Chisel and FIRRTL for next-generation SoC designs

Jack Koenig
SiFive
CWG TAC Member
Introduction

What is the Chisel Working Group (CWG)?
What is Chisel?

- **Constructing Hardware In a Scala Embedded Language**
- Domain Specific Language where the domain is digital design
- NOT high-level synthesis (HLS) nor behavioral synthesis
- Write Scala program to construct and connect hardware objects
  - Parameterized types
  - Object-Oriented Programming
  - Functional Programming
  - Static Typing w/ Powerful Type Inference
- Intended for writing *reusable* hardware generators (libraries)
No Loss of Expressibility: “Verilog-like” Chisel

FIR Filter - 3-point moving sum

What about >3 points?
What about weighted averages?

We want a generic FIR filter!

class MovingSum3(bitWidth: Int) extends Module {
  val io = IO(new Bundle {
    val in = Input(UInt(bitWidth.W))
    val out = Output(UInt(bitWidth.W))
  })

  val z1 = RegNext(io.in)
  val z2 = RegNext(z1)

}
Massive Increase in Parameterizability: “Software-like” Chisel

FIR Filter - Parameterized by bitwidth and coefficients with no loss of expressibility or performance.

Meta-programming enables powerful parameterization.

```scala
class FirFilter(bitWidth: Int, coeffs: Seq[UInt]) extends Module {
  val io = IO(new Bundle {
    val in = Input(UInt(bitWidth.W))
    val out = Output(UInt())
  })
  // Create the serial-in, parallel-out shift register
  val zs = Reg(Vec(coeffs.length, UInt(bitWidth.W)))
  zs(0) := io.in
  for (i <- 1 until coeffs.length) {
    zs(i) := zs(i-1)
  }
  // Do the multiplies
  val products = VecInit.tabulate(coeffs.length)(i => zs(i) * coeffs(i))
  // Sum up the products
  io.out := products.reduce(_ +& _)
}
```
Massive Increase in Parameterizability: “Software-like” Chisel

Meta-programming enables powerful parameterization.

```scala
class FirFilter(bitWidth: Int, coeffs: Seq[UInt]) extends Module {
  val io = IO(new Bundle {
    b0 = In(UInt(bitWidth)),
    b1 = In(UInt(bitWidth)),
    z_n = Out(UInt(bitWidth)),
  })

  override def setup = {
    // same 3-point moving sum filter as before
    val movingSum3Filter = Module(new FirFilter(8, Seq(1.U, 1.U, 1.U)))

    // 1-cycle delay as a FIR filter
    val delayFilter = Module(new FirFilter(8, Seq(0.U, 1.U)))

    // 5-point FIR filter with a triangle impulse response

    // Sum up the products
    io.out := products.reduce(_ +& _)
  }
}
```
Platform-Specific or Application-Specific RTL Changes

- **IBM 45nm SOI**
  - SRAM macros
  - Modified module hierarchy
  - Specialized layout

- **Zynq FPGA**
  - Scan interface
  - Snapshotting
  - Interactive debug

- **ST 28nm FDSOI**
  - Clock-generators
  - SRAMs with init
  - Specialized layout
Realization: We need a software stack, but for hardware

<table>
<thead>
<tr>
<th>projects</th>
<th>BOOM</th>
<th>Rocket</th>
<th>Hwacha</th>
</tr>
</thead>
<tbody>
<tr>
<td>libraries</td>
<td>THE C++ STANDARD LIBRARY</td>
<td>protobuf</td>
<td>rocketchip</td>
</tr>
<tr>
<td>language</td>
<td>C++</td>
<td>clang</td>
<td>Chisel Frontend</td>
</tr>
<tr>
<td>compiler</td>
<td>transforms</td>
<td>ELVM</td>
<td>FIRRTL</td>
</tr>
<tr>
<td>platforms</td>
<td>x86</td>
<td>ARM</td>
<td>RISC-V</td>
</tr>
</tbody>
</table>

- BOOM
- Rocket
- Hwacha
- THE C++ STANDARD LIBRARY
- protobuf
- rocketchip
- chisel-utils
- C++
- clang
- Chisel Frontend
- transforms
- ELVM
- FIRRTL
- x86
- ARM
- RISC-V
- x86
- ARM
- RISC-V
- OpenROAD
- VERILATOR
- OpenROAD
- FPGAs
FIRRTL: An Extensible Hardware Compiler Framework

Modular Compiler Passes (Transforms)

Robust Metadata/Annotations Support
Projects of the Chisel Working Group

• Chisel 3
• FIRRTL
• ChiselTest (formerly Chisel Testers 2)
• Treadle
• Chisel IOTesters
• DSP Tools
• Diagrammer
• Chisel Bootcamp
• Chisel Template

Currently a CHIPS Alliance “Sandbox” project with intent to “Graduate”
Highlights

(From the last six-ish months)
Chisel v3.5.0-RC1 Released!

