE

circuits. From these results, we conclude that the best value of ρ for reconstruction is about 0.024 for ALU circuits and SEC circuits, in ISACAS'85 benchmark.

For sequential circuits, we implement Deep Sparse AE and stacked sparse autoencoder on nine sequential design from ISCAS'89, for feature reduction process and compare validation accuracy of reconstruction data using the same architecture (three sparse layers), as shown in Table 7. From the results, SSAE outperforms Deep AE in several sequential circuits in validation accuracy (VA%) about 99.95% and 99.8% respectively in terms of maximum VA%.

5.2.2 The Complete Semi-supervised FD

After combining Deep AE with Classifier layer using sigmoid activation function for multilabel classification as detecting multiple faults in digital circuits, our model is trained on GPU for classifying multiple stuck-at faults in digital circuits, using the latent space extracted from previous step. Table 8 illustrates the validation accuracy of implementing the complete semi-supervised model (Deep AE combined with logistic classifier) on ten combinational circuits from ISCAS'85. From experiments, our FD model delivers around 99.6% maximum validation accuracy for

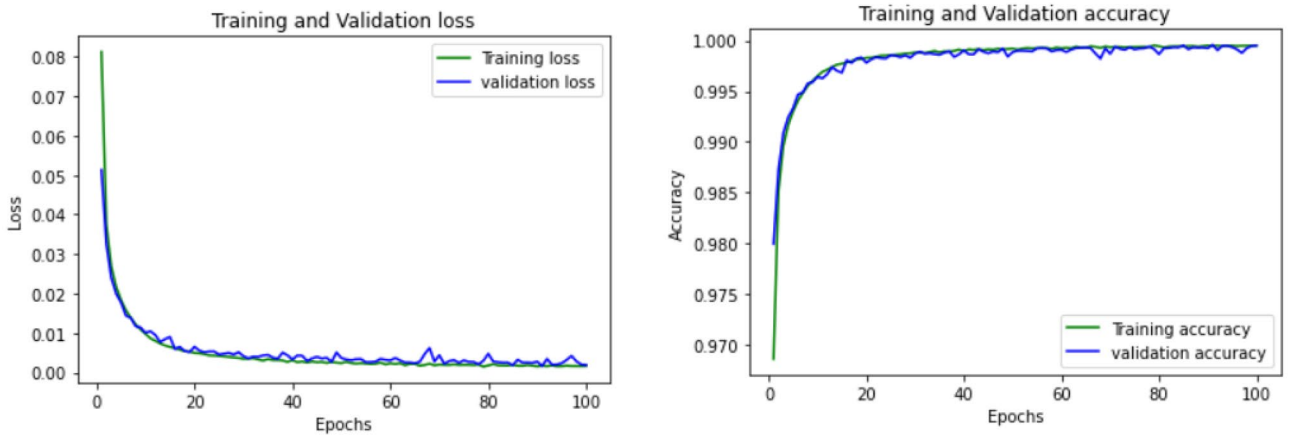


Fig. 17 A visualization of loss and accuracy of SSAE of c2760 circuit

Table 7 Number of hidden neurons and validation accuracy of feature extraction with sparsity constraints

Circuit	#gates	Sum of #Inputs and #outputs	#Faults	# Test Vectors	Architecture (L1, L2, L3)	VA% of SSAE	VA% of Deep AE
S1196	529	64	1242	16,391	(40,30,20)	98.5	97.55
S1238	508	64	1355	16,797	(30,20,10)	94.8	96.8
S1423	657	170	1515	24,359	(70,50,30)	99.7	98.98
S1488	653	39	1486	8627	(20,10,5)	93.1	96.6
S1494	647	39	1506	8630	(30,20,10)	97.9	97.45
S9234.1	5597	497	6927	92,055	(300,200,100)	**99.95**	**99.8**
S5378	2836	427	4551	67,991	(300,200,70)	99.93	99.71
s13207.1	7979	1,490	9815	9661	(300,100,10)	99.88	99.75
s15850.1	9775	1,295	11,322	11,725	(300,100,10)	99.84	99.77

detecting 1923 stuck-at faults (testing data) in c2670. For sequential circuits, the complete model evaluates maximum validation accuracy around 99.8% for classifying multiple stuck-at-faults as illustrated in Table 9. However, with the growing size of inputs and outputs of sequential circuits which causes search space explosion in other approaches, the proposed model accomplishes around 99.7% as validation accuracy to detect possible faults in "s13207.1", which contains 1,490 input and output port.

