FCP-LLM: Functional Coverpoint Plan Generation Using LLM in Early Design Verification Stage

Zhuofan Lin[1,2], Zixian Guo[1,2], Chao Wang[1], Ruixin Zheng[1], Yuxin Ji[1],
Yang Wei Lim[2], Yuhang Zhang[1], Fakhrul Zaman Rokhani[2,*], and Yongfu Li[1,*]
1. Department of Micro-Nano Electronics, Shanghai Jiao Tong University, China
2. Department of Computer and Communication Systems Engineering, University Putra Malaysia, Malaysia
*Email: fzr@upm.edu.my, yongfu.li@sjtu.edu.cn

***Abstract*—During the design process of digital integrated circuits, high-quality coverpoint plan plays a crucial role, as it guides the verification process. This challenging task is accomplished by experienced engineers who consume a massive amount of time to extract critical information from design specifications. In this work, we present the FCP-LLM framework that is designed to understand both design specifications and Hardware Description Language (HDL) code to focus on extracting critical edges of designs' ports. It consists of three main components: CP-Generator for creating coverpoint plans, CP-Reviewer for refining the plans, and SV-Coder for generating SystemVerilog code. It has been evaluated with a series of benchmark circuits, comprising 74X-Series circuits and custom digital designs. Our experimental results show that FCP-LLM outperforms the baseline work with an average improvement of exceeding 300%, demonstrating a better balance between effectiveness and redundancy in coverpoint plan generation.**

***Index Terms*—Design Verification, Functional Coverpoint, Large Language Model**

I. Introduction

As the complexity of digital IC designs explodes, over 50% of the total design cost is now attributed to design verification in the design cycle [1]. Testbench is designed for driving stimulus signals into RTL designs and inspecting outputs to validate the designs. The progress of this verification is measured by a predefined metric, *coverage*, which not only evaluates the quality of the testbench but also indicates the completion of the verification process.

The coverage can be categorized into three types: *code coverage*, *assertion coverage*, and *functional coverage* [2]. *Code coverage* indicates if the structural elements of design such as branches, state transitions, and code lines are tested [3]. Therefore, 100% code coverage is an inherent property of the design. *Assertion coverage* and *functional coverage* focus on specific stimulus signals as per the design specification [4]. In particular, *Assertion coverage* examines the relations between design signals, while *functional coverage* checks if design implementation matches design intent [5], [6]. Both techniques are applied using SystemVerilog code and integrated into the verification environment. To formulate a reference of 100% functional coverage, defining high-quality and low-redundancy coverpoint plan is an important process.

This work is supported by the National Science Foundation of China under Grant No. 62350610271 and No. 6230413. Fakhrul Zaman Rokhani and Yongfu Li are the corresponding authors of this work.

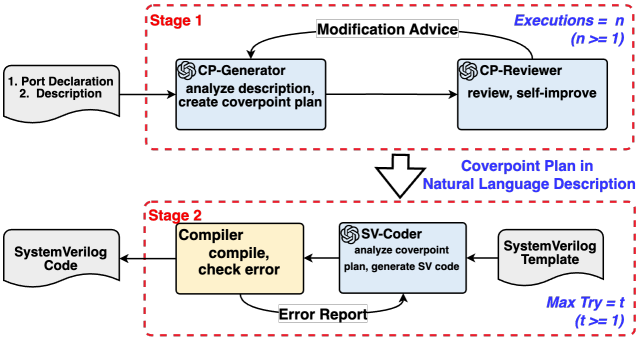

Fig. 1. Functional Coverpoint Plan Generation, FCP-LLM Workflow

Design of coverpoint plans is time-consuming coding process. With the rapid development of Large Language Model (LLM), it has shown impressive capability in comprehension and inference, which produces versatility of studies on hardware task [7]–[11]. For design verification, recent works have demonstrated the significant potential in LLM, emphasizing their strengths in comprehension and inference tasks [12]–[16]. For code coverage, LLMs have been applied to analyze RTL structures and interpret coverage reports to generate stimuli [12]. In the case of assertion-based coverage, LLMs assist in coding SystemVerilog Assertions (SVA) from both human-written descriptions and HDL code [13], [14]. LLMs also improve the efficiency of stimulus generation in achieving functional coverage convergence compared to traditional constrained-random testing [15].

