Project Gabriel: AI-Accelerated Formally Verified FPGA Security for Critical Infrastructure

Caleb Parikh
Independent Researcher

Vy Hong
UK AISI

John Braithwaite
Leonardo

Abstract

Project Gabriel demonstrates how AI can accelerate the development of formally verified hardware security systems for critical infrastructure protection. We developed an FPGA-based authentication gatekeeper that enforces hardware-level access control for microcontroller programming, with all logic formally verified using SymbiYosys. The project explores three progressive stages: (1) a formally verified FPGA security system using challenge-response cryptography, (2) an AI-hardware feedback loop enabling iterative development with direct hardware observation, and (3) adversarial AI verification using red team versus blue team methodology. Our results show that AI can successfully write formally verifiable Verilog code, that hardware feedback loops dramatically accelerate development, and that adversarial AI approaches uncover vulnerabilities missed by single-agent development. This work addresses AI risk by demonstrating scalable defensive capabilities for critical infrastructure and establishes foundations for verifiable compute governance systems. While not production-ready, we successfully derisked major uncertainties and created an MVP formal verification workflow applicable to FPGA security systems.

Keywords: Formal verification, FPGA security, AI-accelerated development, hardware security, critical infrastructure defense, adversarial testing

Introduction

The cybersecurity landscape is evolving rapidly as AI systems become increasingly sophisticated. Recent developments have highlighted the vulnerability of critical infrastructure, industrial control systems, and embedded devices to AI-powered attacks. The October 2024 advances in agentic AI systems demonstrated unprecedented capabilities in orchestrating attacks against both enterprise and consumer-level systems. This creates an urgent need for scalable, verifiable security solutions that can protect critical infrastructure from emerging threats.

However, traditional approaches to hardware security face significant challenges. Formal verification of hardware designs, while providing mathematical guarantees of correctness, has historically been prohibitively expensive and time-consuming.

Human Verilog programmers capable of writing formally verifiable code are scarce and costly. Additionally, the iterative development process for FPGA systems involves long feedback loops, as developers must repeatedly compile, flash, and test hardware to validate their designs.

Project Gabriel addresses these challenges by exploring how AI can accelerate the development of formally verified hardware security systems. Our core innovation is using AI (specifically Claude Code) to dramatically reduce the cost of producing formally verifiable hardware designs while maintaining the rigorous guarantees provided by formal verification tools. We focus on three key research questions:

1. Can AI successfully write hardware description language (Verilog) code that passes formal verification against security specifications?
2. Can direct hardware feedback loops enable AI to iteratively improve FPGA designs based on observed behavior?
3. Can adversarial AI methodologies (red team versus blue team) improve the security properties of formally verified systems?

This project is directly relevant to AI safety and defensive acceleration. By demonstrating that AI can design and verify hardware security systems orders of magnitude faster and cheaper than traditional methods, we enable scalable protection of critical infrastructure that could be targeted under various AI risk scenarios. Furthermore, our architecture establishes foundations for verifiable compute governance, where cryptographic proofs can demonstrate how computing clusters are being used—essential for international AI agreements requiring verification of compute usage restrictions.

We implemented our system on an iCEBreaker FPGA board (Lattice iCE40UP5K) connected to Raspberry Pi Pico microcontrollers. The FPGA acts as a hardware-level security gatekeeper, controlling access to the microcontroller's Serial Wire Debug (SWD) programming interface through challenge-response authentication over UART at 115200 baud. All FPGA logic modules are formally verified using SymbiYosys with bounded model checking over 30+ clock cycles.

Methods

Our methodology progressed through three distinct stages, each building upon the previous:

Stage 1: Formally Verified FPGA Security System

We designed and implemented a hardware authentication gatekeeper using the iCEBreaker FPGA development board. The system architecture consists of several formally verified Verilog modules:

- UART receiver (uart_rx.v) for receiving authentication commands at 115200 baud
- UART transmitter (uart_tx.v) for sending challenge responses

- Linear Feedback Shift Register (lfsr.v) for cryptographic challenge generation
- Top-level controller (top.v) integrating all subsystems and managing SWD access control

The security model enforces hardware-level access control by physically controlling the SWD clock line. When the system is in an unauthenticated state, the FPGA jams the SWD clock signal, preventing any programming attempts. Only after successful challenge-response authentication does the FPGA release control of the clock line, allowing authorized programming.

