Top 10 Common Misconceptions about Chisel

Zihao Yu
ICT, CAS

2021.06@ShangHai

中国科学院计算技术研究所
Institute Of Computing Technology Chinese Academy Of Sciences

Top 10 common misconceptions about Chisel

- Chisel has been proposed for nearly 10 years
- However, people may still come up with some misconceptions about Chisel

- We aims to collect top 10 common misconceptions about Chisel
  - and provide corresponding clarification
  - according to the Chisel experience from ICT (Institute of Computing Technology), CAS (Chinese Academy of Sciences) team

- a wide variety of topics
  - from learning, developing, testing and verification
  - to debugging, optimization, and physical design

Learning - Misconception 1

- The Rocket Chip project written by Chisel is complicated and hard to understand. Therefore Chisel is also complicated and hard to learn.

- **No**. There are a lot of DSLs inside the Rocket Chip Project. They do not belong to Chisel.
  - CDE(the configuration system)
  - Diplomacy(the bus framework)
  - RegMapper(device register mapping definition)
- The range of Chisel is defined in the manual.

Learning - Misconception 1

- Suggestion: If you are new to Chisel, **DO NOT** read the source code of Rocket Chip. Instead, try the following materials:
  - Chisel Bootcamp - support running Chisel online
    - https://github.com/freechipsproject/chisel-bootcamp
  - Chisel Users Guide - deep introduction to key concepts in Chisel
    - https://www.chisel-lang.org/chisel3/docs/introduction.html
  - Chisel cheat sheet - a 2-page reference of the frequent Chisel usage
    - https://github.com/freechipsproject/chisel-cheatsheet/releases/latest/download/chisel_cheatsheet.pdf
  - Chisel API Documentation
    - https://www.chisel-lang.org/api/latest/chisel3/index.html
  - Digital Design With Chisel
    - https://github.com/schoeberl/chisel-book

