Feature Extraction from Design Documents to Enable Rule Learning for Improving Assertion Coverage

Kuo-Kai Hsieh[1], Sebastian Siatkowski[1], Li-C. Wang[1], Wen Chen[2], and Jayanta Bhadra[2]

[1]University of California, Santa Barbara
[2]NXP Semiconductors, Inc.

Abstract— Feature selection is essential to rule learning in the context of functional verification. In practice today, features are selected manually and the selection requires domain knowledge. In contrast, this work proposes using automatic feature extraction from design documents as a viable approach to support rule learning. To demonstrate its effectiveness, document-extracted features are employed to learn the rules for covering a set of assertions based on a commercial SoC. Experiments show that 100%-accurate rules can be obtained for more than 70% of the assertions.

I. Introduction

Despite the advancements of formal methods, simulation-based verification is the mainstay in industry thanks to its scalability and applicability to large modern SoC designs. Coverage metrics are used to measure the completeness of simulation-based verification. Commonly used coverage metrics include code coverage, toggle coverage, and functional coverage. A satisfactory coverage level is necessary for verification sign-off.

Metric-based verification is well adopted in industry, where the verification process is geared towards the area or functionality with insufficient coverage. However, it is a challenging problem to control the stimulus to hit certain design area or functional events since the relationship between the controllable parameters in stimulus and the coverage is quite obscure.

Previous works have been done on building feature-based rule learning frameworks to address this problem [1] [2]. In such learning methodologies, it is crucial to select the proper features on which the learning will be performed. There are several key considerations in the feature selection: (1) features should be selected at the proper level so that the relationship between them and the desired behavior is not too complicated to learn; (2) features should not be difficult to control; (3) features should be representative of capturing design behavior.

Previous works usually select the controllable parameters in the stimulus as the base for learning. While it means that the learned results can be directly applied to guide test generation, it is sometimes infeasible to get meaningful results because there are many levels of abstraction between the stimulus space and the desired behavior space. An alternative is to use signals at architectural and micro-architectural levels since they are "closer" to the desired behavior. In addition, architectural and micro-architectural level signals are not too hard to control. The problem is that there could be thousands of signals at architectural and micro-architectural levels, with there only being hundreds of samples for learning. To make learning effective, it requires selection of representative signals. Although statistical methods can be applied for feature selection, design knowledge plays an indispensable role in such an endeavor.

Manual selection of features could be costly as it requires a lot of efforts on reading design documents and the RTL code, which serve as the essential source of design knowledge. We might want to ask the question: can we automatically extract features from the design documents and RTL code as the initial feature set in rule learning? In this context, the features are a set of relevant signals for a specific assertion coverage event. We assume that the design documents contain most of the important architectural and micro-architectural signals and they are not difficult to control. Motivated by these assumptions, we propose a novel approach to extract relevant signals for learning assertion coverage events. We use the extracted signals as the base of our rule learning framework and evaluate the effectiveness of such an approach.

II. Related Works

For signal-level rule learning, [3] is a tool that is able to extract assertions, i.e. invariants composed of design signals, by analyzing simulation traces. However, the signals may be too low-level for a human to understand their characteristics.

This work is supported by Semiconductor Research Corporation, project 2012-TJ-2268.

978-1-5090-1558-0/17/$31.00 ©2017 IEEE

The signal selection problem in debugging looks similar to our feature selection problem [4] [5], but they are at the FPGA level or the post-silicon level and focus on the observability of signals.

To the best of our knowledge, this is the first work utilizing text mining techniques to extract features from documents and apply them in hardware verification.

III. The Proposed Approach

The proposed approach includes two parts and is illustrated in Fig. 1. The first part is feature extraction from documents, as seen in the upper half of Fig. 1, where text mining techniques are utilized to extract words relevant to design signals from documents. Then, signal mapping methods are employed to map the extracted words to real design signals. These design signals, or features, are the starting point of the rule learning. The second part, shown in the lower half of Fig. 1, depicts the flow of learning rules for improving assertion coverage. It starts with data collection and processing. Then, with the features obtained from the first part, it applies rule learning algorithms to induce rules for assertion coverage points. The four following subsections detail the four procedures in the shape of rectangles in Fig. 1.