5.3 Comparsion to Fault Detection Based on SAT Domain

Table 10 illustrates running time of fault detection algorithm based in SAT domain (using MUS proposed in [32]) compared to consumed time of proposed FD-based on DL after learning how to use multiple test vectors as dataset for single faults. As was said in the previous section III.B the core of most fault detection or diagnosis methods is how to find the minimum number of SAT solver calls for increasing speed-up. However, the extraction of minimal unsatisfiable subsets (MUS) from SAT formulas equivalent to digital circuit can detect faults in digital circuits using advanced parallel SAT solvers. The main problem of FD-based on SAT solvers is the search space explosion by growing on complexity and size of IC designs. Therefore, the proposed FD-based on DL method is based on an attempt to teach our model to extract essential features for quickly detecting stuck-at-faults and getting rid of searching process into SAT formula for unsatisfiable subsets. This comparison is implemented on 9 different combinational and sequential circuits from ISCAS'85 and ISCAS'89 benchmarks. In SAT domain, the input of *Find-MUS* algorithm shown in Eq. 9 is a conjunction between input stimulus I and their corresponding correct outputs O_c of digital circuits, and the equivalent instance of erroneous circuit in conjunction normal form ($CNF(C_f)$). The erroneous circuit is provided by manual modification of the functionality of a single random gate and mapping stuck-at-faults into design fault models, as proposed in section II.A. Also, C++ algorithm of SAT Encoding algorithm [9] can be utilized for storing CNF instances in DIMACS format.

$$\varphi = I.O_c.CNF(C_f) \tag{9}$$

This equation is passed to Find-MUS algorithm proposed in [32], which is an explanation of logical fault existed in erroneous circuit. A parallel CUD@SAT solver proposed

Table 8 Number of hidden neurons and validation accuracy of classification process

Circuit	Architecture (L1, L2, L3)	VA%
c432	30,20,10	97.7
c499	50,30,20	91.1
c880	50,30,20	97.3
c1355	70,50,20	90.68
c1908	50,30,20	96.5
c2670	200,100,50	**99.6**
c3540	50,30,20	98
c5315	100,50,20	97.8
c6288	50,30,20	97.9
c7552	200,100,20	98.41

Table 9 Number of hidden neurons and validation accuracy and time of classification process

Circuit	Sum of #Inputs and #outputs	Architecture (L1, L2, L3)	Testing time (sec)	VA%
S1196	64	(40,30,20)	0.172	97.55
S1238	64	(30,20,10)	0.161	96.8
S1423	170	(70,50,30)	0.249	99
S1488	39	(20,10,5)	0.0969	96.6
S1494	39	(30,20,10)	0.0882	97.45
S9234.1	497	(300,200,100)	0.0014	**99.8**
S5378	427	(300,200,70)	0.895	99.7
s13207.1	1,490	(300,100,10)	0.225	99.75
s15850.1	1,295	(300,100,10)	0.266	**99.8**

Table 10 Number of hidden neurons and validation accuracy and time of classification process

Circuit	FD-based on SAT (time in sec)	FD-based on DL	
		Testing time in sec	Training time in sec
C432	2.18	0.0754	68
C499	4.89	0.109	24.9
C880	10.14	0.146	45.1
C1908	41.69	0.277	111
S1196	21.12	0.147	39.7
S1238	29.62	0.144	52.4
S1423	29.62	0.234	52.6
S1488	34.16	0.0767	10.8
S1494	36.13	0.0765	

in [20] is used in MUS generation for SAT solving on GPU. In Table 10, the proposed FD-based on DL achieved around 187 × as average speed-up compared to FD-based on SAT. Therefore, the performance of proposed FD-based on DL outperforms SAT fault detection, as a result of dispense of searching process and multiple calls of SAT solvers and construct DL model of IC design interpreting possible multiple stuck-at-faults by extracting features from test vectors and classifying multiple faults.

