USING ARTIFICIAL INTELLIGENCE TO ASSIST FORMAL VERIFICATION

UCF | Burnett Honors College

Abstract

To ensure electronic hardware is free from design defects that would impact its quality, it is necessary to verify that it functions as intended. Formal Verification is one method of functional verification utilized in today's hardware design and development process. This method allows engineers to mathematically prove that the electronic hardware designs behave as expected. In practice, it takes time and experience to create Formal Verification assertions in SystemVerilog, a language commonly used in industry, since writing effective properties requires a deep understanding of the hardware. Large Language Models (LLMs) may provide support to engineers to alleviate this gap and assist them in verifying designs. Using two selected RISC-V processors, multiple large language models are used to generate SVA (SystemVerilog Assertions) that can be used to verify the design in a formal environment. Results from the models are compared to human-developed assertions and compared for clarity, completeness, and accuracy.

Motivation/Problem

Formal Verification is a specific subset of Functional Verification that involves an in-depth knowledge of verification, along with the RTL design files to fully verify a design. Compared to simulation-based verification, utilizing this is more complete and can lead to greater coverage of the design compared to directed or constrained random testing.

Because of these advantages, it would be ideal if a way could be created to allow engineers to perform this task more quickly and easily, and potentially allow those with less experience to be able to perform more trivial tasks.

Large Language Models are a hot topic and have been researched for their effectiveness in creating designs. As we can ingest our own data, we want to see if these models can easily use that information to assist an engineer perform formal verification without having to spend as much time to develop a complete understanding of the design.

Hypothesis & Methods

The current hypothesis is that with enough data ingested, LLMs can do basic logical equivalence. The current method is asking it to do Logical Equivalence with two designs based on the specification, after providing it with the designs and specifications. One of the designs has been previously verified to follow the specification, while the other has not, which will allow for the comparison of the unverified design to the specification to ensure compliance.

Dataset
- The dataset will include the intended specification in addition to the two designs being compared

Model/Tool
- Several fine-tined locally-hosted models are being evaluated for their output after being provided the relevant information.

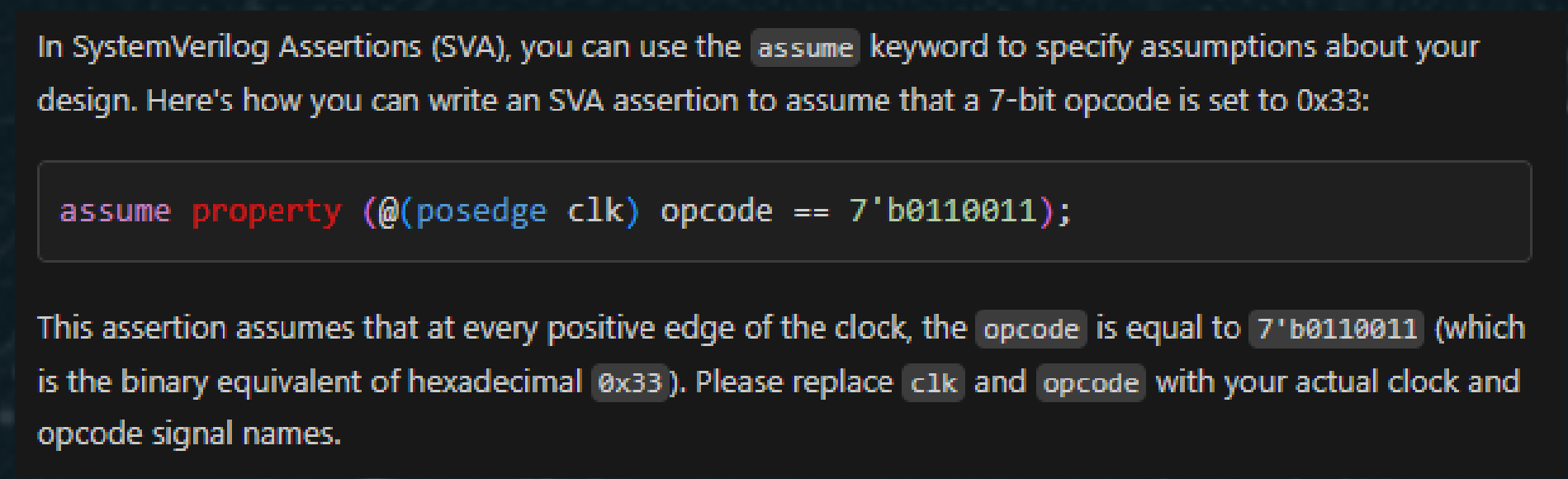
Evaluation
- Two RISC-V Algorithmic Logic Units for logical equivalence. One has been verified to a given specification while the other has not.

