中国科学院计算技术研究所
INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES

中国开放指令生态（RISC-V）联盟
China RISC-V Alliance

鹏城实验室
Peng Cheng Laboratory

Practice of High-performance Chip Agile Development with Chisel

Yinan Xu 徐易难

ICT, CAS

xuyinan@ict.ac.cn

2021/6/26@CCC2021

Open-source processor

- Open-source RISC-V processor
- Chip agile development flow and tools

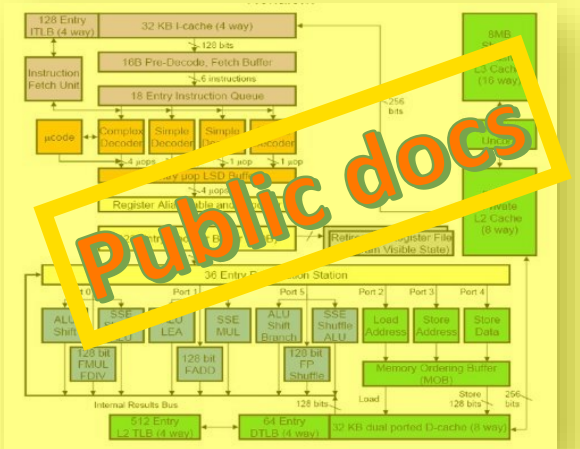
Open Framework

Open Flow

Open Tools

ISA Spec. → microarch. → Docs → coding → RTL → EDA Tools → Layout

Open, Free, Joint design



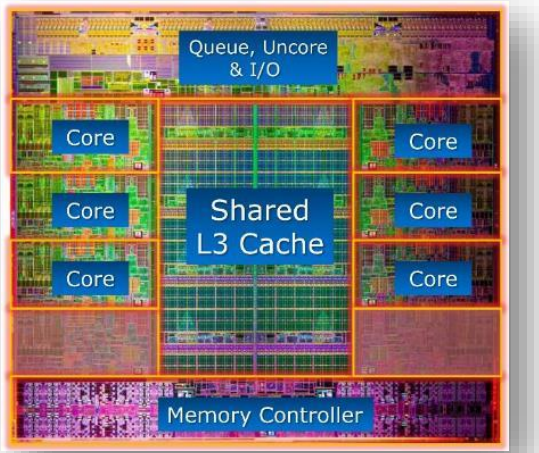
Public docs

```
component DebugCoreTop is
  port (
    -- Trigger and Data
    cu_Clk      : in    std_logic_vector(0 downto 0) := (others => '0');
    cu0_Trig    : in    t_trig_0 := (others => (others => '0'));
    cu1_Trig    : in    t_trig_1 := (others => (others => '0'));
    cu2_Trig    : in    t_trig_2 := (others => (others => '0'));
    cu0_Data    : in    t_data_0 := (others => (others => '0'));
    cu1_Data    : in    t_data_1 := (others => (others => '0'));
    cu2_Data    : in    t_data_2 := (others => (others => '0'));

    -- Downstream
    S[illegible]        : inout std_logic := '0';
    S[illegible]        : inout std_logic := '0';

    -- Upstream
    gt_Ref[illegible]_p : in    std_logic := '0';
    gt_[illegible]erClk_n : in    std_logic := '0';
    gt_RX_p     : in    std_logic_vector(2 downto 0) := (others => '0');
    gt_RX_n     : in    std_logic_vector(2 downto 0) := (others => '0');
    gt_TX_p     : out   std_logic_vector(2 downto 0);
    gt_TX_n     : out   std_logic_vector(2 downto 0)
  );
end component;
```

Open-source

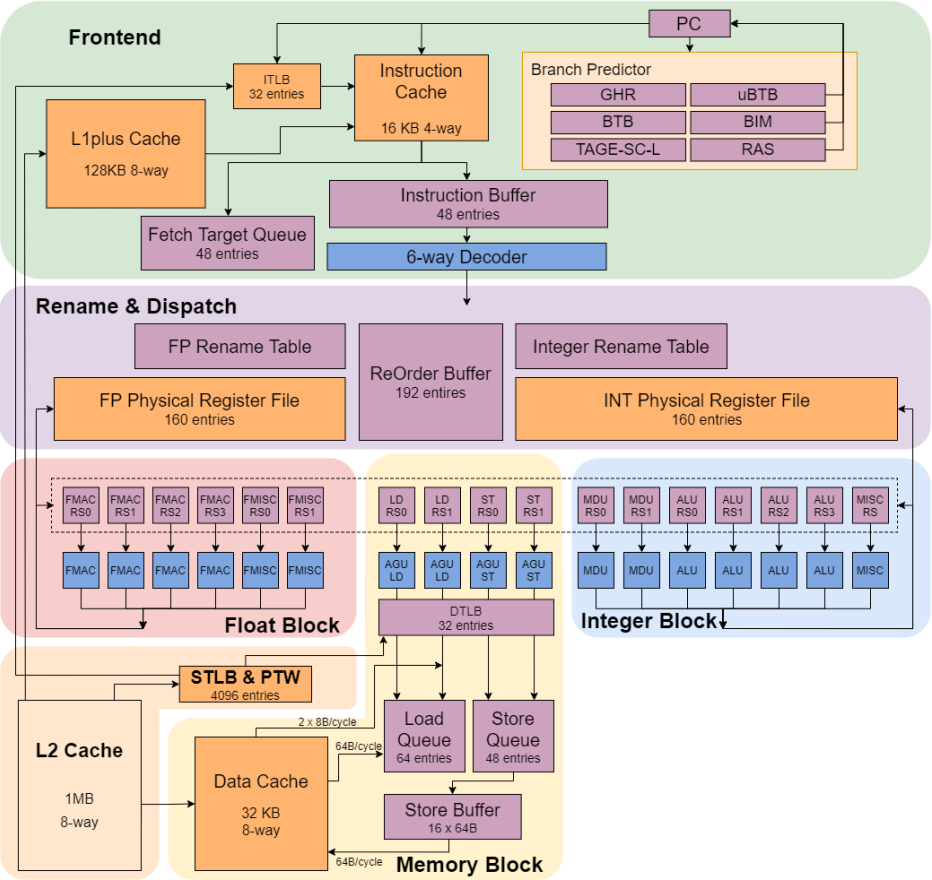


XiangShan: a high-performance processor in Chisel

- **2020-2021: one processor in one year**
  - RV64GC ISA support
  - 11-stage, superscalar, out-of-order, dual-core
  - 5.3 CoreMark/MHz (gcc-9.3.0, -O2)