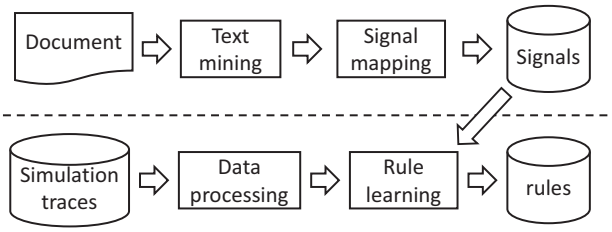

Fig. 1. The overall flow of the proposed approach.

A. Text Mining

The goal of this procedure is to extract words that are relevant to design signal names. Text mining is a process of extracting information from text and typically it involves natural language processing. There are two natural language processing techniques used in our approach: tokenization and part-of-speech tagging (POS tagging). The tagging results are then passed to a selection method to extract words of interest.

A-1. Tokenization

Tokenization is a technique used to segment a string into substrings. There are two types of tokenization: sentence tokenization and word tokenization. Sentence tokenization aims at partitioning text into sentences, while word tokenization splits a sentence into words.

The left side of Fig. 2 shows an example of sentence tokenization, where the input text is

"Are you OK? Dr. Pete is nearby.",

and the expected sentence tokenization result is

{"Are you OK?", "Dr. Pete is nearby."},

which is the same as how human understand the text. Sentence tokenization is not as trivial of a task as it may seem. For example, one may think that splitting a text by periods just works, however in the given example, splitting by periods ends up considering "Dr." as a sentence. One approach to solving this problem is to train a model to identify sentence boundaries [6].

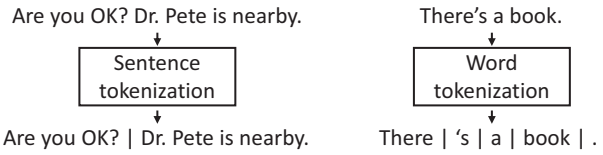

Fig. 2. Exmaples of tokenization.

The right side of Fig. 2 shows an example of word tokenization, where the input text is

"There's a book.",

and the expected sentence tokenization result is

{"There, "'s", "a", "book", "."}.

Note that different tokenizers may treat punctuation differently, e.g. making individual punctuations as tokens. Although splitting a sentence by spaces works most of the time, special cases must be taken care of such as verb contractions (e.g. can't) and Saxon genitive (e.g. mother's).

A-2. POS tagging

Part-Of-Speech (POS) tagging is a process to set a POS label to each word in a sentence (which is word-tokenized), where a POS is a word category in which words possess similar grammatical properties. Some simplified examples of POS are noun, verb, adjective, etc. In reality, there are more specific categories. Fig. 3 shows an example of POS tagging where the input sentence is:

FOO is a read-only, one-hot register.

In the tagging results, NNP stands for proper noun, VBZ stands for present and singular verb, DT stands for determiner, JJ stands for ordinal adjective, and NN stands for common singular noun. Commonly used methods include rule-based models [7] and stochastic models [8].

The POS tags provide grammatical information of each word in a sentence, which is useful for analyzing text. In our application, it is intuitive that the majority of signal names will be nouns or proper nouns, and are highly unlikely to be labeled as tags such as adjective and verb.

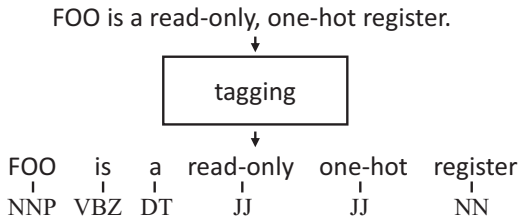

Fig. 3. An example of POS tagging.

A-3. Word selection

The objective of this step is to extract words relevant to the names of design signals. Typically it is customized and ad-hoc. We use a rule-based approach for word selection. First, we find all the words that are labeled as common noun or proper noun. Second, we design rules to select words of interest. Our rules are based on regular expressions and it is easy to add or remove a rule in our rule-based implementation. Example rules for hardware signal extractions include (1) words composed of only uppercase letters, and (2) words that have a underscore, "_". Lastly, a word is not selected if it is in our exclusion list, which contains known words that do not refer to design signals.

