*electronics*

*Article*

# Artificial Intelligence Application in the Field of Functional Verification

**Diana Dranga * and Catalin Dumitrescu ***

Department Electronics and Telecommunication, Faculty of Electronics, Telecommunications and Information Technology, National University of Science and Technology POLITEHNICA from Bucharest, 060042 Bucharest, Romania
* Correspondence: diana.dranga@stud.etti.upb.ro (D.D.); catalin.dumitrescu@upb.ro (C.D.)

**Abstract:** The rising interest in Artificial Intelligence and the increasing time invested in functional verification processes are driving the demand for AI solutions in this field. Functional verification is the process of verifying that the Register Transfer Layer (RTL) implementation behaves according to the specifications provided. This is performed using a hardware verification language (HVL) such as SystemVerilog combined with the Universal Verification Methodology (UVM). Reading, identifying the key elements from multiple documentations, creating the verification plan, building the verification environment, implementing the tests defined, and achieving 100% coverage are usually the steps performed in order to complete the verification process. The verification process is considered finalized when functional coverage is at 100%. There are multiple ideas on how the process can be aided by AI, such as underlining the essential information from documentation, which would help in understanding faster how the Register Transfer Layer implementation works, thus vastly reducing time. In this paper, to greatly reduce the time spent on functional verification, two Convolutional Neural Network (CNN) architectures are implemented to properly classify the information across different documents; both approaches have significant and promising results. The database used for this classification task was created by the researchers using different documentations available.

**Keywords:** artificial intelligence; functional verification; UVM

check for updates

**Citation:** Dranga, D.; Dumitrescu, C. Artificial Intelligence Application in the Field of Functional Verification. *Electronics* **2024**, *13*, 2361. https://doi.org/10.3390/electronics13122361

Academic Editors: Quan Qian and Xing Wu

Received: 12 May 2024
Revised: 4 June 2024
Accepted: 13 June 2024
Published: 17 June 2024



## 1. Introduction

As Integrated Circuit development progresses in complexity and scalability, it becomes essential for them to work as expected, being reliable and robust, especially in critical files such as medical, aviation, and automotive files.

In the broad process of developing a System-on-Chip (SoC), several steps are required. These procedures include the definition and architecting of the system's specification and requirements, implementing it using a Hardware Description Language (HDL) such as Verilog IEEE standard 1800-2023 or VHDL, and conducting physical design and various other laborious processes presented in Figure 1.

As design advances in development, it must be integrated into the verification environment for bug testing and feature verification. Verification activities continue even after tape out in silicon in most cases. Multiple stages are defined when verifying an SoC, such as the following:

- Unit-level verification or block-level verification. In this step, each block of the SoC is verified separately by an engineer or a team of engineers, depending on its complexity. For this block alone, a verification environment is either implemented from scratch or imported from the previous project iterations; new tests are defined, implemented, and debugged. Regressions are run and the coverage percentage is considered when closing the verification of the block. This means coverage must be at 100% for the block to be considered finalized and verified. The verification environment and tests are written in SystemVerilog alongside UVM.

- Cluster-level verification, where a larger unit is verified alongside DPI-C. In a similar manner, there is a verification environment, tests, and coverage. Usually, in this strategy, the CPU is commonly not used.
- System-level verification, where a part of a system is verified, usually including the CPU, DMA, SPI, Debugger, etc. Usually, new tests are implemented in C, assembly, or SystemVerilog, depending on what the testbench supports. Verification is considered done when all the defined tests are passing and where it is critical that some coverage is implemented and is at 100%.
- SoC-level verification, where the engineer can use any block from the entire chip, whether it is analog or digital. Here also, verification is complete when all the tests are passed. This is one of the most time-expensive strategies since simulations usually take hours instead of minutes, as in the previous strategies.

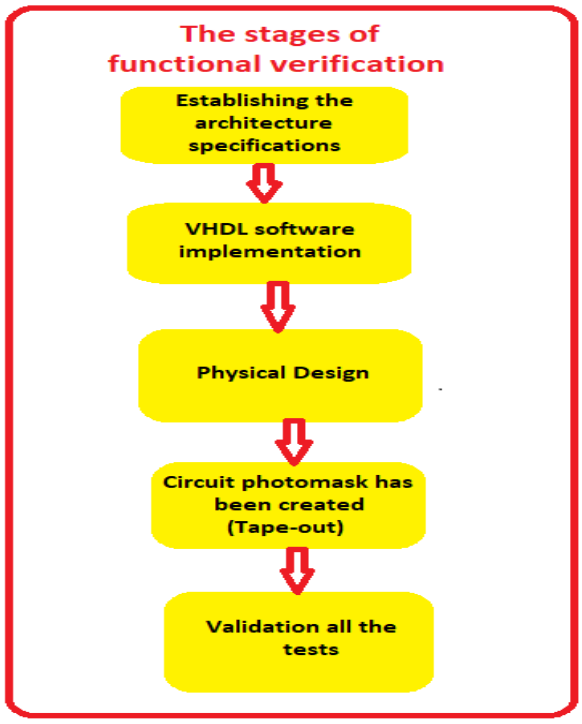


**Figure 1.** System-on-Chip development procedures, and VHDL Verilog IEEE standard 1800-2023 implementation software.

For an SoC, all the stages presented above must be given thorough attention. For a better understanding, usually the verification environment commonly used is sketched in Figure 1 [1].

The presented method in Figure 1 is of high complexity and is time-consuming. One of the key aspects of this approach involves executing, debugging, and rectifying tests as well as achieving a high coverage percentage. Coverage is a measure for ensuring that the defined scenarios and configurations of the design have been thoroughly verified in the environment. The design and the verification environment itself are simulated using software simulators such as Cadence INCISIV, Mentor QuestaSim, and Synopsys VCS simulation modules from Verilog IEEE standard 1800-2023. Regardless of the advances of the aforementioned simulators, such as increased, optimized simulation speed and advanced debugging features, they still fall short of accelerating the verification process. Here, Artificial Intelligence might be able to play a crucial role in decreasing the time spent on the verification process.