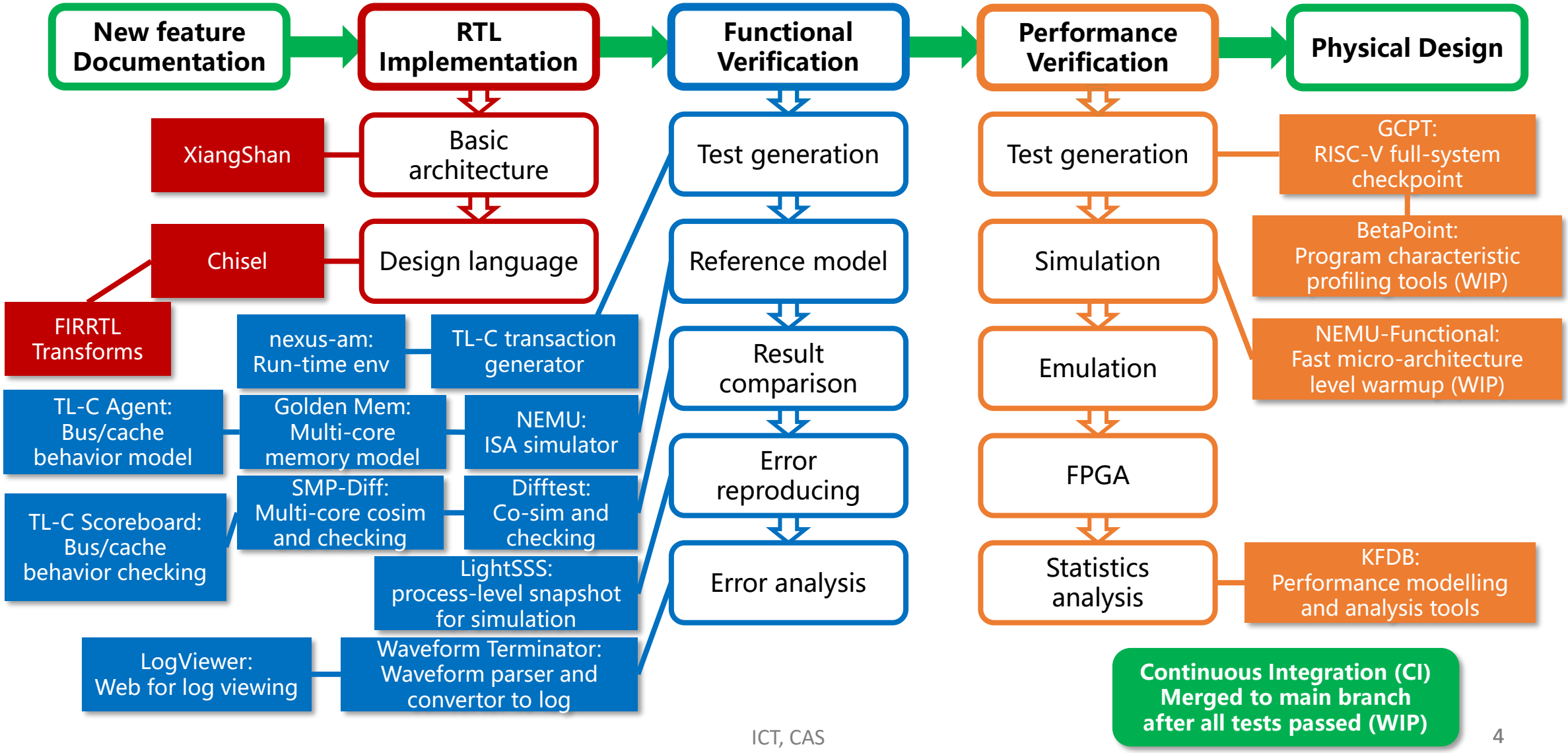
- Using a lot of **open-source tools/designs**
  - Chisel/FIRRTL
  - Verilator
  - rocket-chip (parameters, diplomacy, TL/AXI4 nodes, …)
  - DRAMsim
  - SiFive block inclusive cache
  - Berkeley-hardfloat

- Open-sourced at GitHub/Gitee/Trustie/ihub



XiangShan (yanqihu) microarchitecture

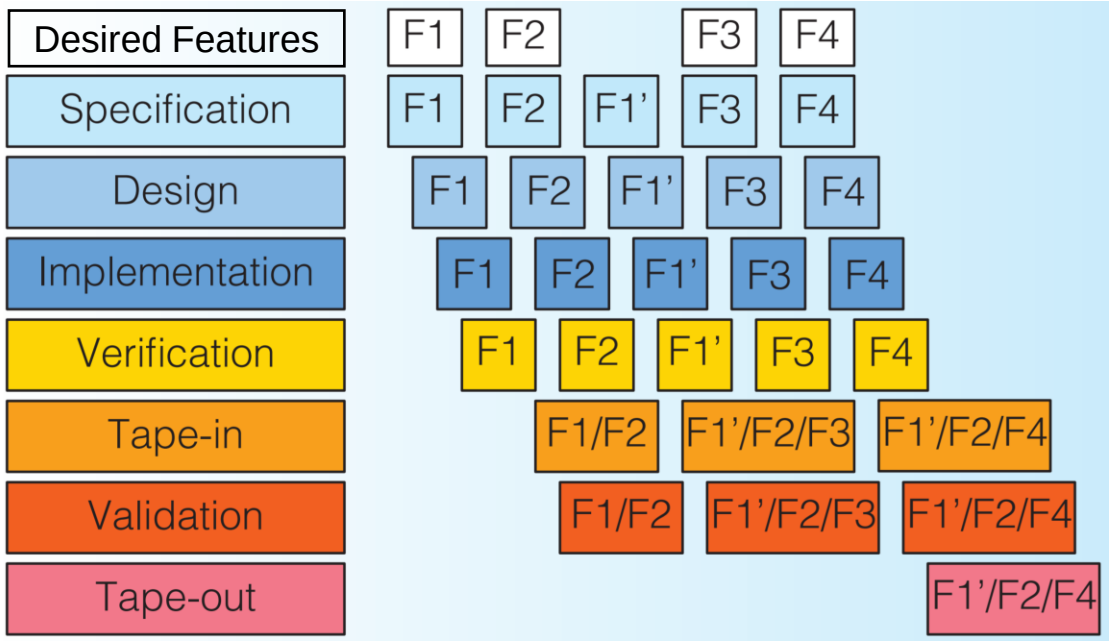
Agile development infrastructures



Agile development with Chisel

- Agile development: iterative, incremental, and evolutionary [1]

- How Chisel helps to survive in a rapidly-changing world



Agile model of hardware design [2]

[1] Agile software development. https://en.wikipedia.org/wiki/Agile_software_development.
[2] Y. Lee et al., "An Agile Approach to Building RISC-V Microprocessors," in IEEE Micro, vol. 36, no. 2, pp. 8-20, Mar.-Apr. 2016.

Bundles: freely packaging signals

- Instructions are decoded into micro-ops and go through the pipeline
- Complex processors define a complex micro-op format
  - Tens of signals with different widths
  - Structuralized control signals (generated, used, dropped at different stages)

- **Chisel YES!**

```
class CfCtrl(implicit p: Parameters) extends XSBundle {
  val cf = new CtrlFlow
  val ctrl = new CtrlSignals
}
```

Feel free to add more here

- **SystemVerilog interface YES!**

SystemVerilog interface YES?

- **Fine. Forget about the structuralized info.**

Specifying modport for interface inside an interface in SystemVerilog

Asked 5 years ago Active 5 years ago Viewed 1k times

3

I have two interfaces with modports and another interface which combines these two interfaces as shown below:

```
interface my_interface1
 // Some signals
 modport tb_to_dut (
    // Signal directions specified
 )
endinterface

interface my_interface2
 // Some signals
 modport tb_to_dut (
    // Signal directions specified
 )
endinterface

interface my_combined_interface
  my_interface1 inf1
  my_interface2 inf2
endinterface
```

I wanted to specify a modport for the combined interface which in turn calls/uses the modport of the individual interfaces. The idea is as shown below.

```
interface my_combined_interface
  my_interface1 inf1
  my_interface2 inf2

  modport tb_to_dut (
     inf1.tb_to_dut,
     inf2.tb_to_dut
  )
endinterface
```

This currently results in a syntax error. Is there a way by which I can specify the modport in the combined interface such that it percolates down to the individual interfaces?

interface system-verilog

Share Follow

asked May 25 '16 at 18:47
Pulimon
1,708 3 29 44

Related

Hot Netw

Question: I have two interfaces with modports and another interface which combines these two interfaces. I wanted to specify a modport for the combined interface which in turn calls/uses the modport of the individual interfaces.

