STAMPS: Fast Synthesis Through Specialized Templates

Kalyan Sriram
University of Wisconsin—Madison
Madison, WI, USA

Joshua San Miguel
University of Wisconsin—Madison
Madison, WI, USA

ABSTRACT

Hardware synthesis aims to lower digital logic descriptions to a specific hardware technology such as an FPGA or standard cells. While improving synthesis quality of results has been studied extensively, the speed of synthesis itself is less explored. At the same time, as the size of FPGAs and ASIC designs continues to increase, synthesis speed has scaled poorly. We believe that the availability of significantly faster methods for hardware synthesis will expose new opportunities for verification, hardware accelerated simulation, and reduce the time needed to implement and validate designs.

We propose STAMPS: Specialized Template Matching for Performant Synthesis. STAMPS uses precomputed templates, inspired by a recently proposed technique called *copy and patch compilation*. We suggest that a high quality of results can be retained while significantly speeding up synthesis.

1 INTRODUCTION

Hardware synthesis lowers digital logic descriptions, usually written at the register transfer level (RTL) of abstraction, to a specific hardware technology (FPGA or standard cells). Traditionally, synthesis incrementally lowers RTL descriptions to simpler structures while performing logic optimization. This optimization aims to improve the *quality of results* (area or throughput) of the synthesized design.

Motivation. This traditional synthesis flow takes substantial time, typically many hours for large designs on FPGAs [7]. This presents a significant barrier to productivity, limiting the "edit and test" iteration speed when testing designs on real hardware. Additionally, very large designs such as multiprocessors are often prohibitively expensive to simulate in software and are instead accelerated entirely or partially in hardware [1, 4]. As FPGAs get larger and hardware designs scale in complexity, improving synthesis speed will expose opportunities for faster verification, accelerated simulation of larger designs, and increased productivity.

Software Compilation. The analogous process of *compilation* translates source to optimized machine code through a similar set of lowering steps. Compilation speed has been extensively studied, suggesting these innovations can be applied to hardware synthesis.

2 RELATED WORK

JITs and Baseline Compilers. Compilation speed has been studied extensively in the context of *just in time compilers* (JITs), which compile source code online and must therefore consider both compilation time and machine code quality. JITs are often classified into two categories — *baseline* compilers, which target generating acceptable machine code as quickly as possible, and *optimizing* compilers, which aim to replace baseline code with faster equivalents.

Copy and Patch Compilation. *Copy and patch compilation* [11] has been recently proposed to improve the performance v. quality pareto frontier in baseline JITs. It uses a library of specialized, precompiled templates to generate acceptable code extremely quickly, and has been shown to generate code two orders of magnitude faster while outperforming other baseline compilers in quality.

Bypassing the entire traditional flow of IR generation, optimization, instruction selection, and assembly, the authors reported a speedup of two orders of magnitude compared to LLVM -O0, and 5x compared to the Liftoff baseline compiler in V8 [11].

JIT Synthesis for FPGAs. Past work has looked at applying JIT methods to FPGA synthesis, with a focus on allowing generic RTL designs to be retargeted to new FPGA architectures on the fly [5, 6].

More recently, coarse-grained predefined blocks have been used to significantly accelerate synthesis with acceptable quality loss [2]. However, the technique requires new designs to be composed of predesigned coarse blocks and is not portable to existing RTL.

3 STAMPS

We propose that this technique of precompiled, specialized templates can be applied to hardware synthesis, and evaluate it to accelerate synthesis to FPGAs.

STAMPS selects fragments of RTL source code, typically a few syntax tree nodes, and builds a library of pre-synthesized netlist fragments *offline* using Yosys. The *online* process of synthesizing a design involves covering a source tree with these netlist fragments using template pattern matching, and connecting the fragments into a final netlist which is run through place and route.

Synthesis Templates. To improve quality of results without an *online* optimization pass, copy and patch relies on specializing templates across semantic variants (data type, stack or register operand, or constant/immediate) to benefit from instruction selection to complex instructions. To limit the number of templates, immediate values and register indices are "patched" into the instantiated machine code fragment. Copy and patch compilation achieves a high quality of results primarily through the following observations:

(1) Templates that cover multiple syntax tree nodes capture the benefits of *instruction selection*, the most impactful optimization during machine code generation, by combining multiple logical operations into a single complex instruction.
(2) Parameterized templates allow better utilization of instruction variants (like register-immediate and register-register operands) instead of selecting a worst case (stack operands).

