VToT: Automatic Verilog Generation via LLMs with Tree of Thoughts Prompting

Yingjie Zhou[1]*, Renzhi Chen[2]*, Xinyu Li[1], Jingkai Wang[1], Zhigang Fang[1], Bowei Wang[1],
Wenqiang Bai[1], Qilin Cao[1], Lei Wang[3,2]†
*(*Equally Contributed Authors, †Corresponding Author)*
[1]National University of Defense Technology, Changsha, China, [2]Qiyuan Laboratory, Beijing, China
[3]Defense Innovation Institute, Academy of Military Sciences, Beijing, China
{zhouyingjie}@nudt.edu.cn

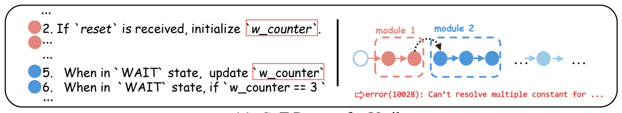
***Abstract*—The automatic generation of Verilog code using Large Language Models (LLMs) presents a compelling solution to enhance the efficiency of hardware design flow. However, the state-of-the-art performance of LLMs in Verilog generation remains limited compared to programming languages such as Python. Previous research, Chain of Thought (CoT), has demonstrated that incorporating intermediate reasoning steps can significantly improve the performance of LLMs in code generation. In this paper, we propose the Verilog Tree of Thoughts (VToT) method. This structured prompting technique addresses the abstraction gap between Verilog and CoT by embedding hierarchical design constraints within the prompt. Experimental results on the VerilogEval and RTLLM benchmarks demonstrate that VToT prompting enhances both the syntactic and functional correctness of the generated code. Specifically, according to the RTLLM benchmark, VToT achieved a correctness rate of 75.9% at pass@5, representing an improvement of 10.4%. Furthermore, in the VerilogEval benchmark, VToT achieved state-of-the-art performance with a correctness rate of 52.4% at pass@1 (an increase of 8.9%) and 65.4% at pass@5 (an increase of 9.6%).**

***Index Terms*—Verilog, LLMs, Chain-of-Thought, Tree-of-Thoughts, Integrated Circuit Design Automatization**

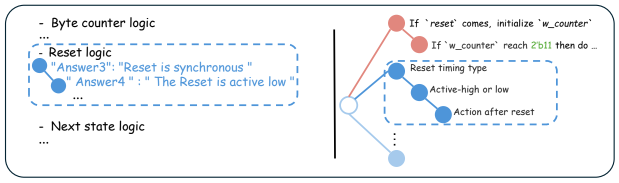
I. Introduction

Recently, the Large Language Model (LLM) has become one of the most effective methods for automatically generating Verilog code. Previous works suggest that the base model's capability to generate Hardware Description Languages (HDLs) can be amplified by fine-tuning [1] and prompt Technique [2] [3]. Despite these advances, existing generation methods pay little attention to the inconsistency between the sequential description of requirements and the parallel description of HDLs.

Sequential behavioral-level descriptions have proven to be an inaccurate prompt for generating Verilog. Figure 1 (a) illustrates a CoT approach to Verilog code, where the sequential input prompt is shown on the left, with different colors representing descriptions of various modules. Despite the generated code adhering to CoT steps, it resulted in an error because the same signal/register was assigned in different blocks. To address the inconsistency between CoT and Verilog, in this paper, we apply the concept of Tree of Thoughts (ToT) to enhance CoT's reasoning path and propose the VToT method. The right side of Figure 1 (b) illustrates the structure of VToT, where each node in the tree represents a specific design constraint, and each branch corresponds to different modules of the target code. The left side of Figure 1 (b) shows the structure of a VToT prompt, which, compared to the direct CoT method, provides relevant "thoughts" for each module and incorporates relationships within and between modules.


...
2. If `reset` is received, initialize `w_counter`.
...
...
5. When in `WAIT` state, update `w_counter`
6. When in `WAIT` state, if `w_counter == 3`
...
module 1
module 2
...
...
error(10028): Can't resolve multiple constant for ...

