Chisel and FIRRTL for next-generation SoC designs

Jack Koenig
SiFive
CWG TAC Member

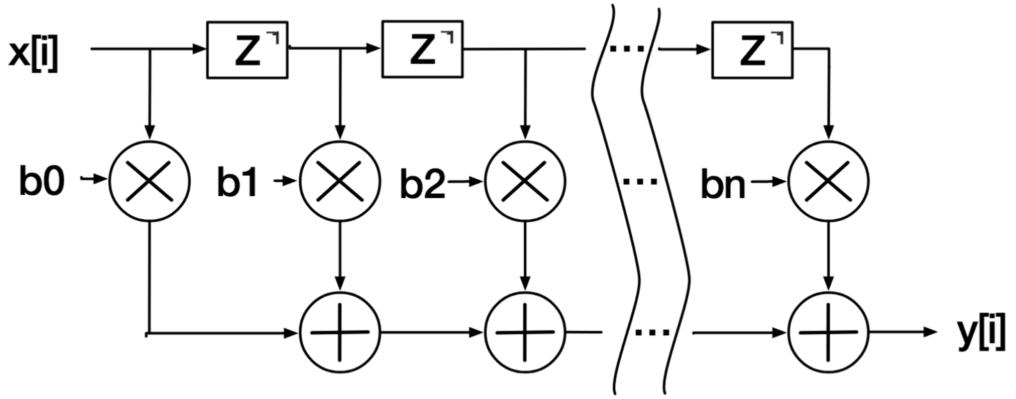
Introduction

What is the Chisel Working Group (CWG)?

What is Chisel?

- **Constructing** **H**ardware **I**n a **S**cala **E**mbedded **L**anguage
- 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



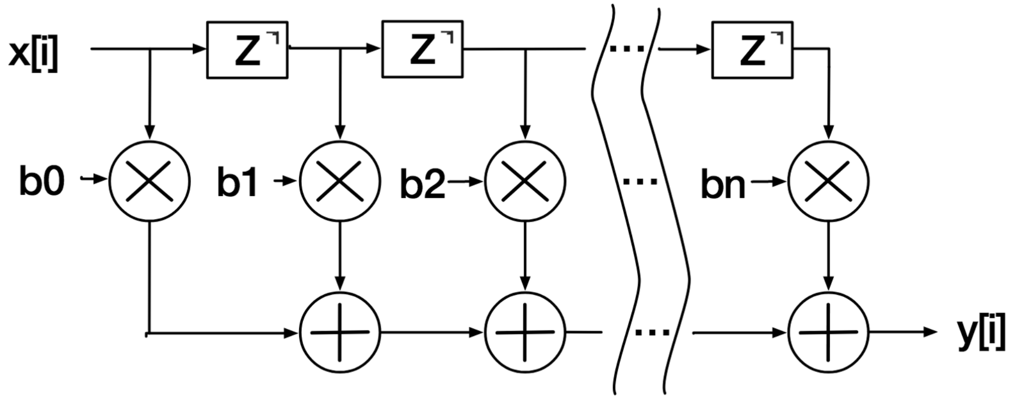
FIR Filter - 3-point moving sum

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)

  io.out := (io.in * 1.U) + (z1 * 1.U) + (z2 * 1.U)
}
```

Massive Increase in Parameterizability: “Software-like” Chisel

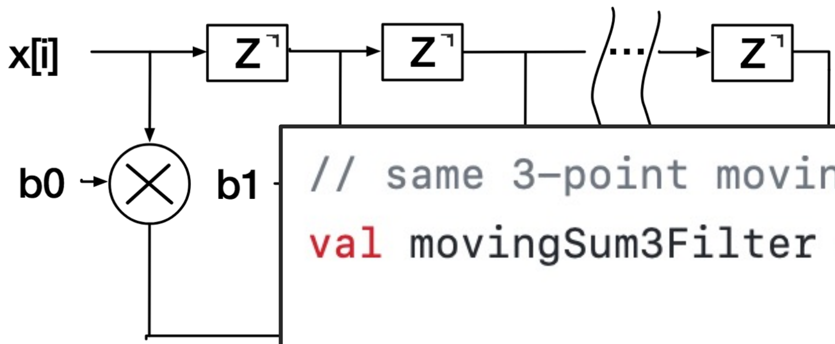


FIR Filter - Parameterized by bitwidth and *coeffients* with no loss of expressibility or performance.

Meta-programming enables powerful parameterization.

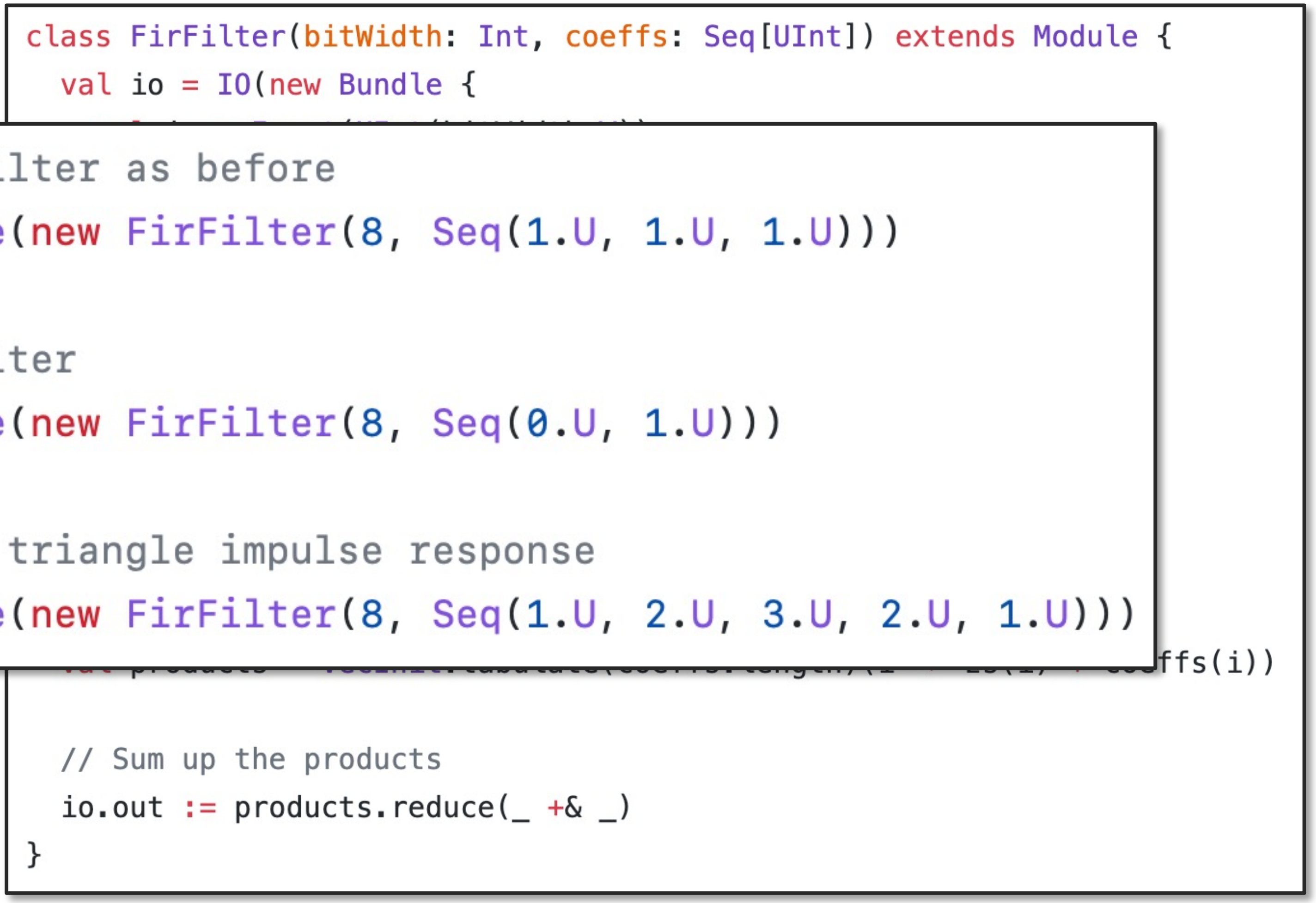
```
class FirFilter(bitWidth: Int, coeffs: Seq[UInt]) extends Module {
  val io = IO(new Bundle {
    val in = Input(UInt(bitWidth.W))
    val out = Output(UInt())
  })
  // Create the serial-in, parallel-out shift register
  val zs = Reg(Vec(coeffs.length, UInt(bitWidth.W)))
  zs(0) := io.in
  for (i <- 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



