VeriGene: A Tool for the Creation of DNA Representations for Hardware Security Analysis

Nicholas Haehn, *Dept. of EECS*, *University of Cincinnati*, Cincinnati, OH, USA, haehnna@mail.uc.edu
Bayley King, *Dept. of EECS*, *University of Cincinnati*, Cincinnati, OH, USA, king2b3@mail.uc.edu
Rashmi Jha, *Dept. of EECS*, *University of Cincinnati*, Cincinnati, OH, USA, jhari@ucmail.uc.edu
Temesguen Kebede, *RWYA*, *AFRL*, Dayton, OH, USA, temesgen.kebede.1@us.af.mil
David Kapp, *RWYA*, *AFRL*, Dayton, OH, USA, david.kapp@us.af.mil

Abstract—**Hardware Trojans (HT), or malicious attacks on electronic circuits, are difficult to detect within a design without a novel analysis methods. Biological paradigms of self-defense exist, providing DNA as a possible basis for Hardware Trojan detection. Genetic circuits utilize DNA structure to allow programmable functions in living cells. The programs used to generate genetic circuits are in their infancy, taking considerable time with a limited success rate. In this study, we propose VeriGene a new tool and method for analyzing malicious attacks in electronic circuits, adapting DNA as a basis for viewing and analyzing hardware designs. Utilizing the general structure and encoding of DNA information into individual nucleotides and methods of genetic circuits, we describe a program to generate DNA representations of circuits. Once analyzed with a Neural Network, the DNA representations from infected and uninfected circuits were classified with high accuracy. The results provide evidence for an alternative DNA lens for the hardware security analysis of circuits and the reliability of the VeriGene.**

Index Terms—**Hardware Trojans, Hardware Security, Genetic Circuits, Artificial Intelligence**

I. Introduction

Throughout the years, semiconductor device manufacturing has globalized alongside an increase in the use of third-party Intellectual Property (IP) in hardware design [1], [2]. While software security has been a focus of research for many years, focus shifted to the security of hardware devices [3]. There are multiple hardware based attacks including HT, IP Piracy, Reverse Engineering, Side-Channel Attacks, and Counterfeiting [4].

Shakya [5] and Choo [6] have excellent reviews over HT that discuss different classification methods; in the most general form a HT can be classified as either pre-silicon (RTL, HDL, etc.) or post-silicon (fabricated devices). Of key interest are pre-silicon HT, or malicious modifications to circuits, causing Denial of Service (DOS), information leakage, circuit function modification, or more [7]–[10]. HT include a trigger and a payload, allowing the HT to avoid detection until the trigger occurs [7], [10].

Among other researchers, Collins [11] summarized a taxonomy of HT in FPGA IP, claiming that HT can be categorized into: Trojans that cause malfunction, Trojans that prevent FPGA operation, fault injecting Trojans, Trojans that cause side effects, information leaking Trojans, FPGA resource wasting Trojans, and Trojans that introduce vulnerabilities.

This research was funded by the National Science Foundation (NSF) Research Experiences for Undergraduates (REU) program under the NSF Award Numbers CCF-1718428 and ECCS 1926465.

HT detection and repair techniques are used to identify flaws and attacks in designs. However, when an attack breaches these safeguards, researchers propose a defense or mitigation to said attack. Current HT detection methods mostly include Side-Channel Analysis (SCA) and logical testing in post-silicon, but other techniques including optical inspection, invasive detection, and neural networks have been cited in recent literature [6], [8], [12]. SCA involves observing power, delay, area, and leakage power in reference to the golden, or unmodified, circuit. Logical testing involves testing the logical responses of a circuit in comparison to the golden circuit [10]. Other methods of detection and removal have been proposed, most relevant being genetic (bio-inspired) algorithms and evolvable hardware [8]–[11], [13]. HT are often difficult to detect, and there is a search for novel detection methods to mitigate the rising challenges presented by HT [14], [15].

In a broad sense, malicious attacks on hardware circuits can be considered similar to malicious viral attacks on DNA, as the infection disrupts circuit functionality. Interestingly, there is a rich body of research regarding defense mechanisms against viral attacks in biological systems. These systems can be developed into defense mechanisms for digital circuitry through converted, analogous DNA sequences. Thus, DNA could provide an alternative lens to view HT through genetic circuits.

Genetic circuits are logical processes similar to electronic circuits. However, instead of being implemented in copper and silicon, these circuits are created as synthetic sequences of DNA.

Genetic parts are created by characterizing the behavior of true DNA sequences, classifying the sequences into a defined type of part [16], [17]. Genetic circuits are created from these parts, either manually or by programs, and function in organisms such as E. Coli (Escherichia Coli, prokaryote) or yeast (Saccharomyces cerevisiae, eukaryote) [18]–[20].