Figure 2 presents how much time is spent on the verification process and on designing RTL. It can be clearly seen that over the years, through the addition of more complex

designs, the verification process has been swiftly becoming the activity on which most of the time is spent.

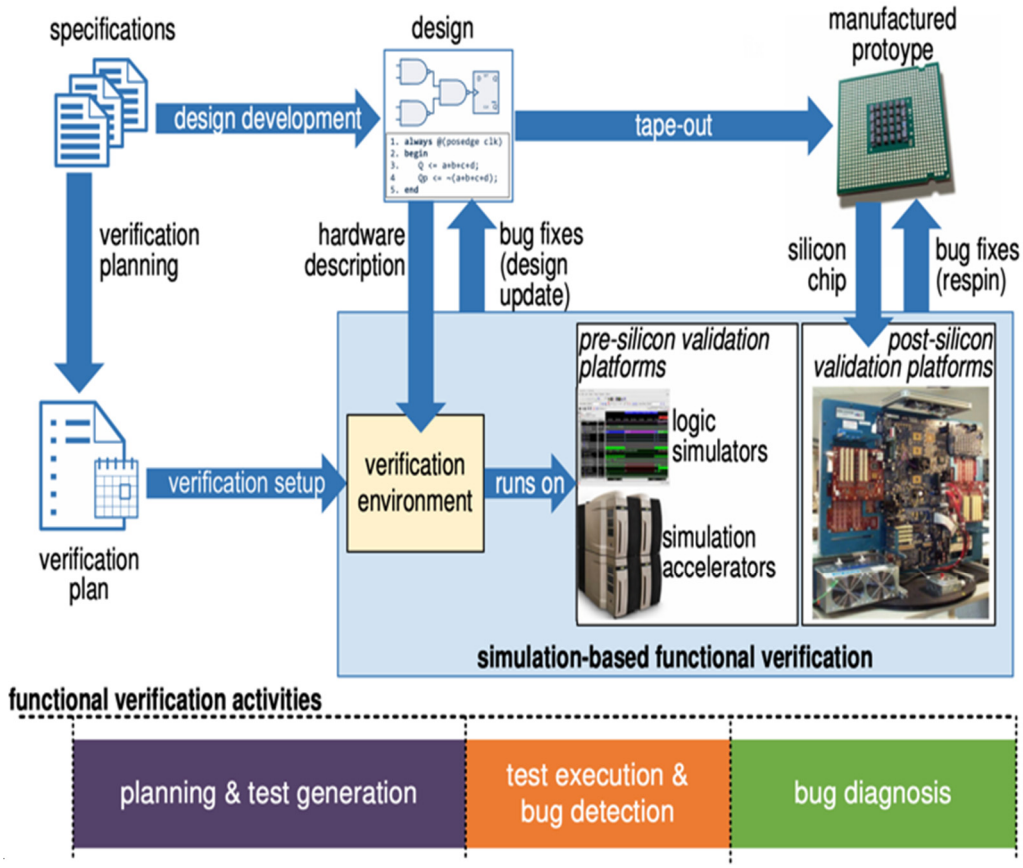


**Figure 2.** Verification process steps [1].

A verification engineer's time is usually spent between test planning, testbench development, implementing, and debugging the tests defined. The largest amount of time, which is 47%, is spent on test planning, which would mean rigorously understanding the module documentation and the protocols used by the module or the system. It is a highly time-consuming task, as more clarifications are usually needed. Verification plan reviews in which multiple designers or verification engineers take part to see if the main scenarios for the respective module are defined and clearly explained for each step. Figure 3 illustrates in detail where the verification engineer's time is focused, which is mostly on test planning (47%) and debugging (21%). Test planning takes a lot of time for the verification engineer as the engineer must thoroughly understand the specifications, protocols, and various other information needed for the test scenarios to be defined. More clarifications are usually needed, the documents are updated with improved features, or sometimes they are reworked from scratch. Regarding debugging time, usually after writing the test in either Assembly, C, or System Verilog, the verification engineer uses a simulator (either Cadence INCISIV, Mentor QuestaSim, or Synopsys VCS) to run the test alongside the environment and Register Transfer Layer (RTL). In most cases, the test fails due to various reasons, either bugs in the testbench, test, or RTL. Depending on the complexity of each test case, some may take a couple of minutes, but some may even take days to complete. The engineer will have to take on a separate task in debugging; thus, the time spent increases yet again.

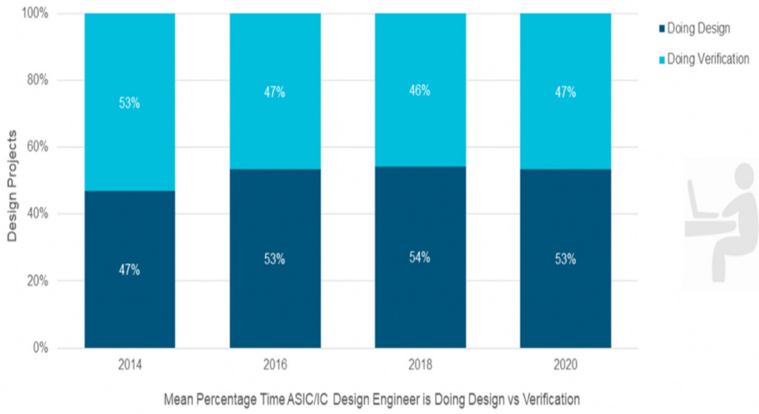


**Figure 3.** Mean percentage of time spent in ASIC Design and Verification [2].

By using Artificial Intelligence, the verification process can be faster and completed in less time. For instance, Artificial Intelligence can be used on input stimulus, which is sent to the RTL from the testbench side with the goal to increase the coverage, on test regression results where on each failure reason the test is assigned into a cluster and on documentation classification. Regarding documentation classification, AI can be used to identify and classify important, complex, and relevant features of the design mentioned in the documentations. Features that were removed in the respective chip iteration and deprecated and irrelevant data would be disregarded.

