Chisel 2.0.0 to Chisel 3.0.0

Because everybody loves a trilogy

Presentation by Adam Izraelevitz
A long time ago, in a laboratory (not) far, far away....
Generators: Type-Safe Meta-Programming for RTL

- Design Reuse
- Type-Safety
- Powerful Language Features
Hired Jonathan Bachrach

Chisel: Constructing Hardware in a Scala Embedded Language

Jonathan Bachrach, Huu Vinh, Brian Richards, Yunup Liu,
Andrew Waterman, Rima SenGupta, John Wawrzynek, Krista Asanovic
EECS Department, UC Berkeley

(permissions@cs.berkeley.edu) (2015/02/05)

ABSTRACT

In this paper we introduce Chisel, a new hardware description language that supports advanced hardware design using highly parameterized generators and domain-specific hardware languages. By embedding Chisel in the Scala programming language, we save the labor of hand-coding abstractions by providing a domain-specific notation for functional, parametrized hardware design. Chisel can generate high-speed C-like code that can be synthesized onto other FPGA types or as a standard ASIC through the Chisel-IR. This paper presents our design, its embedding in Scala, hardware examples, and results for a 2-stage, 16-bit pipeline.

Categories and Subject Descriptors

B.4.2 [Hardware]: Design Aids: automatic synthesis, hardware description languages

General Terms

Design, Languages, Performance

Keywords

CAD

1. INTRODUCTION

The current dominant approach to hardware description languages (HDLs), Verilog and VHDL, were originally developed for hardware simulation, but have only been adopted as a basis for hardware synthesis. Because the remainder of this paper is based on recent advancements, we refer to these languages.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions Dept, ACM, Inc., Fax +1 (212) 869-0481, or permissions@acm.org.

Hired Jonathan Bachrach

Chisel: Constructing Hardware in a Scala Embedded Language

Jonathan Bachrach, Huu Vinh, Brian Richards, Yunup Liu,
Andrew Waterman, Rima SenGupta, John Wawrzynek, Krista Asanovic
EECS Department, UC Berkeley

(permissions@cs.berkeley.edu) (2015/02/05)

ABSTRACT

In this paper we introduce Chisel, a new hardware description language that supports advanced hardware design using highly parameterized generators and domain-specific hardware languages. By embedding Chisel in the Scala programming language, we save the labor of hand-coding abstractions by providing a domain-specific notation for functional, parametrized hardware design. Chisel can generate high-speed C-like code that can be synthesized onto other FPGA types or as a standard ASIC through the Chisel-IR. This paper presents our design, its embedding in Scala, hardware examples, and results for a 2-stage, 16-bit pipeline.

Categories and Subject Descriptors

B.4.2 [Hardware]: Design Aids: automatic synthesis, hardware description languages

General Terms

Design, Languages, Performance

Keywords

CAD

1. INTRODUCTION

The current dominant approach to hardware description languages (HDLs), Verilog and VHDL, were originally developed for hardware simulation, but have only been adopted as a basis for hardware synthesis. Because the remainder of this paper is based on recent advancements, we refer to these languages.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions Dept, ACM, Inc., Fax +1 (212) 869-0481, or permissions@acm.org.

Hired Jonathan Bachrach

Chisel: Constructing Hardware in a Scala Embedded Language

Jonathan Bachrach, Huu Vinh, Brian Richards, Yunup Liu,
Andrew Waterman, Rima SenGupta, John Wawrzynek, Krista Asanovic
EECS Department, UC Berkeley

(permissions@cs.berkeley.edu) (2015/02/05)

ABSTRACT

In this paper we introduce Chisel, a new hardware description language that supports advanced hardware design using highly parameterized generators and domain-specific hardware languages. By embedding Chisel in the Scala programming language, we save the labor of hand-coding abstractions by providing a domain-specific notation for functional, parametrized hardware design. Chisel can generate high-speed C-like code that can be synthesized onto other FPGA types or as a standard ASIC through the Chisel-IR. This paper presents our design, its embedding in Scala, hardware examples, and results for a 2-stage, 16-bit pipeline.