Promoters can be triggered from molecules and proteins present in the cell or inserted externally, similar to an electronic circuit [17], [21], [22]. The ribosome binding site (RBS) allows for transcription to begin [17]. The coding DNA sequence (CDS), or gene, acts as the blueprint for the protein, being repressed or activated based on the promoter input.

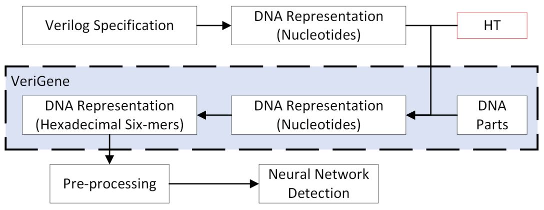

Verilog Specification
DNA Representation (Nucleotides)
HT
VeriGene
DNA Representation (Hexadecimal Six-mers)
DNA Representation (Nucleotides)
DNA Parts
Pre-processing
Neural Network Detection

Fig. 1: A system diagram of VeriGene and its testing.

The ribozyme acts as an insulator for the transcription of the gene segment [19]. The terminator signals the end of the transcription of a gene [23]. Cassettes are genes that allow for the expression of certain proteins. Scars are used to fuse together the different DNA sequence sections [20].

Programs, such as AutoBioCAD [18], Cello [19], [24], and Cello 2.0 [25] offer program-based design solutions for genetic circuits. These often utilize an annealing algorithm, such as Monte Carlo Simulated Annealing, to provide a combination of genetic parts that is most likely to succeed in a living cell [19]. Programs such as iBioSim also exist, which allow for the logical testing of proposed genetic circuit designs [26]. The genetic circuit design softwares often utilize Verilog or other HDL as a basis for the program.

Annealing algorithms take considerable program run-time for a single simple circuit, while also having a relatively low success rate (we observed a success rate of 21.56% across 118,552 Verilog files utilizing Cello 2.0). Additionally, these programs rely upon banks of DNA sequences individually characterized or created by researchers with low variety.

While DNA representations allow for the implementation of bio-inspired algorithms and more for HT detection, we explored a Neural Network based detection approach. The underlying hypothesis was that a DNA representation of circuit will accentuate HT features, making circuits easier to classify by a Neural Network.

For the purposes of analyzing a large number of Verilog Hardware Description Language (HDL) specifications for HT infection, we propose a new DNA generation tool from Verilog, referred to as VeriGene. VeriGene creates DNA representations of hardware circuits that emulate features of genetic circuits. While not meant for creating true genetic circuits, VeriGene is a novel tool for HT analysis through the use of DNA representations. This is not practical with existing genetic circuit design programs due to low success rates and long processing time. To our knowledge, this is the first program created with the purpose of hardware security analysis through DNA representations.

The remainder of the paper is organized as follows: Section II discusses the methodology used in creating and testing VeriGene. Then, Section III presents the results of the Neural Network applied to the sample dataset, and Section IV elaborates on the results.

II. Methodology

A. DNA Part Generation

Based on DNA parts used in the creation of genetic circuits, seven different types of parts are created: promoters, terminators, scars, RBS, ribozymes, cassettes, and CDS.

First, a sequence of a random length within the user-given length range of the DNA part is randomly generated using a, c, t, or g based on the standard DNA nucleotides adenine, cytosine, thymine, and guanine, respectively. If the sequence is repeated, a new sequence is generated until the sequence is unique. Terminators are paired with the promoters and thus must have matching numbers of sequences.

B. Verilog Code Generation

A Verilog code generation program is used to create any number of sample combinational logic circuits to be used to test VeriGene. The program steps through the user-set parameters until every combination of a number of inputs, outputs, and logical gates is reached.

Once the parameters for a specific file are determined, inputs and outputs are created in Verilog followed by gates from the basic combinational logic gates: AND, OR, NOR, NAND, XOR, and XNOR. Every input, output, and wire is used within the circuit, otherwise the program randomly selects two gate inputs and outputs. The given gate is then added to circuit, and the next gate is assigned until all outputs are defined.

VeriGene generates versions of the combinational Verilog that are infected with simple HT based on overall concepts presented on Trust-hub, a hardware security benchmark database [1], [27]–[30]. Each HT type hijacks an existing gate within the design.

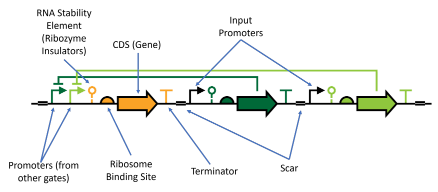
The DOS HT prototype utilizes AND gates to simulate an adversary attempting to disrupt the stability and functionality of the circuit. The complete DOS Prototype attack adds malicious circuitry to every logical gate of the circuit, while the localized attack only attacks a single gate at a time. The Leak HT prototype utilizes either AND, NAND, or NOT gates to facilitate a leak of information to simulate an adversary attempting to gather information from the circuit. The Fault Injection HT utilizes XOR logic gates to flip the output of the existing logical gate to simulate an adversary attempting to alter the functionality of the circuit.