LATTE '26, March 23, 2026, Pittsburgh, USA

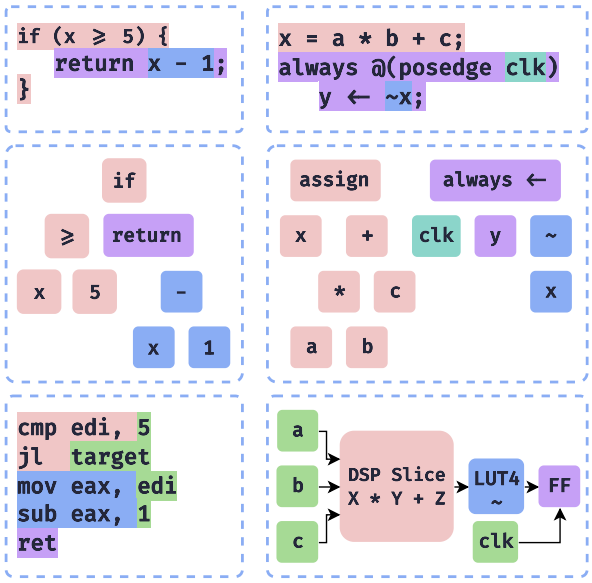

Figure 1: Copy and Patch & STAMPS

Left. copy and patch compilation: source, AST, assembly.
Right. STAMPS synthesis: source, AST, netlist.
Green represents patches, other **colors** represent multi-node templates.

(3) Many optimization gains are observed *locally* within basic blocks and *global* optimization yields diminishing returns.
(4) Copy and patch uses a greedy *register allocation* algorithm to avoid spilling a majority of intermediate values to the stack.

We note that these observations have hardware counterparts:

(1) The equivalent of instruction selection is technology mapping, the process of selecting the most specialized hardware primitive available (FPGA hard block or multi-input standard cell) to cover a subgraph of a logic network.
(2) Synthesis templates can be parameterized on bit-width and operand type to utilize specific hardware resources.
(3) Prior work [8] has performed logic optimization by rewriting small "cuts" of a network, suggesting that optimization benefits to logic networks are also primarily local.
(4) Register allocation does not have a direct hardware equivalent and can be ignored. However, hardware synthesis must also perform place and route, which we discuss later.

Technology Mapping. Our approach identifies common structures (arithmetic, multiplexer trees, memories) in RTL and synthesizes them to technology specific netlists. In order to benefit from good technology mapping, we browse existing patterns in the Yosys *techmap* pass to select templates. For example, Yosys can map fused multiply-add (FMA) operations onto Xilinx DSP slices, suggesting that these syntax fragments should be included directly as library elements to benefit from technology mapping onto DSP hard blocks.

Template Specialization. FPGAs contain hard blocks that efficiently implement common structures, often available in multiple variants. The UltraScale+ architecture has smaller **Block RAMs** and larger **Ultra RAMs**, and Versal has larger fixed point **DSP58** and single precision float **DSPFP32** arithmetic blocks. These can be used effectively by parameterizing library templates in bit width, type, and memory depth.

4 TEMPLATE SELECTION

Template based synthesis can be applied to a wide range of template sizes and complexities, ranging from a small logical expression to an entire IP core such as an AXI controller. However, the choice of template has a significant effect on the quality of the results, synthesis speed, and cost of the template library.

Baseline Coverage. In order to act as a general purpose synthesis tool, STAMPS must be capable of synthesizing any valid input RTL, even if the resulting quality is poor for uncommon hardware constructs. To facilitate this, the template library should contain presynthesized versions of basic logical primitives (all logical operators, flip flops, and multiplexers available in RTL). When falling back to this case, STAMPS imitates a traditional non-optimizing synthesis tool, assembling a large number of small templates into a functionally correct netlist of poor quality.

Improving Quality. In order to win back the quality gap between a baseline synthesis output and that of a fully optimizing flow, larger templates are selected in order to increase the "context window" visible during the offline library generation phase. Since optimization is only performed during initial library generation, larger templates allow an optimizing flow to generate a higher quality netlist for a template than would be possible by stitching together multiple smaller (but logically equivalent) templates. However, the size of the template library increases exponentially with respect to template size. While this can be alleviated by choosing only a sparse set of templates to populate, library size presents an limit on template complexity. Optimal template selection is left to future work.

Synthesizing Large Modules. Past work [2] has explored accelerating synthesis by composing large pre-synthesized "macroblocks", such as FIR filters and other signal processing kernels. As mentioned above, large templates present more opportunities for offline optimization and reduce the number of templates that must be assembled (further increasing synthesis speed). However, these macroblocks are highly specialized and only applicable in certain domains, limiting their value for general purpose synthesis.

5 PRELIMINARY EVALUATION

We show viability of all fundamental claims made, leaving an end to end evaluation for future work.

Template specialization. We synthesize specialized templates using Yosys and observe that technology mapping correctly chooses FPGA hard blocks. We also select a functionally equivalent but poor choice of templates and see that Yosys fails to map effectively, indicating the importance of technology mapping.

Effects of Conventional Optimization. We synthesize the Hazard3 RISC-V core without any logic optimization (*baseline*), with technology mapping to Xilinx (*techmap*), and finally with full optimization (*opt*) and evaluate quality of results. We observe that effective technology mapping accounts for a majority of the increased quality of results (from *baseline* to *techmap*), compared to other optimizations (from *techmap* to *opt*).

6 FUTURE DIRECTIONS

The current goal is to implement an end to end evaluation of template synthesis. We also outline avenues for future work below.

Routing and Bitstream Generation. A substantial amount of total synthesis time is taken by place and route, which takes a technology specific netlist and places it onto a target FPGA fabric while respecting routing resource constraints optimizing for latency. The final place and route solution is used to generate a configuration *bitstream* to program the FPGA.

