Invited: Hammer: A Modular and Reusable Physical Design Flow Tool

Harrison Liew
Daniel Grubb
University of California, Berkeley
Berkeley, CA, USA
harrisonliew@berkeley.edu
dpgrubb@berkeley.edu

John Wright*
Amazon
Sunnyvale, CA, USA
johnwright@berkeley.edu

Colin Schmidt*
SiFive
San Mateo, CA, USA
colins@berkeley.edu

Nayiri Krzysztofowicz
University of California, Berkeley
Berkeley, CA, USA

Adam Izraelevitz*
SiFive
San Mateo, CA, USA

Edward Wang*
Massachusetts Institute of Technology
Cambridge, MA, USA

Krste Asanović
University of California, Berkeley
Berkeley, CA, USA

Jonathan Bachrach*
JITX
Berkeley, CA, USA

Borivoje Nikolić
University of California, Berkeley
Berkeley, CA, USA

ABSTRACT

Process technology scaling and hardware architecture specialization have vastly increased the need for chip design space exploration, while optimizing for power, performance, and area. Hammer is an open-source, reusable physical design (PD) flow generator that reduces design effort and increases portability by enforcing a separation among design-, tool-, and process technology-specific concerns with a modular software architecture. In this work, we outline Hammer's structure and highlight recent extensions that support both physical chip designers and hardware architects evaluating the merit and feasibility of their proposed designs. This is accomplished through the integration of more tools and process technologies—some open-source—and the designer-driven development of flow step generators. An evaluation of chip designs in process technologies ranging from 130nm down to 12nm across a series of RISC-V-based chips shows how Hammer-generated flows are reusable and enable efficient optimization for diverse applications.

CCS CONCEPTS

• **Hardware → VLSI system specification and constraints**; **Physical design (EDA)**; • **Software and its engineering → Reusability**; **Abstraction, modeling and modularity**; **Open source model**.

1 INTRODUCTION

Demand for custom silicon has skyrocketed with the proliferation of domains including IoT, AR/VR, and autonomous vehicles. Yet, non-recurring engineering (NRE) costs have exploded with every generation of chips due to technological advancements (e.g., transistor scaling, specialized architectures, and design heterogeneity) and global forces (e.g., trade restrictions and manufacturing shortages). It is therefore important for the semiconductor industry to adapt to these challenges by increasing the productivity of PD flows.

Towards this goal, we proposed Hammer [12], an open-source PD flow generator which demonstrated extensive flow reuse and large reductions in design effort. Since then, Hammer has focused on enhancing architectural design space exploration (DSE), teaching PD in more university courses, and encouraging development by its user base. Hammer's recent integration into the Chipyard framework [2] and expanded plugin library (Table 1) now enable full-stack implementation of RISC-V SoCs. Due to its reusability, Hammer has been used in at least 11 fabricated chips, 4 student courses, and numerous hardware architecture explorations, some of which are evaluated in this work in Sections 3 & 4.

2 OVERVIEW

Hammer abstracts away the intricacies of PD flows into a generic form compatible with many computer-aided design (CAD) tools and process technologies. It is not itself a CAD tool; rather, it is a Python framework that invokes underlying tools with necessary options and generated Tcl scripts to perform actions such as logic synthesis (Fig. 1). Hammer lowers the barriers to learning and executing PD flows, while encouraging flow reusability across all kinds of hardware designs. To achieve this, Hammer's design principles are:

(1) **Separation of concerns**: Hammer decouples concerns specific to tools, technologies, logical design, and physical design from the flow construction itself.
(2) **Standardization**: Hammer codifies a data interchange schema through which design constraints, flow options, and database files can be specified and propagated.
(3) **Modularity**: Hammer defines abstractions that are implemented by interchangeable and shareable plugins for specific tools and process technologies.

*Affiliated with University of California, Berkeley at the time of this work

DAC '22, July 10–14, 2022, San Francisco, CA, USA

ACM ISBN 978-1-4503-9142-9/22/07.
https://doi.org/10.1145/3489517.3530672

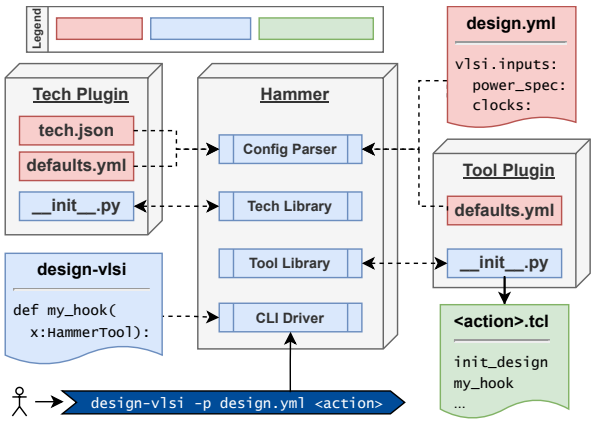


Figure 1: Hammer software architecture

(4) **Incremental adoption**: Hammer flows can mix reusable and custom solutions (e.g. from foundries or tool vendors) as needed to accelerate design and implementation.

