NSPG: Natural language Processing-based Security Property Generator for Hardware Security Assurance

Xingyu Meng
The University of Texas at Dallas, USA

Amisha Srivastava
The University of Texas at Dallas, USA

Ayush Arunachalam
The University of Texas at Dallas, USA

Avik Ray
Amazon Alexa, USA

Pedro Henrique Silva
Technology Innovation Institute, UAE

Rafail Psiakis
Technology Innovation Institute, UAE

Yiorgos Makris
The University of Texas at Dallas, USA

Kanad Basu
The University of Texas at Dallas, USA

ABSTRACT

The efficiency of validating complex System-on-Chips (SoCs) is contingent on the quality of the security properties provided. Generating security properties with traditional approaches often requires expert intervention and is limited to a few IPs, thereby resulting in a time-consuming and non-robust process. To address this issue, we, for the first time, propose a novel and automated Natural Language Processing (NLP)-based Security Property Generator (NSPG). Specifically, our approach utilizes hardware documentation in order to propose the first hardware security-specific language model, HS-BERT, for extracting security properties dedicated to hardware design. It is capable of phasing a significant amount of hardware specification, and the generated security properties can be easily converted into hardware assertions, thereby reducing the manual effort required for hardware verification. NSPG is trained using sentences from several SoC documentations and achieves up to 88% accuracy for property classification, outperforming ChatGPT. When assessed on five untrained OpenTitan hardware IP documents, NSPG aided in identifying eight security vulnerabilities in the buggy OpenTitan SoC presented in Hack@DAC 2022.

KEYWORDS

Hardware Security Property, Property Generation

1 INTRODUCTION

Modern computing systems are built on System-on-Chips (SoCs), as they offer a high level of integration through the use of multiple Intellectual Property (IP) cores. However, this also presents new security challenges, since vulnerabilities in one IP core may affect the security of the entire system [4]. Hence, hardware security validation is imperative to ensure the security and trustworthiness of the design. MITRE's Common Weakness Enumeration (CWE) for hardware categorizes commonly encountered security weaknesses in hardware designs, including issues with security flow, privilege and access control, reset control, memory and storage, peripherals and on-chip fabric, and debugging and test [3].

DAC '24, June 23–27, 2024, San Francisco, CA, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0601-1/24/06
https://doi.org/10.1145/3649329.3656255

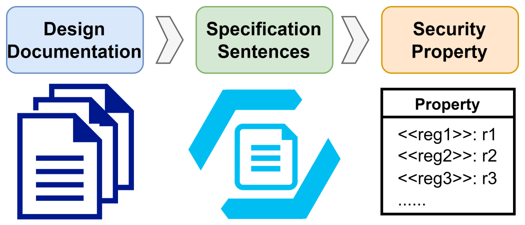

Figure 1: Security property generation from documents.

Traditional property generation approaches, which often require the experience of developers, are time-intensive and non-robust [4]. However, as shown in Figure 1, we reason that it is possible to generate numerous security properties by analyzing operation details from technical documents. Existing research in the biomedical field, such as BioBERT, has shown that text and documents can provide essential information in specialized domains to fine-tune the general Bidirectional Encoder Representations from Transformers (BERT) language model for text generation and phrase classification [1].Hence, we intend to apply this concept to the hardware security domain and develop a fully automatic security property generation engine called NLP-based Security Property Generator engine (**NSPG**). NSPG can significantly reduce the manual effort necessary to automatically generate security properties for SoC verification.

To the best of our knowledge, NSPG is the first technique utilizing SoC documentation to generate SoC security properties. Our contributions are summarized as follows:

(1) We propose an NLP-based security property generation framework, NSPG, to automatically mine security property-related sentences from the SoC documents, assisting in the detection of security vulnerabilities in RTL.
(2) We present a complete end-to-end framework with hardware domain-specific knowledge and data modification techniques to improve the performance of the proposed HS-BERT model by analyzing hardware documentation.
(3) NSPG is evaluated on five unseen OpenTitan design documents and all generated security properties are validated. Furthermore, we apply these properties to search for security violations in the OpenTitan design used in Hack@DAC 2022 and identify eight bugs [8].
(4) To further validate the effectiveness of our methodology and the security properties extracted by NSPG, we compare it against ChatGPT [13].