The complete DOS is randomly chosen to occur 1 out of every 500 circuits. Otherwise, the type of attack will be randomly chosen between the three remaining attacks. A comment is added to every line of Verilog that includes an HT for human readability.

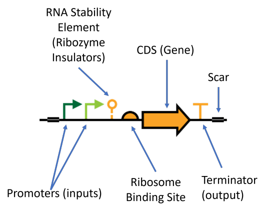
If an analysis closer to true genetic circuits or transistor logic is desired [21], the Verilog circuit can be converted to logically similar NOR-based, or repression-based, Verilog.

C. DNA Representation Creation

DNA representations created by VeriGene utilize a modified structure compared to typical genetic circuits. As shown in Figure 2a, genetic circuits created by Cello function on gene


RNA Stability Element (Ribozyme Insulators)
CDS (Gene)
Input Promoters
Promoters (from other gates)
Ribosome Binding Site
Terminator
Scar

(a) Sample model of an AND gate as would be produced by Cello 2.0. This model utilizes repression-based logic, utilizing three gene sections. The two external promoters receive an input, which sends a signal to the internal promoters of the first genetic section. When both external promoters are present, the output of the entire genetic circuit will appear out of the orange CDS. This output would mimic an AND-gate response. Each terminator stops transcription of that specific genetic section.



(b) Sample model of an AND gate as created with the VeriGene using repurposed DNA parts. When either of the two external inputs are high, the CDS will transcribe the protein, with the terminator type signalling the type of output for the given circuit.

Fig. 2: Synthetic Biology Open Language Visual (SBOL Visual) diagrams of AND gates produced from Cello 2.0 and VeriGene.

repression. Each gene codes for a specific output protein. The terminator then signals the end of transcription for that gene.

The DNA basis only must provide an alternative lens to view the Verilog specification, rather than fully functional DNA for implementation in living cells as genetic circuits. With this modification, the representative DNA sequences can be created more quickly and reliably than true genetic circuits.

As shown in Figure 2b, the DNA representations work based on standard logic as opposed to repression-based logic. The promoters act as inputs to the gate, while the gene provides the functionality of the gate. The terminator was repurposed to signal the desired output of the gate, rather than the end of transcription.

Other DNA elements meant for stability and connecting DNA elements were preserved in the DNA representations. Since the terminator is used to signal the output of the specific gate, genes are open to be used to signal the type of logic being used. This allows for more scalability of the DNA sequences using VeriGene. In standard DNA structuring, a gene produces a specific protein, which is used by other gates to trigger promoters, requiring a large number of gene DNA sequences.

To create the DNA representations, DNA parts are loaded into the program. Each promoter is paired to a unique terminator in order to allow for identifiers to activate sections of the DNA representations. Then, each CDS is assigned a specific use for the DNA representations according to the standard combinational logic gates: NOT, NOR, OR, AND, NAND, XNOR, and XOR.

Combinational Verilog specifications can be created as in Section II-B or inserted externally to create the DNA representations. VeriGene is currently limited to combinational logic circuits due to the structuring of the DNA part assignment, but VeriGene could be expanded to include structural logic functionality in the future. The Verilog is read by storing all the circuit and gate inputs, outputs, wires, and functionality. Then, DNA parts are assigned. First, promoters are assigned to a given circuit input, and terminators are assigned to a given circuit output. A promoter and terminator must both be assigned to a single wire, as wires are used for internal circuit communication.

A scar beings the creation of the DNA representation. Based on the assigned promoters for the gate inputs, the promoter DNA sequences are added to the circuit sequence. A ribozyme and RBS is then added to the circuit, mimicking genetic circuit structure. The CDS is then added to the circuit sequence based on the gate functionality. Finally, a terminator is added to the circuit sequence based on the gate output. This process is repeated for every gate within the circuit, with a scar in between every gate sequence.

Following gate assignment, the final outputs of the entire circuit are specified using cassettes. The promoters paired to the terminators of the circuit outputs are added. Then, a unique cassette sequence is given to that output, which is then followed by a scar. The cassette is used in this manner to mimic the output style as produced by Cello 2.0.

One percent of the nucleotides (a, c, t, or g), are randomly reassigned, or mutated, to prevent Neural Network over-learning. This is similar to adding pixelation to images used for image-based Neural Networks.

Next, the DNA representation sequence is converted into K-mers. K-mers allow for text-classification Neural Networks to be able to analyze the DNA sequence, as the DNA is represented into a series of DNA ”words”. K-mers also allow for the Neural Network to maintain context of the original DNA sequence, which analysis by single base pairs would not allow [31]. In the case of VeriGene, six-mers were created, where six characters comprise each DNA ”word” created using a moving window, as shown in Figure 3. Six-mers were chosen to provide a similar length compared to other studies with k=5, while increasing the k-value by one to decrease file size [31].

