NIST Internal Report
NIST IR 8540

Report on Secure Hardware Assurance Reference Dataset (SHARD) Program

Paul E. Black
Vadim Okun

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8540

NATIONAL INSTITUTE OF
STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE

NIST Internal Report
NIST IR 8540

Report on Secure Hardware Assurance Reference Dataset (SHARD) Program

Paul E. Black
Vadim Okun
Software and Systems Division
Information Technology Laboratory

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8540

October 2024

U.S. Department of Commerce
Gina M. Raimondo, Secretary

National Institute of Standards and Technology
Laurie E. Locascio, NIST Director and Under Secretary of Commerce for Standards and Technology

Publication History
Approved by the NIST Editorial Review Board on 2024-09-23

How to cite this NIST Technical Series Publication:
Paul E. Black, Vadim Okun (2024) Report on Secure Hardware Assurance Reference Dataset (SHARD) Program. (National Institute of Standards and Technology, Gaithersburg, MD), NIST IR 8540. https://doi.org/10.6028/NIST.IR.8540

Author ORCID iDs
Paul E. Black: 0000-0002-7561-6614
Vadim Okun: 0000-0003-2391-3681

Abstract

Significant vulnerabilities have been found in chips. Computer programs and methods have been developed to prevent, find, and mitigate them. We proposed Secure Hardware Assurance Reference Dataset (SHARD) as a repository of reference examples (test cases) of both vulnerable and "clean" hardware chip designs. SHARD test cases will enable tool makers to test their chip designing and checking techniques and allow chip designers to evaluate those tools. SHARD received preliminary funding through NIST Information Technology Laboratory's (ITL) Building The Future (BTF) 2024 program. This reports what we achieved with that funding.

Keywords

Hardware assurance; hardware vulnerabilities.

Acknowledgments

We thank A.J. Stein, Noah Masom, and Michael Ogata for their contributions.

Table of Contents

1. Introduction

The 2022 United States Creating Helpful Incentives to Produce Semiconductors (CHIPS) act proposes to vastly reduce the number of errors in digital designs, among other goals. CHIPS aims for far more powerful design checking tools. Our experience in software assurance tools is that developing and evaluating assurance tools requires a large collection of examples with known bugs. We learned much from operating the Software Assurance Reference Dataset (SARD) [1] for almost 20 years.

We aim to establish an analogous collection of buggy hardware designs called the Secure Hardware Assurance Reference Dataset (SHARD): "... there are no publicly available benchmarks that satisfy our requirements." [2]. Additionally, it is valuable to the community to collect examples since web-based resources disappear as people change positions and funded programs end. We were funded for the following steps:

- Learn what formats are used by hardware assurance tools.
- Identify a few good and bad designs.
- Choose a format (or formats) to make available.
- Decide or discover a format for reporting bugs, e.g., line number.
- Decide the scale of entry: circuits? modules? whole chips? chip sets?

We were awarded $50k through NIST Information Technology Laboratory's (ITL) Building The Future (BTF) 2024 program.

This summarizes our success and progress in fiscal 2024.

2. Accomplishments

2.1. Learn About Formats

The first task is to learn what formats and types of chips are most widely used by hardware assurance tools.

The most widely used hardware design languages (HDL) are Verilog and VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language). SystemVerilog is also widely useful. "SystemVerilog ... combines HDLs and a hardware verification language ... plus it takes an object-oriented programming approach. SystemVerilog includes capabilities for testbench development and assertion-based formal verification." [3] Counts of OpenCores [4] projects confirm this: it hosts 467 designs in Verilog and 471 in VHDL. Other languages have fewer than 10. All three languages embody register-transfer level (RTL) abstractions and have standards. VHDL is Institute of Electrical and Electronics Engineers (IEEE) Standard 1076-2008. Verilog is IEEE 1364-2005 and is in the public domain. SystemVerilog is IEEE 1800-2012. "Designing a complex SoC [System on a Chip] would be impossible without these three specialized hardware description languages." [3]

2.2. Choose Format

The next task is to choose the format and design level to make available.

Given the broad acceptance of the top two (or three, depending on how they are counted) format, it is clear that Verilog and VHDL should be supported. SystemVerilog should be supported, too.

Having chosen these, we will support any designs that can be expressed with these languages.

2.3. Decide Bug Representation

Next we must decide or discover a format for reporting bugs, e.g., line number.

Given the choice of languages, line number and error type are good places to start. Recalling what we learned from SARD, we should consider being able to report source, sink, path, and contributing factors of bugs.

