The Self-Hosted Silicon Compiler

Brendan Sechter

2026-07-10

Articles A200 through A203 covered the historical trajectory, the design space, the state of the practice, and the meta-factory manufacturing prior art for hardware description languages and their surrounding ecosystem. Article A201 specifically identified self-hosted synthesis toolchains as an increasingly reachable research direction, enabled by the maturation of Yosys, nextpnr, and F4PGA into production-adjacent open-source flows. This article addresses the concept of a self-hosted silicon compiler as the concrete integration point between the computational side of the reproduction loop that articles A201 and A202 identified.

The phrase self-hosted silicon compiler refers to a system in which the compilation toolchain that turns a hardware description into a device configuration bitstream runs on the hardware that the toolchain produces. The distinction is analogous to the software-compiler tradition in which a language's own compiler is written in that language, compiled by an earlier bootstrap version, and then capable of compiling itself. The stream-based compilers series of articles A188 through A199 developed this pattern in detail for the software case. The hardware case imposes several additional constraints that the software case does not, principally because hardware description compilation depends on physical device geometries that a programmable-fabric target must be able to represent faithfully.

What Self-Hosting Means for a Silicon Compiler

A silicon compiler in its narrow sense translates a hardware description in a language such as Verilog, VHDL, or one of the embedded-domain-specific-language revival languages, into a device-specific bitstream that programs a field-programmable-gate-array or configures an application-specific-integrated-circuit mask set. The translation runs through several intermediate stages including elaboration, synthesis, technology mapping, place-and-route, and bitstream generation. The traditional industrial flow runs these stages on general-purpose central-processing units, typically x86-64 servers running Microsoft Windows or GNU/Linux, using proprietary tooling from AMD, Intel Altera, Synopsys, Cadence, or Siemens Electronic-Design-Automation.

A self-hosted silicon compiler runs the same translation pipeline on hardware that the compiler itself produces. The strongest form of self-hosting runs the toolchain on a soft processor core implemented in the field-programmable-gate-array fabric that the toolchain also targets. Weaker forms of self-hosting run the toolchain on a hard processor integrated with the target fabric in a system-on-chip package, or on a companion processor that shares the programmable-fabric device in a partitioned configuration.

The distinction matters for two reasons. The first reason is dependency reduction. A system that runs its own compilation toolchain on its own hardware depends neither on a proprietary tool vendor nor on external general-purpose processing hardware. The second reason is the reproduction loop that articles A201 and A202 discussed. A meta-factory that manufactures new field-programmable-gate-array devices must be able to generate appropriate bitstreams for the new devices without external tooling support. Self-hosted silicon compilation provides the computational half of this loop.

The Software Bootstrap Precedent

The compiler-tradition literature addresses the self-hosting question considerably. A self-hosted compiler is a compiler written in its own source language, compiled by an earlier bootstrap version, and then capable of compiling itself. The bootstrap procedure begins with a minimal initial compiler written in a different language, often assembly or an earlier target of the same compiler tradition. The initial compiler compiles a simpler version of the target compiler. The simpler version compiles a more sophisticated version. Successive iterations converge on a fixed point where the compiler reproduces its own binary from its own source.

The pattern is canonical across Wirth's Oberon, Rust, Go, and the GNU Compiler Collection. Article A199, the closing article of the stream-based compilers series, developed the fixed-point condition formally as a coalgebraic self-hosting endpoint. The bootstrap sequence provides one specific mechanism for reaching the fixed point, though alternative mechanisms including cross-compilation from an already-hosted compiler also work.

The software bootstrap faces one fundamental security-adjacent concern that the hardware case inherits. Ken Thompson's Turing Award lecture titled Reflections on Trusting Trust demonstrated that a compiler can be modified to insert malicious code into the binaries it generates without any visible source-level trace. If the modified compiler also inserts the same modification into future versions of itself, the malicious behaviour propagates across successive bootstrap cycles even after the original malicious source is removed.

David A. Wheeler formalised a countermeasure called Diverse Double-Compiling in his two thousand nine work. The procedure compiles the target compiler's source twice, once with a trusted alternative compiler and once with the untrusted compiler-under-test. If the two resulting binaries are bit-for-bit identical, the source code accurately represents the untrusted binary and no trusting-trust attack has been undetected. The Wheeler procedure applies to software self-hosting directly and to hardware self-hosting in analogous form, though the hardware case requires adaptation of the bit-identical comparison across the analog-and-digital boundary that device fabrication introduces.

