Northern Arizona University

Technological Feasibility Analysis

LeGIT/AI - Logical Expression Generator Intended for TLA+ with Artificial Intelligence

CS486C - Senior Capstone Design

Team LegitCode: Gordon Wray, Aryan Sharma, and Brodric Martinez

Client: SanDisk Corp.

19th of March 2026

Table of Contents

- Introduction
- Technological Challenges
- Technology Analysis
    - 3.1 Development Environment & AI Engine
    - 3.2 Formal Verification Tooling
    - 3.3 Data Storage Strategy
- Technology Integration
- Conclusion

1. Introduction

Modern hardware systems, particularly high performance storage technologies, demand extremely precise and reliable specifications to function correctly at scale. At SanDisk, the engineering teams take the quality of storage products seriously, and the company strives to be best in class in upholding the global SanDisk brand reputation. To further improve product design assurance, SanDisk has launched an important initiative to formally specify their hardware designs using PlusCal and TLA+ (Temporal Logic of Actions), enabling formal verification through a model checker. Bug fixes are exponentially more costly the deeper into the production and manufacturing phases they are found. Therefore, catching logical errors at the specification phase is critical for the business.

TLA+ is a mathematical specification language based on set theory and logic, which is used to write rigorous and unambiguous formal specifications. Invented by ACM Turing Awardee Leslie Lamport, TLA+ is an industry standard utilized by major technology companies like AWS, Microsoft, Oracle, Google, and Intel to guarantee system correctness. Before any higher-level product design can be formally verified against its requirements, the requirements themselves must first be formally verified using the common verifier. These requirements must be carefully converted from informal natural language into their shared mathematical formal language, TLA+.

Currently, manually reading through a massive block of requirement text to extract the relevant context for a specific feature and then translating it into logical expressions is a daunting and time-consuming task. For example, engineers often spend months manually translating dense, 800-page specification documents into TLA+. Without immediate feedback from the common verifier, it becomes exceptionally difficult to detect inconsistencies that arise when all logical expressions are combined into a single TLA+ formal specification.

To address these ongoing challenges, this project proposes LeGIT/AI, an automation tool integrated into the VS Code IDE needed to help streamline this challenging process. The requirements will be provided in the form of NVMe and PCIe Standard Specification PDF files. Instead of hardcoding a proprietary parsing application, the solution empowers users to leverage GitHub Copilot in Agent Mode to extract the context of a selected feature from the requirements and generate the necessary definitions and formulas intended for a TLA+ specification draft. This prompt-driven, human-in-the-loop system utilizes a strictly managed repository of formatting rules to guide the AI, ensuring that SanDisk engineers can dramatically reduce the time spent on manual translation while maintaining mathematical rigor.

2. Technological Challenges

Moving from informal natural language requirements into a shared mathematical formal language requires bridging a massive semantic gap. Transitioning to a prompt-driven Copilot architecture shifts the primary technological hurdles from software backend development to advanced prompt engineering, AI context management, and formal verification integration. The core challenges include:

- Context Management and Prompt Engineering: The system must accurately extract structured meaning directly from dense and complex NVMe and PCIe Standard Specification PDF files. Because these documents can span hundreds of pages, the primary technical risk is AI hallucination, where the model might invent constraints or deviate from the required SanDisk TP4163.tla style guide. Overcoming this requires developing highly optimized .md instructional files (Agent Skills) that explicitly define TLA+ syntax boundaries and formatting templates for the Large Language Model (LLM).

- TLA+ Code Generation Accuracy: The application must accurately generate the necessary definitions and formulas intended for a TLA+ specification draft, alongside corresponding TLA+ and CFG files. AI outputs are inherently probabilistic and may produce incorrect, ambiguous, or incomplete logical expressions. The challenge lies in ensuring the AI's output adheres strictly to formal mathematical logic rather than standard imperative programming paradigms.

- Model Checker Integration and Trust: Users (engineers) may trust AI-generated outputs without sufficient validation, especially if the tool appears to automate complex tasks successfully. Therefore, the workflow requires immediate feedback from the TLC model checker. Detecting inconsistencies that arise when logical expressions are combined is difficult without this automated interaction. The challenge is creating a frictionless loop where engineers are forced to verify the AI's probabilistic output against a deterministic mathematical checker.

- Shared State and Dictionary Maintenance: The system must successfully create a dictionary of the VARIABLES and CONSTANTS. This dictionary will be used as

a common global collection of terms or identifiers found in NVMe and PCIe PDF files, which future TLA+ specifications will reference. Ensuring the AI consistently updates and references this file without overwriting historical data across multiple working sessions requires strict prompt directives.

3. Technology Analysis

To overcome these technical challenges, we evaluated the necessary tools to establish a robust, Copilot-driven workflow. Our analysis focuses on maximizing the capabilities of existing VS Code integrations to minimize custom development overhead and ensure long-term maintainability for SanDisk.

