SpecToSVA: Circuit Specification Document to SystemVerilog Assertion Translation

Ganapathy Parthasarathy, Saurav Nanda, Parivesh Choudhary, and Pawan Patil
Synopsys Inc
Mountain View, California, USA
{gpartha,sauravn,parivesh,pawanp}@synopsys.com

ABSTRACT

Assertion-based verification is a technique to ensure that a microelectronic circuit design conforms to its specification and helps detect errors early in the design process. These assertions are complex and difficult to write since verification engineers must manually translate a natural language specification to a formal assertion language, such as SystemVerilog Assertions (SVA). SVA is a regular language that can be compiled and automatically checked. This incurs significant costs in the hardware design and verification cycle in terms of productivity and time, sometimes as much as 50% of the hardware design costs. We propose a machine-learning approach to alleviate this problem that automatically converts content in English language specifications to SystemVerilog Assertions. Our experimental results demonstrate an average precision of 64% on a data set created from proprietary Integrated Circuit (IC) specification documents.

CCS CONCEPTS

• **Computing methodologies → Machine translation**.

KEYWORDS

Deep Neural Networks, Document Parsing, Sentence Classification

1 INTRODUCTION

The increasing complexity of IC designs in size and functionality means that their specifications have evolved from telling designers what to design, to account for verification needs as well. Consequently, modern IC specification documents describe both desired functionality as well as the verification plan and task breakdown. A designer often works with only a small portion of the specification to develop his assigned portion of the product. Verification involves both the operating environment and the internal operation of the IC. Hence, verification engineers developing the verification environment, plan, tasks/items, and associated test-benches must understand the entire specification. The problem is compounded by the fact that the specification and associated supplemental documentation are often subject to changes that must be propagated through the entire design/verification flow.

A primary task in the verification flow is that of translating verification items into formal language models (assertions or checkers). These are written in hardware description languages such as SystemVerilog [1] and used to either prove (using formal methods) or check (using simulation) design properties. This is done under formal assumptions about the environments specified for normal operation of the IC. The complexities and manual work involved in the aforementioned process can introduce a significant number of errors that are costly to detect and fix.

We study the following problem – *Given an English language IC design specification, can we automatically a) identify sentences corresponding to IC functional properties and b) translate them to compileable, semantically correct SystemVerilog Assertions(SVA) ?*

We propose a solution (called SpecToSVA) to the above problem that uses Machine Learning (ML) and Natural Language Processing (NLP). SpecToSVA is a human-in-the-loop system that automatically translates text from IP specifications to formal assertions in the SystemVerilog hardware description language. We assume that there will be some error in the automatically generated SVA statements. Therefore, we provide an interactive system that allows users to correct errors in the system. These user-provided corrections are used as batched training data to minimize the overall error over time. This allows users to converge on desired precision for translation. The benefits of the proposed system include:

- Reduced time to create assertions;
- Reduced error rate;
- Ability to learn and improve quality of results as the size of the training data set increases.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 describes in detail the components of the SpecToSVA system. In Section 4 we present an experimental study and evaluation of the proposed system.

2 RELATED WORK

The earliest related work by Granacki et. al [6] [7] generates partial hardware designs from natural language specifications by identifying a set of concepts and a textual pattern for each concept. Any sentence which matches a textual pattern can be mapped to structures in a design data structure defined by the authors – a so-called slot-filling approach. Much of the current work in the literature emulates this approach in the translation of English natural language specifications with varying levels of sophistication in the analysis of the source natural language.

All prior-art related to the proposed approach that we are aware of, applies some form of rule-based translation from English sentences to SystemVerilog Assertions. For example, the approaches in Fantechi et. al [5] and Holt et. al [9] transform the problem into a direct translation problem by restricting the input English to a constrained subset of English. The approaches described in Harris et. al. relies on creating partial grammars that are defined by hand [21], or through search techniques [8]. Other approaches like those in Soeken et. al [20] [11] attempt to alleviate some of the manual work involved in generating verification collateral from natural language specifications by sentence grouping.