## 2. Related Work

There are a multitude of areas where Artificial Intelligence can aid the verification process.

The tests are grouped according to the failure reasons. As stated in [3], some machine learning algorithms can effectively classify the failing tests in a regression according to their failure reasons. Although, according to [4], the outcome is not as expected.

Stimulus and test generation is conducted by using supervised and reinforcement ML algorithms [5] to hit the planned coverage for the Device under Test (DUT) [6]. In the verification process of a cache controller, a supervised Deep Neural Network (DNN) [7] was used alongside the Q learning algorithm. The DNN is trained to create sequences for four First-in-First-Out (FIFO)s. This method has led to an increase of approximately 55% percent in triggering cache victims as opposed to the random–constrained generated stimuli approach. In [8], features of the "e" HVL are in line with verification methodology [9] to improve a DNN model to learn from the coverage results of a processor. Coverage is read dynamically through the Specman API, and the environment can stop at any time due to the phase approach of the UVM. This has shown a 33% reduction in time spent on simulation. In [10], a supervised ANN is used to generate the stimulus and open source Cocotb as a platform for the testbench, thus improving the number of iterations required to achieve the coverage desired. A limitation of this is that it only works for linear Device Under Test (DUT), not for highly complex modules. The study is very important to show how the solution might be able to scale further.

Documentation classification, for the verification engineer to use, focuses on the relevant parts of the design. In the industry, there are vastly complicated standards such as RDMA, USB, etc., which require an intrinsic understanding of the protocol itself and the module. As in a complex design, there are multiple specifications, and the engineer is required to read and understand every one of them. Such a task is time-consuming as some of the features presented in the specifications sometimes do not apply to the design due to other constraints. This approach might significantly impact the time spent on verification with a focus on the test planning on scenarios. Time spent would be vastly diminished as the engineer would be able to read and make use of the documentations in the most efficient way. There is some work related to text classification, which follows two steps: extracting some feature from the text and then feeding them into a classifier.

Some popular feature representations are Bag of Words (Bag of Words), Term Frequency-Inverse Document Frequency (TF-IDF), etc., and the most used classifiers are Naïve Bayes, k-Nearest Neighbors (kNN), Support Vector Machine (SVM), etc. Deep learning models have also been used to improve language modeling to handle better contextual information, such as in [11–13]. There are various network architectures used, such as Hybrid Deep Learning Network, which constitutes a combination of the Convolutional Neural Network, Long Short-Term Memory (LSTM), and fully connected layer [14]. In text mining and NLP domains, the CNN has demonstrated its capability to solve the issue of text or natural language data. When compared to the Recurrent Neural Network (RNN) side by side to classify medical documents, the CNN can outdo the LSTM. In some studies, the CNN is also combined with the RNN, LSTM, or BiLSTM [15–17]. In [18], a review of different paraphrase identification methods has been conducted; their strengths and disadvantages are discussed. CNNs are a widely popular choice when it comes to paraphrasing tasks because of their ability to record local information and carry out parallelism. Although CNNs may need other sets of modules to deal with remote dependencies in sentences [18]. Approaches such as [19,20] make use of the Time Delay Neural Networks (TDNN) to solve this issue. Feature interaction layers can boost overall CNN performance [21,22]. Regarding RNNs, which are used in paraphrasing because of their ability to handle variable length sentences [18], they are sometimes used with Gated Recurrent Units (GRU) to improve performance, while BiLSTM has the ability to capture word level and contextual information of sentences [18,23–25]. CNNs have also been used for classification or regression tasks in the industry of Electrical equipment by diagnosing faults collected by experimental data and the CRWU-bearing dataset [26,27], health monitoring on NASA milling dataset [28] and fault prediction in machine tools equipped with different sensors in a typical machining workshop in Wuxi, China [29].

At this moment, these are the main ideas by which Artificial Intelligence can vastly improve the verification process. Each of them promises major advantages when implemented in the environments, thus diminishing time and increasing productivity.

## 3. Methodologies, Study Materials, and Databases

In functional verification, rigorously analyzing and understanding the protocols, IPs, and various rules are essential to correctly implement the verification environment alongside the test scenarios. In an SoC, there are a vast number of documentations used. Thus, test development becomes a laborious task that consumes a large amount of time. By aiming to reduce time spent on documentation reading and understanding, the verification time can be tremendously lessened. Most commonly, specifications contain text written in natural English and waveform snippets, which usually refer to the explanations next to them. Figure 4 is a good example.

In Figure 4, waveforms, alongside the signal descriptions, are also presented to thoroughly understand the APB protocol. The driver class of the verification environment must adhere to these rules to correctly send or receive data from the DUT.

In this paper, the researchers implemented the dataset alongside the model. The dataset consists of important or critical information from documentation and non-crucial information. Different specifications and documentations were analyzed and combined to create the dataset. Such documentations include, for instance, ARM's Advanced Microcontroller Bus Architecture APB and AXI. The specifications have been thoroughly analyzed by the engineer and the foremost information and irrelevant data were categorized into relevant and irrelevant information. The structure consists of two folders containing .txt files, which contain a random number of sentences. The classification of critical and non-critical information was performed using the engineer's experience in the field and the general implementations of various projects. Commonly, information such as in Figure 5 is considered relevant in understanding how the protocol functions. In Figure 5, the protocol signals are described succinctly.