B. Mapping Text to Design Signals

This procedure maps the words extracted by text mining to the real design signals. Our approach is based on name matching and filtering. Fig. 4 depicts the flow of the proposed name mapping procedure.

For each word, we search if there are matches from the list of all the design signals. The first step is to find the signals whose names exactly match the queried word. If there is no such signal found, a partial matching method is applied, which tries to find if there are matches for certain substrings of the queried word. A typical substring to be matched can be chosen from segments of the queried word separated by underscores. Sometimes substrings that happen to be commonly used words (e.g., start, stop and etc) are also considered.

A filtering mechanism is required because normally there are lots of signals in the matching result. The filtering outputs signals with the highest score, where the score is calculated based on the length of matched string, the depth of the signal in the design hierarchy and whether the signal is in the module we are interested in.

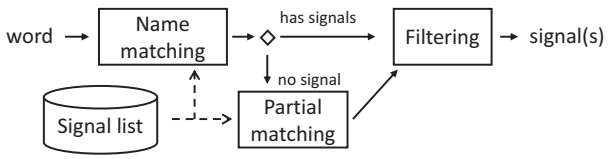

Fig. 4. The flow of the proposed mapping procedure.

C. Data Collection and Processing

The data required in the proposed approach are signal traces and the time when each assertion coverage point was covered. More specifically, given any time, we must be able to know the values of specified signals, and whether an assertion coverage point was hit or not from the collected data.

The goal of data processing is to provide proper inputs to the learning algorithm. It includes four steps as described below.

C-1. Time discretization

Given a set of signals, for each test, we are interested in the timestamps whenever a value change of the signals occurs. Formally speaking, given a test, given a set of signals, and let $V(t)$ be the vector of the signal values at time t, we extract the set of timestamps $\{t_1, t_2, \ldots, t_n\}$ satisfying that $\forall t \in [t_i, t_{i+1})$, $V(t)$ holds the same and that $\forall i, V(t_i) \neq V(t_{i+1})$.

C-2. Binarization

We treat the data as categorical. Suppose during simulation, the set of observed values of Sig_A is $\{v_1, v_2, \ldots, v_n\}$, then n features are created: "Sig_A=v_1", "Sig_A=v_2", ..., "Sig_A=v_n". At time t, the value of "Sig_A=v_i" is 1 if the value of Sig_A is v_i, otherwise "Sig_A=v_i" is 0.

There are potential exponential explosion problems. To our experience, it happened only for data path signals. If a signal has more than 16 values, then we let the users decide whether they want to include this signal in the feature set. Alternatively, they can decide to create fewer bins for the signal on their own.

C-3. Including Value Transitions

For each signal, we create another set of features to indicate the value transitions. Our empirical study shows that without these features, the performance of rule learning is not acceptable because most assertion coverage points involve temporal properties.

The procedure is similar to binarization, but the created features are "Sig_A=v_i to v_j". At time t_k, the value of the feature is 1 if Sig_A is v_i at time $t_k - 1$ and Sig_A is v_j at time t_k, otherwise 0.

C-4. Time alignment

The purpose of time alignment is to deal with the asynchronous relation between assertion coverage and signals. We observed several cases where assertion coverage has its dedicated clock. Creating new timestamps doesn't work because we can have two timestamps, t_1 and t_2, such that $V(t_1) = V(t_2)$ but one hits the coverage point and the other does not.

For each assertion coverage point, we find the maximum timestamp that is not greater than the time when the coverage point is hit. Then we use this timestamp as when the coverage point is hit.

D. Rule Learning Algorithms

Given a set of samples of different classes, rule learning algorithms aim at finding a rule that is able to distinguish samples from different classes. In our application, we have two classes of samples, i.e. positive samples that hit the target assertion coverage point and negative samples that do not. Our goal is to find a rule, composed of features processed from the previous procedure, that can explain why the target assertion coverage point can be hit. For example,

$$!(Sig_A{=}1) \wedge (Sig_B{=}0 \text{ to } 1) \Rightarrow \text{hit target coverage.}$$

There are many off-the-shelf rule learning algorithms. Examples include subgroup discovery [9], CN2 [10], and Classification and regression trees (CART) [11].

