THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU)

AssertLLM: Generating Hardware Verification Assertions from Design Specifications via Multi-LLMs

Zhiyuan Yan[1*], Wenji Fang[2*], Mengming Li[2], Min Li[3], Shang Liu[2], Zhiyao Xie[2†], Hongce Zhang[1†]

[1]Hong Kong University of Science and Technology (Guangzhou)
[2]Hong Kong University of Science and Technology
[3]Huawei Technologies Co., Ltd

Functional Verification

- Check if **implementation** complies with **specification**
  - Implementation (RTL)
    - Hardware description language (HDL)
  - Specification (Assertion)
    - Ad-hoc properties
    - Security properties

    ……

```
always @(posedge clk) begin
        if (!resetn) begin
                mem_la_firstword_reg <= 0;
                last_mem_valid <= 0;
        end else begin
                if (!last_mem_valid)
                        mem_la_firstword_reg <= mem_la_firstword;
                last_mem_valid <= mem_valid && !mem_ready;
        end
end
```

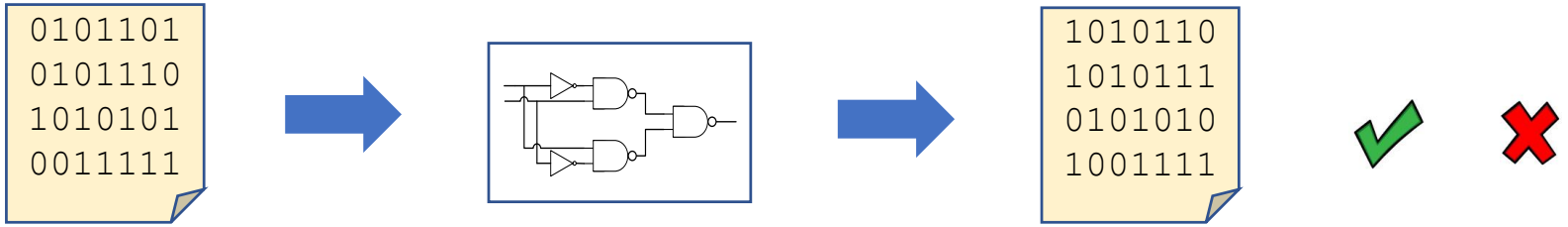
RTL for hardware design

```
assert property (
   ready => ##[3:5] valid ##1  finish);
```

SystemVerilog Assertion

Functional Verification

- Simulation



Test vector Simulation Simulation results

Hardware Formal Verification

- RTL complies with predefined assertions?

```
always @(posedge clk) begin
        if (!resetn) begin
                mem_la_firstword_reg <= 0;
                last_mem_valid <= 0;
        end else begin
                if (!last_mem_valid)
                        mem_la_firstword_reg <= mem_la_firstword;
                last_mem_valid <= mem_valid && !mem_ready;
        end
end
```

Behavioral Model (RTL)

```
req |->
  ##[1:4]
    resp
```

Property (e.g., SystemVerilog Assertions)

Model Checking

Result
- a counterexample, or
- a proof

Assertion for Verification

- System Verilog Assertion (SVA)
  - **Need understanding of**
    - System Verilog syntax
    - Verification techniques
    - Hardware design
  - **Account for**
    - Completeness
    - Coverage

```
assert property (
   ready => ##[3:5] valid ##1  finish);
```

SystemVerilog Assertion

Automatic Assertion Generation

- **Dynamic mining**
  - Simulation traces + design constraint analysis
  - No golden reference, rely on RTL designs under verification(DUT)
- **Static analysis of specifications**
  - Predefined templates → Still need a few human efforts
  - ML-based methods
    - Traditional natural language processing (NLP)
    - Large language model (LLM)