问：我希望将两个接口组合到一起，并使用他们定义的信号组成一个新的模块接口定义

3

Unfortunately, SystemVerilog `interface`s lack compositional qualities and there is no way to specify an interface in pieces. The only way to do this today is to flatten out the combined interface by bringing the lower level interface signals up to the top level. And I would avoid using modports altogether for the testbench.

Share Follow

answered May 25 '16 at 20:45
dave_59
30.7k 3 22 48

Answer: The only way to do this today is to flatten out the interface.

答：唯一可以实现组合的方法，是**把下层信号摊平，都放到顶层**

Source: https://stackoverflow.com/questions/37445330/specifying-modport-for-interface-inside-an-interface-in-systemverilog

Parameterization

- Sometimes we need to change the parameters of the processor
  - A different replacement policy
  - A larger re-order buffer
  - A unified reservation station (from a distributed reservation station)
  - A wider rename pipeline width (multiple modules affected)
- They cause changes to different levels of the design

Series/Scope	Wire Assignments	Internal logics	Module IOs	Module-level Organizations	Example
Youth	👍				Replacement policy
Normal	👍	👍			Queue size
Ultra	👍	👍	👍		Enqueue width of a queue
[BRAND] Edition	👍	👍	👍	👍	Decode/Rename/Dispatch width

Parameterization Youth

- Between modules/methods with the same IO ports
- Example: replacement policy, error detection/correction

```scala
object ReplacementPolicy {
  //for fully associative mapping
  def fromString(s: Option[String],n_ways: Int): ReplacementPolicy = fromString(s.getOrEl
  def fromString(s: String, n_ways: Int): ReplacementPolicy = s.toLowerCase match {
    case "random" => new RandomReplacement(n_ways)
    case "lru"    => new TrueLRU(n_ways)
    case "plru"   => new PseudoLRU(n_ways)
    case t => throw new IllegalArgumentException(s"unknown Replacement Policy type $t")
  }
  //for set associative mapping
  def fromString(s: Option[String], n_ways: Int, n_sets: Int): SetAssocReplacementPolicy =
  def fromString(s: String, n_ways: Int, n_sets: Int): SetAssocReplacementPolicy = s.toLo
    case "random"    => new SetAssocRandom(n_sets, n_ways)
    case "setlru"    => new SetAssocLRU(n_sets, n_ways, "lru")
    case "setplru"   => new SetAssocLRU(n_sets, n_ways, "plru")
    case t => throw new IllegalArgumentException(s"unknown Replacement Policy type $t")
  }
}
```

(1) Implement all possibilities

```scala
icacheParameters: ICacheParameters = ICacheParameters(
  tagECC = Some("parity"),
  dataECC = Some("parity"),
  replacer = Some("setplru"),
  nMissEntries = 2
),
l1plusCacheParameters: L1plusCacheParameters = L1plusCacheParameters(
  tagECC = Some("secded"),
  dataECC = Some("secded"),
  replacer = Some("setplru"),
  nMissEntries = 8
),
dcacheParameters: DCacheParameters = DCacheParameters(
  tagECC = Some("secded"),
  dataECC = Some("secded"),
  replacer = Some("setplru"),
  nMissEntries = 16,
  nProbeEntries = 16,
  nReleaseEntries = 16,
  nStoreReplayEntries = 16
),
```

(2) Use one line of code to choose one of them

Parameterization

- Between different structure sizes (e.g., queue size)
  - usually only the pointer width is affected

```scala
class FtqPtr(implicit p: Parameters) extends CircularQueuePtr[FtqPtr](
  p => p(XSCoreParamsKey).FtqSize
){
  override def cloneType = (new FtqPtr).asInstanceOf[this.type]
}

object FtqPtr {
  def apply(f: Bool, v: UInt)(implicit p: Parameters): FtqPtr = {
    val ptr = Wire(new FtqPtr)
    ptr.flag := f
    ptr.value := v
    ptr
  }
}
```

(1) Define the parameterized queue size and queue pointer

```scala
// multi-write
val update_target = Reg(Vec(FtqSize, UInt(VAddrBits.W)))
val cfiIndex_vec = Reg(Vec(FtqSize, ValidUndirectioned(UInt(log2Up(PredictWidth).W))))
val cfiIsCall, cfiIsRet, cfiIsJalr, cfiIsRVC = Reg(Vec(FtqSize, Bool()))
val mispredict_vec = Reg(Vec(FtqSize, Vec(PredictWidth, Bool())))
```

```scala
when(real_fire) {
  val enqIdx = tailPtr.value      LinJiawei, 5 months ago • [WIP] fix ftq update logic
  commitStateQueue(enqIdx) := VecInit(io.enq.bits.valids.map(v => Mux(v, s_valid, s_invalid)))
  cfiIndex_vec(enqIdx) := io.enq.bits.cfiIndex
  cfiIsCall(enqIdx) := io.enq.bits.cfiIsCall
  cfiIsRet(enqIdx) := io.enq.bits.cfiIsRet
  cfiIsJalr(enqIdx) := io.enq.bits.cfiIsJalr
  cfiIsRVC(enqIdx) := io.enq.bits.cfiIsRVC
  mispredict_vec(enqIdx) := WireInit(VecInit(Seq.fill(PredictWidth)(false.B)))
  update_target(enqIdx) := io.enq.bits.target
}
```

(2) Use queue size to generate registers/wires and queue pointers to index into the queue

Parameterization Ultra

- We need a parameterized scheduler
  - # total entries
  - # selected entries
- Both **IO ports and internal signals** need parameterization
  - **Data width** for input, output, wire, reg, …
  - **Number** of IO ports, wires, regs, …
- How to express these in SystemVerilog and Chisel?

```
module select(
  input  [63:0] request,
  output [63:0] grant_0,
  output [63:0] grant_1,
  output [63:0] grant_2
);

// module body here

endmodule
```

Verilog example for select logic

Chisel: SystemVerilog generate Pro Max

- Parameterized scheduler: # total entries, # selected entries
- How to express these in SystemVerilog and Chisel?

```
interface select_port #(
  parameter NUM_ENTRIES = 4
);
  logic [NUM_ENTRIES - 1 : 0] grant;
  modport sel_iface(output grant);
endinterface

interface scheduler_interface #(
  parameter NUM_ENTRIES = 64.
  parameter NUM_SELECT = 3
);
  logic [NUM_ENTRIES – 1 : 0] request;

  genvar i;
  generate
    for (i = 0; i < NUM_SELECT; i++) begin
      select_port #(.NUM_ENTRIES(NUM_ENTRIES)) grants();
    end
  endgenerate
endinterface
```

SystemVerilog

```
class SelectPortIO(numEntries: Int) extends Bundle {
  val grant = Output(UInt(numEntries.W))
}

class SchedulerIO(
  numEntries: Int,
  numSelect: Int
) extends Bundle {
  val request = Input(UInt(numEntries.W))
  val grant = Vec(numSelect, new SelectPort(numEntries))
}
```