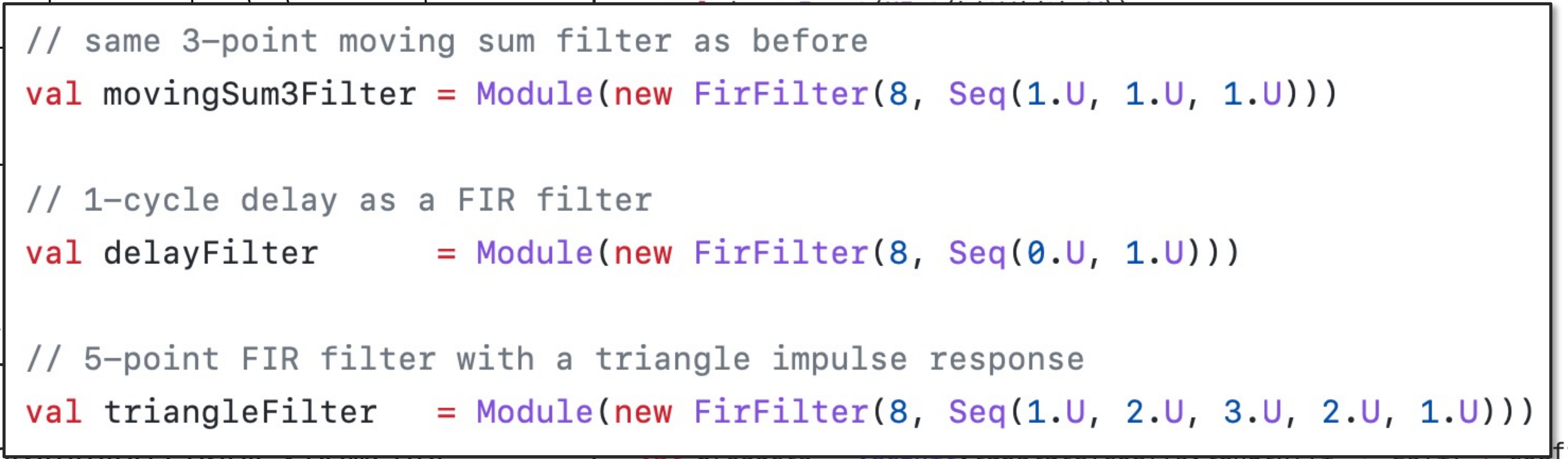
FIR Filter
coeffients
performar

Meta-programming enables powerful parameterization.



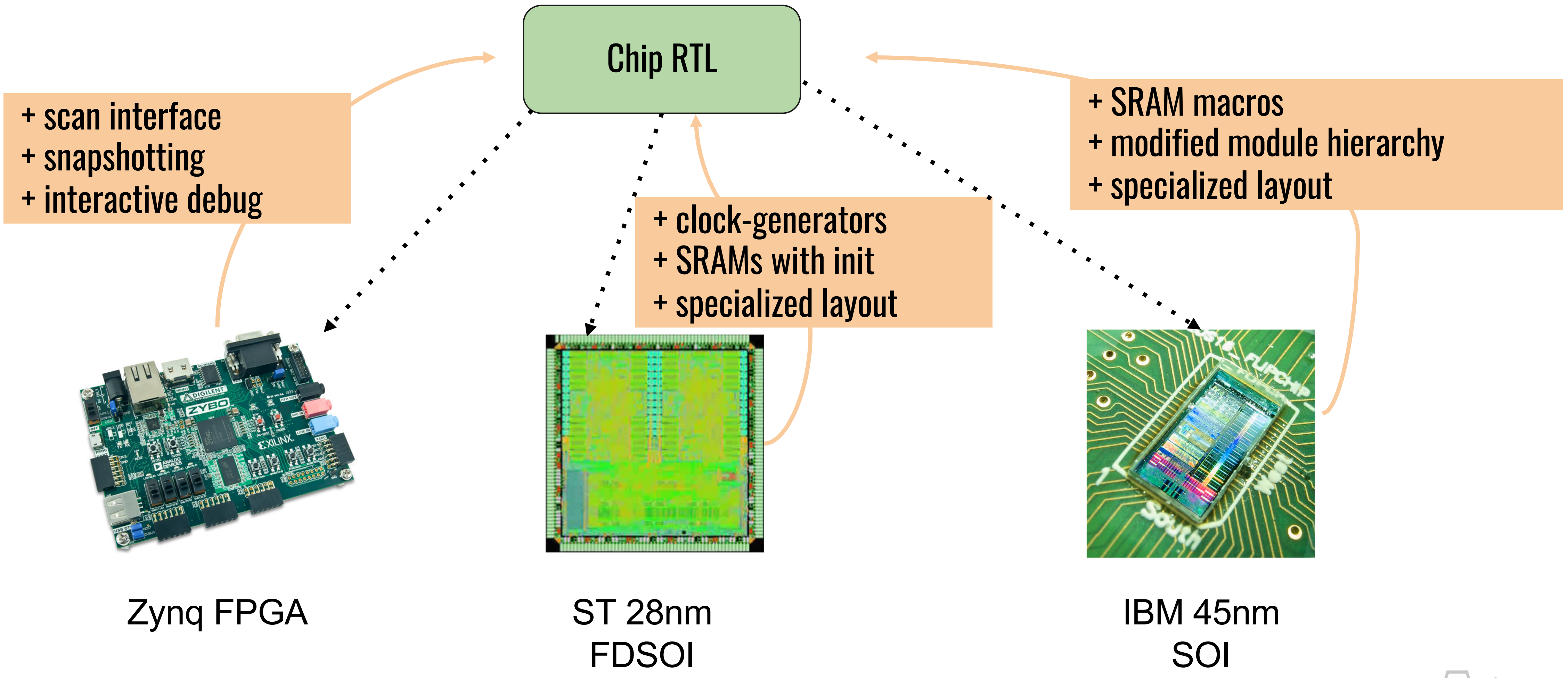
```
class FirFilter(bitWidth: Int, coeffs: Seq[UInt]) extends Module {
  val io = IO(new Bundle {
```

```
ffs(i))

  // Sum up the products
  io.out := products.reduce(_ +& _)
}
```



```
// 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
val triangleFilter   = Module(new FirFilter(8, Seq(1.U, 2.U, 3.U, 2.U, 1.U)))
```