Each Verilog module includes formal properties specified using SystemVerilog Assertions (SVA). We used SymbiYosys (sby), a front-end for Yosys formal verification, with the SMT solver backend (yosys-smtbmc) to prove correctness properties. Verification was performed using bounded model checking with depth of 30+ clock cycles for each module, with separate verification files (uart_rx.sby, uart_tx.sby, lfsr.sby, top.sby) specifying the formal properties and verification parameters.

The development toolchain consisted of:

- IceStorm for FPGA place-and-route
- Yosys for synthesis
- nextpnr-ice40 for placement and routing
- SymbiYosys for formal verification
- Z3 SMT solver for proof generation

Stage 2: AI-Hardware Development Loop

We developed a novel workflow enabling Claude Code to receive direct feedback from FPGA hardware. This closed-loop system works as follows:

1. Claude writes Verilog code based on specifications and previous feedback
2. Code is automatically compiled using the IceStorm toolchain
3. Resulting bitstream is flashed to the FPGA via USB
4. Hardware behavior is observed through UART output and LED indicators
5. Observations are fed back to Claude as text descriptions
6. Claude analyzes the feedback and iteratively improves the design

We created helper scripts in the claude-fpga-helpers/ directory to automate this process. The workflow dramatically reduced iteration time from hours to minutes, as Claude could "see" the actual hardware behavior rather than relying solely on simulation.

Example experiments included progressively complex LED blinking patterns, pin state transfer between input and output pins, and eventually the complete secure gatekeeper implementation. Each experiment validated that Claude could understand hardware feedback and make appropriate design modifications.

Stage 3: Adversarial AI Verification

We implemented a red team versus blue team methodology using separate Claude instances with different prompts and objectives:

Blue Team (Defensive AI):

- Writes formal specifications for security properties
- Implements Verilog modules satisfying specifications
- Adds assertion-based verification properties
- Responds to red team findings by patching vulnerabilities

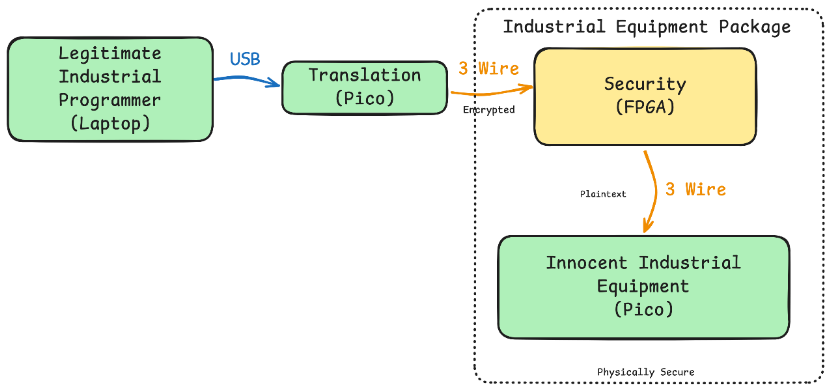
Red Team (Offensive AI):

- Analyzes specifications for completeness gaps
- Identifies potential exploit vectors
- Documents attack scenarios
- Tests formal verification assumptions

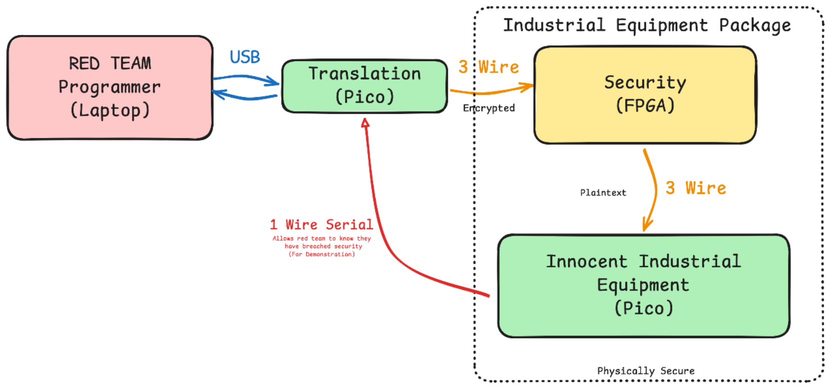
Both teams work from a shared codebase in the red-blue-exercise/shared/ directory, with each team's Claude instance operating from team-specific prompts that define their role and objectives. The red team attempts to find vulnerabilities, specification gaps, or incorrect assumptions, while the blue team iteratively hardens the system.