Chisel

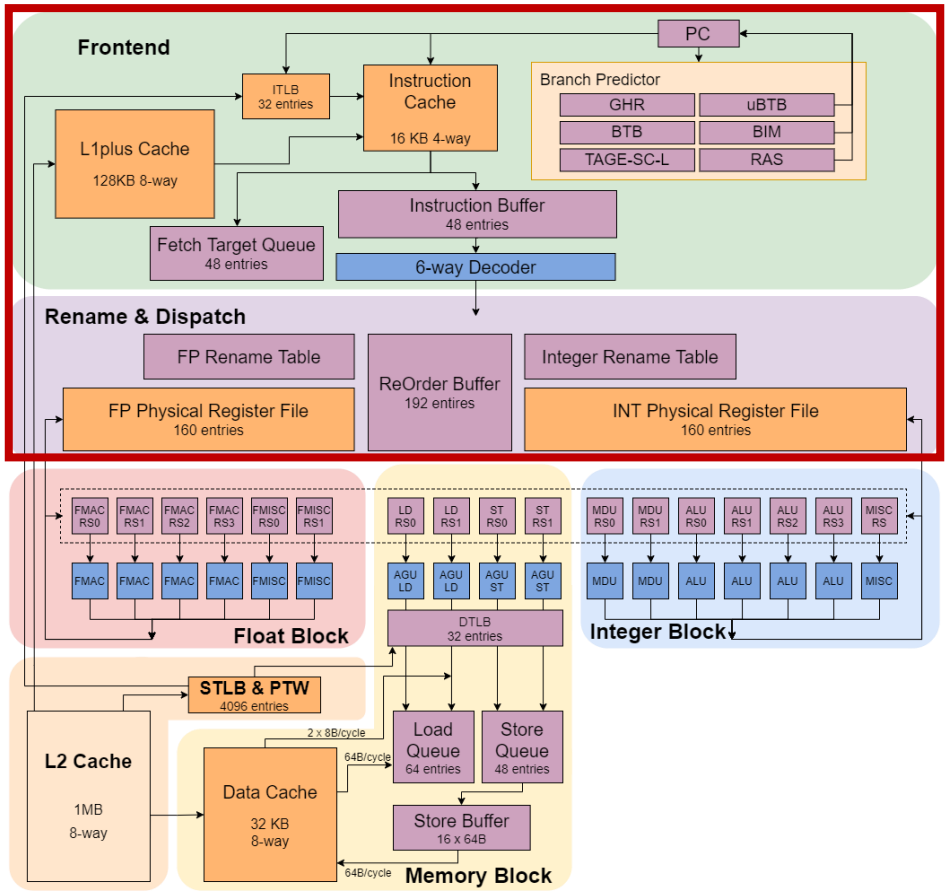
Chisel has fewer LOC and saves time.

Parameterization [some brand here] Edition

- Simply changing `FetchWidth/DecodeWidth/RenameWidth` to `4/2/2`
  - We didn't expect it to work perfectly

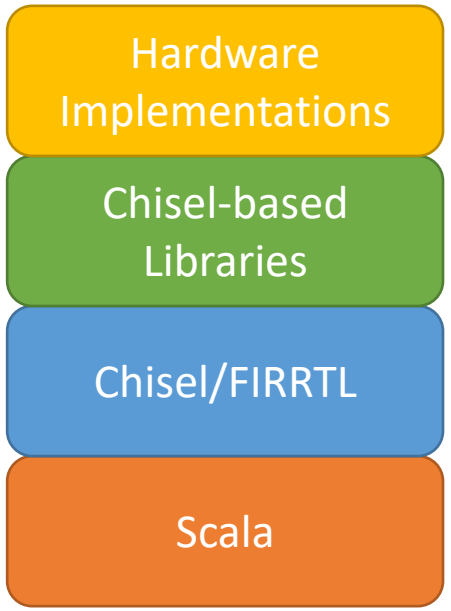
```scala
// Synthesizable minimal XiangShan
// * It is still an out-of-order, super-scalaer arch
// * L1 cache included
// * L2 cache NOT included
// * L3 cache included
class MinimalConfig(n: Int = 1) extends Config(
  new DefaultConfig(n).alter((site, here, up) => {
    case SoCParamsKey => up(SoCParamsKey).copy(
      cores = up(SoCParamsKey).cores.map(_.copy(
        DecodeWidth = 2,
        RenameWidth = 2,
        FetchWidth = 4,
        IssQueueSize = 8,
        NRPhyRegs = 80,
        LoadQueueSize = 16,
        StoreQueueSize = 16,
        RoqSize = 32,
        BrqSize = 8,
        FtqSize = 16,
        IBufSize = 16,
        StoreBufferSize = 4,
        StoreBufferThreshold = 3,
        dpParams = DispatchParameters(
          IntDqSize = 8,
          FpDqSize = 8,
          LsDqSize = 8,
```

All affected
But it works



More Chisel features for agile hardware development

- Scala infrastructures
  - Strong type system
  - Full-featured IDEs from community
- Chisel/FIRRTL infrastructures
  - SRAM type/size inference
  - ChiselTest [1]
  - FIRRTL transforms [2]
- Chisel-based Libraries
  - Configs/Parameters
  - Diplomacy framework



Levels of hardware development (with Chisel)

[1] Chuanqi Zhang. Agent Faker: Verification framework for TL-C caches. RVWC2021.
[2] Jiawei Lin. Use Firrtl Transform to Control the Effective Range of 'printf' in Large Scale Circuits. CCC2021.

Chisel for high-performance chip design

- Chisel efficiently produces structuralized and parameterized designs
  - Fast, clear and simple

- But a lot of people have told us that they
  - do NOT need agile development
  - do NOT need parameterization
  - care about ONLY **the design quality**

- **How does Chisel ensure PPA?**

High performance comes from the precise control of RTL

- Chisel is more like **syntax sugar or an advanced generator for Verilog**
  - Chisel expressions strictly map to Verilog expressions
  - Last month we fixed a bug that causes setting the PLIC claim register with a wrong value

```
4 ■■■■ src/main/scala/device/AXI4Plic.scala

@@ -145,9 +145,9 @@ class AXI4Plic
145 145        }
146 146        }
147 147
148     -      val pendingVec = Cat(pending.map(x => Cat(x.reverse)))
    148 +      val pendingVec = Cat(pending.reverse.map(x => Cat(x.reverse)))
149 149        claimCompletion.zipWithIndex.map { case (r, hart) => {
150     -        val takenVec = pendingVec & Cat(enable(hart))
    150 +        val takenVec = pendingVec & Cat(enable(hart).reverse)
151 151          r := Mux(takenVec === 0.U, 0.U, PriorityEncoder(takenVec))
152 152        }
153 153        }
```

```
1511,1512c1511,1512
<   wire [159:0] pendingVec = {_T_79,_T_78,_T_77,_T_76,_T_75}; // @[Cat.scala 30:58]
<   wire [159:0] _T_236 = {enable_0_4,enable_0_3,enable_0_2,enable_0_1,enable_0_0}; // @[Cat.scala 30:58]
---
>   wire [159:0] pendingVec = {_T_75,_T_76,_T_77,_T_78,_T_79}; // @[Cat.scala 30:58]
>   wire [159:0] _T_236 = {enable_0_0,enable_0_1,enable_0_2,enable_0_3,enable_0_4}; // @[Cat.scala 30:58]
```

