# Using Large Language Models to Improve Open-Source IP: Automated Verilog to TL-Verilog Coversion



## Project Information

**Organization:** FOSSi Foundation

**Project:** Using AI to Improve Open-source IP

**Mentors:** Steve Hoover

**Repo:**https://github.com/stevehoover/conversion-to-TLV

**Duration:** 350 hours.

**Language/T**ools : Verilog, Python, TL-Verilog.

# Proposal

## Abstract

This project seeks to develop an intelligent framework for converting Verilog code into Transaction-Level Verilog (TL-Verilog) using Large Language Models (LLMs). The framework will facilitate a semi-automated conversion process, leveraging AI to identify and transform design patterns efficiently. To ensure functional equivalence between the original and converted code, formal verification will be integrated at each stage of the conversion. By combining AI-driven automation with rigorous verification, the project aims to streamline the transition from Verilog to TL-Verilog, reducing errors, improving efficiency, and enhancing the usability of open-source hardware IPs. This initiative will empower hardware designers to more easily adopt advanced, high-level design methodologies while maintaining the integrity and correctness of their IPs.

## Introduction

Hardware Description Languages (HDLs), particularly Verilog, are fundamental in digital circuit design. However, as designs grow in complexity, traditional Verilog becomes difficult to maintain and modify. Transaction-Level Verilog (TL-Verilog) offers a higher level of abstraction, improving code organization, maintainability, and verification. Despite its advantages, converting Verilog to TL-Verilog is a time-consuming and error-prone process. This project aims to automate the transformation using Large Language Models (LLMs), incorporating formal verification at each step to ensure functional equivalence between the original and converted code. By reducing manual intervention and potential errors, this approach will enhance the overall efficiency of design workflows. The goal is to make the transition to TL-Verilog more accessible, facilitating easier adoption of modern design methodologies in open-source hardware projects.

## Background

The hardware design community has built a vast repository of open-source Verilog code over the years. While these designs are functionally robust, they often suffer from common issues, including:

- Verbose and repetitive code structures
- Complex clock and reset logic

- Difficult-to-maintain pipeline stages
- Limited parameterization capabilities
- Challenging verification processes

Transaction-Level Verilog (TL-Verilog) addresses many of these challenges by introducing higher-level abstractions, streamlining pipeline specifications, and improving the overall structural organization of the code. While TL-Verilog enhances design maintainability and verification, the process of manually converting legacy Verilog code into TL-Verilog remains time-consuming and prone to errors. Automating this conversion could significantly improve workflow efficiency and reduce human error, paving the way for broader adoption of TL-Verilog in the open-source hardware community.

# Problem Statement

The challenge lies in creating an efficient, reliable method to convert existing Verilog code to TL-Verilog while ensuring:

- Functional equivalence between original and converted code
- Preservation of design intent and comments
- Optimal use of TL-Verilog features
- Systematic verification at each conversion step

# Proposed Solution

The project harnesses the power of Large Language Models (LLMs) to develop a semi-automated conversion framework that:

I. **Breaks down the conversion process into incremental, verifiable steps:**

- Systematic module-by-module conversion
- Step-by-step transformation of design elements
- Clear checkpoints for verification
- Rollback capabilities for error recovery

II. **Utilizes AI to assist in transforming Verilog code to TL-Verilog:**

- Intelligent code structure analysis
- Pattern recognition for common design elements
- Automated comment preservation and enhancement
- Suggestion of TL-Verilog optimizations

III. **Integrates formal verification to ensure functional correctness at each step:**

- Equivalence checking between versions
- Property verification maintenance
- Systematic test coverage analysis
- Verification report generation

IV. **Tracks conversion history to enable continuous improvements:**

- Detailed conversion step logging
- Success and failure pattern analysis
- Version control integration

V. **Provides comprehensive documentation and tutorials:**

- Example conversions repository
- Code snippets library
- Common design pattern catalog

VI. **Establishes contribution guidelines:**

- Code style standards
- Documentation requirements
- Testing procedures
- Review process
- Issue reporting templates

This approach not only simplifies the conversion process but also generates valuable training data for future AI-assisted hardware design tools. By combining human expertise with AI capabilities, we aim to make the transition from Verilog to TL-Verilog more efficient, accessible, and reliable.

# Project Objectives

**Technical Goals**

- Develop a reliable and efficient conversion framework for Verilog to TL-Verilog.
- Implement comprehensive verification methods to ensure functional correctness at each step.
- Build a library of verified conversions for both basic and advanced hardware designs.
- Document common conversion patterns and best practices for future use.

**Community Benefits**

- Improve the quality and maintainability of existing open-source hardware designs.
- Provide educational resources to promote the adoption of TL-Verilog in the hardware design community.
- Create valuable training data to enhance AI-assisted hardware design tools.
- Foster collaboration and knowledge-sharing between traditional Verilog and modern TL-Verilog communities.

**Long-term Impact**

- Accelerate the transition to modern, high-level HDL practices in hardware design
- Reduce the ongoing maintenance burden of complex hardware designs
- Enable more flexible and efficient design modifications and updates
- Enhance the verification process for digital hardware, ensuring higher reliability and correctness

# Technical Details:

## Core Components & Tools:

### Primary Script and Files:

#### Covert.py:

The convert.py script is the core engine driving the Verilog-to-TL-Verilog conversion process. It identifies target Verilog modules, applies step-by-step transformations, and ensures correctness through formal verification using SymbiYosys. The script maintains conversation context with the LLM via JSON-based message files, capturing each refactoring step in structured history directories for tracking and rollback. It also manages

file organization, creates directory structures, and coordinates interactions between different conversion components, ensuring a structured and traceable transformation workflow.

