AUT Journal of Electrical Engineering

AUT J. Electr. Eng., 56(3) (2024) 419-438
DOI: 10.22060/eej.2024.22910.5572

Hardware Trojan vulnerability assessment in digital integrated circuits using learnable classifiers

Hadi Jahanirad*, Mohammad Fathi

Department of Electronics and Communication Engineering, University of Kurdistan, Sanandaj, Kurdistan, Iran

ABSTRACT: In the current distributed integrated circuits (IC) industry, the possibility of adversarial hardware attacks cannot be ignored. Hardware Trojans (HT) attacks may lead to information leakage or failure in security-critical systems. The wide range of HT types and related insertion strategies makes the HT detection process very complex. Consequently, developing IC design methodologies that are robust against HT insertion would be of great merit. To measure the HT robustness, a vulnerability analysis of the proposed circuits should be performed which involves several interrelated factors (e.g. the layout of white spaces distribution, the unutilized routing resources, the activity of the circuit nodes, the delay values of circuit paths, etc.). In this paper, a novel framework is proposed to classify the IC vulnerability level. First, a comprehensive dataset is generated considering different HTs insertion into the ISCAS 85 and ISCAS 89 benchmark circuits. Then extraction of efficient features from the input image is accomplished by pre-trained deep neural networks. Finally, the vulnerability level (which is defined as low vulnerable, moderately vulnerable, and highly vulnerable) of every circuit is extracted using various trained classifiers (Ensemble, SVM, Naïve Bayes, and KNN). Simulation results confirm a 25% improvement in classification accuracy in the most successful classifier (97%) compared with the most successful previous study (72%).

Review History:

Received: Jan. 21, 2024
Revised: Apr. 12, 2024
Accepted: Apr. 24, 2024
Available Online: Jul. 01, 2024

Keywords:

Ensemble Learning
Learnable Classifiers
Deep Neural Networks
Digital Circuits
Vulnerability Analysis
Hardware Trojans

1- Introduction

Integrated circuits play a vital role in modern intelligent systems among them security-critical systems and highly sensitive ones (e.g. internet network-based systems and airplanes) should be protected against adversarial attacks. The main hardware-based attacks on integrated circuits are related to HT insertion. The inserted HT constitutes a trigger mechanism and a payload part [1-3]. When the trigger condition becomes active, the payload section causes destructive effects in the IC. Various HT detection methods are developed to handle the newly emerged Trojans every year [4-7]. These detection methods can be categorized as follows.

There are two types of Trojan detection methods: the logic testing approach and the side-channel analysis [8-9]. The former is suitable only for always-activated HTs and, the last one can be utilized for even small-sized Trojans. IC temperature, power consumption, and critical path delay are the main side-channel effects that are utilized to detect the HTs. The thermal map of under test IC would be investigated to detect unwanted activity of circuit nodes that are interpreted as HTs [10]. In [11-14] and [9] machine learning and image processing techniques are employed for HT detection. A modified version of the heatmap according to the IC layout is utilized for HT detection in [15], [10]. Due to emerging more complicated structures for HTs, several studies have been devoted to utilizing deep learning approaches for detection affairs [16], [17], [7], [18].

On the other hand, to avoid costly detection approaches, we could re-design integrated circuits so that the chance of HT insertion becomes very low. This design technique will reduce the HT insertion vulnerability significantly. For instance, if the available white space area in the IC layout by dummy Flip-flops then the available space to insert the HT-related hardware will reduce dramatically.

A prerequisite of HT vulnerability classification is the investigation of all HT insertion scenarios [19]. The following points are deduced from such a comprehensive investigation: The layout nodes with low transition probabilities are usually utilized for HT triggers [19]. Moreover, the distribution of white spaces as well as the availability of unused routing resources are two essential factors for HT insertion [20] and [21]. The routing resources near the less active nodes which does not belong to the critical path are used to connect the trigger to the payload modules. Such routing mechanisms reduce the probability of HT detection by side-channel approach [22]. These observations satisfy us to consider the vulnerability analysis as a very complex problem.

Another worth noting point is related to very costly

*Corresponding author's email: h.jahanirad@uok.ac.ir

operations which should be applied in the design and fabrication stages to improve the vulnerability of ICs. To reduce the cost of such modifications, the vulnerability level of the design should be assessed in the early stages of the IC fabrication. The previous approaches for vulnerability evaluation of ICs suffer from several shortcomings. First, any previous approaches consider all effective factors so the derived results are not reliable. Second, each of the previous analyses is accomplished at a specific level of design (e.g. gate level, layout level, etc.). Furthermore, the major vulnerability factors are not homogeneous. So, deriving a comprehensive analysis would be done by combining these non-homogeneous factors.

In this paper, we develop a machine learning-based framework to handle the mentioned shortcomings of previous approaches. The proposed framework contains three major phases: HT dataset generation, feature extraction using pre-trained DNN, and the classification stages. In the first stage, various HTs are inserted in the layout of the benchmark circuits and the resulting layouts are converted to RGB images. The collection of the generated images constructs the required dataset. The performance of classifiers is very poor when the classifiers utilize the raw generated images. So, to achieve better classification, we should extract the important features of the dataset's images. In our proposed framework, feature extraction is accomplished by pre-trained deep neural networks (e.g. AlexNet, GoogleNet, and so on). The pre-trained DNNs have been well-trained using a standard dataset (e.g. ImageNET) and their convolutional layers have been trained to extract important features of such a huge database. In the last stage, several classifiers are trained using the extracted features to classify the IC layout into low, medium, and high vulnerable levels.

The main contributions of the paper are:

- Generation of a proper dataset for Hardware Trojan study in integrated circuits.
- Extraction of the main features of integrated circuit for HT vulnerability analysis.
- Development of a classification-based framework to classify IC layout regarding Hardware Trojans' vulnerability.

2- Related Works

The previous related studies use various metrics to evaluate IC's vulnerability at various design abstractions. In [23], Trojan paths in a digital circuit are defined as paths that are not activated during comprehensive simulation runtime. The time complexity of this method is unacceptable for large circuits due to its simulation-based nature. Moreover, if the HT trigger does not derive from circuit nodes directly, then they may be undetectable using this method. Waksman et al. [24] consider the low controllable logic gates as the candidates for HT triggers. Sullivan et al. [25] in Fight-metric applied necessary modifications to cover the sequential circuits. We can mention two problems regarding these methods. First, the computational complexity grows rapidly with a slight increase in the size of the circuit. Second, discrimination of Trojan and non-Trojan gates is made by a threshold value that is varied for various circuits.

Four different features are utilized for HT trigger determination in FASTrust [26]. The first feature is the existence of large loops which is essential for time-triggered Trojans implementation. The second feature is a number of large in-degree gates that are used to implement data-triggered Trojans. The third feature is related to the existence of large total in-degree groups to implement sequential triggers. Finally, the fourth feature corresponds to the nodes with a small out-degree, which can be utilized to activate the HTs implicitly. Another approach based on multilevel feature analysis is developed in [27]. In this methodology, first, trigger features are extracted using the circuit's information flow graph (IFG) at the flip-flop level. The feature extraction of ML-FASTrust speeds up significantly due to the small size IFG. When candidate locations are determined based on IFG, another analysis in the combinational level (CL) retrieves the lost information. The utilized features of ML_FASTrust are similar to the features of FASTrust. Such features are very poor in separating the HT-inserted nodes from the normal nodes. Moreover, FASTrust and ML-FASTrust ignore controllability, observability, and node transition rate as essential factors for vulnerability assessment.

Salamni [28] utilized circuit nodes' Combinational Controllability and Combinational Observability as the vulnerability metrics which are very effective in determining the low-testable nodes that can be safe locations for HT triggers. Due to the very low transition probability of these nodes, the conventional test algorithms cannot activate the related triggers. Later in [29] a modified version of this approach is presented for sequential circuits which is based on the sequential type of circuit nodes' controllability and observability. Controllability and observability become a common feature to detect HT triggers in other studies [30-31]. Even for gate-level vulnerability analysis consideration of just controllability and observability features results in ignorance of HT triggers which are constructed based on the nodes' in/out degrees in the circuit graph as well as the size of loop groups in sequential circuits.

The 11 features have been defined to measure the vulnerability of logic circuits at the gate level by Kento Hasegawa et al. [32-34]. Among these features, the number of fan-ins that are placed three or five stages away from the current node, the number of logic stages that should be traversed to connect the current gate to the input of a DFF, and the minimum distance between Primary Inputs (PIs) and Primary Outputs (POs) could be considered as the most effective ones. This approach suffers from significant misclassification errors due to the ignorance of the other basic features (e.g. signal activity, controllability, and observability).

[35] and [36] developed model-checking-based frameworks which are related to the combinational and sequential circuits, respectively. In this method, first of all, several counterexample circuits that contain various types of HTs are generated. Next, the deviation of dynamic and leakage power consumptions as well as the path delay values

are compared with the HT-free circuit. A predefined threshold level is used to determine if the variation value is sufficient to be considered as an HT.