This adversarial approach mirrors security testing methodologies used in software development but applies them to formally verified hardware, where the goal is not just to find bugs but to identify gaps in formal specifications themselves.

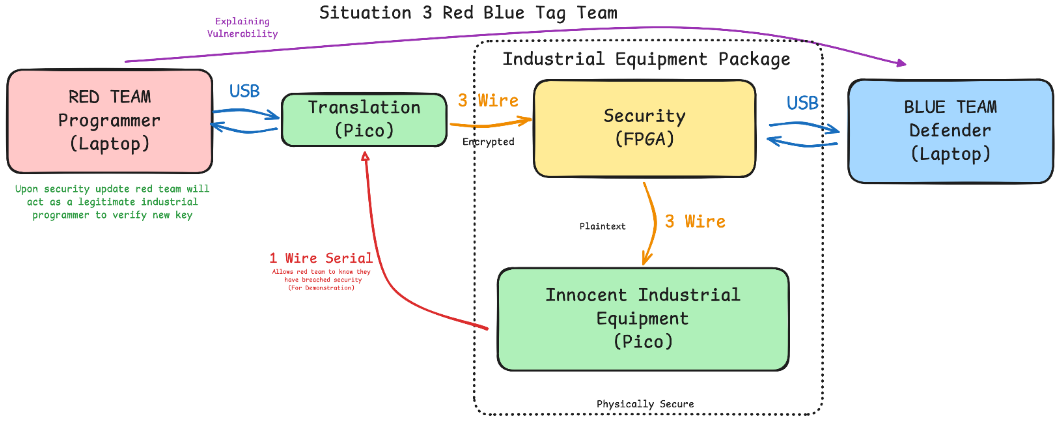
Situation 1 Legitimate Actor


Legitimate Industrial Programmer (Laptop)
USB
Translation (Pico)
3 Wire
Encrypted
Industrial Equipment Package
Security (FPGA)
Plaintext
3 Wire
Innocent Industrial Equipment (Pico)
Physically Secure

Situation 2 Red Team Attack


RED TEAM Programmer (Laptop)
USB
Translation (Pico)
3 Wire
Encrypted
Industrial Equipment Package
Security (FPGA)
Plaintext
3 Wire
1 Wire Serial
Allows red team to know they have breached security (For Demonstration)
Innocent Industrial Equipment (Pico)
Physically Secure

Situation 3 Red Blue Tag Team


Explaining Vulnerability
RED TEAM Programmer (Laptop)
USB
Translation (Pico)
3 Wire
Encrypted
Industrial Equipment Package
Security (FPGA)
USB
BLUE TEAM Defender (Laptop)
Upon security update red team will act as a legitimate industrial programmer to verify new key
Plaintext
3 Wire
1 Wire Serial
Allows red team to know they have breached security (For Demonstration)
Innocent Industrial Equipment (Pico)
Physically Secure

Hardware Setup

The complete system requires:

- iCEBreaker FPGA board (Lattice iCE40UP5K FPGA)
- Raspberry Pi Pico as target device
- Optional second Raspberry Pi Pico configured as PicoProbe for SWD debugging
- USB cables and jumper wires for connections

All code, build scripts, and documentation are available in our public GitHub repository at github.com/Calebp98/project_gabriel.

Results

Our project successfully demonstrated that AI can accelerate the development of formally verified hardware security systems across all three stages:

Stage 1 Results: Formal Verification Success

All core Verilog modules passed formal verification:

- UART receiver module verified for correct state machine operation and data capture
- UART transmitter module verified for proper baud rate timing and data transmission
- LFSR module verified for correct pseudo-random sequence generation
- Top-level integration module verified for correct authentication state transitions

The formal verification used bounded model checking with 30+ cycle depth, proving that the specified security properties hold for all possible input sequences within the bounded time frame. Key verified properties included:

- Authentication state cannot transition to "authorized" without correct challenge response
- SWD clock control remains jammed in unauthorized state
- UART communication maintains protocol timing constraints
- No state machine deadlocks exist in any module