Motivated by these works, we propose FCP-LLM to leverage LLM's capability in understanding and writing predefined functional coverpoint plan in the early design verification stage as shown in Fig. 1. To facilitate evaluation, we created a benchmark suite comprising digital circuit designs and their corresponding descriptions, with target coverpoint plans as a golden reference. Additionally, we established a metric to assess the quality of the generated functional coverpoint plans, enabling an objective evaluation of the outputs produced by the LLMs.

The rest of this article is organized as follows. Section II explains the proposed framework in detail. Section III provides experimental results in the benchmark suite. Section IV concludes the work and reveals future directions.

II. Details of FCP-LLM Methodology

FCP-LLM is an LLM prompt-based framework to enhance functional coverpoint plan generating in the early design verification stage. As shown in Fig. 1, the framework operates in two stages, both driven by different prompt modules. *Stage 1* works on coverpoint plan generation and self-refinement, focusing on extracting critical edges from natural language specifications and a portion of HDL code. Coding is omitted at this stage to minimize token usage. *Stage 2* handles the generation of SystemVerilog code by LLM with the optimized prompt modules and corrects syntax errors using feedback from the compiler.

The framework consists of three prompt modules: (a) *CP-Generator* iteratively generates a coverpoint plan in natural language for ports declaration of the target design; (b) *CP-Reviewer* provides recommendations to CP-Generator and improves the generated coverpoint plan; and (c) *SV-Coder* converts the natural language coverpoint plan into SystemVerilog code for easy integration. Together, these modules form an efficient workflow for generating coverpoint plan for critical edges.

A. Coverpoint Plan Generator

To precisely extract critical functional coverpoint plan for designs, it is crucial that our CP-Generator can interpret the design specifications in both HDL code and natural language. However, LLM models may produce hallucinations when processing complex specifications and HDL code due to limited SoC design knowledge in their training datasets [17]. Thus, developing an optimized prompt module that ensures concise yet precise input is a key challenge.

In this work, we design the prompt structure as illustrated in Fig. 2. In manual coverpoint plan design, the design specification serves as the golden reference. Following this principle, we provide the full design specification to the CP-Generator. Since the development of the coverpoint plan is part of the design verification stage, where HDL code correctness is not assured. Therefore, relying on incorrect HDL code can lead to potential errors of critical edges. To eliminate this problem, the CP-Generator should only uses the port declaration of the design as a reference. Our method reduces token usage and improves precision by eliminating the potential influence of erroneous HDL code. Hence, the CP-Generator can produce a coverpoint plan in natural language, as shown in Fig. 2. Same as the prompt in CP-Generator, the response is formatted into port name and values.

B. Coverpoint Plan Reviewer

As the quality of initial outputs from LLM is not always guaranteed to be accurate, we introduce CP-Reviewer to refine the results generated by CP-Generator so that it allows a self-reflection thought process in the LLM. As shown in Fig. 2, the prompts are designed based on empirical studies and focus on two key aspects [2]:

- Coverpoints should target corner cases without attempting to traverse all possible values of the bus size.

Prompt (CP-Generator)

A ***Name_of_Design*** has the following specification: **Initial Prompts**

Natural Language Description of RTL Design

This DUT has the following ports:

Ports Declaration in HDL Code

Analyze design description and ports declaration, define critical coverpoint plan with description to examine the functions of the design. Show the coverpoint plan with only the port name and corresponding values with no explanation.