Prompting
- Ask the LLM if the designs are Logically Equivalent, along with any reasoning or proof behind its answer.

Result Recording/ Editing
- Record the data and attempt to either generate a better response if inaccurate or test with more complex data if sufficiently accurate.

Current Work & Results

Off-the-shelf models have proven to lack the context necessary to generate relevant assertion properties to verify a singular design.
The following are attempts at generating a simple statement to check a single value on two different models.

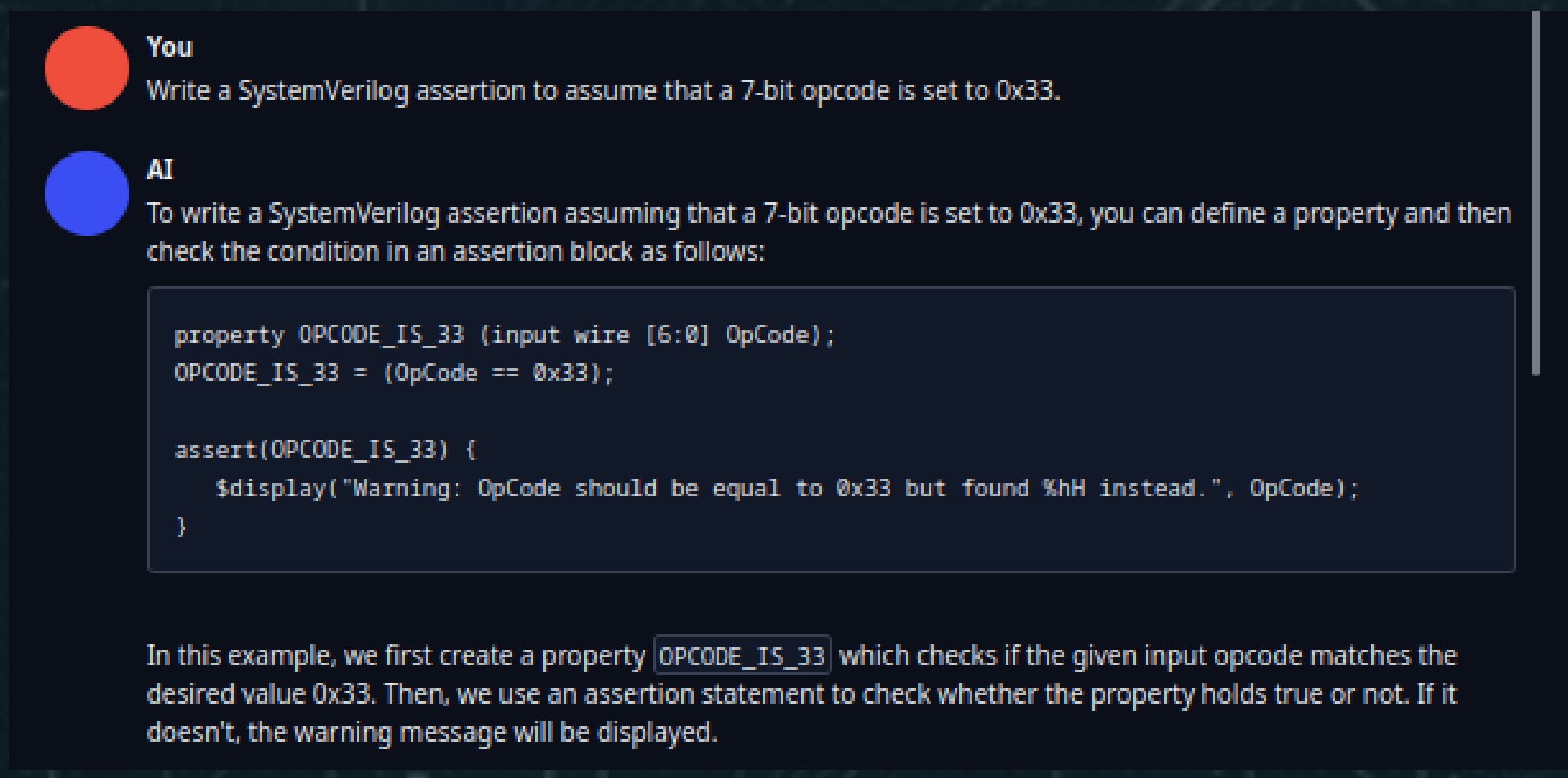


In SystemVerilog Assertions (SVA), you can use the `assume` keyword to specify assumptions about your design. Here's how you can write an SVA assertion to assume that a 7-bit opcode is set to 0x33:

```
assume property (@(posedge clk) opcode == 7'b0110011);
```

This assertion assumes that at every positive edge of the clock, the `opcode` is equal to `7'b0110011` (which is the binary equivalent of hexadecimal `0x33`). Please replace `clk` and `opcode` with your actual clock and opcode signal names.

Good Example – GitHub Copilot / GPT-4



You
Write a SystemVerilog assertion to assume that a 7-bit opcode is set to 0x33.

AI
To write a SystemVerilog assertion assuming that a 7-bit opcode is set to 0x33, you can define a property and then check the condition in an assertion block as follows:

```
property OPCODE_IS_33 (input wire [6:0] OpCode);
OPCODE_IS_33 = (OpCode == 0x33);

assert(OPCODE_IS_33) {
   $display("Warning: OpCode should be equal to 0x33 but found %hH instead.", OpCode);
}
```

In this example, we first create a property `OPCODE_IS_33` which checks if the given input opcode matches the desired value 0x33. Then, we use an assertion statement to check whether the property holds true or not. If it doesn't, the warning message will be displayed.