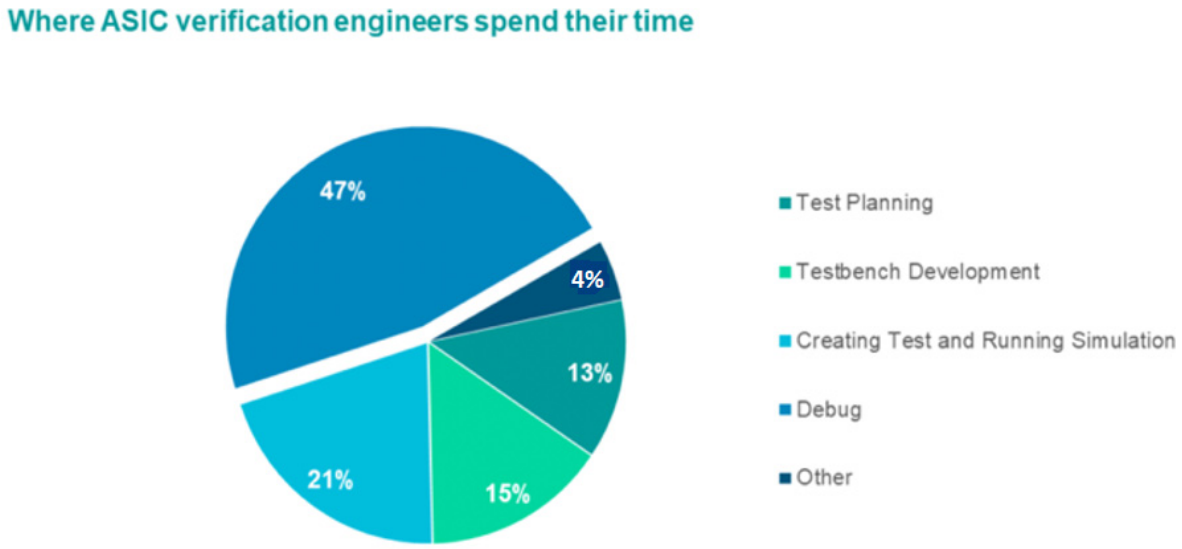


**Figure 4.** Time spent when performing functional verification [2].

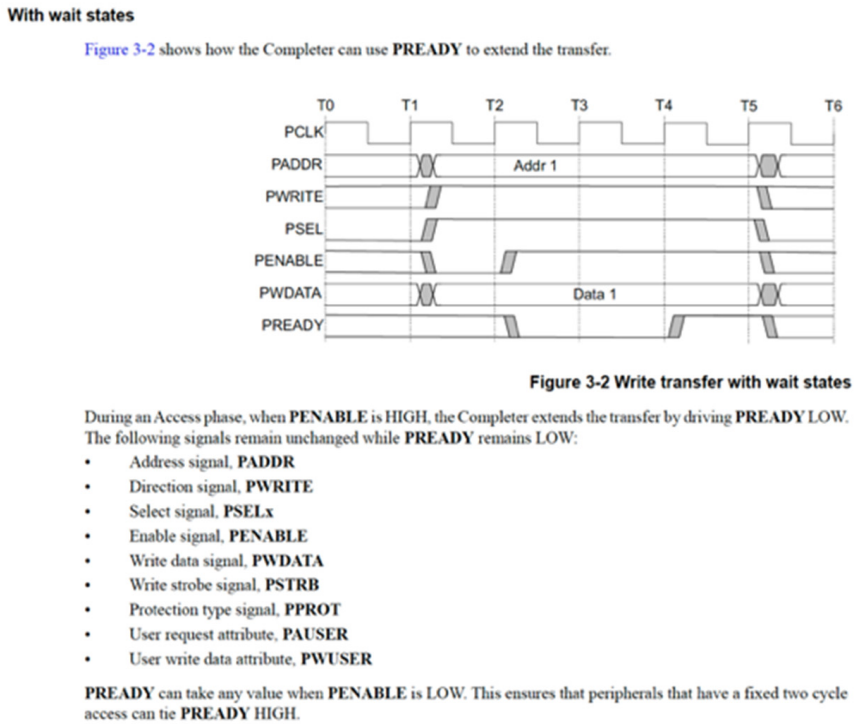


**Figure 5.** AMBA APB Write with no wait states transfer [30].

Figure 6 is an example of other relevant data that must be considered by the engineer. It contains a description of the PSLVERR, PREADY, and PENABLE signals. These are signals usually used in an APB transfer. The signal behavior will be implemented in the driver class using the Universal Verification Methodology (UVM). It is mandatory to understand and correctly implement the behavior of these signals to correctly drive data into the DUT. If it is not implemented correctly, more debug time will be added.

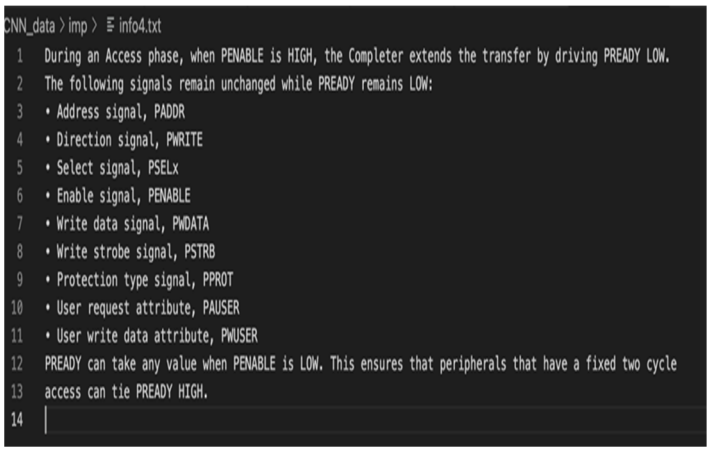


**Figure 6.** APB signal short description.

These are some examples of relevant data in the created database. As for irrelevant data, these can contain parts such as copyright rules that, to the engineer, are not applicable when implementing the verification environment, and the test scenarios and parts may not be implemented in the RTL design. Due to various reasons, some parts of different modules, protocols, etc., may not be implemented in the design. The causes can vary from reduced time for the project, instances when the final client does not need that feature, or instances when they are not pertinent to the SoC application and, thus, need to be discarded. Figures 7–9 illustrate an example of irrelevant information mentioned above.