My improvements focus on Modularity**,** Performance**,** and reliability while preserving the existing workflow:

- **Modular Architecture for Maintainability:**

  Refactor the script into distinct components, such as preprocessing, transformation, and verification, for better code organization, while improving extensibility to allow future enhancements without disrupting core functionality.

- **Advanced Error Handling:**

  Implement detailed debugging messages that offer meaningful insights instead of generic error outputs, and categorize errors, such as parsing issues, LLM misinterpretations, and verification failures, to simplify troubleshooting.

- **Intelligent Pattern Recognition:**

  Detect and optimize common Verilog structures, such as state machines, counters, and pipelines, by applying targeted transformation strategies rather than relying on generic rule-based conversions.

- **Performance Optimizations:**

  Reduce LLM API calls by optimizing prompt efficiency and context handling, while implementing smarter resource management to minimize redundant computations.

- **Enhanced Documentation & Explainability:**

  Generate step-by-step transformation logs that explain the reasoning behind each modification, offering users clear insights into the AI's decision-making process and enhancing transparency.

### prompt.json:

The prompts.json file serves as the recipe book for Verilog-to-TL-Verilog conversion, containing structured prompts that guide the LLM through refactoring and structural transformations. Each prompt includes an ID, description, and detailed instructions, encoding expert knowledge into an executable format.

Proposed enhancements focus on Advance Prompt Engineering to improve accuracy and efficiency. Key improvements include:

- **Pattern-Specific Templates**: Recognizing common hardware structures (state machines, pipelines, memory interfaces) and applying optimized transformation strategies.
- **Improved Context Management**: Carrying forward critical details about module structure, signal relationships, and design intent to reduce redundancy and enhance transformation coherence.
- **Verification-Aware Prompts**: Embedding guidance for preserving functionality, signal timing, and edge case handling to improve formal verification success rates.
- **Adaptive Conversion Paths**: Moving beyond a fixed sequence, enabling dynamic prompt selection based on the module's characteristics for more tailored transformations.

# Verification Tools:

The project leverages industry-standard formal verification tools to ensure correctness throughout the Verilog to TL-Verilog conversion process. This robust verification framework is essential for maintaining functional equivalence between original and transformed designs.

## A)SymbiYosys

SymbiYosys (SBY) serves as my primary formal verification engine, providing a critical safety net for the conversion process. This powerful tool enables us to mathematically prove that my transformations preserve the original design's functionality. In my implementation, SymbiYosys is configured through carefully crafted fev.sby files that specify verification parameters. These configurations define the modules to compare, verification bounds, and proof strategies.

The tool performs exhaustive state space exploration to ensure that for all possible inputs, the original and converted designs produce identical outputs under the same conditions. For complex conversions, SymbiYosys helps identify subtle behavioral differences that might be missed through traditional simulation. When verification fails, it generates counter examples that pinpoint exactly where and how the designs diverge, enabling precise error correction.

## B)Yosys

Yosys forms the synthesis backbone of the verification framework. This open-source synthesis tool processes RTL designs and generates netlists that represent the actual hardware implementation. In the conversion pipeline, Yosys performs several crucial functions:First, it synthesizes both the original Verilog and the transformed code, converting high-level descriptions into optimized gate-level representations. This step normalizes different coding styles, making equivalence checking more reliable.Second, Yosys enables advanced optimization techniques that can identify functionally equivalent circuits despite significant structural differences. This is particularly valuable when converting to TL-Verilog, as the transformed code often employs different structural patterns to achieve the same functionality.

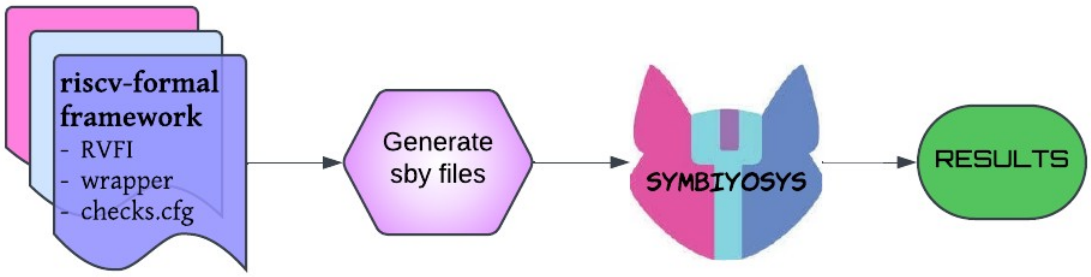


Figure 1: FOSS SymbiYosys formal verification flow

## Integration in to Conversion Pipeline

The framework integrates these tools seamlessly through the convert.py script. After each transformation step, the script automatically:

1. Generates appropriate verification scripts
2. Invokes SymbiYosys with optimal configuration
3. Analyzes verification results
4. Provides detailed feedback on success or failure

This integration ensures that every transformation is verified, maintaining a continuous chain of correctness from the original Verilog to the final TL-Verilog implementation.By leveraging these powerful verification tools, the project delivers not just code transformation but guaranteed functional equivalence, a critical requirement for hardware design. This verification-first approach distinguishes my work from simple syntax translators and ensures that converted designs can be trusted in real-world applications.

# Tools Integration:

## SandPiper Integration:

SandPiper plays a crucial role in Verilog to TL-Verilog conversion framework, transforming TL-Verilog descriptions into synthesizable SystemVerilog. It simplifies pipeline stage management by automatically generating register structures and timing logic while also centralizing clock and reset definitions, reducing complexity.

To enhance its integration,I propose leveraging SandPiper's parsing capabilities earlier in the conversion process. This will allow better utilization of TL-Verilog features such as pipelined timing abstractions and hierarchical scopes, resulting in more optimized code.

## M5 Preprocessing:

M5 extends the system's capabilities through macro-based preprocessing, enabling flexible, parameterized code generation. It adapts prompt templates dynamically and ensures uniform conversion quality by handling repetitive code structures efficiently.