The rest of this paper is organized as follows. Section 2 introduces the background. Section 3 includes the proposed technique. Section 4 demonstrates the evaluation of the proposed technique. Section 5 discusses the capabilities of the proposed technique. Section 6 presents the related works on security property generation. Finally, Section 7 concludes our paper.

2 BACKGROUND

In this section, we will provide some background on HS properties, and NLP used for developing the proposed NSPG framework.

2.1 Hardware Security (HS) Property

The specification of security properties tends to vary based on the chosen method of security analysis, resulting in specifications that are often specific to the employed verification tools or models. Listing 1 presents an example of SystemVerilog assertion based on the description of the security property. It shows that in order to generate an appropriate term for a verification mechanism, the descriptions of security properties must contain strong reasoning and details of the operation. Listing 2 shows a paragraph from the OpenTitan AES document [12]. The sentences marked in blue are security property-related.

Listing 1: Security Property and SystemVerilog Assertion

```
Security Property Description:
If the AES unit wants to finish encryption/decryption
of a data block but the previous output data has not
yet been read by the processor, AES unit is stalled.

SystemVerilog Assertion:
assert  property (
  (posedge clk) disable iff (rst)// Security Property
    aes.done |-> aes.out == $past(aes.key)
  )
  else                            // Error Message
    $error("%m previous key has not been read");
```

Listing 2: A example paragraph in AES Document

```
Also, there is a back-pressure mechanism for the
output data. If the AES unit wants to finish the
encryption/decryption of a data block but the previous
output data has not yet been read by the processor,
the AES unit is stalled. It hangs and does not drop
data. The order in which the output registers are read
does not matter. Every output register must be read
at least once for the AES unit to continue. This is
 the default behavior. It can be disabled by setting
the MANUAL_OPERATION bit in CTRL_SHADOWED to 1.
```

It can be observed that the security properties exhibit distinct characteristics, including domain-specific terms and relational contexts. Thus, pinpointing sentences relevant to security properties can enhance the language model's understanding and by emphasizing these aspects during fine-tuning, we can better identify and enhance security-related content within sentences. Further details about the extraction techniques utilized in our framework will be presented in Section 3.

2.2 Natural Language Processing

NLP plays a crucial role in various industries and has a wide range of applications, ranging from real-time translation and social media search engines to sentiment analysis [6]. In this section, we will discuss the NLP facets that are integrated into NSPG.

2.2.1 *BERT Model.*

Most state-of-the-art natural language models are built on transformer architectures, such as Bidirectional Encoder Representations from Transformers (BERT), which are effective at modeling long-range dependencies in text [7]. These models utilize a multi-layer, multi-head self-attention mechanism, and contextual embeddings which allow for efficient parallel computation on GPUs.

2.2.2 *Data Augmentation.*

Data augmentation (DA) is a method of enhancing the diversity of training data without collecting more data. It involves adding modified copies of existing data or creating synthetic data to act as a regularizer and reducing overfitting during the training of machine learning models.

3 NSPG FRAMEWORK

In this section, we will demonstrate the details of each process flow in the proposed NSPG framework. The first step involves comprehending the context of the hardware domain from the hardware design documentation and generating the HS-specific BERT model. Next, we alter them using data augmentation and hardware domain-specific modification to create an enhanced dataset for fine-tuning the pre-trained BERT model. Each extracted sentence will be further processed to generate security property and verification assertion. We compare the performance of various modified datasets used to fine-tune the classification model and select the best one for security property extraction.