- Culmination of almost a year of work
- Lightning Highlights
  - Vec literal support
  - Scala 2.13 support (2.11 EOL)
  - Decoder + minimizer API in chisel3'util (w/ Espresso integration)
  - Source locator compacting
- Far too many things to cover, see:
  - https://github.com/chipsalliance/chisel3/releases/tag/v3.5.0-RC1
  - https://github.com/chipsalliance/firrtl/releases/tag/v1.5.0-RC1
  - Other project notes to come by v3.5.0

Note: v3.5 Docs will not be reflected on chisel-lang.org until v3.5.0 is released
Chisel v3.5.0-RC1 Released!

- Culmination of almost a year of work
- Lightning Highlights
  - Vec literal support
  - Scala 2.13 support (2.11 EOL)
  - Decoder + minimizer API in chisel3.util (w/ Espresso integration)
  - Source locator compacting
- Far too many things to cover, see:
  - https://github.com/chipsalliance/chisel3/releases/tag/v3.5.0-RC1
  - https://github.com/chipsalliance/firrtl/releases/tag/v1.5.0-RC1
  - Other project notes to come by v3.5.0

Note: v3.5 Docs will **not** be reflected on chisel-lang.org until v3.5.0 is released
Chisel v3.5.0-RC1 Released!

- Culmination of almost a year of work
- Lightning Highlights
  - Vec literal support
  - Scala 2.13 support (2.11 EOL)
  - Decoder + minimizer API in chisel3.util (w/ Espresso integration)
  - Source locator compacting
- Far too many things to cover, see:
  - https://github.com/chipsalliance/chisel3/releases/tag/v3.5.0-RC1
  - https://github.com/chipsalliance/firrtl/releases/tag/v1.5.0-RC1
  - Other project notes to come by v3.5.0

```scala
val table = TruthTable(
  Map(
    // BitPat("b000") -> BitPat("b0"),
    BitPat("b001") -> BitPat("b?"),
    BitPat("b010") -> BitPat("b?"),
    // BitPat("b011") -> BitPat("b0"),
    BitPat("b100") -> BitPat("b1"),
    BitPat("b101") -> BitPat("b1"),
    // BitPat("b110") -> BitPat("b0"),
    BitPat("b111") -> BitPat("b1")
  ),
  BitPat("b0") // default
)
output := decoder(input, table)
```

Note: v3.5 Docs will **not** be reflected on chisel-lang.org until v3.5.0 is released.
Chisel v3.5.0-RC1 Released!

• Culmination of almost a year of work
• Lightning Highlights
  • Vec literal support
  • Scala 2.13 support (2.11 EOL)
  • Decoder + minimizer API in chisel3.util (w/ Espresso integration)
  • Source locator compacting
• Far too many things to cover, see:
  • https://github.com/chipsalliance/chisel3/releases/tag/v3.5.0-RC1
  • https://github.com/chipsalliance/firrtl/releases/tag/v1.5.0-RC1
  • Other project notes to come by v3.5.0

Note: v3.5 Docs will not be reflected on chisel-lang.org until v3.5.0 is released
ChiselTest Improvements

• Improved Verilator simulation performance via JNA
• Verilator backend now supports dumping FST instead of VCD
• PeekPokeTester compatibility API
  • Helps migrate users off old chisel-iotesters
• Simulation constructs can now be annotated
• assert/assume/cover graduated out of experimental
• Simulation binary caching
• Support for bounded model checking (next slide)
Native Formal Verification Support

• Formal verification is assumed to be difficult for users

• Good tooling and sensible defaults can help
  • Similar to simulator-based flow
  • Safe `past` function
  • Automatic reset guarding (default but disableable)

• Close integration with simulation testing flow
  • Same basic APIs
  • Same IDE and tooling integration

• Automatically runs counter examples through a simulator to provide a waveform

• Native FIRRTL -> SMTLib or btor2 output

• Works with Z3 and CVC4

See Kevin Laeufer's WOSET Paper [https://woset-workshop.github.io/WOSET2021.html#article-3](https://woset-workshop.github.io/WOSET2021.html#article-3)
Native Formal Verification Support

