POLITECNICO DI TORINO
Repository ISTITUZIONALE

Reproducible Builds as a Security Property for Open FPGA-Based Hardware Security Modules Design

Original
Reproducible Builds as a Security Property for Open FPGA-Based Hardware Security Modules Design / Corvaglia, F., Lentini, N., Mannella, L., Savino, A., Di Carlo, S.. - (2026), pp. 9-18. (ACM CCS TrustAICyberSec '26: Proceedings of the 3rd Workshop on Trusted Computing and Artificial Intelligence applied to Cybersecurity L'Aia (NL) 15-19 November 2026) [10.1145/3847160.3848074].

Availability:
This version is available at: 11583/3017410 since: 2026-11-02T14:15:31Z

Publisher:
ACM

Published
DOI:10.1145/3847160.3848074

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

10 November 2026

Reproducible Builds as a Security Property for Open FPGA-Based Hardware Security Modules Design

Francesco Corvaglia*✉
francesco.corvaglia@polito.it
Politecnico di Torino
Torino, Italy

Niccolò Lentini*
niccolo.lentini@polito.it
Politecnico di Torino
Torino, Italy

Luca Mannella
luca.mannella@polito.it
Politecnico di Torino
Torino, Italy

Stefano Di Carlo
stefano.dicarlo@polito.it
Politecnico di Torino
Torino, Italy

Alessandro Savino
alessandro.savino@polito.it
Politecnico di Torino
Torino, Italy

Abstract

Hardware Security Modules (HSMs), secure elements, and Root of Trusts (RoTs) protect cryptographic operations and secrets, but their hardware artifacts and implementation flows are often difficult to verify independently. This paper argues that reproducibility should be treated as a security property for code-defined hardware, firmware, and software artifacts, and evaluates this claim in the Field-Programmable Gate Array (FPGA) setting.

We present a minimal FPGA-based HSM building block composed of a RISC-V processor, a coherent-sampling oscillator True Random Number Generator (TRNG), and a hardware Secure Hash Algorithm (SHA)-256 module. The design is instantiated with VexRiscv and Ibex to assess portability across open RISC-V control cores. A Nix flake pins the hardware-generation, verification, synthesis-input, and firmware dependencies, enabling auditors to rebuild the generated Register-Transfer Level (RTL), reproduce the OpenTRNG verification workflow, execute firmware-level tests, and validate the SHA-256 module through known-answer tests.

The result is a reproducible and auditable FPGA-based security substrate whose guarantees hold up to the explicit synthesis boundary, where proprietary vendor tooling remains a residual trust assumption.

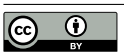
CCS Concepts

• **Security and privacy → Hardware security implementation**; **Embedded systems security**; • **Hardware → Functional verification**; *Hardware validation*; • **Computer systems organization** → *Embedded systems.*

Keywords

Hardware Security Module, Reproducible Hardware, FPGA, RISC-V, True Random Number Generator, Nix, Open Hardware

*Both authors contributed equally to this work.



This work is licensed under a Creative Commons Attribution 4.0 International License.
TrustAICyberSec '26, The Hague, Netherlands
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-3021-4/2026/11
https://doi.org/10.1145/3847160.3848074

ACM Reference Format:
Francesco Corvaglia, Niccolò Lentini, Luca Mannella, Stefano Di Carlo, and Alessandro Savino. 2026. Reproducible Builds as a Security Property for Open FPGA-Based Hardware Security Modules Design. In *3rd Workshop on Trusted Computing and Artificial Intelligence applied to Cybersecurity (TrustAICyberSec '26), November 15–19, 2026, The Hague, Netherlands.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3847160.3848074

1 Introduction

Cryptographic systems ultimately depend on the trustworthiness of the hardware that generates entropy, protects secrets, and executes security-critical operations. Hardware Security Modules (HSMs), secure elements, Trusted Platform Modules (TPMs), and silicon Root of Trusts (RoTs) are therefore widely deployed to move sensitive operations away from general-purpose software. This is especially important in embedded, edge, and Internet of Things (IoT) systems, where devices are resource-constrained, physically exposed, and often used as entry points for attacks against larger infrastructures [2]; such constraints have motivated dedicated protection mechanisms for constrained deployments [5]. To address this need at the hardware substrate level, open hardware RoTs and HSMs isolate cryptographic operations and security-critical state on embedded platforms [7, 18, 23].

However, the hardware components that establish trust are often difficult to inspect independently. Commercial HSMs and secure elements are commonly distributed as black-box devices: their internal microarchitecture, entropy-generation circuitry, synthesis constraints, firmware, and implementation flow are not generally available to end users or external researchers [4, 20]. Certification frameworks such as Federal Information Processing Standards (FIPS) 140-3 [22] establish the baseline security requirements and validation protocols necessary for deploying cryptographic modules in regulated environments. However, compliance alone does not guarantee that an implementation is reproducible from source or verifiable, artifact by artifact, by an independent third party. Users must still trust that the evaluated artifact corresponds to the device they operate.

Open hardware initiatives improve this situation by making security-critical designs inspectable. OpenTitan [18], for example, provides an open-source silicon RoT, while open RISC-V cores such as VexRiscv [33] and Ibex [19] enable researchers to build complete open systems-on-chip without relying on proprietary

embedded processors. Yet the openness of the source code is only one part of the trust problem. An auditor may inspect the Register-Transfer Level (RTL) code of a hardware block and yet be unable to reproduce the synthesized netlist if its generation depends on non-reproducible Electronic Design Automation (EDA) tools or Intellectual Property (IP)-core libraries. Firmware and software face the same issues due to compiler versions and software dependencies. Therefore, open RTL without a reproducible build process has the same limitation as open-source software before reproducible builds: the source can be audited, but the resulting artifact cannot necessarily be verified against it.

This paper uses the term *build* in a broad sense. Hardware and software communities use different terminology, but both define security-relevant systems in terms of source artifacts processed by toolchains. In digital hardware, RTL descriptions, hardware generators, IP blocks, constraints, synthesis tools, and implementation tools produce netlists, Field-Programmable Gate Array (FPGA) bitstreams, or Application-Specific Integrated Circuit (ASIC) implementation artifacts. In firmware and software, source code, libraries, compilers, linkers, and package dependencies produce executable images and binaries. The terminology differs, but the trust question is the same: can an independent party regenerate the artifact from the declared sources and the toolchain and verify that it has not been substituted or silently modified?

This paper argues that reproducibility is itself a security property for code-defined hardware, firmware, and software artifacts. For hardware construction, reproducible flows help bind generated hardware artifacts to their declared source descriptions, applications and toolchains. This mirrors the role of reproducible builds in software, where independent parties verify that a distributed binary corresponds to publicly available source code and a well-defined build environment, reducing reliance on opaque build infrastructure [16].