Another viewpoint in vulnerability analysis is focusing on IC layout rather than the circuit design. H. Salamani and M. M. Tehranipoor [37] presented a layout-level vulnerability analysis wherein the interpretation is performed in three levels (cell, routing, and net analyses). In this method, the chip area is divided into square parts, and then for every region (r) the normalized values of white space (WS(r)) and number of unused routing (UR(r)) is calculated. The vulnerability of a region (V(r)) is computed according to (1). Furthermore, time-triggered HT vulnerability, power-triggered HT vulnerability and, time-power-triggered HT vulnerability of a grid could be calculated using (2), (3) and, (4), respectively. In these equations, $N_{NC}(r)$ is the number of paths that are not critical, $N_{LP}(r)$ is the number of nets for which the transition probability is less than P_{th}, and $N_{NC\ \&\ LP}(r)$ is the number of nets with which belong to both of the former sets.

$$V(r) = WS(r) \times UR(r) \tag{1}$$

$$V_{Td}(r) = V(r) \times N_{NC}(r) \tag{2}$$

$$V_{Tp}(r) = V(r) \times N_{LP}(r) \tag{3}$$

$$V_{Tdp}(r) = V(r) \times N_{NC\&LP}(r) \tag{4}$$

In net-level analysis, the low transition probability and the low testability are considered as the major features of HT payload. Consequently, 0-controllability (CC0), 1-controllability (CC1), and combinational observability (CO) are utilized to determine the candidate nodes to be HT payloads. The authors show quantitatively that the determined nets are inserted near the regions which are highly vulnerable in TrustHub benchmark circuits.

The space and trigger modeling in this approach encounter the following issues. Calculation of the probability of HT insertion into an integrated circuit using only the white space ratio of grids is not perfect. For example, the probability of HT insertion in a grid with a connected white space area is more than in a grid with the same white space ratio but the white space is divided into several disconnected parts. Furthermore, in several situations, the white space in adjacent grids could be merged to construct a wider white space area to insert HTs. This case is fully ignored in the previous modeling. One may use the routing resources from several grids to make the required connection between the trigger and payload parts. Consequently, the UR(r) should be extended to these aspects. Moreover, the previous studies only consider the single node (with low testability or low activity) as the candidate trigger. However, hardware Trojan designer can combine several conventional nodes using AND operation to produce a proper trigger node. As the last important point, this approach describes a quantitative relationship between trigger nets and related grids which leads to significant model inaccuracy.

M. Bakhshozadeh and A. Jahanian [38] divided the chip area into the square grid to extract the Trojan Vulnerability Map (TVM) of the integrated circuit. In the extracted TVM, the ratio of white space and the number of available DFF are used to assign a gray level to every grid. Consequently, in TVM, low-vulnerable grids have light colors and the high_ vulnerable regions become darker. They defined the Trojan Insertion Simplicity Factor (TISF) according to (5).

$$TISF = \frac{\frac{BWS}{TWS} \times \frac{BFF}{TFF}}{WSD} \tag{5}$$

$$WSD = \sum_{i=1}^{n} \frac{1}{n} [(x_i - x_m)^2 + (y_i - y_m)^2] \tag{6}$$

In this equation, a grid's white space ratio is indicated by BWS which is defined similarly to [37]. Moreover, WSD (White Space Distribution) indicates how white space distributes in a grid geographically which is calculated according to (6). In this equation, *n,* (x_m, y_m), and, (x_i, y_i) represent the number of the grid's white space units, the grid's center position, and the center position of the *i*'th white space unit. TWS represents the total white space ratio in the entire area of the integrated circuit. Moreover, due to the consideration of sequential hardware Trojans, the authors represented the probability of the insertion of sequential HT into a grid using the ratio of the number of grid's DFFs (BFF) to the number of IC's DFFs (TFF).

The white space modeling of this approach is incomplete similar to [37]. Moreover, this approach focused on white space and DFF numbers and, the vital vulnerability factors such as routing resource utilization, node activity and, testability factors were not considered in this framework. Consequently, the vulnerability analysis encounters significant inaccuracy.

A 3-metrics base framework has been developed by T. Trippel et al. [21] to assess the vulnerability of a chip at the layout level. The first metric is a histogram-based white space model. In the related layout, the occupied parts of the integrated circuit layout are colored and the white space regions are connected using a 4-neighbor approach. The final white space-related regions are utilized to derive the first metric of this approach. The second metric which is called Net Blockage represents how it is possible to block the hardware Trojan's payload to the available security-critical nets. The authors show that this probability reduces significantly when the routing congestion around such nets increases. As the third metric, the Manhattan distance between payload modules and the non-blocked security-critical nets is measured to derive

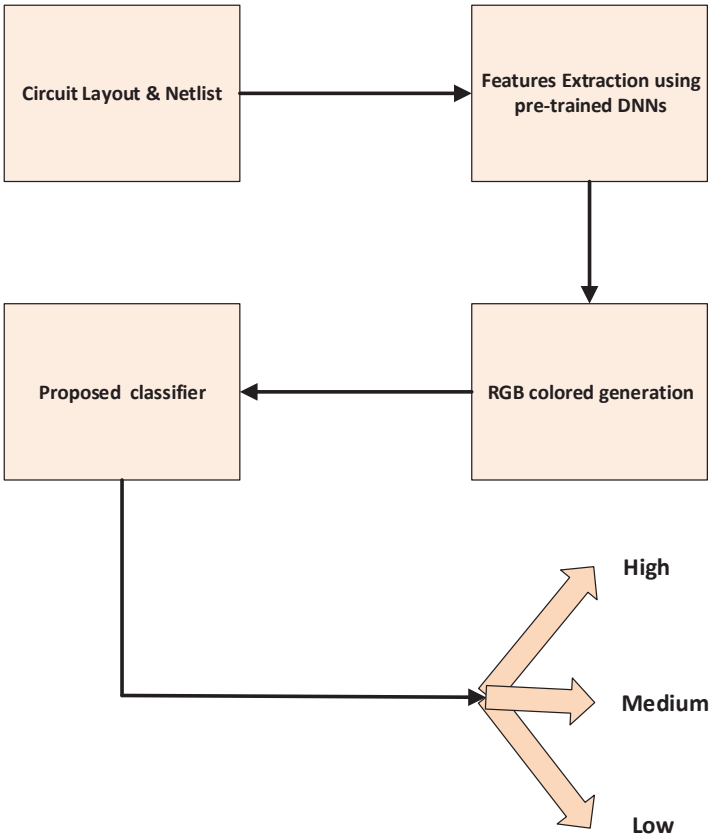


Fig. 1. The flowchart of the proposed approach.

the vulnerability of the co-existence of the payload and security-critical nets in the IC's layout. Inaccurate white space model, imperfect trigger nets modeling, and poor modeling of possible routing schemes between trigger and payload are the essential cones of the ICAS method.

3- Preliminary

A comprehensive investigation of HT architectures discloses that the rouge person usually follows some dedicated rules. The layout's white space area which is close to the unutilized routing modules, are ideal locations for HT insertion. Moreover, the circuit's nodes with low switching activity are utilized for triggering the HTs. Such nodes are connected to the payload using non-critical routing resources. On the other hand, some HT triggers originate from low controllable and observable circuit nodes which is robust against logic test detection methods. These general rules are converted to measurable features which finally are utilized by the proposed flow for the layout's vulnerability level classification. Fig. 1 shows the flowchart of the proposed approach. First of all, the essential features such as white space, routing congestion, signal probability, path delay, and testability metrics, are extracted using the graph of the circuit and its layout. Then as mentioned before by completing the dataset generation steps, the colored image of the circuit is generated. Next, the pre-trained DNN is utilized to extract the major features of the RGB-color image to be fed to the trained classifiers. We describe briefly how the essential features are extracted in the following sub-sections.

3- 1- *White spaces*

A pre-requisite to insert the hardware Trojans into the layout is the existence of a white space area. Generally, the integrated circuit's layout is generated by the Cadence Design System tool based on a synthesized file of the digital system's HDL code. The Cadence tool produces an optimum layout using available modules (cells) in the design library. The final layout of b15 benchmark circuit containing 3500 basic cells is illustrated in Fig. 2. The routing congestion in the margins of the layout is very sparse. Furthermore, the zoomed version of the three boxes highlighted in Fig. 2 is illustrated in Fig. 3.

For example, the extracted layout of b15 benchmark circuit from the Cadence Design System tool is . So, if the layout be divided into 10×10 grid area (containing 100 tiles), then the area of every tile would be $\frac{300\times300\mu m^2}{100}=30\times30\mu m^2$. In the resulting layout, INVX0 is utilized as the smallest gate

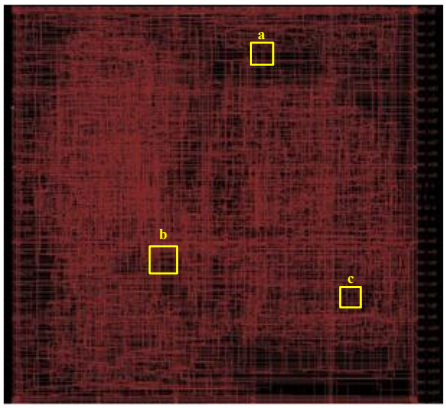


Fig. 2. Layout of b15 benchmark circuit

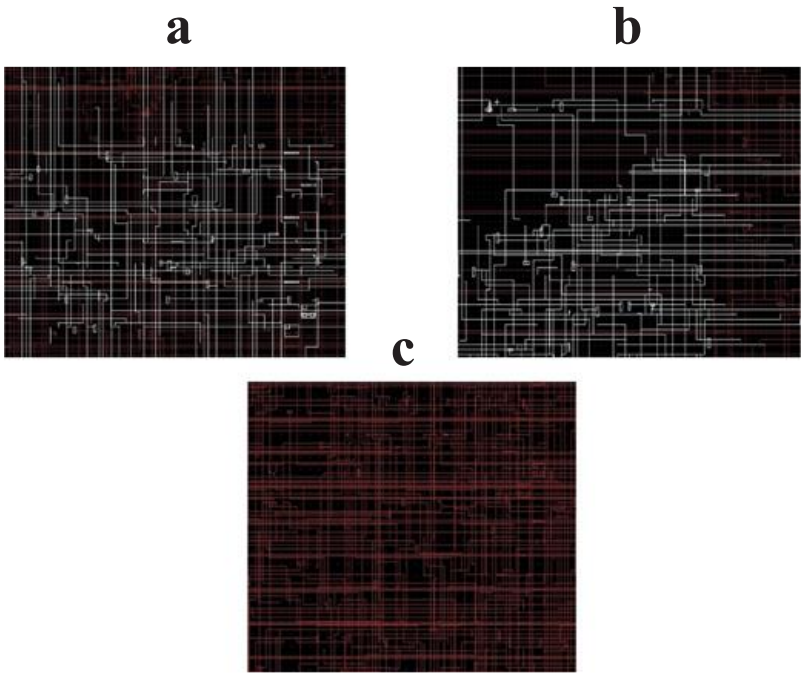


Fig. 3. The zoomed illustration of highlighted boxes in Fig. 2.