Platform-Specific or Application-Specific RTL Changes

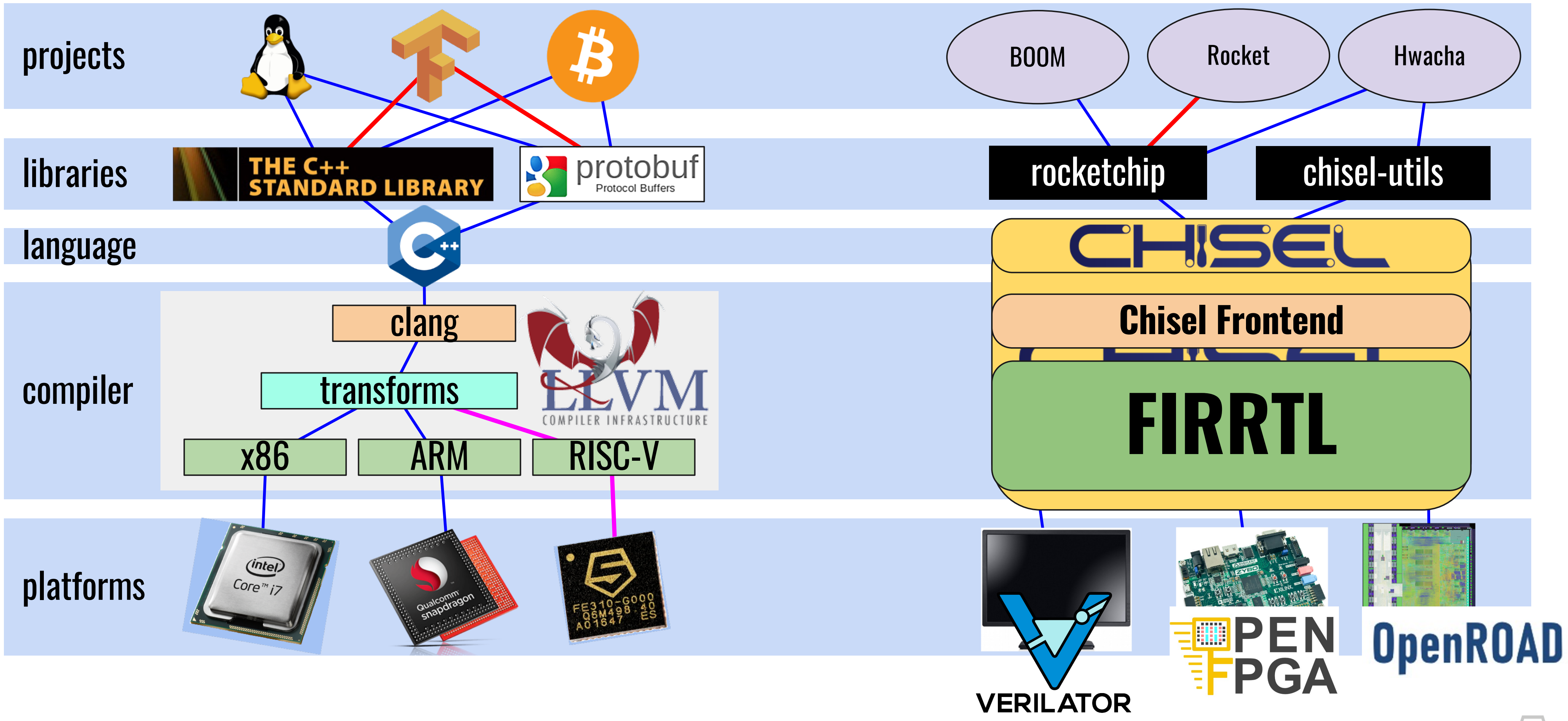


+ scan interface
+ snapshotting
+ interactive debug

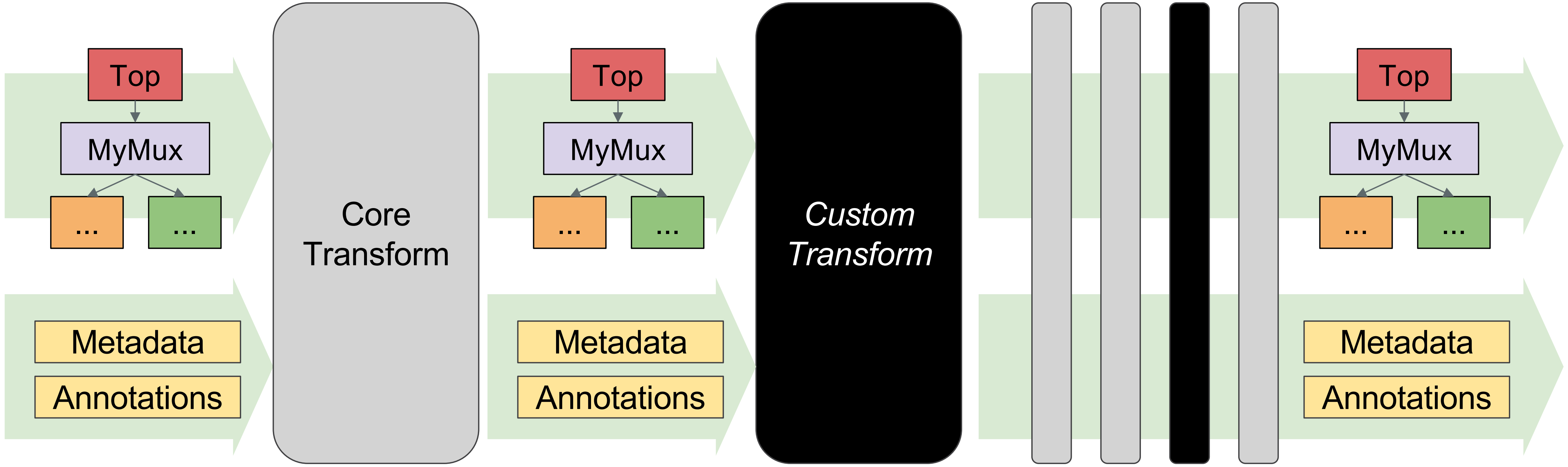
+ clock-generators
+ SRAMs with init
+ specialized layout

+ SRAM macros
+ modified module hierarchy
+ specialized layout

Realization: We need a software stack, but for hardware



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

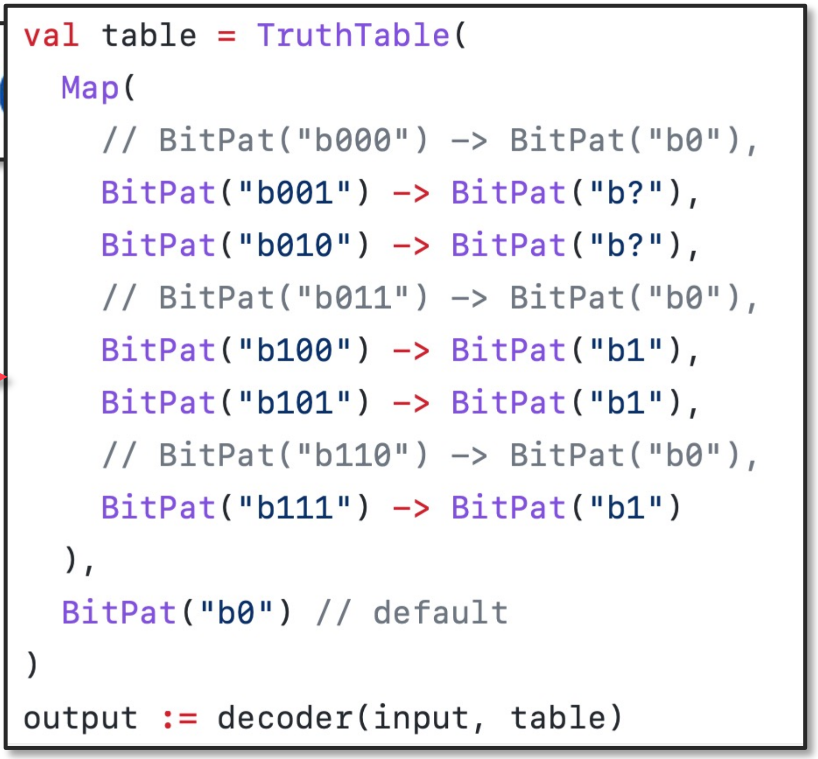
```
Vec.Lit(0xa.U, 0xb.U)
```

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

```
Vec.Lit(
```



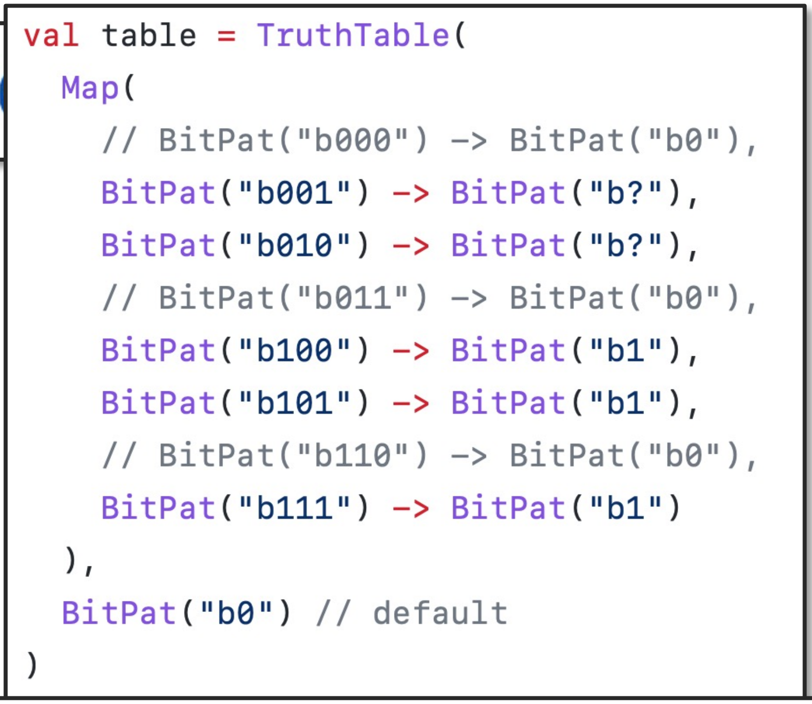
```
val table = TruthTable(
  Map(
    // BitPat("b000") -> BitPat("b0"),
    BitPat("b001") -> BitPat("b?"),
    BitPat("b010") -> BitPat("b?"),
    // BitPat("b011") -> BitPat("b0"),
    BitPat("b100") -> BitPat("b1"),
    BitPat("b101") -> BitPat("b1"),
    // BitPat("b110") -> BitPat("b0"),
    BitPat("b111")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

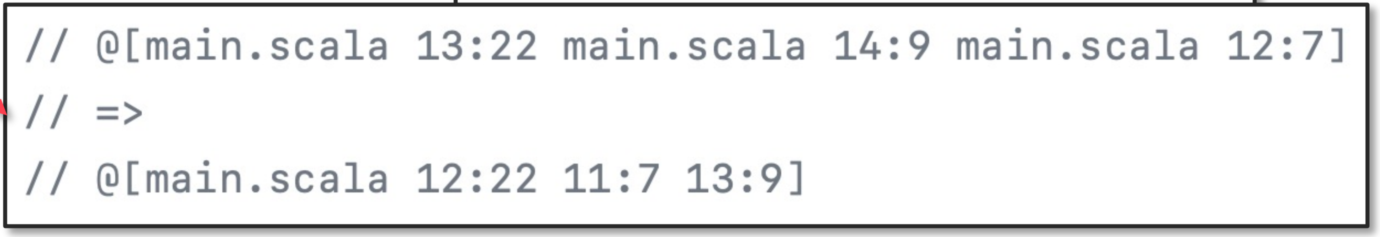
Chisel v3.5.0-RC1 Released!

- 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

```
Vec.Lit(
```



```
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
)
```



```
// @[main.scala 13:22 main.scala 14:9 main.scala 12:7]
// =>
// @[main.scala 12:22 11:7 13:9]
```

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

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