To achieve reproducibility, we employed Nix [9]: a declarative package manager and build system that enables deterministic build environments through explicit dependency management and isolated builds [8]. We apply these principles to FPGA-based hardware security modules, extending trust from the hardware design to the process that generates the corresponding artifacts. The argument is broader than just FPGA, it extends to hardware, firmware, and software supply chains whose artifacts are generated from source descriptions. This paper focuses specifically on FPGA-based hardware security components, where the boundary is especially visible: RTL, hardware generators, verification inputs, pre-synthesis checks, and firmware images can be captured in a reproducible environment. At the same time, proprietary vendor synthesis and implementation tools remain an explicit residual surface of trust. We implement our proposed approach as a Nix-based hardware-construction workflow (Figure 1) that encompasses hardware generation, verification, synthesis invocation, and firmware, within a reproducible environment, enabling independent parties to regenerate and verify the security-critical artifacts from source. Because commercial FPGA EDA toolchains remain proprietary, the guarantees extend up to the synthesis boundary, an explicit residual trust assumption. We demonstrate the methodology on a minimal HSM comprising an open RISC-V core, a True Random Number Generator (TRNG), and a hardware Secure Hash Algorithm (SHA)-256 module instantiated on both VexRiscv [33] and Ibex [19] to show its reusability across open RISC-V cores.

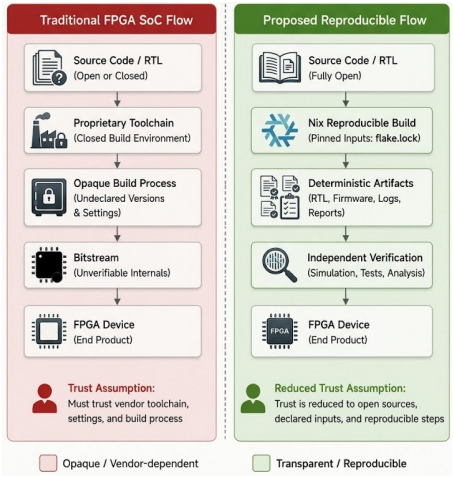


Figure 1: Traditional opaque FPGA flow vs. the proposed reproducible workflow: a Nix-managed environment delivers verifiable, reproducible artifacts (RTL, verification, firmware), reducing reliance on trusted toolchains.

To conclude, this paper provides the following contributions:

- We formulate reproducible construction as a security property for code-defined hardware, firmware, and software artifacts. We instantiate this principle for open FPGA-based HSM building blocks, reducing the build-time trusted computing base up to an explicit synthesis boundary.
- We present the architecture of a minimal, open HSM-oriented System-on-Chip (SoC) that integrates a RISC-V processor, a Coherent Sampling Oscillator (COSO)-based TRNG, and a hardware SHA-256 module.
- We describe a Nix-based build and verification flow that pins the hardware generation, simulation, synthesis inputs, and firmware inputs, and we empirically verify that the generated RTL and firmware images rebuild *bit-for-bit* identically, binding each artifact to its declared source and toolchain.
- We evaluate portability by instantiating the same security peripherals on VexRiscv and Ibex, demonstrating that the verification methodology is independent of the processor integration.

The remainder of the paper covers related work (Section 2); the proposed approach (Section 3), composed of the (i) threat model, (ii) architecture, and (iii) reproducible build and verification flow; the experimental setup (Section 4) and results (Section 5); a discussion (Section 6) and the conclusions (Section 7).

2 Related Work

This work lies at the intersection of open trusted hardware, reproducible build systems, FPGA toolchains, and validation of hardware security primitives. Prior work provides important building blocks in each of these areas, but these threads remain *only partially connected*. To the best of our knowledge, there is no canonical

peer-reviewed work that combines an FPGA-based HSM-oriented SoC, open security primitives, processor portability, and a hermetic reproducible build flow as the main security contribution.

2.1 Open HSMs and Hardware RoTs

HSMs, TPMs, secure elements, and RoTs are commonly used to isolate security-critical operations from general-purpose software. Certification frameworks such as FIPS 140-3 [22] define security requirements for cryptographic modules and remain an important basis for deployment-oriented assurance. However, a certified module may satisfy an evaluation process without allowing third parties to reproduce the implemented hardware artifact or its associated firmware from source.

Open hardware projects have reduced this opacity. OpenTitan [18] is a prominent open-source silicon RoT that provides security-oriented hardware IP, firmware, documentation, and verification infrastructure. Recent work has further studied OpenTitan as a silicon-ready secure element and as a cryptographic offloading component in open SoCs [6, 28]. CrypTech [7] and NetHSM [23] further illustrate practical efforts toward more transparent HSM designs and implementations. Knox [3] is closely related from an assurance perspective since it verifies HSM implementations against functional specifications while accounting for information-preserving refinement.

Our work is complementary to these systems. Indeed, we focus on the **reproducible construction of a minimal FPGA-based HSM** building block and on an independent auditor's ability to inspect and rebuild its hardware and firmware artifacts.

2.2 Reproducible Builds and FPGA Toolchains

Reproducible builds were developed within the software ecosystem to close the gap between source-code availability and trust in generated binaries. The same source-to-artifact trust problem exists in hardware construction: generated RTL, synthesis inputs, firmware images, and verification evidence must correspond to public sources and to a specified build environment [16]. Supply-chain frameworks such as in-toto [34] and Supply-chain Levels for Software Artifacts (SLSA) [24] extend this idea by recording provenance and integrity metadata across build steps. Nix provides a practical mechanism for hermetic builds through a purely functional deployment model in which dependencies are explicit and isolated from machine-local state [8, 9].

Applying these ideas to FPGA security artifacts is non-trivial. FPGA builds depend on a heterogeneous toolchain spanning RTL generation, synthesis, and implementation flows, device constraints, verification tools, and firmware compilation environments. Open FPGA tools such as Yosys [37] and Yosys+nextpnr [32] have made substantial progress toward transparent RTL-to-bitstream flows for selected FPGA families, while Project X-Ray [12] and F4PGA [11] support bitstream-format documentation and open implementation flows for some devices.

In our setting, the final FPGA implementation may still require vendor tooling. Therefore, we treat bitstream reproducibility carefully and distinguish it from reproducible RTL development, pre-synthesis verification, simulation, and firmware compilation.

2.3 Open RISC-V Platforms, Security Primitives, and Verification

Open RISC-V cores provide a practical foundation for transparent embedded security systems. VexRiscv [33] is a configurable FPGA-oriented 32-bit soft core; Ibex [19] is a small open core developed by lowRISC and used in security-oriented designs. Larger ecosystems such as PULPissimo [31] and LiteX [17] demonstrate how open cores can be integrated into reusable SoC platforms. We use VexRiscv and Ibex to evaluate whether the proposed methodology is portable across processors with substantially different microarchitectures, with processor-specific effort confined to the integration layer.