5.4 Comparsion to Other Models in ML and DL

In this subsection, we present results of fault detection using other models in machine learning and deep learning. Validation Accuracy of both proposed method and fault detection using Radial Basis Function network (RBFN) is illustrated in Fig. 19 in terms of validation accuracy. Although RBFNs are common with their speed but its validation accuracy in our application is not efficient in terms of detecting faults, compared to model of using Deep AE. The FD based on RBFNs achieves maximum validation accuracy about 97.81% where the proposed algorithm accomplishes about 99.6% maximum validation accuracy. Also, fault detection using RBF model has an average validation accuracy about 90.6% but our proposed algorithm (Deep AE and classifier using Sigmoid) achieved 96.1 for SSAE phase and 94.9% for classification phase. Also, Fig. 20 illustrates the variation of validation accuracy to s15850.1 circuit to show the improvement of validation accuracy as increasing epochs from 1 to 80.

Also, a comparison between our proposed FD model and some classic models of machine learning such as Decision Tree (DT), Random Forest (RF) and Gradient Boosting classifiers in terms of validation accuracy is shown in Fig. 21. It can be seen that other classic machine learning models do not perform well in detecting faults in eight digital circuits. On the contrary, our proposed FD

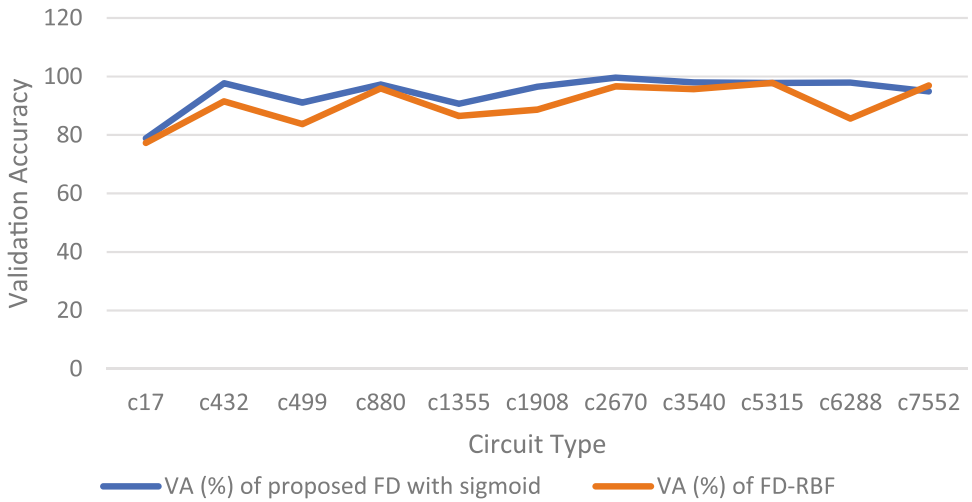


Fig. 19 Validation accuracy (VA%) of Classification process using different DL Architectures

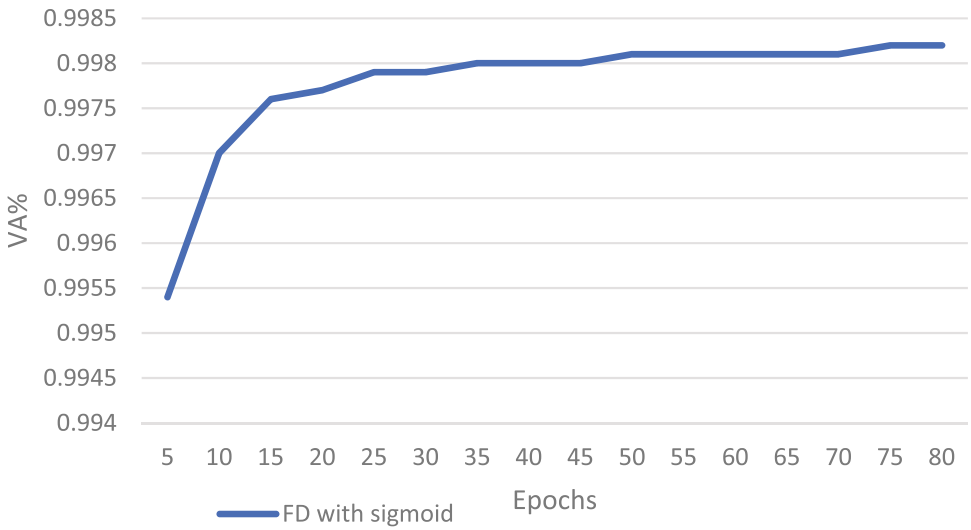


Fig. 20 Validation accuracy (VA%) of Classification process of s15850.1 during epochs