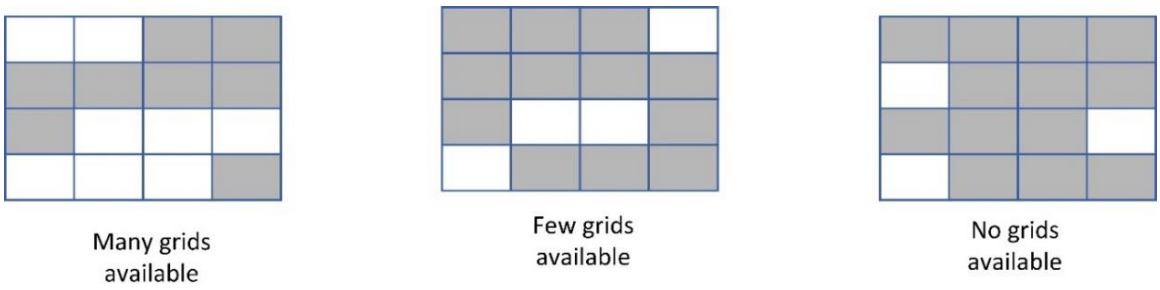


Fig. 4. Various five units white space units distribution.

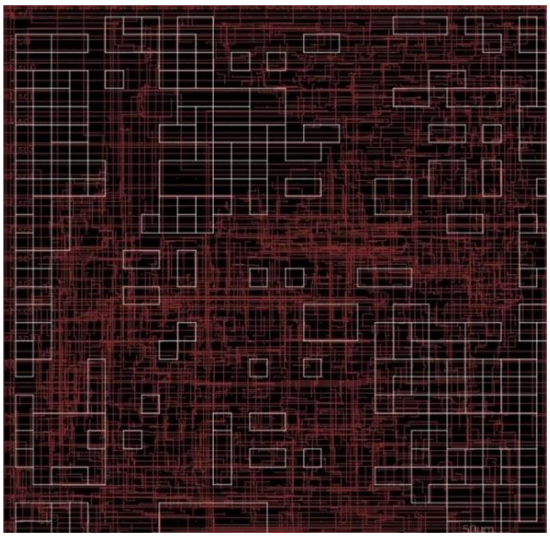


Fig. 5. The top-view of white space areas over the metal layer of EthernetMAC10GE benchmark layout.

in the SAED-EDK90nm library.

[39] mentioned that 30-40 % of the chip area is not utilized by the circuit modules which provides a sufficient white space area to be used by HT designers. However, the irregular distribution of the white space area may lead to some difficulty in HT insertion process. For example, if five unit blocks are required for an HT insertion, then five unoccupied unit space areas should be available in a compact area (e.g. a grid) for the insertion process. Three different cases are shown in Fig. 4. In the left figure, the white space units could be utilized easily for HT insertion. In the middle figure, despite the availability of five unoccupied units, their distribution makes HT insertion very difficult. Finally, in the right figure, the HT insertion is impossible due to the fact that there are not enough white space units. To prevent the HT insertion ideally, we should pursue the third case in the IC design phase. To fill up all white space units in the layout, we can use large DFFs or even smaller combinational gates (Talaee & Jahanian, 2017). As an example, the white space area in the metal layer of the EthernetMAC10GE-T100 benchmark (Trust Hub benchmark suit) is illustrated in Fig. 5.

3- 2- Routing congestion

The routing congestion is an important factor in vulnerability assessment due to the fact that the HTs require extra spaces to realize the trigger to payload routing [19]. Fig. 6 illustrates a HT inserted in a sea of logic modules. The Trigger and Payload modules of HT are shown in red color. The logic module which feeds the required signals for the Trigger module is indicated in Orange color and the

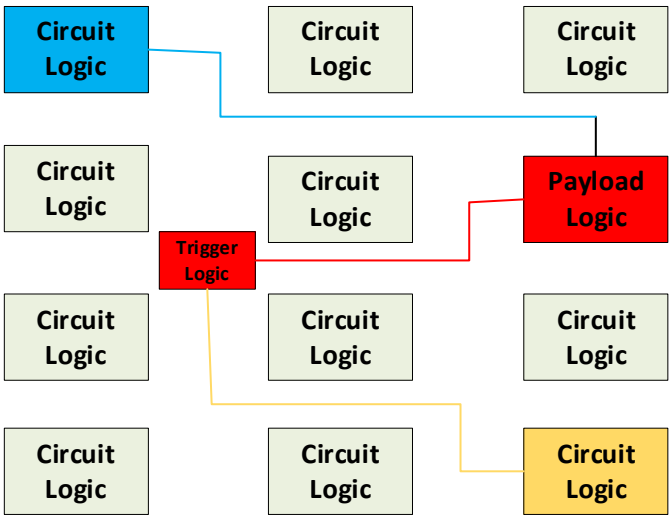


Fig. 6. An example of HT mechanism implementation.

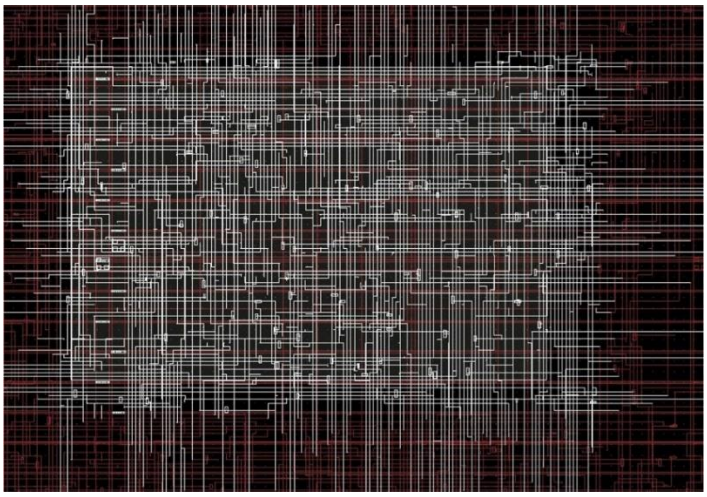


Fig. 7. Zoomed in image of metal layers for b15 benchmark circuit.

module which is affected by the payload of HT is shown in Blue color. We also illustrated the required routing paths that should be realized to correctly implement the HT mechanism. If the routing resources in every routing channel which the HT paths are passed through them are utilized by the circuit's logic modules, then the HT mechanism will fail. So, the HT mechanism needs available unused routing resources in the specified routing channels. Consequently, the existence of unused routing resources increases the vulnerability level significantly.

Derivation of the routing congestion of an integrated circuit layout is done using the output files of the Layout Editor tool. The routing information is extracted from Design Exchange Format (DEF) and Library Exchange Format (LEF) and the congestion in every unit region of the layout area could be calculated based on the available wires divide by the total realizable wires in that unit region. A zoomed version of the routing layer of metal layers for b15 benchmark circuit is illustrated in Fig. 7.

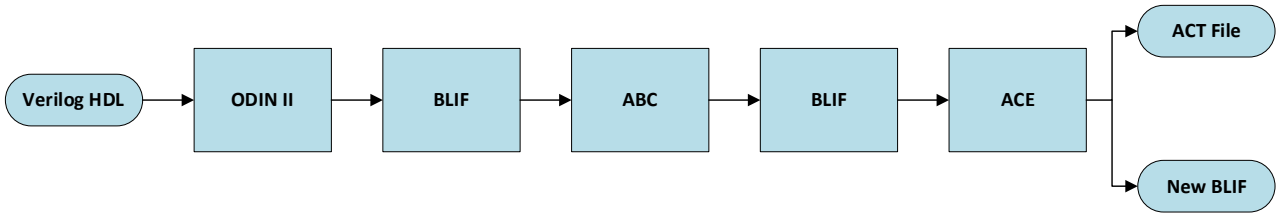


Fig. 8. The flowchart of switching activity calculation.

Table 1. signal probability and switching activity of internal nodes of c17 benchmark circuit.