To improve M5,I propose developing a conversion-specific macro library that encodes expert transformation patterns for consistent, high-quality conversions. Additionally, we aim to implement an intelligent feedback loop between verification results and M5 preprocessing, allowing automatic refinement of templates based on detected issues, reducing manual intervention.

These enhancements will significantly boost conversion quality and automation, making the process more robust and efficient.

# Example Circuits and Verification Strategy:

To demonstrate the effectiveness of Verilog to TL-Verilog conversion framework,I will be taking a structured approach that begins with simple digital circuits and gradually progresses to more complex designs. This ensures a strong foundation, allowing us to validate core conversion capabilities while refining verification techniques and creating reusable transformation patterns.

I will begin with fundamental combinational and sequential circuits—basic yet essential components in digital design. These examples help establish verification patterns, optimize transformation templates, and ensure the framework handles a variety of design structures correctly. By starting with smaller circuits, we can build confidence in the conversion process before moving on to larger and more intricate designs.

The implementation follows a clear workflow. First, we select representative circuits from open-source repositories and apply my conversion framework while documenting each transformation step. This not only helps in tracking progress but also creates a knowledge base for future optimizations. Once the initial conversion is complete, we analyze common patterns in the designs and refine transformation templates accordingly. A key focus is on optimizing clock and reset handling, as well as ensuring efficient pipeline integration, especially when dealing with TL-Verilog's pipeline abstraction.

Since verification is critical, I will incorporate multiple validation techniques. My approach includes developing comprehensive test cases, implementing formal property checks with SymbiYosys, and establishing reusable verification patterns that can be applied across different designs. This allows us to systematically prove functional equivalence between the original Verilog and the converted TL-Verilog implementations.

Beyond just correctness, these initial conversions serve an educational purpose. By showcasing fundamental transformation patterns, they provide valuable learning resources for new users. They also act as modular building blocks, helping us scale the framework for more sophisticated circuits while ensuring consistency in transformation quality.

Looking ahead, this structured foundation enables us to extend the framework to more complex designs. By supporting modular composition, improving pattern recognition, and enhancing automation, we are paving the way for a highly adaptable and efficient Verilog-to-TL-Verilog conversion system. Through continuous refinement and validation, we aim to make this framework not just a conversion tool but a reliable methodology for transitioning to TL-Verilog with confidence.

## WARP-V Integration Strategy:

WARP-V, a well-known open-source RISC-V implementation in TL-Verilog, serves as an ideal case study for showcasing the framework's ability to transform complex processor designs. By converting traditional RISC-V implementations in Verilog into TL-Verilog, we aim to demonstrate how TL-Verilog's structured abstractions improve clarity, maintainability, and performance in modern processor architectures.

## Integration Approach

My strategy focuses on adapting existing RISC-V implementations to TL-Verilog in a way that highlights key advantages in pipeline structuring, control logic, and memory interfacing.

## 1. Pipeline Structure Conversion

- Transform traditional Verilog pipeline stages into TL-Verilog's transaction-based abstractions.
- Simplify pipeline control flow and improve design readability.
- Optimize timing relationships while demonstrating pipeline stage retiming capabilities.
- This transformation will serve as a valuable reference for engineers and researchers looking to modernize processor design with TL-Verilog.

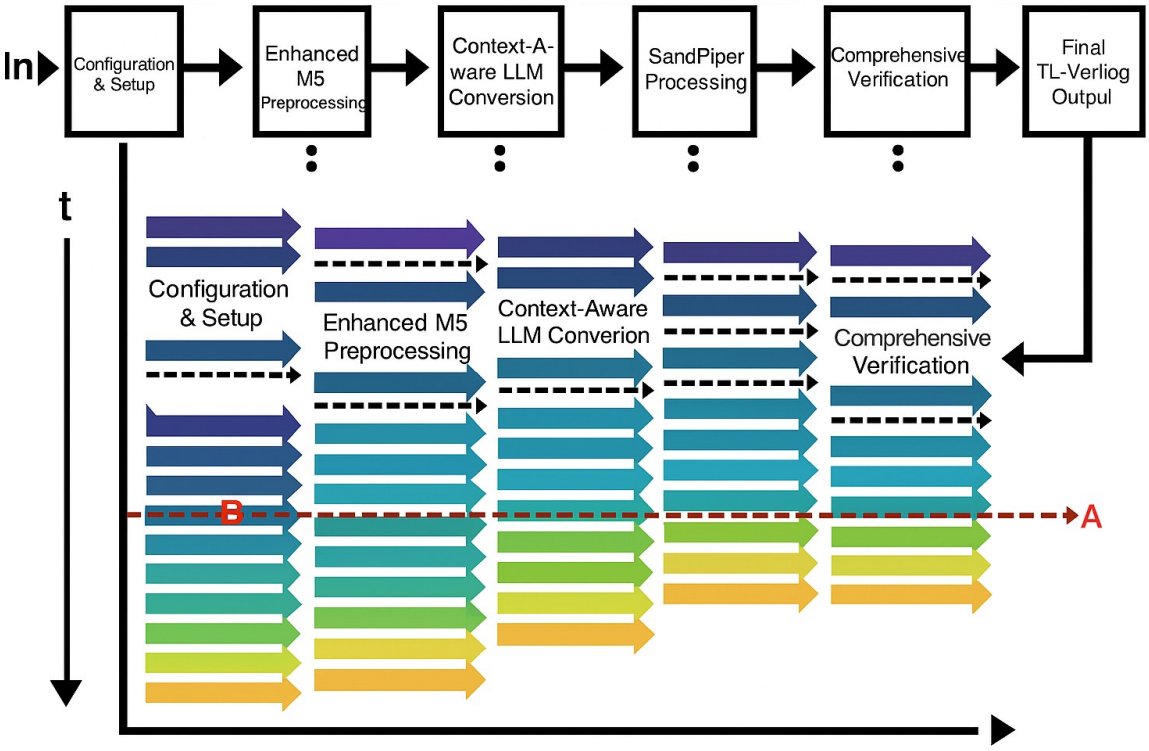


*Figure : Proposed Pipeling in the Project*

## 2. Control Logic Transformation

- Convert complex state machines into TL-Verilog's transaction-based model, reducing design complexity.
- Optimize instruction decode logic for improved clarity and efficiency.
- Adapt branch prediction and exception handling for better modularity.