Chat History with Suggestion from CP-Reviwer **Subsequent Prompts**

Refine the coverpoint plan based on the suggestion. Show the coverpoint plan with only the port name and corresponding values

Prompt (CP-Reviewer)

Chat History with Result From CP-Generator

Analyze the generated coverpoint plan for the design in these 2 aspects:

1. Coverpoints should target corner cases without attempting to traverse all possible values of the bus size.
2. Coverpoints should focus on key features relevant to functional intents, including essential values that activate different functions.

Coverpoint Plan Deisign Principle

Make suggestions to improve the coverpoint plan in bullet points with 50 words for each.

Response

Refined Coverpoint Plan: **Example of Coverpoint Plan for ISCAS74283 4-Bit Fast Adder**

- **C0**: {0, 1}
- **A**: {4'b0000, 4'b1111}
- **B**: {4'b0000, 4'b1111}
- **S**: {4'b0000, 4'b1111}
- **C4**: {0, 1}

Fig. 2. Prompts of Stage 1 in FCP-LLM, and response example of ISCAS74283 4-bit fast adder. This stage creates a comprehensive natural language coverpoint plan.

- Coverpoints should focus on key features relevant to functional intents, including essential values that activate different functions.

As LLMs can sometimes stubbornly and persistently repeat certain behaviors [18], we define a threshold value, *Executions*, defaulted to trigger CP-Reviewer once during the workflow to improve the generated coverpoint plan.

C. SystemVerilog Coder

As shown in Fig. 3, the V-Coder generates SystemVerilog code based on the natural language coverpoint plan from Stage 1. Since LLMs can produce varying styles of SystemVerilog code, a predefined template is used to ensure the generated code is suitable for plug-in usage and analysis. Additionally, a SystemVerilog compiler is integrated into Stage 2 to provide SV-Coder with error reports, enhancing the LLM's ability to produce syntax-error-free code. Similar to Stage 1, we define a threshold, *Max Try*, to limit the number of iterations in this stage, addressing potential LLM stubbornness.

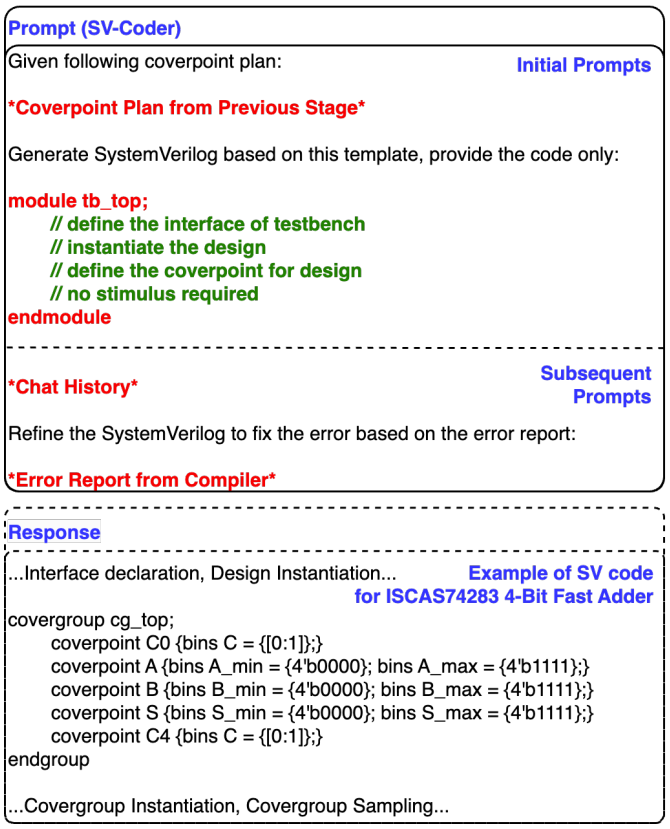

Fig. 3. Prompts of Stage 2 in FCP-LLM and example response for ISCAS74283 4-bit fast adder. This stage generates a syntax-error-free SystemVerilog code based on coverpoint plan.

