Generate Compilers from Hardware Models!

Gus Smith, PhD Candidate, University of Washington
PLARCH 2023
The Hardware Lottery

Sara Hooker

Google Research, Brain Team
shooker@google.com
Hardware and compilers have a disproportionate role in deciding which research ideas succeed or fail.
Hardware and compilers have a disproportionate role in deciding which research ideas succeed or fail.

Takeaway for our community: new platforms (hardware + compiler stack) should be easier to build, so that the best ideas win!
Hardware and compilers have a disproportionate role in deciding which research ideas succeed or fail.

Takeaway for our community: new platforms (hardware + compiler stack) should be easier to build, so that the best ideas win!
Hardware and compilers have a disproportionate role in deciding which research ideas succeed or fail.

Takeaway for our community: new platforms (hardware + compiler stack) should be easier to build, so that the best ideas win!
Why are compilers hard to build?
Why are compilers hard to build?
Why are compilers hard to build?
Why are compilers hard to build?
Why are compilers hard to build?
Why are compilers hard to build?
Why are compilers hard to build?
Why are compilers hard to build?
Why are compilers hard to build?
Why are compilers hard to build?
Why are compilers hard to build?
Why are compilers hard to build?
Why are compilers hard to build?
Why are compilers hard to build?

- Programs compiled by Compiler
- Compiler onto Hardware Design
- Docs informs
- Hardware Design builds onto Compiler
- Compiler builds onto Programs
- Programs builds onto Hardware Design
- Hardware Design talks with Compiler
Why are compilers hard to build?

Formal Verification Model

Programs

compiled by

Compiler

onto

Hardware Design

builds

builds

builds

informs

informs

informs

talks with

Docs
Why are compilers hard to build?
Why are compilers hard to build?
Why are compilers hard to build?

- Formal Verification Model builds
- Programs compiled by
- Talks with
- Models
- Verifies
- Talks with
- Hardware Design builds
- Compiler compiles
- Talks with
- Docs informs
- Compiled by
- Inform
Why are compilers hard to build?
Why are compilers hard to build?

- Formal Verification Model builds Hardware Design
- Programs compiled by Compiler
- Docs informs
- Hardware Design builds
- Compiler verifies
- Docs models
- Compiler informs
- Docs builds
- Compiler talks with
- Hardware Design builds
- Programs informs
Why are compilers hard to build?
Why are compilers hard to build?

Formal Verification Model builds onto Hardware Design

Compiler compiles Programs

Docs informs Programs

Programs compiled by the Compiler

Hardware Design builds onto Compiler

Compiler verifies models

Programs talks with Formal Verification Model

Why are compilers hard to build?
Why are compilers hard to build?
Why are compilers hard to build?

gcc: 1000s of contributors over 35+ years  
Yosys: 200+ contributors over 10+ years  
TVM: 800+ contributors over 7 years
Why are compilers hard to build?
Why are compilers hard to build?
Why are compilers hard to build?
Why are compilers hard to build?

Finding and Understanding Bugs in C Compilers
Xuejun Yang Yang Chen Eric Eide John Regehr
University of Utah, School of Computing
{jxyang, chenyang, eode, reeghr}@cs.utah.edu

Finding and Understanding Bugs in FPGA Synthesis Tools
Yann Herklotz
yann.herklotz15@imperial.ac.uk
Imperial College London
London, UK
John Wickerson
j.wickerson@imperial.ac.uk
Imperial College London
London, UK

A Comprehensive Study of Deep Learning Compiler Bugs
Qingchao Shen
College of Intelligence and Computing, Tianjin University
School of New Media and Communication, Tianjin University
China
qingchao@tju.edu.cn

Haoyang Ma
College of Intelligence and Computing, Tianjin University
China
haoyang_9804@tju.edu.cn

Junjie Chen
College of Intelligence and Computing, Tianjin University
China
junjiechen@tju.edu.cn

(Csmith) (Verismith)
Why are compilers hard to build?
Why are compilers hard to build?
Why are compilers hard to build?
Why are compilers hard to build?
Building a compiler requires significant engineering effort and induces numerous bugs.
Building a compiler requires significant engineering effort and induces numerous bugs. Those costs are multiplied with every new hardware design.
Why are compilers hard to build?