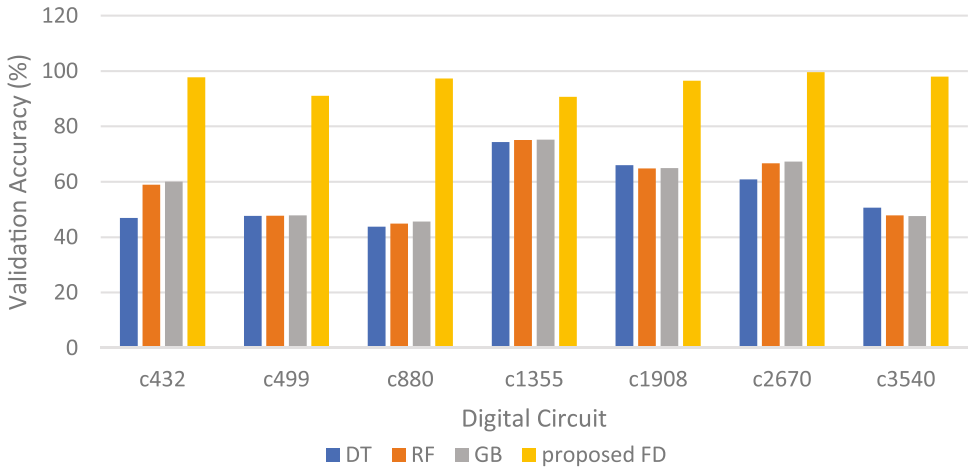


Fig. 21 A comparison between our proposed FD model and other ML models

model based on deep learning achieved a better performance of about 35.1% higher than DT model, and about 33.1% higher than RF and GB classifiers.

6 Conclusion

The main contribution of this work is to solving the search space explosion due to large size circuits by designing a semi-supervised model that is mainly based on Deep Sparse Auto-encoder for dimensionality reduction of features represented correct IC design. Different types of autoencoders (normal AE, Deep Sparse AE and stacked sparse AE) as unsupervised model, were implemented in order to find the latent representation of test patterns with their correct response for the next phase of fault classifier. Also, our model can extract the main connections in a given circuit which leads to multiple equivalent faults. The proposed FD model is tested on ten combinational and nine sequential circuits from ISCAS'85 and ISCAS'89 benchmarks. During experiments, Stacked Sparse AE extracted more robust latent features across source data set of sequential circuits while Deep Sparse AE was sufficient for combinational circuits. In addition, we compared the complete semi-supervised model to other fault diagnosis based on SAT solvers in terms of consumed time. The main advantage of our model is avoiding multiple calls of SAT solver and the reduction of search space problem for generating a specific MUS that can extract faults. The drawback of our model is that we must collect as many correct data as possible to approximate the connections between lines accurately for detecting the root cause of faults.

Funding Open access funding provided by The Science, Technology & Innovation Funding Authority (STDF) in cooperation with The Egyptian Knowledge Bank (EKB). There's no financial/personal interest or belief that could affect the objectivity.

Data Availability The proposed semi-supervised FD-model is available in [39] for detecting stuck-at faults in combinational and sequential circuits along with some of datasets and SSAE models for different design circuits.

Declarations

Conflict of Interest There's no financial/personal interest or belief that could affect the objectivity. Also, there is no conflict of interest.