D. Evaluation Metric

The design of coverpoint plan is a high-level system task in the front end of the SoC design flow. As a result, there are few established evaluation metrics for this problem. To formulate the problem of evaluation, we first denote a well-defined natural language specification as S, and each port in the design as p_i. Notably, this coverpoint plan considers input and output ports while excluding the internal wires or registers. The process of the LLM-driven framework, denoted as Gen, involves analyzing the specification S to generate a comprehensive set of coverpoints for each port p_i of the design. Meanwhile, the process of defining a set of target coverpoints for a design is symbolized as Tar. Therefore, the framework aims to generate coverpoints that closely match the target coverpoints as a golden reference, expressed as:

$$\forall p_i \in S, Gen(S, p_i) \cap Tar(S, p_i) \rightarrow Tar(S, p_i) \quad (1)$$

Based on the defined target, we introduce a metric, Quality of Coverpoint Plan (QCP), to quantitatively assess the generation process as follows:

$$QCP = \frac{Gen(S, p_i) \cap Tar(S, p_i)}{Tar(S, p_i)} \times \left(1 - \frac{\Delta Gen(S, p_i)}{Gen(S, p_i)}\right) \quad (2)$$

where $\Delta Gen(S, p_i)$ represents a non-intersection portion of generated with target set of coverpoints. This formula evaluates both the accuracy and redundancy of the generated coverpoint plan in comparison to the golden reference. By penalizing the incorrect or redundant coverpoints of the plan, QCP ensures that the generated set is not only comprehensive but also precise, and closely aligned with the intended functionality.

TABLE I
CIRCUITS CHARACTERISTICS IN BENCHMARK SUITE

Circuits	Number of Ports		Max Bus Size
	Inputs	Outputs	
74X-Series Circuits			
74283	3	2	4
74181	5	5	4
74L85	5	3	4
Customized Circuits			
Arithmetic Unit	3	1	8
1-Hot Decoder	2	1	16

This table shows circuit characteristics including number of input/output ports and maximum bus size.

III. EXPERIMENTAL RESULTS

For each design in our benchmark suite, we provide the design specification, Verilog code, and target coverpoints. The evaluation criteria, including syntax and QCP, are used to compare the generation quality of our framework against a baseline using the same LLM. The LLMs employed in our experiments are OpenAI's *GPT-4o* and *o1-mini* [19].

A. Benchmark Suite

This study focuses on generating critical coverpoint plan for the ports of digital circuit designs. We collected and designed a benchmark suite based on frequently used modules in digital circuits. Meanwhile, we added the design specifications for these circuits, focusing on the explanation of functionalities for each input and output port, which helped LLM to understand the design intent of the given design.

Table I shows the details of the benchmark suite, including three 74X-Series circuits [20] and two custom circuits. The functions of these circuits are as follows:

1) *74283*: A 4-bit fast full adder based on a carry look-ahead module, which is more efficient than ripple-carry adder.
2) *74181*: A 4-bit arithmetic logic unit which allows 16 logic and 16 arithmetic functions.
3) *74L85*: A 4-bit magnitude comparator which enables cascading by three dedicate inputs, allowing results of one comparator to be fed into the next.
4) *Arithmetic Unit*: A 4-bit arithmetic unit which contain addition, multiplication, subtraction and division.
5) *1-Hot Decoder*: A 16-bit 1-hot decoder which converts 1-hot codes to 4-bit binary representation.