By illustrating these changes, we provide a scalable approach that can be applied to other complex digital architectures, making processor design more accessible.

### 3. Memory Interface Adaptation

- Restructure memory access protocols to align with TL-Verilog's efficient design philosophy.
- Transform cache controller logic for optimized performance.
- Streamline memory pipeline integration, reducing design overhead.

These improvements will not only demonstrate TL-Verilog's advantages but also create reusable transformation templates, helping future developers accelerate their own conversions.

## Impact Prospects:

Integrating WARP-V into framework establishes a strong foundation for adapting complex processors to TL-Verilog. It provides reusable templates, best practices, and insights that simplify Verilog-to-TL-Verilog transitions. By demonstrating TL-Verilog's advantages in large-scale designs, this work encourages wider adoption, fostering structured and maintainable hardware development. Additionally, it serves as a valuable educational resource while paving the way for future adaptations of other RISC-V cores and advanced digital architectures.

## Makerchip Circuit Integration Plan

The framework will leverage Makerchip's ecosystem to demonstrate robust Verilog-to-TL-Verilog conversions. By focusing on well-established examples, I will bridge traditional Verilog implementations with modern TL-Verilog designs.

Standard Example Conversion**:** I'll convert key digital design examples, including sequential logic, timing-critical circuits, interface protocols, and arithmetic units. These conversions will serve as reference implementations, ensuring functional correctness and improved clarity.

Pipeline Design Transformation**:** TL-Verilog's timing abstraction will enhance pipeline conversions, enabling multi-stage data processing, flexible retiming, throughput optimization, and clock domain management.

State Machine Evolution**:** I'll refine state machines like protocol controllers, sequence generators, and event handlers, showcasing TL-Verilog's improved state transition representation while maintaining precise timing.

Memory Controller Enhancement**:** Complex memory interfaces, including cache controllers, arbitration, and buffer management, will be converted to TL-Verilog, demonstrating its ability to simplify timing relationships without losing accuracy.

Expected Impact**:** This integration will establish a verified library of conversions, define best practices, and provide learning resources. It will also showcase AI-assisted automation in hardware design, emphasizing TL-Verilog's advantages in clarity, efficiency, and maintainability.

# Conversion Process Enhancement

A seamless and efficient conversion process is key to ensuring accuracy and maintainability. My approach focuses on refining convert.py, strengthening verification, and improving file management to create a well-documented and reliable transformation pipeline.

## Improvements in Python Script:

To make convert.py more effective, I will introduce several enhancements. First, error handling will be significantly improved to detect issues early and provide better recovery mechanisms. The script will also feature real-time progress tracking, offering clear insights into each step of the conversion.

Beyond that, I work on intelligent pattern recognition, allowing the tool to better understand common Verilog structures and adapt accordingly. Managing context across multiple transformation steps will also be more efficient, ensuring smoother transitions from one stage to the next. Lastly, automated documentation will be integrated, keeping a structured record of every change without requiring manual effort.

## Verification Flow

Ensuring correctness is just as important as performing the conversion itself. The verification flow will include systematic equivalence checking, ensuring that the transformed TL-Verilog version functions exactly as the original Verilog design. The process will also generate detailed reports, making it easy to identify discrepancies or issues.

To further strengthen validation, I will introduce automated test case generation, allowing converted designs to be tested without manual intervention. I will also implement property preservation checks, ensuring key design constraints remain intact throughout the transformation. If any issues arise, the system will provide detailed error analysis along with suggestions for fixing them.