Previous ML-based Assertion Generation

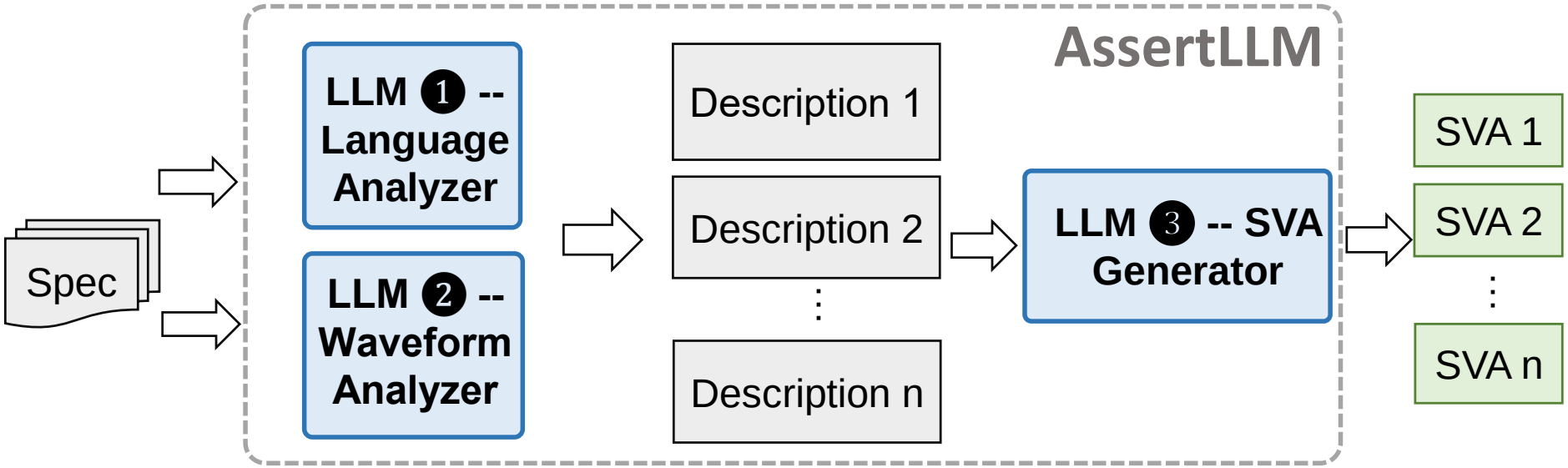
- **LLM-based[1]**
  - Human-written specification sentences (from comments)
  - No golden reference
- **NLP-based[2]**
  - Human-extracted specification sentences
  - Hard to generalize across distinct grammar
  - Waveform diagrams in specification are ignored
  - **Can we directly generate assertions from SPEC?**

1. Kande et al. LLM-Assisted Generation of Hardware Assertions, *arXiv* (2023).
2. Frederiksen et al. Automated assertion generation from natural language specifications, *ITC* (2020).

Challenges of Generating Assertion from SPEC

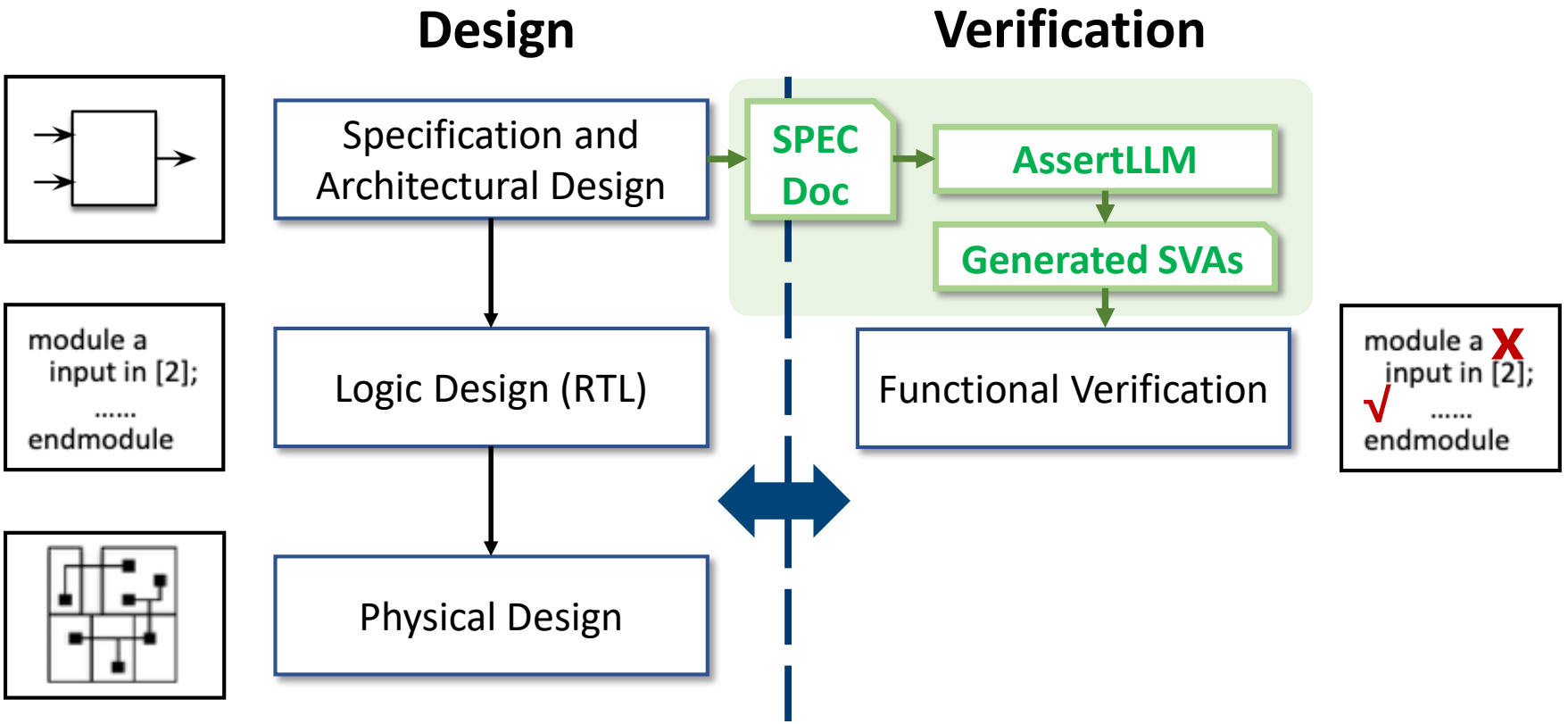
- **Natural language SPEC is not uniformly structured**
- **Translating NL to assertion is a non-trivial task**
- **Waveform diagrams need to be tackled**