B. Framework Evaluation

Table II presents the results of coverpoint plan generation by both the baseline model and FCP-LLM by using GPT-4o

TABLE II
EVALUATION AND COMPARISON OF COVERPOINT PLAN GENERATION BETWEEN BASELINE AND FCP-LLM

Circuit	Redundant / Effective Coverpoints				Target Coverpoints	Quality of Generated Coverpoint Plan			
	GPT-4o		o1-mini			GPT-4o		o1-mini	
	Baseline	FCP-LLM	Baseline	FCP-LLM		Baseline	FCP-LLM	Baseline	FCP-LLM
74283	39/13	9/13	39/13	4/13	13	25.00%	**58.21%**	25.00%	**76.47%**
74181	39/37	3/25	39/37	3/26	37	48.68%	**60.33%**	48.68%	**62.29%**
74L85	20/17	3/18	26/18	3/18	18	44.71%	**87.10%**	40.91%	**84.38%**
Arithmetic Unit	101/9	2/12	269/12	10/13	15	5.25%	**66.40%**	3.41%	**50.79%**
1-Hot Decoder	24/8	3/8	24/8	2/8	8	25.00%	**70.59%**	25.26%	**77.42%**

- In this experiment, *Executions* and *Max Try* of FCP-LLM were set to 1 and 3 separately.
- The experiments for each method have been ran three times.

and o1-mini. *Executions* and *Max Try* were set to 1 and 3 separately for acquiring the best results from our experiments, which minimized the hallucination effects. The experiment was conducted in a zero-shot model, without fine-tuning or retrieval techniques. For a fair comparison, both models were provided the same information which includes the design specification, port declarations, and a SystemVerilog template. Additionally, the generated code was immediately suitable for plug-and-play usage, facilitating rapid evaluation of stimuli.

First, FCP-LLM consistently generated syntax-error-free SystemVerilog code throughout the experiments, while the baseline model produced only three error-free outputs across 30 runs, highlighting the robust coding capabilities of FCP-LLM. In coverpoint plan generation, both FCP-LLM and the baseline approached the target coverpoints effectively as shown in Table II. However, the baseline produced much more redundant coverpoints compared to FCP-LLM. The baseline's QCP metrics fell below 50% for five benchmark circuits, whereas FCP-LLM substantially optimized the results, achieving an average improvement exceeding 300%.

As shown in Table II, the baseline model sometimes outperformed than FCP-LLM in generating effective coverpoints for the two custom designs, primarily because it exhaustively traversed the full bus size for each port in all designs. This exhaustive approach, however, led to more redundant coverpoints. In contrast, FCP-LLM produced fewer redundant coverpoints. According to the QCP metric, FCP-LLM outperformed the baseline, indicating better coverpoint plan generation quality by striking a balance between effectiveness and redundancy. To be more specific, the benchmark suite used in this study has a maximum 16-bit bus size, significantly smaller than the bus sizes of modern SoC designs. As the bus size increases, the testing space for each port expands exponentially. Therefore, FCP-LLM is expected to show even more efficient coverpoint plan generation and convergence in early design verification stages for larger designs.

IV. CONCLUSION

The FCP-LLM framework shows promising results in automating functional coverpoint plan generation for early design verification stages. Experimental results on the benchmark circuits demonstrate that FCP-LLM optimized the generation compared to the baseline. The results reveal that FCP-LLM enable LLM to analyze design and generate reasonable coverpoint plan for it. However, coverpoints are not the only consideration in complete functional coverage plan. The future research directions include generating cross coverage and conditional coverage by LLM. To further improve the methodology, the techniques like retrieval-augmented generation (RAG) and model context protocol (MCP) can be considered. RAG can improve the intelligence of LLMs, and MCP may help LLMs understand the tool chain better. As LLM technology advances, framework like FCP-LLM are expected to play an increasingly important role in streamlining and enhancing the design verification process for complex SoC designs, though continued refinement and expansion of the approach will be necessary.

REFERENCES

[1] U. Farooq and H. Mehrez, "Pre-silicon verification using multi-FPGA platforms: A review," *Journal of Electronic Testing*, vol. 37, no. 1, pp. 7–24, 2021.

[2] C. Spear and G. Tumbush, *SystemVerilog for Verification: A Guide to Learning the Testbench Language Features*. Boston, MA: Springer US, 2012.