Entropy generation and hashing are the two security primitives at the core of our HSM building block. COSO-style TRNGs [29] use coherent oscillator sampling as a portable FPGA entropy source; National Institute of Standards and Technology (NIST) SP 800-90B [35] defines the corresponding validation requirements, and recent work highlights the importance of pairing open implementations with standards-aware evaluation [30]. SHA-256 is specified by FIPS 180-4 [21] and is part of OpenTitan's openly documented Hash-based Message Authentication Code (HMAC) block [25]; unlike work that focuses on cryptographic offloading and throughput [6, 28], we treat SHA-256 primarily as a correctness-validated component whose integration is part of the reproducible build flow. Open simulation tools—Verilator [36], GHDL [13], and Cocotb [15]—form the verification infrastructure included in our Nix-managed environment.

3 Proposed Approach

This section is structured as follows: Section 3.1 defines the threat model and the resulting design goals; Section 3.2 presents the SoC architecture; Section 3.3 describes the Nix-based reproducible build flow; and Section 3.4 details the layered verification strategy used to evaluate the approach.

3.1 Threat Model and Design Goals

We consider a supply-chain adversary targeting the construction and verification workflow of an FPGA-based HSM. We protect three classes of asset: **(A1)** the build inputs, including source artifacts, build dependencies, and toolchain configuration; **(A2)** the generated artifacts, including the RTL, verification evidence, synthesis inputs, and firmware; and **(A3)** the correspondence between the published sources and the generated artifacts.

An adversary may compromise this correspondence (A3) by modifying source inputs, introducing undeclared dependencies, substituting tool versions, altering build configurations, or providing pre-generated artifacts that cannot be independently reproduced. This threat model follows the observation from reproducible-builds and hardware/software supply-chain research that source availability alone is insufficient to establish trust in generated artifacts [16, 24, 34].

Table 1 structures these threats with the Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege (STRIDE) model [14]. **S**poofing of inputs, **T**ampering with the build, and **R**epudiation of provenance are in scope and map directly to the mitigations developed in this work. By contrast,

Information disclosure, Denial of Service (DoS), and **E**levation of Privilege concern the deployed device at run time, notably physical, side-channel, and FPGA bitstream-readback attacks, and are left to future work.

The goal of this work is to reduce the amount of undocumented infrastructure that must be trusted during hardware construction. The asset model is general to code-defined hardware, firmware, and software supply chains; in this paper, we instantiate it for an FPGA flow and focus on the integrity and reproducibility of artifact generation up to the synthesis boundary, which remains an explicit residual trust assumption.

The following three design goals follow from this threat model:

- *Auditability*: auditors should be able to inspect the RTL, reproduce hardware-generation checks, run simulation-based tests, rebuild firmware, and reproduce the analysis flow for TRNG output.
- *Portability*: the security-peripheral interfaces, validation procedures, and firmware drivers should remain valid across different RISC-V control processors.
- *Reproducibility*: hardware, verification, synthesis-input, and firmware artifacts should be regenerated from declared inputs using a pinned build environment.

3.2 SoC Architecture

The proposed HSM-oriented building block is centered on three hardware-level security components: a RISC-V control processor, a COSO-based TRNG, and a hardware SHA-256 module. Figure 2 shows how the design is instantiated with two different open RISC-V processors, while keeping the security peripherals and validation procedures shared across configurations.

VexRiscv variant. The first configuration uses VexRiscv [33], an FPGA-oriented 32-bit RISC-V soft core generated with SpinalHDL [27]. The processor is connected to an 8 KB Block Random Access Memory (BRAM) and to memory-mapped peripherals through the SoC bus infrastructure. The peripheral map includes a Universal Asynchronous Receiver-Transmitter (UART), a machine timer, the COSO TRNG, and the SHA-256 hardware module. The core runs at 12 MHz and is compiled for the `rv32i/ilp32` target.

The generated RTL references the security peripherals as external instances resolved from the Nix-pinned inputs at build time, tying the SoC RTL to explicit store paths.

Ibex variant. The second configuration uses Ibex [19], a small open-source 32-bit RISC-V core developed by lowRISC and used in security-oriented embedded systems. The Ibex-based SoC uses the same security peripherals but a different Central Processing Unit (CPU) integration wrapper and bus adapter. The SHA-256 base address is kept identical across configurations to reuse the same driver, and an Ibex bus adapter translates the gls sha -256 request-response handshake into the SHA-256 register interface. Firmware targets `rv32imc_zicsr/ilp32`, differing from VexRiscv only in startup (Ibex initializes `mtvec` explicitly).

Shared security-peripheral software. Both configurations share the SHA-256 firmware driver, which implements padding and multi-block chaining without using the C standard library, as well as the self-test interface used for known-answer tests. A shared command exposes the TRNG over UART for offline statistical analysis.

Entropy datapath and conditioning. The COSO entropy source instantiates two free-running ring oscillators on the FPGA fabric, whose relative jitter is sampled and digitized into a raw bit stream. On the FPGA this stream is processed entirely on-chip by the OpenTRNG pipeline: an extended Von Neumann conditioner that removes bias. This bitpacker assembles 32-bit words from independent samples, and online health and alarm monitors. Firmware reads the resulting words from the memory-mapped `DATA` register; both processor configurations enable this on-chip conditioning.

The SHA-256 hardware module is a separate hashing peripheral, not a stage of the entropy datapath. It is exercised by the FIPS 180-4 known-answer self-test (Section 5.3) and by a firmware key-derivation routine that hashes collected TRNG words into a 256-bit key.

3.3 Nix-Based Reproducible Build Flow

To ensure strict reproducibility of evaluation, the entire build and verification environment is managed via Nix *flakes* as shown in Figure 3. In this work, a single `flake.nix` file declares the project's buildable hardware, verification, and firmware outputs, while an accompanying `flake.lock` file records the exact, resolved cryptographic revisions of all external inputs, including the RISC-V processor sources, the TRNG source, the SHA-256 hardware module, and firmware dependencies. Each distinct build step maps to a Nix *derivation*, which executes within a sandboxed, isolated container that receives only its declared inputs, thereby ensuring deterministic, reproducible artifact generation.

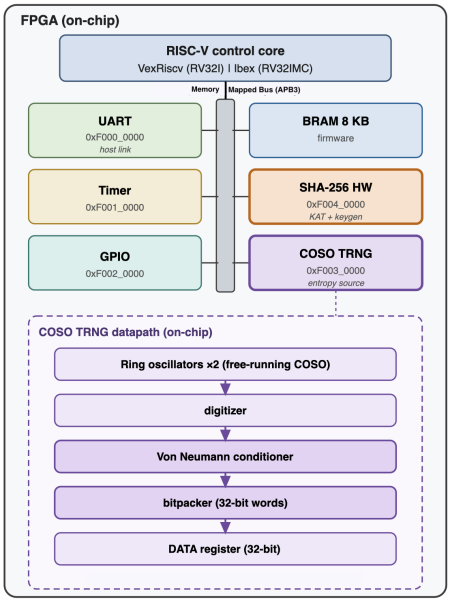


Figure 2: Architecture of the proposed FPGA-based HSM building block. The same TRNG and SHA-256 hardware modules are integrated with two alternative RISC-V control processors, VexRiscv and Ibex, through a memory-mapped peripheral interface.