Surely existing chip assurance tools have some current format to report say, you have a short circuit here. Maybe Static Analysis Results Interchange Format (SARIF) [5] can be adapted, or maybe there is a SARIF-like format for hardware. This requires further investigation.

2.4. Choose Scale of Cases

An additional task is to decide the scale of entries. That is, should entries be small circuits, functional modules, whole chips, entire chip sets, or some combination?

We have no conclusion that, say, designs must be larger than 100 gates or that designs should not be larger than 1 million gates. For now, we will accept all sizes of designs.

We *have* decided that designs must be all-digital for now; we will not accept designs with analog components.

2.5. Identify Designs

The final task is to identify a few bad (or good) designs.

We found that thousands of designs are available through chip design or chip assurance tool web sites. It remains to collect them, label them, and most importantly make them useful.

Correctness is a high priority in hardware designs: it is nearly impossible[1] to patch an integrated circuit after it is manufactured. Many of the designs are even proven correct.

Because one strong motivation for these sites is *correct* designs, not many designs have (identified) bu gs. To be useful for assurance tool development and evaluation, we must insert errors into designs or generate buggy designs. Insertion may be accomplished with automated tools or manually. Generation might be with a relatively simple macro effort or a full-blown generator, like the Vulnerability Test Suite Generator (VTSG) [7].

Roughly by order of our interest, the sites are

- OpenCores [4], which has about 1000 designs. (The site is no longer actively maintained, and the designs are difficult to access.)
- Trust-Hub [8], which has 2815 designs under Hardware.
- Open Source Hardware Association, which has 2839 projects [9]. (Last update was 2021.)
- Library of Arithmetic Units in VHDL [10] and SystemVerilog [11].

Other sites that have additional designs that we may be able to use are

- NLnet Foundation [12]
- OpenTitan [13]
- Efabless [14]
- Micro-Electronics Security Training Center [15]
- System Security Integration Through Hardware and Firmware (SSITH) program [16]
- Synopsis [17]

Notes or comments on these sites are in App. A.

2.6. Papers

We reviewed many papers to find those that may be helpful. Appendix B is an annotated bibliography of papers that are likely to be useful.

Other papers we reviewed are listed in App. C.

3. Future Tasks

Collect example designs. This task is more than downloading the designs. We must carefully document use rights for each design. Most designs also need to be prepared for use, that is, by documenting existing bugs or injecting bugs.

We need to become familiar with one or more tools. Steps for that are:

[1]Systems can be designed with extra components, such as spare gates or whole processing elements. These extra components are “patched in” to repair erroneous functions or add overlooked security processing. See for example [6].

1. List many hardware assurance tools.
2. Review their inputs and outputs.
3. Investigate if there is a SARIF-like format for bugs.
4. Gain in-group expertise about one, or a few, tools to get a better understanding of what tools report, how they report them, and how useful they are. We will probably select tools that we can use at no cost. For example, Makerchip [18] "provides free and instant access to the latest … open-source tools and proprietary" tools, which support Transaction-Level Verilog https://www.tl-x.org/ that "adds powerful constructs for pipelines and transactions."

Identify a few entities who would use SHARD, that is, users. Get (informal) agreements that they will collaborate.

Develop a searchable database and web application, analogous to SARD.

High-level languages, like Verilog and VHDL, are "compiled" to lower-level representations, such as a netlist. This is analogous to C or Rust being compiled to binary. In the future, we will support designs in netlist format, too.

The full power of quantum computing is only possible through careful design of low-level circuits. More expressive languages will be used for quantum computing. Researchers have not come up with a quantum von Neumann architecture upon which a high-level quantum algorithm language can be based. Some proposed languages are Quipper [19], Scaffold [20], and $\mathcal{Q}$WIRE [21]. A future quantum language may span what we now consider both the software and hardware levels. In short, we must wait to see what quantum language(s) we need in order to support our stakeholders.

References

[1] Software Assurance Reference Dataset (SARD). Accessed 25 April 2024. Available at https://samate.nist.gov/SARD/.

[2] Ahmad H, Huang Y, Weimer W (2022) CirFix: automatically repairing defects in hardware design code. *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '22)*, pp 990–1003. https://doi.org/10.1145/3503222.3507763

[3] Dekker R (2014) What's the difference between VHDL, Verilog, and SystemVerilog?, https://www.electronicdesign.com/resources/whats-the-difference-between/article/21800239/whats-the-difference-between-vhdl-verilog-and-systemverilog.

[4] OpenCores, https://opencores.org/.

[5] (2020) Static analysis results interchange format (SARIF) version 2.1.0, https://docs.oasis-open.org/sarif/sarif/v2.1.0/sarif-v2.1.0.html. Accessed 8 February 2022.