Building a compiler requires significant engineering effort and induces numerous bugs.

Those costs are multiplied with every new hardware design.

What if compilers were synthesized? (i.e., automatically generated?)
Why are compilers hard to build?

What if compilers were synthesized?
Why are compilers hard to build?

What if compilers were synthesized?
Why are compilers hard to build?
What if compilers were synthesized?
Why are compilers hard to build?

What if compilers were synthesized?
Why are compilers hard to build?

What if compilers were synthesized?
Why are compilers hard to build?

What if compilers were synthesized?

Programs compiled by Compiler

Compiler generates onto Compiler Generator

Compiler Generator reads by

Hardware Design builds

builds

Why are compilers hard to build?

What if compilers were synthesized?
Why are compilers hard to build?

What if compilers were synthesized?
What if compilers were synthesized?

Why are compilers hard to build?
What if compilers were synthesized?

Why are compilers hard to build?
What if compilers were synthesized?
What if compilers were synthesized?

Why are compilers hard to build?

The Hardware Lottery
Why are compilers hard to build?

What if compilers were synthesized?
Automatically generating compilers can reduce engineering effort and eliminate bugs.
Automatically generating compilers can reduce engineering effort and eliminate bugs. Furthermore, the approach scales with new hardware designs, thus fighting against the hardware lottery!
Compilers should be generated from formal models of hardware.
Compilers should be generated from formal models of hardware.

With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.
Generating Compilers → Why Now? → Case Study: Lakeroad → Call to Action
Generating Compilers → Why Now? → Case Study: Lakeroad → Call to Action
Compilers should be generated from formal models of hardware.

With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.
Compilers should be generated from formal models of hardware.

With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.
Generating Compilers  →  Why Now?  →  Case Study: Lakeroad  →  Call to Action

Google TPU

NVIDIA Tensor Cores

AWS Inherentia
Generating Compilers  Why Now?  Case Study: Lakeroad  Call to Action

Google TPU

NVIDIA Tensor Cores

AWS Inferentia

Apple A16

Xilinx Zynq
Generating Compilers  Why Now?  Case Study: Lakeroad  Call to Action

Google TPU

Apple A16

NVIDIA Tensor Cores

AWS Inferentia

Xilinx Zynq

Gerasimova et al. Connectable DNA Logic Gates: OR and XOR Logics.
Generating Compilers

Why Now?

Case Study: Lakeroad

Call to Action


Gerasimova et al. Connectable DNA Logic Gates: OR and XOR Logics.
Hardware is growing more diverse; more compilers are desperately needed!

Gerasimova et al. Connectable DNA Logic Gates: OR and XOR Logics.

Generating Compilers
Why Now?
Case Study: Lakeroad
Call to Action

Hardware is growing more diverse; more compilers are desperately needed!
How could we possibly support all of this hardware?

Gerasimova et al. Connectable DNA Logic Gates: OR and XOR Logics.

Hardware is growing more diverse; more compilers are desperately needed!

How could we possibly support all of this hardware?

As as hardware diversifies, it gets more specialized, and thus easier to target!
Generating Compilers

Why Now?

Case Study: Lakeroad

Call to Action


Gerasimova et al. Connectable DNA Logic Gates: OR and XOR Logics.

NVIDIA Tensor Cores

Google TPU

AWS Inferentia

NVIDIA Tensor Cores

Apple A16

Xilinx Zynq
Generating Compilers

Why Now?

Case Study: Lakeroad

Call to Action

Describing Instruction Set Processors Using nML

A. Fauth\textsuperscript{1} \hspace{1cm} J. Van Praet\textsuperscript{2} \hspace{1cm} M. Freericks\textsuperscript{1}

\textsuperscript{1}Institut für Technische Informatik
Tech. Univ. Berlin, Franklinstr. 28/29
D-10587 Berlin, Germany

\textsuperscript{2}IMEC
Kapeldreef 75
B-3001 Leuven, Belgium

1995

Retargetable Generation of Code Selectors from HDL Processor Models

Rainer Leupers, Peter Marwedel
University of Dortmund, Dept. of Computer Science 12, 44221 Dortmund, Germany
email: leupers@marwedel@is12.informatik.uni-dortmund.de

1997

Automatic Tool Generation from Structural Processor Descriptions

Florian Brandner

2009

Generating compilers for general-purpose hardware is difficult.
Generating Compilers

Why Now?

