Prerequisites

1) Python 3.6 or newer
2) pip for installing dependencies
3) Riviera Pro (licensed installation required)

Installation

1) Set up a virtual environment (optional but recommended):
   python3 -m venv venv
   venv\Scripts\activate

2) Install the required python packages:
   pip3 install -r requirements.txt

3) Activate API Key:
   set OPENAI_API_KEY = "your API key"

4) Copy the design and testbench files from verilogeval_prompts_tbs directory and paste it in autochip_scripts directory.

5) Copy paths of both the files and paste it corresponding to ‘prompt’ and ‘testbench’ of config json respectively.

6) Go in the autochip_scripts directory in command prompt and run generate_verilog.py to start a compilation and simulation cycle.

Function of each file:

config.json: Configuration file for setting up model parameters and run settings (our main script for interacting with the model's parameters)

generate_verilog.py: Main execution script that manages Verilog code generation process using JSON configuration files to set model parameters and control the iterative generation and testing workflow.

rivierapro_backend.py: Script that implements a backend interface for the Riviera-PRO simulator, providing methods to initialize libraries, compile Verilog files, and run simulations with real-time output handling.

verilog_handling.py: Script that manages Verilog code generation and validation, including code cleaning, module extraction, compilation error analysis, and interaction with Riviera-PRO simulator through multiple iterations to generate working Verilog designs.

languagemodels.py: Script that implements various LLM interfaces (ChatGPT, Claude, Gemini, CodeLlama, RTLCoder) and handles their interactions with Verilog code generation, including a response class to process and evaluate generated code. For now, we are using only GPT while commenting out the rest of the LLMs.

verilogeval_prompts_tbs: Directory containing test prompts and testbenches from HDLbits (copy any design and testbench you want and paste it into autochip_scripts directory. Then, set its path in config.json)

test_outdir: Default output directory for generated files. It will generate a file for each iteration's generated output code. In addition, it will also create a log.

log.txt: Default log file recording model outputs and system messages

auto_create_verilog.py: Automates Verilog code generation and testing using LLMs, with iterative error correction based on testbench results.

config_handler.py: Manages configuration loading and command-line argument parsing for the Verilog generation tool, handling both JSON config files and CLI inputs with validation.

conversation.py: Class that manages chat messages between user and LLM, handling message storage, logging to file, and conversation history manipulation with methods for adding, removing, and retrieving messages.

parse_data.py: Script that processes log files from Verilog generation runs, extracting metrics like generation time, best iteration, and response rankings, then compiles the data into a CSV file for analysis.

parse_parameter_sweep: Script that processes parameter sweep experiment logs, extracting metrics from hierarchical directory structure organized by candidate depth, then compiling results into a CSV file for analysis.

utils.py: Script containing utility class that redirects standard output to a file, enabling logging of model outputs while maintaining original stdout functionality.

tools.py: Script defining an abstract base class for compilation tools, providing a template for implementing different hardware design compilation backends through a standardized interface.