The proposed approach does not depend on a specific rule learning algorithm. In our experiment, we used CART.

CART belongs to a family of decision tree classifiers. Training decision tree classifier models is an iterative process. Given a set of samples, the training algorithm checks all the features and decides which one can best split the set into two subsets, where the best split is that where the two subsets are close to pure. A set is pure if it contains only samples of a single class. Commonly used metrics to measure the quality of splitting are Gini impurity and information gain. Then, the same procedure continues on the two subsets. This iterative process ends when a subset is pure, or there is no feature for further splitting.

Each path starting from the root of a decision tree model corresponds to a rule. The rule is the conjunction of the decisions along the path. Note that a path does not necessarily end at a leaf. Fig. 5 shows a decision tree example. All the right branches are True branches, and all the left branches are False branches. The highlighted path corresponds to the rule

$$!(Sig_A{=}1) \wedge (Sig_B = 0 \text{ to } 1).$$

Depending on the training results, we have different methods to extract rules from a decision tree. (1) If there are nodes that are pure and contain only positive samples, then the extracted rule is the disjunction of rules that correspond to the paths to all the pure and positive nodes. All the pure and positive nodes are treated equally. (2) Otherwise, the extracted rule is the one corresponding to the path leading to the node with the highest ratio of positive samples.

D-1. Dealing with Overfitting

Overfitting is a common machine learning problem whose root cause is either too many features, i.e. high dimensionality, or too few training data samples. To overcome this problem, we need to either remove irrelevant features or increase the number of training data samples. Increasing the number of training data samples does not work because of the extremely low hit rates of some assertion coverage points. Therefore, we resort to learning from smaller groups of features. This leads to the question of what features should be discarded and what features should be kept.

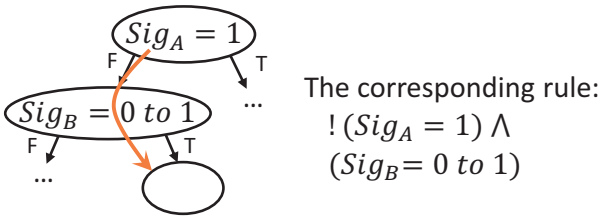

Fig. 5. Rule extraction of a given node.

The idea is to learn from other assertion coverage points. Often, not all the assertion coverage points are independent. A group of features that is relevant to one assertion coverage point can be relevant to another assertion coverage point. Following this thought, for each assertion coverage point that has 100%-accurate rules, a group is created containing all the features used in its decision tree. After this step, multiple groups are created. Then, for each group, rule learning algorithms are applied to assertion coverage points that have no 100%-accurate rule. The rule with the highest accuracy is reported as the final result.

IV. Experimental Results

The proposed approach was evaluated on a commercial dual-core microcontroller SoC targeting ultra-low power applications. The experiments focused on its system low-power control unit, which controls the system to enter and exit various power modes. This unit monitors triggering events of power mode transitions and then generates proper control sequences to clocks, power, and system memories to execute the transitions.

The in-house verification environment was a C-test based environment. The C stimuli were compiled into machine code and then executed on cores in RTL simulation. The correctness was insured both by self-checks in C-stimuli and the checkers in the testbench. Assertion coverage data was collected by a commercial coverage reporting tools.

We applied the proposed approach to 168 assertion coverage points created by other engineers during project development. Note that these assertion coverage points concern not only the system low-power control unit activities, but also other activities in the system to capture the overall system states of interest.

1000 tests were pre-run in this experiment, and they were partitioned into two equally sized sets for training and validation respectively. For each assertion coverage point, based on the features extracted from the document, a rule was learned from the training tests. Then, each rule was validated on the other 500 tests. We show the

effectiveness of our approach by comparing the hit rate of the training tests, which are randomly generated, to the hit rate of tests satisfying the rules of all the assertion coverage points.

Without feature selection processes, it is infeasible to apply rule learning algorithms. Before running into the theoretical overfitting problem, in practice, the cost of collecting and processing simulation data is prohibitive in terms of time and space. For instance, it required more than 2TB storage to save the whole chip simulation traces of 1000 tests.