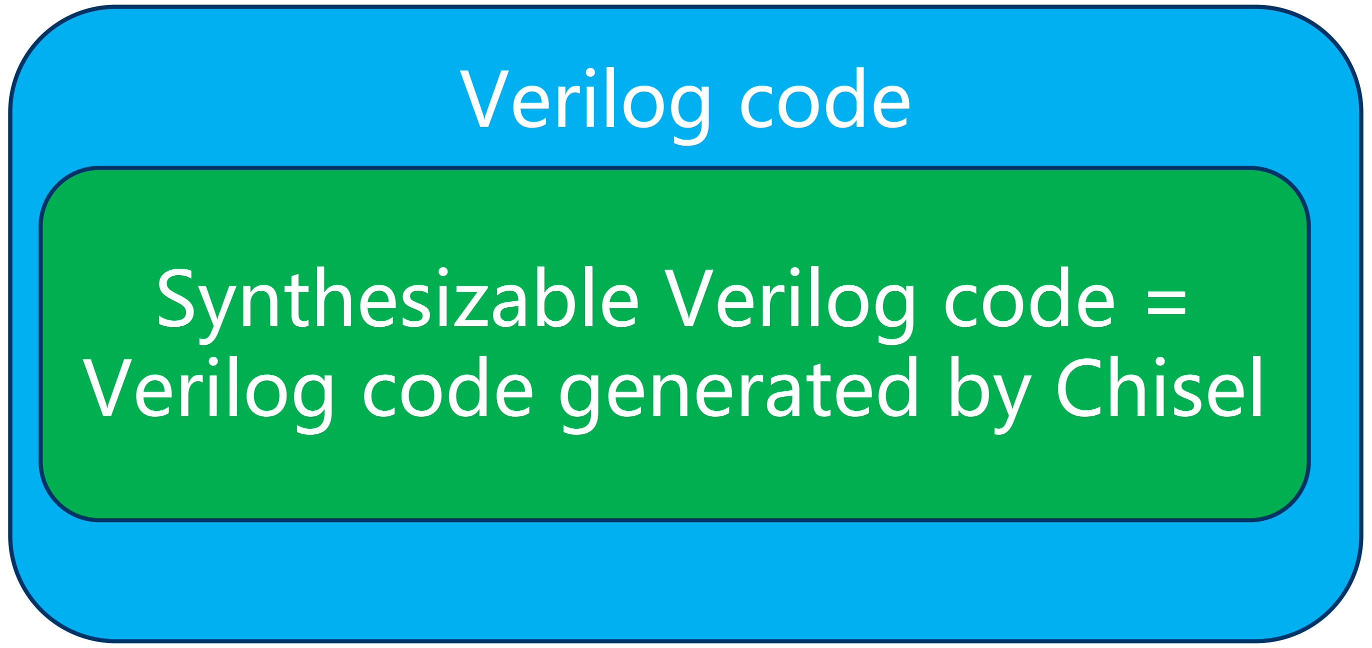
Learning - Misconception 1

- Suggestion: Feel free to try the following example projects:
  - Chisel Project Template
    - https://github.com/freechipsproject/chisel-template
  - Chisel Project Template (mill version)
    - https://github.com/OpenXiangShan/chisel-playground
  - Sodor Processor
    - https://github.com/ucb-bar/riscv-sodor
  - NutShell (果壳) - a processor developed by undergraduates (一生一芯项目)
    - https://github.com/OSCPU/NutShell

Developing - Misconception 2

- For RTL development, there is something that Verilog can do but Chisel can not.

- **No**. Synthesizable circuit = instantiation + wiring
  - Instantiate modules, combination logic, registers...
  - They are supported by Chisel.



Verilog code
Synthesizable Verilog code = Verilog code generated by Chisel

Developing - Misconception 3

- It is annoying to fix so many hard-to-understand compile errors with Chisel. It is easier to pass compilation with Verilog.
- 5 types of error during development
  - Scala compile error
    - Syntax error, static type checking error
  - Scala run-time error
    - Dereferencing null object, dynamic type checking error, requirement fail
  - Chisel build error
    - Wrong direction of wiring, Chisel type/Chisel object mismatch
  - FIRRTL transform error
    - Unconnected signal, combinational loop
  - Circuit simulation error
    - Assertion fail, functional bug

Developing - Misconception 3

- It is annoying to fix so many hard-to-understand compile errors with Chisel. It is easier to pass compilation with Verilog.

- This should be considered from the view of programming language.

- code successfully compiled ≠ correct code
- Three concepts about debugging from software engineering
  - Fault - buggy code, e.g. uninitialized variable
  - Error - unexpected state during run time, e.g. returning a garbage value
  - Failure - observable fatal result, e.g. segmentation fault

Developing - Misconception 3

- bug propagation
  - fault ->(maybe) error ->(maybe) failure
- In Verilog, wrong signal connection -> CPU reads wrong data -> memory access exception -> days of debugging
- In Chisel, wrong signal connection -> Scala static type checking error (w.h.p.) -> expose fault w.h.p.

- Stronger compiler can expose more hidden fault to observable failure.
  - Code successfully compiled may be correct with higher probability.
  - In a long run, spending hours to solve an error reported by compiler is still cost-effective.

Developing - Misconception 4

- There is no way to integrate a Chisel module into a Verilog project, and vice versa.

- **No**.
  - Chisel code can be compiled to Verilog.
  - Verilog code can be integrated into a Chisel project with Blackbox.
    - Example from www.chisel-lang.org

```
import chisel3._
class BlackBoxRealAdd extends BlackBox {
  val io = IO(new Bundle {
    val in1 = Input(UInt(64.W))
    val in2 = Input(UInt(64.W))
    val out = Output(UInt(64.W))
  })
}
```