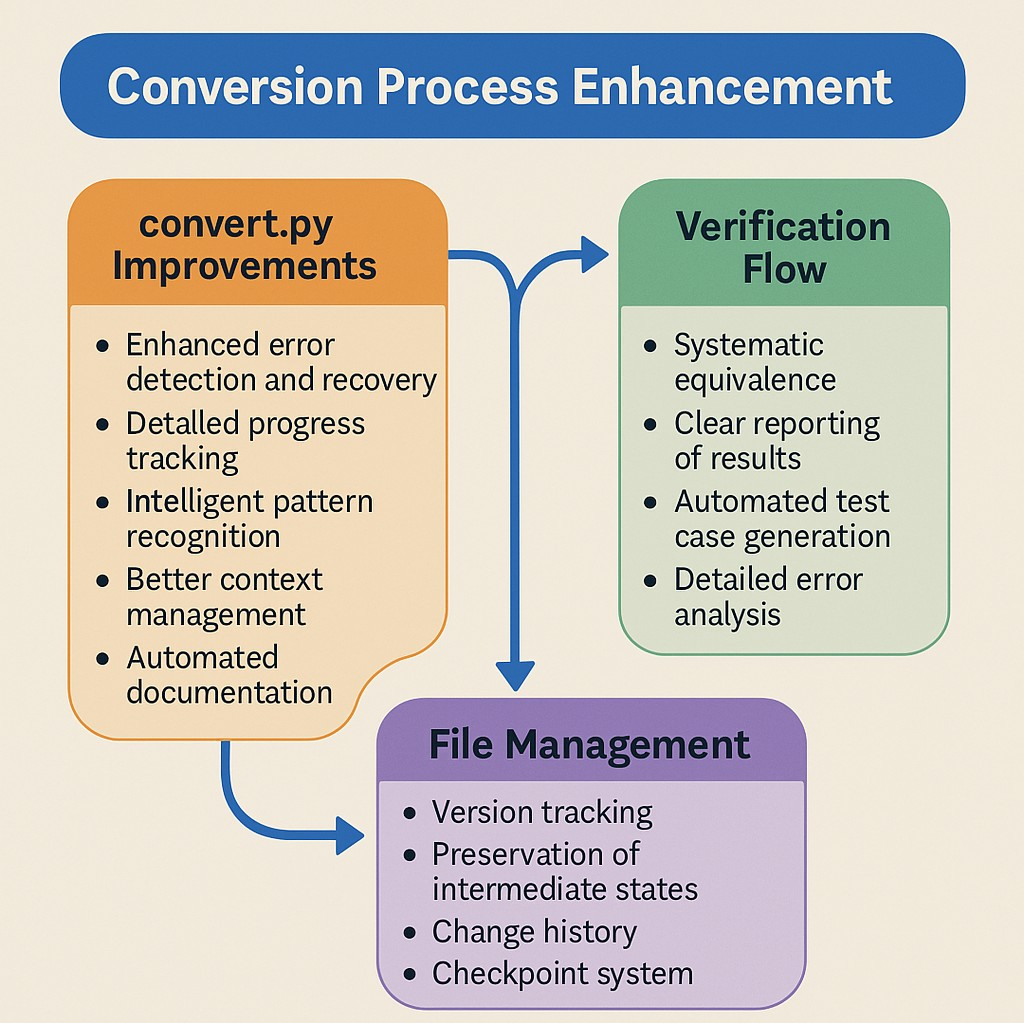


*Figure : Proposed Conversion System*

## File Management:

A well-organized file management system is crucial for tracking changes and maintaining a clear history of transformations. My approach includes version tracking, where every transformation is logged, along with documentation on the reasoning behind key design choices. Intermediate states will be preserved, ensuring that no critical information is lost.To keep things structured, we are introducing a history management system that records every conversion step, tracks how patterns were recognized and applied, and maintains a log of key decision points. Alongside this, a checkpoint system will be implemented, automatically saving critical stages of the conversion while also allowing users to define their own checkpoints.

Mistakes and unexpected challenges are inevitable, so rollback capabilities will also be included. If needed, users can revert to previous states, compare different conversion attempts, and choose the most suitable version. To simplify debugging, the system will even suggest potential recovery paths.

Together, these improvements will make the entire conversion process more transparent**,** reliable, and easy to navigate, ensuring that every transformation is well-documented and easy to revisit when needed.

# Challenges to Look After:

Developing a seamless Verilog-to-TL-Verilog conversion framework presents a set of significant challenges, primarily in three areas: tool integration, code transformation, and verification. Each stage in the conversion pipeline must work cohesively, ensuring that the final TL-Verilog output is not only functionally equivalent to the original Verilog design but also optimized for clarity and efficiency.

## Tool Integration

A fundamental challenge is ensuring smooth interaction between various tools involved in the framework. M5 preprocessing, the LLM-based transformation, SandPiper for TL-Verilog processing, and formal verification tools all have distinct functionalities, and coordinating their operation is crucial.

- **Tool Chain Coordination**: Each tool has unique input and output formats, and managing a consistent data flow between them requires careful synchronization. File format mismatches, incorrect configurations, or missing dependencies can cause failures at various stages. A structured approach is required to ensure that preprocessing, transformation, and verification steps align seamlessly.
- **Processing Pipeline Management**: The preprocessing and transformation stages must handle tool-specific syntax, intermediate representations, and conversion logic in a structured sequence. Template processing, pattern recognition, and syntax adaptation should happen in a well-defined order to ensure the correct flow of information.

Without a carefully designed pipeline, inconsistencies could arise in intermediate representations, affecting the final TL-Verilog output. Thus, robust integration mechanisms and error-handling strategies are essential for a smooth conversion experience.

## Code Transformation

The transition from Verilog to TL-Verilog is not just about changing syntax; it involves fundamental shifts in how designs are structured. Pipeline handling, clock/reset management, and state machine conversions require deep transformations while ensuring functional equivalence.

- **Pipeline Structure Analysis**: Traditional Verilog designs often have implicit pipeline stages scattered throughout the code, requiring careful identification. The challenge lies in determining optimal stage boundaries, preserving timing relationships, and translating the pipeline logic into TL-Verilog's structured @ notation. Improper stage handling can lead to incorrect execution timing, affecting the final design’s performance.
- **Clock and Reset Management**: Unlike Verilog, where clock and reset signals are explicitly defined within every sequential block, TL-Verilog allows a more centralized approach. The challenge is to retain the original design’s clock domains, reset handling, and timing constraints while streamlining them in TL-Verilog. If not handled carefully, clock mismatches or unintended domain crossings could introduce synchronization issues.
- **State Machine Transformation**: Many Verilog designs rely on complex finite-state machines (FSMs) with intricate transition conditions and control flow dependencies. Converting these into TL-Verilog’s transaction-based modeling requires restructuring them while preserving the original behavior. Ensuring that state transitions remain functionally equivalent without introducing unnecessary complexity is crucial for correctness.

This phase demands a balance between automation and accuracy—automated transformations must be precise enough to maintain functionality, while also leveraging TL-Verilog’s advantages to enhance readability and modularity.

## Verification Challenges

Once the code is transformed, ensuring correctness through verification is essential. Even the slightest timing deviation could lead to incorrect execution, making thorough validation a non-negotiable step in the framework.

- **Functional Equivalence**: The converted TL-Verilog design must retain the original behavior, including its control sequences, data paths, and pipeline execution timing. Formal verification methods will be used to check whether transformations introduce or remove unintended behaviors.