References

1. Jo S, Matsumoto T, Fujita M (2014) SAT-based automatic rectification and debugging of combinational circuits with LUT insertions. IPSJ Transactions on System LSI Design Methodology 7:46–55
2. Rashinkar P, Paterson P, Singh L (2007) Singh, System-on-a-chip verification: methodology and techniques: Springer Science & Business Media
3. Gaber L, Hussein AI, Moness M (2019) Improved automatic correction for digital VLSI circuits. In 2019 Proceeding 31st international conference on microelectronics (ICM), 2019, pp 18–22
4. Gaber L, Hussein AI, Moness M (2020) Incremental Automatic Correction for Digital VLSI Circuits. Presented at the Proceeding 11th International Conference on VLSI (VLSI 2020)
5. Gaber L, Hussein AI, Moness M (2021) Fast Auto-Correction algorithm for Digital VLSI Circuits. Procedia Computer Science 182:95–102
6. Osama M, Gaber L, Hussein AI, Mahmoud H (2018) An Efficient SAT-Based Test Generation Algorithm with GPU Accelerator. J Electron Test 34:511–527
7. Rodríguez Gómez L (2017) Machine Learning Support for Logic Diagnosis. Doctoral dissertation, university of Stuttgart
8. El Mandouh E, Wassal AG (2018) Application of Machine Learning Techniques in Post-Silicon Debugging and Bug Localization. J Electron Test 34:163–181
9. Gaber L, Hussein AI, Mahmoud H, Mabrook MM, Moness M (2020) Computation of minimal unsatisfiable subformulas for SAT-based digital circuit error diagnosis. J Ambient Intel Humanized Comp pp 1–19
10. Jutman A, Ubar R (2000) Design error diagnosis in digital circuits with stuck-at fault model. Microelectron Reliab 40:307–320
11. Wahba A, Borrione D (1995) Design error diagnosis in sequential circuits. In Proceeding Advanced Research Working Conference on Correct Hardware Design and Verification Methods pp 171–188
12. Gao Z, Cecati C, Ding SX (2015) A survey of fault diagnosis and fault-tolerant techniques—Part I: Fault diagnosis with model-based and signal-based approaches. IEEE Trans Industr Electron 62:3757–3767
13. Ng A (2011) Sparse autoencoder. CS294A Lecture notes 72:1–19
14. Baldi P (2012) Autoencoders, unsupervised learning, and deep architectures. In Proceedings of ICML workshop on unsupervised and transfer learning pp 37–49
15. Vincent P, Larochelle H, Bengio Y, Manzagol PA (2008) Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th international conference on Machine learning pp 1096–1103
16. Rifai S, Vincent P, Muller X, Glorot X, Bengio Y (2011) Contractive auto-encoders: Explicit invariance during feature extraction. In Icml
17. Lynce I, Marques-Silva J (2001) Efficient data structures for fast sat solvers
18. Ali LG, Hussein AI, Ali HM (2016) Parallelization of unit propagation algorithm for SAT-based ATPG of digital circuits. In 2016 Proceeding 28th International Conference on Microelectronics (ICM) pp 184–188
19. Eén NSN (2016) The MiniSat Page. Available: http://minisat.se
20. Dal Palù A, Dovier A, Formisano A, Pontelli E (2015) Cud@ sat: Sat solving on gpus. J Exp Theor Artif Intell 27:293–316
21. Cook SA (1971) The complexity of theorem-proving procedures. In Proceedings of the third annual ACM symposium on Theory of computing pp 151–158
22. Gaber L, Hussein AI, Moness M (2020) Fast Auto-Correction algorithm for Digital VLSI Circuits. Presented at the 17th International Learning & Technology Conference
23. Bendík J, Černá I, Beneš N (2018) Recursive online enumeration of all minimal unsatisfiable subsets. In International Symposium on Automated Technology for Verification and Analysis pp 143–159
24. Bendík J, Cerná I (2018) Evaluation of Domain Agnostic Approaches for Enumeration of Minimal Unsatisfiable Subsets. In LPAR pp 131–142
25. Guthmann O, Strichman O, Trostanetski A (2016) "Minimal unsatisfiable core extraction for SMT," in. Formal Methods in Computer-Aided Design (FMCAD) 2016:57–64
26. Liffiton MH, Previti A, Malik A, Marques-Silva J (2016) Fast, flexible MUS enumeration. Constraints 21:223–250
27. Arodytska N, Bjørner N, Marinescu MC, Sagiv M (2018) Core-Guided Minimal Correction Set and Core Enumeration. In IJCAI pp 1353–1361
28. Becker AJ (2018) Satisfiability-Based Methods for Digital Circuit Design, Debug, and Optimization. EPFL
29. Leo K, Tack G (2017) Debugging unsatisfiable constraint models. In Proceeding International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems pp 77–93
30. Shimakawa M, Hagihara S, Yonezaki N (2018) Efficiency of the strong satisfiability checking procedure for reactive system specifications. In Proceeding AIP Conference pp 040051
31. Marques-Silva J (2012) Computing Minimally Unsatisfiable Subformulas: State of the Art and Future Directions. J Multiple-Valued Logic & Soft Comp 19
32. Bendík J, Černá I (2020) MUST: Minimal Unsatisfiable Subsets Enumeration Tool. In Proceeding International Conference on Tools and Algorithms for the Construction and Analysis of Systems pp 135–152
33. Selsam D (2019) Neural Networks and the Satisfiability Problem: Stanford University
34. Fišer P (2005) *Atalanta-M*. Available: https://ddd.fit.cvut.cz/prj/Atalanta-M/
35. Bryan D (1985) The ISCAS'85 benchmark circuits and netlist format. North Carolina State University 25
36. Brglez F, Bryan D, Kozminski K (1989) Combinational profiles of sequential benchmark circuits. In IEEE international symposium on circuits and systems pp 1929–1934
37. FC et al (2015) keras. Available: https://keras.io/