After producing the K-mers, the sequence is vectorized into hexadecimal values corresponding to the K-mer to reduce storage usage. Each K-mer is first converted into a number in base-four, due to each character in the K-mer having four options: a, c, t, or g. This base-four number is then converted into base-16, which is formatted into standard hexadecimal format. A Verilog specification and its respective DNA sequence is shown in Figure 4.

D. Preparing Data for Neural Network Analysis

Each vectorized DNA sequences is randomly assigned to the testing set with a 60% chance, training set with a 25% chance, or validation set with a 15% chance. The training

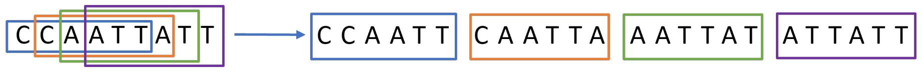


Fig. 3: An example of a 9 character long DNA sequence split into six-mers. Each colored block on the left represents one six-mer.

```
module Example(output out0, input in0, in1);
    wire wire0, wire1, wire2, wire3;

    nand(wire0,in1,in0);
    xnor(wire1,wire0,in1);
    nand(wire2,wire0,in0);
    and(wire3,wire1,in0);
    or(out0,wire2,wire3);
endmodule
```

(a) A sample combinational Verilog module as created by the Verilog generation program. This Verilog HDL specification includes two inputs (in0 and in1) and one output (out0). The circuit includes 4 wires (wire0 through wire3) and 5 gates. Each gate is represented as the gate's type (NAND), followed by the gate's output (wire0), and the gate's inputs (in1 and in0).

```
5D7 5F1 D7C F13 7C4 139 C46 395 469 95A 69E
5AF 9E7 AF5 E7D F5F 7D3 5F0 D3C F0F 3CB 0FA
CB2 FAC B23 AC4 239 C4E 393 4E0 934 E05 345
05D 457 5D9 576 D99 762 998 622 980 224 801
24C 013 4C4 139 C4A 39E 4AB 9E2 AB8 E22 B80
22C 80B 2CE 0BF CEF BF7 EFD F7B FDA 7BA DAA
BA2 AA4 A29 A4E 293 4E0 938 E0A 38E 0AB 8E6
AB1 E60 B10 600 10C 007 0C1 078 C1E 78B 1EA
8BE EA3 BE8 A36 E8D 363 8D4 639 D4E 397 ...
```

(b) A small portion of the complete vectorized K-mers from the DNA sequence created from the provided Verilog specification. Each three-character section represents six-mer translated into a unique hexadecimal representation.

Fig. 4: A sample Verilog specification and its respective vectorized DNA sequence.

dataset is used to train the Neural Network model's layers, adding weights to each layer to better predict the model. The validation dataset is used to validate the weights set by the training dataset. The test dataset is used as a reference dataset to evaluate the trained model.

The percentage of a given Verilog file and respective DNA representation that is made of HT material is calculated. The infection percentage in a Verilog fileis determined by how many gates are related to HTs, as marked by comments during the creation of the Verilog dataset. To determine the percentage of infections in the DNA representation, the DNA representation creation process is replicated. It is noted what portion of the DNA characters occur due to an HT as indicated by the Verilog comments.

E. Sample Dataset Generation

To demonstrate the effectiveness of VeriGene at creating DNA representations for classification, a sample dataset was created and tested in a Neural Network.

To increase the variety of Verilog specifications delivered to VeriGene, data storage, and Neural Network efficiency, the Verilog creation process of Section II-B was modified. The Verilog was generated using inputs and outputs in ranges of powers of two from 2 to 64. 1 to 64 internal gates were used in each combination of inputs and outputs, with a total of 1,548 uninfected and infected files each created. These input, output, and gate combinations were selected to provide a large variety in design sizes without exhaustively testing each combination level to save storage space. Benchmark circuits were not able to be tested as most benchmark circuits available do not follow the format that VeriGene expects, including the addition of sequential logic [1]. Additionally, larger circuits were not tested, as the DNA representations become too large to store. Once these Verilog specifications were created, they were converted to NOR-based logic.

The standard combinational logic and NOR-based logic Verilog specifications were converted into DNA representations as in Section II-C and split into training, testing, and validation datasets. All versions of the data were then tested using the method described in Section II-D to determine the infection rates of each set.

F. Testing and Classification of DNA Representations

The sample datasets from Section II-E were fed into a sequential one-dimensional convolutional TensorFlow Neural Network utilizing the Keras library. The data was split into 32 batches over 30 epochs for standard combinational logic and 20 epochs for NOR-based logic. The network consisted of the following layers in order: an embedding layer, a convolutional layer with "relu" activation, a dropout layer, a global max pooling layer, another dropout layer, and a dense layer. A convolutional neural network was chosen due to existing DNA classification applications with k-mers and training time advantages compared to LSTM networks [32].