3.1 Hardware Documentation Dataset

The sentences used for training and fine-tuning are extracted from various documentation and paragraphs similar to the example shown in Listing 2. **Since this is the first work that utilizes design documentation for HS, we choose our data samples from the open-source documentation.** We create three datasets, as follows: (1) 15583 sentences from OpenTitan, RISC-V, OpenRISC, MIPS, and OpenSPARC documentation are used for pre-training the BERT model. This dataset will be called $\mathcal{D}_{pre}$. (2) To fine-tune the classification model, we manually label training samples consisting of 4427 sentences (out of which 2191 are security property-related sentences) from the top module and test cases documentation in OpenTitan design, MIPS, OpenSPARC as well as non-privileged documentation for RISC-V. This dataset will be called $\mathcal{D}_{cls}$. (3) In order to simulate the scenario of processing unseen documentation, we also manually labeled 470 security properties and non-properties from OpenTitan AES and ADC documentation, 78 sentences from OpenRISC, and 160 sentences from RISC-V trace documentation. All of these sentences are not included in $\mathcal{D}_{pre}$, and this dataset will be called $\mathcal{D}_{val}$. We spent approximately 24 engineer-hours meticulously verifying all 5135 sentences used in the $\mathcal{D}_{cls}$ and $\mathcal{D}_{val}$ datasets. Please note that manual effort is spent on the creation of the training dataset, which is crucial for the proper training of any machine learning model. Once this step is accomplished, the rest of the procedure can be automated.

3.2 Comprehending the Hardware Domain

First, the language model needs to recognize the contextual differences between security properties and regular sentences. Training the BERT model with its corresponding mask language model (MLM) could augment the performance of the model by introducing

Table 1: The performances of HS-BERT models with the original sentence from $\mathcal{D}_{pre}$ and after the random swap, random deletion, synonym replacement, and random insertion.

Dataset	Training Samples	Runtime	Perplexity
Original Documents	12472	57 mins	6.018
Original & RS	24065	109 mins	5.018
Original & RD	24946	114 mins	5.046
Original & SR	24208	110 mins	5.029
Original & RI	24931	113 mins	4.795

contextual embeddings of the specific domain, HS. Since we have limited data samples for hardware domain-specific documentation (compared to 1.14M papers from Semantic Scholar to train Sci-Bert), data augmentation is needed to improve the dataset [1].

3.2.1 *Data Augmentation for Hardware Documentation.*

DA is a key component for enhancing the quantity of in-domain data samples, which involves generating two correlated views of a data point in order to increase the amount and diversity of training data. Most of the security properties involve the behaviors of two or more entities in the design. Therefore, the context of operational relations and hardware terminology needs to be preserved in fine-tuning MLM. We need to avoid the essential information of the operating entities and only apply DA to swap or replace the rest of the tokens in the sentence. We analyze four popular DA techniques:
Random Swap (RS): We exchange the positions of two randomly selected phrases such as nouns and conjunctions, while maintaining the operation behavior and the content of the original sentence.
Random Deletion (RD): To keep the essential context, we only delete one of the adjectives, determiners, or adverbs available in the sentence, which does not impact the operation descriptions.
Synonym Replacement (SR): In order to preserve the hardware components described in the sentence, we only replace verbs with their closest synonyms obtained from the WordNet database [10].
Random Insertion (RI): For this operation, we first summarize all the adverbs that are used in the documentation, and randomly select one to insert into the sentence near verbs. To avoid altering the context, we randomly choose the most common adverb.

Each method introduces diversity in the original sentence and increases possible word usage, thereby reducing data overfitting and boosting the generalization ability of the model. We train four BERT models using different augmented data (in combination with the original data) and evaluate their performance.

3.2.2 *Pre-training BERT with MLM.*

The purpose of MLM is to understand the detail of each sentence demonstrated in the document and predict the masked content. Hence, we will select the model having the lowest *perplexity*, where a lower score indicates that the model has a better comprehension of the hardware domain and prediction of the masked word in the in-domain sentences [11]. We split the samples from $\mathcal{D}_{pre}$ into training and validation datasets with a ratio of 80% and 20%. Each model is trained with samples from the OpenTitan, RISC-V, OpenRISC, MIPS, and OpenSPARC documentation, which consist of 12473 sentences and additional data from the DA task. Table 1 shows the runtime, and perplexity for each DA. The DA task significantly improves the performance of the BERT model comprehension (lower perplexity) with scalable runtime increase. Random insertion operation performs the best among all four approaches with a perplexity score of 4.795. Therefore, we use the BERT model pre-trained with data from RI for further processes. This BERT model will be called Hardware Security-specific BERT (**HS-BERT**).