[3] M. Jammigumpula and P. K Shah, "A new mechanism in functional coverage to ensure end to end scenarios," in *2020 IEEE International Conference for Innovation in Technology (INOCON)*, 2020, pp. 1–8.

[4] H. Witharana, Y. Lyu, S. Charles, and P. Mishra, "A Survey on Assertion-based Hardware Verification," *ACM Comput. Surv.*, vol. 54, no. 11s, Sep. 2022.

[5] S. El-Ashry and K. Salah, "A functional coverage approach for direct testing: An industrial IP as a case study," in *IEEE EUROCON 2015 - International Conference on Computer as a Tool (EUROCON)*, 2015, pp. 1–6.

[6] A. Yao, J. Wu, and Z. Zhang, "Functional Coverage Driven Verification for TAU-MVBC," in *2010 Fifth International Conference on Internet Computing for Science and Engineering*, 2010, pp. 89–92.

[7] M. Liu, T.-D. Ene, R. Kirby *et al.*, "ChipNeMo: Domain-Adapted LLMs for Chip Design," 2024, arXiv:2311.00176.

[8] S. Liu, W. Fang, Y. Lu, Q. Zhang, H. Zhang, and Z. Xie, "RTLCoder: Outperforming GPT-3.5 in Design RTL Generation with Our Open-Source Dataset and Lightweight Solution," in *2024 IEEE International Workshop on LLM-Aided Design*. IEEE, 2024.

[9] M. Li, W. Fang, Q. Zhang, and Z. Xie, "SpecLLM: Exploring Generation and Review of VLSI Design Specification with Large Language Model," 2024, arXiv:2401.13266.

[10] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "VerilogEval: Evaluating Large Language Models for Verilog Code Generation," in *2023 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023.

[11] Y.-D. Tsai, M. Liu, and H. Ren, "RTLFixer: Automatically Fixing RTL Syntax Errors with Large Language Models," 2024, arXiv:2311.16543.

[12] R. Ma, Y. Yang, Z. Liu, J. Zhang, M. Li, J. Huang, and G. Luo, "VerilogReader: LLM-Aided Hardware Test Generation," in *2024 IEEE LLM Aided Design Workshop (LAD)*, 2024, pp. 1–5.
[13] W. Fang, M. Li, M. Li, Z. Yan, S. Liu, H. Zhang, and Z. Xie, "AssertLLM: Generating and Evaluating Hardware Verification Assertions from Design Specifications via Multi-LLMs," 2024, arXiv:2402.00386.
[14] R. Kande, H. Pearce, B. Tan, B. Dolan-Gavitt, S. Thakur, R. Karri, and J. Rajendran, "(Security) Assertions by Large Language Models," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 4374–4389, 2024.
[15] Z. Zhang, G. Chadwick, H. McNally, Y. Zhao, and R. Mullins, "LLM4DV: Using Large Language Models for Hardware Test Stimuli Generation," 2023, arXiv:2310.04535.
[16] C. Wang, Y. Zhang, W. Lu, J. Huang, M. Yin, Y. Zhang, Z. Li, and Y. Li, "D2D-GPT: Leveraging Incremental Learning GPT for Seamless Design Rule Conversion Across EDA Tools," in *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, 2024, pp. 1–5.
[17] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung, "Survey of Hallucination in Natural Language Generation," *ACM Comput. Surv.*, vol. 55, no. 12, Mar. 2023.
[18] J. Xie, K. Zhang, J. Chen, R. Lou, and Y. Su, "Adaptive Chameleon or Stubborn Sloth: Revealing the Behavior of Large Language Models in Knowledge Conflicts," 2024, arXiv:2305.13300.
[19] OpenAI, J. Achiam, S. Adler *et al.*, "GPT-4 Technical Report," 2024, arXiv:2303.08774.
[20] "ISCAS_hlm." [Online]. Available: https://web.eecs.umich.edu/~jhayes/iscas.restore/benchmark.html