Case Study: Lakeroad

Call to Action


Google TPU

NVIDIA Tensor Cores

AWS Inferentia

Apple A16

Xilinx Zynq

Gerasimova et al. Connectable DNA Logic Gates: OR and XOR Logics.
Generating Compilers

Why Now?

Case Study: Lakeroad

Call to Action

Specialized hardware is easier to target with automated reasoning tools!

Gerasimova et al. Connectable DNA Logic Gates: OR and XOR Logics.

results of the SAT competition/race winners on the SAT 2009 application benchmarks, 20 min timeout

Järvisalo et al. 2012. The international SAT solver competitions.
Generating Compilers

Why Now?

Case Study: Lakeroad

Call to Action

Järvisalo et al. 2012. The international SAT solver competitions.
Generating Compilers

Why Now?

Case Study: Lakeroad

Call to Action

SAT/SMT

Equality Saturation

Vectorization for Digital Signal Processors via Equality Saturation

Alexa VanHattum  
Cornell University  
Ithaca, NY, USA

Rachit Nigam  
Cornell University  
Ithaca, NY, USA

Vincent T. Lee  
Facebook Reality Labs Research  
Redmond, WA, USA

James Bornholt  
The University of Texas at Austin  
Austin, TX, USA

Adrian Sampson  
Cornell University  
Ithaca, NY, USA

Järvisalo et al. 2012. The international SAT solver competitions.
Generating Compilers

Why Now?

Case Study: Lakeroad

Call to Action

SAT/SMT

Järvisalo et al. 2012. The international SAT solver competitions.

Equality Saturation

Vectorization for Digital Signal Processors via Equality Saturation

Alexa VanHattum
Cornell University
Ithaca, NY, USA

Rachit Nigam
Cornell University
Ithaca, NY, USA

Vincent T. Lee
Facebook Reality Labs Research
Redmond, WA, USA

James Bornholt
The University of Texas at Austin
Austin, TX, USA

Adrian Sampson
Cornell University
Ithaca, NY, USA

Benchmarking Large Language Models for Automated Verilog RTL Code Generation

Shailja Thakur*, Baleegh Ahmad*, Zhenxing Fan*, Hammond Pearce*, Benjamin Tan†, Ramesh Karri*, Brendan Dolan-Gavitt†, Siddharth Garg*

*New York University, †University of Calgary

ML/LLMs
Generating Compilers

Why Now?

Case Study: Lakeroad

Call to Action

SAT/SMT

Järvisalo et al. 2012. The international SAT solver competitions.

Equality Saturation

Vectorization for Digital Signal Processors via Equality Saturation

Alexa VanHattum
Cornell University
Ithaca, NY, USA

Rachit Nigam
Cornell University
Ithaca, NY, USA

Vincent T. Lee
Facebook Reality Labs Research
Redmond, WA, USA

James Bornholt
The University of Texas at Austin

Adrian Sampson
Cornell University

Automated reasoning tools are ready for the task of compiler generation.

Models for Automated Verilog RTL Code Generation

Shailja Thakur*, Baleegh Ahmad*, Zhenxing Fan*, Hammond Pearce*,
Benjamin Tan†, Ramesh Karri†, Brendan Dolan-Gavitt*, Siddharth Garg*

*New York University, †University of Calgary
Compilers should be generated from formal models of hardware.

With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.
Compilers should be generated from formal models of hardware.

With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.

Hardware is diversifying, and we need new compilers.
Compilers should be generated from formal models of hardware.

With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.

Hardware is diversifying, and we need new compilers. Modern targets are more amenable to automated methods.
Compilers should be generated from formal models of hardware.

With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.

Hardware is diversifying, and we need new compilers.
Modern targets are more amenable to automated methods.
Automated reasoning tools are ready for the task of compiler generation.
Generating Compilers ➔ Why Now? ➔ Case Study: Lakeroad ➔ Call to Action
Generating Compilers → Why Now? → Case Study: Lakeroad → Call to Action
Compilers should be generated from formal models of hardware.

With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.
Compilers should be generated from formal models of hardware.

With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.

