Summary of Problems and Experiences during the Processor Development based on Chisel

Yue Jin

Institute of Computing Technology, Chinese Academy of Sciences

2021.6@Shanghai, China

Outline

- A Short Introduction to Chisel
  - Features and Benefits
  - Where to Learn Chisel
- Sharing of Experience Using Chisel
  - Problems We Met
  - Experiences Sharing
  - Syntactic Salt
  - …
- A RTL designer's perspective for Chisel

1. A Short Introduction to Chisel

IS

- Object-Oriented Programming
- Various and Convenient Hardware Generator
- Overall Signal Connection

Features

✓Improved readability and much shorter lines of code

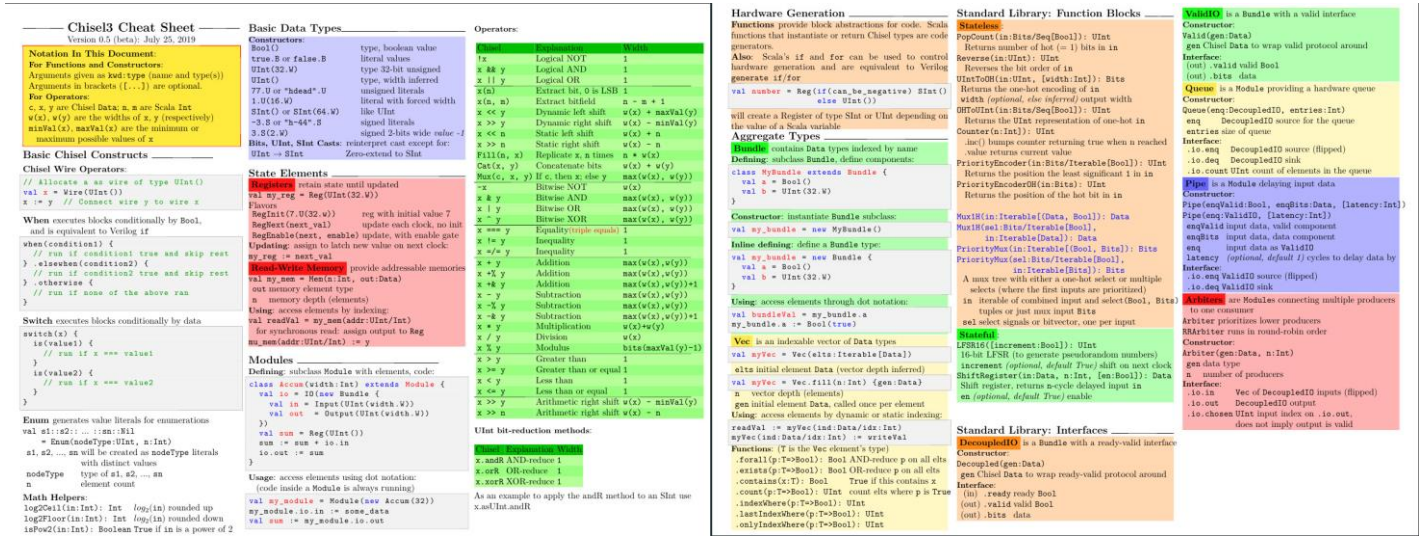
✓Flexible hardware module writing

✓Make designers focus on design, not some dirty work

Benefits

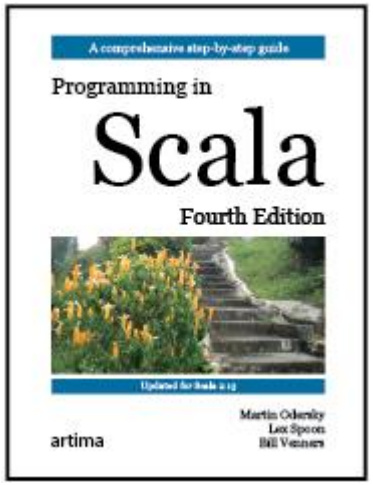
Where to Learn?

