A Leap Forward in Formal Verification Using Generative AI

Dr. Shahid Ikram, Marvell Semiconductor
Sean Safarpour, Synopsys
Alex Chang, Synopsys

SPONSORED BY CEDA IEEE Council on Electronic Design Automation SIGDA acm

Motivation

Challenges in Formal Verification

- **Steep Learning Curve**: FV is one of the most challenging areas in verification.
- **Complexity in FPV**: Formal Property Verification (FPV) is the most demanding aspect of FV.
- **Core of FPV**
  - **Property Generation**: Involves creating assertions, assumptions, and covers.
    - **Design Insights**: Designers and verifiers have insights into design behavior.
    - **Expression Complexity**: Accurately expressing insights in System Verilog Assertions (SVA) is complex.
  - **Requirements for Property Generation**
    - **Deep Design Understanding**: Requires knowledge of design features and signal names.
    - **SVA Syntax and Semantics**: Solid grasp of SVA syntax and semantics is essential.
    - **Design Guidelines:** The team legacy, macros, and methodology.

Generative AI Applications

- **Generative AI** is a versatile tool transforming diverse domains, fostering creativity, and automating complex workflows.
- Applications:
  - Content Generation
  - Code Generation
  - Data Augmentation
  - Content Transformation
  - Summarization
  - Personalized Experience
  - Personal Assistance
  - Training Bot
  - Reference Bot
  - Interactive Explorations
  - Knowledge Discovery
  - Etc.

When content generation becomes trivial then Challenge lies in “Data Curation”

FV and GenAI

- Test plans:
- English language
- An item-by-item feature listing of the design-under-test(DUT)
- What is needed to be covered
- What is needed to be checked
- What are the constraints of the design

```
 9 assume Thershold1 will be greate than zero.
10 assume n_i stays constant
11 assume swoverride value stable
12 assume SEL stays stable
13 assume once SEL goes high it will stay high
14 assume once SEL signal is asserted it will stay asserted
15 assume CURR_SAMPLE_SYMB stable
16 Assertions:
17 Devload output will be deasserted after the reset
18 cbusy flag and devload_swoverride reset devload output
19 In the case of hyst disable, devload output stale value after first cbusy valid.
20 Hyst moving average has stale value after the first busy.
21 Check dvload_o for all cbusy along with all input combinations.
22 Check devload output when hyst is disabled
23 Check devload output is 2 when hyst is disabled.
24 Check devload output is 2 when hyst and SWoverride is disabled.
25 check devload output is devload_swoverride2 when devload_swoverride_i is enabled
```

Prompt

Create SVA properties for the provided RTL code and the section of Spec file selected.

Answer

These were the generated properties:

```
// Reason:This assertion checks that when hyst is disabled, devload output remains stale after the first cbusy valid.
property asrt_devload_output_stale_value_after_first_cbusy_valid;
  @(posedge clk) disable iff (!main_rst_clk_n)
  $rose(cbusy_valid_i) && (devload_swoverride_i == 0) && (hysten_i == 0) |-> (devload_o == 2'b00)[*6];
endproperty
assert_asrt_devload_output_stale_value_after_first_cbusy_valid:assert property (asrt_devload_output_stale_
```