Concatenation order is changed as expected

```
<   wire [63:0] _lo_T_6 = _GEN_1252 & _T_1072; // @[BitUtils.scala 17:36]
<   wire [63:0] _lo_T_7 = _T_1071 | _lo_T_6; // @[BitUtils.scala 17:25]
<   wire [63:0] _GEN_638 = _T_2 & _GEN_13[25:0] == 26'h200c ? _lo_T_7 : {{32'd0}, enable_0_3}; // @[RegMap.scala 14:48 RegMap.scala 14:52 AXI4Plic.scala 107:67]
---
>   wire [63:0] _hi_T_2 = _GEN_1252 & _T_1072; // @[BitUtils.scala 17:36]
>   wire [63:0] _hi_T_3 = _T_1071 | _hi_T_2; // @[BitUtils.scala 17:25]
>   wire [63:0] _GEN_638 = _T_2 & _GEN_13[25:0] == 26'h200c ? _hi_T_3 : {{32'd0}, enable_0_3}; // @[RegMap.scala 14:48 RegMap.scala 14:52 AXI4Plic.scala 107:67]
2277,2279c2277,2279
<   wire [63:0] _hi_T_2 = _GEN_1263 & _T_1072; // @[BitUtils.scala 17:36]
<   wire [63:0] _hi_T_3 = _T_1071 | _hi_T_2; // @[BitUtils.scala 17:25]
<   wire [63:0] _GEN_649 = _T_2 & _GEN_13[25:0] == 26'h2010 ? _hi_T_3 : {{32'd0}, enable_0_4}; // @[RegMap.scala 14:48 RegMap.scala 14:52 AXI4Plic.scala 107:67]
---
>   wire [63:0] _lo_T_6 = _GEN_1263 & _T_1072; // @[BitUtils.scala 17:36]
>   wire [63:0] _lo_T_7 = _T_1071 | _lo_T_6; // @[BitUtils.scala 17:25]
>   wire [63:0] _GEN_649 = _T_2 & _GEN_13[25:0] == 26'h2010 ? _lo_T_7 : {{32'd0}, enable_0_4}; // @[RegMap.scala 14:48 RegMap.scala 14:52 AXI4Plic.scala 107:67]
2341,2343c2341,2343
<   wire [63:0] _hi_T_6 = _GEN_1279 & _T_1072; // @[BitUtils.scala 17:36]
<   wire [63:0] _hi_T_7 = _T_1071 | _hi_T_6; // @[BitUtils.scala 17:25]
<   wire [63:0] _GEN_665 = _T_2 & _GEN_13[25:0] == 26'h2004 ? _hi_T_7 : {{32'd0}, enable_0_1}; // @[RegMap.scala 14:48 RegMap.scala 14:52 AXI4Plic.scala 107:67]
---
>   wire [63:0] _lo_T_10 = _GEN_1279 & _T_1072; // @[BitUtils.scala 17:36]
>   wire [63:0] _lo_T_11 = _T_1071 | _lo_T_10; // @[BitUtils.scala 17:25]
>   wire [63:0] _GEN_665 = _T_2 & _GEN_13[25:0] == 26'h2004 ? _lo_T_11 : {{32'd0}, enable_0_1}; // @[RegMap.scala 14:48 RegMap.scala 14:52 AXI4Plic.scala 107:67]
2597,2599c2597,2599
<   wire [63:0] _lo_T_10 = _GEN_1343 & _T_1072; // @[BitUtils.scala 17:36]
<   wire [63:0] _lo_T_11 = _T_1071 | _lo_T_10; // @[BitUtils.scala 17:25]
<   wire [63:0] _GEN_729 = _T_2 & _GEN_13[25:0] == 26'h2000 ? _lo_T_11 : {{32'd0}, enable_0_0}; // @[RegMap.scala 14:48 RegMap.scala 14:52 AXI4Plic.scala 107:67]
---
>   wire [63:0] _hi_T_6 = _GEN_1343 & _T_1072; // @[BitUtils.scala 17:36]
>   wire [63:0] _hi_T_7 = _T_1071 | _hi_T_6; // @[BitUtils.scala 17:25]
>   wire [63:0] _GEN_729 = _T_2 & _GEN_13[25:0] == 26'h2000 ? _hi_T_7 : {{32'd0}, enable_0_0}; // @[RegMap.scala 14:48 RegMap.scala 14:52 AXI4Plic.scala 107:67]
```

- It's the designer that determines how the hardware works
  - Chisel is not HLS. It does not translate/interpret what you write
  - Chisel itself never hurts the PPA as well
- **Then why Chisel? How does Chisel help ensure PPA?**

Revisiting Chisel: what is it

- Chisel is a Scala-based advanced Verilog code generator
  - Generating Verilog with Scala/Chisel libraries

- Rule-based code generation always works on Chisel
  - Programming languages are primarily intended for expressing algorithms in a form that can be executed by a computer [1]
  - Describe the rules in Scala and let Chisel generates the Verilog
  - Example: auto-generated Wallace tree [2]

- **Key: find the rules and use Chisel/Scala to describe it**

[1] Algorithm. https://en.wikipedia.org/wiki/Algorithm. Wikipedia.
[2] Jiawei Lin. Implementation of a Highly Configurable Wallace Tree Multiplier with Chisel. CCC2021.

How Chisel helps high-performance designs

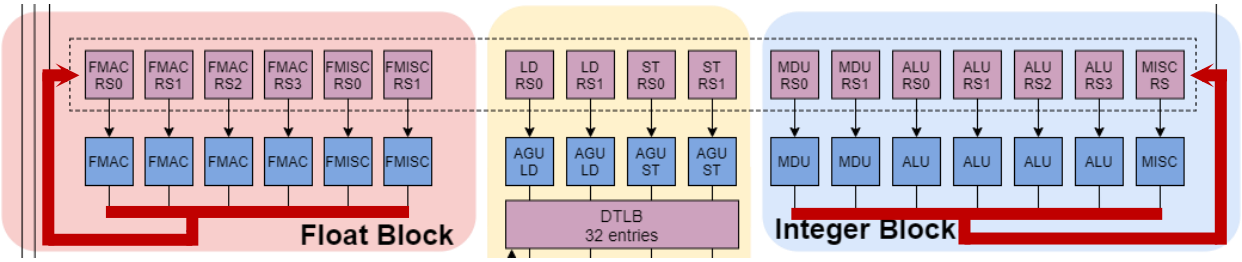
- Complexities cause difficulties in high-performance designs
  - Wider: pipeline width changes from scalar to superscalar
  - More complex: more pipelines, more structures

- Most logics in microarchitecture designs can be expressed by rules
  - Duplications: decoders, ALUs, FMAs
  - Dependences: branch predictors, selections, arbiters
  - Most logics are either parallel or serialized

- Chisel easily generates complex but deterministic logics
  - Because it is built on Scala programming language

Duplications: wakeup

- Wakeup in reservation stations
  - Not-ready instructions wait in the RS
  - Executed instructions wake them up
  - >10 wakeup sources