Gates and nodes	Signal probability	Switching activity
G1	0.49560	0.19920
G2	0.48360	0.20440
G3	0.49500	0.19900
G4	0.55880	0.11893
G5	0.56400	0.22627
G6	0.75200	0.15437
Node [1]	0.38220	0.008024
Node [2]	0.37060	0.050244
Node [3]	0.38220	0.008024
Node [4]	0.24600	0.044761

3- 3- Dynamic power and transition probability

The transition probability of every internal net of the circuit is directly related to the dynamic power consumption of the net. The transition density of node s (D_s) is calculated using (7) where n_s is the number of node transitions in the time interval equals T [40-41]:

$$D_s = \lim_{T\to\infty} \frac{n_s}{T} \tag{7}$$

Najm [40] defines the transition probability as the ratio of the number of clock cycles where the signal toggles to the total number of clock cycles in a specific time interval. The relationship between dynamic power consumption ($P_{dyn}(s)$) and the related node's transition density (D_s) is indicated in (9) [40]:

$$D_s \geq \frac{P_{dyn}(s)}{T} \tag{8}$$

Moreover, [40] defines the average dynamic power consumption (P_{av}) according to (9) where C is the total capacitance value of node s and V_{dd} is the IC's power supply.

$$P_{av} = \lim_{T\to\infty} V_{DD} . \frac{CV_{DD}\, n_s/2}{T} \tag{9}$$

The capacitance of every circuit node is derived from the output files of the Cadence Design System tool. The V_{dd} is a predefined parameter and the n_s should be derived according to Fig. 8. In the indicated flow, the Verilog code related to the circuit is fed into ODIN II tool ([48]) to produce the BLIF file [42].

The ABC tool makes the BLIF file generated by ODIN II a more optimized BLIF by logic and technology mapping approaches [49]. In the next step, ACE tool is utilized to extract the switching activity and transition probability of the circuit's nodes [50]. Based on the derived information the transition distribution could be presented in geometrical form according to gates' locations throughout the chip area. As an example, the final transition probability distribution of c17 benchmark circuit is shown in Fig. 9 [15]. The related Signal Probability and Switching Activity values of c17 are reported in Table 1.

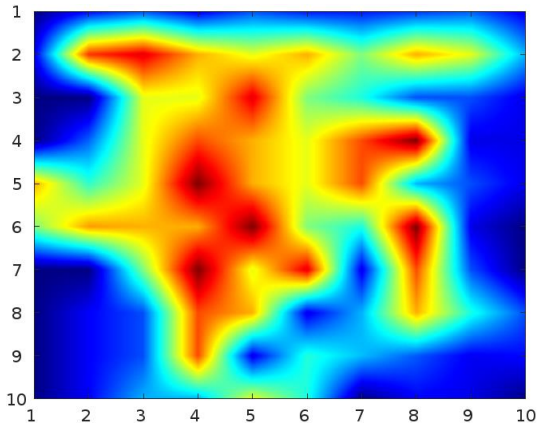


Fig. 9. The heatmap of b15's switching activity.

Table 2. Definition of testability metrics.

$CC0$	**Combinational 0-controllability of s**
$CC1$	Combinational 1-controllability of s
CO	Combinational observability of s
$SC0$	Sequential 0-controllability of s
$SC1$	Sequential 1-controllability of s
SO	Sequential observability of s

3- 4- Controllability

To major factors of digital circuit testability are the node's observability and node controllability. The most famous tool to derive the testability factors is the Sandia Controllability Analysis Program (SCOAP) [43]. The parameters combinational zero-controllability (CC0), combinational one-controllability (CC1), and combinational observability (CO) as well as sequential zero-controllability (SC0), sequential one-controllability (SC1), and sequential observability (SO) are utilized to calculate the testability of digital logic circuits in SCOAP [28]. Definitions of these parameters are reported in Table 2. The range of controllability values starts from 0 to infinity. Moreover, the controllability values of primary inputs are set to one, and the application of the related calculations increases the controllability values toward primary outputs. The related calculations to derive the controllability parameters in the output of primary gates are reported in Table 3. As a general rule, by increasing the value of controllability value, the capability of test algorithms to detect the existence of HT at that node would be decreased. Consequently, the high controllability of a node makes it more suitable to be utilized as HT trigger. To merge the controllability and observability measures of a node for testability related metrics we use (10) :

$$CC(i)=\begin{cases}\sqrt{CC0(i)^2+CC1(i)^2+CO(i)^2} & \text{Combinational Circuits}\\ \sqrt{SC0(i)^2+SC1(i)^2+SO(i)^2} & \text{Sequentional Circuits}\end{cases} \quad (10)$$

As an example, the calculation of four different logic gate's combinational and sequential controlabilities is presented in Table 3.

4- Proposed method

4- 1- Dataset

We have used two suites of benchmark circuits (ISCAS 85 and ISCAS 89) which contain 25 digital circuits to generate the required dataset. These benchmark circuits constitute primary inputs and outputs as well as logic gates with a variety of types and sizes. Implementing the circuits on a chip includes placement of IO and logic gates so that the area, speed, and other design objectives tend to be optimal. After placement, the places IO and logic gates are connected using wire and other routing resources. The main goal of the routing phase is to realize the required routes with minimal delay overhead and low congestion. Taking different placements for a logic circuit leads to producing different implementations. So, for 25 selected benchmark circuits, we

Table 3. Controllability evaluation for basic logic gates.

	CC0	SC0	CC1	SC0
AND	min[CC0(inp1),CC0(inp2)]+1	min[SC0(inp1),SC0(inp2)]	CC1(inp1)+CC1(inp2)+1	SC1(inp1)+SC1(inp2)
OR	CC0(inp1)+CC0(inp2)+1	SC0(inp1)+SC0(inp2)	min(CC1(inp1),CC1(inp2)]+1	min[SC1(inp1),SC1(inp2)]
XOR	min[(CC0(inp1)+CC0(inp2)),(CC1(inp1)+ CC1(inp2))]+1	min[(SC0(inp1)+SC0(inp2)),(SC1(inp1)+SC1(inp2))]	min[(CC0(inp1)+CC1(inp2)), (CC1(inp1)+CC0(inp2))]+1	min[(SC0(inp1)+SC1(inp2)), (SC1(inp1)+SC0(inp2))]
DFF	min[CC1(RESET)+CC1(clk)+CC0(clk), CC0(D)+CC1(clk)+CC0(clk)]	min[SC1(RESET)+SC1(clk)+SC0(clk), SC0(D)+SC1(clk)+SC0(clk)]+1	CC1(D)+CC1(clk)+CC0(clk)+ CC0(RESET)	SC1(D)+SC1(clk)+SC0(clk)+ SC0(RESET)+1

try to extend the number of implementations using different placements. In our dataset generation procedure, we have generated 400 implementations for every benchmark circuit. Consequently, the total number of implemented integrated circuits in the proposed dataset would be 10000.

The parameters that are varied during the dataset generation process include chip area size, placement of circuit modules, and the type of gates logic approach. The range of chip area for every benchmark circuit is one to ten times of the minimum required area. The white space distribution for various chip areas for a benchmark circuit is varied randomly, so the major HT insertion-based factor would be swept appropriately. On the other hand, by utilizing different placement approaches, the locations of primary IOs and logic gates on the chip area are distributed randomly. This policy leads to produce a variety of testability, signal activity, and routing congestion distribution for every benchmark circuit. Finally, choosing a different design for logic gates in the generated implementations (e.g. complementary logic, ratioed logic, transmission gate logic, pass transistor logic, and dynamic logic) results in the generation of different layouts wherein the white space and routing patterns would experience a wide range of variation.

In the next stage of dataset generation, the 10000 circuit implementations are converted to RGB images which are proper representations for training the proposed classifier. To do so, first, the chip area is divided into square-shaped grids wherein every grid is associated with a pixel of the final image. The Red, Green, and Blue components of a specific pixel are represented by an N-bit binary number. For example, there are 256 different levels for every component when an 8-bit binary number is utilized. The five major features (White space distribution, Signal activity of the nodes, Controllability and Observability of the circuit's nets, and Routing utilization of the routing network) related to the vulnerability of logic circuits to HT insertion process should be assigned to three RGB components of the pixels. Because of the essential relevance of testability metrics (Controllability and Observability metrics) with switching activity, we merged three major factors to construct the green component value on the pixels. To complete the process, we first normalize the values of CC0, CC1, CO, and the signal activity (*act*) associated with every grid by dividing the related value by the maximum possible value of these parameters. The normalized parameters are merged according to (11) to achieve the real number RR_{norm}. Finally, the resulting number is transformed to an 8-bit binary number which acts as the green component of the pixel.

$$G_{norm} = \sqrt{(CC0_{norm} + CC1_{norm} + CO_{norm})^2 + act_{norm}{}^2} \quad (11)$$

We define the white space value of a grid as the ratio of the unoccupied region of the grid to the total grid area. For instance, if ¼ of a grid area is occupied by the logic part, then the white space ratio of that grid will be 0.75. Consequently, the White Space Ratio is normalized by the definition and can be directly transformed to 8-bit binary number. We assigned the resulting number to the Red component of the pixel which is related to the current grid.

The last component (Blue) is associated with the routing congestion factor. To measure this component value, we evaluate the number of routing wires that pass across the above space of the grid. The resulting value is divided to the total possible number of wires that can routed through the above space of the grid. The resulting number is in the range of 0 to 1 and can be directly converted to 8-bit binary number.