Table 1: Overview of the STRIDE threat model applied to the reproducible build-and-integration pipeline (assets A1–A3 and the adversary are defined in the text). Tampering is the primary in-scope threat.

STRIDE category	In scope	Threat in the build/integration pipeline	Mitigation in this work
Spoofing	✓	A substituted dependency or compromised upstream masquerades as a legitimate input (A1).	`flake.lock` pins every input by revision and content hash; a substituted input fails the fetch-time hash check. *(Auditability)*
Tampering	✓	Modifying source, a dependency, a tool version, or the build environment to yield a divergent or backdoored artifact (A1 → A2).	Hermetic pinned builds regenerate A2 deterministically; independent rebuild-and-digest comparison (mutation-tested) detects any behavior-changing change. *(Reproducibility, Auditability)*
Repudiation	✓	A builder cannot prove which inputs and toolchain produced a deployed artifact (A3).	`flake.lock` plus the published digests form a verifiable build record binding A2 to A1. *(Auditability)*
Info. disclosure, DoS, elevation of privilege	—	Run-time and physical threats: key/secret leakage, side-channel or bitstream readback, availability, privilege escalation.	Out of scope: future work. Nix sandboxing still isolates each build step.

Listings 1 and 2 show the two structural elements that make the flow reproducible. The `inputs` block (Listing 1) pins each external RTL source to an exact Git commit recorded in `flake.lock`; replacing a Uniform Resource Locator (URL) and re-running `nix flake update` is the only change required to adopt a different processor or security peripheral in an independent design. The RTL-generation derivation (Listing 2) maps those pinned store paths to the environment variables consumed by the SpinalHDL hardware generator, builds within a sandbox that receives only its declared inputs, and captures the generated Verilog as its named output; the firmware derivation follows the same pattern with `make` in place of Gradle.

```
inputs = {
  nixpkgs.url = "github:NixOS/nixpkgs/nixos-unstable";

  # Each entry is pinned by commit hash in flake.lock.
  vexriscv = {
    url   = "github:SpinalHDL/VexRiscv";
    flake = false;
  };
  opentrng = {
    url   = "github:opentrng/ptrng/port_verilog";
    flake = false;
  };
  ...
};
```

Listing 1: `flake.nix` inputs: each RTL dependency pinned by Git commit in `flake.lock`; swapping a URL adopts a different processor or peripheral.

```
pkgs.stdenv.mkDerivation {
  pname = "taurus-soc-rtl";
  src   = ./step1-rtl;   # SpinalHDL Scala sources

  nativeBuildInputs = with pkgs; [
    gradle_8  jdk17  verilator  riscv-toolchain.gcc
  ];

  preConfigure = ''
    # Pinned store paths become environment variables
    # consumed by the hardware generator.
    export VEXRISCV_SRC="${vexriscv}"
    export OPENTRNG_SRC="${opentrng}"
    export SECWORKS_SHA256_SRC="${secworks-sha256}"
  '';

  installPhase = ''
    mkdir -p $out/rtl
    cp Taurus.v $out/rtl/
  '';
}
```

Listing 2: RTL-generation derivation (`rtl-generation.nix`): tools pinned in `nativeBuildInputs`, flake inputs injected as environment variables in `preConfigure`, generated Verilog captured in `installPhase`.

The flake exposes separate packages for RTL generation, simulation, VexRiscv firmware, and Ibex firmware.

Vivado [1] synthesis is invoked through a Nix Filesystem Hierarchy Standard (FHS) wrapper that provides the system environment required by the vendor tool. This wrapper improves repeatability of the synthesis invocation, but Vivado remains an externally installed tool and is not itself built by Nix. Consequently, the reproducibility claim is limited to the Nix-managed artifacts: generated RTL, simulation inputs, pre-synthesis checks, and firmware images. The final bitstream is documented as an output of the pinned RTL and synthesis scripts, but *it is not claimed to be bit-for-bit reproducible.*

Reproducibility in this flow is up to, but not limited to, Vivado: every stage upstream of synthesis is already pinned and hermetically built by Nix, so an auditor who substituted an open-source synthesis toolchain in place of Vivado, and pinned it the same way, would extend the reproducibility guarantee all the way to the bitstream itself.

Table 2 makes this boundary explicit, separating the inputs that are pinned and reproduced from the residual trust surface at and beyond synthesis. Reproducibility thus shrinks the build-time trusted computing base to a small, named set rather than eliminating trust.

3.4 Verification Strategy

The proposed approach achieves verifiability through a layered strategy, each layer producing evidence that an independent auditor can replicate. The implementation is *source-auditable*: the security-peripheral RTL, processor-integration RTL, build scripts, and firmware are released. The build inputs are *pinned and rebuildable*, so each generated artifact can be checked against the process

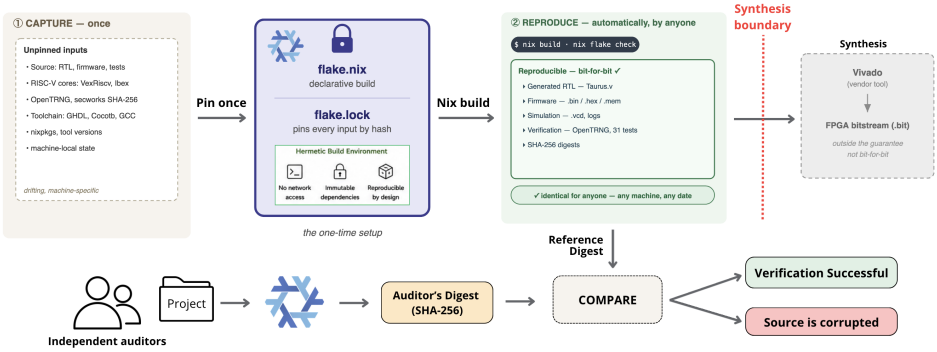


Figure 3: Nix-based reproducible build-and-verification flow. Inputs are pinned once in `flake.lock`; Nix then deterministically regenerates the RTL, verification, simulation, and firmware artifacts, which any auditor can rebuild and check identically (up to the synthesis boundary, beyond which Vivado produces the bitstream).

Table 2: Trusted computing base of the build flow. Pinned inputs are reproduced bit-for-bit; the trusted rows delimit the residual trust surface at and beyond the synthesis boundary.

Build input	Status
RISC-V cores (VexRiscv, Ibex), OpenTRNG RTL, secworks SHA-256	pinned, reproduced
RISC-V GCC, GHDL, Verilator, Cocotb, Python, `nixpkgs`	pinned, reproduced
Vivado (synthesis, place-and-route, bitstream)	trusted
Host OS/kernel, FPGA silicon behavior	trusted

that produced it, not only against the source files that appear to describe it [16]. The entropy source is evaluated *statistically* with the OpenTRNG entropy and autocorrelation tools [26], a reproducible first-level validation, with a full NIST SP 800-90B [35] assessment left as future work. The SHA-256 module is checked *functionally* through FIPS 180-4 [21] known-answer tests, exercised over the full processor-to-peripheral path (register map, driver, padding, and multi-block chaining). Finally, the design is checked for *processor portability* by repeating the whole procedure on both VexRiscv and Ibex. Section 5 reports each layer.