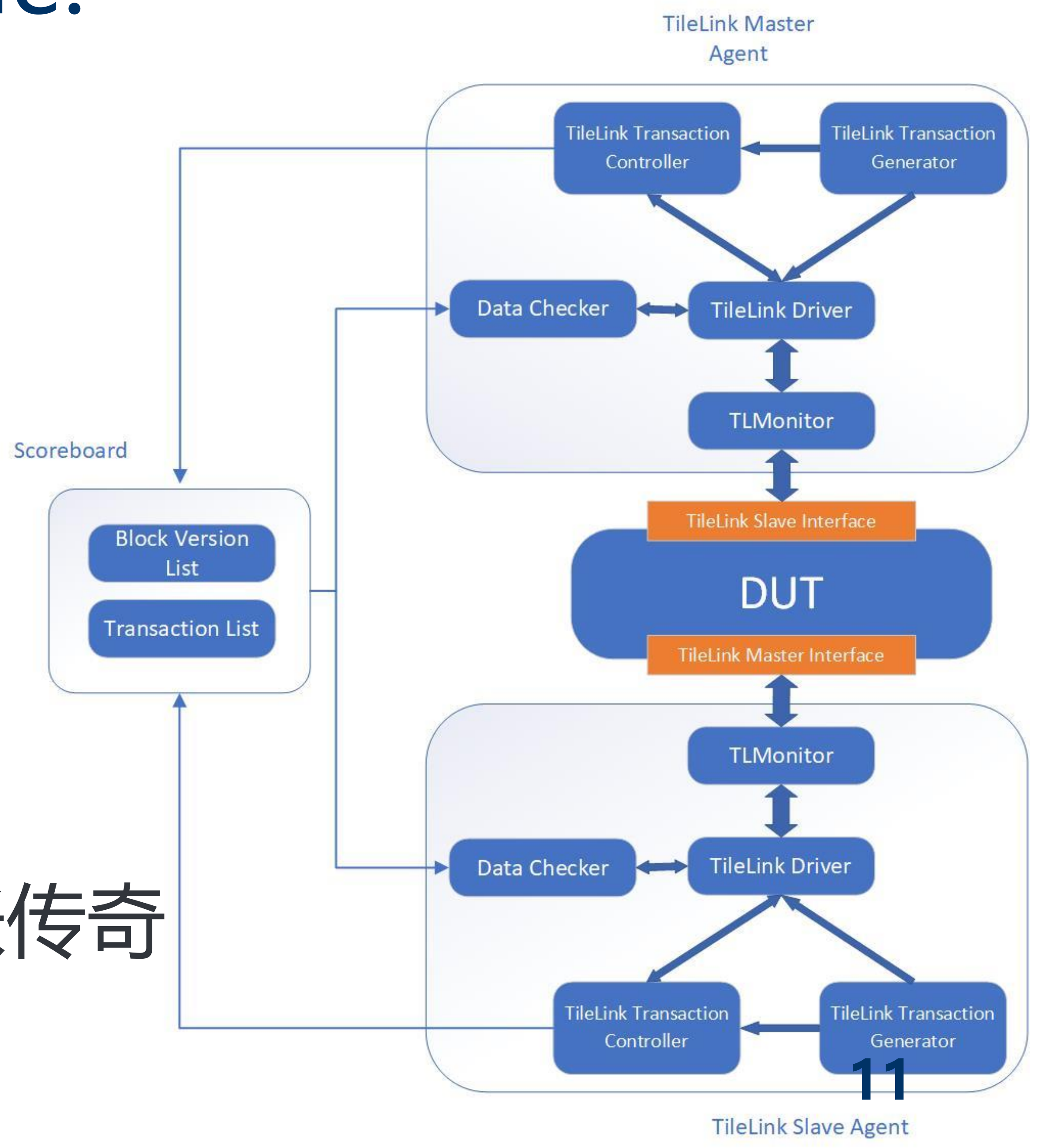
```
module BlackBoxRealAdd(
    input  [63:0] in1,
    input  [63:0] in2,
    output reg [63:0] out
);
  always @* begin
    out <= $realtobits($bitstoreal(in1) + $bitstoreal(in2));
  end
endmodule
```

Testing and Verification - Misconception 5

- There is no way to perform simulation or testing with Chisel.