Because of using the supervised approach to derive the HT vulnerability class of the implemented circuits, we should determine the class label (High, Medium, and Low classes) for every image of the generated dataset. The procedure of class labeling includes the measurement of the difficulty of various HTs insertion into the implemented circuits. For every type of HT architecture, first of all, we seek the required white space area. If there is not such a white space region, then the process of HT insertion will fail and the zero is registered

as the score. On the other hand, if the proper white space is available, then the trigger formation is investigated. For every possible trigger which can be constructed by nearby logic gates and routing resources, the difficulty of connecting the trigger to the HT payload is evaluated based on the trigger distance among the trigger and the payload module. Finally, the total score is calculated by the addition of all available triggers' scores.

Based on the calculated scores of all images, the images are partitioned into three parts; the first part is related to the images with a score greater than 75% of all images. We labeled this part with a highly vulnerable class. The second part is related to the images with a score less than 75% of all images' scores which are labeled as Low vulnerable class. The remaining images are labeled with a Medium vulnerable class.

It is worth noting that due to the feature extraction of images using pre-trained DNNs, we should resize the images to be matched with the utilized deep neural network. For example, the generated image in the dataset should be resized to 224×224 image for VGG 16.

To evaluate the dataset generation mechanism, we extract the average feature value of every pixel in all dataset images for High, Medium, and Low vulnerable classes. As we declared, the correlated features (Controllability, Observability, and Signal activity) are merged in the Red component of the image pixels as well as the White space and Routing congestion features are assigned to the Green and Blue components, respectively. The average of these components for every vulnerability class shows the distribution of the related features in the images that belong to the class. Suppose that class *c* contains N_c images and every image is constructed from an H×V pixel. Then the average value of the image components for (i,j)'s pixel is calculated using (12). In this equation, $image_k(i,j,1)$, $image_k(i,j,2)$ and, $image_k(i,j,3)$ are the Red, Green and, Blue values of pixel (i,j) in the k'th image.

$$\begin{aligned} R_{avg}(i,j) &= \frac{1}{N_c}\sum_{k=1}^{N_c} image_k(i,j,1) \\ G_{avg}(i,j) &= \frac{1}{N_c}\sum_{k=1}^{N_c} image_k(i,j,2) \\ B_{avg}(i,j) &= \frac{1}{N_c}\sum_{k=1}^{N_c} image_k(i,j,3) \end{aligned} \tag{12}$$

Figure 10 illustrates the pairwise comparison of these components for three vulnerability classes. Red component comparison depicts that the testability and signal activity features which are combined using (11) are slightly efficient to distinguish the images belonging to the low vulnerable class from the other two classes. However, class Medium and class High show very similar behavior according to the Red component. On the other side, the Green component comparison reveals the efficiency of the white space feature in vulnerability class discrimination illustrated in the middle row of Figure 10. According to these figures, class High shows a meaningful difference from the other two classes as well and the white space feature can slightly separate the class Low and class Medium. Finally, the third row of Figure 10 showcases the pairwise comparison of routing congestion-related components. We deduce from these figures that the Blue component can moderately separate the images of class High from the other classes meanwhile slightly can be utilized to distinguish class Low and Medium as well.

4- 2- Utilized Classifiers

4- 2- 1- Ensemble Classifier

Ensemble Learning is widely used to achieve better classification results using multiple base classifiers. In the homogenous ensemble of classifiers, a similar type of base classifiers is utilized. The base classifiers are trained and validated using the appropriate part of the dataset (training + validation data). Then the ensemble of classifiers is tested using the other unseen part of a data set. There are several methods to choose the training set from the dataset among them Bagging (Bootstrap Aggregating) is the famous approach [44]. Bagging follows the sampling with replacement approach to choose N data from the available part of the dataset to train every base classifier in turn. It is obvious that some data would not be selected ever according to this approach. These data are collected to compose the validation set. It was approved that this policy choose 63.2% of the first part of the dataset for training and the remaining 36.8% of them are used in the validation phase. The final phase of the ensemble classification is to test the overall performance of the ensemble. Based on the ensemble's architecture, the decisions of base classifiers are combined (e.g., majority voting approach) and the final decision is compared again the correct result.

In the Gradient Boosting (GB) approach, the learning process is started by generating a random weak learner. Then the related loss function is evaluated for training data. Based on the classification result and the evaluated loss function for every individual of training data, the residual values for training data are extracted. In the next stage of gradient boosting, the learning algorithm tries to fit an improved version of DT to minimize the residual values. After the construction of the new DT, the previous process is repeated for the new DT and the algorithm moves to the next stage [45]. Later in [46], a modified version of GB ensemble was presented wherein a regularization term was added to the loss function to control the growth of the model complexity and better avoidance of overfitting (XGB). It is worth noting that the major hyperparameters of gradient boosting algorithms are the number of trees, depth of trees, and learning rate (that compromises computational complexity and accuracy).

4- 2- 2- Naïve Bayes Classifier

One of the simplest classification algorithms is NB classifier which is constructed based on Bayes' theorem. Suppose that we want to construct a Naïve Bayes classifier

Fig. 10. Pairwise comparison of class features in the generated dataset.

for a dataset in which every individual has n features (x_1, x_2,…, x_n). If the total number of classes is K, then for test data including the arbitrary features, the probability of belonging to C_j would be calculated according to (eq. 13).

$$P(C_j \mid x_1, x_2, ..., x_n) = \frac{P(C_j)\prod_{i=1}^{n} P(x_i \mid C_j)}{\sum_{k=1}^{K} P(C_k)\prod_{i=1}^{n} P(x_i \mid C_k)} \quad (13)$$

In this equation, $P(C_i)$ is the probability of class C_i occurrence, $P(x_i|C_i)$ is the probability of occurrence of feature x_i in class C_i. All of these values can be evaluated based on training data. After derivation of all related probabilities, the data has been assigned to the class with maximum probability. The major disadvantage of Naïve Bayes classifier is the independence assumption among various features which is considered as the source of errors.

4- 2- 3- Multiclass SVM

Binary Support Vector Machine (SVM) is a popular classifier in machine learning applications. The SVM tries to find a hyperplane that separates the training data of two classes maximally. The nearest data to the derived hyperplane are called support vectors. In the case of more complicated features, the nonlinear SVM classifier is constructed by means of applying the linear SVM to the transformed feature space. The transformed feature space is generated by the substitution of a nonlinear kernel (e.g. Gaussian radial-based

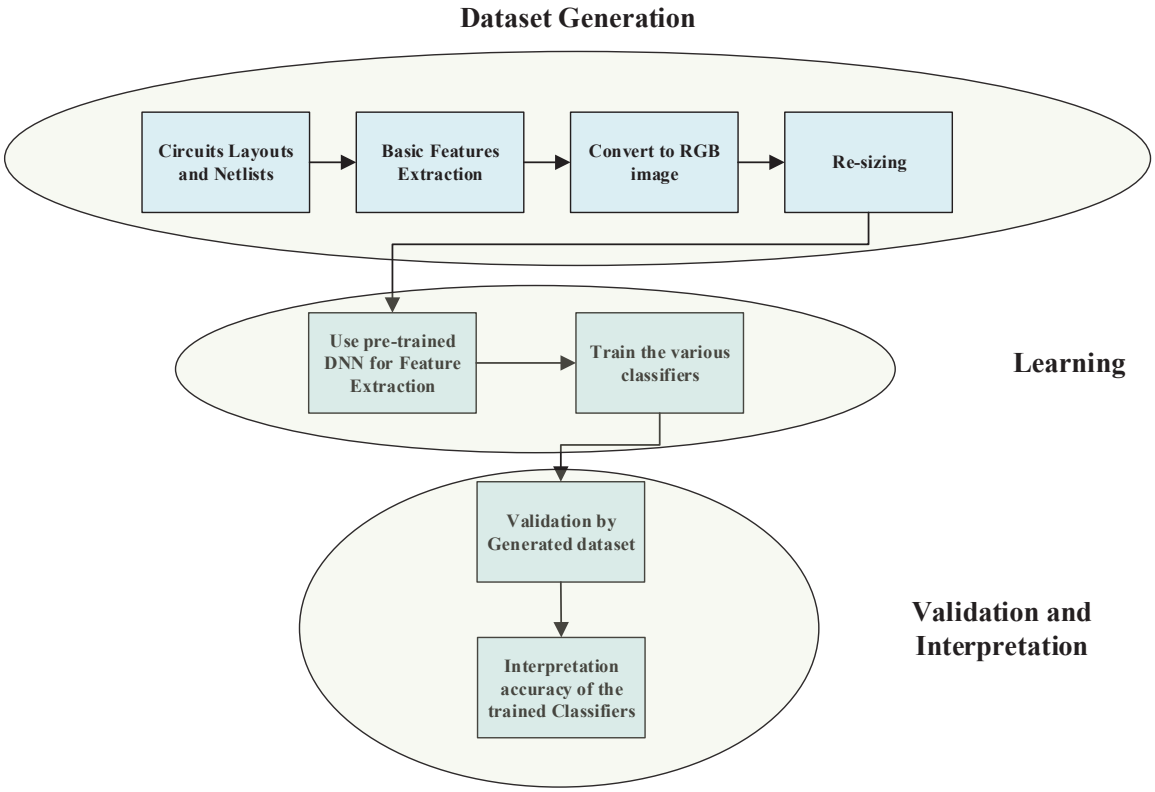


Fig. 11. The details of proposed framework's stages.

kernel) instead of a linear dot product.