A. Signal Extraction Based on Design Documents

The design document to start with is the design reference manual, which is a 49-page PDF file. There is no formal format of this document. In natural language, it describes design functionality, register usages, interface protocols, etc., with plenty of tables, diagrams, and waveforms.

We processed this document in Python2.7. The Python package pdfminer [12] was used to extract text from the PDF document. Next, the Python package nltk [13] was used to tokenize the text and tag the words. An exclusion list was created to ensure that we did not select words that were obviously irrelevant to design signals. After this procedure, 66 words were extracted from the document for the mapping procedure.

After executing the mapping procedure, there were 42 words that could be mapped to design signals. The 24 words that did not have mappings included the names of other hardware modules and special abbreviations. 46 design signals were obtained after the mapping procedure. We observed that two words may map to the same design signals, which is reasonable because the document is written in natural language and there is no strict format requirement to the document. Also, a word may map to multiple design signals with different prefix or postfix in the signals names. These facts explain why the number of words having signal mapping is different from the number of signals of the mapping result. The signal extraction results is shown in Table I.

TABLE I
TEXT MINING RESULTS

# words after text mining	66
# words having signal mapping	42
# signals of the mapping result	46

B. Rule Learning Based on the Extracted Features

We implemented our learning methods in Python using Pandas [14] and scikit-learn [15]. Pandas was used to process data as described in Section C, and scikit-learn provided us the implementation of CART.

There are 300 features in total after running the data processing procedure. The number of training samples for rule learning is 9216.

Fig. 6 shows the rule learning results based on all the signals extracted from documents. The y-axis is the hit rate, and the x-axis is the assertion coverage point sorted by its hit rate after learning. Each assertion coverage point has two bars. The orange bar is the hit rate after learning, and the blue bar is the hit rate of random test generation. The results show that for more than 60% of the assertion coverage points, 100%-accurate rules can be learned. The result implies that given an assertion coverage point, with chances over 50%, an engineer without much design knowledge can obtain accurate rules.

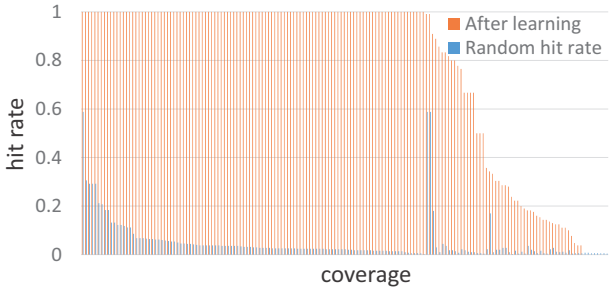

Fig. 6. Hit rate improvement between random and rule learning results of all the assertion coverage points.

To overcome the overfitting problem, we ran the rule learning algorithms again with smaller groups of features. For each assertion coverage point that has 100%-accurate rules, a signal group is created comprising all the features in the corresponding decision tree. Then the rule learning algorithm is applied with each signal group.

Fig. 7 shows the learning results on the assertion coverage points that have no 100%-accurate rule at the previous stage. 38 out of 58 assertion coverage points have hit rate improvement. In addition, 11 assertion coverage points have 100%-accurate rules.

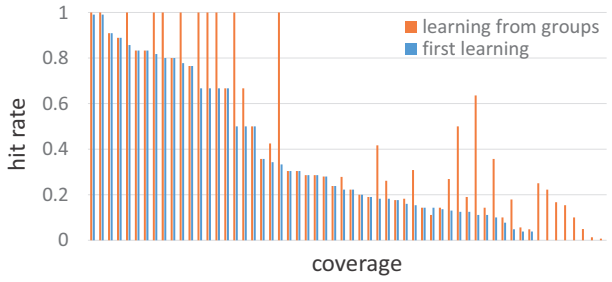

Fig. 7. Hit rate improvement after learning from signal groups.

Table II shows the hit rate improvement of selected assertion coverage points with very low random hit rate. It clearly demonstrates the overfitting phenomena: when data is insufficient and the number of features is large, the learning result cannot be generalized, thus we have rules with 0 hit rate when learning with all the signals from

documents. However, if we can exclude irrelevant features, the learning works with the same amount of data. This explains why we got hit rate improvement by learning from smaller groups.