Our Method: AssertLLM



- **Generating Assertions from Design Specifications by incorporating three LLMs**

VLSI Design and Verification Flow



Natural Language Extraction

- **LLM1: Natural Language Analyzer**
  - **Input**
    - Complete SPEC document file
  - **Customized LLM**
    - Custom instructions
  - **Output**
    - Template-based extraction
      - Signal name
      - Description
      - Related signals

[Custom Instructions for SPEC Analyzer]

- Please act as a professional VLSI specification analyzer.
- Firstly, I'll upload the the **design specification file** (in PDF format).
- Each time I ask a question, please read and analyze the specification PDF file carefully, and answer the question **only based on the uploaded files**, don't use any content outside the file.
- When I ask for information on a **[xxx] signal**, please **extract all the information related to this signal** in the spec file, note that the extracted information should be **suitable for the SystemVerilog Assertions (SVAs) generation**, and output all the information in the following format:

[*Signal name*]: name in SPEC
[*Description*]: including but not limited to
    [*Definition*]: such as bit-width, signal type (in/out/register), etc
    [*Functionality*]: all the function information illustrated in SPEC
    [*Interconnection*]: the relationship with all other signals
    [*Additional information*]: all other related information
[*Related Signals*]: names of all other signals mentioned in the *Description* part

Natural Language Extraction

- **Example**
  - **Structural SPEC for each signal**

[Prompt for SPEC Analyzer]

- [Upload the original SPEC file] Here is the design specification file, please analyze it carefully.
- [For each signal] Please extract all the information related to the signal control register (CTR) from the spec file.

[Response from SPEC Analyzer]

The Control Register (CTR) in the I2C-Master Core specification has the following characteristics:

1. [Signal Name]: In Spec - Control Register (CTR).
2. [Description]:
   2.1. [Definition]:
      - Bit-width: 8 bits
      - Signal Type: Register (Access Type: RW - Read/Write)
   2.2. [Functionality]:
      - Bit 7: I2C core enable bit. When set to '1', the core is enabled, and when set to '0', the core is disabled.
      - Bit 6: I2C core interrupt enable bit. When set to '1', interrupts are enabled, and when set to '0', interrupts are disabled.
      - Bits 5:0: Reserved. These bits are read/write but are reserved for future use.
   2.3. [Interconnection]: This register directly controls the operation of the I2C core, enabling or disabling it and controlling interrupt behavior.
   2.4. [Additional Information]:
      - Reset Value for the Control Register is 0x00
3. [Related Signals]:
      - EN: Controls the enable state of the I2C core.
      - IEN: Controls the interrupt enable state of the I2C core.