```
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 ChiselScalatestTester with Formal {

  "Quiz15" should "pass with WriteFirst" in {
    verify(new Quiz15, Seq(BoundedCheck(5)))
  }
}
```

See Kevin Laeufer's WOSET Paper https://woset-workshop.github.io/WOS

Definition / Instance

- Historically, Chisel elaborates every module instance and then deduplicates structurally equivalent modules
- New experimental API for definining (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

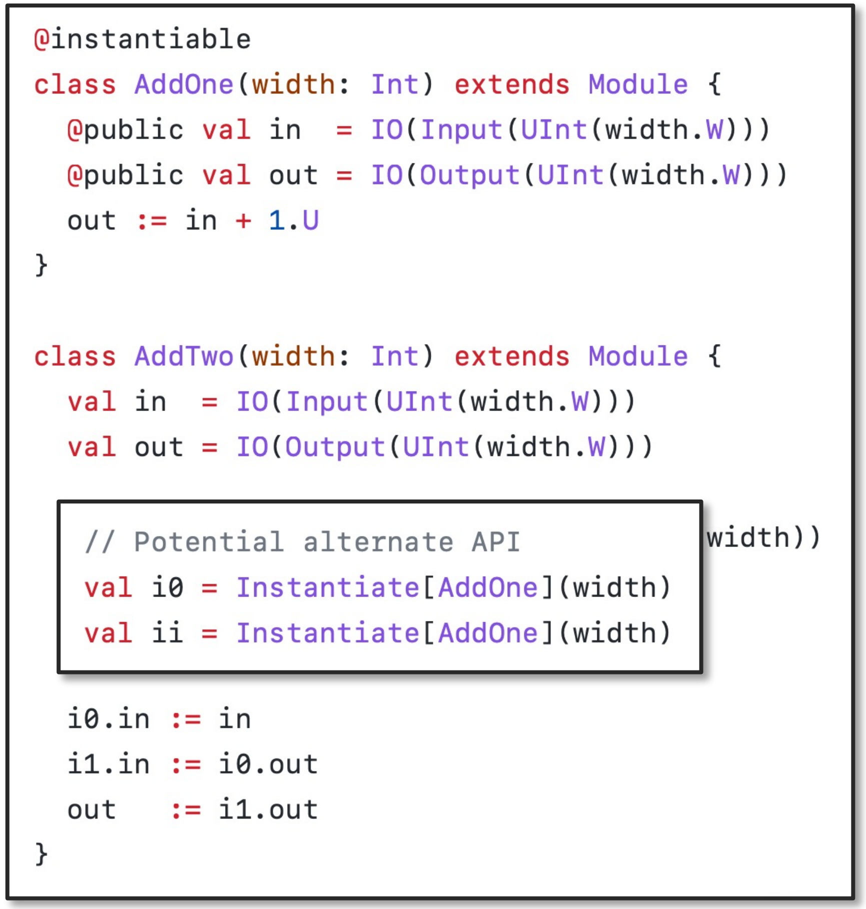
- Historically, Chisel elaborates every module instance and then deduplicates structurally equivalent modules
- New experimental API for definining (and elaborating) a module once and instantiating multiple times
  - Definition – Elaborates implementation of module
  - Instance – Merely instantiates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

```
@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
}
```

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 definining (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



```
@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)))

                                        width))


  i0.in := in
  i1.in := i0.out
  out   := i1.out
}
```

```
// Potential alternate API
val i0 = Instantiate[AddOne](width)
val ii = Instantiate[AddOne](width)
```

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

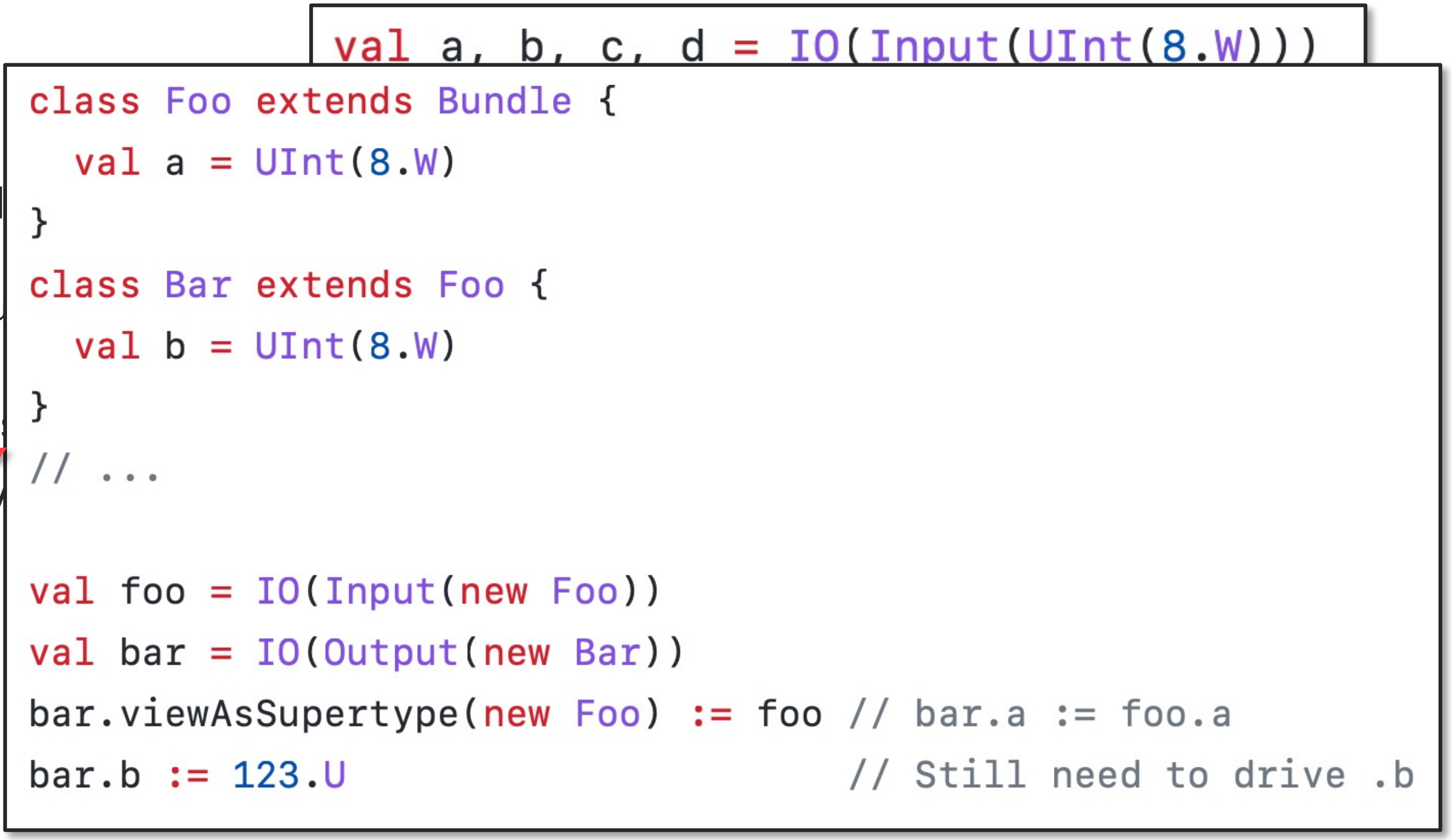
```
val a, b, c, d = IO(Input(UInt(8.W)))
val w, x, y, z = IO(Output(UInt(8.W)))
((w, x), (y, z)) := ((a, b), (c, d))
```

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



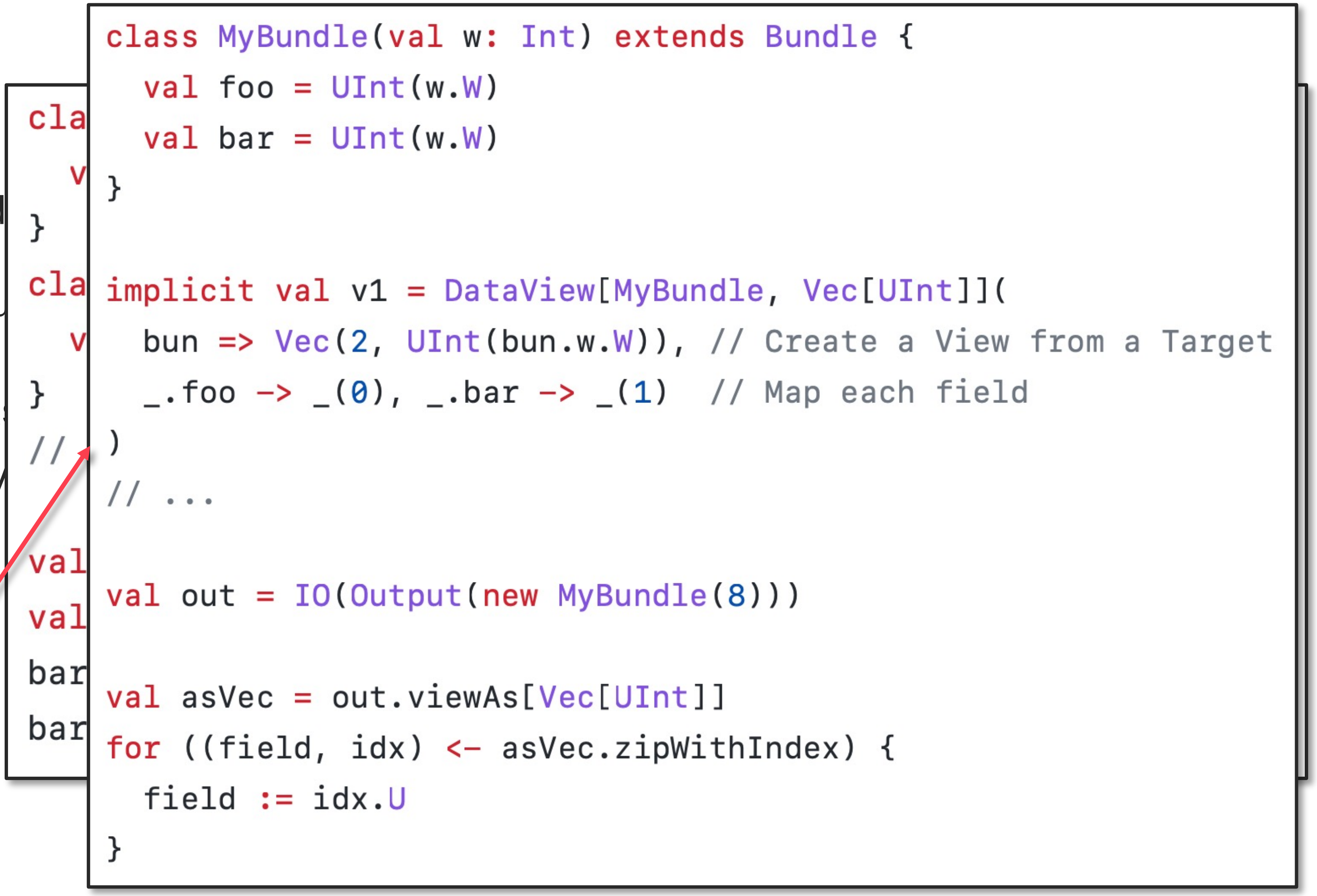
```
val a, b, c, d = IO(Input(UInt(8.W)))
```