Let’s look at a concrete example: FPGAs.
Primitives

Lookup Tables
(Programming logic)
Primitives

- Lookup Tables (Programmable logic)
- Carry Chains
Are new primitives a challenge for FPGA compilers?

Compilers should be generated from formal models of hardware. With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.

Generating Compilers  Why Now?  Case Study: Lakeroad  Call to Action

With the growing diversity of hardware

Even *within* FPGAs, hardware is diversifying.
Are new primitives a challenge for FPGA compilers?

Compilers should be generated from formal models of hardware.

With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.

Even *within* FPGAs, hardware is diversifying.

Are new primitives a challenge for FPGA compilers?
Are new primitives a challenge for FPGA compilers?

\(((d + a) * b)^c\)
Are new primitives a challenge for FPGA compilers?

\[(d + a) \times b \land c\]
Are new primitives a challenge for FPGA compilers?

<table>
<thead>
<tr>
<th>Cell</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSP48E1</td>
<td>2</td>
</tr>
<tr>
<td>LUT2</td>
<td>10</td>
</tr>
<tr>
<td>SRL16E</td>
<td>10</td>
</tr>
<tr>
<td>FDRE</td>
<td>10</td>
</tr>
</tbody>
</table>
Are new primitives a challenge for FPGA compilers?

<table>
<thead>
<tr>
<th></th>
<th>Cell</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DSP48E1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>LUT2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>SRL16E</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>FDRE</td>
<td>10</td>
</tr>
</tbody>
</table>

Should use only a single DSP!
Are new primitives a challenge for FPGA compilers?
Are new primitives a challenge for FPGA compilers?

On brief inspection, yes!
On brief inspection, yes!

But this is unsurprising—DSPs are complicated.
MULTISIGNOUT and CARRYCASCOUT .......................................................... 72
Summary ........................................................................................................ 73

Appendix A: Additional Resources and Legal Notices

Xilinx Resources ....................................................................................... 75
Solution Centers ....................................................................................... 75
Documentation Navigator and Design Hubs ............................................. 75
References ................................................................................................. 76
Please Read Important Legal Notices .................................................... 77
Generating Compilers

Why Now?

Case Study: Lakeroad

Call to Action

DSP manual is over 75 pages long
module DSP48E2 #(
    parameter integer ACASCREG = 1,
    parameter integer ADREG = 1,
    parameter integer ALUMODEREG = 1,
    parameter AMULTSEL = "A",
    parameter integer AREG = 1,
    parameter AUTORESET_PATDET = "NO_RESET",
    parameter AUTORESET_PRIORITY = "RESET",
    parameter A_INPUT = "DIRECT",
    ...
)

(output [29:0] ACOUT,
output [17:0] BCOUT,
output CARRYCASCOUT,
...

input [29:0] A,
input [29:0] ACIN,
input [3:0] ALUMODE,
input [17:0] B,
input [17:0] BCIN,
input [47:0] C,
...
);

module DSP48E2 #(
    parameter integer ACASCREG = 1,
    parameter integer ADREG = 1,
    parameter integer ALUMODEREG = 1,
    parameter AMULTSEL = "A",
    parameter integer AREG = 1,
    parameter AUTORESET_PATDET = "NO_RESET",
    parameter AUTORESET_PRIORITY = "RESET",
    parameter A_INPUT = "DIRECT",
    ...
)
output [29:0] ACOUT,
output [17:0] BCOUT,
output CARRYCASCOUT,
...}

input [29:0] A,
input [29:0] ACIN,
input [3:0] ALUMODE,
input [17:0] B,
input [17:0] BCIN,
input [47:0] C,
...
module DSP48E2 #(
  parameter integer ACASCREG = 1,
  parameter integer ADREG = 1,
  parameter integer ALUMODEREG = 1,
  parameter AMULTSEL = "A",
  parameter integer AREG = 1,
  parameter AUTORESET_PATDET = "NO_RESET",
  parameter AUTORESET_PRIORITY = "RESET",
  parameter A_INPUT = "DIRECT",
  ...
);

output [29:0] ACOUT,
output [17:0] BCOUT,
output CARRYCASCOUT,
input [29:0] A,
input [29:0] ACIN,
input [3:0] ALUMODE,
input [17:0] B,
input [17:0] BCIN,
input [47:0] C,
...);