More closely related work is the approach by Zhao et. al. [21] that creates a syntactic parse tree of English using the NLP toolkit

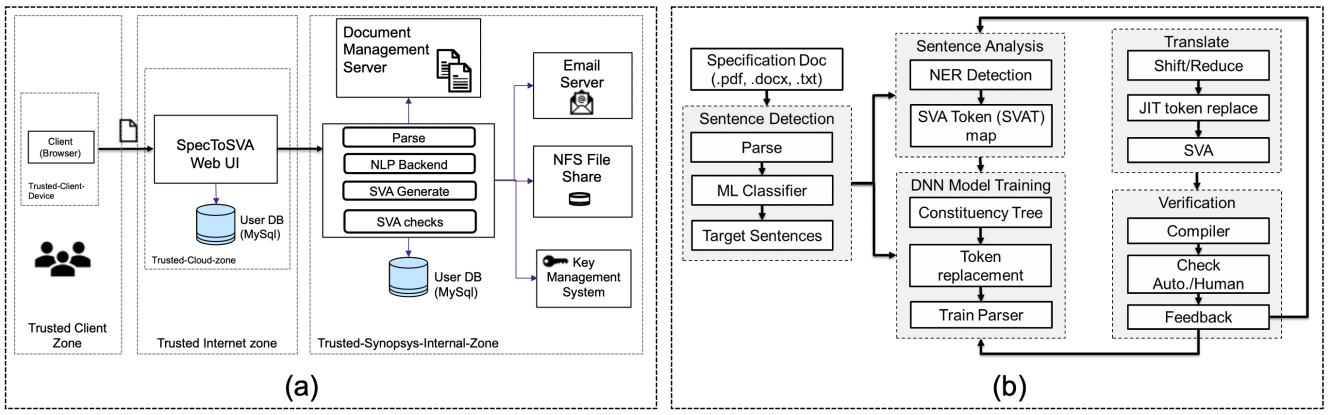

Figure 1: System architecture and proposed solution for each sub-problem including the execution flow.

[16]. Their approach attempts to locate subtrees associated with important phrases heuristically, such as the antecedent and consequent of an implication. Formal assertions are generated using a recursive walk over the subtrees to fill a set of assertion templates using a slot-filling method. The shortcomings of this approach besides its heuristic nature are both the small set of templates leading to limited forms of generated SystemVerilog, and the potentially infinite set of rules and heuristics required to find matches between semantic parts of target SVA templates and subtrees of the English parse tree. Similarly, Krishnamurthy et. al. [14] analyzes the English parse tree to determine if a sentence can be translated to SystemVerilog by a production rule guided slot-filling method.

3 SPECTOSVA APPROACH

SpecToSVA is a commercial system with an intuitive Software-as-a-Service(SaaS) architecture, as shown in Figure 1(a), and a set of components that allow users to upload documents, track changes, and generate SVA assertions from uploaded documents. In addition, users can manually correct the results of intermediate steps in the translation and provide feedback on inaccuracies in the translation. The offline component shown in Figure 1(b) is the central part of the work described in this paper.

Our approach translates the high-level problem posed in the last section to solving the following four distinct sub-problems in a systematic flow as shown in Figure 1(b) – Target Sentence Detection, Sentence Analysis, Deep Neural Network (DNN) [19] model training, and finally the actual translation. We also have a final step for verifying the results from the translation and generating curated data for the sentence analysis and model training steps to improve quality of results over time. These are described in the following sub-sections 3.1 to 3.4.

As in any natural language processing flow, we first parse and extract all English sentences from a given IP design specification documents and store it in an input format agnostic database. This database serves as both a primary data-store and as a knowledge base for the entire system. We use a normalized schema to store all extracted information from the IP design specification documents, such as the section/subsection headings with their relative hierarchy, English sentences, and text automatically extracted from tables and figures.

In addition, basic tasks such as sentence detection, tokenization, part-of-speech tagging, named-entity tagging, and key phrase recognition are performed and the results of each task are aggregated and stored as metadata per sentence. The metadata per sentence also includes scope information such as captions, equations, table headers, and formatted lists.

3.1 Sentence Classification

The sentence classification in SpecToSVA uses an ensemble of atomic weak learners feeding a binary classifier as described by Ratner et. al. [18]. The input to a weak-learner is an ordered set of tokens, and the output is a binary label (*to_be_translated* or *not_to_be_translated*). These weak learners comprise procedures that use a range of techniques from regular expressions to rule-based procedures over token spans.

We create training data manually from randomly sampled sentences in documents marked for training. This training data is used to train a machine-learning based binary classifier that classifies a token-span through the ensemble of weak-learners to the appropriate bucket. We use a random-forest classifier for this step, following the standard methods of train/validation data split, train, and hyper-parameter tuning for highest quality of results. In general, any good binary classifier can be substituted in this step. We then improve the quality of results in the previous step that may be limited by the small size of initial training data by generating additional training data based on the following two techniques:

(1) We find sentences that are closest to the initial training data for the classifier using unsupervised machine learning based clustering techniques. The clustering algorithm uses n-gram similarity [13] as a distance metric for the clustering algorithm. The sentences in each cluster are then manually curated for validity and fed back to the training flow.
(2) We allow user-feedback at the system-level to correct misclassified sentences that are then fed back to the sentence classification flow as new training samples.