4 Experimental Setup

All experiments target a Xilinx ZCU102 evaluation board (`xczu9eg-ffvb1156-2-e`, Zynq UltraScale+) [38]. Both SoC configurations use a 12 MHz programmable-logic clock. The firmware image is initialized into the on-chip BRAM at synthesis time using `$readmemh`. A UART connection at 115 200 baud provides the interactive test interface, the SHA-256 self-test output, and the TRNG data-dump channel.

The reproducible build environment is evaluated on a workstation running NixOS [10] with Nix 2.18. The `flake.lock` file pins the external hardware, verification, and firmware inputs used by the build. Vivado 2025.2 is installed separately on the host and is invoked through a Nix FHS wrapper for synthesis and implementation. Since Vivado is not a Nix-managed derivation, final bitstream generation is treated as outside the strict reproducibility claim.

Simulation-based verification reuses the OpenTRNG regression suite [26]: eleven Cocotb-based [15] suites, executed with GHDL [13], that cover the components of the entropy-generation pipeline. They run unchanged within the hermetic Nix environment from the RTL and tool versions pinned in `flake.lock`.

Two firmware images are built from the same source tree. The VexRiscv firmware targets `rv32i/ilp32`, while the Ibex firmware targets `rv32imc_zicsr/ilp32`. Runtime functionality that would otherwise depend on external libraries is implemented locally in the firmware to avoid implicit dependencies on the host toolchain or on compiler-provided runtime helpers.

For SHA-256 validation, three FIPS 180-4 known-answer vectors are executed directly on the SHA-256 hardware module through the firmware self-test command. The selected vectors exercise three relevant cases: a short single-block message, the empty-message edge case, and a message that crosses the SHA-256 padding boundary and therefore requires an additional compression block.

For TRNG validation, a capture of approximately 1 Mbit (1 048 512 bits, i.e. 32 766 32-bit words) is collected over UART from the COSO TRNG. The resulting bitstream is then analyzed offline using the OpenTRNG entropy and autocorrelation tools.

5 Experimental Results

The proposed HSM-oriented SoC is evaluated from five complementary perspectives. (i) We demonstrate that the original OpenTRNG [26] verification framework can be reproduced within the proposed Nix-managed environment, and that the pre-synthesis hardware and firmware artifacts regenerate bit-for-bit identically. (ii) We then validate the functionality of the integrated cryptographic peripherals, (iii) evaluate the statistical quality of the entropy source using the same analysis methodology adopted by OpenTRNG, (iv) assess the FPGA implementation results, and, finally, (v) investigate portability across two independent RISC-V processor implementations. Together, these perspectives close the loop with the threat model of Section 3.1. Perspective (i) operationalizes the mitigations against the in-scope spoofing, tampering, and repudiation threats (see Table 1), whereas (ii) to (v) establish that the protected artifact is a functionally correct, statistically characterized, and portable HSM building block.

5.1 Reproducible Verification of the OpenTRNG Framework

The primary contribution of this work is *the reproducible execution of a verification methodology*. To evaluate this claim, we reproduced the complete OpenTRNG verification suite described in Section 4 within the proposed Nix-based build environment without modification. Because the suite runs from the pinned RTL and tool versions, every execution reconstructs an identical verification environment, without relying on machine-local software installations or undeclared dependencies.

All 11 regression suites, 31 tests in total, execute successfully inside the Nix-managed environment. Each suite targets one RTL building block of the pipeline introduced in Section 3.2: COSO, ERO, and MURO are alternative ring-oscillator entropy-source architectures shipped with OpenTRNG, of which COSO is the one integrated in our HSM; Clock Divider and Frequency Counter check the sampling-clock generation and oscillator-monitoring logic feeding the source; Conditioner and Bitpacker validate, respectively, the Von Neumann bias-removal encoder and the assembly of conditioned bits into 32-bit words; Alarm and Online Test check the continuous health monitor and the total-failure threshold alarm built on it; FIFO validates the buffer used internally by the online test's moving cumulative sum; and PTRNG Integration exercises the full source-to-output pipeline end to end. We summarize each suite's target rather than its internal test vectors; the full RTL, testbenches, and per-suite documentation are available in the OpenTRNG repository [26].

Unlike conventional hardware-validation studies, this work shows that an existing open verification framework can be reconstructed from pinned dependencies, allowing independent auditors to reproduce both the hardware and its regression suite using only public inputs. We report only the reproduced pass/fail outcome of each suite; readers seeking the underlying test vectors, assertions, or coverage details are referred to the OpenTRNG project itself [26], which our Nix flake pins and rebuilds rather than reimplements.

5.2 Reproducible Artifact Generation

Reproducing the environment is necessary but not sufficient; the artifacts themselves must be reproducible so an auditor can confirm that each artifact corresponds to the declared source and toolchain. We rebuilt each Nix output in a fresh sandbox with `nix build --rebuild`, which regenerates the derivation and errors on any divergence from the previous result. The SpinalHDL-generated RTL and the two firmware images are rebuilt *bit-for-bit*, confirming that the flow leaks no build timestamps, host paths, or nondeterministic ordering.

Table 3 reports the SHA-256 digests of the principal artifacts because each digest is a deterministic function of the inputs pinned in `flake.lock`, any auditor who rebuilds from the same inputs obtains identical values, binding every artifact to the source and toolchain that produced it. This *bit-for-bit guarantee* holds for all pre-synthesis artifacts; as discussed in Section 3.3, the final bitstream is produced by Vivado and is therefore excluded from the claim.

To confirm that these digests serve as a tamper-evident mechanism rather than a mere fingerprint, we evaluated them using a mutation-testing campaign. Each mutant is a localized change to a declared build input, and is *detected* when it alters the corresponding artifact digest. A parameter mutation of the generated RTL was detected, as were constant-, relational-operator-, and statement-level mutations of the firmware source (for example, the firmware digest moved from `faf089ac...` to `b5f48a53...`). In contrast, an equivalent comment-only mutant correctly left the digest unchanged. Since each output is a deterministic function of the pinned inputs, a behavior-changing modification to the hashed sources cannot go undetected, thereby turning reproducibility into an operational supply-chain check.

Table 3: SHA-256 digests of the pre-synthesis artifacts, each verified to rebuild bit-for-bit via `nix build --rebuild` (last 16 hex; full values regenerate from `flake.lock`).

Artifact	SHA-256 (last 16 hex)
Generated RTL `Taurus.v`	`e995e31009d062b6`
BRAM images (`4×.bin`)	`51689f18098ca64a`
VexRiscv firmware `.bin`	`6ba5224dc90cfbd6`
VexRiscv firmware `.hex`	`40b8ea9805fcde58`
Ibex firmware `.bin`	`dd49b58b5eb26fa4`
Ibex firmware `.mem`	`2cb3aafc399ffe99`

5.3 Functional Validation

While the previous subsection validates the individual OpenTRNG building blocks, the following experiments evaluate the complete integration of the security peripherals within the proposed HSM-oriented SoC.