Refine Results

LLM succeeded in generating the properties

System Prompt

Deduced from the RTL

Few Shot refinement

Took FV person half day to figure it out

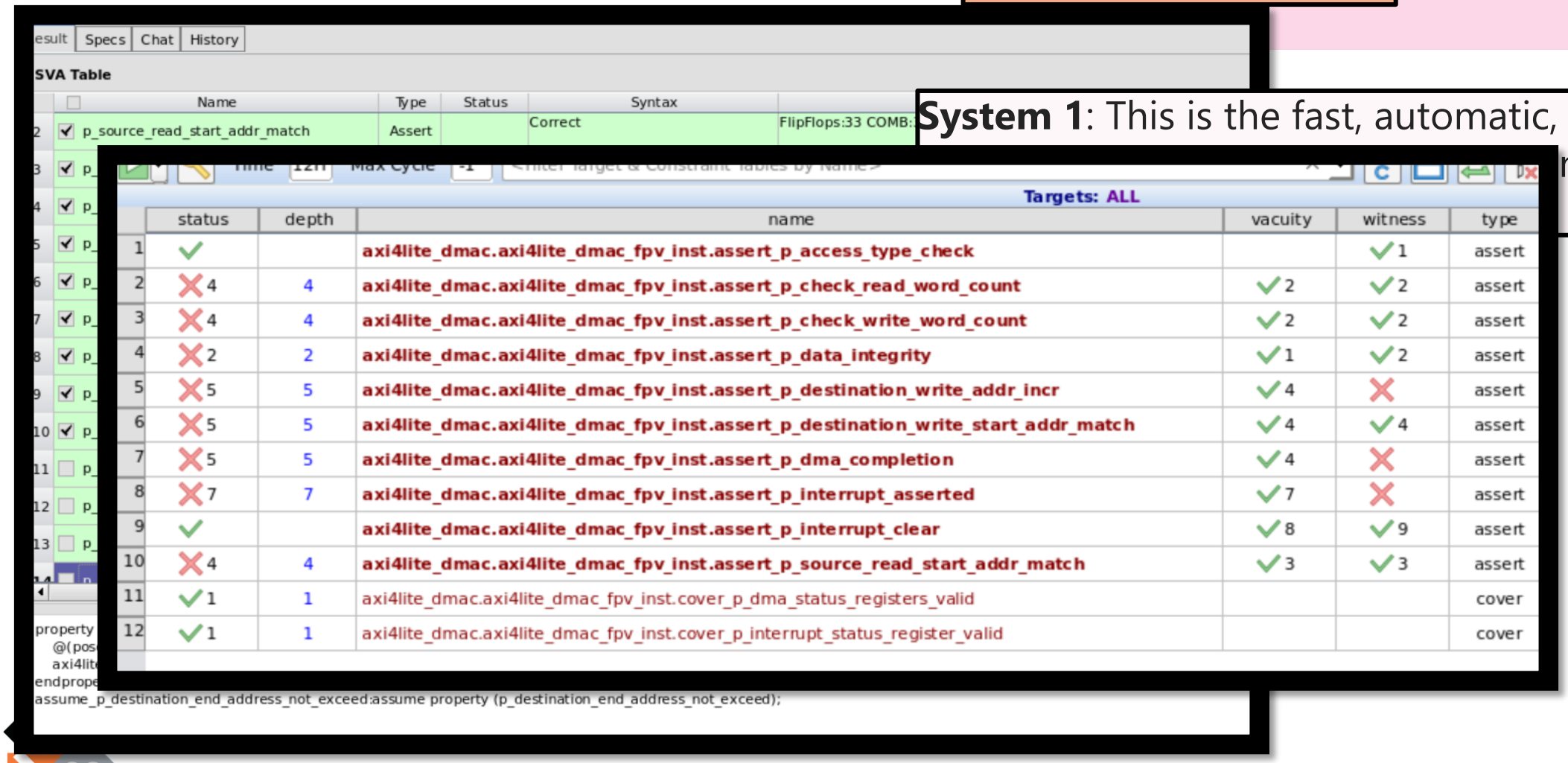
- Testbenches
  - Synthesizable Verilog code
  - Assertions, Covers, Assumptions
  - Auxiliary code