```
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.viewAsSupertype(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

DataView

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



```
cla
  v
}
cla
  v
}
//

val
val
bar
bar
```

```
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 asVec = out.viewAs[Vec[UInt]]
for ((field, idx) <- asVec.zipWithIndex) {
  field := idx.U
}
```

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

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:

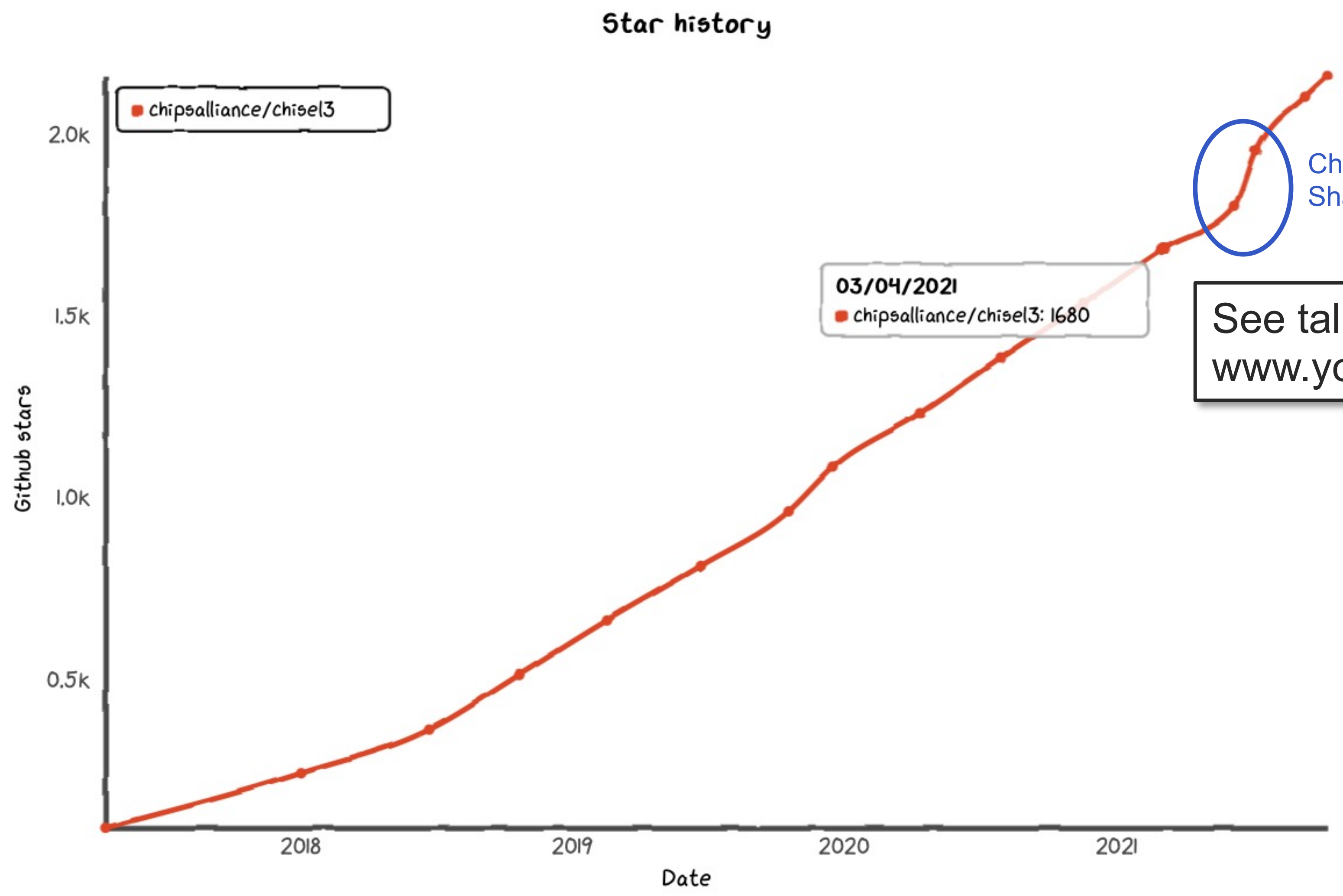
```
class MyBundle(w: Int) extends Bundle {
  val foo = UInt(w.W)
  override def cloneType = new MyBundle(w).asInstanceOf[this.type]
}
```

After:

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

Community

Continued Growth

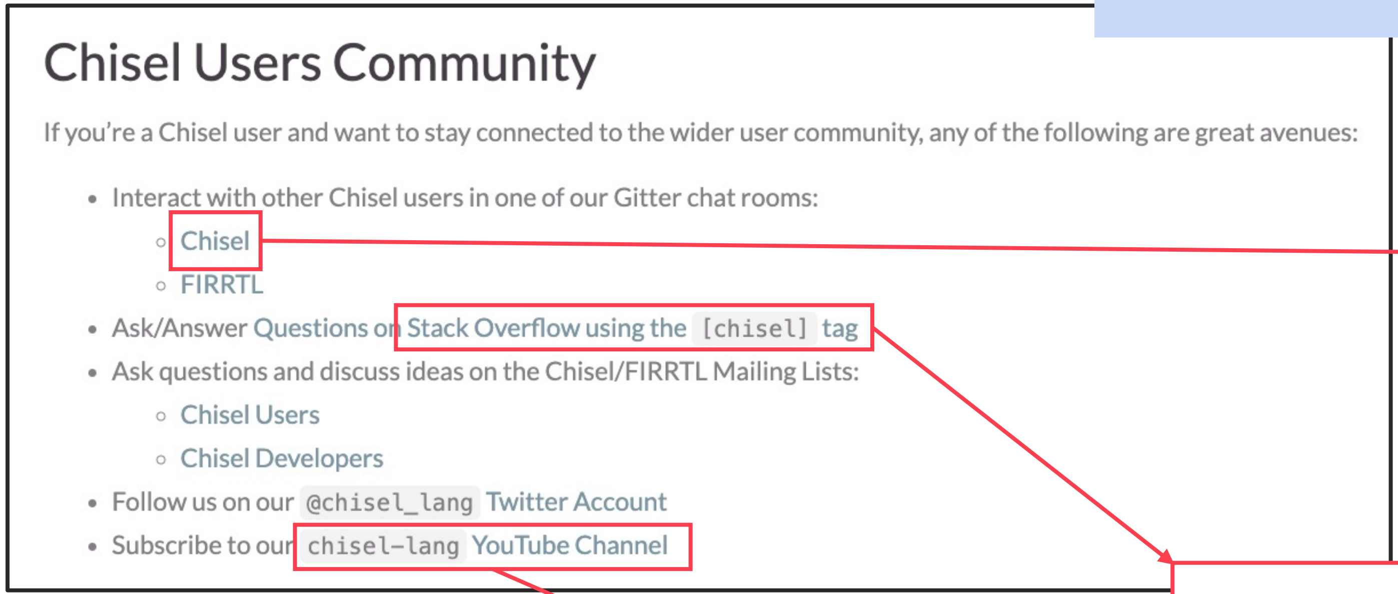


Chisel Community Conference
Shanghai, June 2021

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

Get Involved

www.chisel-lang.org

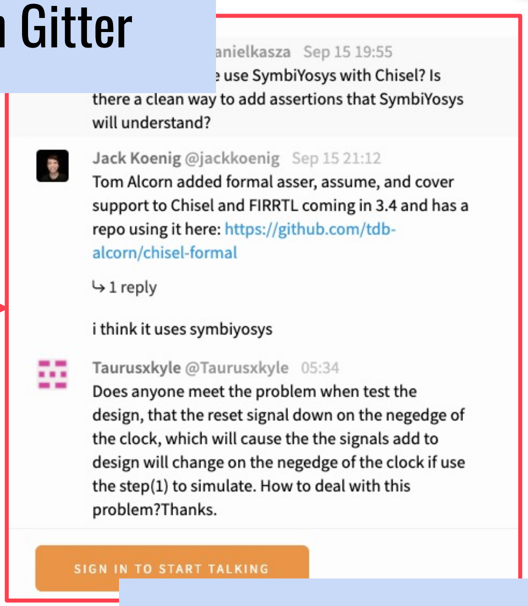


Chisel Users Community

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



anielkasza Sep 15 19:55
e use SymbiYosys with Chisel? Is there a clean way to add assertions that SymbiYosys will understand?

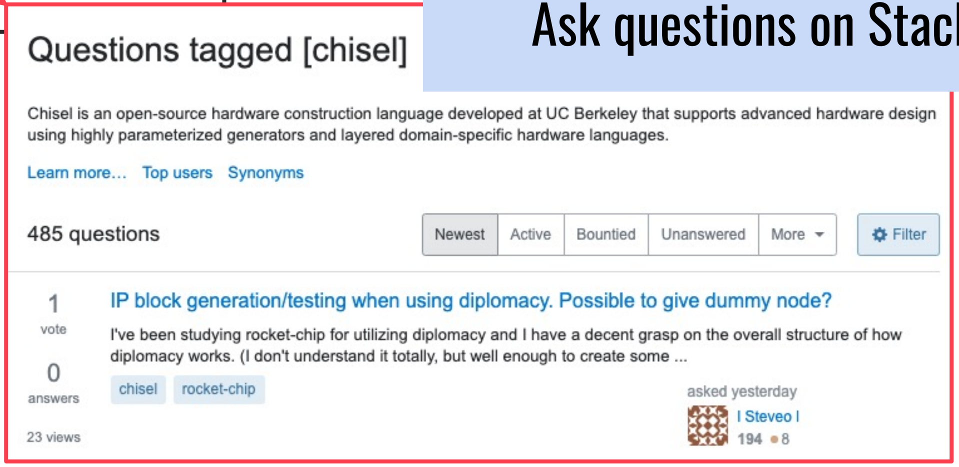
Jack Koenig @jackkoenig Sep 15 21:12
Tom Alcorn added formal asser, assume, and cover support to Chisel and FIRRTL coming in 3.4 and has a repo using it here: https://github.com/tdb-alcorn/chisel-formal

↳ 1 reply

i think it uses symbiyosys

Taurusxkyle @Taurusxkyle 05:34
Does anyone meet the problem when test the design, that the reset signal down on the negedge of the clock, which will cause the the signals add to design will change on the negedge of the clock if use the step(1) to simulate. How to deal with this problem?Thanks.

Ask questions on StackOverflow



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 Synonyms

485 questions

1 vote | 0 answers | 23 views

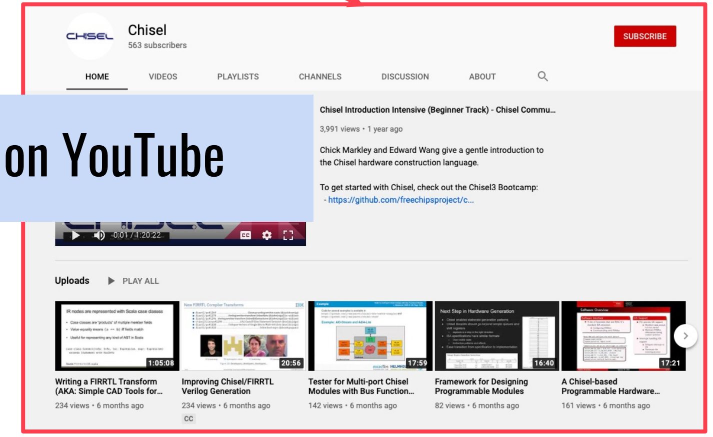
IP block generation/testing when using diplomacy. Possible to give dummy node?

I've been studying rocket-chip for utilizing diplomacy and I have a decent grasp on the overall structure of how diplomacy works. (I don't understand it totally, but well enough to create some ...

chisel rocket-chip

asked yesterday
I Steveo I
194 ●8

Watch talks on YouTube



Chisel
563 subscribers

Chisel Introduction Intensive (Beginner Track) - Chisel Commu...
3,991 views • 1 year ago
Chick Markley and Edward Wang give a gentle introduction to the Chisel hardware construction language.

To get started with Chisel, check out the Chisel3 Bootcamp:
- https://github.com/freechipsproject/c...

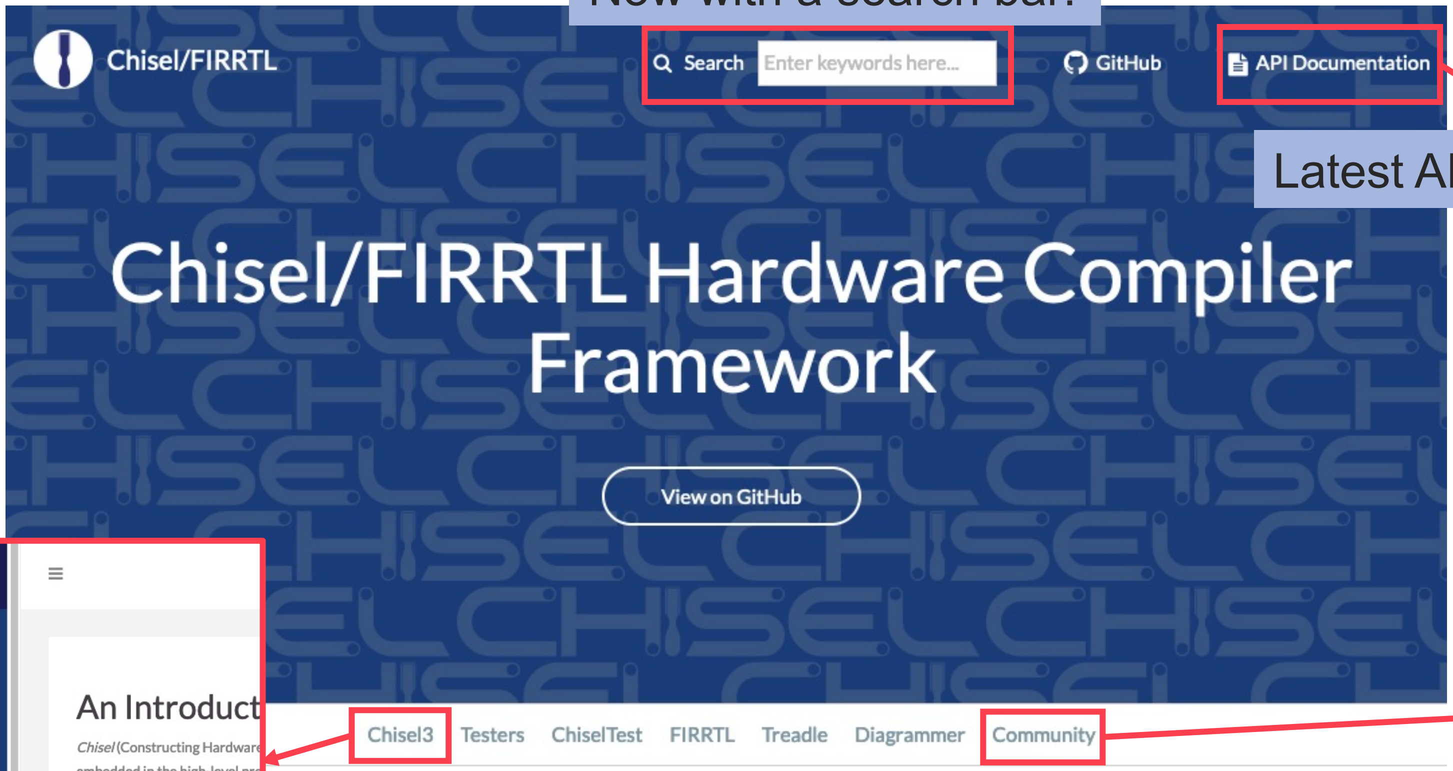
Writing a FIRRTL Transform (AKA: Simple CAD Tools for... — 1:05:08 — 234 views • 6 months ago
Improving Chisel/FIRRTL Verilog Generation — 20:56 — 234 views • 6 months ago — CC
Tester for Multi-port Chisel Modules with Bus Function... — 17:59 — 142 views • 6 months ago
Framework for Designing Programmable Modules — 16:40 — 82 views • 6 months ago
A Chisel-based Programmable Hardware... — 17:21 — 161 views • 6 months ago