- **Property Verification**: Many Verilog designs include assertions and timing constraints to ensure correctness. The challenge here is to translate these properties into TL-Verilog and validate them using SymbiYosys (SBY) and Yosys synthesis verification. Protocol compliance, safety properties, and functional correctness must be rigorously checked.
- **Quality Assurance**: A comprehensive test suite is required to detect potential errors in transformed designs. This includes edge case detection, error identification, and performance validation to ensure that optimizations do not introduce unintended side effects. Establishing a feedback loop—where verification failures refine preprocessing templates and transformation rules—will allow the framework to continuously improve over time.

## My Approach:

To tackle these challenges, my approach is rooted in systematic problem decomposition, incremental solution development, and robust error handling. Instead of attempting a one-size-fits-all transformation, we will break down the process into manageable tasks, refining conversion strategies iteratively.

A well-defined debugging mechanism will be crucial, ensuring that automated checks detect issues early, rather than letting them propagate to later stages. Additionally, clear documentation of conversion rules and best practices will serve as a reference, making the framework reproducible and adaptable for future enhancements.

By methodically addressing these challenges, we aim to create a conversion framework that not only maintains design integrity but also enhances code quality, modularity, and scalability. The end goal is not just transformation, but an intelligent, automated system capable of producing high-quality TL-Verilog with minimal manual intervention.