System I and System II

A Copilot

Thinking Fast and Slow
Daniel Kahneman

System 1: This is the fast, automatic, and intuitive
nd
instincts



Targets: ALL

	status	depth	name	vacuity	witness	type
1	✓		axi4lite_dmac.axi4lite_dmac_fpv_inst.assert_p_access_type_check		✓1	assert
2	✗4	4	axi4lite_dmac.axi4lite_dmac_fpv_inst.assert_p_check_read_word_count	✓2	✓2	assert
3	✗4	4	axi4lite_dmac.axi4lite_dmac_fpv_inst.assert_p_check_write_word_count	✓2	✓2	assert
4	✗2	2	axi4lite_dmac.axi4lite_dmac_fpv_inst.assert_p_data_integrity	✓1	✓2	assert
5	✗5	5	axi4lite_dmac.axi4lite_dmac_fpv_inst.assert_p_destination_write_addr_incr	✓4	✗	assert
6	✗5	5	axi4lite_dmac.axi4lite_dmac_fpv_inst.assert_p_destination_write_start_addr_match	✓4	✓4	assert
7	✗5	5	axi4lite_dmac.axi4lite_dmac_fpv_inst.assert_p_dma_completion	✓4	✗	assert
8	✗7	7	axi4lite_dmac.axi4lite_dmac_fpv_inst.assert_p_interrupt_asserted	✓7	✗	assert
9	✓		axi4lite_dmac.axi4lite_dmac_fpv_inst.assert_p_interrupt_clear	✓8	✓9	assert
10	✗4	4	axi4lite_dmac.axi4lite_dmac_fpv_inst.assert_p_source_read_start_addr_match	✓3	✓3	assert
11	✓1	1	axi4lite_dmac.axi4lite_dmac_fpv_inst.cover_p_dma_status_registers_valid			cover
12	✓1	1	axi4lite_dmac.axi4lite_dmac_fpv_inst.cover_p_interrupt_status_register_valid			cover

Chatting with Copilot

```
module arbmux_v1_0
 #( parameter  NR = 8, // number of requesters
    parameter  DWIDTH = 2  // data bits     )
(   input                       rst_n,
 input                        clk,
 input [NR-1:0]                  in_req,
 input [NR-1:0] [DWIDTH-1:0]    in_dat,
 input                        in_gnt,
 output [DWIDTH-1:0]             out_dat,
 output [NR-1:0]                 out_sel,
 output                       any   );
```

if only one requestor is valid, the select always matches that requestor

```
//Auxiliary logic:
logic [7:0] valid_requestors;
        always_comb begin
           valid_requestors = in_req;
        end
property p_only_one_requestor_is_valid;
        @(posedge clk) disable iff (!rst_n)
        (valid_requestors == 1) |-> (out_sel == valid_requestors);
        endproperty
assert_prop_only_one_requestor_is_valid:assert property (p_only_one_requestor_is_valid);
```

Refined: The one request can be any bit not just bit 0

```
property p_only_one_requestor_is_valid;
   @(posedge clk) disable iff (!rst_n)
   ($onehot(valid_requestors)) |-> (out_sel == valid_requestors);
endproperty
assert_p_only_one_requestor_is_valid:assert_prop_only_one_requestor_is_valid: assert property (p_only_one_requestor_is_valid);
```

Customization

- Almost no quality code of SV and SVA is available in Public Domain.
- Every Chip Design team has developed its own design methodology and patterns.

Language	After dedup		After filters and decont.		Weight	Percentage
	Num. files	Volume (GB)	Num. files	Volume (GB)		
ada	31,291	0.30	30,934	0.26	0.26	0.034
agda	17,608	0.07	17,554	0.07	0.07	0.009
alloy	5,374	0.01	5,368	0.01	0.01	0.001
antlr	7,082	0.05	7,917	0.05	0.05	0.007
sql	994,019	12.22	975,420	11.09	11.09	1.446
stan	5,441	0.01	5,429	0.01	0.01	0.001
standard-ml	48,995	0.52	19,630	0.19	0.19	0.025
stata	31,282	0.41	24,208	0.33	0.33	0.043
systemverilog	46,915	0.41	46,270	0.39	0.39	0.051
tcl	50,579	0.40	49,335	0.35	0.35	0.046

Table 1: Overview of the training data for StarCoder. For the selected programming languages, we show the number of files and data volume after near-deduplication, as well as after filtering. See also Table 2.