Stage 2 Results: AI-Hardware Feedback Loop

The hardware feedback loop successfully enabled iterative AI-driven development. Claude Code was able to:

- Write initial Verilog implementations based on natural language specifications
- Receive feedback from actual FPGA behavior via UART and LED observations
- Debug timing issues by correlating observed behavior with code
- Iteratively improve designs based on hardware testing results

Development cycle times improved dramatically compared to traditional workflows. Tasks that would typically require hours of manual coding, debugging, and testing were completed in 15-30 minute iterations. The AI successfully handled complex debugging scenarios, such as UART baud rate mismatches and clock domain issues, by analyzing the symptoms described in feedback.

Stage 3 Results: Adversarial Testing Findings

The red team versus blue team methodology uncovered several important security considerations that were not initially apparent:

- Specification gaps in timing requirements that could allow race conditions
- Incomplete formal properties that verified correct behavior but not all failure modes
- Assumptions about physical layer security that required explicit documentation
- Need for additional verification of reset behavior and initialization states

The adversarial approach proved particularly valuable because it identified not just implementation bugs, but gaps in the formal specifications themselves—a category of vulnerability that would not be caught by verification of incomplete specifications.

Overall Project Impact

While we did not achieve a production-ready system within the hackathon timeframe, we successfully:

1. Demonstrated that AI can write formally verifiable hardware at significantly reduced cost
2. Proved that hardware feedback loops are practical for AI-driven FPGA development
3. Showed that adversarial AI methodologies improve security properties
4. Created a reusable MVP workflow for formal verification of FPGA security systems
5. Derisked major technical uncertainties about AI-accelerated hardware development

The cost reduction potential is substantial: Claude Code is orders of magnitude cheaper than human Verilog programmers for writing specifications and implementations, making formal verification economically feasible for applications that were previously too expensive to secure at the hardware level.

Discussion and Conclusion

Project Gabriel demonstrates that AI can significantly accelerate the development of formally verified hardware security systems, addressing a critical need in defensive AI safety. Our three-stage approach validated that modern AI systems can successfully write hardware description language code that passes rigorous formal verification, that direct hardware feedback enables rapid iterative improvement, and that adversarial AI methodologies uncover important security considerations.

Implications for AI Safety and Critical Infrastructure

The ability to rapidly develop formally verified hardware security systems has profound implications for AI safety. As AI systems become more capable, critical infrastructure including industrial control systems, power grids, telecommunications networks, and medical devices become potential targets for AI-powered attacks. Traditional hardware security development is too slow and expensive to keep pace with rapidly evolving threats. By reducing the cost of formal verification by orders of magnitude, AI-accelerated development enables scalable protection of infrastructure that was previously economically impractical to secure at the hardware level.

Our architecture also establishes foundations for verifiable compute governance. The challenge-response authentication system we developed could be extended to cryptographically prove how computing clusters are being utilized—for example, proving that hardware is only performing inference rather than training. This capability is essential for international AI agreements that require verification of compute usage restrictions without revealing proprietary model details.

Technical Contributions

From a technical perspective, our work makes several contributions. First, we demonstrate that formal verification workflows can be integrated into AI-assisted development loops without compromising verification rigor. The AI generates code that must still pass the same formal proofs as human-written code, maintaining mathematical guarantees of correctness. Second, we show that hardware feedback loops are practical and valuable for AI systems, enabling debugging capabilities that dramatically reduce development time. Third, we introduce adversarial AI verification as a methodology for improving not just implementations but the formal specifications themselves.

The red team versus blue team approach proved particularly valuable because it identified specification gaps—vulnerabilities that exist because security properties were incompletely specified rather than incorrectly implemented. This is a subtle but important category of security flaw that traditional testing and even formal verification of incomplete specifications would miss.

Limitations and Future Work

Our current implementation has several limitations that future work should address. The bounded model checking approach verifies properties for limited time depths (30+ cycles), while unbounded verification would provide stronger guarantees. The LFSR-based challenge-response authentication is suitable for proof-of-concept but would need to be replaced with cryptographically secure protocols for production use. The system currently lacks defense against physical side-channel attacks such as power analysis or electromagnetic emanation monitoring.