Once the model was trained, the model was applied to predict the classes of the test dataset. A confusion matrix was prepared by comparing the predictions against the true class. A confusion matrix allows the viewer to determine the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). Precision was determined as the number of true positives out of the number of predicted positives. Recall was calculated as the true positives over the amount of actual positives. F1 score combined precision (p) and recall (r) metrics.

III. Results

All 3,096 Verilog specifications were converted to NOR-based logic using VeriGene. Furthermore, all Verilog specifications were converted into DNA representations. The NOR-based logic and Standard logic DNA representation datasets were split into Training, Validation, and Testing datasets, as shown by Table I.

	Standard Logic		NOR-based Logic	
	Uninfected	Infected	Uninfected	Infected
Training	950	910	910	921
Validation	216	259	234	228
Testing	382	379	404	399
Total	1,548	1,548	1,548	1,548

TABLE I: Dataset split between uninfected and infected files of Standard and NOR-based logic.

		Predicted Class		
		Uninfected	Infected	Total
Actual Class	Uninfected	343	39	382
	Infected	24	255	279
	Total	367	294	661

(a) Confusion matrix for the Neural Network predictions on the test dataset using DNA representations using standard logic.

		Predicted Class		
		Uninfected	Infected	Total
Actual Class	Uninfected	395	9	404
	Infected	53	346	399
	Total	448	355	803

(b) Confusion matrix for the Neural Network predictions on the test dataset using DNA representations using NOR-based logic.

TABLE II: Confusion matrices for Neural Network predictions.

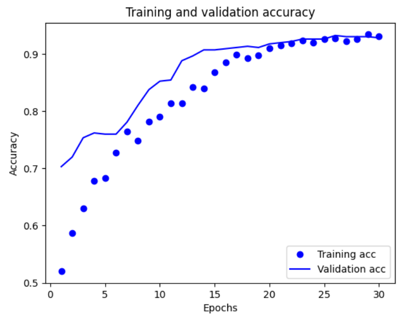
HT makeup in Verilog and DNA representations were not dissimilar. Infected standard combinational logic Verilog was calculated to have a median of 7.14% infected lines with standard deviation of 9.25%. For infected NOR-based logic Verilog specifications, the median was 7.04% infected lines with a standard deviation of 9.40%. For the DNA representations made from infected standard combinational logic, the median was 5.68% infected lines with standard deviation of 6.52%. DNA representations made from infected NOR-based logic, meanwhile, had a median of 6.54% infected lines with a standard deviation of 8.37%.

All datasets were successfully run through the Neural Networks. The DNA representations created from standard logic saw training loss of 0.1735, validation loss of 0.2221, and testing loss of 0.2397 across 30 epochs. The training accuracy reached 96.02%, validation accuracy reached 92.84%, and testing accuracy reached 91.72%. This trial's training and validation accuracy and loss across the 30 epochs are shown in Figures 5a and 5b, respectively. The confusion matrix for this trial is shown in Table IIa, with a 0.8673 precision, 0.9140 recall, and 0.8901 F1 score.

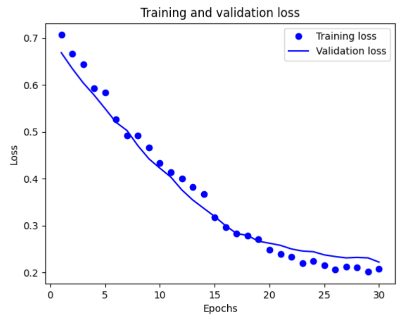
Using DNA representations created from NOR-based Verilog across 20 epochs, training loss was 0.2888, validation loss was 0.3029, and testing loss was 0.3062. Training accuracy was 91.64%, validation accuracy was 92.64%, and testing accuracy reached 92.28%. This trial's training and validation accuracy and loss across the 20 epochs are shown in Figures 5c and 5d, respectively. The confusion matrix for this trial is shown in Table IIb, with a 0.9746 precision, 0.9672 recall, and 0.9178 F1 score.

IV. Discussion

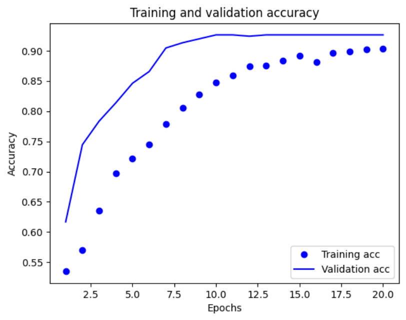
As shown by the results in Section III, Verilog specifications can be generated, converted to NOR-based logic, converted



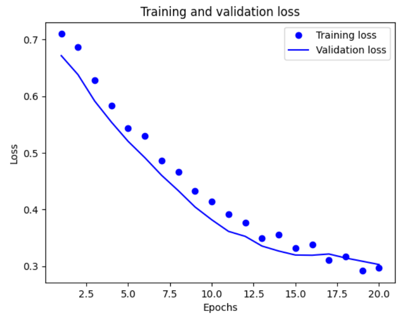
(a) TensorFlow Neural Network accuracy results from standard combinational logic DNA representations.