Table 2-4: OPMODE Control Bits Select X Multiplexer Outputs

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>xx</td>
<td>xxx</td>
<td>xx</td>
<td>00</td>
<td>0</td>
<td>Default</td>
</tr>
<tr>
<td>xx</td>
<td>xxx</td>
<td>01</td>
<td>01</td>
<td>M</td>
<td>Must select with OPMODE[3:2] = 01</td>
</tr>
<tr>
<td>xx</td>
<td>xxx</td>
<td>10</td>
<td>P</td>
<td></td>
<td>Requires PREG = 1</td>
</tr>
<tr>
<td>xx</td>
<td>xxx</td>
<td>11</td>
<td>A:B</td>
<td>48-bits wide</td>
<td></td>
</tr>
</tbody>
</table>

When either TWO24 or FOUR12 mode is selected, the multiplier must not be used, and USE_MULTI must be set to NONE.

Notes:
1. When these data pins are not used and to reduce leakage power dissipation, the data pin input signals must be tied High, the input register must be selected, and the CE and RST input control signals must be tied Low. An example of unused C input recommended settings would be setting C[47:0] = all ones, CREG = 1, CEC = 0, and RSTC = 0.
2. These signals are dedicated routing paths internal to the DSP48E2 column. They are not accessible via general routing resources.
3. All signals are active High.
module DSP48E2 #(
    parameter integer ACASCREG = 1,
    parameter integer ADREG = 1,
    parameter integer ALUMODEREG = 1,
    parameter AMULTSEL = "A",
    parameter integer AREG = 1,
    parameter AUTORESET_PATDET = "NO_RESET",
    parameter AUTORESET_PRIORITY = "RESET",
    parameter A_INPUT = "DIRECT",
    ...
)
output [29:0] ACOUT,
output [17:0] BCOUT,
output CARRYCASCOUT,
...

input [29:0] A,
input [29:0] ACIN,
input [3:0] ALUMODE,
input [17:0] B,
input [17:0] BCIN,
input [47:0] C,
...

Configuring the DSP requires setting 100+ ports and parameters.

When either TWO24 or FOUR12 mode is selected, the multiplier must not be used, and USE_MULT must be set to NONE.

### Table 2-4: OPMODE Control Bits Select X Multiplexer Outputs

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>xx</td>
<td>xx</td>
<td>xx</td>
<td>00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>xx</td>
<td>xx</td>
<td>01</td>
<td>01</td>
<td>M</td>
<td>Must select with OPMODE[3:2] = 01</td>
</tr>
<tr>
<td>xx</td>
<td>xx</td>
<td>xx</td>
<td>10</td>
<td>P</td>
<td>Requires PREG = 1</td>
</tr>
<tr>
<td>xx</td>
<td>xx</td>
<td>xx</td>
<td>11</td>
<td>A:B</td>
<td>48-bits wide</td>
</tr>
</tbody>
</table>

### Notes:
1. When these data pins are not used and to reduce leakage power dissipation, the data pin input signals must be tied High, the input register must be selected, and the CE and RST input control signals must be tied Low. An example of unused C input recommended settings would be setting C[47:0] = all ones, CREG = 1, CEC = 0, and RSTC = 0.
2. These signals are dedicated routing paths internal to the DSP48E2 column. They are not accessible via general routing resources.
3. All signals are active High.

Configuring a DSP sounds a lot like writing a program!
**Insight #1:** configuring DSPs and other complex primitives is similar to writing a program...
module DSP48E2 #(
    parameter integer ACASCREG = 1,
    parameter integer ADREG = 1,
    parameter integer ALUMODEREG = 1,
    parameter integer AMULTSEL = "A",
    parameter integer AREG = 1,
    parameter AUTORESET_PATDET = "NO_RESET",
    parameter AUTORESET_PRIORITY = "RESET",
    parameter A_INPUT = "DIRECT",
    ...
)
(output [29:0] ACOUT,
output [17:0] BCOUT,
output CARRYCASCOUT,
input [29:0] A,
input [29:0] ACIN,
input [3:0] ALUMODE,
input [17:0] B,
input [17:0] BCIN,
input [47:0] C,
...);

**Insight #1:** configuring DSPs and other complex primitives is similar to writing a program…

…so use *program synthesis.*
Insight #1: configuring DSPs and other complex primitives is similar to writing a program…

...so use program synthesis.