Additionally, while we demonstrated the AI-hardware feedback loop concept, the system currently requires manual observation and description of hardware behavior. Future work should automate this feedback using programmatic UART capture and LED monitoring, creating a fully autonomous development loop.

The formal verification process itself could be enhanced by having AI generate not just implementations but also the formal specifications and verification properties. Currently, these specifications are human-authored, but AI-generated specifications could enable even faster iteration while maintaining verification rigor.

Broader Context

Our work relates to recent advances in both AI-assisted software development and hardware security. The ability of large language models to write and debug code has been well-documented in software contexts, but hardware development presents unique challenges due to parallelism, timing constraints, and the need for formal verification. Our results suggest that these challenges are surmountable and that AI-accelerated hardware development is practical today.

This has implications beyond cybersecurity. The same techniques could accelerate development of safety-critical hardware in aerospace, automotive, medical devices, and other domains where formal verification is required but prohibitively expensive. By making formal verification economically practical, AI could enable a broader shift toward provably correct hardware systems.

In conclusion, Project Gabriel successfully demonstrated that AI can accelerate development of formally verified hardware security systems while maintaining rigorous correctness guarantees. This capability addresses critical needs in AI safety by enabling scalable protection of infrastructure and establishing foundations for verifiable compute governance. While significant work remains to achieve production-ready systems, we have validated the core technical approach and created reusable workflows that can be extended and refined in future research.

References

Microsoft AI Red Team Resources. (2024). *AI Red Team building future of AI*. Microsoft Learn. https://learn.microsoft.com/en-us/security/ai-red-team/

Mindgard. (2025). *31 Best Tools for Red Teaming*. https://mindgard.ai/blog/best-tools-for-red-teaming

NRI. (2025). *5 AI-Powered Cybersecurity Tools You Should Know*. https://www.nri.com/en/media/column/scs_blog/20250826.html

OECD. (2024). *Guide on Privacy Enhancing Technologies*. https://www.oecd.org/en/topics/sub-issues/privacy-enhancing-technologies.html

ProjectDiscovery. (2024). *Nuclei - Fast and Customizable Vulnerability Scanner*. https://projectdiscovery.io/

SymbiYosys Documentation. (2024). *SymbiYosys: Front-end for Yosys-based formal verification flows*. https://symbiyosys.readthedocs.io/

The Alan Turing Institute. (2024). *Awesome Automated Vulnerability Detection*. GitHub Repository. https://github.com/alan-turing-institute/awesome-AVD

Wolf, C. (2024). *Yosys Open SYnthesis Suite*. https://yosyshq.net/yosys/

Zama. (2024). *Concrete ML: Privacy-Preserving Machine Learning*. https://www.zama.ai/concrete-ml

Appendix: Security Considerations

While Project Gabriel successfully demonstrates AI-accelerated development of formally verified hardware security systems, we must carefully consider the limitations and potential vulnerabilities of our current implementation. This appendix outlines these considerations with appropriate caution and conservatism.

Cryptographic Limitations

Our proof-of-concept implementation uses a Linear Feedback Shift Register (LFSR) for challenge-response authentication. While suitable for demonstrating the formal verification workflow, LFSRs are not cryptographically secure and should not be used in production systems. The LFSR output is deterministic and predictable given sufficient observation of the sequence. An attacker who observes multiple challenge-response pairs could potentially reverse-engineer the LFSR state and predict future challenges.

Recommendation: Production implementations should replace the LFSR with cryptographically secure protocols such as HMAC-based challenge-response authentication or public-key cryptography. These should themselves be formally verified using the same workflow we developed.

Bounded Verification Depth

Our formal verification uses bounded model checking with 30+ cycle depth. This proves that security properties hold for all possible inputs within that bounded time window, but does not provide unbounded guarantees. Subtle timing vulnerabilities that manifest only after longer sequences could exist outside the verified bounds.

Recommendation: Extend verification to use unbounded model checking (k-induction) where computationally feasible, or significantly increase bounded depths with more powerful verification hardware. Document the specific verification bounds as part of system security specifications.

Physical Attack Vectors

Our implementation focuses on logical security properties but does not address physical attacks. The system may be vulnerable to:

- Power analysis attacks that infer secret states from power consumption patterns
- Electromagnetic emanation monitoring that captures signal states
- Fault injection attacks using voltage or clock glitching
- Physical access attacks that bypass the FPGA entirely

Recommendation: Future implementations should incorporate power analysis resistance through constant-time operations and power consumption normalization. Critical signals should be encrypted even at the physical layer. The complete threat model must specify which physical attacks are in scope.