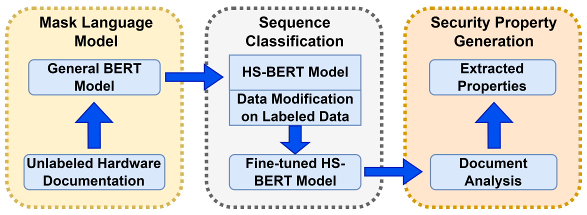

Figure 2: HS-BERT model training process.

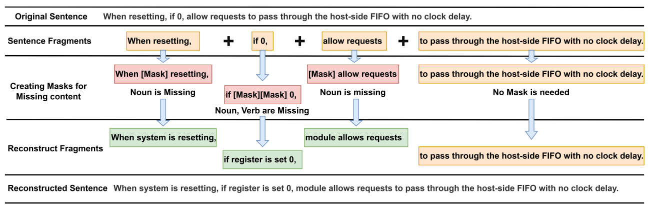
Original Sentence
When resetting, if 0, allow requests to pass through the host-side FIFO with no clock delay.
Sentence Fragments
When resetting,
+
if 0,
+
allow requests
+
to pass through the host-side FIFO with no clock delay.
Creating Masks for Missing content
When [Mask] resetting,
Noun is Missing
if [Mask][Mask] 0,
Noun, Verb are Missing
[Mask] allow requests
Noun is missing
to pass through the host-side FIFO with no clock delay.
No Mask is needed
Reconstruct Fragments
When system is resetting,
if register is set 0,
module allows requests
to pass through the host-side FIFO with no clock delay.
Reconstructed Sentence
When system is resetting, if register is set 0, module allows requests to pass through the host-side FIFO with no clock delay.

Figure 3: Sentence modification in documents.

3.3 Security Property Classification

In this section, we will explain the process of security property classification from the design documentation utilizing the Sequence Classification Model (SCM). Figure 2 illustrates the training procedure of NSPG framework, which consists of three stages. In the first stage, we obtain the HS-BERT model with sentences from hardware design documentation. In the second stage, we will modify the labeled $\mathcal{D}_{cls}$ with the HS-BERT model to fine-tune SCM. In the third stage, the HS-BERT model will be compared against other state-of-the-art BERT models. Finally, we will validate each fine-tuned SCM with $\mathcal{D}_{val}$ and utilize the best-performing one to generate security properties from the documents. In this work, we employ the SciBERT and general BERT models and compare their performances against HS-BERT in Section 3.3.2.

3.3.1 *Data Modification for Property Classification.*

Data modification techniques are utilized to generate unbiased data, which in turn is critical to furnish reliable machine learning algorithms. In order to improve the performance of sequence classification, we propose a hardware domain-specific modification, **Fragment Insertion**, to further differentiate the features of each sentence in the documents, which enforces the BERT model to recognize more contextual dependencies. For instance, sentences including hardware operations and behavior are considered in-domain, while common descriptions are regarded as out-of-domain. Fragment insertion adds more in-domain context to the original sentence, which helps the SCM identify the operation context. Intuitively, the prevailing assumption is that using more in-domain text in the training datasets should help with domain-specific classification. Although each sentence in the document is constructed differently, we will explore this concept by adding domain-specific portions for each sentence. Figure 3 shows an example of modifying a sentence in the document with fragment insertion. First, we break the sentence "When resetting, if 0, allow requests to pass through the host-side FIFO with no clock delay." into fragments by identifying

Table 2: Performance comparison between general BERT, SciBERT and HS-BERT with data modifications from $\mathcal{D}_{cls}$.

	General BERT		SciBERT		HS-BERT	
	Accuracy	Recall	Accuracy	Recall	Accuracy	Recall
Baseline	83.1%	81%	83.2%	98%	84%	97%
MT	83.2%	93.5%	82.2%	96%	85.1%	97.3%
MOT	86%	96.4%	88.1%	98%	90.1%	98.3%
MTT	81.2%	83.4%	87.1%	95%	88.1%	97.5%
MOTMT	83.2%	70.6%	87.5%	97%	88.5%	98.1%

Table 3: Sequence Classification for $\mathcal{D}_{val}$. Each row represents the accuracy, recall, and F1-score for different HS-BERT models and different modified labeled datasets.