- **No**. There are two ways to test the Chisel code.
  - test from Chisel layer - write testbench with Scala
    - Chisel Testers
      - https://github.com/freechipsproject/chisel-testers
    - Chisel Testers2
      - https://github.com/ucb-bar/chisel-testers2
  - test from Verilog layer - UVM

- We develop UVM above Chisel Testers2.
  - Agent Faker: TL-C一致性Cache的软件测试框架, 张传奇



Testing and Verification - Misconception 6

- There is no way to guarantee the semantic equality between the Chisel code and the generated Verilog code.

- Unnecessary, since every tool is software, and it is difficult to guarantee to be bug-free.

- Commercial EDA tools may be buggy.
  - Segmentation fault -> modify the design?
- But why we believe them?
  - Because they have been used by many users.

Testing and Verification - Misconception 6

- Chisel and FIRRTL may also be buggy.
  - No one can prove that they are bug-free.
- But a lot of processors written in Chisel are successfully taped-out.
  - RocketChip (at least 11 times), BOOM, lowRISC…
- Cases in our team (ICT, CAS)
  - NutShell (taped-out in 2019.12)
    - https://github.com/OSCPU/NutShell
  - Labeled RISC-V (taped-out in 2020.07)
    - https://github.com/LvNA-system/labeled-RISC-V
  - XiangShan (taped-out in 2021.07, still in progress)
    - https://github.com/OpenXiangShan/XiangShan
- According to our experience, the behavior of Chisel and generated Verilog is still the same.

Quality - Misconception 7

- Since the performance of Java programs are slower than C programs, the performance of Chisel circuits will also be slower than Verilog circuits.

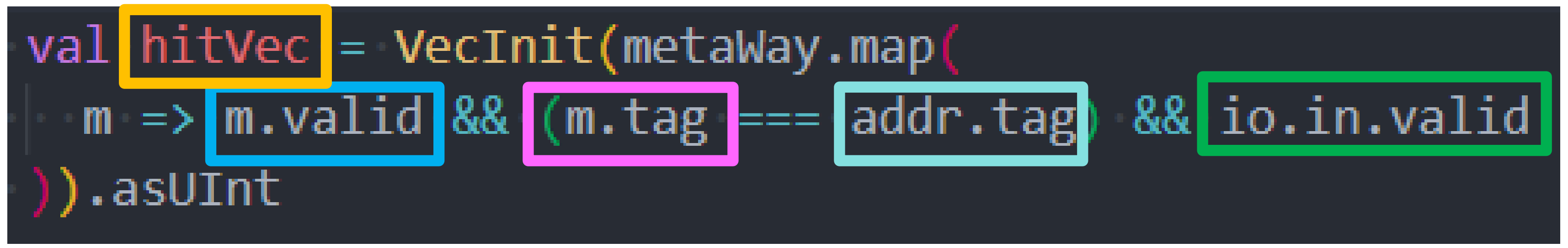
- **No**.
  - Java is slower than C, because of the Java runtime is heavier than C.
  - The performance of the circuit mainly depends on the design itself, not the language used to describe the circuit.
    - If you describe an adder, it should generate the adder exactly you want, no matter you use Chisel or Verilog.
- The performance of high level language has nothing to do with the performance of the circuit.