Reset and Initialization Security

While we verified normal operation states, reset and initialization sequences warrant additional scrutiny. If an attacker can trigger unexpected resets, they might be able to:

- Force the system into a known initial state
- Bypass authentication by interrupting the authentication process
- Exploit race conditions during initialization

Recommendation: Add comprehensive formal verification of all reset and initialization sequences. Ensure that resets always return the system to a secure default state with authentication required. Implement reset anomaly detection.

Specification Completeness

The adversarial AI testing revealed that initial formal specifications were incomplete. We verified the properties we specified, but some security-relevant behaviors were not initially captured in formal specifications. This is a fundamental challenge: formal verification proves that implementations match specifications, but cannot prove that specifications are complete.

Recommendation: Adopt defense-in-depth approaches where multiple layers of security exist even if one layer's specifications are incomplete. Use adversarial review of specifications, not just implementations. Consider having independent teams write specifications and implementations to avoid shared blind spots.

Side-Channel Vulnerabilities in AI-Generated Code

AI-generated code may inadvertently introduce subtle side-channels that would not be present in security-focused human code. For example, timing variations, data-dependent branching, or unintentional data leakage through shared resources.

Recommendation: All AI-generated security-critical code should undergo security-focused code review by experts familiar with side-channel vulnerabilities. Augment formal verification with timing analysis and information flow analysis tools.

Supply Chain and Toolchain Trust

Our verification relies on the correctness of the verification tools themselves (SymbiYosys, Yosys, Z3 solver). If these tools have bugs, our proofs may be unsound. Additionally, the FPGA toolchain (IceStorm, nextpnr) must correctly implement the verified design.

Recommendation: Use multiple independent verification tools where possible. Consider verified toolchains or add runtime monitoring that checks security properties even in deployed systems. Maintain diversity in verification approaches to reduce common-mode failures.

Scalability to Production Systems

Our proof-of-concept system is relatively small and simple. Production systems would be significantly more complex, with more modules, more security properties, and more intricate timing requirements. It is unclear whether our AI-accelerated verification approach will scale to systems with 10x or 100x more complexity.

Recommendation: Conduct incremental scaling experiments to identify scalability limits. Develop hierarchical verification strategies that verify subsystems independently and then verify their composition. Invest in automated specification generation to reduce the manual burden as systems scale.

Adversarial AI Risks

While we use adversarial AI to improve security, the same techniques could be used by attackers. An adversarial AI with access to our specifications and code could potentially identify vulnerabilities more quickly than human attackers.

Recommendation: Treat all specifications and implementation details as sensitive information. Even for open-source projects, consider keeping detailed formal specifications private until after extensive security review. Implement responsible disclosure practices for any vulnerabilities found through adversarial AI testing.

Verification vs. Validation Gap

Formal verification proves that our implementation matches our specifications, but does not prove that our specifications match the actual security requirements. There may be a gap between what we specified and what is actually needed for security.

Recommendation: Complement formal verification with extensive security testing, penetration testing, and threat modeling. Ensure specifications are derived from comprehensive threat models that consider the full range of attacker capabilities and motivations.

Economic Incentives and Adversarial Pressure

The dramatic cost reduction enabled by AI-accelerated verification may create perverse incentives. Organizations might rush to deploy "formally verified" systems without adequate security review, treating formal verification as a checkbox rather than a meaningful security guarantee.

Recommendation: Educate stakeholders that formal verification is necessary but not sufficient for security. Maintain rigorous security review processes even for formally verified systems. Consider independent security audits as a mandatory step before production deployment.

Conclusion

Project Gabriel demonstrates exciting potential for AI-accelerated development of formally verified hardware security systems. However, our current implementation should be viewed as a research prototype, not a production-ready security solution. The limitations and vulnerabilities outlined above represent important future work. We believe the path forward involves combining AI acceleration with defense-in-depth, multiple independent verification approaches, comprehensive threat modeling, and ongoing security review by domain experts. Only through such conservative and rigorous approaches can we realize the promise of scalable, verifiable hardware security while avoiding the risks of overconfidence in nascent techniques.