Bad Example – Fine-Tuned MistralAI

Now focusing on providing context to models in the form of existing Algorithmic Logic Unit Designs. This can allow for verifying a design directly to a specification document (Formal Property Verification) or to an existing known-good design (Logical Equivalence Testing). Currently focusing on Logical Equivalence Testing and comparing to human-generated responses.

Known-Good Design 1 ⟷ Unknown Design 2

Discussion

In general, most of the models that have been tested such far, especially local / open-source models, have lacked the context necessary to provide very useful assertion properties without additional inputs.

It appears, however, that it will be **possible to generate acceptable responses to the equivalence prompts** if relevant documentation such as specification documents and known-good verified designs are passed into the model.

Future Work

Future work will be on expanding the scope of how LLMs can assist engineers with performing Formal Verification. In addition, we will start to perform Formal Property Verification to directly verify a design from a specification after this work with Logical Equivalence Checking, in which more complex specification documents will likely be used to test how much the model can assist engineers with this task.

Acknowledgements

This work is funded by AMD through its Undergraduate Research Fellowship program at the University of Central Florida.

Lana Perkins lana.perkins@ucf.edu
Michael Castiglia michaelc@ucf.edu

DRACO LAB | www.ece.ucf.edu/DRACO
Director: Dr. Mike Borowczak
Dept. Of Electrical & Computer Engineering
College of Engineering and Computer Science
University of Central Florida

Student Scholars Symposium – March 26-27th 2024

AMD

DRACO