Solver-aided program synthesis: using SMT/SAT/etc. to generate programs by solving a set of constraints.
module DSP48E2 #(
    parameter integer ACASREG = 1,
    parameter integer ADREG = 1,
    parameter integer ALUMODEREG = 1,
    parameter AMULTSEL = "A",
    parameter integer AREG = 1,
    parameter AUTORESET_PATDET = "NO_RESET",
    parameter AUTORESET_PRIORITY = "RESET",
    parameter A_INPUT = "DIRECT",
    ...
);

output [29:0] ACOUT,
output [17:0] BCOUT,
output CARRYCASCOUT,
...

input [29:0] A,
input [29:0] ACIN,
input [3:0] ALUMODE,
input [17:0] B,
input [17:0] BCIN,
input [47:0] C,
...
);

Table 2.4: OPMODE Control Bits Select X Multiplexer Outputs

<table>
<thead>
<tr>
<th>W</th>
<th>Z</th>
<th>Y</th>
<th>X</th>
<th>X Multiplexer Output</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx</td>
<td>xxx</td>
<td>xx</td>
<td>00</td>
<td>0</td>
<td>Default</td>
</tr>
<tr>
<td>xx</td>
<td>xxx</td>
<td>01</td>
<td>01</td>
<td>M</td>
<td>Must select with OPMODE[3:2] = 01</td>
</tr>
<tr>
<td>xx</td>
<td>xxx</td>
<td>xx</td>
<td>10</td>
<td>P</td>
<td>Requires PREG = 1</td>
</tr>
<tr>
<td>xx</td>
<td>xxx</td>
<td>xx</td>
<td>11</td>
<td>A:8</td>
<td>48-bits wide</td>
</tr>
</tbody>
</table>

When either TWO24 or FOUR12 mode is selected, the multiplier must not be used, and USE_MULT must be set to NONE.

Notes:
1. When these data pins are not used and to reduce leakage power dissipation, the data pin input signals must be tied High, the input register must be selected, and the CE and RST input control signals must be tied Low. An example of unused C input recommended settings would be setting C[47:0] = all ones, CREG = 1, CEC = 0, and RSTC = 0.
2. These signals are dedicated routing paths internal to the DSP48E2 column. They are not accessible via general routing resources.
3. All signals are active High.
timescale 1 ps / 1 ps

module DSP48E2 # ();// Copyright (c) 1995/2017 Xilinx, Inc.
// All Right Reserved.
// ...

// Generates Compilers
// Why Now?
// Case Study: Lakeroad
// Call to Action
Simulating DSP48E2.v

Simulation models provide the formal semantics of behaviors and constraints necessary for automated reasoning!
Insight #1: configuring DSPs and other complex primitives is similar to writing a program, so use *program synthesis*.