[6] Chean M, Fortes JAB (1990) A taxonomy of reconfiguration techniques for fault-tolerant processor arrays. *Computer* 23(1):55–69. https://doi.org/10.1109/2.48799

[7] Black PE, Mentzer W, Fong E, Stivalet B (2023) Vulnerability test suite generator (VTSG) version 3 (National Institute of Standards and Technology), NIST-IR 8493. https://doi.org/10.6028/NIST.IR.8493
[8] Trust-Hub, https://www.trust-hub.org/.
[9] Open Source Hardware Association Certified open source hardware projects, https://certification.oshwa.org/list.html.
[10] Zimmermann R (1998) VHDL library of arithmetic units, https://iis-people.ee.ethz.ch/~zimmi/arith_lib.html#library.
[11] Scheffler P, Sauter P (2024) Library of arithmetic units, https://github.com/pulp-platform/elau.
[12] (2024) NLnet foundation, https://nlnet.nl.
[13] OpenTitan, https://opentitan.org/.
[14] Efabless, https://efabless.com/.
[15] Micro-electronics security training center, https://mestcenter.org/.
[16] System Security Integration Through Hardware and Firmware (SSITH) program, https://www.darpa.mil/program/ssith/.
[17] Synopsis, https://www.synopsis.com/.
[18] Makerchip, https://www.makerchip.com/.
[19] Valiron B, Ross NJ, Selinger P, Alexander DS, Smith JM (2015) Programming the quantum future. *Commun ACM* 58(8):52–61. https://doi.org/10.1145/2699415
[20] Javadi-Abhari A, Patil S, Kudrow D, Heckey J, Lvov A, Chong FT, Martonosi M (2014) ScaffCC: a framework for compilation and analysis of quantum computing programs. *Proceedings of the 11th ACM Conference on Computing Frontiers* CF ’14 (Association for Computing Machinery, New York, NY, USA). https://doi.org/10.1145/2597917.2597939
[21] Paykin J, Rand R, Zdancewic S (2017) QWIRE: a core language for quantum circuits. *SIGPLAN Notices* 52(1):846–858. https://doi.org/10.1145/3093333.3009894
[22] Leenaars M (2024) Update on libre silicon and OSHW related efforts within NGI and NLnet, https://wiki.f-si.org/index.php?title=2024-Talk-MichielLeenaars. Presented at the Free Silicon Conference (FSiC) 2024.
[23] Introduction to OpenTitan, https://opentitan.org/book/doc/introduction.html.

Appendix A. Web Sites with Possible Designs

This section lists web sites that may serve as sources of d esigns. We add notes or comments about the sites.

OpenCores [4] collected designs for over a decade.

- The site is no longer actively maintained; the latest front page news was 2021. However, the latest module update was April 2024.
- The designs are difficult to access.
- The web site allows search by language: Verilog, VHDL, SystemC, Bluespec, C/C++, and “other”. The designs are divided into many classes, like controllers, crypto chips, and systems.

Trust-Hub[8] is starting to collect designs that can be used to test tools. See https://trust-hub.org/#/hardware/fpga.

Open Source Hardware Association https://stateofoshw.oshwa.org

- Latest update was 2021.
- All of the projects [9] appear to have a link to the hardware “source”, but it may take a bit of link-following to find each one.

Library of Arithmetic Units [10, 11]

- “The library contains various arithmetic operations with multiple architectural choices for different speed requirements. All operations are paramet[e]rized in width and performance grade. This project is still under active development; some parts may not yet be fully functional, and existing interfaces, toolflows, and conventions may be broken without prior notice.”
- Some units are adder, add with carry, subtractor, 2s complement, comparator, multiplier, divide, square root, binary-to-Gray converter, and counter.
- The SystemVerilog site was cited in Sauter and Benz’ 2024 “Achieving Competitive Performance with Open EDA Tools on a 2MGE Open-Source Linux-Capable RISC-V SoC”. https://wiki.f-si.org/index.php?title=Achieving_Competitive_Performance_with_Open_EDA_Tools_on_a_2MGE_Open-Source_Linux-Capable_RISC-V_SoC
- The SystemVerilog version is “Based on the VHDL library written by Reto Zimmermann.”

Michiel Leenaars’ 2024 “Update on libre silicon and OSHW related efforts within NGI and NLnet” [22] has links to many open source hardware projects: look for “devices like”.

“OpenTitan is an open source secure silicon ecosystem producing both silicon IP and complete top-level designs … including … an integrated secure execution environment … OpenTitan is administered by lowRISC CIC …” [23]. The list of standard assets and security measures is at https://opentitan.org/book/doc/contributing/hw/comportability/#security-countermeasures

Efabless [14] is a company that produces small chips. Makers or others that want to make tiny chips can join together on one chip, analogous to ride share launches to space.