Template based synthesis can be applied to speed up this process by performing place and route on small designs, followed by bitstream generation. The resulting fragment is relocatable to anywhere on an FPGA fabric, a technique evaluated by past work [2]. Similar to immediates and register indices in software compilation, bitstream fragments can be precomputed and "patched" with constants, routing locations, and operation selects.

Past work has also identified a performance bottleneck in intra-CLB routing [9], suggesting that precomputing these local routes may uncover increased parallelism.

Exact Synthesis. Exact synthesis [10] finds optimal implementation of logic networks, typically using SAT, and has been used to discover rewrite rules for logic optimization [3]. While exact synthesis scales poorly to large inputs, it can be effectively applied on synthesis templates (which have a limited input size) to discover optimal implementations offline. Additionally, only the selected templates need to be synthesized, rather than the much larger number of NPN classes needed for logic rewriting.

7 CONCLUSION

Our work suggests that innovations from software just-in-time compilation can be applied towards accelerating hardware synthesis. Using pre-synthesized templates can bypass the overhead of traditional logic optimization and technology mapping while maintaining an acceptable quality of results. Future work will further evaluate this technique end to end.

REFERENCES

[1] David Biancolin, Sagar Karandikar, Donggyu Kim, Jack Koenig, Andrew Waterman, Jonathan Bachrach, and Krste Asanovic. 2019. FASED: FPGA-Accelerated Simulation and Evaluation of DRAM. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Seaside, CA, USA) *(FPGA '19)*. Association for Computing Machinery, New York, NY, USA, 330–339. https://doi.org/10.1145/3289602.3293894

[2] James Coole and Greg Stitt. 2012. BPR: fast FPGA placement and routing using macroblocks. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis* (Tampere, Finland) *(CODES+ISSS '12)*. Association for Computing Machinery, New York, NY, USA, 275–284. https://doi.org/10.1145/2380445.2380491

[3] Winston Haaswijk, Mathias Soeken, Luca Amarù, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2017. A novel basis for logic rewriting. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. 151–156. https://doi.org/10.1109/ASPDAC.2017.7858312

[4] Sagar Karandikar, Howard Mao, Donggyu Kim, David Biancolin, Alon Amid, Dayeol Lee, Nathan Pemberton, Emmanuel Amaro, Colin Schmidt, Aditya Chopra, Qijing Huang, Kyle Kovacs, Borivoje Nikolic, Randy Katz, Jonathan Bachrach, and Krste Asanović. 2018. FireSim: FPGA-accelerated Cycle-exact Scale-out System Simulation in the Public Cloud. In *Proceedings of the 45th Annual International Symposium on Computer Architecture* (Los Angeles, California) *(ISCA '18)*. IEEE Press, Piscataway, NJ, USA, 29–42. https://doi.org/10.1109/ISCA.2018.00014

[5] R. Lysecky, F. Vahid, and S.D.-X. Tan. 2005. A study of the scalability of on-chip routing for just-in-time FPGA compilation. In *13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05)*. 57–62. https://doi.org/10.1109/FCCM.2005.12

[6] Roman Lysecky, Frank Vahid, and Sheldon X.-D. Tan. 2004. Dynamic FPGA routing for just-in-time FPGA compilation. In *Proceedings of the 41st Annual Design Automation Conference* (San Diego, CA, USA) *(DAC '04)*. Association for Computing Machinery, New York, NY, USA, 954–959. https://doi.org/10.1145/996566.996819

[7] Chandra Mulpuri and Scott Hauck. 2001. Runtime and quality tradeoffs in FPGA placement and routing. In *Proceedings of the 2001 ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays* (Monterey, California, USA) *(FPGA '01)*. Association for Computing Machinery, New York, NY, USA, 29–36. https://doi.org/10.1145/360276.360294

[8] Heinz Riener, Winston Haaswijk, Alan Mishchenko, Giovanni De Micheli, and Mathias Soeken. 2019. On-the-fly and DAG-aware: Rewriting Boolean Networks with Exact Synthesis. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1649–1654. https://doi.org/10.23919/DATE.2019.8715185

[9] Shashwat Shrivastava, Stefan Nikolić, Chirag Ravishankar, Dinesh Gaitonde, and Mirjana Stojilović. 2023. IIBLAST: Speeding Up Commercial FPGA Routing by Decoupling and Mitigating the Intra-CLB Bottleneck. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. 1–9. https://doi.org/10.1109/ICCAD57390.2023.10323897

[10] Mathias Soeken, Winston Haaswijk, Eleonora Testa, Alan Mishchenko, Luca G. Amaru, Robert K. Brayton, and Giovanni De Micheli. 2018. Practical Exact Synthesis. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*. IEEE, Dresden, Germany. Executive Session Paper.

[11] Haoran Xu and Fredrik Kjolstad. 2021. Copy-and-patch compilation: a fast compilation algorithm for high-level languages and bytecode. *Proc. ACM Program. Lang.* 5, OOPSLA, Article 136 (Oct. 2021), 30 pages. https://doi.org/10.1145/3485513