Firmware-level validation includes known-answer tests for the hardware SHA-256 module based on the FIPS 180-4 test vectors. The tests exercise both single- and multi-block message processing and execute successfully on the VexRiscv and Ibex configurations, validating the hardware module together with the shared firmware driver. Firmware-level tests further validate the integration of the TRNG by exercising its memory-mapped interface, including peripheral configuration, status polling, and random-data acquisition. Together, these experiments confirm the correct interaction between the processors, firmware drivers, and integrated security peripherals.

5.4 TRNG Statistical Validation

Beyond functional correctness, the entropy source is evaluated with the statistical analysis methodology distributed with the OpenTRNG project, executed inside the proposed Nix-managed environment to show that the analysis workflow itself is reproducible.

Two datasets were analyzed. The OpenTRNG software emulator provides a deterministic reference execution with a fixed random seed, allowing the analysis pipeline to be reproduced without FPGA hardware. The board dataset is a capture collected over UART from the COSO TRNG on the ZCU102 board running the Ibex-HSM build. As described in Section 3.2, this on-chip output has already passed through the OpenTRNG Von Neumann conditioner and bitpacker. Because the COSO source is sensitive to placement and routing, and the sampling configuration is build-specific, we scope these

entropy results to the Ibex-HSM build that produced the capture rather than claiming identical entropy across both configurations.

Figure 4 shows the normalized autocorrelation of the board capture: a residual temporal correlation from the physical ring-oscillator behavior survives the on-chip bias-removal conditioning.

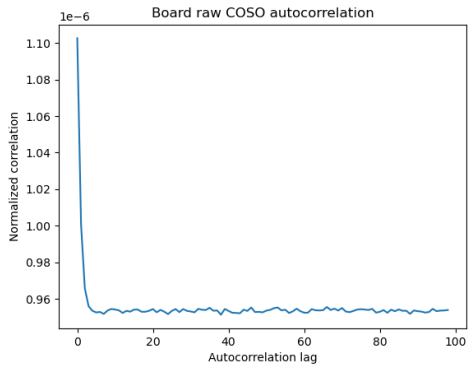


Figure 4: Normalized autocorrelation of the board COSO capture (on-chip conditioned); the elevated lag-1 correlation reflects residual physical ring-oscillator behavior not removed by on-chip conditioning.

Table 4 summarizes the metrics from the OpenTRNG analysis scripts: the emulator is a near-ideal deterministic reference and the board capture shows the expected lag-1 increase from the physical oscillators. The columns capture complementary properties. Shannon and Most Common Value (MCV) are bias-only estimates: both stay near ideal for the raw capture (Shannon = 1.0000, MCV= 0.9929), since the COSO output is essentially unbiased, and neither detects temporal dependence. Markov and lag-1, by contrast, are sensitive to inter-bit dependence: for the raw capture, the Markov min-entropy drops to 0.7899 and lag-1 rises to 0.5780, exposing the dependence that the bias metrics miss. This also explains why the near-ideal emulator shows a marginally lower MCV (0.9625) than the raw board: MCV reflects only the emulator's small residual bias, not the correlation that distinguishes the two datasets.

Table 4: Entropy metrics produced by the reproduced OpenTRNG analysis workflow for the on-chip conditioned board capture and the OpenTRNG software emulator.

Dataset	Shannon	MCV	Markov	Lag-1
Emulator COSO	1.0000	0.9625	0.9830	0.5019
Board (raw)	1.0000	0.9929	0.7899	0.5780

5.5 FPGA Implementation Results

Table 5 summarizes the post-implementation resource utilization and timing results obtained for the two SoC variants on the `xczu9eg-ffvb1156-2-e` FPGA. Both implementations comfortably satisfy the target operating frequency of 12 MHz, with large positive Worst Negative Slack (WNS). Consequently, the operating frequency is determined by the COSO TRNG operating conditions rather than by the processor or cryptographic logic.

The VexRiscv implementation requires fewer Look Up Tables (LUTs) and registers than the Ibex variant, reflecting differences in processor microarchitecture and integration strategy rather than modifications to the shared security subsystem. Both configurations use the same amount of BRAM and identical security peripherals, confirming that the proposed methodology can accommodate different processors without significantly affecting the hardware cost of the security building blocks.

The unconstrained registers reported by Vivado are part of the free-running oscillator domain used by the COSO TRNG. Since these signals intentionally operate asynchronously with respect to the system clock, they are excluded from the synchronous timing analysis and do not affect the design's functional correctness.

Table 5: Post-implementation resource utilization on the `xczu9eg` FPGA. Available resources: 274 080 LUTs, 548 160 FFs, 912 BRAM tiles, and 2 520 DSPs.

Resource	VexRiscv	Ibex
CLB LUTs	1 236 (0.45 %)	5 114 (1.87 %)
CLB Registers	1 720 (0.31 %)	3 577 (0.65 %)
BRAM	4×RAMB18	2×RAMB36
DSPs	0	1
WNS @ 12 MHz	+78.1 ns	+68.3 ns
Estimated F_{max}	≈192 MHz	≈67 MHz

5.6 Processor Portability

To show that the methodology is not tied to a single processor, the same security subsystem is integrated with two independent open RISC-V cores, VexRiscv and Ibex, preserving common security peripherals, verification methodology, firmware Application Programming Interface (API), and build environment.

Table 6 summarizes the resulting porting effort. The COSO TRNG and SHA-256 modules, the OpenTRNG suite, the analysis workflow, the firmware drivers, and the Nix build infrastructure are shared across both configurations, with the security-peripheral RTL sourced from the same sources. The OpenTRNG core is configured per variant, though: the VexRiscv build enables a sampling clock divider and different online-test thresholds than Ibex, so the two entropy sources are not guaranteed identical. Consistent with Section 5.4, we scope the entropy characterization to the build that produced the capture. Processor-specific changes otherwise stay in the integration layer (CPU wrapper, startup code, compiler target, bus adapter).

All functional tests execute unchanged on both configurations. With processor-specific effort confined to the integration layer, the security peripherals, verification procedures, firmware drivers, and reproducible build environment are reused across both SoCs, supporting the claim that the methodology is portable across open RISC-V implementations.

Table 6: Portability comparison between the two SoCs.

Parameter	VexRiscv	Ibex
ISA	RV32I	RV32IMC
RTL origin	SpinalHDL-generated	SystemVerilog
Firmware ABI	`rv32i/ilp32`	`rv32imc_zicsr/ilp32`
Text size	5 093 B	3 881 B
Total footprint	6 128 B / 8 KB	5 040 B / 8 KB
SHA-256 driver	Shared source	
OpenTRNG tests	11/11 suites PASS	
SHA-256 KAT	3/3 PASS	
Pinned Nix inputs	Shared `flake.lock`	
CPU-specific code	Integration layer only	

6 Discussion

This work started from the observation that commercial HSMs are trusted as black boxes, in which microarchitecture, implementation flow, and firmware interfaces are not independently inspectable. Having built and evaluated an open, reproducible FPGA-based alternative, we can state how far the box has been opened and where it remains closed.