	OpenTitan		RISCV		Openrisc	
	Accuracy	Recall	Accuracy	Recall	Accuracy	Recall
Base-HS-BERT	70.6%	72.8%	71.9%	71.2%	80.3%	80.2%
MT-HS-BERT	74.5%	80.4%	75.8%	79.2%	83.6%	82.6%
MOT-HS-BERT	81.5%	93%	79.1%	90.1%	88.3%	87%
MTT-HS-BERT	75%	85%	74.3%	83.5%	86.3%	82.6%
MOTMT-HS-BERT	76%	84.1%	73.3%	80.5%	87%	84.7%

each conjunction "When", "if", and "to" in the sentence, and separating the sentence into fragments as: "When resetting", "if 0", "allow requests", and "to pass through the host-side FIFO with no clock delay". In these fragments, "When resetting", and "allow requests" are missing nouns, "if 0" is missing noun and verb, and "to pass through the host-side FIFO with no clock delay" is complete. We will add these missing components into the fragments by adding a [Mask] token into the fragment and applying the pre-trained HS-BERT to place appropriate in-domain terms for each [Mask] token. Hence, "When resetting", "if 0", and "allow requests" will be transformed into "When system is resetting", "if value is set 0", and "module allows requests".

3.3.2 General BERT vs SciBERT vs HS-BERT. General BERT utilizes WordPiece for unsupervised tokenization of input sequences, building its vocabulary with the most frequently used words or subword units. SciBERT, on the other hand, is constructed with a new WordPiece vocabulary on a scientific corpus using the SentencePiece1 library [14]. The token overlap between BERT and SciBERT vocabulary is 42%, indicating a substantial difference in the frequently used words between scientific and general domain texts. SciBERT was trained on a dataset comprising 1.14M papers from Semantic Scholar, where 82% of the papers belong to the biomedical domain, while the remaining 18% pertain to computer science [1]. Table 2 shows the comparison between the general BERT model, SciBERT, and the proposed HS-BERT, training the same data modification method and the same 3927 samples from $\mathcal{D}_{cls}$. Another 500 samples from $\mathcal{D}_{cls}$ are used for validation. Modifications are applied to both in-domain and out-of-domain sentences. The baseline refers to no modification on both training and testing data. We have considered the following cases for modification: 1) **MT** refers to only modifying training data, 2) **MTT** refers to modifying training and testing data. Note that the modification only changes the content of testing sentences and does not increase the size of test samples, 3) **MOT** refers to both modified and original training data, 4) **MOTMT** refers to the original training data, its modified counterpart, and modified testing data. The results indicate that all three BERT models demonstrate improved performance in comparison to their baseline models, with an average increase of 0.6% accuracy with General BERT, 3.1% accuracy with SciBERT, and 3.95% accuracy with HS-BERT. However, HS-BERT, with fine-tuning, outperforms the state of art General BERT and SciBERT models on accuracy and recall, while performing similarly on precision and F1-score. Based on these observations, we determine that HS-BERT is more suitable for extracting potential security properties, which we have subsequently incorporated in NSPG.

3.3.3 Fine-tuning SCM model. In this stage, NSPG will apply fine-tuned HS-BERT SCM to identify the security properties in the SoC documentation. In order to emulate this scenario in which the SCM is applied on an unseen document to determine whether the sentence is a security property or not, we will test each trained sequence classification model on $\mathcal{D}_{val}$. Table 3 shows the results for HS-BERT performance with no modification and the data modification approach, as described in Section 3.3.2. Base-HS-BERT refers to the SCM performance fine-tuned with no data modification. The HS-BERT model trained with the MOT modification performs the best with an average of 82% accuracy, and 90% recall score. Since this method consists of both the original and the modified sentences, it improves the diversity of the training dataset, which helps the model to learn more features and context of a property-related sentence. Therefore, we will use MOT modification as our final data modification approach for the SCM model.