The application of SVM for multiclass problems is accomplished by reducing the original problem to multiple binary classification problems. In the one-versus-all strategy, the class label that is assigned to the instance belongs to the classifier with the highest output score. On the other hand, in the one-versus-one strategy, the instance is introduced to all pairs of binary classifiers, and the instance label would belong to the class with a maximum number of wins. Some other solutions have been proposed to solve the multiclass SVM problem such as Directed Acyclic Graph SVM, Error Correcting Output Codes etc.

4- 2- 4- K-Nearest Neighbors classifier

In the k-nearest neighbors classifier, the instances of training data are labeled with the available classes. Then every test (new) data is labeled according to the most frequent labels of all k nearest classified samples to the instance. The major issue of the k-NN algorithm is the metric of distance measurement among the test data and the other labeled data to find its nearest neighbors. For instance, Euclidean distance is suitable for continuous variables and Hamming distance is efficient for discrete variables. Furthermore, for large datasets, the computation of distances between the test instance and all the other labeled data would be very complex. To overcome this problem some nearest neighbor search algorithms have been proposed. In the modified version of k-NN a weight (proportional to the inverse of distance) is assigned to the neighbors to emphasize the nearer neighbors in the decision-making of the algorithm.

4- 3- Proposed HT vulnerability analysis framework

Details of the proposed vulnerability analysis framework are illustrated in Fig. 11. The first step in the proposed framework is dataset generation using the procedure described in section 4.2.1. The generated images (which are labeled by their vulnerability levels) are fed into the pre-trained deep neural network platforms (e.g. GoogleNet, VGG 16 etc.) to extract the suitable features. In this stage, the user can select the feature extractor as well as the output stage of the selected DNN. Consequently, at the end of this stage, all dataset's images are converted to the related matrices which contain the features.

In the next stage, a classifier is selected among the mentioned classifiers in section 4,2 (Ensemble, Naïve Bayes, SVM, and KNN classifiers). Then a subset of the dataset (the

Table 4. The optimizable hyperparameters of various classifiers

Classifier	Optimizable Hyperparameters
Ensemble	- Ensemble method - Maximum number of splits - Number of learners - Learning rate - Number of predictors to sample
Naïve Bayes	- Distribution names - Kernel type
SVM	- Kernel function - Box constraint level - Kernel scale - Multiclass method - Standardized data
KNN	- Number of neighbors - Distance metrics - Distance weight - Standardized data

matrices of extracted features) is selected as training data and the remaining portion is considered for test purposes. The selected classifier is trained and validated using training data and the related evaluation metrics (Accuracy, True Positive Rate, True Negative Rate) are derived using test data.

The problem of the most appropriate hyperparameter selection is accomplished using optimization techniques. The hyperparameters related to the utilized classifiers are reported in Table 4. In our proposed platform, the following steps are traversed: first of all, the hyperparameters that should be optimized for current classifiers are selected. Next, the method of optimization for hyperparameters is selected. Bayes optimization, Grid search, and Random search are common optimization techniques. After the selection of optimizable hyperparameters and the optimization method, the exploration of search space is done until the algorithm reaches the stop criteria or the maximum number of iterations.

5- Results

We utilized Deep Network Designer and Classification Learner Tools from MATLAB 2022 software to perform the simulations. The specifications of the personal computer are Core i7 CPU and internal random access memory with 8 Giga Bytes capacity. The feature extraction of the images is completed using various DNN architectures that are available in the Deep Network Designer tool of MATLAB. These networks are pre-trained using very large datasets (e.g. ImageNet) which gives a high capability to these networks to extract the essential features of any other datasets. The best values for hyperparameters were derived by exploration of several possible combinations of DNN's parameters [47].

The performance of every classifier in our simulations is evaluated using the following metrics:

True Positive (TP): The images which are truly recognized as a member of the current class.

True Negative (TN): The images which are truly recognized that do not belong to the current class.

False Positive (FP): The images which are incorrectly recognized as a member of the current class.

False Negative (FN): The images which are incorrectly recognized that do not belong to the current class.

$$\text{Accuracy} = (TP + TN)/(TP + TN + FP + FN) \quad (14)$$

$$TPR = TP/(TP+FN) \quad (15)$$

$$TNR = TN/(TN+FP) \quad (16)$$

Eventually, based on the above definitions, we use three metrics to evaluate the performance of the trained classifiers as follows: 1- Accuracy: the number of truly recognized images (the members of the class and the non-member of the specific class) divided by all images according to (14). 2- Precision: The number of truly recognized images of the current class divided by the total number of images of the current class (15). 3- Specificity: The total number of non-

Table 5. Accuracy of various classifiers for vulnerability evaluation

	Accuracy %			
	Low	Medium	High	Average
Ensemble	96.47	95.89	96	96.12
Naïve Bayes	85.63	83.81	90.99	86.81
SVM	97.38	96.15	96.54	96.69
KNN	94.91	93.01	95.89	94.60

Table 6. The comparison of the accuracy of various methods

Method	Accuracy %
Ensemble	96.12
Naïve Bayes	86.81
SVM	96.69
KNN	94.60
[21]	72
[37]	65.5
[38]	60.35

member of images of the dataset which are truly recognized by the classifiers divided by the total number of images that do not belong to the current class (16).

Table 5 reports classifier accuracy. The accuracy values start at 83.81 % of Naïve Bayes and end at 97.38% of SVM. Fig. 12 illustrates the confusion matrixes for the utilized classifiers. To test the trained classifiers, we selected 1500 images randomly from the dataset and derived the performance metrics. The performance of every classifier is approximately uniform for all low, medium, and high vulnerable classes. Ensemble and SVM outperform KNN in all cases but the Naïve Bayes classifier accuracy never reaches 90% accuracy.

Figure 13 shows TPR and TNR for different utilized classifiers. The first metric (TPR) represents the ability of the trained classifiers to recognize truly the vulnerability levels of the related implemented circuit. On the other hand, the latter metric represents the ability of the trained classifier to dump out the images that do not belong to the current class truly.

Our proposed method is a machine learning-based approach to assess the vulnerability level of an implemented digital circuit. There is no straightforward previous method to be utilized to be compared with our proposed method. However, we have independently extracted the performance of [21], [37] and, [38] approaches to classify the circuits of the generated dataset. These methods evaluate the vulnerability level of the implemented circuit by dividing the layout into square-based grids. A short introduction to these methods is presented in the related work sub-section. The results of the comparison of these approaches with our ML-based methodology are reported in Table 6. The superiority of the ML-based approach is clear in comparison to the previous methods. This is mainly due to the consideration of all important factors in dataset construction and classifier training. Due to better modeling of white space distribution in [21], the accuracy of this method is significantly better than the other two methods. However, the other features (routing congestion and trigger modeling) are not investigated accurately by [21] which leads to 28% inaccuracy. [37] achieves more accuracy in comparison with [38] due to considering the testability metrics for trigger modeling. The other shortcomings of the [38] approach are the inapplicability of the method to combinational circuits along with the ignorance of proper trigger modeling.