Quality - Misconception 8

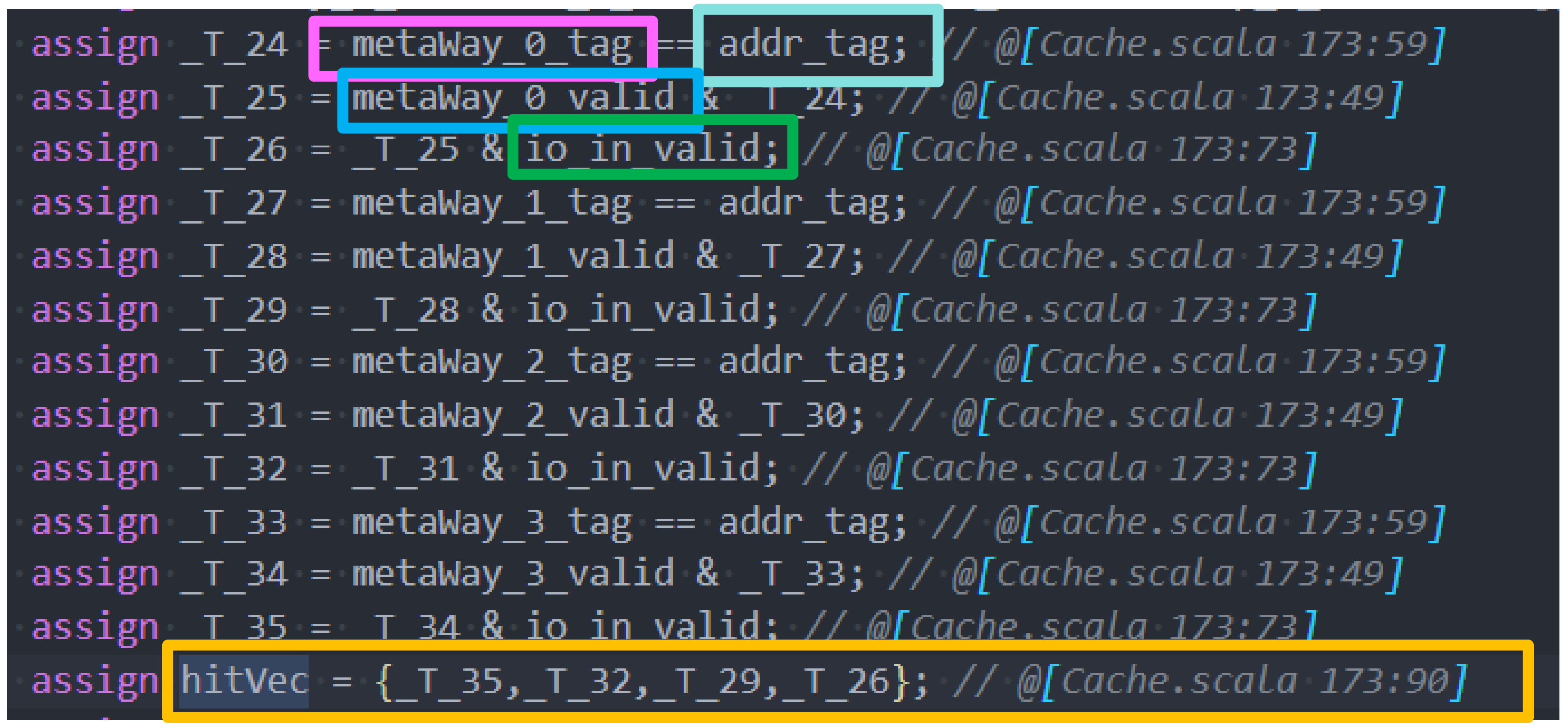
- Similar to HLS, writing Chisel code with advanced features may lead to bad PPA.

- **No. Chisel is not HLS.**
  - The advanced feature in Scala is used to describe the circuit conveniently.
  - still get the circuit exactly we want

- HLS generates circuit based on algorithm.
  - can not precisely control the generated circuit.