3.3.4 Extracted Sentence Examination. To generate security properties that can be effectively transformed into verification assertions, we employ a process to verify the integrity of each extracted sentence. As mentioned in Section 2.1, a complete security property should consistently take the form of a sentence or conjunction of sentences that constitute a property-related sequence with a register name (noun), an operation (verb), and a value (number or noun). This structure allows for the establishment of a clear relationship between registers, facilitating the construction of verification assets, such as hardware assertions. Typically, a sentence may contain one or more conjunctions separated by words such as ",", "when" or "if" Subsequently, each conjunction is scrutinized to confirm whether it adheres to the format of a property-related sequence. To enhance the completeness of extracted sentences related to security properties, we have developed a methodology for enriching their content. For example, when we encounter a sentence like "Once firmware initialization is complete, it is important to exit boot request mode if the endpoints ever need FIPS-approved random values," we identify that "boot request mode" refers to "Boot-time Request Mode" (BOOT_REQ_MODE), as specified in the document. To achieve this, we extract this information from the register checklist within the document and replace "boot request mode" with the corresponding register information, resulting in a finalized security property like "Once firmware initialization is complete, it is important to exit BOOT_REQ_MODE if the endpoints ever need FIPS-approved random values." This comprehensive approach ensures that the security properties encompass both the relevant registers and the necessary operational details.

Table 4: Number of processed sentences, security properties generated, and properties covered and not covered by design verification (DV) documentation.

Hardware IP	Sentences	Extracted	Properties	Covered by DV	Not covered by DV
Key Manager	241	51	47	40	7
LC Controller	375	79	76	65	11
HMAC	170	28	27	19	8
KMAC	367	84	74	60	14
OTP Controller	570	105	102	84	18
Total	1723	347	326	268	58

4 EXPERIMENTS

In this section, we will demonstrate the evaluation of our framework NSPG with five OpenTitan IP documents, and discuss its performance on security property extraction.

4.1 Experimental Setup

All of our experiments are run on a server consisting of 40 CPUs of 64-bit Intel(R) Xeon(R) E5-2698 v4 @ 2.20GHz. The entire SoC documentation comprises of 33 IP design specifications, with 10865 sentences. Each IP design documentation demonstrates information on register descriptions, functionalities, and operation processes, including the security features for various modules, that will be used for evaluating NSPG [12]. It is important to note that our current framework is designed to process documentation written in English. Before using NSPG, we extract all sentences and standardize them, removing additional spaces and non-English terms. Among these, the contents of the operational processes or behaviors can be transformed into security properties, while the others are treated as non-properties. Moreover, we utilize these security properties to search for potential violations in relevant hardware IPs. Since the list of registers in SoC design is available to us, we craft the detailed security properties from the IP security specification and checklist of SoC registers.

4.2 NSPG Framework Evaluation

First, NSPG processes each text file consisting of sentences in the design documentation. It filters out any sentence having less than 10 words since they usually do not contain enough information about operation behaviors. The rest of the sentences will be parsed through the trained HS-BERT sequence classification model discussed in Section 3, and the extracted sentences from each IP document will be listed in property text files. As shown in Table 4, 1723 sentences are processed, and 344 sentences are extracted as potential security properties. Overall, 326 sentences (94% of the 347 extracted sentences) can be utilized to generate security properties for design validation. We compare the generated security properties with the test cases listed in the DV documentation, which describes all the test cases and IP operations needed to be checked by the SoC designer. We spend roughly 3 engineer-hours verifying the extracted properties. **While 268 of our generated security properties are covered in DV, 58 properties are not covered in the test cases, which clearly demonstrates that NSPG is adept at accounting for specifications that are not covered in DV.** This shows that NSPG is able to efficiently identify security properties in the documents. We use these newly generated security properties to verify the bugs.

4.3 Effectiveness in Discovering Violations

The extracted security properties provide us with essential information to construct various constraints when generating test cases for vulnerable IP designs. We choose to transfer the properties into SystemVerilog assertion format. If the security property has complete integrity, it can be transformed seamlessly into hardware assertion [9]. We demonstrate the capability of our framework, using assertion-based security verification. By creating constraints based on the acquired security properties and generating test cases on the design, we are able to detect eight vulnerabilities in the buggy IP designs.