Reproducibility survives processor substitution. The central portability result is evidence that the auditability property itself is independent of the control processor (see Table 6). The same security-peripheral RTL, `flake.lock`, OpenTRNG suite, known-answer tests, and firmware driver are reused on both VexRiscv and Ibex, with processor-specific effort confined to the integration layer. This shows that the reproducibility guarantee is a property of the build-and-verification methodology, not of any one core's integration details.

Reproducibility reduces the build-time trusted computing base without closing it entirely. Section 3.1 defines an adversary operating through undeclared dependencies, drifting tool versions, or substituted inputs. The three in-scope STRIDE threats (Table 1) each map to a concrete mitigation: `flake.lock` pinning addresses *spoofing*; the bit-for-bit rebuild (Table 3), corroborated by mutation testing, detects *tampering*; and the lock plus published digests form a build record countering *repudiation*. Reproducing the externally authored OpenTRNG suite (Section 5.1) further shows that the approach composes with third-party hardware-verification artifacts, not only self-authored ones. What must still be trusted is reduced to the small set of Table 2: Vivado, invoked through an FHS wrapper but not built by Nix, could alter the bitstream invisibly to our checks, and the claim certifies the *inputs to* implementation, not the bitstream-to-silicon correspondence. This is the current ceiling of reproducible-builds reasoning for FPGA artifacts, and precisely naming it is part of the contribution.

Reproducibility, certification, and functional verification are complementary, not substitutable. FIPS 140-3 certifies that a module meets fixed requirements under a defined evaluation, but does not require that a third party regenerate the evaluated hardware or firmware artifact from source. Our flow guarantees regeneration but asserts nothing about tamper or side-channel resistance (as reported in Table 1, run-time and physical threats are out of scope). Implementation-verification efforts such as Knox [3] address behavioral correctness with respect to a specification, which is largely orthogonal to building provenance. We position reproducibility as a precondition that strengthens the other two: an unreproducible artifact makes it harder to bind either a certification result or a formal-verification proof to the device actually deployed in the field. Reproducibility does not replace certification; it makes certification's guarantees easier to bind to the artifact under operation.

The entropy *evaluation* is reproducible even though the entropy *source* is not portable. The autocorrelation (Figure 4) and entropy results (Table 4) are obtained by rerunning the OpenTRNG analysis pipeline within the Nix hermetic environment, while fixing the evaluation procedure independently of the fabric. This is deliberately narrower than reproducible entropy: COSO TRNGs are inherently sensitive to placement, routing, and process variation [29], so no build system can make the physical source bit-reproducible across devices. Nix fixes the *criterion*, not the sample; an auditor porting the design obtains legitimately different numbers but measures them against the same procedure used for Table 4. This separation of reproducible methodology over a non-reproducible physical process is the honest boundary for statistical entropy claims, leaving full NIST SP 800-90B [35] validation to future work.

7 Conclusion

This paper presents a reproducible methodology for constructing and verifying an open FPGA-based HSM. By integrating a minimal security-oriented SoC with a Nix-managed build environment, we extend reproducibility to hardware, firmware, and software artifacts in an FPGA setting, enabling independent reconstruction of these components from defined inputs.

The methodology was validated by reproducing the OpenTRNG verification workflow, rebuilding the generated RTL and firmware images bit-for-bit, confirming the integrated security peripherals, and demonstrating compatibility across two open RISC-V cores. These results show that the verification process can be replicated without modifying the security subsystem.

Overall, this work highlights that reproducibility is essential for trustworthy open hardware. While providing public RTL is important, the ability to independently regenerate, verify, and validate these artifacts is crucial for confidence in security-critical systems. This reproducibility guarantee extends beyond Vivado; integrating an open-source synthesis toolchain into the Nix-managed workflow would also ensure reproducibility for the bitstream.

Acknowledgments

This work was supported by Agenzia per la cybersicurezza nazionale under the programme for promotion of XLI cycle PhD research in cybersecurity – E14D25005370006 and by the European Union under the Horizon Europe research and innovation programme, Grant Agreement No. 101298767 (WISE4 — Made-in-Europe, Secure, Resilient, Efficient Open-Source Computing Infrastructure for Web 4.0). Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the Agenzia per la cybersicurezza nazionale, European Union, or the European Commission.

The authors would like to thank Giovanni Nicosia for his contributions to this research during his master's thesis.

References

[1] Advanced Micro Devices, Inc. 2026. *Vivado Design Suite.* AMD. Version 2025.2. https://xilinx.com

[2] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. 2017. Understanding the Mirai Botnet. [Online] Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis. In *26th USENIX Security Symposium (USENIX Security 17), Vancouver, BC, USA, 16–18 August 2017.* USENIX Association, 2560 Ninth St, Berkeley, CA, USA, 1093–1110.

[3] Anish Athalye, M. Frans Kaashoek, and Nickolai Zeldovich. 2022. Verifying Hardware Security Modules with Information-Preserving Refinement. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22).* USENIX Association, Carlsbad, CA, 503–519. https://www.usenix.org/conference/osdi22/presentation/athalye

[4] Marouene Boubakri and Belhassen Zouari. 2025. A Survey of RISC-V Secure Enclaves and Trusted Execution Environments. *Electronics* 14, 21 (2025), 4171. doi:10.3390/electronics14214171

[5] Daniele Canavese, Luca Mannella, Leonardo Regano, and Cataldo Basile. 2024. Security at the Edge for Resource-Limited IoT Devices. *Sensors* 24, 2 (Jan. 2024), 1–16. doi:10.3390/s24020590

[6] Maicol Ciani, Emanuele Parisi, Alberto Musa, Francesco Barchi, Andrea Bartolini, Ari Kulmala, Rafail Psiakis, Angelo Garofalo, Andrea Acquaviva, and Rossi Davide. 2025. Unleashing OpenTitan's Potential: a Silicon-Ready Embedded Secure Element for Root of Trust and Cryptographic Offloading. *ACM Trans. Embed. Comput. Syst.* 24, 5, Article 67 (Sept. 2025), 29 pages. doi:10.1145/3690823

[7] CrypTech Project Contributors. 2026. CrypTech: An Open-Source Hardware Security Module. https://cryptech.is/

[8] Eelco Dolstra. 2006. *The Purely Functional Software Deployment Model.* Ph. D. Dissertation. Utrecht University. https://edolstra.github.io/pubs/phd-thesis.pdf

[9] Eelco Dolstra and Merijn de Jonge. 2004. Nix: A Safe and Policy-Free System for Software Deployment. In *Proceedings of the 18th Large Installation System Administration Conference (LISA '04).* USENIX Association, Atlanta, GA, USA, 79–92. https://www.usenix.org/conference/lisa-04/nix-safe-and-policy-free-system-software-deployment

[10] Eelco Dolstra and Andres Löh. 2008. NixOS: a purely functional Linux distribution. *SIGPLAN Not.* 43, 9 (Sept. 2008), 367–378. doi:10.1145/1411203.1411255

[11] F4PGA Contributors. 2026. F4PGA: Open Source FPGA Toolchain. https://f4pga.org/

[12] F4PGA Contributors. 2026. Project X-Ray: Documenting the Xilinx 7-Series Bitstream Format. https://github.com/f4pga/prjxray

[13] Tristan Gingold and GHDL Contributors. 2024. GHDL: An open-source simulator and compiler for the VHDL language. https://github.com/ghdl/ghdl

[14] Shawn Hernan, Scott Lambert, Tomasz Ostwald, and Adam Shostack. 2006. Threat Modeling: Uncover Security Design Flaws Using the STRIDE Approach. MSDN Magazine. Available online: https://learn.microsoft.com/en-us/archive/msdn-magazine/2006/november/uncover-security-design-flaws-using-the-stride-approach.