- Their GitHub site has 102 public repositories in Verilog. https://github.com/efabless
- OpenLane2 "is an ASIC infrastructure library based on several components including OpenROAD, Yosys, Magic, Netgen, CVC, KLayout and a number of custom scripts for design exploration and optimization". https://github.com/efabless/openlane2
- Tiny Tapeout `https://tinytapeout.com`, which uses Efabless, may be a source of designs. Because of the shared nature of chips, the designs themselves are required to be open source.
- The Open Source Silicon Design Community https://open-source-silicon.dev/ has a Slack channel and has links to the Caravel chip project, SKY130, "a collaboration between Google and SkyWater Technology Foundry to provide a fully open source Process Design Kit (PDK) and related resources ...", and GF180MCU "a collaboration between Google and GlobalFoundries ...".

The Micro-Electronics Security Training (MEST) Center [15] has webinars, on-site and virtual training, on-campus training, certificates, and long- and short-duration courses. It's likely that they have examples of buggy hardware.

Synopsis [17] says they have "... the most advanced chip design, verification, IP integration, and software security and quality solutions ...". They surely have some internal examples for regression testing. They may have examples they use for training. They may be willing to donate them to SHARD.

Appendix B. Annotated Bibliography of Relevant Papers

These papers are likely to be useful. We include comments or notes about each one.

Hammad Ahmad, Yu Huang, and Westley Weimer, "CirFix: automatically repairing defects in hardware design code" [2]

- There were "32 defect scenarios in the authors' testbed". The "benchmark suite of 32 defect scenarios" is available at https://zenodo.org/record/5846419
- The 32 benchmarks are under the "benchmark" subdirectory, which has seven design subdirectories: `decoder_3_to_8`, `flip_flop`, `mux_4_1`, etc. The opencores subdirectory has four design subdirectories: i2c, pairing (`tate_pairing` in the paper), `reed_solomon_decoder`, and sha3. (7 + 4 = 11) Each of these has a "correct" base version of the design, e.g., `decoder_3_to_8.v`, and several versions with "buggy" in the name, such as `decoder_3_to_8_buggy_num.v` and `decoder_3_to_8_super_buggy.v`. (At least one of the buggy versions is not valid: `decoder_3_to_8_kgoliya_buggy1.v` has a comment "This does not compile. Not usable for our experiments.") Bugs are seeded, that is, experts inserted defects "from real-life experience."

- Versions with "tb" in the file name, like `decoder_3_to_8_tb_t1.v`, are part of the Test Bench, that is, a driver that instantiates the design, supplies inputs, and prints outputs.
- Each design subdirectory has a run.sh, which customizes the generic run commands, in `vcs_sim_command`, for each buggy version and runs the commands. Edit the parameters in `run.py`, which is in `/prototype`, to have it save the output. There are other files, such as oracle*.txt, which are expected output.
- CirFix reports how "fit" a fix would be once produced.
- "We chose six projects from undergraduate VLSI courses ... we include a project from each of the key cores listed on the OpenCores [4] website"

Roselyne Chotin and Gabriel Gouvine, "Moosic: Writing a Yosys Plugin for Design for Trust", 2024. Slides at https://wiki.f-si.org/index.php?title=Moosic:_Writing_a_Yosys_Plugin_for_Design_for_Trust

- A Yosys plugin that takes a netlist design and "locks" it by adding extra gates dependent upon a key. Without the key, the circuit produces garbage.
- This may be useful as an obfuscator or to test comparators: does the original netlist and the "locked" netlist compute the same function (when key inputs are set properly)?

Ghada Dessouky, David Gens, Patrick Haney, Garrett Persyn, Arun Kanuparthi, Hareesh Khattri, Jason M. Fung, Ahmad-Reza Sadeghi, and Jeyavijayan Rajendran, "HardFails: Insights into Software-Exploitable Hardware Bugs", 2019. https://www.usenix.org/conference/usenixsecurity19/presentation/dessouky They injected bugs to make 30 buggy RTL-level versions of RISC-V.

Cynthia Sturton, "Hardware is the New Software: Finding Exploitable Bugs in Hardware Designs," January 28, 2019 https://www.usenix.org/conference/enigma2019/presentation/sturton

- They found 5 general bug classes (see slide 26)
- 18 security properties written (see slide 33)
- Total of 25 security properties after mining and looking at other bugs (see slide 42)
- They tracked down 31 actual (production) bugs in a RISC-V design.
- They found "new bugs in the open-source RISC-V and OR1k CPU architectures."