A. Key Manager: The key manager implements the root key operation for the system and allows it to protect critical assets from malicious software. Two vulnerabilities are found in this IP: (1) The security property requires the key manager to wipe internal storage when it is in an invalid state. However, the implementation reverses the operation and wipes the key under a valid state. (2) It is required to continuously wipe the secret key with entropy during the operation state. However, the implementation does not replace the key registers, leaving sensitive information vulnerable. These vulnerabilities could potentially impact the confidentiality of the secret key, allowing the attacker to reveal the key information.

B. LC Controller: The life cycle (LC) controller controls the peripheral interactions on the chip interconnect bus. The security property requires the signal *fatal_bus_integ_error_q* to be set to one, when any bus integrity fault is detected. However, the boundary-scan test controller (JTAG) Test Access Port (TAP) does not provide a bus integrity check signal, which causes integrity check failure in the life cycle controller and unexpected behavior in the system.

C. HMAC: The HMAC module implements a SHA-256 hash-based authentication module to ensure the integrity of any incoming message and its encryption code from the secret key. Three security bugs are found in this IP: (1) The first bug occurs when the software wants to convert the message byte order. It is required to set *CFG.endian_swap* register to one. However, the implementation reverses the operation, making the IP convert the message, when *CFG.endian_swap* is zero. This will cause unexpected operations due to incorrect instructions for converting messages. (2) The second bug occurs when the SHA engine is disabled. Although it is required to clear the digest in HMAC, the design does not satisfy this property. (3) The third bug involves key randomization when the CPU writes values to the secret key. The key is required to be wiped with randomly generated value; however, this property is not satisfied in the RTL.

D. KMAC: The KMAC module is a Keccak-based message authentication code generator used to verify incoming messages. It utilizes masked permutations to prevent side-channel attacks. The security property requires the software to provide the key in masked form, when the *EnMasking* parameter is not set and the *SwKeyMasked* parameter is set. However, this mechanism is not correctly implemented, leaving the software with an unmasked key. This causes key leakage through an unprivileged software adversary.

E. OTP Memory Controller: The OTP memory controller is a module that provides a device with a one-time programming functionality. The security property only allows the IP to respond and write into the readout register when the lock control is not active.

However, the IP is implemented with a mechanism that allows it to bypass the read-and-write lock control signal every four clock cycles. This will cause unpredictable behavior of the controller and allow the software adversary to attack the integrity of the module.

In summary, we have discovered eight bugs in five hardware IP designs from the 326 security properties we generated using NSPG. These vulnerabilities may cause information leakage, unexpected behavior, and unprivileged accesses in these IPs. It proves that these extracted security properties can provide valuable information to generate constraints for the hardware verification process.

5 DISCUSSION

A. False Positives (FPs) and False Negatives (FNs): The FPs in our evaluation comprise of short sentences that furnish equations or incomplete register interactions. In contrast, FNs usually involve only the structure of the sentences It is crucial to recognize that not all extracted security properties can be automatically converted into assertions and seamlessly integrated into target modules and there is potential for further improving the performance. However, the framework's primary aim is not to completely eliminate manual inspection or achieve complete coverage. Instead, NSPG aims to reduce manual effort in the verification process .

B. Comparison to text classification models: We also explore standard text classification models such as TF-IDF and Bag-of-Words trained with $\mathcal{D}_{cls}$ and tested on $\mathcal{D}_{val}$. TF-IDF achieved 70% accuracy for OpenTitan, and 61% accuracy for RISC-V, Bag-of-Words achieved 51% accuracy for OpenTitan, and 53.12% for RISC-V. Hence, we can conclude that HS-BERT is significantly more effective than standard text classifiers.

C. Comparison to ChatGPT: We compare NSPG against ChatGPT, a popular chatbot based on the OpenAI Generative Pre-trained Transformer (GPT)-3.5 language model that is adept at generating human-like text responses to any input in a conversational context [13]. We trained ChatGPT by consistently labeling sentences as "security properties" or "non-security properties" and added sentences from the training dataset as prompts. Their performances are evaluated on a reduced dataset of 50 sentences, which contained 25 property-related and 25 non-property-related sentences (the dataset size was necessitated by resource limitations since GPT-3.5 is not publicly available as an open-source model). ChatGPT's evaluation resulted in numerous false positives and false negatives, with only 35 sentences correctly classified, many of which were unsuitable for SoC security verification. The accuracy, recall and F1-score obtained by the ChatGPT model were 68%, 88% and 73%, respectively. On the contrary, NSPG outperforms ChatGPT by identifying all 25 property-related sentences, thereby furnishing an accuracy, recall, and F-1 score of 100%, respectively.