3.2 Name Entity Recognition (NER) Detection

We used the popular open-source NLP package – Spacy [10], for NER detection. We enhanced Spacy's English model with additional keywords/phrases that correspond to common terms and phrases in the circuit design and verification vernacular and SVA language reserved keywords to create a custom NER detector for our application. Examples of such tokens are variable names, and their semantics like clocks and resets, registers, bit-vectors, bit-fields, Boolean logic, and temporal behavior such as transitioning from one value to another over time. The interested reader is referred to the *SystemVerilog Language Reference Manual* [1] for more details. The details of our method are described as follows:

(1) We create a global set of labels that includes the full set of keywords and operators in the SystemVerilog LRM [1], and any NER tokens.
(2) We use heuristic methods to find noun phrases using meta-information generated during sentence pre-processing. The meta-information includes data from tokenization, chunking, and part-of-speech (POS) tagging. We then use constituency tree parsing to detect sequences of tokens (spans) that may have semantic meaning.
(3) We use custom rule-based routines to tag detected spans in a sentence with corresponding labels for each span. These rules vary from simple regular expressions to heuristic combinations of part-of-speech tags to mark a span with a specific label. In addition, we also create manually tagged spans where no rule-based mechanism can be created. These tagged spans are used to create training data for the next step.
(4) We then train a custom Spacy NER detection model using a word embedding to create a multi-label classifier that detects and assigns token-spans to the appropriate label.
(5) This labelling or mapping between the English token spans to SVA tokens and literals is stored in the document database and is used in the final translation step.

3.3 Model Training

We use the *English Language Parse* (ELP) tree as the fundamental data-structure to generate the final translated SVA. We use the structure of the English sentence as a guide to yield a parse schedule using shift-reduce operations [2] to generate the SVA representation. We use the constituency tree created using the deep neural network approach proposed by Kitaev et. al. [12], as a good approximation of the parse tree for a sentence . We use a second neural network model that uses an post-order traversal sequence of the ELP tree tokens to train a model that creates a pointer state to help traverse the ELP tree encoding in stack-order. The pointer state produces a new state at every step of the tree traversal given the current state of the traversal. This encodes a shift-reduce schedule for the ELP tree. Given these models, we can train the network with constituency trees from the sentences in our training data to create a joint encoding of the ELP tree and a traversal that approximates shift and reduce actions as described in [4].

Our method enhances the Kitaev model output by post-processing the constituency tree to mark sub-trees corresponding to detected reserved keyword NER tags and replacing vernacular NER tags with a special symbol – "OOV". This is done using a straight-forward post-order traversal of the ELP tree from the Kitaev model and results in an enhanced ELP tree. This increases the probability that the two learned embeddings (tree/pointer) will be able to represent the boundaries of phrases and sub-phrases with high accuracy. This is shown in the example in Figure 2. The intuition behind the necessity for this step is that many NERs and phrases in specification documents depend on artificial names that are created by IC design engineers for a specific IC. It is highly unlikely that they will repeat across documents. As a result, the overall accuracy of the embedding learned on a sampled training set tends to suffer.

We use tags of the form "(S1", "(S2", "(S3", …, "(SN" to mark the beginning of sub-phrases as opposed to current state-of-art [12] that marks each boundary with the same tags e.g., "(S", ")" to mark begin and end of a sentence/sub-phrase. The beginning, "S" and end ")", demarcation tags of a sentence are left unchanged. The training sequence to the probabilistic parser model is the enhanced ELP tree tokens sequence as an post-order traversal instead of the original constituency tree.

3.4 Translation to SystemVerilog Assertions

The final translation relies on the probabilistic shift-reduce schedule generated from the enhanced ELP model described earlier. The translation process uses the following steps:

- Given a sentence, we generate the enhanced constituency tree with NER mapping.
- The enhanced constituency tree is converted to a sequence that is input to the probabilistic parser.
- The probabilistic parser produces a sequence of shift-reduce actions represented by pointer states as an implicit bottom-up traversal equivalent to a stack of shift-reduce actions and a sequence of translated SVA keyword, OOV tokens, or an end of sequence marker.
- We track shift actions for every token, by tracking the parent and child constituency nodes in the tree. We also keep track of the last generated OOV symbol.
- We execute shift actions until the parse predicts a reduce action, which is characterized by one of the following: (1) We see an end of sequence marker. Here the reduce action is to terminate the translation with a semi-colon; or (2) We see an OOV marker. The OOV marker is mapped to its corresponding token span based on the position of the OOV in the enhanced ELP. The OOV is then translated to the appropriate SystemVerilog token span from the NER mapping defined in section 3.2.