Somlo's Trustworthy Libre Self-Hosting Computer

The strongest existing demonstration of a self-hosted hardware-and-software computing system is Gabriel Somlo's Trustworthy Free Libre Linux-Capable Self-Hosting sixty-four-bit RISC-V Computer at Carnegie Mellon University's Software Engineering Institute. Somlo's project addresses a goal, namely building a free-and-open-source computer from the ground up so that the entire hardware-and-software system's behaviour is one hundred percent attributable to its fully available hardware-description-language and software sources.

The system architecture consists of several composable components. A Rocket Chip RISC-V processor, which is written in Chisel, provides the central processing unit. LiteX provides the rest of the system-on-chip including memory controllers, peripheral interfaces, and interconnect. The design is deployed on a Lattice ECP5 field-programmable-gate-array board. The bitstream is generated from sources by a fully free-libre toolchain consisting of Yosys for synthesis, Project Trellis for ECP5 bitstream documentation, and nextpnr for place-and-route. The software stack runs Fedora Linux on the RISC-V soft processor.

The self-hosting property that Somlo's project demonstrates is considerable. The system is capable of recompiling not only its own software including the Linux kernel, the GNU C Library, and the GNU Compiler Collection, but also its own gateware, namely the field-programmable-gate-array bitstream, completely from source code. The recompilation runs on the RISC-V soft processor that the bitstream itself produces. The self-hosting property therefore holds at the source-to-bitstream level even though it does not extend to the physical silicon fabrication of the underlying Lattice ECP5 device.

The silicon boundary is explicit in Somlo's project design. The programmable-fabric device itself was fabricated by Lattice Semiconductor using proprietary photolithographic processes, foundry equipment, and device masks that Somlo does not control. The self-hosting property holds above the silicon boundary, specifically for everything that is representable as source code that runs on the RISC-V processor or as gateware that runs on the ECP5 fabric. Below the silicon boundary, the device depends on the same proprietary fabrication supply chain that industrial hardware design flows depend on.

The Silicon Boundary

The silicon boundary identifies where existing self-hosting technology ends and where significantly harder research directions begin. Above the silicon boundary, the design is representable as source code and compilable by the toolchain that the design itself implements. Below the silicon boundary, the design depends on physical fabrication processes that require significant industrial infrastructure including photolithographic steppers, chemical vapour deposition equipment, ion implanters, plasma etching systems, and deep-ultraviolet or extreme-ultraviolet light sources. None of these components are representable in any current hardware description language, and none of them can be manufactured by the same hardware description language toolchain that generates the design they will fabricate.

Two distinct research directions approach the silicon boundary from above. The first direction compacts the self-hosting toolchain into the minimum possible implementation that still supports useful hardware description work. The compaction reduces the trust surface, because a smaller toolchain has fewer opportunities for subversion and can be audited more thoroughly. The compaction also reduces the compile-time cost, which matters when the toolchain runs on a soft processor core whose performance is substantially below a general-purpose central processing unit's.

The second direction targets the fabrication process itself. Research work in electron-beam lithography, maskless lithography, and molecular self-assembly suggests that future fabrication processes might be markedly more representable in description languages than current photolithographic processes are. A hypothetical fabrication process whose setup and control were representable in a hardware description language extension could in principle be generated by the same toolchain that generates the device designs that the fabrication process produces. Article A202 addressed the meta-factory prior art that grounds this research direction in substantial engineering literature.

The silicon boundary therefore represents a current technological limit rather than a fundamental theoretical barrier. Extending the self-hosting property below the silicon boundary requires marked advances in fabrication technology that are not principally computer-science research directions. The intersection of computer science and semiconductor process engineering that would be required is extensive but not unprecedented, and the meta-factory literature that article A202 covered provides several starting points for the integration work.

Research Directions Toward Compact Self-Hosting Toolchains

The first of the two research directions identified above, namely compact self-hosting toolchains, has several prototype and research artefacts that demonstrate partial progress.

Compact synthesis toolchains. Yosys is appreciably more compact than the proprietary industrial toolchains that it competes with. The Yosys source is on the order of several hundred thousand lines of C-plus-plus, which is several orders of magnitude smaller than the multi-million-line codebases that Vivado and Quartus Prime represent. The compactness enables Yosys to run on soft processor cores that would not support proprietary toolchains, which is what enables the Somlo project's self-hosting property. Further compaction of Yosys or successor synthesis toolchains represents one active research direction.