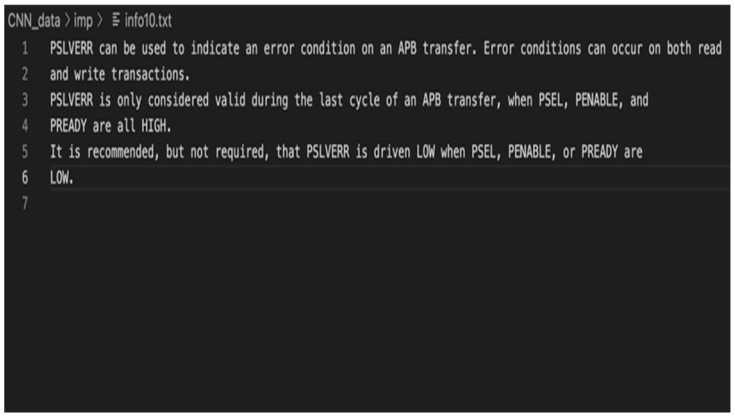


**Figure 7.** Succinct APB list of protocol rules.


**Figure 8.** AMBA AXI Copyright information.

```
CNN_data > non_imp > info44.txt
1  The signal conventions are:
2  • Signal level - The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW.
3  Asserted means:
4  - HIGH for active-HIGH signals.
5  - LOW for active-LOW signals.
6  • Lowercase n - At the start or end of a signal name denotes an active-LOW signal.
7  • Lowercase x - At the second letter of a signal name denotes a collective term for both Read and Write. For
8  example, AxCACHE refers to both the ARCACHE and AWCACHE signals.
9
```

**Figure 9.** AMBA AXI signal conventions.

Considering the means above, a database was constructed from scratch using the verification engineer's expertise and experience in the field. For this dataset, information was carefully extracted from the Advanced Microcontroller Bus Architecture Advanced Peripheral Bus (AMBA APB) and from the Advanced Microcontroller Bus Architecture Extensible Interface (AMBA AXI). After splitting the dataset into relevant and irrelevant information and double-checking for any missed or misused data, a starting dataset was constituted. This dataset will be used to train the model presented in this paper. For this implementation, a deep learning model was used: the Convolutional Neural Network (CNN). A Convolution Neural Network is a category of Artificial Neural Networks that has attracted a very large amount of interest from different researchers from various domains. CNNs are widely used for image recognition, text classification, etc. They can automatically learn a hierarchy of features that can be used for classification. To reach this objective, a hierarchy of feature maps is constructed through successive convolutions of the input with the learned filters [31]. Typically, a CNN has several layers: an input layer, several convolutional layers depending on the application, hidden layers, and an output layer (Figure 10).

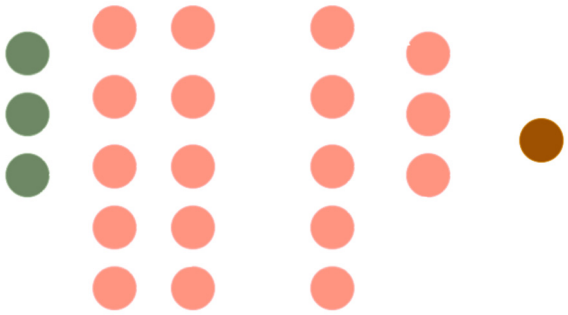

**Figure 10.** Typical CNN architecture.

## 4. Results

In this presented application, data preprocessing was performed before feeding it into the model. Data were initially presented in natural English in .txt files; the application parsed the folders and files and added labels on which was considered important or not. Data were augmented using special libraries to enhance the database rapidly. For this, the nlpaug library was used together with a synonym method, which was applied to the dataset. The labels were encoded using LabelEncoder, and data were then Tokenized. Test and training data were split into 80% training and 20% tests.

Two CNN sequential model implementations were tested.

The architecture of the first CNN model used is described in Figure 11. The sequential model is a deep learning model that is made from linear stack layers. Layers are added to the model in sequential order, and the output of each layer is the input of the next layer. The first proposed model consists of an Embedding Layer, a convolution layer, an activation

layer, another convolution layer, an activation layer, a MaxPooling layer, a Flatten layer, and one dense layer. For the activation layer, the Rectified Linear Unit (ReLU) activation function was used.


**Figure 11.** First CNN architecture.

This first proposed model ran 20 epochs, having at the end a test accuracy of 98.333340883255% and a loss of 10.25% with data augmentation applied one time and concatenated with the created dataset. This first CNN model is a tryout to see what needs to be done next to improve the accuracy and loss. There are a couple of options, such as augmenting the data even more, adding more layers, or hyper parametrizing the model. Figure 12 describes the training and validation accuracy graph as well as the training and validation loss graph.

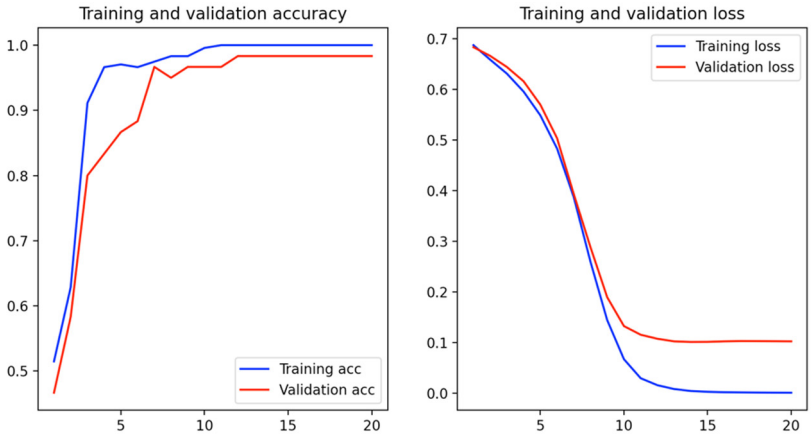


**Figure 12.** First CNN model graph for training and validation.

The precision, recall, and F1-score are presented in Table 1.