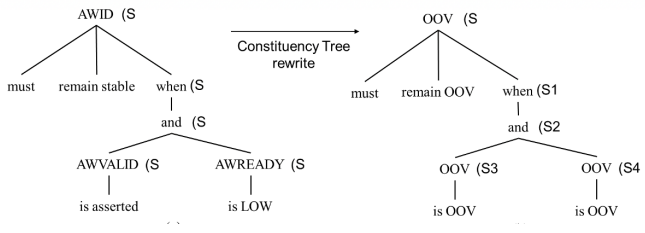

Figure 2: Illustration of constituency tree enhancement for the sentence "AWID must remain stable when AWVALID is asserted and AWREADY is LOW".

##	Input English	Output SVA
01	IRESP remains stable when IVALID is asserted and IREADY is LOW.	(ivalid && ! iready) |=>$stable (iresp)
02	IREQ is only permitted to change from HIGH to LOW when IACK is HIGH.	$fell (ireq) |->iack
03	When *transmit data is written to SPDR/SPDR_HA* and when *the transmit buffer of SPDR/SPDR_HA is empty (data for the next transfer is not set)*, the SPI writes data to the transmit buffer and clears the SPSR.SPTEF flag to 0.	((*def_wr_spdr_spdr_ha*) && (*user_def_spdr_spdr_ha_empty*) |-> (## *user_def_tim01*) $fell(sptef))

Table 1: Translations examples: English sentences to SVA

4 EXPERIMENTAL RESULTS

We implemented SpecToSVA as a commercial system that was used by customers on their proprietary IC design specification documents to generate SVA translations. The results of the system were validated by selected verification engineers from the customers IC verification teams(s). We created training data from a public specifications such as the ARM AMBA 3 AXI Protocol Checker User Guide [3] and from proprietary IC design specification documents.

Some example translations performed by SpecToSVA are shown in Table 1. Note that the first two example translations are complete and correct SVA expressions that do not require further modification from the user. The third example in Table 1 is a more complex sentence that requires user intervention for completion. We can observe that the translation correctly infers that a delay needs to be inserted before SPTEF signal goes to the value 0. SpecToSVA inserts a tag *user_def_tim01* since the translation does not know the exact value of the delay. This indicates that the user needs to modify the generated SVA to the correct value. The translation infers that the phrase – *the transmit buffer of SPFT/SPDR_HA is empty*, is part of the pre-condition for the value change on SPTEF. SpecToSVA inserts a tag – *user_def_spdr_spdr_ha_empty*, since the exact logic conditions for this phrase cannot be inferred from this sentence alone. This indicate that the user needs to insert the correct logic expression corresponding to this tag. Note that this is counted as a correct translation, since the system correctly indicates when and where the user needs to intervene.

We have not used traditional metrics, such as BLEU [17] or ROUGE [15] score to measure the translation precision as our correctness criterion is more stringent – *produce syntactically and semantically correct SVAs that can be compiled successfully*. Hence, our metric is based on the more traditional precision metric from binary classification. This is represented in the equation 1 below.

$$Precision = TP/(TP + FP) \tag{1}$$

where, TP represents the True Positive and FP represents False Positive. SVA translations that were marked as correct by human engineers are counted as TP and all others are counted as FP. The FP instances, by definition, include all the sentences which were classified as *to_be_translated*, but were not translate-able or marked by a human being as incorrectly classified.

Dataset	Sentences	Classified	Translated	Precision
1	4896	4896	3605	73.63%
2	43	43	27	62.79%

Table 2: Quality of results on internal IP datasets

We were able to achieve a precision of 73% on a data set we created from proprietary IC design documents and 63% on handcrafted benchmarks datasets created by customers, as shown in Table 2. SpecToSVA also demonstrated satisfactory results on large customer documents with excellent quality of results (QoR) as shown in Table 3. These results were validated by actual users of the system on documents that they were familiar with.

Docs	Sentences	Classified	Translated	Precision
Doc-1	679	622	476	70.1%
Doc-2	1124	981	555	49.37%
Doc-3	12962	11745	8274	63.83%
Doc-4	19060	16933	12186	63.93%

Table 3: Quality of results on commercial IC Specifications.