Minimal-grammar hardware description languages. A hardware description language with a minimal grammar requires a greatly smaller compiler than a language with a rich grammar. Silice, the hardware description language by Sylvain Lefebvre at the National Institute for Research in Digital Science and Technology in France, provides a considerably simpler grammar than Verilog or VHDL while still supporting useful hardware description work. Silice compiles to Verilog and then to Yosys for synthesis, so it does not itself reduce the toolchain footprint, but its grammar suggests that significantly simpler languages are compatible with useful hardware description.

Compact-toolchain-friendly language design. Article A201 identified the streaming compilation discipline that Keleusma demonstrates in its software-target implementation. The Keleusma language, a total functional stream processor that compiles to bytecode for embedded scripting and high-assurance embedded control contexts, implements a compact compilation toolchain whose worst-case memory usage and worst-case execution time are statically bounded at compile time. Adapting the Keleusma compilation discipline to a hardware-target implementation remains an open research question, and its design-in-progress status means that whether the software analysis passes adapt to a hardware description target is not yet established. If the adaptation succeeded, the resulting hardware description language would support a substantially smaller compilation toolchain than current alternatives, which would advance the compact-toolchain research direction.

On-fabric compilation acceleration. Recent research including graphics-processing-unit-accelerated register-transfer-level simulation suggests that sizable parts of the synthesis pipeline are amenable to hardware acceleration. An accelerator core that implemented the most expensive compilation stages in dedicated hardware running on the same field-programmable-gate-array device as the target design would markedly reduce the compile-cycle time that the soft-processor implementation currently imposes. The research direction combines compact toolchain design with domain-specific hardware acceleration.

Bootstrap procedure design. The bootstrap sequence for a self-hosted silicon compiler requires a minimal initial compiler that generates a minimal initial bitstream capable of running a subset of the target hardware description language. The initial bitstream runs a simpler version of the target compiler, which generates a more sophisticated bitstream, and so on until the sequence reaches the fixed point. The design of the minimal initial compiler and the corresponding initial bitstream is the specific research problem that each self-hosted silicon compiler project must solve. Somlo's project addresses this problem by cross-compilation from an already-hosted GNU toolchain. Somlo references Wheeler's Diverse Double-Compilation procedure as a related mitigation technique for the underlying trust concern, though the integration of Diverse Double-Compilation into the Somlo bootstrap sequence remains a follow-on research direction rather than an implemented component of the current system.

Applications for Self-Hosted Silicon Compilation

The applications for self-hosted silicon compilation concentrate in target contexts where the tooling complexity that the property adds is justified by the specific architectural benefits.

Trust-adjacent computing. The Trusting Trust countermeasure that Wheeler's Diverse Double-Compilation provides depends on the existence of a fully self-hosted toolchain so that the entire binary state can be reproduced from source without external dependencies. Somlo's project targets this application principally in its research framing. Applications for trust-adjacent computing include high-assurance systems where the software supply chain must be auditable end to end, and long-term autonomous systems where the operating environment does not provide external tool vendor support.

Educational applications. A self-hosted silicon compiler running on a soft processor core that students can inspect and modify provides a appreciably more transparent learning en-

vironment than proprietary industrial tools that students cannot audit or extend. The educational context values the source-level attributability of the entire toolchain even when the compile-cycle time is greatly longer than the industrial alternative.

Long-term autonomy contexts. Article A202 discussed the meta-factory prior art for autonomous manufacturing in contexts where external software support cannot be assumed. A self-hosted silicon compiler provides the computational half of the reproduction loop that such systems require. The application is considerably longer-term than the trust-adjacent or educational applications, but the underlying technical requirements are similar, and work that serves the short-term applications directly supports the long-term application as well.

Reproducible builds for hardware. The reproducible-builds movement in software distribution requires that each release binary be reproducible bit-for-bit from its stated source under a specified build environment. Applying the same discipline to hardware description requires a fully self-hosted toolchain whose outputs are bit-for-bit reproducible from source. Somlo's project demonstrates this property at the source-to-bitstream level above the silicon boundary. The reproducible-hardware-builds community is significantly smaller than the reproducible-software-builds community but serves comparable audit-and-verification goals in hardware distribution.