3.1 Development Environment & AI Engine

- VS Code IDE: The system must run within the VS Code IDE environment. Standalone applications lack integration with existing developer setups, and web-based interfaces limit local tooling capabilities. VS Code provides the exact lightweight, extensible environment needed for this project, allowing engineers to operate where they already write code.
- GitHub Copilot (Agent Mode): We will utilize GitHub Copilot with agent mode, prioritizing Claude Sonnet 4.5 or Opus 4.5. Rule-based parsing systems lack the flexibility required for complex technical documents, and traditional NLP tools struggle with domain-specific semantics. Large Language Models demonstrate the strong contextual understanding and adaptability needed for this parsing phase. By utilizing Claude's expansive context window, the system can ingest the massive NVMe and PCIe specification PDFs directly without requiring a custom-built PDF parsing backend.

3.2 Formal Verification Tooling

- TLA+ Foundation Extension: We will use the TLA+ extension by the TLA+ Foundation to provide syntax highlighting and environment integration. This ensures that as the AI generates the code, the IDE correctly recognizes the .tla syntax.
- TLC Model Checker: The generated code will be verified against the TLC Model Checker. AI results are probabilistic, but they can be exhaustively checked across all possible scenarios by the model checker, unlike traditional testing. This is possible because the AI output is in TLA+ form, which can be verified by TLC,

an external tool for model-checking TLA+. This entirely mitigates the risk of hallucinated logic.
- Java Runtime: The TLC model checker strictly requires a local JRE properly mapped to the system PATH. We will utilize OpenJDK >= 11.0.6 (strictly non-Oracle versions) to support the model checker and prevent execution failures during state-space exploration.

3.3 Data Storage Strategy

- JSON Dictionary System: The system must maintain a reusable dictionary of VARIABLES and CONSTANTS. While relational or NoSQL databases offer robust features, they add unnecessary complexity and overhead at this early stage of development. JSON is simple and entirely sufficient for structured key-value data. A simple flat-file storage system allows the Copilot Agent to easily read, update, and sort variables in the workspace without requiring external database connections.

4. Technology Integration

To ensure all parts of the system work together smoothly, the overall design follows a pipeline architecture integrated directly into the VS Code environment. This allows users to move from raw PDF requirements to a verified TLA+ specification without switching between multiple tools. The system workflow relies on a human-in-the-loop methodology:

1. Repository Setup & Skill Configuration: The engineer opens the strictly controlled project repository in VS Code. This repository contains the essential .md files (Agent Skills) that define SanDisk's specific TLA+ formatting rules, translation heuristics, and the global JSON dictionary.
2. Document Upload & Prompting: The user provides NVMe or PCIe specification PDF files. They attach the PDF directly to the GitHub Copilot Chat interface and apply the predefined prompt to target a specific hardware feature.
3. AI Analysis & Draft Generation: The tool uses GitHub Copilot in Agent Mode to extract the context of a selected feature from the requirements. Based on the extracted context and guided by the .md rules, the system generates a draft TLA+ specification along with the required CFG file. This allows the specification to be immediately prepared for verification without manual setup.
4. Manual Review & Verification: Users will be able to manually edit this TLA+ specification draft within the VS Code IDE. The user then manually executes the

TLC model checker against the draft to explore all possible system states. This step ensures that the generated logic is valid and free of inconsistencies.

5. Iterative Feedback Loop: If errors are found, the user can edit the TLA+ specification directly in VS Code. Alternatively, the user copies the TLC error output and pastes it back into the Copilot chat. The draft and any edits can be iteratively verified with the TLC model checker, either automatically through the tool or via Agent Mode AI. This creates a continuous feedback loop until the specification is mathematically correct.
6. Dictionary Update: Once verification is successful, the system extracts all VARIABLES and CONSTANTS from the TLA+ file and stores them in a JSON-based dictionary. This dictionary acts as a shared reference for future specifications, ensuring consistent nomenclature across the engineering team.
7. Final Output: The final result is a successfully model-checked TLA+ formal specification of the targeted feature from the requirements.

5. Conclusion

The process of turning written requirements into a formal language like TLA+ is a difficult but important problem in modern system design. This project focuses on finding a realistic way to do that by combining different tools into one system. By leveraging the built-in capabilities of GitHub Copilot and the Claude LLM models, the engineering teams can sidestep the complexities of building custom parsing software, while effectively handling massive 800-page specification documents.

One of the main strengths of this system is how everything is connected into a single workflow inside VS Code. The system moves from reading the PDF, to generating TLA+, to verifying it with the TLC model checker all in one place. The proposal mitigates the tedious verification of AI-generated results because the output can be verified by an external tool. Even though AI is not always perfect, the verification step helps ensure that the final output is reliable. The proposed prompt-driven, human-in-the-loop workflow successfully addresses SanDisk's core needs: it significantly reduces the time and effort required to manually interpret and formalize requirements, improving overall engineering productivity. Based on our analysis, we believe this system is feasible and ready to move forward into development.