38. Gaber L, Hussein AI, Moness M (2021) Fault Detection based on Deep Learning for Digital VLSI Circuits. Procedia Computer Science 194:122–131
39. Mohammed Moness LG, Hussein AI, Ali HM. Automated Design Error Debugging of Digital VLSI Circuits [Online]. Available: https://drive.google.com/drive/folders/1Q2NswbxkbvioZ5YD5g1yPFKKPXwXmAqo?usp=sharing

Mohammed Moness received the B.Sc. (Hons.) and M.Sc. degrees in electronics and communication engineering from Assiut University, Egypt, and the Ph.D. degree in control engineering from BME, Budapest, Hungary. From 1975 to 1985, he worked as a Lecturer and an Assistant Professor with the Department of Electrical Engineering, Assiut University. In 1985, he joined the University of Minia, Egypt, where he worked as an Associate Professor and a Professor of systems and control engineering. From 1995 to 2018, he served as the Chairman for the Department of Computers and Systems Engineering, and the Vice Dean and the Dean for the Faculty of Engineering, Minia University. On the following topics, he published over 80 articles. His current research interests include multivariable systems, evolutionary algorithms, computational intelligence, and embedded systems. Dr. Moness is a member of the Engineering Advisor Committee, Supreme Council of Universities, Egypt. He is a Senior Member of IEEE.

Lamya Gaber Ali received her B.Sc., and M.Sc. degrees in Computer Engineering from Minia University, Egypt in 2014 and 2017 respectively. In 2014, she joined the department of computer and systems engineering, Minia University, Egypt as teaching assistant. She has been an assistant lecturer since 2018. Her research interests include Test Pattern Generation, Formal Verification, Parallel Programming, High Performance Computing and Artificial Intelligence.

Aziza I. Hussein received her Ph.D. degree in Electrical & Computer Engineering from Kansas State University, USA in 2001 and the M.Sc. and B.Sc. degrees from Assiut University, Egypt in 1989 and 1983, respectively. She joined Effat University in Saudi Arabia In 2004 and established the first Electrical and Computer Engineering program for women in the country and taught related courses. She was the head of the Electrical and Computer Engineering Department at Effat University from 2007-2010, 2016-2021. She was the head of Computer and Systems Engineering Department, Faculty of Engineering, Minia University, Egypt from 2011-2016. Currently, she is a professor and researcher at the Electrical & Computer Engineering Department at Effat University Saudi Arabia. Her research interests include microelectronics, analog/digital VLSI system design, RF circuit design, high-speed analog-to-digital converters design and wireless communications.

Hanafy M. Ali is an assistant professor at the Department of Computers and Systems Engineering, Faculty of Engineering, Minia University, El Minia, Egypt. He received his B.Sc., M.Sc. and PhD. degrees from the Electrical Engineering Department, College of Engineering, Minia University, Minia, Egypt in 1997, 2002 and 2008, respectively.