These principles ensure that PD intent is encoded in a format that is understandable by other users and applicable to other designs, making Hammer-generated flows increasingly reusable over time. Key components in Hammer's software architecture (Fig. 1) are:

(1) **Hammer intermediate representation (IR)** is the standard configuration data interchange schema. It is serializable in YAML for human readability and JSON for programmatic generation. It uses **metaprogramming**, wherein snippets of IR can use and modify other snippets, transclude files, and much more. Notably, these features expose Hammer IR as an annotation format for higher-level generators (see Sec. 6).

(2) **Hammer tool & tech plugins** implement Hammer's abstractions that encapsulate CAD tool- and process tech-specific concerns. Plugins inherit common methods from core Hammer classes and supply default Hammer IR configurations. Tool plugins define PD flow steps and methods for generating tool-specific Tcl based on design configurations. Tech plugins enumerate PDK source files and also supply default IR. All supported plugins (Table 1) are interoperable, but note that some tech plugins are proprietary.

(3) **Hammer hooks** are Python methods that replace, modify, or add to a tool plugin's default flow steps. Users write hooks to inject Tcl to customize the flow as required by a design. Tech plugins may also specify hooks to automatically include commands needed by a given process technology. Hooks promote agile Hammer development, since users write them to incrementally prototype features, before contributing them to plugins and/or core Hammer for reuse.

(4) The **Hammer driver** is the command-line and Python interface through which a user orchestrates their flow graph. After the user selects plugins, specifies flow steps to execute, and inserts custom hooks, Hammer generates a set of build dependencies for the entire flow graph as a Make fragment. In a hierarchical flow, Hammer builds a flow graph for each submodule in the physical hierarchy and links them together in the correct order of assembly up to the top level.

Table 1: Supported tool (left) and tech (right) plugins

Action	Tool	Foundry	Node
Logic Synthesis	Genus[C], Yosys, Vivado[X], DC[S]	A	16nm FinFET 28nm Planar
Place and Route	Innovus[C], Vivado[X], OpenROAD [1], ICC[S]	B	16nm FinFET 22nm FinFET
DRC/LVS	Calibre[M], ICV[S], Magic/Netgen	C	12nm FinFET 14nm FinFET
Simulation	VCS[S], Xcelium[C]	D	28nm SOI
Power, EM/IR	Joules[C], Voltus[C]	Education	ASAP7 FreePDK45
LEC	Conformal[C], Magic	Skywater	130

[C] Cadence [S] Synopsys [M] Siemens Mentor [X] Xilinx

Additional high-level APIs enable users to specify complex design-specific features with simple Hammer IR inputs. Boundary timing constraints and power domains are generic inputs that Hammer translates into standard constraint file formats like SDC and CPF. Power meshes can be generated from simple target parameters such as track allocation and density. Hammer combines these with stackup information from the selected tech plugin to automatically calculate legal widths and pitches. These APIs are useful for reducing startup overhead and getting sane early physical feedback.

3 ACCELERATING ARCHITECTURE DSE

With deep transistor scaling and compute architecture specialization, it is increasingly important to evaluate the physical feasibility and performance of an architecture at the exploration stage of the design process. Hammer speeds up the creation of custom PD flows, enabling architects to perform more accurate post-place-and-route (P&R) analysis when they may have previously reported post-synthesis metrics only. Below, we study cases of hardware architects with no prior PD experience using Hammer to efficiently evaluate design spaces in multiple process technologies.

3.1 Gemmini Deep-Learning Accelerator

The Gemmini systolic array generator [5] used a Hammer-generated flow for simulation, synthesis, P&R, and power analysis. The designers explored various design points across CPU, systolic array, and scratchpad memory design parameters. Hammer enabled them to rapidly experiment with different floorplans, clock frequencies, and other physical parameters to optimize for timing, area, power density, and energy consumption.