**Table 1.** First CNN model results.

| | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.97 | 1.00 | 0.98 | 32 |
| 1 | 1 | 0.96 | 0.98 | 28 |
| Macro avg | 0.98 | 0.98 | 0.98 | 60 |
| Weighted avg | 0.98 | 0.98 | 0.98 | 60 |

Table 1 presents the general parameters used after running the first CNN model with two Conv1D layers. The precision parameter measures the accuracy of positive predictions, meaning the model's ability to detect positive samples. The formula used is presented in (1).

$$\text{Precision} = \frac{(\text{True Positive})}{\text{True Positives} + \text{False Positives}} \quad (1)$$

The recall parameter or sensitivity means the ratio of correctly predicted positive observations to all observations in the actual class, meaning it measures the ability of the model to find the relevant cases. F1 is largely used in LLMs evaluation and binary

classification. A high F1-score means that the model can identify positive and negative cases. The formula used is shown in (2).

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positive} + \text{False Negatives}} \quad (2)$$

The F1-score is the mean between the precision and recall. It integrates the precision recall into one metric. The F1-score is computed as (2) presents.

$$F1 - score = \frac{2 \times \text{precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (3)$$

As presented in Table 1, the model presents high percentages of precision—97%, recall—100%, F1-score—98% for 0, and for 1, precision—100%, recall—96%, F1-score—98%. These results make the model reliable for both cases, with high precision, recall, and F1-score, performing very well in identifying relevant instances and making accurate predictions, with a good balance between precision and recall (achieving high precision and high recall at the same time). Thus, these parameters indicate the model is sturdy and dependable for the task given.

Another sequential model was developed to try and achieve better results. An architecture was defined using three convolutional layers, three activation layers, an embedded layer, one MaxPooling layer, one Flatten layer, and one dense layer, as presented in Figure 13.


**Figure 13.** Second CNN model architecture.

After applying data augmentation, we split the dataset into 80% training and 20% validation and trained the model with an accuracy of 96.66666984558105% and a validation loss of 5%. Further results, such as the precision, recall, and F1-score, are available in Table 2 and in Figure 14.

**Table 2.** Second CNN model results.

| | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 1.00 | 0.94 | 0.97 | 32 |
| 1 | 0.93 | 1.00 | 0.97 | 28 |
| Macro avg | 0.97 | 0.97 | 0.97 | 60 |
| Weighted avg | 0.97 | 0.97 | 0.97 | 60 |

This model architecture uses three convolutional layers, has very high precision scores for both 1 and 0 (100% and 93%), has a high recall percentage (94% and 100%), and has a good F1-score (97%). The performance of the second architectured model, which has three convolutional layers, is favorable and resilient. The results show that the model is robust and provides positive results in terms of accuracy, precision, recall, and F1-score, making it a viable and adequate model.

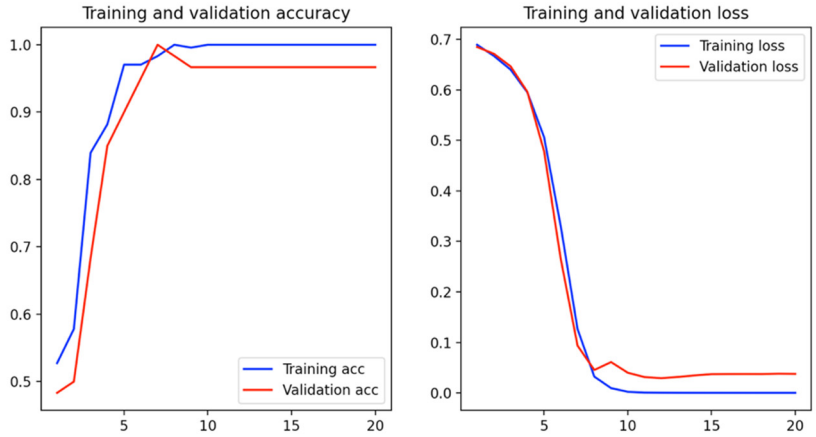


**Figure 14.** Second CNN model with three convolutional layers training and validation.

In this paper [32], a multitude of models were used to classify the binary data for two datasets: DATASET-1 (similar to the Stanford Sentiment Treebank with only negative and positive reviews) and DATASET-2 (Amazon review of Sentiment Analysis Dataset). The best models applied on those datasets, TextConvoNet_4 (four convolutional layers) and TextConvoNet_6 (six convolutional layers), achieved accuracies of 82.2% and 81.9%, respectively, on DATASET-1. For DATASET-2, TextConvoNet_4 has an accuracy of 90.4%, while TextConvoNet_6 has an accuracy of 87.2%. If these results are taken into account, the proposed two CNN architectures would outperform the TextConvoNet_4 and TextConvoNet_6 models. The accuracy that the Bidirectional Encoder Representations from Transformers (BERT), Hierarchical Attention Networks (HAN), and BerConvoNet models have achieved are 77.6%, 80.2%, and 83.1% on DATASET-1. For DATASET-2, the same models have reached 77.2% (Bidirectional Encoder Representations from Transformers-BERT), 86.3% (HAN), and 88.3% (BerConvoNet). Therefore, the models presented in this paper are performing better than the ones used in [32].

In [33], they used different learners on a cost estimation domain; by using different specific traits of different projects, they asked their learners to estimate the project development time [34]. As learners, they decided to use a Neural Network (NN) algorithm, a Genetic Program (GP) algorithm, and a Linear Standard Regression (LSR) algorithm. The accuracies provided by each approach are the following: 55.6% (NN), 55.6% (LSR), and GP (23.6%) [34]. Thus, the model presented in this paper outperforms these ones.

In the paper [35], several Neural Networks have been used both for binary classification and multiclass classification. For binary classification, the researchers used the IMDB database, including 25,000 movie reviews and other unlabeled data. The researchers changed the last fully connected layer to have, in one approach, a sigmoid function and, in the other approach, to make use of the softmax function [35]. The Neural Network architectures that used the sigmoid function reached 78.6% (Separable Convolutional Neural Networks—SCNN), 76.40% (Fully Connected Networks—FCN), RNN (68.35%), GRU (67.88%), and LSTM (68.89%), while the Neural Networks that used the softmax activation function have achieved 79.40% (SCNN), 76.95% (FCN), 78.25% (RNN), 79.25% (GRU) and 77.75% (LSTM) [35]. Taking the above results into consideration, the CNN implementations presented in this paper surpassed previous architectures presented on a newly created database, as the accuracies of both solutions in this research are 98.333340883255% for the first CNN implementation and 96.66666984558105% for the second CNN architecture.

