Assertion-Gen

Multi-LLM Assertion Generation for Formal Verification Applications

Ekansh Bhatnagar (836000172)

CSCE-689: Programming LLMs

Introduction

Why use LLMs in Formal Verification ?

Formal Verification is hard!

Formal Verification, though very useful is hard for verification engineers to implement due to it's differences with widely used simulation based verification.

Complex Designs require advanced convergence techniques.

Complexity for formal verification increases exponentially with register/logic counts. Advanced techniques are need to create formal friendly assertions.

Aggressive verification timelines for IP level verifications.

With yearly tapeouts, verification timelines are pretty aggressive. If possible, automation in formal verification testbenches will save crucial engineer hours.

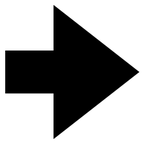
Challenges

Privacy and other concerns with Large Language Models

IP security and Confidentiality is paramount!

Lack of Open Source or available training data.

Formal methods require high reasoning effort due to mathematical nature.



Cloud-based Frontier Models

Frontier Models have good accuracy.

But, are not secure and private.

Local Open-Weight Models

Local Open-Weights Models are free and open source.

But, are not accurate with System Verilog Generation

Proposed Solution

Mix use of Local and Frontier Models

Stage 1

Specification to English Language Assertion Statements

Anthropic/
Claude-3-Opus

- Only specification.md is shared.
- No RTL specific data is shared to external servers.

Stage 2

English Language Assertion Statements to SystemVerilog Assertions

Anthropic/
Claude-3-Opus

- Only Stage 1 output is shared.
- No RTL specific data is shared to external servers.
- Syntactially correct but generic assertions are generated.

Stage 3

Refining raw assertions with RTL knowledge and create testbench with aux-code.

Meta/
Llama-7b-rtlcoder-verilog

- RTL code + Stage 2 output is shared.
- DSPy is used to optimize prompts and use examples for few-shot learning
- Feedback loop with Verilator (Open source verilog simulator) to lint and check for syntax issues in refined testbench.

Stage 4

Generating run scripts in TCL for JasperGold.

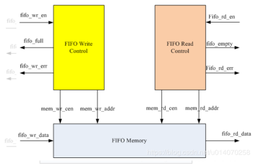
Meta/
Llama-7b-rtlcoder-verilog

- RTL Code + Stage 3 output is shared.
- DSPy is used to optimize prompts using TCL examples.
- Working JasperGold FPV tcl is generated for a complete solution.

Using DSPy for prompt optimization and few-shot learning

Experiments and Results

First-In-First-Out (FIFO) and Round Robin Arbiter



Source: programmingsought.com

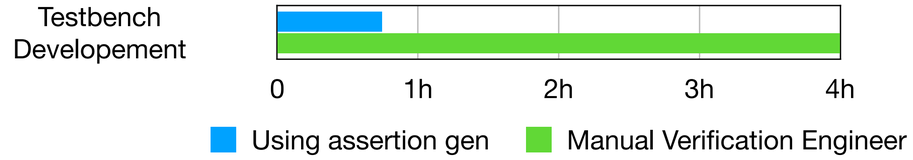
FIFO

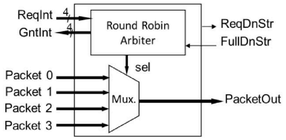
Generated Example:

```
assert property (@(posedge clk) (full
&& wr_en) |-> overflow);
```

Generated Assertions: 11

Human Review: 7 high quality, 3 medium quality, 1 low quality





Source: semanticscholar.org

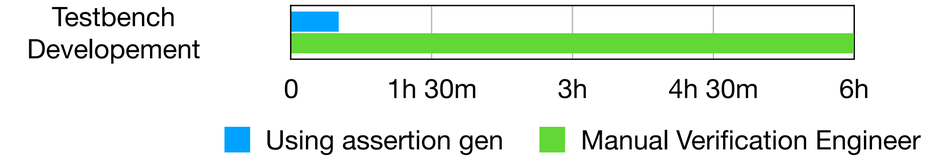
Round Robin Arbiter

Generated Example:

```
as__req_implies_gnt: assert property
(@(posedge clk) disable iff (!rst_n) (|
req) |-> (|gnt));
```