(b) TensorFlow Neural Network loss results from standard combinational logic DNA representations.



(c) TensorFlow Neural Network accuracy results from NOR-based logic DNA representations.



(d) TensorFlow Neural Network loss results from NOR-based logic DNA representations.

Fig. 5: TensorFlow Neural Network results for standard combinational logic DNA representations and NOR-based DNA representations. Both trials shown here utilize a 60% training, 20% testing, and 20% validation dataset split.

into DNA representations, and pre-processed and analyzed. The runtime to create a DNA representation from the Verilog specification is approximately 0.25 seconds, which is significantly less time than existing genetic circuit programs. Additionally, files can be analyzed in the trained neural network in less than 5 minutes.

With the ability to convert Verilog into DNA representations, a different and novel context is provided for the analysis of hardware circuits and malicious attacks on them. Across all datasets, there was a high standard deviation in HT make-up, and all datasets were skewed right. Despite the change in context of the circuit specifications, HTs were not represented in statistically significant different magnitudes between DNA representations and Verilog in both standard and NOR-based logic. This demonstrates that the DNA representations allow for a similar but different context to Verilog specifications. The difference in the median and standard deviation between the Verilog and DNA representations can likely be accounted to the length and variation in the DNA parts used to create the DNA representations.

Compared to methods for creating genetic circuits from Verilog specifications such as Cello and Cello 2.0, there was a greater success rate in DNA representation creation and variety utilizing VeriGene. Hardware Security analysis does not require that the circuits perform perfectly in living cells. Rather, DNA can simply provide a different context from Verilog, removing the need for complex algorithms, improving the success rate of DNA creation. Available DNA sequences for DNA parts tend to be similar overall to each other, which can be difficult to distinguish. The randomly generated DNA parts can vary greatly and be created in any number to increase the scale of DNA representations.

As shown by the results of the Neural Networks, both DNA representations created from standard combinational logic and NOR-based logic can be analyzed for HT detection. Classification accuracy on the testing datasets using the DNA sequence representations reached values above 91% for all trials. This demonstrates that the model, once trained, can accurately classify the DNA representations of both NOR-based and standard combinational logic. Analyzing the loss graphs in Figures 5b and 5d, the training and validation loss for both trials remained relatively close to each other across the entirety of training. This represents minimal over-learning of the model throughout training. Due to the lack of differences in infection rates and classification of Trojans between standard and NOR-based logic, standard logic is preferred for future analysis. However, NOR-based logic provides a more comparable relationship to the true functioning of DNA, which will likely be useful in applying genetic-based algorithms.

Furthermore, both trials resulted in high F1 scores of 0.8901 and 0.9178 for standard combination logic and NOR-based logic, respectively. This shows that the models were well suited for the classification of the DNA representations. Both trials presented relatively minimal overall false positives and false negatives. In Table IIa, the trained model is shown to lean on the side of over-predicting infected circuits by a precision of 0.8673. Meanwhile, Table IIb, shows a higher precision and recall with NOR-based logic DNA representations.

Overall, both models and trials for standard and NOR-based logic DNA representations demonstrate successful classification. Based on these results, VeriGene can be used to create DNA representations for the hardware security analysis of circuit specifications. With the ability to effectively classify circuits as either infected or uninfected, Neural Networks show one valid option for analysis of DNA representations of Verilog specifications.

Some methods of pre-silicon analysis, including feature extraction and other neural network based approaches, have shown results with higher accuracy than the results presented with the Neural Network in this paper [12]. VeriGene's purpose, however, is to provide an alternative lens on the HDL circuit for the application of future algorithms. While a simple neural network was used to analyze the DNA representations in this work, the results from VeriGene can be applied to genetic-based algorithms or other neural-network approaches to possibly achieve a higher accuracy than in the HDL state.

V. Conclusion

In this work, VeriGene, a new tool for the analysis of HTs and other hardware security attacks in Verilog utilizing DNA representations was presented. To our knowledge, VeriGene is the first program for DNA representation creation from Verilog for the purpose of hardware security analysis.

To produce DNA "words," the DNA sequence is split into K-mers of 6 characters in length and vectorized into a hexadecimal format. The Verilog and DNA sequences can be tested using VeriGene to determine the amount of a given specification that is due to HT.