Some of Gemmini's architectural features were constrained or removed as a result of P&R feedback. For instance, routing congestion and power limited systolic array dataflow reconfigurability and informed the insertion of an arbiter between the L2 cache and DMA in order to share memory ports. RTL simulation and physically-aware synthesis—where most architects stop—did not reveal these problems and post-synthesis power estimates were pessimistic by up to 2×. Hammer's flow generation capabilities and IR interface streamlined this exploration by designers with little PD experience.

Gemmini instances were hierarchically placed inside a many-core SoC in Foundry A's 16nm node (HugeFlyingSoC in Table 2) and a heterogenous SoC in Foundry B's 22nm node [6]. Another version of Gemmini containing both floating-point and integer processing elements was included in an SoC in Foundry C's 12nm node (MythicChip in Table 2). The interchangeability of Hammer tech plugins meant the Gemmini designers could efficiently optimize architectural parameters using PD feedback from multiple process technologies.

3.2 Hyperdimensional Computing

An approach to hyperdimensional computing (HDC) is proposed in [9], which drastically reduces power and area by generating hypervectors (HVs) instead of storing them in memory [8]. A Hammer flow in Foundry A's 28nm node was used to perform rapid exploration of the following design spaces:

HV memory: The designers needed to compare HV generation against traditional methods of storing HVs in memory arrays such as SRAMs. Having memory compilation and post-P&R power analysis in a unified flow tool enabled rapid comparison between many memory configurations, resulting in an up to 87.2% reduction in area for HV memory.

Vector folding: HDC presents opportunities to trade off leakage power and fanout with dynamic power, area, and latency by folding (i.e. serializing) the datapath. This design space can be explored by changing PD parameters such as floorplans, clocks, supply voltages, and cell threshold voltages. These parameters are standardized in Hammer IR and encapsulated separately from tool- and tech-specific concerns, hence the Hammer inputs were trivial to programatically generate. This optimization process reduced area by up to 88.6% and leakage power by over two orders of magnitude.

Architecture comparison: A competing design based on a support vector machine (SVM) algorithm was also pushed through the same Hammer PD and analysis flow, providing a proper apples-to-apples (i.e. same tools, process) comparison and lending credence to the 9.5× better energy efficiency claims of HDC over SVM.

Impressively, a single part-time (8 hrs/week) undergraduate student performed most of the DSE over a few months and reduced the headline energy efficiency metric by 93.9% vs. an existing design.

4 FLOW REUSE ACROSS DIVERSE CHIPS

Hammer has been used to tape out a diverse set of chips since those evaluated in [12]. Several noteworthy chips are compared in chronological order in Table 2 and Fig. 2. The most revealing metrics are 1) the portion of unique lines of code (LoC) in Fig. 2 (i.e. design-specific Hammer IR, hooks, and other scripts) and 2) the person-months spent on PD in Table 2, which is the number of designers multiplied by the equivalent number of months working full-time on PD. The drop in both metrics over time despite little application and floorplan commonality (Fig. 3) demonstrates Hammer's increasing reusability. Over the same period, Hammer's codebase—excluding proprietary tech plugins—grew by 3× from 11,000 to 33,000 lines. Several clusters of chips designed simultaneously demonstrate effective reuse:

Eagle, HugeFlyingSoC, and NavRx: HugeFlyingSoC is an evolution of Eagle and was taped out together with the NavRx chip

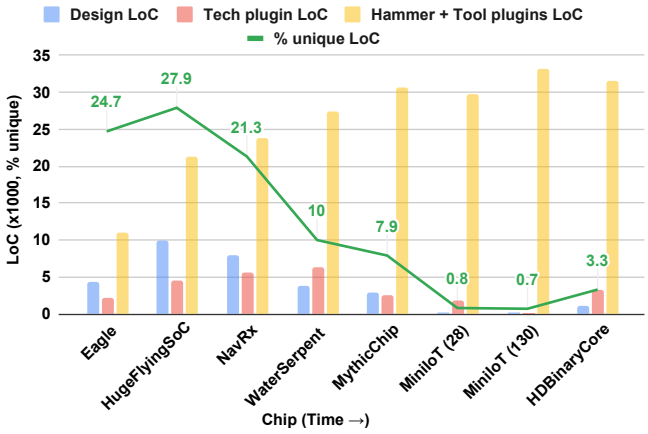
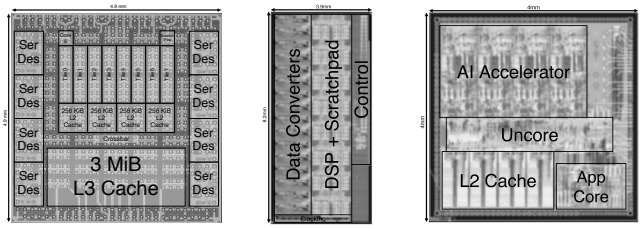