**Insight #2:** we can extract the semantics necessary for automated reasoning directly from simulation models.
Lakeroad: a hardware synthesis tool utilizing program synthesis and semantics extracted from simulation models to target complex, programmable FPGA primitives.
<table>
<thead>
<tr>
<th>Workload</th>
<th>Signed?</th>
<th># Stages</th>
<th>Yosys</th>
<th>SOTA</th>
<th>Lakeroad</th>
</tr>
</thead>
<tbody>
<tr>
<td>((d + a) \ast b) \mid c</td>
<td>X</td>
<td>1</td>
<td>1 DSP, 20 LUT</td>
<td>1 DSP, 10 LUT</td>
<td>1 DSP</td>
</tr>
<tr>
<td>((d - a) \ast b) \mid c</td>
<td>✓</td>
<td>2</td>
<td>1 DSP, 20 LUT</td>
<td>1 DSP, 10 LUT</td>
<td>1 DSP</td>
</tr>
<tr>
<td>((d - a) \ast b) \wedge c</td>
<td>✓</td>
<td>3</td>
<td>1 DSP, 22 LUT</td>
<td>2 DSP, 11 LUT</td>
<td>1 DSP</td>
</tr>
<tr>
<td>((d + a) \ast b) &amp; c</td>
<td>✓</td>
<td>3</td>
<td>1 DSP, 22 LUT</td>
<td>2 DSP, 11 LUT</td>
<td>1 DSP</td>
</tr>
<tr>
<td>((d + a) \ast b) \wedge c</td>
<td>X</td>
<td>2</td>
<td>1 DSP, 18 LUT</td>
<td>1 DSP, 9 LUT</td>
<td>1 DSP</td>
</tr>
<tr>
<td>Workload</td>
<td>Signed?</td>
<td># Stages</td>
<td>Yosys</td>
<td>SOTA</td>
<td>Lakeroad</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------</td>
<td>----------</td>
<td>---------------------</td>
<td>--------------------</td>
<td>----------</td>
</tr>
<tr>
<td>((d + a) * b)</td>
<td>c</td>
<td>X</td>
<td>1</td>
<td>1 DSP, 20 LUT</td>
<td>1 DSP</td>
</tr>
<tr>
<td>((d - a) * b)</td>
<td>c</td>
<td>✓</td>
<td>2</td>
<td>1 DSP, 20 LUT</td>
<td>1 DSP</td>
</tr>
<tr>
<td>((d - a) * b) ^ c</td>
<td>✓</td>
<td>3</td>
<td>1 DSP, 22 LUT</td>
<td>2 DSP, 11 LUT</td>
<td>1 DSP</td>
</tr>
<tr>
<td>((d + a) * b) &amp; c</td>
<td>✓</td>
<td>3</td>
<td>1 DSP, 22 LUT</td>
<td>2 DSP, 11 LUT</td>
<td>1 DSP</td>
</tr>
<tr>
<td>((d + a) * b) ^ c</td>
<td>X</td>
<td>2</td>
<td>1 DSP, 18 LUT</td>
<td>1 DSP, 9 LUT</td>
<td>1 DSP</td>
</tr>
<tr>
<td>Workload</td>
<td>Signed?</td>
<td># Stages</td>
<td>Yosys</td>
<td>SOTA</td>
<td>Lakeroad</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------</td>
<td>----------</td>
<td>---------------</td>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>((d + a) * b)</td>
<td>c</td>
<td>X</td>
<td>1 DSP, 20 LUT</td>
<td>1 DSP, 10 LUT</td>
<td>1 DSP</td>
</tr>
<tr>
<td>((d - a) * b)</td>
<td>c</td>
<td>✓</td>
<td>1 DSP, 20 LUT</td>
<td>1 DSP, 10 LUT</td>
<td>1 DSP</td>
</tr>
<tr>
<td>((d - a) * b) ^ c</td>
<td>✓</td>
<td>3</td>
<td>1 DSP, 22 LUT</td>
<td>2 DSP, 11 LUT</td>
<td>1 DSP</td>
</tr>
<tr>
<td>((d + a) * b) &amp; c</td>
<td>✓</td>
<td>3</td>
<td>1 DSP, 22 LUT</td>
<td>2 DSP, 11 LUT</td>
<td>1 DSP</td>
</tr>
<tr>
<td>((d + a) * b) ^ c</td>
<td>X</td>
<td>2</td>
<td>1 DSP, 18 LUT</td>
<td>1 DSP, 9 LUT</td>
<td>1 DSP</td>
</tr>
</tbody>
</table>
Compilers should be generated from formal models of hardware.
Compilers should be generated from formal models of hardware.

Lakeroad automatically enables compilation to FPGA primitives, given the simulation models of those primitives.
Compilers should be generated from formal models of hardware.

Lakeroad automatically enables compilation to FPGA primitives, given the simulation models of those primitives.

With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.
Compilers should be generated from formal models of hardware.

Lakeroad automatically enables compilation to FPGA primitives, given the simulation models of those primitives.

With the growing diversity of hardware and the rapid improvement of automated reasoning, now is the time to make this a reality.

Lakeroad demonstrates that automated methods are now powerful enough address gaps in existing state-of-the-art tools.
Generating Compilers ➔ Why Now? ➔ Case Study: Lakeroad ➔ Call to Action
Generating Compilers  ➔ Why Now?  ➔ Case Study: Lakeroad  ➔ Call to Action
The Hardware Lottery

Sara Hooker

Google Research, Brain Team

shooker@google.com

2020
It’s on on all of us to fight against the hardware lottery, by making sure that practitioners in all fields have the hardware and compilers they need to advance their research.
Thank you!