Wakeup sources from RS/FU

- What to do: compare the producers' pdest with consumers' psrc
  - Store Data RS: 16-entry RS, 8 integer + 8 FP wakeup ports

```
class StatusArray(config: RSConfig)(implicit p: Parameters) extends XSModule
  with HasCircularQueuePtrHelper {
  val io = IO(new Bundle {
    val redirect = Flipped(ValidIO(new Redirect))
    val flush = Input(Bool())
    // current status
    val isValid = Output(UInt(config.numEntries.W))
    val canIssue = Output(UInt(config.numEntries.W))
    // enqueue, dequeue, wakeup, flush
    val update = Vec(config.numEnq, new StatusArrayUpdateIO(config))
    val wakeup = Vec(config.numWakeup, Flipped(ValidIO(new MicroOp)))
    val wakeupMatch = Vec(config.numEntries, Vec(config.numSrc, Output(UInt(config.numWakeup.W))))
    val issueGranted = Vec(config.numDeq, Flipped(ValidIO(UInt(config.numEntries.W))))
    val deqResp = Vec(config.numDeq, Flipped(ValidIO(new Bundle {
      val rsMask = UInt(config.numEntries.W)
      val success = Bool()
    })))
    val stIssuePtr = if (config.checkWaitBit) Input(new SqPtr()) else null
  })
```

16

16

For each entry, produce a matchVec

For each source, produce a matchBit

```
def wakeupMatch(psrc: UInt, srcType: UInt) = {
  val matchVec = VecInit(io.wakeup.map(w =>
    w.valid && w.bits.pdest === psrc &&
      (SrcType.isReg(srcType) && w.bits.ctrl.rfWen &&
       psrc =/= 0.U || SrcType.isFp(srcType) && w.bits.ctrl.fpWen)
  ))
  XSError(PopCount(matchVec) > 1.U, p"matchVec ${Binary(matchVec.asUIn
  matchVec.asUInt
}
```

Serialized logics: issue selection

- Issue: select the *n*-th one from a bit vector of length *m*

- Algorithm (for software)
  - Let matrix[*m*][*n*] be whether first *m* bits has *n* ones
  - If *n* is 0, matrix[*m*][*n*] = (whether first *m* bits has zero ones) = !bits.take(*m*).orR
  - Otherwise, matrix[*m*][n] = bits[*m* - 1] && matrix[*m* - 1][*n* - 1] || !bits[*m* - 1] && matrix[*m* - 1][*n*]
  - Issue Grant Vector = bits & matrix[:][*n* - 1]

- In Chisel, it's also allowed to recursively generate Verilog based on above rules
  - But recursion never really happens in Verilog

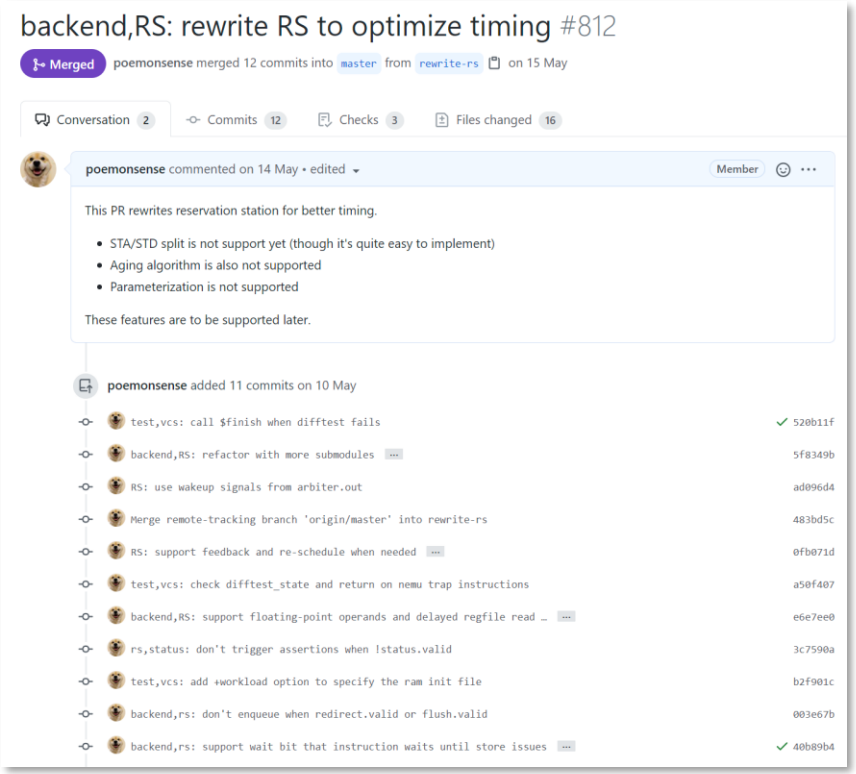
```
for (i <- 0 until config.numDeq) {
  io.grant(i).valid := OnesMoreThan(request, i + 1)
  io.grant(i).bits := request.getNthOH(i + 1).asUInt
```

Define this function and use it everywhere

Chisel makes RTL coding faster

- Case study: rewriting RS in three days
  - Modules: wakeup, select, status, payload, …
  - Without enqNum/deqNum parameterization
  - Correctly running CoreMark

- **Chisel: reducing the RTL coding time**
  - Scala for expressing the rules/algorithms
  - Chisel libraries for generating the Verilog code
  - Let computers do the boring job
  - No more genvar, generate, begin…end keywords
  - No more worries about whether it’s synthesizable
  - The fundamental advantage of Chisel against Verilog



RS[1]: a case for agile development

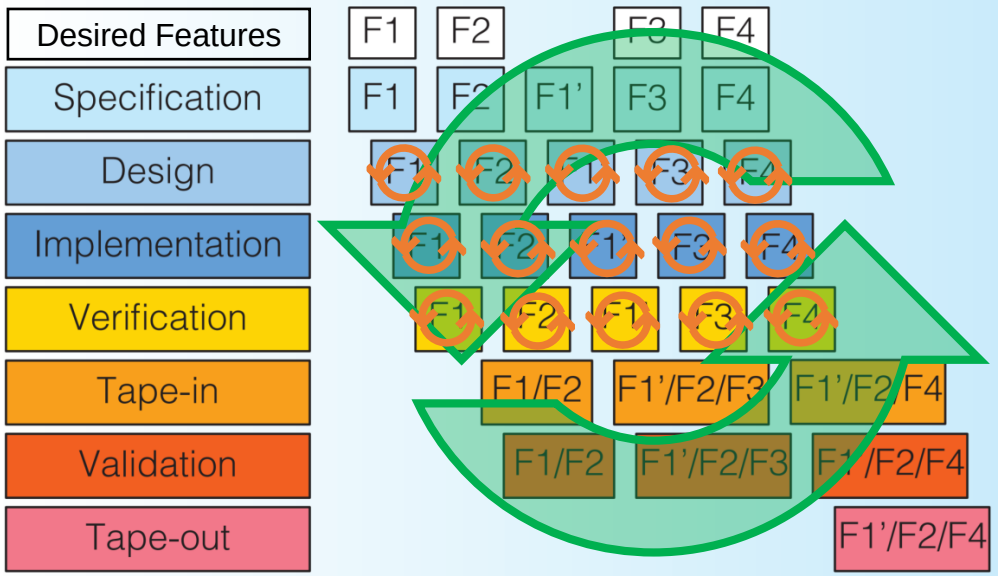
[1] https://github.com/OpenXiangShan/XiangShan/pull/812