6 RELATED WORK

A recent approach, *Isadora*, combines information flow traces with security specifications, but its reliance on conclusive testbenches and simulation traces may limit scalability for complex SoCs [5]. Conversely, PCHIP introduces theorem generation functions, aiding in the development of data secrecy properties independently of information flow traces [2]. However, it is only limited to cryptographic circuits.*SCIFinder* generates cecurity-critical invariants for DV but is limited in studying security properties and lacks scalability for complex processors [15]. In comparison, NSPG utilizes a BERT model to automatically identify and generate new security property-related sentences.

7 CONCLUSION

This paper presents NSPG, the first NLP-based automated HS property generation method, that utilizes SoC documentation to extract security properties. NSPG includes a novel hardware security-specific language model (HS-BERT) and a data modification technique to improve automated security property generation. NSPG shows promise in addressing a significant portion of the specifications delineated in the documentation and can identify the ones that have been extracted. Consequently, this reduces the manual effort substantially, enabling a concentrated focus on covering the remaining specifications. NSPG is evaluated on OpenTitan SoC documentation, resulting in 326 correctly extracted security properties from 1723 sentences for five hardware IPs. Furthermore, these security properties help discover eight vulnerabilities in a buggy SoC, which proves the effectiveness of the generated security properties. Additionally, our evaluations prove that NSPG furnishes better performance than ChatGPT for SoC security property generation. With the advent of LLMs, we envision that NSPG will lay the foundation for utilizing NLP approaches in SoC verification.

8 ACKNOWLEDGMENT

This research is partially supported by Technology Innovation Institute (TII), Abu Dhabi, UAE.

REFERENCES

[1] Iz Beltagy et al. 2019. SciBERT: A pretrained language model for scientific text. *arXiv preprint arXiv:1903.10676* (2019).

[2] Mohammad-Mahdi Bidmeshki et al. 2017. Data secrecy protection through information flow tracking in proof-carrying hardware IP—Part II: Framework automation. *IEEE TIFS* (2017).

[3] CWE. [n. d.]. CWE - CWE-1194: Hardware Design (4.0). https://cwe.mitre.org/data/definitions/1194.html. (Accessed on 05/15/2020).

[4] Ghada Dessouky et al. 2019. {HardFails}: Insights into {Software-Exploitable} Hardware Bugs. In *28th USENIX Security*. 213–230.

[5] Calvin Deutschbein et al. 2021. Isadora: Automated information flow property generation for hardware designs. In *Proceedings of the 5th ASHES Workshop*. 5–15.

[6] Steven Y Feng et al. 2021. A survey of data augmentation approaches for NLP. *arXiv preprint arXiv:2105.03075* (2021).

[7] Anthony Gillioz et al. 2020. Overview of the Transformer-based Models for NLP Tasks. In *15th FedCSIS*. IEEE.

[8] HACK@DAC22. [n. d.]. HACK@DAC22 – Hack@EVENT HW CTF. https://hackatevent.org/hackdac22/.

[9] Rahul Kande et al. 2023. LLM-assisted Generation of Hardware Assertions. *arXiv preprint arXiv:2306.14027* (2023).

[10] George A Miller. 1995. WordNet: a lexical database for English. *Commun. ACM* 38, 11 (1995), 39–41.

[11] Robert C Moore et al. 2010. Intelligent selection of language model training data. In *Proceedings of the ACL 2010 conference short papers*.

[12] OpenTitan. [n. d.]. OpenTitan | OpenTitan Documentation. https://docs.opentitan.org/.

[13] John Schulman et al. 2022. ChatGPT: Optimizing language models for dialogue. *OpenAI blog* (2022).

[14] Rico Sennrich et al. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Berlin, Germany, 1715–1725. https://doi.org/10.18653/v1/P16-1162

[15] Rui Zhang et al. 2017. Identifying security critical properties for the dynamic verification of a processor. *ACM SIGARCH Computer Architecture News* 45, 1 (2017), 541–554.