The Meta-Factory Connection

Article A202 addressed the prior art for meta-factories, namely factories whose primary product is other factories. The article covered von Neumann's Universal Constructor, the nineteen eighty NASA studies of lunar self-replicating factories, Freitas and Merkle's two thousand four kinematic self-replicating machines survey, Adrian Bowyer's RepRap project from two thousand five, and industrial digital-twin meta-factory platforms from Hyundai Motor Group and comparable automotive manufacturers.

A self-hosted silicon compiler provides the computational half of the reproduction loop that these meta-factory prior art traditions address. The meta-factory manufactures the physical devices, including the programmable-fabric target that the compiler runs on and the soft processor core that the compiler targets. The self-hosted silicon compiler generates the bitstreams that program the newly manufactured devices to implement the meta-factory's next generation of control systems, signal processing, and sensor interfaces. The two components together close the reproduction loop that each alone does not.

The closure does not imply that the loop is complete without appreciable additional components. Article A202 identified three additional system components that a complete autonomous manufacturing system requires, namely the physical materials refinery that processes raw inputs into usable feedstocks, the kinematic fabricator that converts digital layouts into physical devices, and a meta-cognitive orchestration layer that manages the entire lifecycle. The self-hosted silicon compiler addresses only the integration point between the hardware description tradition and the manufacturing tradition, not the complete autonomous manufacturing system.

The von Neumann probe concept, discussed occasionally in the interstellar-mission speculative literature, provides one motivating application for a fully closed reproduction loop. This article, matching articles A201 and A202, does not develop the interstellar case in detail because the terrestrial applications of self-hosted silicon compilation, including trust-adjacent computing and long-term autonomy contexts, provide substantially more concrete engineering targets than the speculative interstellar case.

Conclusion

The self-hosted silicon compiler represents the specific integration point between the computational and manufacturing halves of the reproduction loop that articles A201 and A202 identified. The concept has considerable existing prior art in the software compiler tradition, which the stream-based compilers series of articles A188 through A199 developed in detail, and significant current-generation demonstration in Gabriel Somlo's Trustworthy Free Libre Linux-Capable Self-Hosting sixty-four-bit RISC-V Computer at Carnegie Mellon University's Software Engineering Institute. Somlo's project demonstrates the self-hosting property above the silicon boundary, specifically for everything that is representable as source code that runs on the RISC-V soft processor or as gateware that runs on the Lattice ECP5 fabric.

The silicon boundary represents the current technological limit of self-hosting rather than a fundamental theoretical barrier. Extending the property below the silicon boundary requires substantial advances in fabrication technology that are principally semiconductor process engineering research directions rather than computer science research directions. The intersection between the two research communities that extending self-hosting would require is marked but not unprecedented, and the meta-factory literature that article A202 covered provides several starting points.

Applications for self-hosted silicon compilation concentrate in trust-adjacent computing, educational contexts, long-term autonomous system applications, and reproducible-builds hardware distribution. The applications are extensive enough to justify continued research investment in compact toolchain design, minimal-grammar hardware description languages, on-fabric compilation acceleration, and bootstrap procedure design. The research directions are principally computer science directions, though several of them extend into adjacent engineering disciplines.

Articles A200 through A203 covered the historical trajectory, the design space, and the state of the practice for hardware description languages. Article A202 covered the meta-factory manufacturing prior art. This article completes the four-article thread by identifying the specific integration point between the computational and manufacturing components of the reproduction loop. The thread does not prescribe a specific implementation approach for the integrated system, because the design space supports multiple implementation approaches, but the thread does record that the underlying prior art across both components is sizable enough to justify active research investment in their integration.

References

Reference

- Bootstrapping in the compiler tradition
- Gabriel Somlo's Trustworthy Libre Self-Hosting RISC-V Computer project
- Keleusma total functional stream processor
- LiteX system-on-chip generator framework
- Project Trellis Lattice ECP5 bitstream documentation
- Rocket Chip RISC-V processor generator
- Silice hardware description language
- Von Neumann probe concept

Related Post

- A History of Hardware Description Languages, article A200 in this blog
- The Design Space for Next-Generation Hardware Description Languages, article A201 in this blog
- The Meta-Factory, Prior Art and the Reproduction Loop, article A202 in this blog

- Hardware Description Languages, the State of the Practice, article A203 in this blog
- The Stream Processor as Compiler and the Compiler as Stream Processor, article A199 in the compilers streaming series

Research

- Wheeler, Fully Countering Trusting Trust through Diverse Double-Compiling, arxiv 2010
- Somlo, Bootstrapping a Libre, Self-Hosting RISC-V Computer, 2021