Advice: 2 pages of chisel-cheatsheet and an 896 pages scala book(just kidding…).



chisel-cheatsheet (colored version)
https://github.com/freechipsproject/chisel-cheatsheet/pull/2

+



Programming in Scala(available online)

- Other excellent learning resources
  - chisel-cookbook https://www.chisel-lang.org/chisel3/docs/introduction.html
  - chisel-bootcamp https://github.com/freechipsproject/chisel-bootcamp
  - chisel-book(pdf) https://github.com/schoeberl/chisel-book

2. Problems & Experience Using Chisel

Software thinking

vs

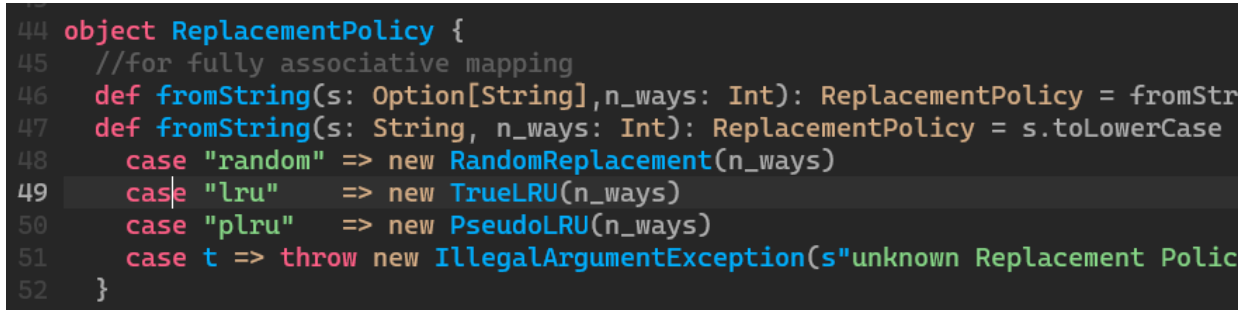
Hardware thinking

Advice you may need

`def` is not for object declaration

```
class RandomReplacement(n_ways: Int) extends
 ReplacementPolicy {
  //...
  def nBits = 16
  def perSet = false
  //..
```

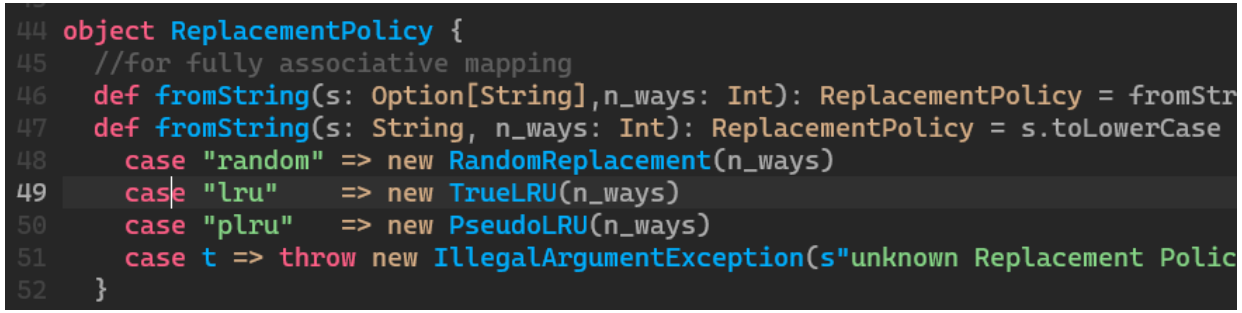
```
  def ptwl2replacer = ReplacementPolicy.
fromString(Some("plru"))
  /*
     ...
   */
  ptwl2replacer.access(hitWay)
  val victimway = ptwl2replacer.way()
```



`def` is not for object declaration

```
class RandomReplacement(n_ways: Int) extends
 ReplacementPolicy {
  //...
  def nBits = 16
  def perSet = false
  //..
```

```
  def ptwl2replacer = ReplacementPolicy.
fromString(Some("plru"))
  /*
     ...
   */
  ptwl2replacer.access(hitWay)
  val victimway = ptwl2replacer.way()
```

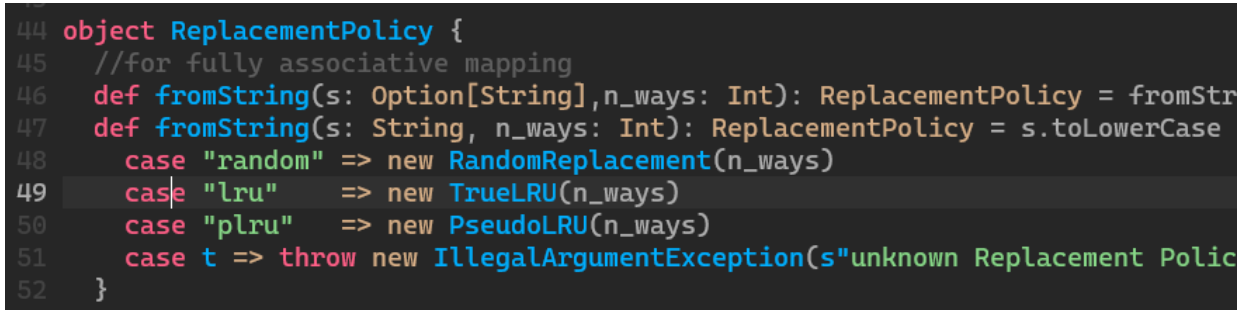


Victim way will always be the first(way 0)