Put it all together: agile development with Chisel

- Requirements for **better agile development flows**
  - To speed up every single step
  - To shorten the overall iterative process

- Chisel approaches to agile development
  - Abstraction (bundles), parameterization, module re-use
  - A lot of Chisel/Scala infrastructures
  - Scala-based advanced code generation
  - ……

- **Chisel: more optimizing iterations** in a fixed time interval
  - Chisel significantly reduces the RTL coding time
  - Faster than using Verilog/SV for iterative development



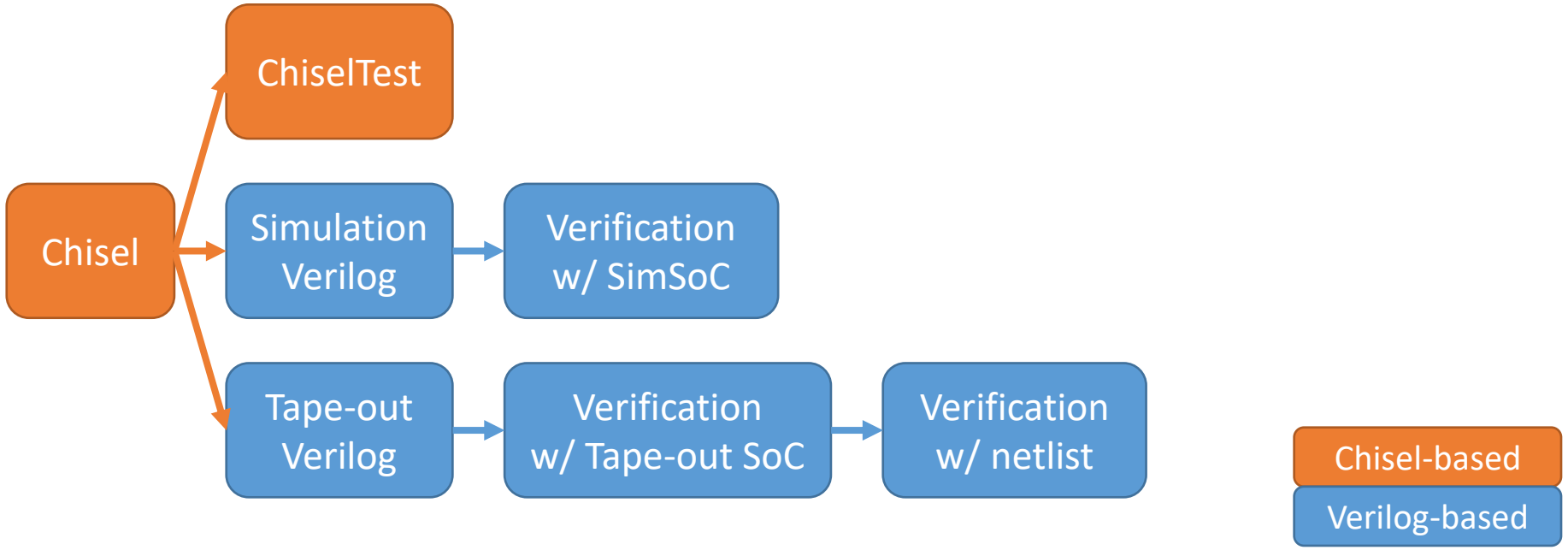
Agile model of hardware design [1]

[1] Y. Lee et al., "An Agile Approach to Building RISC-V Microprocessors," in IEEE Micro, vol. 36, no. 2, pp. 8-20, Mar.-Apr. 2016.

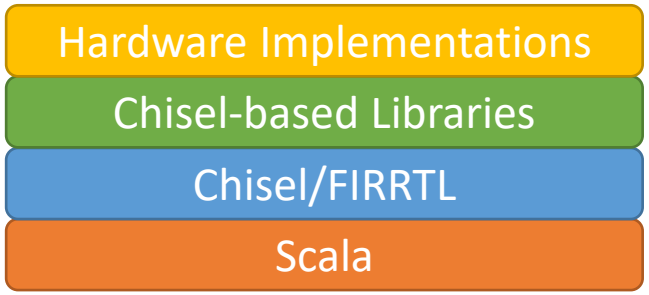
XiangShan Tape-out (July 2021)

- Single XiangShan Core with 1MB L2 Cache
  - 28nm HPCP, 1.3GHz@0.9V

- How does Chisel work in the tape-out process?



Chisel: the learning curve

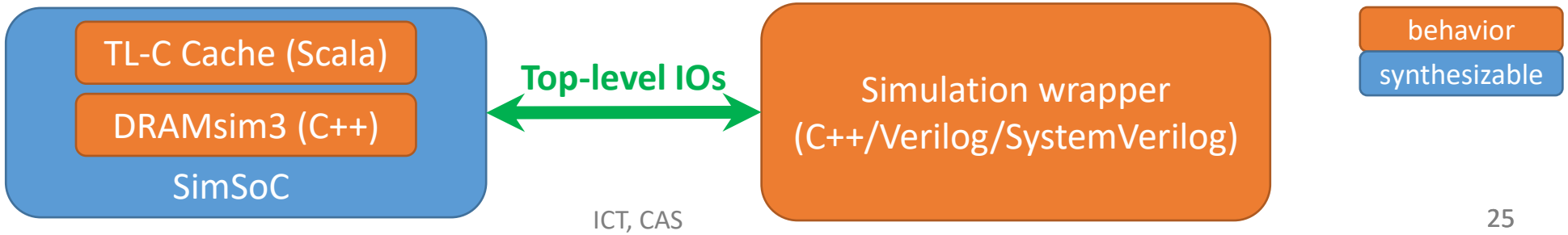


- **People are not confused about Chisel itself**
  - chisel-cheatsheet [1] (2 pages) tells everything about Chisel
  - Wire, Reg, := in Chisel are the same as wire, reg, assign in Verilog
  - No always keyword in Chisel since assignments to Reg implicitly imply an always block
  - Chisel itself does make RTL coding faster than ever before
- People are confused about advanced Scala features and Chisel-related libraries
  - How to use object, abstract class, trait, …
  - How to use Parameters, Diplomacy, …
  - They are about how we build the software-level frameworks, not hardware
  - They are not prerequisites for Chisel (and can be simply ignored if they cause difficulties)
- **Start with plain Chisel only** and learn Scala in the way to advanced agile development
  - Only seven people in XiangShan team were experienced Chisel writers before June 2020
  - But others, the Chisel beginners, have written/modified a lot of Chisel code as well by June 2021

[1] chisel-cheatsheet. https://github.com/freechipsproject/chisel-cheatsheet.