## 5. Conclusions

In this paper, two AI implementations were used in order to aid the verification process in finishing in less time. The two approaches consisted of using two Convolutional Neural Network architectures, one with two convolutional layers and one with three convolutional layers. In both cases, data augmentation was performed. This means that, on the original

dataset, a method was applied to substitute some words with their respective English synonyms to enhance the dataset further without adding any entries. The dataset was created, developed, and analyzed by the researchers, using their experience in the functional verification field. Both Convolutional Neural Network models were developed using Python 3.12.3 and ran using an Apple M1 hardware on a dedicated 10-core GPU with an Anaconda environment.

For the first Convolutional Neural Network, a sequential model was defined, and layers were added to it. The sequential model is a deep learning model which consists of linear stack layers. The layers are added to the model in order, as the output from the previous layer is the input to the next layer. The layers added are an Embedding Layer, a convolutional layer, an activation layer, a convolutional layer, a MaxPooling layer, a Flatten layer, and one dense layer. In regards to the activation layer, the Rectified Linear Unit (ReLU) function was used. This model implementation generated fine results, high test accuracy (0.98333340883255%), high recall, precision and F1-score, and low validation loss (10.25%), making it a reliable, robust, and well-adjusted model. As the dataset continues to evolve, the model will be further adjusted and hyper parametrized to maintain its overall reliability and good performance.

The second Convolution Neural Network architecture is similar to the first architecture present, but an extra layer of Convolution was added to further enhance the results. The sequential model was used, and the embedded layer, the three convolution layers, three activation layers, one MaxPooling Layer, one Flatten layer, and one dense layer were added. As in the previous implementation, data were augmented and fed into the model. As before, the model generated dependable results with a validation accuracy of 96.66666984558105%, favorable precision, recall, and F1-score, and low validation loss −5%. These results make the model performant and sturdy on the existing data.

Regarding the problem of the increasing time spent in functional verification when designing and implementing an SoC, several solutions are considered. Artificial Intelligence can solve these issues by enhancing the verification process through different techniques. Three of them are generally taken into account when thinking about how Artificial Intelligence can improve the time spent on functional verification: grouping tests according to their failure reasons in order to ease the debug process of the verification engineer, tests generation in order to regenerate the stimuli to have a higher coverage percentage and document classification in order to determine the important features, implementations, and details in order to extract relevant data for the SoC in development. In this paper, the researchers have proposed two Convolutional Neural Network architectures, one with two convolutional layers and one with three convolutional layers. The dataset was created by the researchers based on their experience in the field and augmented using specific methods. The first architecture generated high accuracy 98.333340883255%, low validation 10.25%, and a good overall score for precision, recall, and F1-score. These results make the model well-built and dependable.

The second architecture generated high accuracy at 96.66666984558105% and low validation at 5%. The scores for precision, recall, and F1-score are also good overall, making the model trustworthy and solid.

Taking into consideration the other Neural Networks used for classification, such as Bidirectional Encoder Representations from Transformers (BERT), LSTM, and TextConvoNet, the CNN architectures presented in this paper outperform those respective architectures by a considerable amount. Thus, the results in this paper would be used as a starting point in the semiconductor industry. As more and more System-On-Chips are getting more intricate and sophisticated, the time spent on verification has greatly increased. With the rising complexity of System-on-Chips, different protocol specifications and documentations have become more and more difficult and arduous. Therefore, in this paper, the researchers created a dataset consisting of different text documentations and labeled it as important, crucial to the chip, or more redundant. Using these Convolutional Neural Network Archi-

tectures present in this paper might be a promising solution in order to aid the verification process by emphasizing the information needed from documentations.

Further improvements will be made as the database will be improved and expanded, and more augmentation will be added to it to enhance it. As the database grows, the two architectures will be more hyper parametrized and improved to continuously generate good overall results.

**Author Contributions:** Methodology, D.D. and C.D.; Software, D.D.; Validation, C.D.; Formal analysis, D.D. and C.D.; Writing—original draft, D.D.; Writing—review & editing, C.D. All authors have read and agreed to the published version of the manuscript.

