Generate Compilers from Hardware Models!

Gus Henry Smith  
University of Washington  
Seattle, USA  
gussmith@cs.washington.edu

Ben Kushigian  
University of Washington  
Seattle, USA  
benku@cs.washington.edu

Vishal Canumalla  
University of Washington  
Seattle, USA  
vishalc@cs.washington.edu

Andrew Cheung  
University of Washington  
Seattle, USA  
acheung8@cs.washington.edu

René Just  
University of Washington  
Seattle, USA  
rjust@cs.washington.edu

Zachary Tatlock  
University of Washington  
Seattle, USA  
ztatlock@cs.washington.edu

Abstract
Compiler backends\(^1\) should be automatically generated from hardware design language (HDL) models of the hardware they target. Generating compiler components directly from HDL can provide stronger correctness guarantees, ease development effort, and encourage hardware exploration. Past work has already championed this idea; here we argue that advances in program synthesis make the approach more feasible. We present a concrete example by demonstrating how FPGA technology mappers can be automatically generated from SystemVerilog models of an FPGA’s primitives using program synthesis.

ACM Reference Format:

1 Our Position
The semantics of HDLs are very rich. From the advanced type systems of new languages such as Aetherling [9] or Filament [13], to the high-level, algorithmic expressiveness of High Level Synthesis, hardware design languages convey much useful information about the hardware they describe. Even stalwart SystemVerilog and VHDL accurately capture a precise description of how a hardware design functions, including the ability to specify low-level details like latency.

Despite its richness, the HDL description of a hardware design is currently only used by the lowest layers of the software/hardware toolchain. Figure 1 (left) visualizes this. The HDL model of a design is used to build simulators and compile the final fabricated design, but the same model is not used when building higher-level toolchain components such as code optimizers or instruction selectors. Instead, these parts of the toolchain often contain handwritten (and sometimes implicit) models of the target hardware, e.g., a model of the hardware’s memory hierarchy built into the optimizer.

It is our position that compiler backends should be automatically generated from the HDL model of the target hardware. Automatically generating compiler backends (1) provides stronger correctness guarantees as compiler components no longer rely on handwritten, implicit, potentially buggy models of hardware. Instead, compiler backends would rely on the same HDL source from which the hardware is fabricated, guaranteeing that the compiler’s hardware model matches the fabricated hardware. For similar reasons, automatically generating compiler backends (2)

\[^1\]We broadly define a compiler backend as any program that modifies, optimizes, or lowers high-level, hardware-independent code into low-level, hardware-specific code. This broad definition includes software compilers like gcc and libraries like CUDA, but also hardware compilers like FPGA synthesis or High-Level Synthesis (HLS) tools.
reduces compiler development effort by removing the need to build a duplicate hardware model into the compiler backend. Lastly, we believe this approach (3) encourages hardware exploration. By providing more confidence in the correctness of the toolchain and reducing the burden of building a compiler for a new piece of hardware, hardware designers will be emboldened to experiment with new designs.

Furthermore, it is our position that recent advances in programming languages make automated compiler construction feasible. The idea of automatically generating compiler backends is not new: previous work includes synthesizing instruction selectors [3, 6–8] and code generators [4, 11], among other work. However, much of this work is a decade old, if not more, and does not benefit from advances in languages and type systems for hardware [9, 12, 13], equational reasoning via equality saturation [17, 19], program synthesis [15, 18], and machine learning for program generation [1, 2].

To ground our position, we present a concrete example, in which we use SystemVerilog models of FPGA primitives to automatically build technology mappers using modern program synthesis techniques.

2 Generating Technology Mappers

Technology mapping is an FPGA compiler backend step in which a high-level hardware design is lowered to use hardware primitives (small functional blocks) available on the target FPGA. Currently, technology mappers are often implemented as hand-written pattern matchers, which look for patterns in high-level HDL code and rewrite them to instances of FPGA primitives. Some automation does exist; the VTR project [14] seeks to automatically provide compiler backends for hardware given just an architecture description using tools like ABC [5] and ODIN-II [10].

Existing technology mapping approaches—hand-written pattern matchers and automated tools—have a number of weaknesses. They fail to provide strong correctness guarantees: hand-written patterns can be incorrect. They require significant developer effort: when an automated tool cannot support an FPGA primitive, developers must support the primitive by hand. Finally, current tools limit exploration: each new FPGA primitive represents a potentially high cost to support.

We have prototyped a tool which generates technology mappers automatically from the HDL models of the target FPGA. Our tool automatically extracts bitvector semantics from the SystemVerilog models of FPGA primitives provided by each FPGA vendor. We then apply program synthesis, a technique which utilizes SMT solvers to generate programs. We use the bitvector semantics extracted from each primitive’s SystemVerilog model to check—via the application of modern techniques, such as machine learning or equational reasoning via equality saturation. We quantify our approach’s reduction of development effort by listing the primitives automatically imported (and thus supported) by our tool in Table 1. Finally, our approach can encourage the exploration of new FPGA primitives, by quickly generating technology mappers for hardware prototypes during the development process.

<table>
<thead>
<tr>
<th>FPGA</th>
<th>Primitive</th>
<th>SystemVerilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xilinx Ultrascale+</td>
<td>LUT6</td>
<td>88</td>
</tr>
<tr>
<td></td>
<td>CARRY8</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>DSP48E2</td>
<td>1426</td>
</tr>
<tr>
<td>Lattice ECP5</td>
<td>LUT2</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>LUT4</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>CCU2C</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>ALU24B</td>
<td>672</td>
</tr>
<tr>
<td></td>
<td>MULT18X18D</td>
<td>985</td>
</tr>
<tr>
<td>SOFA [16]</td>
<td>frac_lut4</td>
<td>69</td>
</tr>
<tr>
<td>Intel Cyclone</td>
<td>altmult_accum</td>
<td>1460</td>
</tr>
</tbody>
</table>

Table 1. FPGA primitives imported automatically (and thus available for technology mapping) from vendor-provided SystemVerilog models, with source lines of code of the original SystemVerilog models.

3 Conclusion and Future Directions

We have argued that compiler backends should be automatically generated from the HDL models of the hardware they target. Furthermore, we provided a concrete demonstration of this idea via a prototype tool which generates FPGA technology mappers given the SystemVerilog models of an FPGA’s hardware primitives.

We call on others in the field to revive this idea with us via the application of modern techniques, such as machine learning or equational reasoning via equality saturation.
References