Categories and Subject Descriptors

B.4.2 [Hardware]: Design Aids: automatic synthesis, hardware description languages

General Terms

Design, Languages, Performance

Keywords

CAD
At the end of ParLab, we solved hardware design
And so, the problem of hardware design was forever solved.
Just kidding.
Solving Hardware Design ≠ Solving The Hardware Design Loop

* from “How to Draw Chip and Dale booklet”. (Walt Disney, 1955)
What had to change?

- Hardware Design Ecosystem
- Stable and User-Friendly API
- External Collaborations
Platform-Specific or Application-Specific RTL Changes

- Scan interface
- Snapshotting
- Interactive debug

Zynq FPGA

+ Clock-generators
+ SRAMs with init
+ Specialized layout

ST 28nm FDSOI

+ SRAM macros
+ Modified module hierarchy
+ Specialized layout

IBM 45nm SOI
What were we doing?

- Manually change RTL?
  - Obfuscates/specializes RTL
- Use CAD tool scripts?
  - Many unsupported use cases
- Python script to edit RTL
  - Not reusable/robust/composable

* from [https://useravatar.services.amuniversal.com/user_avatars/avatars_gocomicsver3/3001000/3001618/IMG_7625.GIF](https://useravatar.services.amuniversal.com/user_avatars/avatars_gocomicsver3/3001000/3001618/IMG_7625.GIF), accessed 10/23/17
Realization: We need a software stack, but for hardware

- **Projects**: Linux, Bitcoin
- **Libraries**: C++ Standard Library, Protocol Buffers
- **Language**: C++, Chisel Frontend
- **Compiler**: Clang, LLVM, Chisel Utilities
- **Platforms**: x86, ARM, RISC-V
- **Frameworks**: BOOM, Rocket, Hwacha, rocketchip, chisel-utils, FIRRTL
Second-System-Syndrome: But... do we really need all that?

<table>
<thead>
<tr>
<th>System</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sodor</td>
<td>6451</td>
</tr>
<tr>
<td>Hwacha</td>
<td>6953</td>
</tr>
<tr>
<td>BOOM</td>
<td>8298</td>
</tr>
<tr>
<td>RocketChip</td>
<td>9578</td>
</tr>
<tr>
<td>Chisel</td>
<td>19248</td>
</tr>
</tbody>
</table>
For impact, we need an ecosystem
Developing, Porting, Code Reviews, Testing, and so forth

- Spring 2015: Designed FIRRTL Compiler
- Summer 2015: Designed Chisel 3 Frontend
- Fall 2015: Ported RocketChip
- Winter 2015: Ported Chisel Testers
- Spring 2016: Added Fixed-Point Type
- Summer 2016: Added Analog Type
- Fall 2016: Added withClock
- Winter 2016: Released Chisel 3.0.0
- Spring 2017: Released FIRRTL 1.0.0
Chisel 3.0.0 and FIRRTL 1.0.0 have been released!

- **Projects**
  - FireSim - Datacenter Emulation on FPGAs
  - Strober - Fast+Accurate Power Sims. for Long Programs
  - Hurricane 2 - Multi-Core DVFS (Sub-Core)

- **Transformations**
  - Quick (and Semi-Accurate) Timing and Area Estimation
  - Automatic Combinational Cycle Removal
  - Snapshotting and Hardware Assertions for FPGAs

- **New Features**
  - Hardware Types vs Hardware Components
  - New types (e.g. Complex, DspReal, Fixed-Point, Analog)
  - Chisel Library support (annotations)
  - Invalidate API for safer connections

---

**Chisel 3**
https://github.com/freechipsproject/chisel3/releases/tag/v3.0.0

**FIRRTL**
https://github.com/freechipsproject/firrtl/releases/tag/v1.0.0
Intel: Fast and Semi-Accurate FIRRTL Timer
What had to change?