• Formal verification is assumed to be difficult for users
• Good tooling and sensible defaults can help
  • Similar to simulator-based flow
  • Safe `past` function
  • Automatic reset guarding (default but disableable)
• Close integration with simulation testing flow
  • Same basic APIs
  • Same IDE and tooling integration
• Automatically runs counter examples through a simulator to provide a waveform
• Native FIRRTL -> SMTLib or btor2 output
• Works with Z3 and CVC4


```scala
class Quiz15 extends Module {
  /* [...] I/O definitions */
  val mem = SyncReadMem(256, UInt(32.W), WriteFirst)
  when(iWrite) { mem.write(iWAddr, iData) }
  oData := mem.read(iRAddr, iRead)
  when(past(iWrite && iRead &&
       iWAddr === iRAddr)) {
    verification.assert(oData === past(iData))
  }
}

class ZipCpuQuizzes extends AnyFlatSpec
  with Chisel Scalatest Tester with Formal {
    "Quiz15" should "pass with WriteFirst" in {
      verify(new Quiz15, Seq(BoundedCheck(5)))
    }
  }
```
Definition / Instance

• Historically, Chisel elaborates every module instance and then deduplicates structurally equivalent modules

• New experimental API for defining (and elaborating) a module once and instantiating multiple times
  • Definition – Elaborates implementation of module
  • Instance – Merely instantiates public API

• Major performance optimization for very large or hierarchical designs

• Composes with cross-module reference annotations

See https://github.com/chipsalliance/chisel3/pull/2045 for docs
Definition / Instance

- Historically, Chisel elaborates every module instance and then deduplicates structurally equivalent modules
- New experimental API for defining (and elaborating) a module once and instantiating multiple times
  - Definition – Elaborates implementation of module
  - Instance – Merely instantiates public API
- Major performance optimization for very large or hierarchical designs
- Composes with cross-module reference annotations

See https://github.com/chipsalliance/chisel3/pull/2045 for docs

```scala
@instantiable
class AddOne(width: Int) extends Module {
  @public val in  = IO(Input(UInt(width.W)))
  @public val out = IO(Output(UInt(width.W)))
  out := in + 1.U
}

class AddTwo(width: Int) extends Module {
  val in  = IO(Input(UInt(width.W)))
  val out = IO(Output(UInt(width.W)))

  val addOneDef = Definition(new AddOne(width))
  val i0 = Instance(addOneDef)
  val i1 = Instance(addOneDef)

  i0.in := in
  i1.in := i0.out
  out := i1.out
}
```
Definition / Instance

• Historically, Chisel elaborates every module instance and then deduplicates structurally equivalent modules

• New experimental API for defining (and elaborating) a module once and instantiating multiple times
  • Definition – Elaborates implementation of module
  • Instance – Merely instantiates public API

• Major performance optimization for very large or hierarchical designs

• Composes with cross-module reference annotations

See https://github.com/chipsalliance/chisel3/pull/2045 for docs
DataView

- Often users want to manipulate hardware values as if they were a different type
  - AXI-style flat bus interface used as more structured hierarchy
  - Manipulate 1D Array of Reg as if it were 2D
- Allows treating objects of one type as another
- A superpowered union or cast, like View in SQL
- Used to implement:
  - Seamless integration with Scala types
  - Bundle upcasting
  - User-defined mappings between types

See https://github.com/chipsalliance/chisel3/pull/1955 for docs
DataView

- Often users want to manipulate hardware values as if they were a different type
  - AXI-style flat bus interface used as more structured hierarchy
  - Manipulate 1D Array of Reg as if it were 2D
- Allows treating objects of one type as another
- A superpowered union or cast, like View in SQL
- Used to implement:
  - Seamless integration with Scala types
  - Bundle upcasting
  - User-defined mappings between types

```scala
val a, b, c, d = IO(Input(UInt(8.W)))
val w, x, y, z = IO(Output(UInt(8.W)))
((w, x), (y, z)) := ((a, b), (c, d))
```

See https://github.com/chipsalliance/chisel3/pull/1955 for docs
Often users want to manipulate hardware values as if they were a different type

- AXI-style flat bus interface used as more structured hierarchy
- Manipulate 1D Array of Reg as if it were 2D

Allows treating objects of one type as another

- A superpowered union or cast, like View in SQL

Used to implement:
- Seamless integration with Scala types
- Bundle upcasting
- User-defined mappings between types

DataView

```scala
val a, b, c, d = IO(Input(UInt(8.W)))

class Foo extends Bundle {
  val a = UInt(8.W)
}
class Bar extends Foo {
  val b = UInt(8.W)
}

// ...

val foo = IO(Input(new Foo))
val bar = IO(Output(new Bar))
bar.getViewAsSupertype(new Foo) := foo // bar.a := foo.a
bar.b := 123.U // Still need to drive .b
```