Generated Assertions: 6

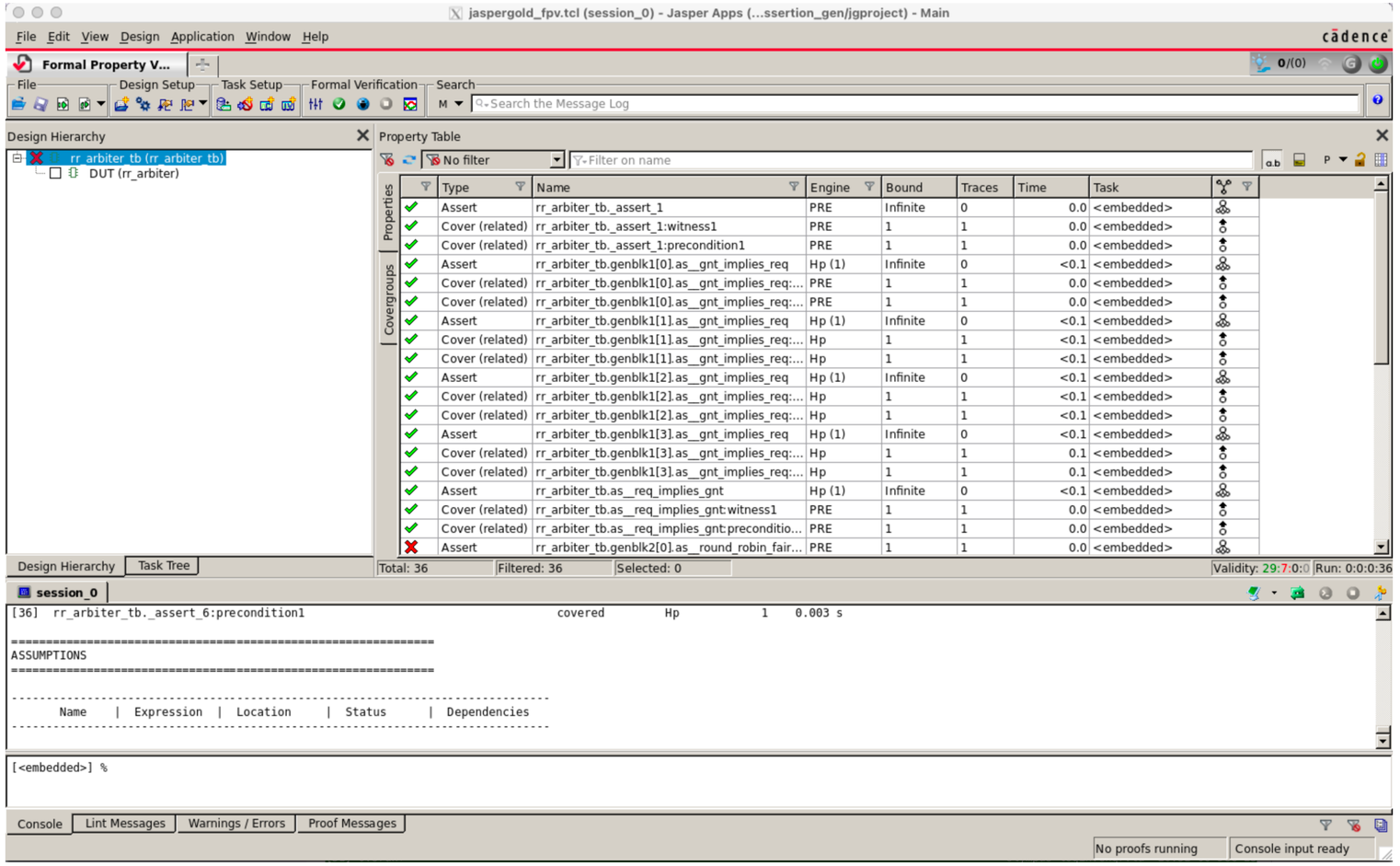
Human Review: 3 high quality, 2 medium quality, 1 low quality



JasperGold Outputs

Using TAMU - Olympus Server

JasperGold GUI



Demo