[15] Stuart Higgs and Chris Hodgson. 2010. Cocotb: A coroutine-based cosimulation testbench environment for verifying VHDL/Verilog RTL using Python. In *Proceedings of the Design and Verification Conference (DVCon).* Accellera, San Jose, CA, USA, 1–8. https://github.com/cocotb/cocotb

[16] Chris Lamb and Stefano Zacchiroli. 2022. Reproducible Builds: Increasing the Integrity of Software Supply Chains. *IEEE Software* 39, 2 (2022), 62–70. doi:10.1109/MS.2021.3073045

[17] LiteX Contributors. 2026. LiteX: Build Your Hardware, Easily. https://github.com/enjoy-digital/litex

[18] lowRISC CIC and OpenTitan Contributors. 2026. OpenTitan: Open Source Silicon Root of Trust. https://opentitan.org/

[19] lowRISC Contributors. 2026. Ibex: An Embedded 32-bit RISC-V CPU Core. https://ibex-core.readthedocs.io/

[20] Ahmad Moghimi. 2020. *Revisiting Isolated and Trusted Execution via Microarchitectural Cryptanalysis.* Ph. D. Dissertation. Worcester Polytechnic Institute. https://digital.wpi.edu/concern/etds/w9505326n

[21] National Institute of Standards and Technology. 2015. *FIPS 180-4: Secure Hash Standard.* Federal Information Processing Standards Publication 180-4. National Institute of Standards and Technology. doi:10.6028/NIST.FIPS.180-4

[22] National Institute of Standards and Technology. 2019. *FIPS 140-3: Security Requirements for Cryptographic Modules.* Federal Information Processing Standards Publication 140-3. National Institute of Standards and Technology. doi:10.6028/NIST.FIPS.140-3

[23] Nitrokey. 2026. NetHSM: Open Source Hardware Security Module. https://www.nitrokey.com/products/nethsm

[24] OpenSSF. 2026. SLSA: Supply-chain Levels for Software Artifacts. https://slsa.dev/

[25] OpenTitan Contributors. 2026. OpenTitan HMAC Hardware IP Technical Specification. https://opentitan.org/book/hw/ip/hmac/

[26] OpenTRNG Project. 2024. OpenTRNG: An Open-Source Platform for the Design and Evaluation of TRNGs. https://github.com/opentrng/ptrng.

[27] Charles Papon. 2014. SpinalHDL: An open-source high-level hardware description language. https://github.com/SpinalHDL/SpinalHDL

[28] Emanuele Parisi, Alberto Musa, Maicol Ciani, Francesco Barchi, Davide Rossi, Andrea Bartolini, and Andrea Acquaviva. 2024. Assessing the Performance of OpenTitan as Cryptographic Accelerator in Secure Open-Hardware System-on-Chips. In *Proceedings of the 21st ACM International Conference on Computing Frontiers* (Ischia, Italy) *(CF '24).* Association for Computing Machinery, New York, NY, USA, 172–179. doi:10.1145/3649153.3649213

[29] Adriaan Peetermans, Vladimir Rožić, Ingrid Verbauwhede, Ian Sourdis, Christos-Savvas Bouganis, Carlos Alvarez, Lourdes Toledo, Pedro Valero, and Xavier Martorell. 2019. A Highly-Portable True Random Number Generator Based on Coherent Sampling. In *2019 29th International Conference on Field Programmable Logic and Applications.* IEEE, Barcelona, Spain, 1–8.

[30] Valeria Piscopo, Alessandra Dolmeta, Mattia Mirigaldi, Maurizio Martina, and Guido Masera. 2025. A High-Entropy True Random Number Generator with Keccak Conditioning for FPGA. *Sensors* 25, 6, Article 1678 (2025), 16 pages. doi:10.3390/s25061678

[31] Pasquale Davide Schiavone, Davide Rossi, Antonio Pullini, Alfio Di Mauro, Francesco Conti, and Luca Benini. 2018. Quentin: An Ultra-Low-Power PULPissimo SoC in 22nm FDX. In *2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference.* IEEE, Millbrae, CA, USA, 1–3. doi:10.1109/S3S.2018.8640145

[32] David Shah, Eddie Hung, Clifford Wolf, Serge Bazanski, Dan Gisselquist, and Miodrag Milanović. 2019. Yosys+nextpnr: An Open Source Framework from Verilog to Bitstream for Commercial FPGAs. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines.* IEEE, San Diego, CA, USA, 1–4. doi:10.1109/FCCM.2019.00010

[33] SpinalHDL Contributors. 2026. VexRiscv: A FPGA Friendly 32-bit RISC-V CPU Implementation. https://github.com/SpinalHDL/VexRiscv

[34] Santiago Torres-Arias, Hammad Afzali, Trishank Karthik Kuppusamy, Reza Curtmola, and Justin Cappos. 2019. in-toto: Providing Farm-to-Table Guarantees for Bits and Bytes. In *Proceedings of the 28th USENIX Security Symposium.* USENIX Association, Santa Clara, CA, USA, 1393–1410. https://www.usenix.org/conference/usenixsecurity19/presentation/torres-arias

[35] Meltem Sönmez Turan, Elaine Barker, John Kelsey, Kerry A. McKay, Mary L. Baish, and Mike Boyle. 2018. *SP 800-90B: Recommendation for the Entropy Sources Used for Random Bit Generation.* NIST Special Publication 800-90B. National Institute of Standards and Technology. doi:10.6028/NIST.SP.800-90B

[36] Verilator Contributors. 2026. Verilator: Open-Source SystemVerilog Simulator and Lint System. https://verilator.org/

[37] Clifford Wolf. 2013. Yosys: Open Synthesis Suite. https://yosyshq.readthedocs.io/projects/yosys/en/latest/bib.html

[38] Xilinx 2021. *ZCU102 Evaluation Board User Guide (UG1182)* (v1.6 ed.). Xilinx. Featuring the Zynq UltraScale+ MPSoC (`xczu9eg-ffvb1156-2-e`). https://docs.amd.com/v/u/en-US/ug1182-zcu102-eval-bd