**Funding:** The research received no external funding.

**Data Availability Statement:** Data are contained within the article. Data are not public.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Mammo, B. Reining in the Functional Verification of Complex Processor Designs with Automation, Prioritization, and Approximation. 2017. Available online: https://deepblue.lib.umich.edu/bitstream/handle/2027.42/137057/birukw_1.pdf (accessed on 15 May 2024).
2. Siemens. Available online: https://blogs.sw.siemens.com/verificationhorizons/2021/01/06/part-8-the-2020-wilson-research-group-functional-verification-study/ (accessed on 15 May 2024).
3. Truong, A.; Hellström, D.; Duque, H.; Viklund, L. Clustering and Classification of UVM Test Failures Using Machine Learning Techniques. In Proceedings of the Design and Verification Conference (DVCON), San Jose, CA, USA, 26 February–1 March 2018.
4. El Mandouh, E.; Maher, L.; Ahmed, M.; ElSharnoby, Y.; Wassal, A.G. Guiding Functional Verification Regression Analysis Using Machine Learning and Big Data Methods. In Proceedings of the Design and Verification Conference and Exhibition Europe (DVCon), Munchen, Germany, 25 September 2018.
5. Ismail, K.A.; Ghany, M.A.A.E. Survey on Machine Learning Algorithms Enhancing the Functional Verification Process. *Electronics* **2021**, *10*, 2688. [CrossRef]
6. Zaruba, F.A. *An Open-Source 64-bit RISC-V Application Class Processor and Latest Improvements*; ETH: Zurich, Switzerland, 2018.
7. Hughes, W.; Srinivasan, S.; Suvarna, R.; Kulkarni, M. Optimizing Design Verification using Machine Learning: Doing better than Random. In Proceedings of the Design and Verification Conference (DVCON-Europe), Virtual Conference, 26–27 October 2021.
8. Dinu, A.; Ogrutan, P.L. Opportunities of using artificial intelligence in hardware verification. In Proceedings of the 2019 IEEE 25th International Symposium for Design and Technology in Electronic Packaging (SIITME), Cluj-Napoca, Romania, 23 October 2019.
9. Accelera. "UVM Guide". Available online: https://www.accellera.org/images/downloads/standards/uvm/uvm_users_guide_1.2.pdf (accessed on 15 May 2024).
10. Varambally, B.S.; Sehgal, N. Optimising Design Verification Using Machine Learning. An Open-Source Solution. *arXiv* **2020**, arXiv:2012.02453.
11. Bengio, Y.; Ducharme, R.; Vincent, P. A neural probabilistic language model. *Adv. Neural Inf. Process. Syst.* **2001**, *3*, 1–11.
12. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
13. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef] [PubMed]
14. Yan, R.; Xiao, X.; Hu, G.; Peng, S.; Jiang, Y. New deep learning method to detect codeinjection attacks on hybrid applications. *J. Syst. Softw.* **2018**, *137*, 67–77. [CrossRef]
15. Agarwal, B.; Ramampiaro, H.; Langseth, H.; Ruocco, M. A deep network model forparaphrase detection in short text messages. *Inf. Process. Manag.* **2018**, *54*, 922–937. [CrossRef]
16. Zheng, S.; Hao, Y.; Lu, D.; Bao, H.; Xu, J.; Hao, H.; Xu, B. Joint entity and relation extraction based on a hybrid neural network. *Neurocomputing* **2017**, *257*, 59–66. [CrossRef]
17. Li, B.; Wang, Q.; Wang, X.; Li, W. *Tag Prediction in Social Annotation Systems Based on CNN and BiLSTM*; Springer International Publishing: Cham, Switzerland, 2017; Volume 10385.
18. Zhou, C.; Qiu, C.; Acuna, D.E. Paraphrase Identification with Deep Learning: A Review of Datasets and Methods. *arXiv* **2022**, arXiv:2212.06933.
19. Collobert, R.; Weston, J. A unified architecture for natural language processing: Deep neural networks with multitask learning. In Proceedings of the 25th International Conference on Machine Learning, Ser. ICML '08, New York, NY, USA, 5–9 July 2008.
20. Kalchbrenner, N.; Grefenstette, E.; Blunsom, P. A convolutional neural network for modelling sentences. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Baltimore, MD, USA, 22–27 June 2014.
21. Gong, Y.; Luo, H.; Zhang, J. Natural language inference over interaction space. *arXiv* **2017**, arXiv:1709.04348.

22. He, H.; Lin, J. Pairwise word interaction modeling with deep neural networks for semantic similarity measurement. In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego, CA, USA, 12–17 June 2016.
23. Graves, A.; Fernández, S.; Schmidhuber, J. Bidirectional lstm networks for improved phoneme classification and recognition. In Proceedings of the International Conference on Artificial Neural Networks, Warsaw, Poland, 11–15 September 2005; Springer: Berlin/Heidelberg, Germany, 2005; pp. 799–804.
24. Wang, Z.; Hamza, W.; Florian, R. Bilateral multi-perspective matching for natural language sentences. *arXiv* **2017**, arXiv:1702.03814.
25. Chen, Q.; Zhu, X.; Ling, Z.; Wei, S.; Jiang, H.; Inkpen, D. Enhanced lstm for natural language inference. *arXiv* **2016**, arXiv:1609.06038.
26. Ucar, A.; Karakose, M.; Kırımça, N. Artificial Intelligence for Predictive Maintenance Applications: Key Components, Trustworthiness, and Future Trends. *Appl. Sci.* **2024**, *14*, 898. [CrossRef]
27. Souza, R.; Nascimento, E.; Miranda, U.; Silva, W.; Lepikson, H. Deep learning for diagnosis and classification of faults in industrial rotating machinery. *Comput. Ind. Eng.* **2021**, *153*, 107060. [CrossRef]
28. Pierleoni, P.; Palma, L.; Belli, A.; Raggiunto, S.; Sabbatini, L. Supervised Regression Learning for Maintenance-related Data. In Proceedings of the 2022 IEEE International Conference on Dependable, Autonomic and Secure Computing, Calabria, Italy, 12–15 September 2022.
29. Liu, C.; Zhu, H.; Tang, D.; Nie, Q.; Zhou, T.; Wang, L.; Song, Y. Probing an intelligent predictive maintenance approach with deep learning and augmented reality for machine tools in IoT-enabled manufacturing. *Robot. Comput. Integr. Manuf.* **2022**, *77*, 10235. [CrossRef]
30. ARM. Available online: https://developer.arm.com/documentation/ihi0024/latest/ (accessed on 15 May 2024).
31. Taye, M.M. Theoretical Understanding of Convolutional Neural Network: Concepts, Architectures, Applications, Future Directions. *Computation* **2023**, *11*, 52. [CrossRef]
32. Soni, S.; Chouhan, S.S.; Rathore, S.S. TextConvoNet: A convolutional neural network based architecture for text classification. *Appl. Intell.* **2023**, *53*, 14249–14268. [CrossRef] [PubMed]
33. Burgess, C.J.; Lefley, M. Can genetic programming improve software effort estimation? A comparative evaluation. *Inf. Softw. Technol.* **2001**, *43*, 863–873. [CrossRef]
34. Menzies, T.; Pecheur, C. Verification and Validation and Artificial Intelligence. *Adv. Comput.* **2005**, *65*, 153–201. [CrossRef]
35. Solovyeva, E.; Abdullah, A. Binary and Multiclass Text Classification by Means of Separable Convolutional Neural Network. *Inventions* **2021**, *6*, 70. [CrossRef]