(a) CoT Prompt for Verilog


- Byte counter logic
...
- Reset logic
"Answer3": "Reset is synchronous "
" Answer4 " : " The Reset is active low "
...
- Next state logic
...
If `reset` comes, initialize `w_counter`
If `w_counter` reach 2'b11 then do ...
Reset timing type
Active-high or low
Action after reset

(b) VToT Prompt for Verilog

Fig. 1: Comparison of the structure of CoT and VToT prompts.

II. VToT Method

The VToT process consists of three primary stages: the reasoning phase, the logic verification phase, and the final generation phase.

The reasoning phase aims to extract design-related constraints from confusing descriptions. VToT refines the functional descriptions into specific constraints about the target codes and transposes these constraints into a tree structure. In the "constrains tree", as shown in the right of Figure 1 (b), the sequential description, as the root node, is split into branches according to the functional modules, and the specific constraints are put as tree nodes. Within the same branch,

This work was supported in part by the National Natural Science Foundation of China under Grants 62372461, 62032001 and 62203457, in part by National Defense Key Laboratory Project of the State Administration of Science and Industry for National Defense under Grant WDZC20235250112 and in part by the Key Laboratory of Advanced Microprocessor Chips and Systems.(corresponding author: Lei Wang)

[1]Lei Wang is with Defense Innovation Institute, AMS. E-mail:leiwang@nudt.edu.cn.

constraints are formed as parent-child or sibling, depending on whether they are logically sequential or branching.

In the logic verification phase, the inferences drawn from the 'constraint Tree' are Verified, and potential errors are identified through dynamic checking and rectification mechanisms. Occasionally, there may be multiple branches containing errors. VToT abstracts the entire design process into distinct modules, allowing the user to correct each erroneous module independently, without affecting the results of other parts.

In the final generation phase, a hierarchical code format provides the LLM with constraints and their relationships. In this format, each independent branch from the " constraints tree" is converted into annotations that correspond to a specific module.

III. Experiment

To verify the effectiveness of the VToT method, We adopt two representative Verilog generation benchmarks, VerilogEval [4] and RTLLM [5].

TABLE I: Comparison of VToT and solutions. The improvement of the base model by the VToT method is marked by (delta). The data in the figure represents the pass rate (

Model Type	Evaluated Model	VerilogEval-Human		RTLLM pass@5	
		pass@1	pass@5	Syntax	Func
Domain-specific Model †	ChipNeMo-13B	22.4	N/A	N/A	N/A
	VerilogEval-13B	28.8	45.5	N/A	N/A
	RTLCoder-7B	36.7	45.5	96.6	48.3
	RTLCoder-6.7B	41.6	50.1	93.1	48.3
General Model	Claude 3	34.4	48.3	93.1	55.2
	GPT-4	43.5	55.8	**100**	65.5
Prompt Technique †	GPT-4 + SP	N/A	N/A	90	63
	Assistant + GPT-3.5	34.4	51.3	93.1	48.3
	Assistant + GPT-4	50.5	62.8	100	**75.9**
Ours	VToT + Claude 3	39.5	54.0	93.1	55.2
	Improvement (Δ)*	(+5.1)	(+5.7)	(+ 0)	(+ 0)
	VToT + GPT-4	**52.4**	**65.4**	**100**	**75.9**
	Improvement (Δ)*	(+8.9)	(+9.6)	(+ 0)	(+10.4)

∗ The improvement is directly compared to the performance of the same model without employing the VToT framework.
† The results are referenced from the original papers respectly.

The experimental results derived from VToT are presented in Tables 1 and 2, leading to two primary conclusions as follows.

1) VToT reported state-of-the-art (SOTA) performance. Table I presents our comparison with existing Verilog generation solutions. VToT demonstrates a significant improvement over different base models. Specifically, when applied to Claude 3, VToT increases the pass @1 rate on VerilogEval-Human to 39. 5% and the pass@5 rate to 54.0%, representing improvements of 5.1% and 5.7%, respectively. For GPT-4, VToT achieves state-of-the-art results on all benchmarks, with improvements of 8. 9% (pass@1), 9. 6% (pass@5) on VerilogEval-Human, and 10. 4% on RTLLM. These results indicate that VToT is highly effective in enhancing the accuracy and reliability of Verilog code generation across different base models.