To test the effectiveness of utilizing DNA representations to analyze hardware security in Verilog specifications, sample combinational logic and NOR-based logic Verilog files were created using VeriGene and tested in a TensorFlow and Keras Neural Network.

During this testing, all Verilog specifications were successfully transitioned into DNA representations, illustrating the scalability and reliability of the system. HTs presented themselves similarly in both Verilog and DNA representations in terms of circuit make-up. The Neural Network was able to successfully classify the DNA sequences with a high degree of accuracy. This demonstrated that the DNA representations offered a valid lens to analyze HTs in circuit specifications. Although a Neural Network was used in this study, other classification methods could be used to achieve higher accuracy.

Based on the results presented, DNA representations provide a viable alternative for the analysis of hardware circuits for possible hardware security faults. VeriGene allows for rapid and reliable production of these DNA representations for simple combinational logic. This novel lens for hardware security analysis could be expanded to include logic beyond simple combinational logic as well as biological detection and repair techniques. This would allow for the analysis of more complex circuitry and HTs.

Data Availability

The code for VeriGene is available on the corresponding author's GitHub: github.com/nicholash85/VeriGene. Raw Verilog and DNA representation data are available on reasonable request to the corresponding author at haehnna@mail.uc.edu.

Author contributions statement

N.H. researched and tested genetic circuit generation programs, created the DNA representation creation tools, and wrote the manuscript. B.K. and R.J. provided N.H. technical and professional guidance throughout the project and assisted with the manuscript. R.J. conceived the concept of utilizing genetic concepts for the detection of HTs. T.K. and D.K. provided feedback to N.H.. All authors reviewed and revised the manuscript.

References

[1] K. Xiao and M. Tehranipoor, "Tutorial: Hardware Trojan Insertion on FPGA."

[2] D. Staub, R. Jha, and D. Kapp, "A CRISPR-Cas-Inspired Mechanism for Detecting Hardware Trojans in FPGA Devices," *CoRR*, vol. abs/2005.07332, 2020. [Online]. Available: https://arxiv.org/abs/2005.07332

[3] S. Koley and P. Ghosal, "Addressing hardware security challenges in internet of things: Recent trends and possible solutions," in *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*. IEEE, 2015, pp. 517–520.

[4] M. Rostami, F. Koushanfar, and R. Karri, "A Primer on Hardware Security: Models, Methods, and Metrics," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[5] B. Shakya, T. He, H. Salmani, D. Forte, S. Bhunia, and M. Tehranipoor, "Benchmarking of hardware trojans and maliciously affected circuits," *Journal of Hardware and Systems Security*, vol. 1, no. 1, pp. 85–102, 2017.

[6] H. S. Choo, C. Y. Ooi, M. Inoue, N. Ismail, and C. H. Kok, "A review of hardware trojan detection: An overview of different pre-silicon techniques," pp. 1–21, 2020. [Online]. Available: https://uc.idm.oclc.org/login?qurl=https%3A%2F%2Fwww.proquest.com%2Fother-sources%2Freview-hardware-trojan-detection-overview%2Fdocview%2F2394536604%2Fse-2%3Faccountid%3D2909

[7] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware Trojan Attacks: Threat Analysis and Countermeasures," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1229–1247, 2014.

[8] S. Bhasin and F. Regazzoni, "A survey on hardware trojan detection techniques," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2015, pp. 2021–2024.

[9] H. Li, Q. Liu, and J. Zhang, "A survey of hardware Trojan threat and defense," *Integration*, vol. 55, pp. 426–437, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167926016000067

[10] C. S. Sruthi, M. Lohitha, S. K. Sriniketh, D. Manassa, K. Srilakshmi, and M. Priyatharishini, "Genetic Algorithm based Hardware Trojan Detection," in *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*, vol. 1, 2021, pp. 1431–1436.

[11] Z. Collins, "Hardware trojans in fpga device ip: Solutions through evolutionary computation," Ph.D. dissertation, University of Cincinnati, 2019.

[12] R. Yasaei, S. Faezi, and M. A. A. Faruque, "Golden reference-free hardware trojan localization using graph convolutional network," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 1–11, 2022.

[13] B. King, R. Jha, T. Kebede, and D. Kapp, "Securing 3rd-party hdl ip: a feasibility study using evolutionary methods," *Journal of Hardware and Systems Security*, pp. 1–15, 2022, reprinted in Springer.

[14] S. Wei and M. Potkonjak, "The undetectable and unprovable hardware trojan horse," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2013, pp. 1–2.

[15] Y. Jin, N. Kupp, and Y. Makris, "Experiences in hardware trojan design and implementation," in *2009 IEEE International Workshop on Hardware-Oriented Security and Trust*. IEEE, 2009, pp. 50–57.