Waveform Information Extraction

- **LLM2: Waveform Analyzer**
  - **Input**
    - Waveform Diagram in SPEC
  - **Customized LLM**
    - Multi-step
    - Custom instructions
  - **Output**
    - Extracted descriptions

Waveform Information Extraction

- **Step 1: Template Generation**
  - **Input**
    - Customized prompt
  - **Output**
    - Templates

[Prompt for Template Generation]

You are an expert in digital hardware design at the register-transfer level (RTL). Please generate a set of prompt templates. We will use these prompt templates to summarize the behaviors that can be observed in hardware waveforms, and these observations will be further utilized to generate assertions in SystemVerilog format. As a consequence, the generated templates need to describe all logic behaviors at the RTL level for a variable or across different variables. And it needs to be easily transferred to the SystemVerilog assertions by human engineers. Here are the requirements for the answer:

1. Please keep templates as brief and clear as possible.
2. Please number the templates.

[Response from Template Generation]

Here are some prompt templates for summarizing observed behaviors in hardware waveforms at the RTL level:

1.If <signal> is high, then <variable> must be low in the next cycle.
2.When <condition> occurs, <variable1> should be equal to <variable2>.

…

…

18.<variable> should remain stable for <number> cycles after <event>.
19.<variable1> must be greater than <variable2> after <event>.
20.<variable> must be equal to <expression> when <condition> is true.

Waveform Information Extraction

- **Step 2: Description Generation**
  - **Input**
    - Customized instruction
    - Generated templates
    - Waveform diagram
  - **Output**
    - Waveform descriptions

[Response from Description Generation]

1. When byte_controller.dcnt is 3'b000, byte_controller.cnt_done is set to high.

…

…

4. byte_controller.cnt_done remains high once byte_controller.dcnt is 3'b000.

1.When ien rises, irq_flag should be high in the next cycle.

…

…

4.irq_flag should be high only if ien has been true for 1 cycle.

[Prompt for Description Generation]

You are an expert in digital hardware design and verification at the Register-transfer level (RTL). Your task is to analyze the behaviors from the given hardware waveform for a variable or across different variables. Please use different serial numbers to represent different behaviors from the waveform.

Furthermore, we provide the potential templates for describing behaviors. There is no need for you to use all templates. You only need to select some templates. Also, you can revise the templates if they can better describe the behavior in the waveform:

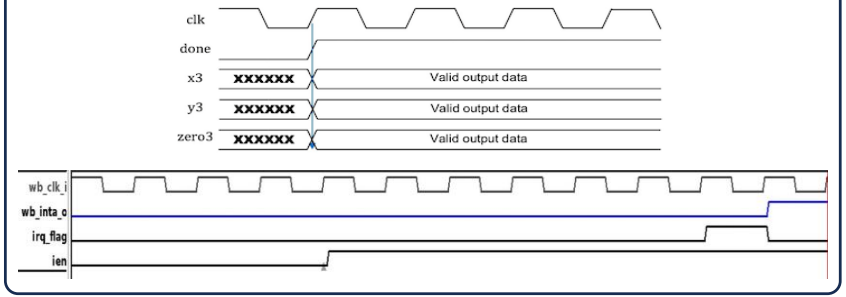
- [Upload the generated templates from Template Generator]

…

…

I will provide the waveform you need to analyze in the following blocks. Please do use the English to answer the question.

- [Upload the waveform diagram]



Automatic Assertion Generation

- **LLM3: SVA Generator**
  - **Input**
    - Extracted information
    - Architecture diagram(if available)
  - **Customized LLM**
    - Retrieval Augmented Generation
    - Custom instructions
  - **Output**
    - Multi-type SVAs
      - Width + Connectivity + Function

[Custom Instructions for SVA Generator]

- Please act as a professional VLSI verification engineer. You can refer to the uploaded **knowledge base** about the SystemVerilog assertion (SVA) and formal verification techniques.
- Firstly, I'll upload an workflow **diagram** of the design architecture.
- Then every time I provide the **specification of a signal** of the design, please write **all the corresponding SVAs** based on the defined Verilog signals that benefit both the RTL design and verification processes. Please note that don't generate signals that do not exist in the signal declaration Verilog file.
- Please make sure that the generated SVAs have **no syntax error**, and strictly **follow the function** of the given specification/description.
- The generated SVAs should include but not be limited to the following **types**:

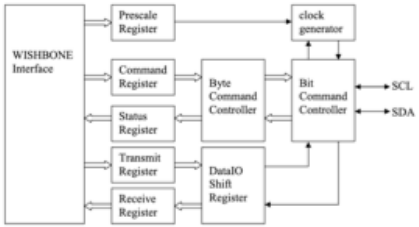
[*width*]: Check the signal width using $bits operator
[*connectivity*]: Check if the signal can be correctly exercised and also the value propagation among all connected signals.
[*function*]: Check if the function defined in the specification is implemented as expected.