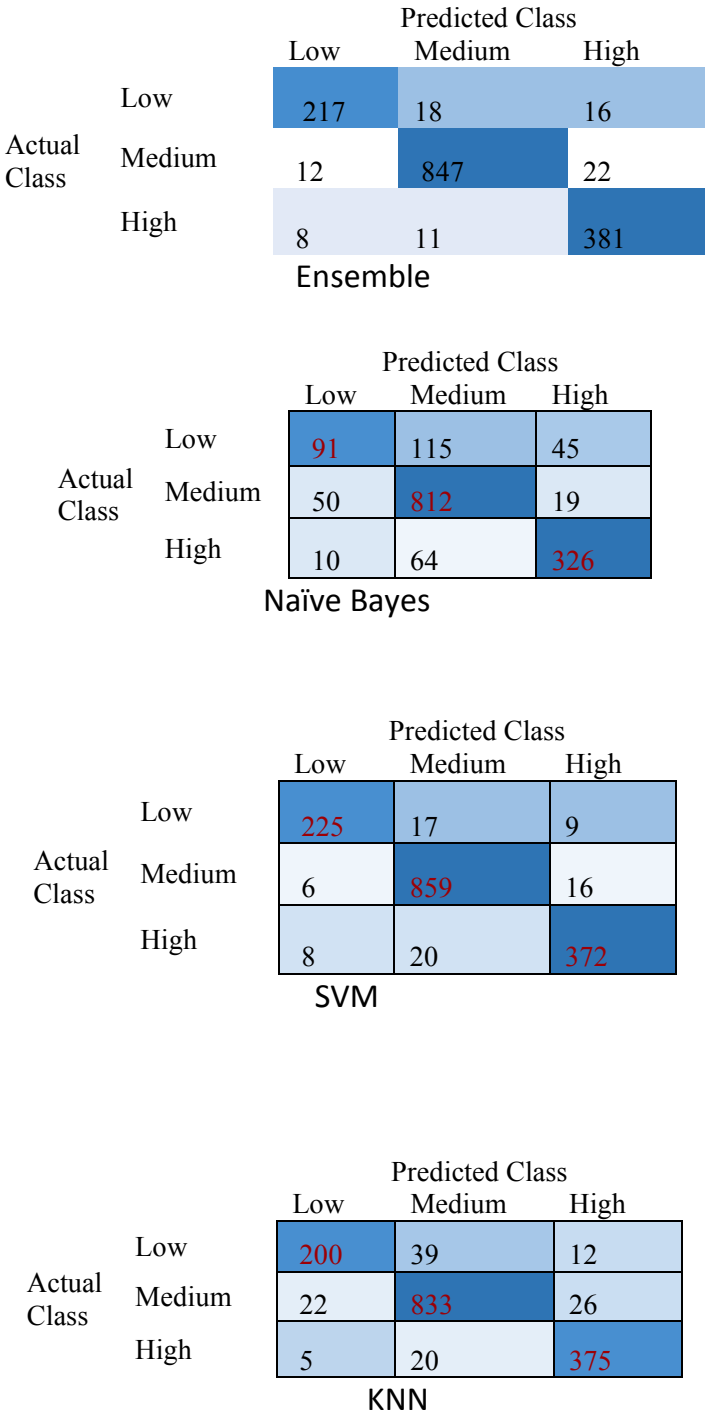
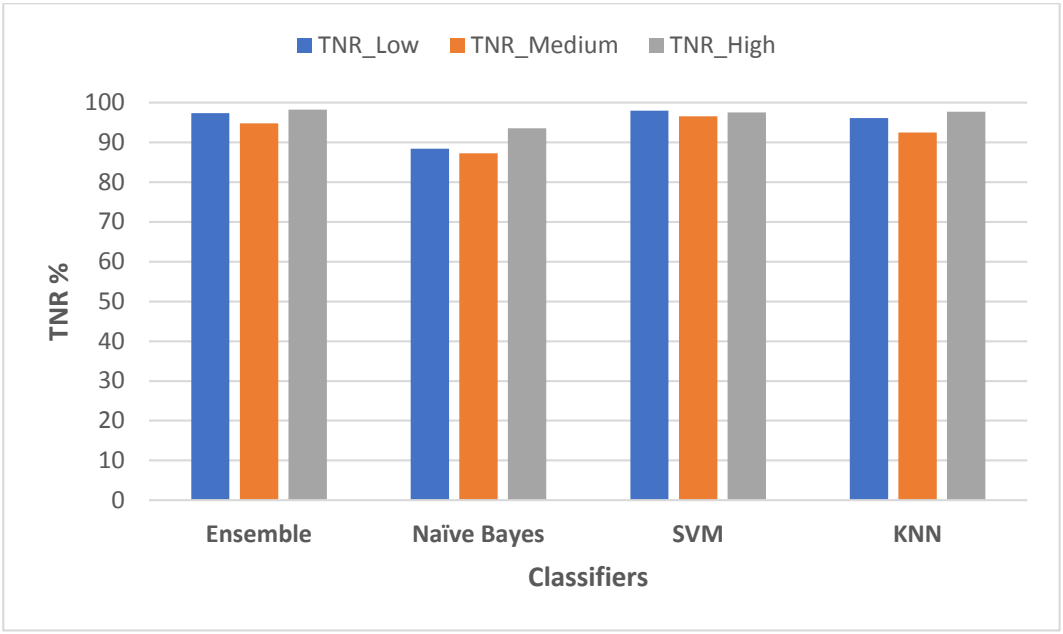
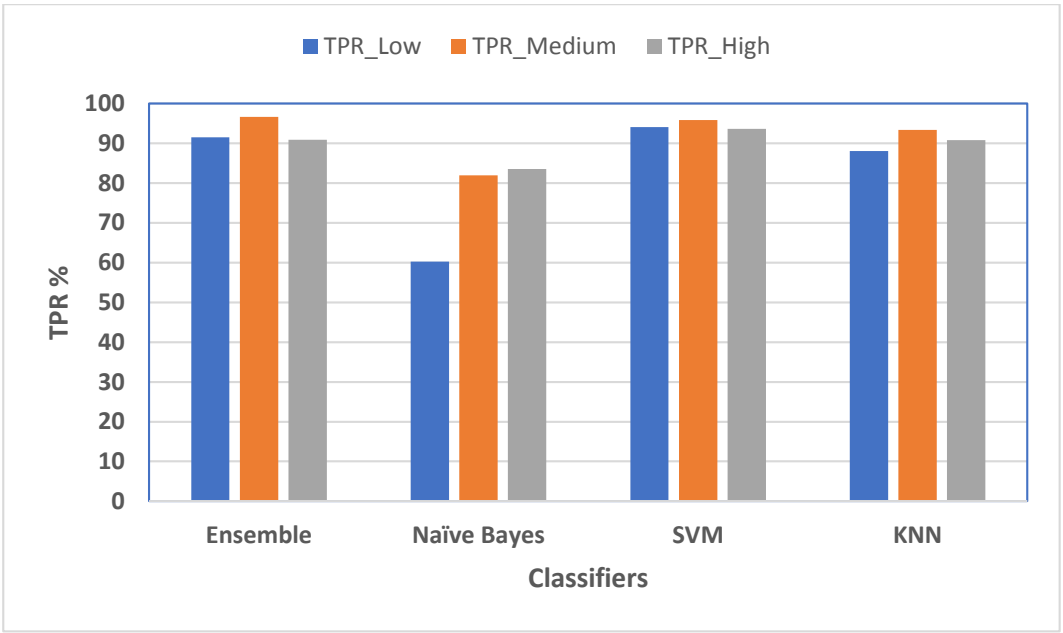


Fig. 12. Confusion matrices of various classifiers.



a



b

Fig. 12. Confusion matrices of various classifiers.

6- Conclusion

In this study, a machine learning-based framework is developed to classify the vulnerability of digital integrated circuits regarding the Hardware Trojans insertions. The major motivation of this paper is the inaccurate and non-comprehensive modeling of vulnerability in the previous studies. The interrelated and complex effective features for vulnerability assessment of ICs hinted to us to utilize different linear classifiers to handle the problem. In our proposed framework, first, a comprehensive dataset of images is generated for the classifier training goal wherein every image represents an implemented circuit that belongs to ISCAS 85 and ISCAS 89 benchmark circuits. Due to the supervised learning approach, every image is labeled to indicate the vulnerability class (low, moderate, and highly vulnerable classes) of the related implemented circuit. We well-trained four famous leaner classifiers (Ensemble, Naïve Bayes, SVM, and KNN) using the generated dataset. SVM and Ensemble classifiers achieve more than 96% accuracy according to the simulation results. The lowest accuracy belongs to Naïve Bayes classifier (~86.81%) which is much better than the best previous studies (72%). As the major axis of future research, we propose extending the vulnerability assessment to the pre-layout stage (RTL stage) as well as the industrial adaptation of the proposed framework for inclusion in the computer-aided design tools (CAD tools) is of great merit.

Funding

There is no funding related to this study.

Data Availability

The datasets generated and/or analyzed during the present study are available from the corresponding author on reasonable request.

Conflict of Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Hassan, R., Meng, X., Basu, K. and Dinakarrao, S.M.P., 2023. Circuit Topology-aware Vaccination-based Hardware Trojan Detection. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Jan 6 (2023).

[2] Moein, S., Gulliver, T. A., Gebali, F., & Alkandari, A. (2016). A new characterization of hardware trojans. IEEE Access, 4, 2721-2731.

[3] Tehranipoor, M., Salmani, H., Zhang, X., Wang, M., Karri, R., Rajendran, J., & Rosenfeld, K. (2010). Trustworthy hardware: Trojan detection and design-for-trust challenges. Computer, 44(7), 66-74.

[4] Hasegawa, K., Seira H., Kohei N., Shinsaku K., and Nozomu T.. "R-HTDetector: Robust hardware-Trojan detection based on adversarial training." IEEE Transactions on Computers 72, no. 2 (2023): 333-345.

[5] Chen, K., Arias, O., Guo, X., Deng, Q. and Jin, Y., 2022. IP-Tag: Tag-Based Runtime 3PIP Hardware Trojan Detection in SoC Platforms. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 42(1), pp.68-81.

[6] Cruz, J., Gaikwad, P., Nair, A., Chakraborty, P. and Bhunia, S., 2022, December. A Machine Learning Based Automatic Hardware Trojan Attack Space Exploration and Benchmarking Framework. In 2022 Asian Hardware Oriented Security and Trust Symposium (AsianHOST) (pp. 1-6). IEEE, Dec. 2022.

[7] Vaziri, M., Rahimifar, M.M. and Jahanirad, H., Robustness Scan of Digital Circuits Using Convolutional Neural Networks. In 2022 12th International Conference on Computer and Knowledge Engineering (ICCKE) (pp. 013-018). IEEE, Nov. 2022.

[8] Salmani, H., & Tehranipoor, M. M. (2016). Vulnerability analysis of a circuit layout to hardware Trojan insertion. IEEE Transactions on Information Forensics and Security, 11(6), 1214-1225.

[9] Bazzazi, A., Shalmani, M. T. M., & Hemmatyar, A. M. A. (2017). Hardware Trojan detection based on logical testing. Journal of Electronic Testing, 33(4), 381-395.

[10] Nowroz, A. N., Hu, K., Koushanfar, F., & Reda, S. (2014). Novel techniques for high-sensitivity hardware Trojan detection using thermal and power maps. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 33(12), 1792-1805.

[11] Kulkarni, A., Pino, Y., & Mohsenin, T. (2016, May). Adaptive real-time Trojan detection framework through machine learning. In 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST) (pp. 120-123). IEEE.

[12] Guazzelli, R. A., Trindade, M. G., Guimarães, L. A., de Paiva Leite, T. F., Fesquet, L., & Bastos, R. P. (2020). Trojan Detection Test for Clockless Circuits. Journal of Electronic Testing, 1-9.