Figure 2: LoC and PD effort trends



(a) Eagle **(b) WaterSerpent** **(c) MythicChip**

Figure 3: Diversity of Hammer-generated chips

one year later. Between Eagle and HugeFlyingSoC/NavRx, 16,000 lines of reusable code were added to the Foundry A 16nm plugin and Hammer, demonstrating how PD intent from Eagle was made generic for subsequent chips.

WaterSerpent and MythicChip: The WaterSerpent chip was taped out in Foundry B's 22nm and drove development of the IC Validator plugin. It was followed by MythicChip, which started in Foundry B's 22nm node before switching to Foundry C's 14nm node and finally its 12nm node. This underscores how users drive Hammer plugin development and switching foundries and nodes is made seamless by plugin interchangeability. Both chips also utilized the hierarchical flow developed for the Eagle chip and were built primarily by one graduate student.

MiniIoT and HDBinaryCore: The MiniIoT chip was designed by undergraduates as part of a course and the HDBinaryCore chip was designed by a single graduate student with little prior experience in PD. Hammer was an ideal tool in these chip designs because flows are easy to compose and already encode PD experience from advanced Hammer users. Again, due to plugin interchangeability, MiniIoT was taped out in SKY130 only a few weeks later.

5 RELATED WORK

5.1 Vendor/Foundry Reference Flows

Tool vendors and foundries provide reference flows for tools and process technologies, which typically consist of Tcl template scripts, but are not modular and require customization for any given design.

Table 2: Comparison of Chips using Hammer

	Eagle [10]	**HugeFlyingSoC**	**NavRx**	**WaterSerpent**	**MythicChip**	**MiniIoT**	**HDBinaryCore**
Description	9-core RISC-V SoC	22-core RISC-V SoC	GPS receiver SoC	MU-MIMO baseband SoC	RISC-V SoC for ML	Bluetooth SoC	HDC processor
Foundry node	A 16nm	A 16nm	A 16nm	B 22nm	C 12nm	A 28nm, SKY130	A 28nm
Die Area (mm^2)	24	56.3	24	32	16	0.33, 7.6	4
Signoff frequency	1.05 GHz	1.05 GHz	500 MHz	2 GHz	1.1 GHz	50 MHz	—
Hierarchy levels	3	3	1	3	2	1	1
Person-months	22	10	6	5	4	8, 1	8

They also lack a framework for applying relevant customizations to other designs, limiting reusability. In contrast, Hammer facilitates initial customization via hooks, which can be made reusable by integration into plugins in a design-agnostic way. Reference flows also do not include safety checks such as input checking, which can cause unexpected failures. Open-source flows like qFlow [4] and OpenLane [11] avoid some of these issues but are not designed to be modular and portable to other tools and technologies.

5.2 SiliconCompiler and mflowgen

SiliconCompiler and mflowgen [3] are the most comparable open-source flow tools to Hammer. SiliconCompiler's strengths are in its API for its data schema and flow graph, parallelized cloud compute, and metrics extraction. However, its data schema does not enforce a separation of concerns and does not codify design inputs as clearly as Hammer IR, which limits flow reusability between designs. mflowgen's strengths are in its flow graph management and modularity via tool step and PDK nodes. However, the modularity is so fine-grained that there is no method to inject common functionality between nodes of a similar flow step in different tools. Both tools do not currently have features analogous to Hammer IR's metaprogramming or Hammer hooks.

6 FUTURE WORK

Hammer auto-generates a full flowgraph as Make dependencies, but future versions will include an interactive flow management interface and robust metrics extraction and post-processing to further enhance DSE. Hammer IR's metaprogramming, while powerful, should be traceable. Keys will be annotated with where they are set, modified, and consumed, and then checked for validity. Hammer will continue adding support for more open-source tools and process technologies, additional PD analysis tools (e.g. aging and manufacturability), and other features such as ECO flows for late design closure. Continuous integration with sample designs will be setup to ensure that plugins remain compatible across all tool and PDK versions. Hammer will continue to deepen its integration within Chipyard, including compiling physical design annotations from Chisel generators, such as abstract floorplan constraints [7].