Automatic Assertion Generation

- **Example**

[Prompt for SVA Generator]

- [Upload the architecture diagram] Here is the overall diagram workflow of the design architecture.



- [For each signal, upload the extracted specification and signal mapping] Please generate SVAs for the signal control register (CTR).

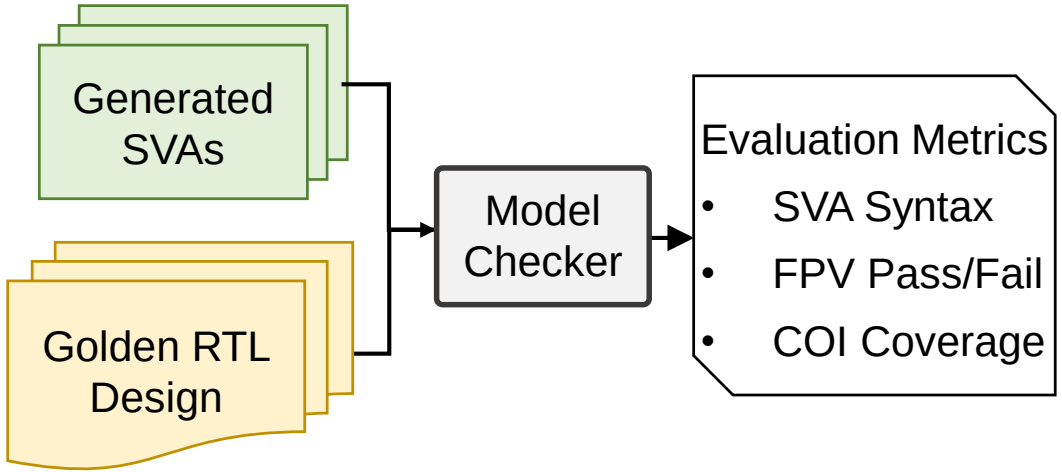
[Response from SVA Generator]

Based on the provided specification for the control register (CTR), we can write SVAs to validate its behavior. Let's break down the SVAs according to the types you specified:

```
[width]
assert property (@(posedge wb_clk_i) $bits(ctr) == 8);
[connectivity]
// Check that the I2C core enable bit (Bit 7) properly controls the EN signal
assert property (@(posedge wb_clk_i) (ctr[7] == 1'b1) |-> (core_en == 1'b1));
assert property (@(posedge wb_clk_i) (ctr[7] == 1'b0) |-> (core_en == 1'b0));
// Check that the I2C core interrupt enable bit (Bit 6) properly controls the IEN signal
assert property (@(posedge wb_clk_i) (ctr[6] == 1'b1) |-> (ien == 1'b1));
assert property (@(posedge wb_clk_i) (ctr[6] == 1'b0) |-> (ien == 1'b0));
[function]
// Ensure that changes to the CTR register propagate correctly to EN and IEN
assert property (@(posedge wb_clk_i) $past(ctr[7]) != ctr[7] |-> ##1 (core_en == ctr[7]));
assert property (@(posedge wb_clk_i) $past(ctr[6]) != ctr[6] |-> ##1 (ien == ctr[6]));
// Assert that the I2C core is disabled on reset
assert property (@(posedge wb_clk_i) (wb_rst_i == 1'b1) |-> (ctr == 8'h00));
// Assert that the I2C core can be enabled
assert property (@(posedge wb_clk_i) (wb_rst_i == 1'b0 && ctr[7] == 1'b1) |-> (core_en == 1'b1));
// Assert that interrupts can be enabled
assert property (@(posedge wb_clk_i) (wb_rst_i == 1'b0 && ctr[6] == 1'b1) |-> (ien == 1'b1));
```

Generated SVA Evaluation

- **Generated SVA + Golden RTL Design**
  - RTL is error-free and strictly implemented according to SPEC
- **Model Checker**
  - Formal verification (FPV)
- **Evaluation Metrics**
  - SVA syntax
  - FPV pass/fail
  - COI coverage