Retrieval Augmented Generation(RAG)

- Retrieval augmented generation
  - Vector databases
- Semantic similarity
- Local, private, and secure
- Experts and legacy code to populate a local RAG
- Captures patterns and methodology
- Complex cases of NL(natural language) to SVA
- A reference guide for the novices

Benchmarking

- Use Case Driven
- Reference Designs
  - Previously verified designs with test plans and testbenches
- Human Review
  - RTL Designers,
  - FV, and DV teams
  - IT and AI teams
- AI Setup:
  - Embeddings:
    - Sentence Transformers: allenai-specter, all-MiniLM-L6-v2, all-mpnet-base-v2
  - LLMS
    - OpenAI, Llama3.1, Claude Sonnet 3.4
  - RAG
    - FAISS, ChromaDB

Common Modules FVIP

- **Target Audience**: DV/FV Engineers
- **Test**: Development of formally verified IP (FVIP) for common modules
- **Record**:
  - Reduction in human effort
  - Ease in formal testbench creation and full coverage

Unit-Level Testing

- **Target Audience**: RTL designers
- **Approach**: Test-driven design development
- **Record**:
  - Design size challenges
  - Full verification
  - Ease of use and quick turnaround time

Results

- A work in progress
- Common RTL designs in Verilog
- Few designs were previously formally verified

Challenge: 10 Common RTL modules in 10 workdays using Confluence documentation and Legacy RTL.

Property Complexity	RAG	Clock/Reset Accuracy	Syntax Accuracy	Functional Accuracy	Subjective Productivity (1-5)	Average Number of Edits
Simple	OFF	100%	93%	82%	5.00	1.00
	ON	100%	100%	95%	5.00	1.00
Medium	OFF	100%	70%	52%	3.35	3.9
	ON	100%	89%	70%	4.1	3.2
Complex	OFF	100%	35%	10%	1.83	5.22
	ON	100%	47%	25%	2.5	3.7

Formal For GenAI

- RTL Generation Vulnerabilities
- CWEs(Common Weakness Enumerations)
  - A community-developed list of common software and hardware weakness types that could have security ramifications.
- Study:
  - “All Artificial, Less Intelligence: GenAI through the Lens of Formal Verification”, Deepak Narayan Gadde, el.
- Purpose:
  - Evaluate safety and security of hardware designs generated by LLMs using Formal Verification
- Results:
  - ~60% of generated designs were vulnerable to CWEs
- Conclusions:
  - LLMs can generate hardware design skeletons, but manual review is essential(and FV can be a big help).