[16] A. A. K. Nielsen and C. A. Voigt, "Multi-input CRISPR/Cas genetic circuits that interface host regulatory networks," *Molecular Systems Biology*, vol. 10, no. 11, p. 763, nov 2014. [Online]. Available: https://doi.org/10.15252/msb.20145735

[17] D. Densmore and S. Hassoun, "Design automation for synthetic biological systems," *IEEE Design and Test of Computers*, vol. 29, no. 3, pp. 7–20, 2012.

[18] G. Rodrigo and A. Jaramillo, "AutoBioCAD: Full Biodesign Automation of Genetic Circuits," *ACS Synthetic Biology*, vol. 2, no. 5, pp. 230–236, may 2013. [Online]. Available: https://doi.org/10.1021/sb300084h

[19] A. A. K. Nielsen, B. S. Der, J. Shin, P. Vaidyanathan, V. Paralanov, E. A. Strychalski, D. Ross, D. Densmore, and C. A. Voigt, "Genetic circuit design automation," *Science*, vol. 352, no. 6281, p. aac7341, apr 2016. [Online]. Available: http://science.sciencemag.org/content/352/6281/aac7341.abstract

[20] Y. Chen, S. Zhang, E. M. Young, T. S. Jones, D. Densmore, and C. A. Voigt, "Genetic circuit design automation for yeast," *Nature Microbiology*, vol. 5, no. 11, pp. 1349–1360, 2020. [Online]. Available: https://doi.org/10.1038/s41564-020-0757-2

[21] M. Taketani, J. Zhang, S. Zhang, A. J. Triassi, Y.-J. Huang, L. G. Griffith, and C. A. Voigt, "Genetic circuit design automation for the gut resident species Bacteroides thetaiotaomicron," *Nature Biotechnology*, vol. 38, no. 8, pp. 962–969, 2020. [Online]. Available: https://doi.org/10.1038/s41587-020-0468-5

[22] J. A. N. Brophy and C. A. Voigt, "Principles of genetic circuit design," *Nature Methods*, vol. 11, no. 5, pp. 508–520, 2014. [Online]. Available: https://doi.org/10.1038/nmeth.2926

[23] T. Nguyen, T. S. Jones, P. Fontanarrosa, J. V. Mante, Z. Zundel, D. Densmore, and C. J. Myers, "Design of Asynchronous Genetic Circuits," *Proceedings of the IEEE*, vol. 107, no. 7, pp. 1356–1368, 2019.

[24] CidarLab, "Cello," https://github.com/CIDARLAB/cello, 2016.

[25] T. S. Jones, S. Oliveira, C. J. Myers, C. A. Voigt, and D. Densmore, "Genetic circuit design automation with cello 2.0," *Nature Protocols*, vol. 17, no. 4, pp. 1097–1113, 2022.

[26] L. Watanabe, T. Nguyen, M. Zhang, Z. Zundel, Z. Zhang, C. Madsen, N. Roehner, and C. Myers, "iBioSim 3: A Tool for Model-Based Genetic Circuit Design," *ACS Synthetic Biology*, vol. 8, no. 7, pp. 1560–1563, jul 2019. [Online]. Available: https://doi.org/10.1021/acssynbio.8b00078

[27] H. Salmani, M. Tehranipoor, and R. Karri, "On design vulnerability analysis and trust benchmarks development," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, 2013, pp. 471–474.

[28] B. Shakya, T. He, H. Salmani, D. Forte, S. Bhunia, and M. Tehranipoor, "Benchmarking of Hardware Trojans and Maliciously Affected Circuits," *Journal of Hardware and Systems Security*, vol. 1, no. 1, pp. 85–102, 2017. [Online]. Available: https://doi.org/10.1007/s41635-017-0001-6

[29] J. Harrison, N. Asadizanjani, and M. Tehranipoor, "On malicious implants in PCBs throughout the supply chain," *Integration*, vol. 79, pp. 12–22, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167926021000304

[30] S. Amir, B. Shakya, X. Xu, Y. Jin, S. Bhunia, M. Tehranipoor, and D. Forte, "Development and Evaluation of Hardware Obfuscation Benchmarks," *Journal of Hardware and Systems Security*, vol. 2, no. 2, pp. 142–161, 2018. [Online]. Available: https://doi.org/10.1007/s41635-018-0036-3

[31] R. Rizzo, A. Fiannaca, M. La Rosa, and A. Urso, "A Deep Learning Approach to DNA Sequence Classification," in *Computational Intelligence Methods for Bioinformatics and Biostatistics*, C. Angelini, P. M. V. Rancoita, and S. Rovetta, Eds. Cham: Springer International Publishing, 2016, pp. 129–140.

[32] M. A. Lo Bosco Giosué and Di Gangi, "Deep Learning Architectures for DNA Sequence Classification," in *Fuzzy Logic and Soft Computing Applications*, V. Petrosino Alfredo and Loia and P. Witold, Eds. Cham: Springer International Publishing, 2017, pp. 162–171.