TABLE II
HIT RATE IMPROVEMENT BETWEEN LEARNING FROM ALL THE EXTRACTED SIGNALS AND FROM SIGNAL GROUPS

	random	all doc signals	signal groups
Assertion 1	1/500	0%	25%
Assertion 2	1/500	0%	22%
Assertion 3	1/500	0%	17%
Assertion 4	2/500	0%	15%

Table III summarizes the overall results of our experiments. With the signals extracted from documents, we obtained 100%-accurate rules for more than 70% of the assertion coverage.

TABLE III
THE PERCENTAGE OF THE ASSERTION COVERAGE HAVING 100%-ACCURATE RULES

random	all doc signals	signal groups
0.0%	64.9%	71.4%

The result suggests that the set of signals extracted from documents provides a good starting point for engineers who do not have deep knowledge of the design under verification. If the goal is to find accurate rules for all the assertion coverage points, the proposed approach ramps up from nothing to 70% without much effort. On the other hand, rules not 100%-accurate also provide information for engineers to analyze designs and tests, and assist in applications such as generating more tests and debugging.

V. CONCLUSION

This work proposed an approach to enable rule learning for improving assertion coverage by feature extraction from design documents. Text mining methods are applied to obtain words likely to be the names of design signals. These words are then mapped to the real design signals to create the feature set. The effectiveness of these features is experimented by learning rules for improving assertion coverage. The experimental result shows that for more than 70% assertions, 100%-accurate rules can be obtained. Moreover, it significantly boosts the hit rate for those rarely-hit assertion coverage points.

REFERENCES

[1] W. Chen, L.-C. Wang, J. Bhadra, and M. Abadir, "Simulation knowledge extraction and reuse in constrained random processor verification," in *Proceedings of the 50th Annual Design Automation Conference*, ACM, 2013, p. 120.

[2] Y. Katz, M. Rimon, A. Ziv, and G. Shaked, "Learning microarchitectural behaviors to improve stimuli generation quality," in *Proceedings of the 48th Design Automation Conference*, ACM, 2011, pp. 848–853.

[3] S. Vasudevan, D. Sheridan, S. Patel, D. Tcheng, B. Tuohy, and D. Johnson, "Goldmine: Automatic assertion generation using data mining and static analysis," in *Proceedings of the conference on Design, automation and test in Europe*, European Design and Automation Association, 2010, pp. 626–629.

[4] X. Liu and Q. Xu, "Trace signal selection for visibility enhancement in post-silicon validation," in *Proceedings of the Conference on Design, Automation and Test in Europe*, European Design and Automation Association, 2009, pp. 1338–1343.

[5] H. F. Ko and N. Nicolici, "Algorithms for state restoration and trace-signal selection for data acquisition in silicon debug," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 2, pp. 285–297, 2009.

[6] J. C. Reynar and A. Ratnaparkhi, "A maximum entropy approach to identifying sentence boundaries," in *Proceedings of the fifth conference on Applied natural language processing*, Association for Computational Linguistics, 1997, pp. 16–19.

[7] E. Brill, "A simple rule-based part of speech tagger," in *Proceedings of the workshop on Speech and Natural Language*, Association for Computational Linguistics, 1992, pp. 112–116.

[8] T. Brants, "Tnt: A statistical part-of-speech tagger," in *Proceedings of the sixth conference on Applied natural language processing*, Association for Computational Linguistics, 2000, pp. 224–231.

[9] N. Lavrač, B. Kavšek, P. Flach, and L. Todorovski, "Subgroup discovery with cn2-sd," *Journal of Machine Learning Research*, vol. 5, no. Feb, pp. 153–188, 2004.

[10] P. Clark and T. Niblett, "The cn2 induction algorithm," *Machine learning*, vol. 3, no. 4, pp. 261–283, 1989.

[11] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

[12] Y. Shinyama, *Pdfminer: Python pdf parser and analyzer (2010)*.

[13] S. Bird, "Nltk: The natural language toolkit," in *Proceedings of the COLING/ACL on Interactive presentation sessions*, Association for Computational Linguistics, 2006, pp. 69–72.

[14] *Http://pandas.pydata.org/*.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.