[13] Kulkarni, A., Pino, Y., & Mohsenin, T. (2016, March). SVM-based real-time hardware Trojan detection for many-core platform. In 2016 17th International Symposium on Quality Electronic Design (ISQED) (pp. 362-367). IEEE.

[14] Liakos, K. G., Georgakilas, G. K., Moustakidis, S., Karlsson, P., & Plessas, F. C. (2019, November). Machine Learning for Hardware Trojan Detection: A Review. In 2019 Panhellenic Conference on Electronics & Telecommunications (PACET) (pp. 1-6). IEEE.

[15] Rahimifar, M. M., & Jahanirad, H. (2020, October). Employing Image Processing Techniques for Hardware Trojans Detection. In 2020 10th International Conference on Computer and Knowledge Engineering (ICCKE) (pp. 187-192). IEEE.

[16] Vishnupriya, R., Nirmala Devi, M. (2021). Hardware

Trojan Detection Using Deep Learning-Deep Stacked Auto Encoder. In: Gunjan, V.K., Zurada, J.M. (eds) Proceedings of International Conference on Recent Trends in Machine Learning, IoT, Smart Cities and Applications. Advances in Intelligent Systems and Computing, vol 1245. Springer, Singapore.

[17] Reshma, K., Priyatharishini, M., Nirmala Devi, M. (2019). Hardware Trojan Detection Using Deep Learning Technique. In: Wang, J., Reddy, G., Prasad, V., Reddy, V. (eds) Soft Computing and Signal Processing . Advances in Intelligent Systems and Computing, vol 898. Springer, Singapore.

[18] Priyatharishini, M. and Devi, M.N., 2022. A deep learning based malicious module identification using stacked sparse autoencoder network for VLSI circuit reliability. Measurement, 194, p.111055.

[19] Tehranipoor, M., & Koushanfar, F. (2010). A survey of hardware trojan taxonomy and detection. IEEE design & test of computers, 27(1), 10-25.

[20] Hossein-Talaee, H. and Jahanian, A., 2017, July. Layout vulnerability reduction against trojan insertion using security-aware white space distribution. In 2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI) (pp. 551-555). IEEE.

[21] Trippel, Timothy, Kang G. Shin, Kevin B. Bush, and Matthew Hicks. "ICAS: an extensible framework for estimating the susceptibility of ic layouts to additive trojans." In 2020 IEEE Symposium on Security and Privacy (SP), pp. 1742-1759. IEEE, 2020.

[22] Bhunia, S., Hsiao, M. S., Banga, M., & Narasimhan, S. (2014). Hardware Trojan attacks: Threat analysis and countermeasures. Proceedings of the IEEE, 102(8), 1229-1247.

[23] Hicks, Matthew, Murph Finnicum, Samuel T. King, Milo MK Martin, and Jonathan M. Smith. "Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically." In 2010 IEEE symposium on security and privacy, pp. 159-172. IEEE, 2010.

[24] Waksman, Adam, Matthew Suozzo, and Simha Sethumadhavan. "FANCI: identification of stealthy malicious logic using boolean functional analysis." In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pp. 697-708. 2013.

[25] Sullivan, Dean, Jeff Biggers, Guidong Zhu, Shaojie Zhang, and Yier Jin. "FIGHT-metric: Functional identification of gate-level hardware trustworthiness." In Proceedings of the 51st Annual Design Automation Conference, pp. 1-4. 2014.

[26] Yao, Song, Xiaoming Chen, Jie Zhang, Qiaoyi Liu, Jia Wang, Qiang Xu, Yu Wang, and Huazhong Yang. "FASTrust: Feature analysis for third-party IP trust verification." In 2015 IEEE International Test Conference (ITC), pp. 1-10. IEEE, 2015.

[27] Chen, Xiaoming, Qiaoyi Liu, Song Yao, Jia Wang, Qiang Xu, Yu Wang, Yongpan Liu, and Huazhong Yang. "Hardware trojan detection in third-party digital intellectual property cores by multilevel feature analysis." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 37, no. 7 (2017): 1370-1383.

[28] Salmani, Hassan. "COTD: Reference-free hardware trojan detection and recovery based on controllability and observability in gate-level netlist." IEEE Transactions on Information Forensics and Security 12, no. 2 (2016): 338-350.

[29] Tebyanian, Mahshid, Azadeh Mokhtarpour, and Alireza Shafieinejad. "SC-COTD: Hardware Trojan Detection Based on Sequential/Combinational Testability Features using Ensemble Classifier." Journal of Electronic Testing 37, no. 4 (2021): 473-487.

[30] Kok, Chee Hoo, Chia Yee Ooi, Mehrdad Moghbel, Nordinah Ismail, Hau Sim Choo, and Michiko Inoue. "Classification of Trojan nets based on SCOAP values using supervised learning." In 2019 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1-5. IEEE, 2019.

[31] Kok, Chee Hoo, Chia Yee Ooi, Michiko Inoue, Mehrdad Moghbel, Sreedharan Baskara Dass, Hau Sim Choo, Nordinah Ismail, and Fawnizu Azmadi Hussin. "Net classification based on testability and netlist structural features for hardware trojan detection." In 2019 IEEE 28th Asian Test Symposium (ATS), pp. 105-1055. IEEE, 2019.

[32] Hasegawa, Kento, Masao Yanagisawa, and Nozomu Togawa. "Trojan-feature extraction at gate-level netlists and its application to hardware-Trojan detection using random forest classifier." In 2017 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1-4. IEEE, 2017.

[33] Hasegawa, Kento, Youhua Shi, and Nozomu Togawa. "Hardware trojan detection utilizing machine learning approaches." In 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), pp. 1891-1896. IEEE, 2018.

[34] Hasegawa, Kento, Masao Yanagisawa, and Nozomu Togawa. "Trojan-net classification for gate-level hardware design utilizing boundary net structures." IEICE TRANSACTIONS on Information and Systems 103, no. 7 (2020): 1618-1622.

[35] Abbassi, Imran Hafeez, Faiq Khalid, Osman Hasan, Awais Mehmood Kamboh, and Muhammad Shafique. "McSeVIC: A model checking based framework for security vulnerability analysis of integrated circuits." IEEE Access 6 (2018): 32240-32257.

[36] Khalid, Faiq, Imran Hafeez Abbassi, Semeen Rehman, Awais Mehmood Kamboh, Osman Hasan,

and Muhammad Shafique. "ForASec: Formal Analysis of Hardware Trojan-Based Security Vulnerabilities in Sequential Circuits." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 41, no. 4 (2021): 1167-1180.

[37] Salmani, Hassan, and Mark M. Tehranipoor. "Vulnerability analysis of a circuit layout to hardware trojan insertion." IEEE Transactions on Information Forensics and Security 11, no. 6 (2016): 1214-1225.

[38] Bakhshizadeh, Mahmoud, and Ali Jahanian. "Trojan Vulnerability Map: an efficient metric for modeling and improving the security level of hardware." IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 97, no. 11 (2014): 2218-2226.

[39] Ba, P. S., Dupuis, S., Palanichamy, M., Flottes, M. L., Di Natale, G., & Rouzeyre, B. (2016, July). Hardware trust through layout filling: A hardware Trojan prevention technique. In 2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI) (pp. 254-259). IEEE.

[40] Najm, F. N. (1994). A survey of power estimation techniques in VLSI circuits. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2(4), 446-455.

[41] Nasser, Y., Lorandel, J., Prévotet, J. C., & Hélard, M. (2020). RTL to Transistor Level Power Modelling and Estimation Techniques for FPGA and ASIC: A Survey. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[42] Murray, K. E., Petelin, O., Zhong, S., Wang, J. M., Eldafrawy, M., Legault, J. P., ... & Betz, V. (2020). Vtr 8: High-performance cad and customizable fpga architecture modelling. ACM Transactions on Reconfigurable Technology and Systems (TRETS), 13(2), 1-55.

[43] Acero, C., Feltham, D., Liu, Y., Moghaddam, E., Mukherjee, N., Patyra, M., ... & Zawada, J. (2017). Embedded deterministic test points. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 25(10), 2949-2961.

[44] Breiman, Leo. "Random forests." Machine learning 45 (2001): 5-32.

[45] Friedman, Jerome H. "Greedy function approximation: a gradient boosting machine." Annals of statistics (2001): 1189-1232.

[46] Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, pp. 785-794. 2016.

[47] Savari, M. A., & Jahanirad, H. (2020). NN-SSTA: A deep neural network approach for statistical static timing analysis. Expert Systems with Applications, 149, 113309.

[48] P. Jamieson, K. B. Kent, F. Gharibian, and L. Shannon. 2010. Odin II - An open-source verilog hdl synthesis tool for CAD research. In Proceedings of the International Symposium on Field-Programmable Custom Computing Machines (FCCM'10). 149–156.

[49] Berkley Logic Synthesis and Verification Group. 2018. ABC: A System for Sequential Synthesis and Verification. Retrieved from http://www.eecs.berkeley.edu/ ~ alanmi/abc/.

[50] J. Lamoureux and S.J.E. Wilton, "Activity estimation for Field Programmable Gate Arrays", Proc. Intl Conf. Field-Prog. Logic and Applications (FPL), 2006, pp. 87-94.

HOW TO CITE THIS ARTICLE

H. Jahanirad, M. Fathi. Hardware Trojan vulnerability assessment in digital integrated circuits using learnable classifiers. AUT J Electr Eng, 56(3) (2024) 419-438.

DOI: 10.22060/eej.2024.22910.5572