Edu4Chip https://github.com/Edu4Chip is an EU collaboration between many universities. One tape-out per year per school. There is one master or base chip. https://github.com/Edu4Chip/Didactic-SoC

A. Klaiber and S. Chau, "Automatic detection of logic bugs in hardware designs," Proceedings 4th International Workshop on Microprocessor Test and Verification - Common Challenges and Solutions, Austin, TX, USA, 2003, pp. 47-53, doi: 10.1109/MTV.2003.1250262. They verified a piece of hardware they were designing.

Ghada Dessouky, David Gens, Patrick Haney, Garrett Persyn, Arun Kanuparthi, Hareesh Khattri, Jason M. Fung, Ahmad-Reza Sadeghi, and Jeyavijayan Rajendran. " HardFails: insights into software-exploitable hardware bugs", Proceedings 28th USENIX Conference on Security Symposium (SEC'19). USENIX Association, USA, pp. 213-230.

- Video presentation: https://www.usenix.org/conference/usenixsecurity19/presentation/dessouky
- HackDAC is available: https://github.com/hackdac/hackdac_2018_beta It may be difficult to install.

Appendix C. Annotated Bibliography of Papers Unlikely to Help Further

We record these here so we don't waste time studying them again.

Tai-Ying Jiang, C-NJ Liu, and Jing Ya Jou. 2005. "Estimating likelihood of correctness for error candidates to assist debugging faulty HDL designs." 2005 IEEE International Symposium on Circuits and Systems. IEEE, 5682-5685, Vol. 6, doi: 10.1109/ISCAS.2005.1465927 https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1465927

- They give a formula to "estimate likelihood of correctness" of a design. Their experiments were based "on five designs written in Verilog HDL." They mutate them ("arbitrarily change some statements") resulting in 50 mutated designs.
- They make no mention of making the mutants available. As of 2009 Jiang was pursuing a PhD according to https://ieeexplore.ieee.org/author/37279308000

Kai-hui Chang, Ilya Wagner, Valeria Bertacco, and Igor L Markov. 2007. "Automatic error diagnosis and correction for RTL designs". In 2007 IEEE International High-Level Design Validation and Test Workshop. IEEE, 65-72. https://web.eecs.umich.edu/~valeria/research/publications/IWLS07RTLDiag.pdf

- The seven examples were "selected from Open-Cores[4] (Pre norm, MD5, MiniRISC, and CF FFT), the picoJava-II microprocessor (Pipe), DLX, and Alpha. Bugs … were injected into these benchmarks, with the exception of DLX and Alpha, which already included bugs."
- Table 2 lists 21 bugs.
- Many designs have multiple bugs.

Jiann-Chyi Ran, Yi-Yuan Chang, and Chia-Hung Lin. 2003. "An efficient mechanism for debugging RTL description". Proceedings The 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications, 2003. pp 370-373. https://ieeexplore.ieee.org/document/1213064 They experiment "on four designs written in Verilog". Table 2 seems to say that they made 10 variations of each design.

Roderick Bloem and Franz Wotawa. 2002. "Verification and fault localization for VHDL programs". Journal of the Telematics Engineering Society (TIV) 2 (2002), pp 30-33. https:

//www.researchgate.net/publication/240767493_Verification_and_Fault_Localization_in_VHDL_Programs

Stefan Staber, Barbara Jobstmann, and Roderick Bloem. 2005. “Finding and fixing faults.” In Advanced Research Working Conference on Correct Hardware Design and Verification Methods. Springer, pp. 35-49. https://www.semanticscholar.org/paper/Finding-and-fixing-faults-Jobstmann-Staber/e34670945b5be85db9866491510f6f441cfed7f1

- They have three examples in Figs. 6, 7, and 8. They are written in a Verilog-like language.
- They “assume that a (partial) specification is given in linear-time temporal logic (LTL) …”.

J. C. Madre, O. Coudert, and J. P. Billon. 1989. “Automating the diagnosis and the rectification of design errors with PRIAM”, In 1989 IEEE International Conference on Computer-Aided Design. Digest of Technical Papers. 30-33. DOI: 10.1109/ICCAD.1989.76898

Claire Le Goues, Michael Dewey-Vogt, Stephanie Forrest, and Westley Weimer, “A Systematic Study of Automated Program Repair: Fixing 55 out of 105 Bugs for $8 Each” in 2012 34th International Conference on Software Engineering (ICSE). Available at https://clairelegoues.com/assets/papers/legoues12icse.pdf This deals with software, not hardware.