## Approach Chart Diagrams:

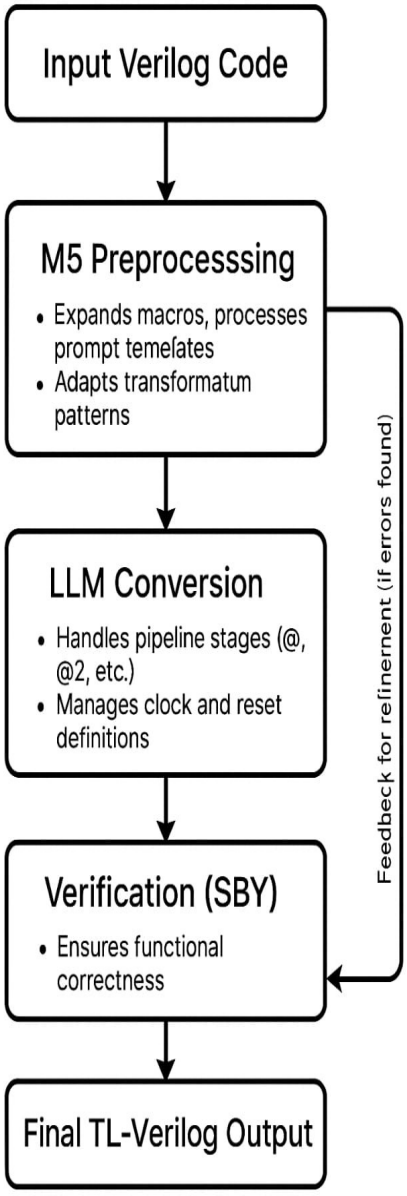


*Figure 1: Initial Project Approach*

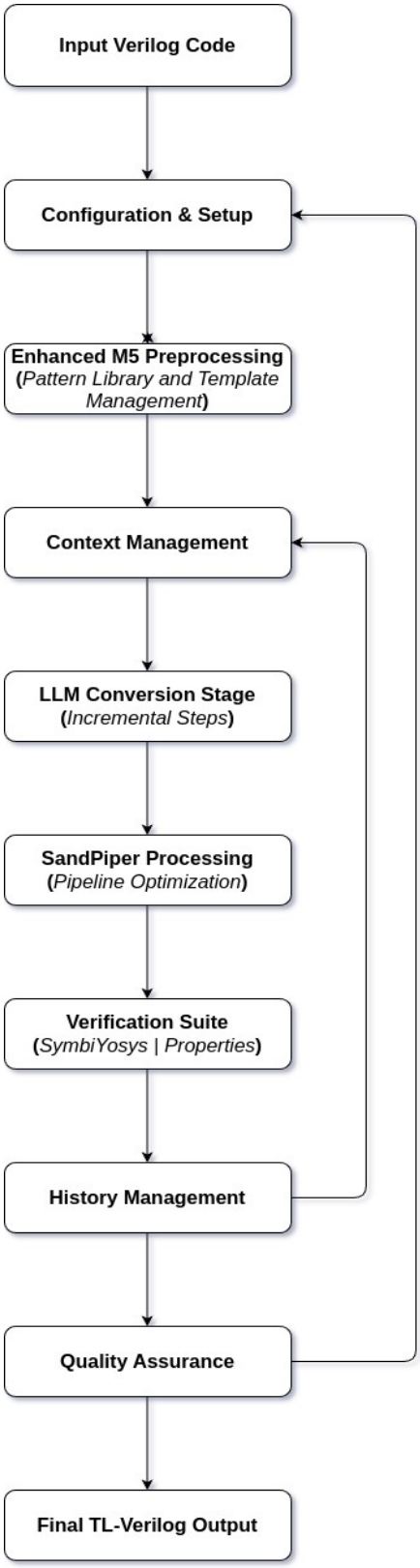


*Figure 2:My Project Approach*

# Schedule of Deliverables:

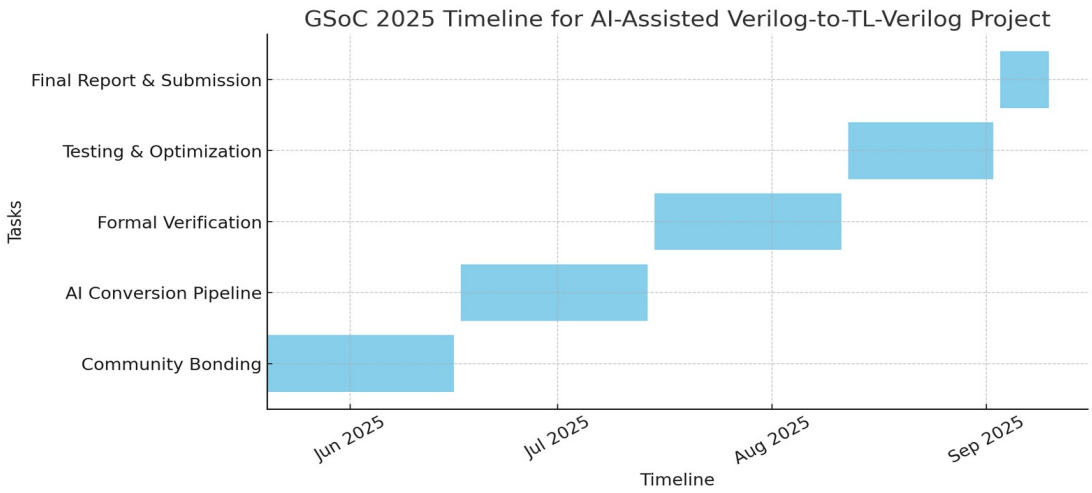


*Figure : Brief Schedule*

## Before Community Bonding

✔ The time period from April 9- May 8 is the pre - community bonding phase.

✔ Objective is to push a good number of Pull Requests to the Conversion to TLV repo.

✔ Get Familiar with the Project Work flow, Verification Concepts, Dependencies Installation.

## Community Bonding Period

- **May 8 - June 1**

✔ Get familiar with mentors, community, and existing tools.

✔ Study conversion methodologies and formal verification.

✔ Set up development, testing, and version control environment.

✔ Identify key challenges in Verilog-to-TL-Verilog conversion.

### Week 1 (May 8 – May 15)

✔ Connect with mentors and the Community.

✔ Review the Project Scope, Goals, and Expected Outcomes.

✔ Set up and test the development environment (Python, Yosys, SymbiYosys, Makerchip).

- ✔ Study the structure of convert.py and fev.sby.

**Week 2 (May 16 – May 23)**

- ✔ Analyse existing conversion patterns in Verilog and TL-Verilog.
- ✔ Review WARP-V and Makerchip examples for pipeline conversion strategies.
- ✔ Document the challenges in conversion (clock domain handling, pipeline structure,statemachines).
- ✔ Identify initial circuits for conversion.

**Week 3 (May 24 – June 1)**

- ✔ Plan the project roadmap with deliverables and milestones.
- ✔ Define testing and validation methodologies.
- ✔ Set up formal verification tools (SymbiYosys, property checking).
- ✔ Finalize the approach for pipeline transformation and state machine conversion.

## Coding Period - Phase 1

- **June 2 – July 14** (Midterm Evaluation: **July 14 - July 18**)

  Developing the core Verilog-to-TL-Verilog conversion pipeline and establish initial verification flow.

**Week 4 – 5 (June 2 - June 16)**

- ✔ Implement the basic conversion pipeline (convert.py enhancements).
- ✔ Add core pattern recognition for Verilog constructs.
- ✔ Develop first combinational circuit conversions with TL-Verilog. Implement file tracking and version control for conversion steps.

**Week 6 - 7(June 17 - July 1)**

- ✔ Extend support for sequential circuits and basic pipelining.
- ✔ Establish a formal verification framework using SymbiYosys.
- ✔ Create automated test case generation for verification.
- ✔ Develop a rollback mechanism for debugging failed transformations.

### Week (July 2 - July 14)

- ✔ Conduct debugging and optimize transformation logic.
- ✔ Document Phase 1 progress and prepare midterm evaluation report.
- ✔ Gather mentor feedback for next steps.

## Coding Period - Phase 2

### July 14 - August 25

Extend the framework to handle complex circuits, pipelines, and advanced optimizations.

### Week 9 - 10(July 14 - July 28)

- ✔ Implement pipeline optimization techniques for efficient TL-Verilog conversion.
- ✔ Improve state machine transformation for control logic.
- ✔ Develop memory interface handling for complex designs.
- ✔ Test and validate against a set of larger open-source circuits.

### Week 11 - 12(July 29 - August 11)

- ✔ Enhance error detection and reporting mechanisms.
- ✔ Develop template-based transformation for reusable conversions.
- ✔ Optimize clock domain handling and reset logic.
- ✔ Perform stress testing with complex TL-Verilog designs.

### Week 13 - 14(August 12 - August 25)

- ✔ Finalize WARP-V integration as a proof of concept.
- ✔ Complete documentation and user guides.
- ✔ Conduct a final round of verification and debugging.
- ✔ Prepare for the final submission.

## Final Evaluation Period

### August 25 - September 1

- ✔ Final testing and performance optimizations.
- ✔ Prepare and submit the final work product.
- ✔ Submit final project report and evaluation.

## Extended Coding Period

**September 1 - November 9**

**For additional improvements:**

- Further optimizations in transformation logic.
- Additional formal verification techniques.
- Documentation enhancements for long-term maintainability.

## Contributions and pull requests to the project

**1. Pull Request: #04**- Enhanced initialization script with comprehensive tool installation and dependency management guide.

**State:** Merged.

**2. Pull Request: #06**- Add examples and tutorials for TL-Verilog conversion: - Basic examples (D flip-flop, counter, mux) - Intermediate examples (shift register, traffic light) - Tutorials (getting started, common patterns, troubleshooting)

**State:** Closed.

**3. Pull Request: #09**- Add reference to conversion examples repository

State: Merged.

# Why This Project?

I am deeply passionate about advancing hardware description languages and making digital design more efficient and accessible. The Verilog-to-TL-Verilog conversion project aligns perfectly with my interests, offering a unique opportunity to merge traditional hardware design with modern AI-driven automation.