Extra / Old Slides

www.chisel-lang.org

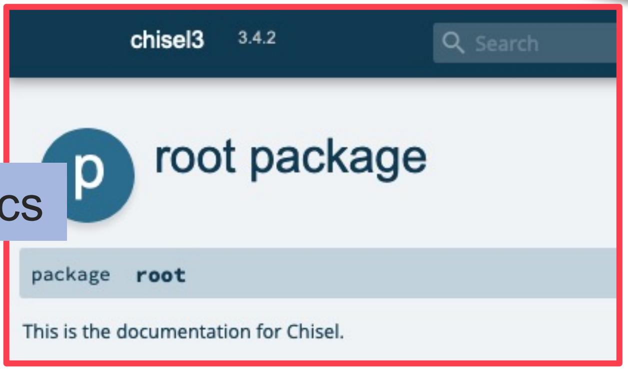
Further Improved Website

Now with a search bar!



Chisel/FIRRTL Hardware Compiler Framework

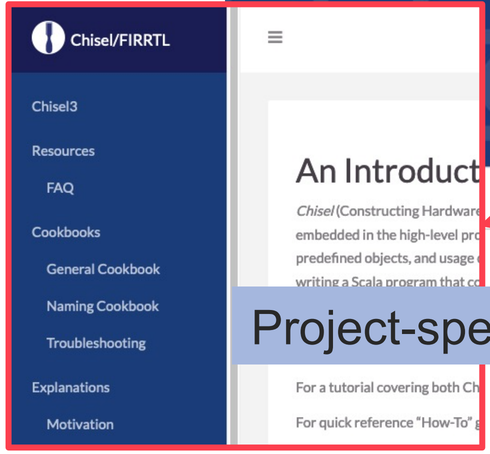
Latest API Docs



root package

```
package root
```

This is the documentation for Chisel.



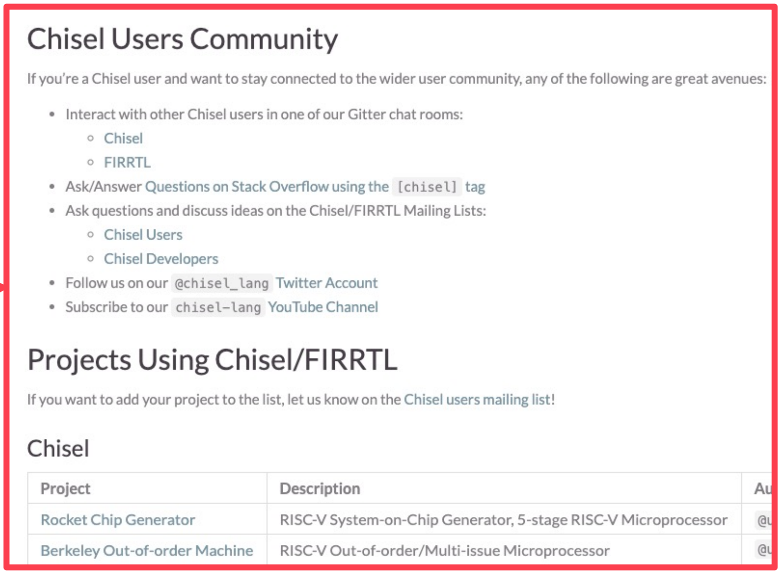
An Introduct

Chisel (Constructing Hardware
embedded in the high-level pro
predefined objects, and usage
writing a Scala program that co

For a tutorial covering both Ch

For quick reference "How-To" g

Project-specific documentation



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

Projects Using Chisel/FIRRTL

If you want to add your project to the list, let us know on the Chisel users mailing list!

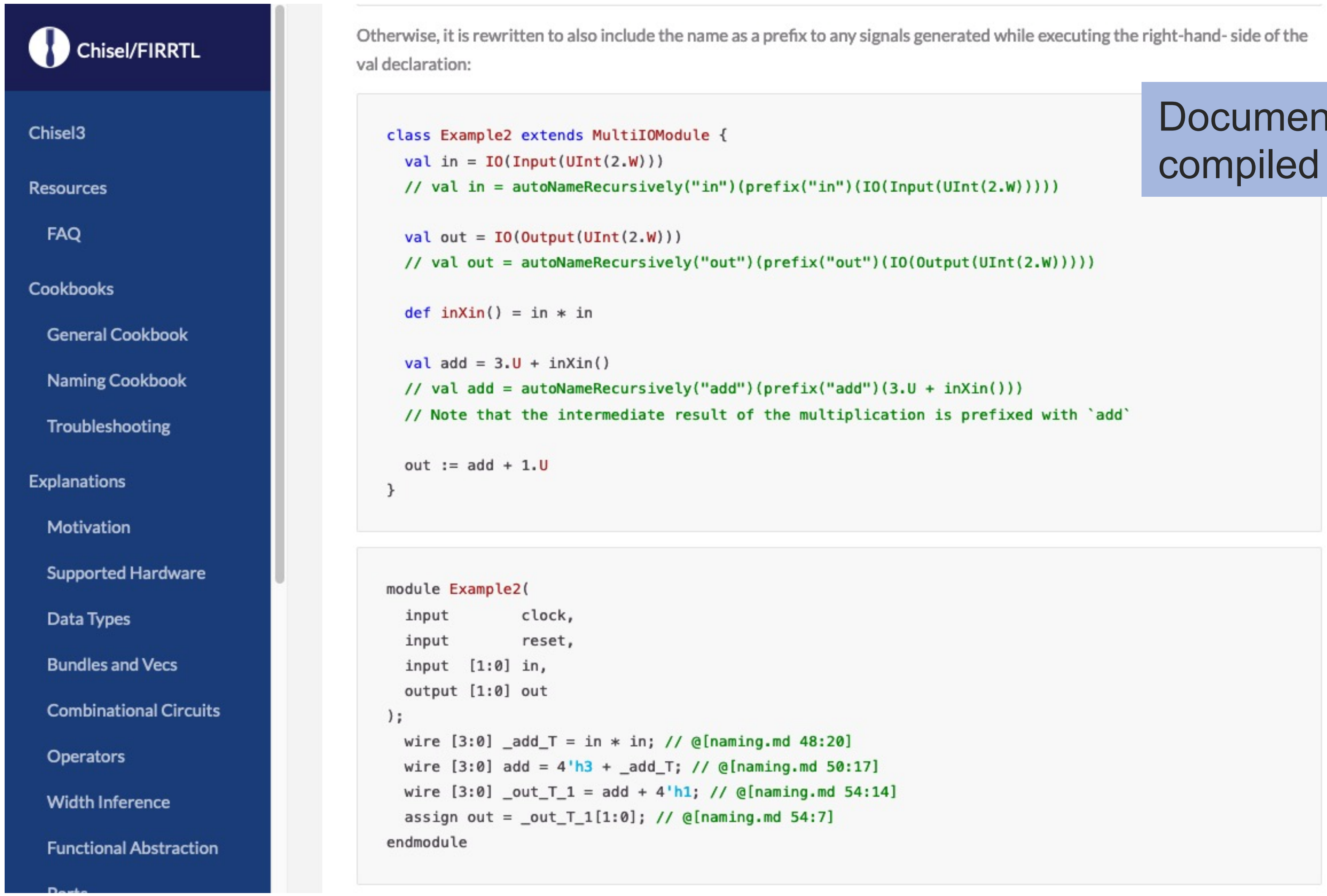
Chisel

Project	Description	Au
Rocket Chip Generator	RISC-V System-on-Chip Generator, 5-stage RISC-V Microprocessor	@
Berkeley Out-of-order Machine	RISC-V Out-of-order/Multi-issue Microprocessor	@

Community page

Further Improved Website

www.chisel-lang.org



Chisel/FIRRTL

- Chisel3
- Resources
  - FAQ
- Cookbooks
  - General Cookbook
  - Naming Cookbook
  - Troubleshooting
- Explanations
  - Motivation
  - Supported Hardware
  - Data Types
  - Bundles and Vecs
  - Combinational Circuits
  - Operators
  - Width Inference
  - Functional Abstraction