A Chisel Compiler

Stable and User-Friendly API

External Collaborations
Emphasis on Clarity

**Chisel 2.0.0**

1. Register of type UInt, width of 3
2. Register whose next cycle's value is 3
3. Register whose initial value is 3
4. Register no initial value and width of 2

**Chisel 3.0.0**

1. Register of type UInt, width of 3
2. Register whose next cycle's value is 3
3. Register whose initial value is 3
4. Register no initial value and width of 2
Less Error-Prone API's

Chisel 2.0.0

```scala
def func[T <: Chisel.Data](other: T) = {
  io.out := Reg(null.asInstanceOf[T], next = other, null.asInstanceOf[T])
}
```

[info] [0.000] Elaborating design...
[info] [0.022] Done elaborating.

Chisel 3.0.0

```scala
def func[T <: Chisel.Data](other: T) = {
  io.out := RegNext(other)
}
```

[info] [0.000] Elaborating design...
[info] [0.022] Done elaborating.
Lightweight Support for Multi-Clock/Reset

withReset(io.alternateReset) {
    val altRst = RegInit(0.U(10.W))
    ...
}

withClock(io.alternateClock) {
    val altClk = RegInit(0.U(10.W))
    ...
}

Everything in this scope with have io.alternateReset as the reset

Everything in this scope with have io.alternateClock as the clock
Exciting Future Work: The Unified Chisel Tester

- **Fragmented Landscape**
  - BasicTester (Hardware Testing Hardware)
  - PeekPokeTester (Interactive and Slow)
  - AdvancedTester (Limited Concurrency)

- **Unified Chisel Tester**
  - Lightweight, powerful, fast
  - Multiple circuit drivers, multithreaded
  - Integration with Verilator, VCS, Interpreter

- **If you have thoughts - send them our way!**
What had to change?

- A Chisel Compiler
- Stable and User-Friendly API
- External Collaborations
Welcome to the Chisel 3 wiki!

If you are completely new to Chisel, check out A Short Users Guide to Chisel.

Chisel is constantly being improved. See the latest Release Notes.

For migrating from Chisel 2 to Chisel 3, check out Chisel3 vs Chisel2.

The ScalaDoc for Chisel3 is available at the API tab on the Chisel web site.

For useful design patterns, see the Cookbook.

For cool new features on the leading and bleeding edge, see Experimental Features.

If you’re developing a Chisel library, see Developers.

Other interesting pages:

- Frequently Asked Questions
Module 2.1: A Simple Chisel Module

A Tiny Module

Like Verilog, we can declare module definitions in Chisel. The following example is a Chisel module, Tiny that has one input, in, and one output, out, and inside it conditionally connects in and out, so in drives out.

```chisel
// Chisel code: Declare a new module definition
class Tiny extends Module {
  val io = Bundle {
    val in = Input(UInt(4.W));
    val out = Output(UInt(4.W));
  }

  io.out <= io.in
}
println(getPrinter(new Tiny()));
```

There's a lot here! The following explains how to think of each line in terms of the hardware we are describing.

```chisel
class Tiny extends Module {

  We declare a new module called Tiny.

  val io = Bundle;

  We declare all our input and output ports into a special io variable.

  new Bundle;
```
Open-Development via Github Issues/Pull Requests
Stack Overflow!
Academic Impact: 188 citations (in 5 years)
Active Users (That We Know Of)
OMG! I have so many data. This is making my algorithm slow.

I can apply my algorithm directly on the FPGA... but I understand only low-level language.

It is so difficult to implement my algorithm in Verilog. This means redesigning everything! It is going to take so much time!

...and I don’t even know hardware programming.

Hey! Have you tried Chisel?

Well, google it.

Chisel? What is that?

Whaow! It generates Verilog and I can even use all advantage that Scala has in my code.

This is awesome!!! Thank you Berkeley.
Ideas for Governance? Maintenance? Workshops?
Thanks to all Chiselers out there! (And many more!!)
So long (ASPIRE), and thanks for all the fish chips!