See https://github.com/chipsalliance/chisel3/pull/1955 for docs
Often users want to manipulate hardware values as if they were a different type:

- AXI-style flat bus interface used as more structured hierarchy
- Manipulate 1D Array of Reg as if it were 2D
- Allows treating objects of one type as another
- A superpowered union or cast, like `View` in SQL

Used to implement:

- Seamless integration with Scala types
- Bundle upcasting
- User-defined mappings between types

See https://github.com/chipsalliance/chisel3/pull/1955 for docs

```
class MyBundle(val w: Int) extends Bundle {
  val foo = UInt(w.W)
  val bar = UInt(w.W)
}

implicit val v1 = DataView[MyBundle, Vec[UInt]]({
  bun => Vec(2, UInt(bun.w.W)), // Create a View from a Target
  _foo -> _(0), _bar -> _(1)  // Map each field
  // ...
})

val out = IO(Output(new MyBundle(8)))
val bar = out.viewAs[Vec[UInt]]
for ((field, idx) <- bar.zipWithIndex) {
  field := idx.U
```
AutoCloneType2

- cloneType is an implementation detail that leaks into the user API (since original Chisel)
- Useless boilerplate
- Original AutoCloneType works okay but has some limitations
  - Parameters must be defined as "vals"
  - Works in typical use cases but not all use cases
  - Slow
- The Chisel compiler plugin now generates cloneType for all Bundles
- Available in Chisel v3.4.3 (opt-in)
  - Improved in v3.4.4
  - Mandatory in v3.5.0

Before:

```scala
class MyBundle(w: Int) extends Bundle {
  val foo = UInt(w.W)
  override def cloneType = new MyBundle(w).asInstanceOf[this.type]
}
```

After:

```scala
class MyBundle(w: Int) extends Bundle {
  val foo = UInt(w.W)
}
```
Community
Continued Growth

Chisel Community Conference
Shanghai, June 2021

See talks on www.youtube.com/chisel-lang
Get Involved

Chisel Users Community

If you’re a Chisel user and want to stay connected to the wider user community, any of the following are great avenues:

- Interact with other Chisel users in one of our Gitter chat rooms:
  - Chisel
  - FIRRTL
- Ask/Answer Questions on Stack Overflow using the [chisel] tag
- Ask questions and discuss ideas on the Chisel/FIRRTL Mailing Lists:
  - Chisel Users
  - Chisel Developers
- Follow us on our @chisel-lang Twitter Account
- Subscribe to our chisel-lang YouTube Channel

Chat with us on Gitter

Questions tagged [chisel]

Chisel is an open-source hardware construction language developed at UC Berkeley that supports advanced hardware design using highly parameterized generators and layered domain-specific hardware languages.

Learn more... Top users Synonyme

485 questions

1 vote
0 answers chisel rocket-chip
23 views

Ask questions on StackOverflow

Watch talks on YouTube

www.chisel-lang.org
Extra / Old Slides
Further Improved Website

Now with a search bar!

Chisel/FIRRTL Hardware Compiler Framework

Latest API Docs

Project-specific documentation

Community page
Further Improved Website

Documentation examples are compiled and run!!!
Enhanced Signal Naming (from last time)

• Historically Chisel has struggled with signal naming
• Chisel 3.4 has much better naming

```python
def func() = {
  val x = a + b
  val y = x - 3.U
  y & 0xcf.U
}
val result = func() | 0x8.U
out := result
```

Old* Verilog

```verilog
wire [7:0] _T_1 = a + b;
wire [7:0] _T_3 = _T_1 - 8'h3;
wire [7:0] _T_4 = _T_3 & 8'hcf;
assign out = _T_4 | 8'h8;
```

Chisel 3.4 Verilog

```verilog
wire [7:0] result_x = a + b;
wire [7:0] result_y = result_x - 8'h3;
wire [7:0] _result_T = result_y & 8'hcf;
assign out = _result_T | 8'h8;
```
Refined Signal Naming

• Optional “tap” output
• What should the name of the port be?
  • port
  • tap_port
  • tap
  • tapPort
• In 3.4.0, the name was “tap_port”
• In 3.4.1 on, the name is “tap”
• Additional improvements to naming (especially when using recursion)
• Now with ~5 months of use, it’s going great!

```scala
class Example(tapWidth: Option[Int]) extends MultiIOModule {
  ...
  val tap = tapWidth.map { width =>
    val port = IO(Output(UInt(width.W)))
    port := ...  
    port
  }

  if (tap.isDefined) {
    val tapPort = tap.get
    ...
  }
}
```
Improved Release Methodology

Automated Backporting + CI