Otherwise, it is rewritten to also include the name as a prefix to any signals generated while executing the right-hand- side of the val declaration:

```scala
class Example2 extends MultiIOModule {
  val in = IO(Input(UInt(2.W)))
  // val in = autoNameRecursively("in")(prefix("in")(IO(Input(UInt(2.W)))))

  val out = IO(Output(UInt(2.W)))
  // val out = autoNameRecursively("out")(prefix("out")(IO(Output(UInt(2.W)))))

  def inXin() = in * in

  val add = 3.U + inXin()
  // val add = autoNameRecursively("add")(prefix("add")(3.U + inXin()))
  // Note that the intermediate result of the multiplication is prefixed with `add`

  out := add + 1.U
}
```

```verilog
module Example2(
  input        clock,
  input        reset,
  input  [1:0] in,
  output [1:0] out
);
  wire [3:0] _add_T = in * in; // @[naming.md 48:20]
  wire [3:0] add = 4'h3 + _add_T; // @[naming.md 50:17]
  wire [3:0] _out_T_1 = add + 4'h1; // @[naming.md 54:14]
  assign out = _out_T_1[1:0]; // @[naming.md 54:7]
endmodule
```

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

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

Old* Verilog

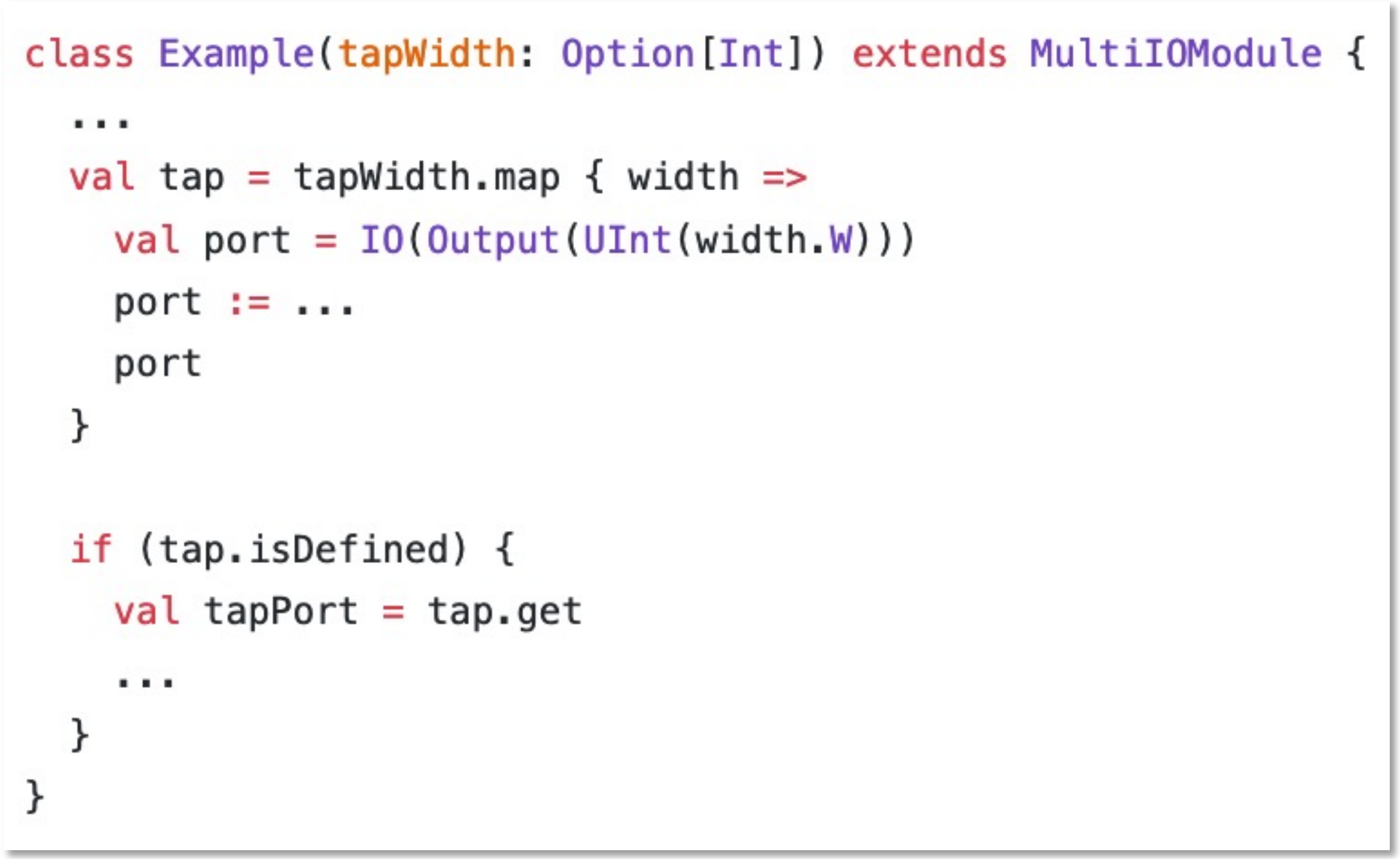
```
wire [7:0] _T_1 = a + b;
wire [7:0] _T_3 = _T_1 - 8'h3;
wire [7:0] _T_4 = _T_3 & 8'hcf;
assign out = _T_4 | 8'h8;
```

Chisel 3.4 Verilog

```
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;
```

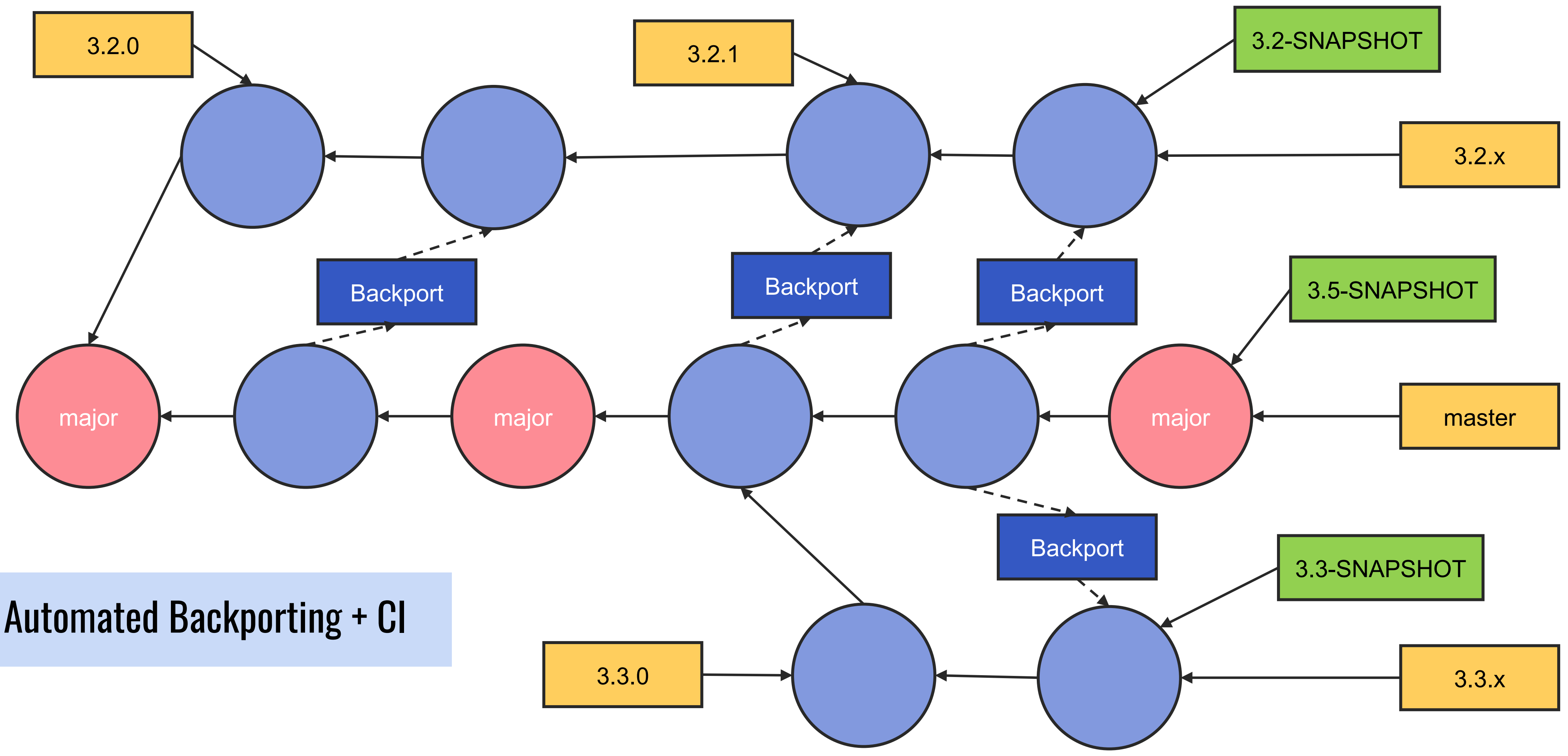
Refined Signal Naming

- 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!



```
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