5 CONCLUSION AND FUTURE WORK

We have described a commercial system (SpecToSVA) for translating English language IC design specification document to SVA. SpecToSVA provides a complete flow for IC designers and verification engineers to identify potential verification items from a IC design specification and translate those verification items to formal language models i.e. assertions or checkers. Experimental results support the utility of SpecToSVA across different commercial/public IC design specification documents with reasonable precision that significantly improved the productivity of verification engineers.

There are open problems that we plan to address in future work. Currently, SpecToSVA can translate only single and self-contained sentences to SVA. However, there are many instances where multiple sentences must be considered for translation to a single SVA construct. The challenge is to establish cross-references between different sentences in a document that may have more than 1000 pages and then combine those sentences in a such a way that you can translate it to meaningful SVA.

Hence, there are two main problems we plan to address: 1) Find co-references between two or more sentences that can potentially be used for translation; and 2) Combine and/or re-write those sentences that can be consumed by SpecToSVA for translation purposes. We will extend SpecToSVA in future to address these two challenges. In addition, all the usual challenges in complex documents such as inferring semantics of diagrams, formatting (such as nested bullet-ed lists) and so on need to be addressed.

REFERENCES

[1] 2018. IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language. *IEEE Std 1800-2017 (Revision of IEEE Std 1800-2012)*, 1–1315. https://doi.org/10.1109/IEEESTD.2018.8299595

[2] Alfred V Aho, Ravi Sethi, and Jeffrey D Ullman. 1986. Compilers, principles, techniques. *Addison wesley* 7, 8 (1986), 9.

[3] ARM. 2009. AMBA 3 AXI Protocol Checker User Guide, ARM, Jun. (2009).

[4] Samuel Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D Manning, and Christopher Potts. 2016. A Fast Unified Model for Parsing and Sentence Understanding. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1466–1477.

[5] Alessandro Fantechi, Stefania Gnesi, Gioia Ristori, Michele Carenini, Massimo Vanocchi, and Paolo Moreschini. 1994. Assisting requirement formalization by means of natural language translation. *Formal Methods in System Design* 4, 3 (1994), 243–263.

[6] JJ Granacki and Alice C Parker. 1987. PHRAN-SPAN: a natural language interface for system specifications. In *Proceedings of the 24th ACM/IEEE Design Automation Conference*. 416–422.

[7] John J Granacki, Alice C Parker, and Yigal Arena. 1987. Understanding system specifications written in natural language. In *Proceedings of the 10th international joint conference on Artificial intelligence-Volume 2*. 688–691.

[8] Christopher B Harris and Ian G Harris. 2016. Glast: Learning formal grammars to translate natural language specifications into hardware assertions. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 966–971.

[9] Alexander Holt and Ewan Klein. 1999. A semantically-derived subset of English for hardware verification. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics*. 451–456.

[10] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. *spaCy: Industrial-strength Natural Language Processing in Python*. https://doi.org/10.5281/zenodo.1212303

[11] Oliver Keszocze, Mathias Soeken, Eugen Kuksa, and Rolf Drechsler. 2013. Lips: An ide for model driven engineering based on natural language processing. In *2013 1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE)*. IEEE, 31–38.

[12] Nikita Kitaev and Dan Klein. 2018. Constituency Parsing with a Self-Attentive Encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2676–2686.

[13] Grzegorz Kondrak. 2005. N-gram similarity and distance. In *International symposium on string processing and information retrieval*. Springer, 115–126.

[14] Rahul Krishnamurthy and Michael S Hsiao. 2019. Controlled natural language framework for generating assertions from hardware specifications. In *2019 IEEE 13th International Conference on Semantic Computing (ICSC)*. IEEE, 367–370.

[15] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*. 74–81.

[16] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*. 55–60.

[17] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 311–318.

[18] Alexander Ratner, Braden Hancock, Jared Dunnmon, Frederic Sala, Shreyash Pandey, and Christopher Ré. 2019. Training complex models with multi-task weak supervision. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4763–4771.

[19] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural networks* 61 (2015), 85–117.

[20] Mathias Soeken, Christopher B Harris, Nabila Abdessaied, Ian G Harris, and Rolf Drechsler. 2014. Automating the translation of assertions using natural language processing techniques. In *Proceedings of the 2014 Forum on Specification and Design Languages (FDL)*, Vol. 978. IEEE, 1–8.

[21] Junchen Zhao and Ian G Harris. 2019. Automatic Assertion Generation from Natural Language Specifications Using Subtree Analysis. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 598–601.