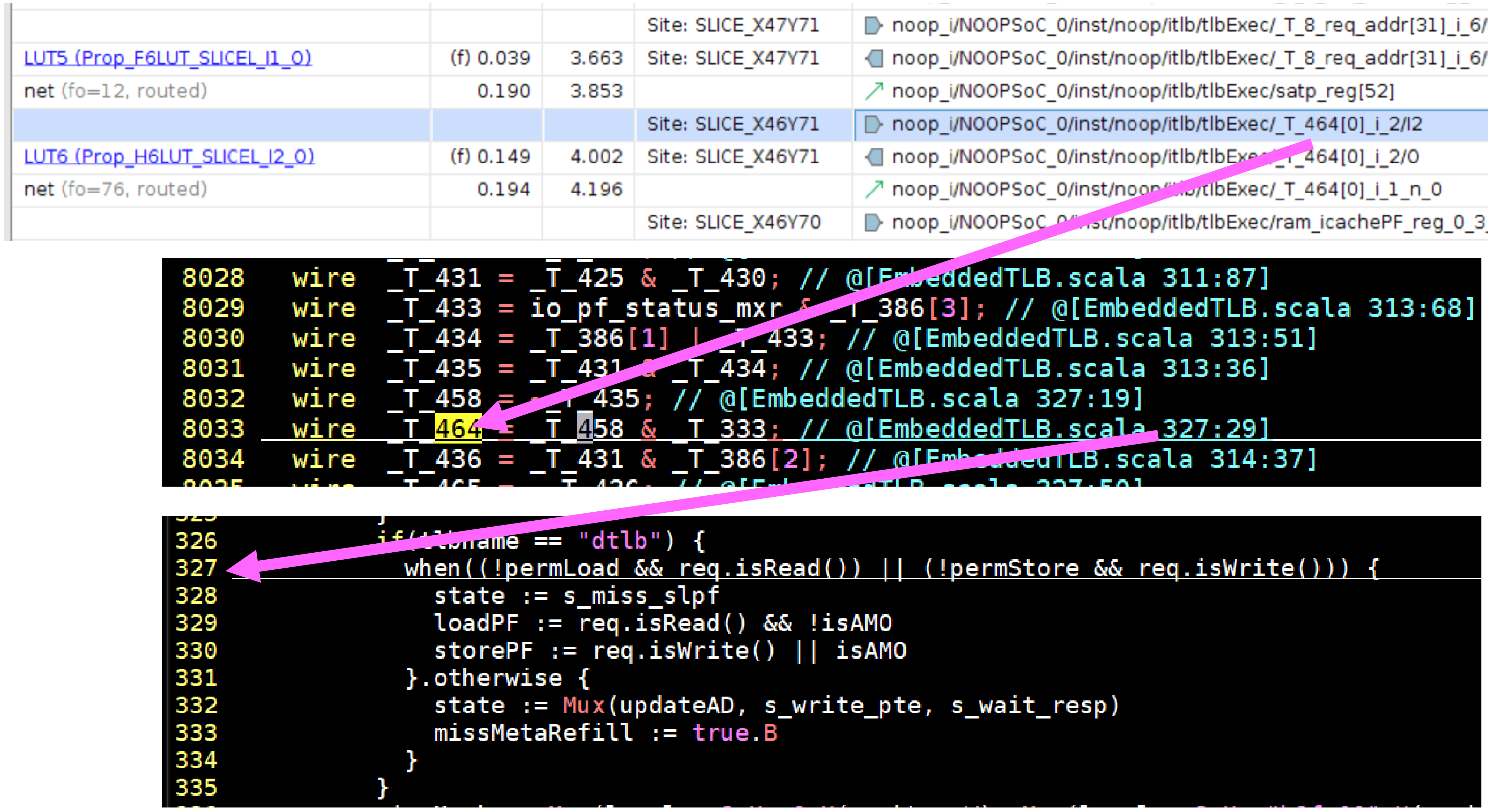
```
val hitVec = VecInit(metaWay.map(
  m => m.valid && (m.tag === addr.tag) && io.in.valid
)).asUInt
```



```
assign _T_24 = metaWay_0_tag == addr_tag; // @[Cache.scala 173:59]
assign _T_25 = metaWay_0_valid & _T_24; // @[Cache.scala 173:49]
assign _T_26 = _T_25 & io_in_valid; // @[Cache.scala 173:73]
assign _T_27 = metaWay_1_tag == addr_tag; // @[Cache.scala 173:59]
assign _T_28 = metaWay_1_valid & _T_27; // @[Cache.scala 173:49]
assign _T_29 = _T_28 & io_in_valid; // @[Cache.scala 173:73]
assign _T_30 = metaWay_2_tag == addr_tag; // @[Cache.scala 173:59]
assign _T_31 = metaWay_2_valid & _T_30; // @[Cache.scala 173:49]
assign _T_32 = _T_31 & io_in_valid; // @[Cache.scala 173:73]
assign _T_33 = metaWay_3_tag == addr_tag; // @[Cache.scala 173:59]
assign _T_34 = metaWay_3_valid & _T_33; // @[Cache.scala 173:49]
assign _T_35 = _T_34 & io_in_valid; // @[Cache.scala 173:73]
assign hitVec = {_T_35,_T_32,_T_29,_T_26}; // @[Cache.scala 173:90]
```