Experimental Results

- **Experimental Setup**
  - SPEC document: PDF format
  - Model checker: Cadence JasperGold
  - LLM: GPT3.5/4o from OpenAI
- **Evaluation Metrics**
  - # generated SVAs
  - # syntax-correct SVAs
  - # FPV-passed SVAs
  - COI Coverage
- **Test Design**
  - I2C with 23 signals (17 IO ports and 6 architecture-level registers)

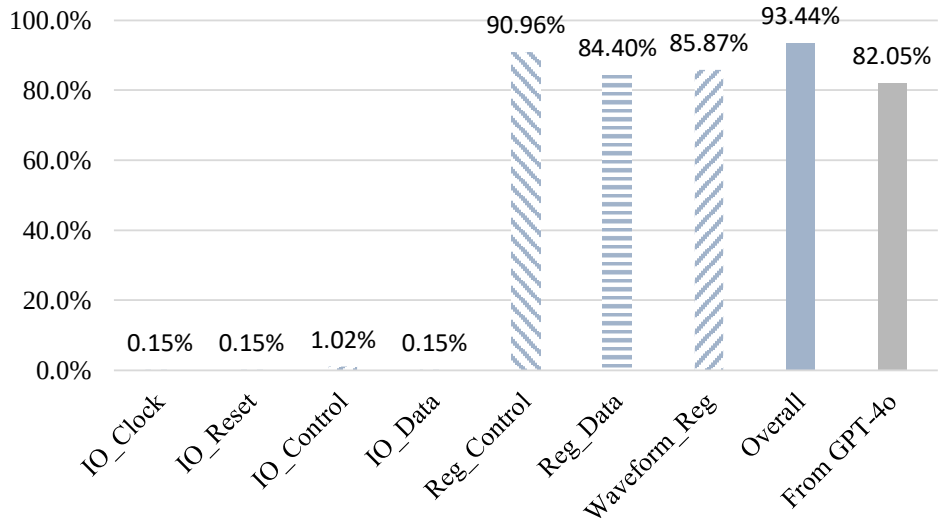
Experimental Results

- All bit-width SVAs are correct
- Few connectivity and function SVAs contain errors
- 86% SVAs are both syntactically and functionally correct

			AssertLLM				GPT-4o	GPT-3.5
From natual language	Signal Type		Assertion Evaluation (#. Generated/#. Syntax Correct/#. FPV Pass)					
			Width	Conectivity	Function	Signal Total	Function	
	IO (17)	Clock (1)	1/1/1	/		1/1/1	3/1/0	Can not handle the original specification files.
		Reset (2)	2/2/2	/		2/2/2	6/2/0	
		Control (3)	3/3/3	4/4/1	/	7/7/4	9/3/0	
		Data (11)	11/11/11	/		11/11/11	33/11/0	
	Reg (6)	Control (2)	2/2/2	10/10/9	13/13/13	25/25/24	6/2/2	
		Data (4)	4/4/4	/	6/6/4	10/10/8	14/4/4	
From waveform			/		9/9/6	9/9/6	4/4/2	
Design Total			23/23/23★	14/14/10	28/28/23	65/65/56	75/27/8	
			100%/100%†	100%/71%	100%/82%	100%/86%	36%/11%	

Experimental Results

- All IO-related SVAs have lower COI coverage
- Register-related and Waveform-related SVAs have high COI
- SVAs generated by AssertLLM achieve 93.44% of coverage



Experimental Results

- Can generate SVAs with high quality on the other designs

Design	SVA Evaluation			
	AssertLLM		GPT-4o	
	Total Correctness	Total Coverage	Total Correctness	Total Coverage
I2C	65/65/56 100%/86%	93%	75/27/8 36%/11%	82%
ECG	22/22/20 100%/91%	99%	11/7/0 64%/0%	0%
Pairing	15/15/14 100%/93%	100%	12/8/1 67%/8%	0%
Average	100%/90%	97%	56%/6%	27%

Conclusion

- AssertLLM: generating assertions from complete design specifications
- Utilize multi-step strategies and LLMs to generate SVAs
- Rely on golden reference specification, rather than DUT
- Can generate SVAs from waveform diagram
- Quantitative evaluation method based on golden RTL

THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY
(GUANGZHOU)

Thank You!

Cone of Influence (COI) Coverage

- COI coverage measures the percentage of design logic that is structurally connected to the assertions.