2) FSM is Effectively Addressed. VToT's structured constraints provide clear guidance for FSM (Finite State Machine)

TABLE II. Passed Tasks on VerilogEval(pass@1).

Task id	GPT4	GPT4+VToT	Task id	GPT4	GPT4+VToT
fsm1	✓	✓	ece241_2014_q6a	✗	✗
fsm1s	✓	✓	ece241_2014_q5b	✗	✗
fsm2	✓	✓	2014_q3fsm	✗	✗
fsm2s	✓	✓	2014_q3bfsm	✓	✓
fsm3comb	✓	✓	2014_q3c	✗	✓
fsm3onehot	✓	✓	m2014_q6b	✗	✓
fsm3	✓	✓	m2014_q6c	✗	✓
fsm3s	✓	✓	m2014_q6	✗	✓
ece241_2013_q4	✗	✗	2012_q2fsm	✗	✓
lemmings1	✗	✗	2012_q2b	✓	✓
lemmings2	✗	✗	2013_q2afsm	✗	✓
lemmings3	✗	✗	2013_q2bfsm	✗	✗
lemmings4	✗	✗	review2015_count1k	✗	✓
fsm_onehot	✓	✓	review2015_shiftcount	✗	✓
fsm_ps2	✗	✓	review2015_fsmseq	✓	✓
fsm_ps2data	✗	✗	review2015_fsmshift	✗	✗
fsm_serial	✗	✓	review2015_fsm	✗	✗
fsm_serialdata	✗	✓	review2015_fancytimer	✗	✗
fsm_hdlc	✗	✓	review2015_fsm_onehot	✗	✗
ece241_2013_q8	✗	✓			

The **Task ID** includes 39 FSM tasks out of a total of 156 questions in the VerilogEval-Human.

problems, which are inherently complex and require precise logical flow. Table II shows the pass@1 rates for the FSM-related questions using both the base model and the VToT-enhanced model. Specifically, VToT enabled the base model to correctly solve 11 of 39 FSM questions which could not be handled previously. This improvement is attributed to VToT's ability to break down the problem into manageable subtasks, allowing the model to focus on specific aspects of the FSM logic. Consequently, VToT significantly enhances the model's performance on FSM-related tasks by providing a structured approach to problem solving.

IV. Conclusion

In this paper, we propose the Verilog-Tree-of-Thoughts (VToT), a prompt-based approach that employs the Tree of Thoughts (ToT) concept in Verilog code generation. Our comprehensive evaluation demonstrates that VToT achieves state-of-the-art (SOTA) performance across all benchmarks, significantly enhancing base models, and outperforming existing prompt-based methods. Supported by experimental evidence, We anticipate that VToT's performance can be further enhanced by integrating domain-specific knowledge and structured information. In conclusion, the VToT represents an advancement in the field of automatic HDL generation.

References

[1] K. Chang, K. Wang, N. Yang, Y. Wang, D. Jin, W. Zhu, Z. Chen, C. Li, H. Yan, Y. Zhou, Z. Zhao, Y. Cheng, Y. Pan, Y. Liu, M. Wang, S. Liang, Y. Han, H. Li, and X. Li, "Data is all you need: Finetuning LLMs for Chip Design via an Automated design-data augmentation framework," Jul. 2024.

[2] Y.-D. Tsai, M. Liu, and H. Ren, "RTLFixer: Automatically Fixing RTL Syntax Errors with Large Language Models," May 2024.

[3] H. Huang, Z. Lin, Z. Wang, X. Chen, K. Ding, and J. Zhao, "Towards LLM-Powered Verilog RTL Assistant: Self-Verification and Self-Correction," May 2024.

[4] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "VerilogEval: Evaluating Large Language Models for Verilog Code Generation," Dec. 2023.

[5] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "RTLLM: An Open-Source Benchmark for Design RTL Generation with Large Language Model," Nov. 2023.