debugging, optimization and physical design - Misconception 9

- The Verilog generated by Chisel is hard to read. Therefore it is difficult to perform debugging, optimization and physical design.
- **No**. We can easily backtrack a path to Chisel code.
  - Path -> Verilog -> comment (with file name and line number) -> Chisel



			Site: SLICE_X47Y71	noop_i/NOOPSoC_0/inst/noop/itlb/tlbExec/_T_8_req_addr[31]_i_6/I1
LUT5 (Prop_F6LUT_SLICEL_I1_O)	(f) 0.039	3.663	Site: SLICE_X47Y71	noop_i/NOOPSoC_0/inst/noop/itlb/tlbExec/_T_8_req_addr[31]_i_6/O
net (fo=12, routed)	0.190	3.853		noop_i/NOOPSoC_0/inst/noop/itlb/tlbExec/satp_reg[52]
			Site: SLICE_X46Y71	noop_i/NOOPSoC_0/inst/noop/itlb/tlbExec/_T_464[0]_i_2/I2
LUT6 (Prop_H6LUT_SLICEL_I2_O)	(f) 0.149	4.002	Site: SLICE_X46Y71	noop_i/NOOPSoC_0/inst/noop/itlb/tlbExec/_T_464[0]_i_2/O
net (fo=76, routed)	0.194	4.196		noop_i/NOOPSoC_0/inst/noop/itlb/tlbExec/_T_464[0]_i_1_n_0
			Site: SLICE_X46Y70	noop_i/NOOPSoC_0/inst/noop/itlb/tlbExec/ram_icachePF_reg_0_3_0_0_i_4/I2

```
8028   wire  _T_431 = _T_425 & _T_430; // @[EmbeddedTLB.scala 311:87]
8029   wire  _T_433 = io_pf_status_mxr & _T_386[3]; // @[EmbeddedTLB.scala 313:68]
8030   wire  _T_434 = _T_386[1] | _T_433; // @[EmbeddedTLB.scala 313:51]
8031   wire  _T_435 = _T_431 & _T_434; // @[EmbeddedTLB.scala 313:36]
8032   wire  _T_458 = ~_T_435; // @[EmbeddedTLB.scala 327:19]
8033   wire  _T_464 = _T_458 & _T_333; // @[EmbeddedTLB.scala 327:29]
8034   wire  _T_436 = _T_431 & _T_386[2]; // @[EmbeddedTLB.scala 314:37]
```

```
326        if(tlbname == "dtlb") {
327          when((!permLoad && req.isRead()) || (!permStore && req.isWrite())) {
328            state := s_miss_slpf
329            loadPF := req.isRead() && !isAMO
330            storePF := req.isWrite() || isAMO
331          }.otherwise {
332            state := Mux(updateAD, s_write_pte, s_wait_resp)
333            missMetaRefill := true.B
334          }
335        }
```

debugging, optimization and physical design - Misconception 9

- The backtracking method can be applied to well-named signals, as well as anonymous signals (e.g., _T_464).

- It can also be applied with different propose.
  - Debugging, optimization and physical design

- In particular, if a physical design engineer wants to understand the semantics of a path, he should co-operate with an RTL engineer.

High performance - Misconception 10

- It is difficult to develop high performance processor with Chisel.

- **No**. XiangShan is a high performance processor developed with Chisel.
  - https://github.com/OpenXiangShan/XiangShan

- Compared with the language difference, the design of the micro-architecture contributes much more to the performance of a processor.

Thank you!