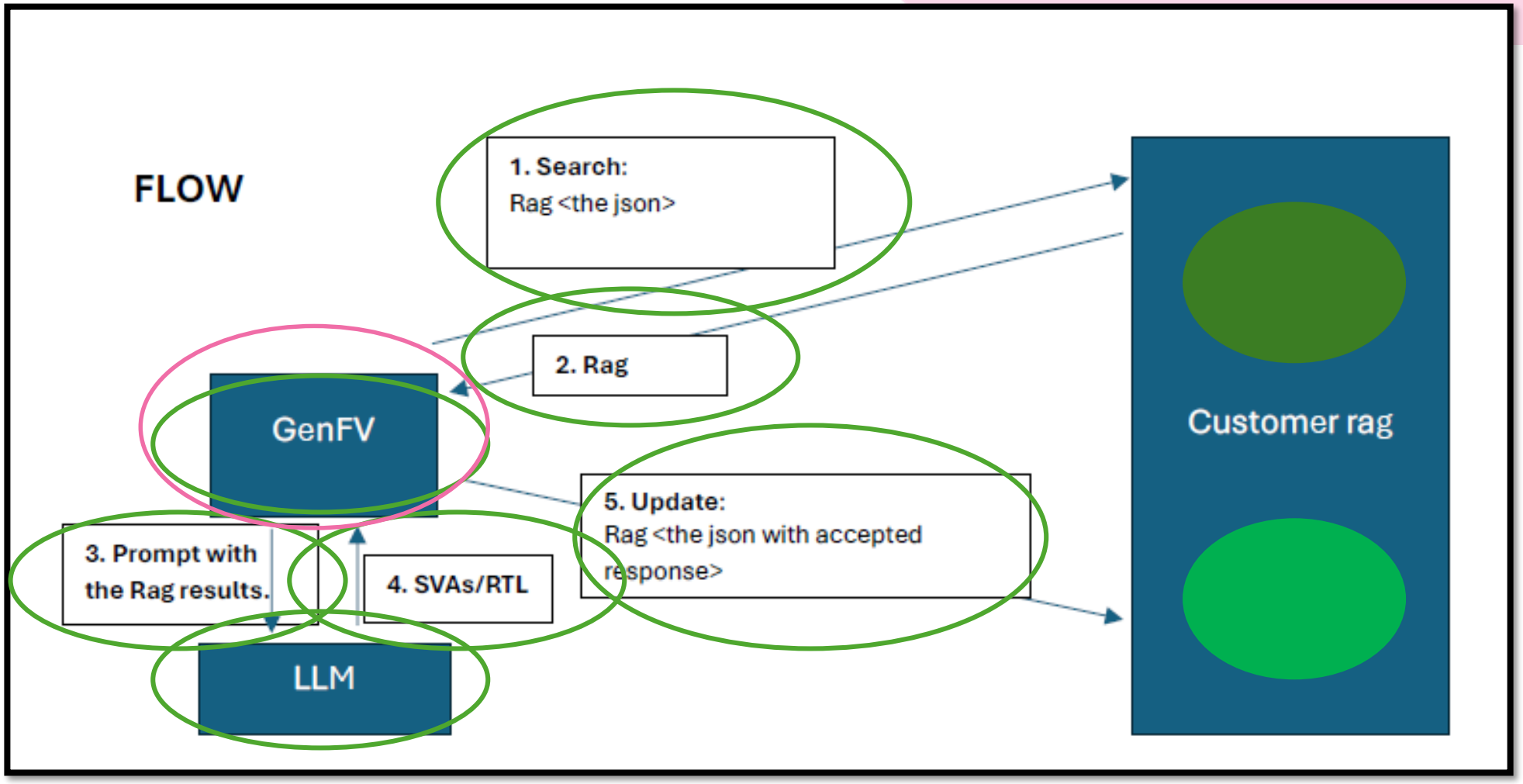
Conclusions

- We need to be very careful using GenAI for hardware design and verification:
  - We cannot patch hardware systems.
  - The cost of second and third spin is too big.
  - GenAI needs humans feedback for some time in the future.
- System II(human rational and reason) are and potentially remain key to the success.
  - We are fighting here with one of the most natural trait of nature i.e. “Nature finds the optimal path(not necessary the shortest path) and humans find the laziest path.”
- FV and GenAI are complementary to each other.
  - Ying and Yang

References

1. Shahid Ikram and Mark Eslinger," Demystifying Formal Testbenches: Tips, Tricks, and Recommendations", DVCON, San Jose, 2023.
2. Ed Cerny el. "SVA: The Power of Assertions in SystemVerilog", 2ns Edition, Springer, 2015, ISBN:3319071386
3. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* *ArXiv*. https://arxiv.org/abs/2005.11401
4. Li, R., Allal, L. B., Zi, Y., Muennighoff, N., Kocetkov, D., Mou, C., Marone, M., Akiki, C., Li, J., Chim, J., Liu, Q., Zheltonozhskii, E., Zhuo, T. Y., Wang, T., Dehaene, O., Davaadorj, M., Monteiro, J., Shliazhko, O., Gontier, N., . . . De Vries, H. (2023). *StarCoder: May the source be with you!* *ArXiv*. https://arxiv.org/abs/2305.06161
5. Zhou, Yi, "Prompt Design Patterns: Mastering the Art and Science of Prompt Engineering", 2023, **ASIN:**B0CL2CY26Z
6. Aman Kumar el. "Saarthi: The First AI Formal Verification Engineer", February 2025, San Jose, CA
7. Kahneman, D. "Thinking, fast and slow". Farrar, Straus and Giroux, 2011.
8. Mortimer J. Adler & Charles Van Doren, How to Read a Book (revised ed., 1972).
9. Deepak Narayan Gadde, el. All Artificial, Less Intelligence: GenAI through the Lens of Formal Verification", arxiv:2403.16750v1 [cs.AI] 24 mar 2024

RAG Based Customization



Peer Programming Cases

- Help us write a Checker/SVA
- Improve an existing Checker/SVA
- Help me understand an existing testing code
- Simplify an existing Checker/SVA
- Make an existing Checker/SVA more efficient
- Add debugging code for an existing Checker/SVA
- And much more…

Shahid Ikram

Distinguished Engineer
Marvell Semiconductor

MARVELL™