## Technical Innovation

This project presents an exciting challenge in developing an intelligent conversion framework powered by large language models (LLMs). It involves working with advanced AI techniques to transform and optimize HDL code, ensuring accuracy and efficiency in the process. By tackling complex code transformations and verification strategies, I will gain hands-on experience in building scalable and maintainable solutions that can benefit the broader hardware design community.

## Community Impact

Simplifying Verilog-to-TL-Verilog conversion can significantly benefit hardware designers, researchers, and students by making existing designs easier to maintain and understand. It reduces barriers to adopting modern HDL practices, fostering a more accessible learning environment. Additionally, contributing to open-source hardware development ensures that valuable educational resources and improved workflows are available to the community.

## Learning Opportunity

This project provides a strong foundation for expanding my knowledge in hardware description languages, formal verification techniques, and AI-assisted code refactoring. By engaging in open-source collaboration, I will gain insights into best practices for developing high-quality software tools while deepening my understanding of modern digital design methodologies.

## Long-Term Vision

Beyond its immediate goals, this project represents a step toward shaping the future of hardware design automation. The tools and methodologies developed here can serve as a foundation for future innovations in HDL transformation. Establishing best practices for Verilog-to-TL-Verilog conversion and fostering collaboration between traditional and modern hardware communities can lead to long-lasting improvements in digital design workflows.

## Personal Growth

This project aligns with my long-term career goals by allowing me to work alongside experienced mentors and contribute meaningfully to open-source development. It provides a unique opportunity to refine my problem-solving and project management skills while bridging the gap between hardware and software domains. Through this experience, I aim to strengthen my expertise in digital design and AI-driven automation, positioning myself for future contributions to the field.

# Biographical Information

## Educational background

I am a third-year undergraduate pursuing a B.Tech in Electronics and Communication Engineering (ECE) at Rajiv Gandhi University of Knowledge Technologies (RGUKT), Ongole Campus, with an academic percentage of 87%. Throughout my academic journey, I have maintained a strong academic record and actively engaged in both technical and extracurricular activities. My coursework and projects have given me hands-on experience with core concepts in digital design, embedded systems, and circuit simulations. I am a member of my college's innovation club, IDEA Forge, where I collaborate on creative technological solutions. Additionally, I am an NSS volunteer and a Bharat Scouts and Guides member, actively participating in community service and leadership initiatives. Beyond academics, I have a keen interest in sports, particularly football, badminton, and chess, which help me stay active and enhance my strategic thinking.

## Technical Interests

I have a deep interest in hardware design, digital circuits, and open-source contributions. My passion lies in exploring Verilog and TL-Verilog, particularly in the context of AI-assisted automation for hardware refactoring. I am also keen on formal verification techniques, ensuring correctness and reliability in hardware designs.I also working on optimization, and verification in Verilog. Additionally, I have experience in CPU benchmarking, FPGA-based projects, MATLAB simulations, and digital protocol implementations. Most importantly, “Building a RISC-V CPU Core”,course of Mr.Steve Hoover in Edx Platform,earned me alot of interest in TL-verilog and Makerchip.

## Notable Projects

- DIR-V Systems Hackathon Project – Optimized CoreMark CPU benchmarks for the SHAKTI processor, focusing on memory optimizations, compiler enhancements, and reducing execution time.
- Gym Trainer Chatbot using LLMs – Built a smart AI chatbot leveraging Large Language Models (LLMs) to provide personalized fitness recommendations and assist users in workout routines.
- UART Protocol Implementation (Verilog) – Developed and verified a Universal Asynchronous Receiver-Transmitter (UART) protocol in Verilog, ensuring reliable serial communication.

- Asynchronous FIFO on Basys-3 Board – Designed and implemented an asynchronous FIFO on an FPGA (Basys-3), ensuring smooth clock domain crossing for high-speed data transfer.
- MATLAB-Based Voting Machine – Designed and implemented a simulation-based voting system with optimized image handling for improved performance.

## Other open-source contributions

As part of the Research Migration Project at FOSSEE, IIT Bombay, I worked on eSim, an open-source EDA tool for analog circuit simulation that integrates KiCad, LTspice, and other embedded tools. My contribution involved designing and simulating wave generators, specifically a sawtooth waveform generator and a triangular waveform generator. The project was successfully completed and submitted, demonstrating my proficiency in circuit simulation and my ability to work with open-source hardware design tools.

## Communication

### Working Hours -

I am flexible with my timings and can work dedicatedly for hours together. I would like to work during the daytime as I find myself more productive during the early hours. Also, I will make sure that GSoC will be my only major commitment this summer and that I give it my full attention.

### Communication preferences -

I am most comfortable with the Google meet or Zoom Meetings but can adjust to whatever suits the mentors.In case of work delay due to personal or other reasons, I assure you that I will work longer and efficiently to complete the backlog in the available time. I plan on writing weekly progress blogs on my medium publication to establish a systematic solution to prevent missing out on a week's work. I would be sharing the same with my mentors while having active communication with them.

Timezone - Indian Standard Time (IST) - UTC+5:30
Email- sugurthimanisharma@gmail.com,secondary: ro200007@rguktong.ac.in
Phone - (+91) 9392521762
Linkedin - https://www.linkedin.com/in/manisharma-sugurthi
Github - https://github.com/sharma-sugurthi

## Commitments and engagements during GSoC

If selected, I'll be fully dedicated to the project. I have no prior commitments during the span of May to September and will be able to provide 40+ hours a week to the project.

## Post GSoC Plans

- I plan to harvest the experience and knowledge I will gain from GSoC and move forward to making regular contributions to open source projects.
- I have been doing projects and uploading them publicly for a year now but wasn't aware of the big community behind many of the tools I have been using.
- Also, I plan to help my juniors prepare for next year's GSoC, introduce them to open source, and help them get a head start on projects my club mates and I have worked on in GSoC.