Co-sim with behavior models

- There're always some behavior models
  - Non-synthesizable C/C++/Verilog/SystemVerilog models
  - FakePTW, RAMHelper, Difftest, DRAMsim3, …, in XiangShan
  - Models used in ChiselTest cannot be directly used when simulating other modules
- Chisel does not provide an elegant way to integrate them
  - A dirty method: import and call DPI-C functions in a Blackbox Verilog module
  - Dirty method creates many nearly empty modules and does not work for Scala models
- What if Scala models can both access to Chisel and be integrated into simulation
  - Using behavior models to replace Chisel modules in simulation
  - Only in the Scala programming language does the Tester/model know Chisel information
  - Every line of code shares the structuralized and parameterized information in Chisel framework



Statistics for the generated Verilog

- We care about the size/elements of the structures in the processor
  - Towards lower power level, less area, better energy efficiency
  - Common structures: queues, SRAMs, tables, …
- The size of the structures cannot be determined in Chisel source code
  - Verilog is generated after many FIRRTL transforms
  - Dead code elimination (DCE) removes some unused wires
- What if Chisel provides post-elaboration assertions or statistics
  - Then we don't need to manually ensure there's no unexpected signal in Verilog

Queue_1.entries (Verilog)

Better if we have

- `assert(!entries.PC.exists)`
- `statistics(entries)`

along with the Verilog

Intermediate variables in Chisel-generated Verilog

- A minor change in Chisel may result in more changes in Verilog than expected
  - Chisel generates a lot of intermediate signals in Verilog
  - They are mostly labeled _T_*, _GEN_*
  - Different elaboration order generates different names

A little bit hard for physical design team to track how the RTL changes.

```
src/main/scala/device/AXI4Plic.scala
@@ -145,9 +145,9 @@ class AXI4Plic
145  145        }
146  146        }
147  147
148       -      val pendingVec = Cat(pending.map(x => Cat(x.reverse)))
     148  +      val pendingVec = Cat(pending.reverse.map(x => Cat(x.reverse)))
149  149        claimCompletion.zipWithIndex.map { case (r, hart) => {
150       -        val takenVec = pendingVec & Cat(enable(hart))
     150  +        val takenVec = pendingVec & Cat(enable(hart).reverse)
151  151          r := Mux(takenVec === 0.U, 0.U, PriorityEncoder(takenVec))
152  152        }
153  153        }
```

```
1511,1512c1511,1512
<   wire [159:0] pendingVec = {_T_79,_T_78,_T_77,_T_76,_T_75}; // @[Cat.scala 30:58]
<   wire [159:0] _T_236 = {enable_0_4,enable_0_3,enable_0_2,enable_0_1,enable_0_0}; // @[Cat.scala 30:58]
---
>   wire [159:0] pendingVec = {_T_75,_T_76,_T_77,_T_78,_T_79}; // @[Cat.scala 30:58]
>   wire [159:0] _T_236 = {enable_0_0,enable_0_1,enable_0_2,enable_0_3,enable_0_4}; // @[Cat.scala 30:58]
```

Concatenation order is changed as expected

```
<   wire [63:0] _lo_T_6 = _GEN_1252 & _T_1072; // @[BitUtils.scala 17:36]
<   wire [63:0] _lo_T_7 = _T_1071 | _lo_T_6; // @[BitUtils.scala 17:25]
<   wire [63:0] _GEN_638 = _T_2 & _GEN_13[25:0] == 26'h200c ? _lo_T_7 : {{32'd0}, enable_0_3}; // @[RegMap.scala 14:48 RegMap.scala 14:52 AXI4Plic.scala 107:67]
---
>   wire [63:0] _hi_T_2 = _GEN_1252 & _T_1072; // @[BitUtils.scala 17:36]
>   wire [63:0] _hi_T_3 = _T_1071 | _hi_T_2; // @[BitUtils.scala 17:25]
>   wire [63:0] _GEN_638 = _T_2 & _GEN_13[25:0] == 26'h200c ? _hi_T_3 : {{32'd0}, enable_0_3}; // @[RegMap.scala 14:48 RegMap.scala 14:52 AXI4Plic.scala 107:67]
2277,2279c2277,2279
<   wire [63:0] _hi_T_2 = _GEN_1263 & _T_1072; // @[BitUtils.scala 17:36]
<   wire [63:0] _hi_T_3 = _T_1071 | _hi_T_2; // @[BitUtils.scala 17:25]
<   wire [63:0] _GEN_649 = _T_2 & _GEN_13[25:0] == 26'h2010 ? _hi_T_3 : {{32'd0}, enable_0_4}; // @[RegMap.scala 14:48 RegMap.scala 14:52 AXI4Plic.scala 107:67]
---
>   wire [63:0] _lo_T_6 = _GEN_1263 & _T_1072; // @[BitUtils.scala 17:36]
>   wire [63:0] _lo_T_7 = _T_1071 | _lo_T_6; // @[BitUtils.scala 17:25]
>   wire [63:0] _GEN_649 = _T_2 & _GEN_13[25:0] == 26'h2010 ? _lo_T_7 : {{32'd0}, enable
2341,2343c2341,2343
<   wire [63:0] _hi_T_6 = _GEN_1279 & _T_1072; // @[BitUtils.scala 17:36]
<   wire [63:0] _hi_T_7 = _T_1071 | _hi_T_6; // @[BitUtils.scala 17:25]
<   wire [63:0] _GEN_665 = _T_2 & _GEN_13[25:0] == 26'h2004 ? _hi_T_7 : {{32'd0}, enable
---
>   wire [63:0] _lo_T_10 = _GEN_1279 & _T_1072; // @[BitUtils.scala 17:36]
>   wire [63:0] _lo_T_11 = _T_1071 | _lo_T_10; // @[BitUtils.scala 17:25]
>   wire [63:0] _GEN_665 = _T_2 & _GEN_13[25:0] == 26'h2004 ? _lo_T_11 : {{32'd0}, enable_0_1}; // @[RegMap.scala 14:48 RegMap.scala 14:52 AXI4Plic.scala 107:67]
2597,2599c2597,2599
<   wire [63:0] _lo_T_10 = _GEN_1343 & _T_1072; // @[BitUtils.scala 17:36]
<   wire [63:0] _lo_T_11 = _T_1071 | _lo_T_10; // @[BitUtils.scala 17:25]
<   wire [63:0] _GEN_729 = _T_2 & _GEN_13[25:0] == 26'h2000 ? _lo_T_11 : {{32'd0}, enable_0_0}; // @[RegMap.scala 14:48 RegMap.scala 14:52 AXI4Plic.scala 107:67]
---
>   wire [63:0] _hi_T_6 = _GEN_1343 & _T_1072; // @[BitUtils.scala 17:36]
>   wire [63:0] _hi_T_7 = _T_1071 | _hi_T_6; // @[BitUtils.scala 17:25]
>   wire [63:0] _GEN_729 = _T_2 & _GEN_13[25:0] == 26'h2000 ? _hi_T_7 : {{32'd0}, enable_0_0}; // @[RegMap.scala 14:48 RegMap.scala 14:52 AXI4Plic.scala 107:67]
```

Only name changes.
No functional change.
These changes can be ignored.

Previous	New
_lo_T_6	_hi_T_2
_lo_T_7	_hi_T_3
_hi_T_2	_lo_T_6
_hi_T_3	_lo_T_7

Take-away messages

- **Chisel can produce high-performance designs**
  - Chisel is an advanced HDL (not HLS)
  - Chisel allows efficient structurization and parameterization
  - Chisel itself never hurts PPA

- **Chisel speeds up agile development**
  - Chisel provides an efficient RTL coding method
  - More iterations in a fixed time interval
  - More optimizations and possibly better PPA in a fixed time interval

- **Chisel is imperfect but we can continuously improve it**
  - To do better on chip agile design, implementation and verification processes

Thanks!

Welcome more people to join us on the 2nd generation of XiangShan!

北京微核芯科技有限公司
BEIJING VCORE TECHNOLOGY CO., LTD.

ByteDance 字节跳动

ESWIN

优矽科技