7 CONCLUSION

In this work, we demonstrate how Hammer helps hardware architects achieve design goals beyond what simulation and synthesis alone can inform, while vastly reducing the PD effort when taping out large SoCs in a variety of process technologies. Hammer's design principles of separation of concerns, standardization, modularity, and incremental adoption combined with open-source availability ensure that it can continuously be improved and maintained by its user base. We hope that this work will enhance a growing ecosystem of agile hardware generators that can lower the NRE cost of implementing increasingly specialized hardware.

ACKNOWLEDGMENTS

DARPA CRAFT under Grant HR0011-16-C0052; NSF CCRI Award 2016662, and BWRC, ADEPT, and SLICE sponsors.

REFERENCES

[1] Tutu Ajayi, Vidya A Chhabria, Mateus Fogaça, Soheil Hashemi, Abdelrahman Hosny, Andrew B Kahng, Minsoo Kim, Jeongsup Lee, Uday Mallappa, Marina Neseem, et al. 2019. Toward an open-source digital flow: First learnings from the openroad project. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–4.

[2] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, et al. 2020. Chipyard: Integrated design, simulation, and implementation framework for custom socs. *IEEE Micro* 40, 4 (2020), 10–21.

[3] Alex Carsello, James Thomas, Ankita Nayak, Po-Han Chen, Mark Horowitz, Priyanka Raina, and Christopher Torng. 2021. Enabling Reusable Physical Design Flows with Modular Flow Generators. *arXiv preprint arXiv:2111.14535* (2021).

[4] R Timothy Edwards, Mohamed Shalan, and Mohamed Kassem. 2021. Real Silicon Using Open-Source EDA. *IEEE Design & Test* 38, 2 (2021), 38–44.

[5] Hasan Genc, Ameer Haj-Ali, Vighnesh Iyer, Alon Amid, Howard Mao, John Wright, Colin Schmidt, Jerry Zhao, Albert Ou, Max Banister, et al. 2019. Gemmini: An agile systolic array generator enabling systematic evaluations of deep-learning architectures. *arXiv preprint arXiv:1911.09925* 3 (2019).

[6] Abraham Gonzalez, Jerry Zhao, Ben Korpan, Hasan Genc, Colin Schmidt, John Wright, Ayan Biswas, Alon Amid, Farhana Sheikh, Anton Sorokin, et al. 2021. A 16mm 2 106.1 GOPS/W Heterogeneous RISC-V Multi-Core Multi-Accelerator SoC in Low-Power 22nm FinFET. In *ESSCIRC 2021-IEEE 47th European Solid State Circuits Conference (ESSCIRC)*. IEEE, 259–262.

[7] Adam Izraelevitz. 2019. *Unlocking Design Reuse with Hardware Compiler Frameworks*. University of California, Berkeley.

[8] Alisha Menon, Anirudh Natarajan, Reva Agashe, Daniel Sun, Melvin Aristio, Harrison Liew, Yakun Sophia Shao, and Jan M. Rabaey. 2021. Efficient emotion recognition using hyperdimensional computing with combinatorial channel encoding and cellular automata. arXiv:2104.02804 [cs.ET]

[9] Alisha Menon, Daniel Sun, Melvin Aristio, Harrison Liew, Kyoungtae Lee, and Jan M Rabaey. 2021. A Highly Energy-Efficient Hyperdimensional Computing Processor for Wearable Multi-modal Classification. In *2021 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, 1–4.

[10] Colin Schmidt, John Wright, Zhongkai Wang, Eric Chang, Albert Ou, Woorham Bae, Sean Huang, Anita Flynn, Brian Richards, Krste Asanović, et al. 2021. 4.3 An Eight-Core 1.44 GHz RISC-V Vector Machine in 16nm FinFET. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 64. IEEE, 58–60.

[11] Mohamed Shalan and Tim Edwards. 2020. Building OpenLANE: a 130nm openroad-based tapeout-proven flow. In *Proceedings of the 39th International Conference on Computer-Aided Design*. 1–6.

[12] Edward Wang, Colin Schmidt, Adam Izraelevitz, John Wright, Borivoje Nikolić, Elad Alon, and Jonathan Bachrach. 2020. A methodology for reusable physical design. In *2020 21st International Symposium on Quality Electronic Design (ISQED)*. IEEE, 243–249.