`def` is not for object declaration

```
class RandomReplacement(n_ways: Int) extends
 ReplacementPolicy {
  //...
  def nBits = 16
  def perSet = false
  //..
```

```
  def ptwl2replacer = ReplacementPolicy.
fromString(Some("plru"))
  /*
     ...
   */
  ptwl2replacer.access(hitWay)
  val victimway = ptwl2replacer.way()
```



```
object ReplacementPolicy {
  //for fully associative mapping
  def fromString(s: Option[String],n_ways: Int): ReplacementPolicy = fromStr
  def fromString(s: String, n_ways: Int): ReplacementPolicy = s.toLowerCase
    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 Polic
  }
```

Victim way will always be the first(way 0)

Advice: Use `def` to declare methods, and `val` to declare objects

Cat-map problem

```
// Is there an MSHR free for this request?
val mshr_validOH = Cat(mshrs.map(_.io.status.valid).reverse)
val mshr_free = ((~mshr_validOH).asUInt & prioFilter).orR()
```

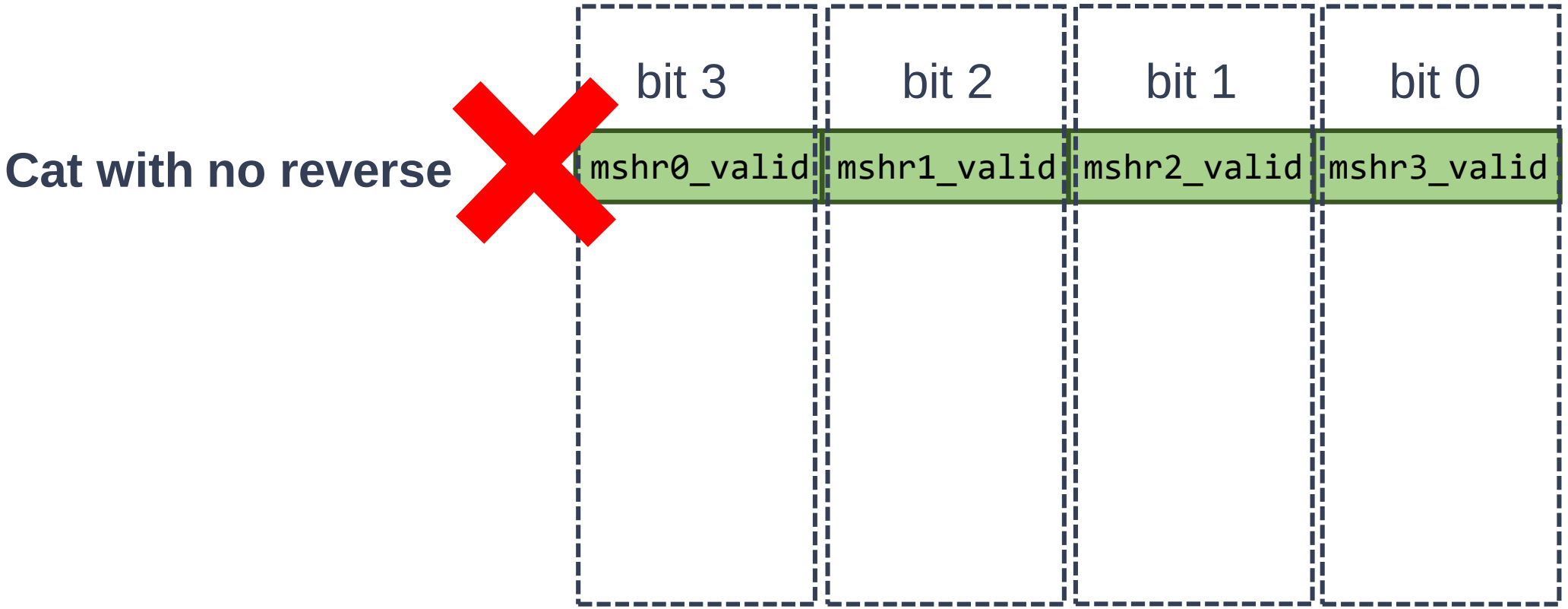
Cat-map problem

```
// Is there an MSHR free for this request?
val mshr_validOH = Cat(mshrs.map(_.io.status.valid))
val mshr_free = ((~mshr_validOH).asUInt & prioFilter).orR()
```

Cat with no reverse

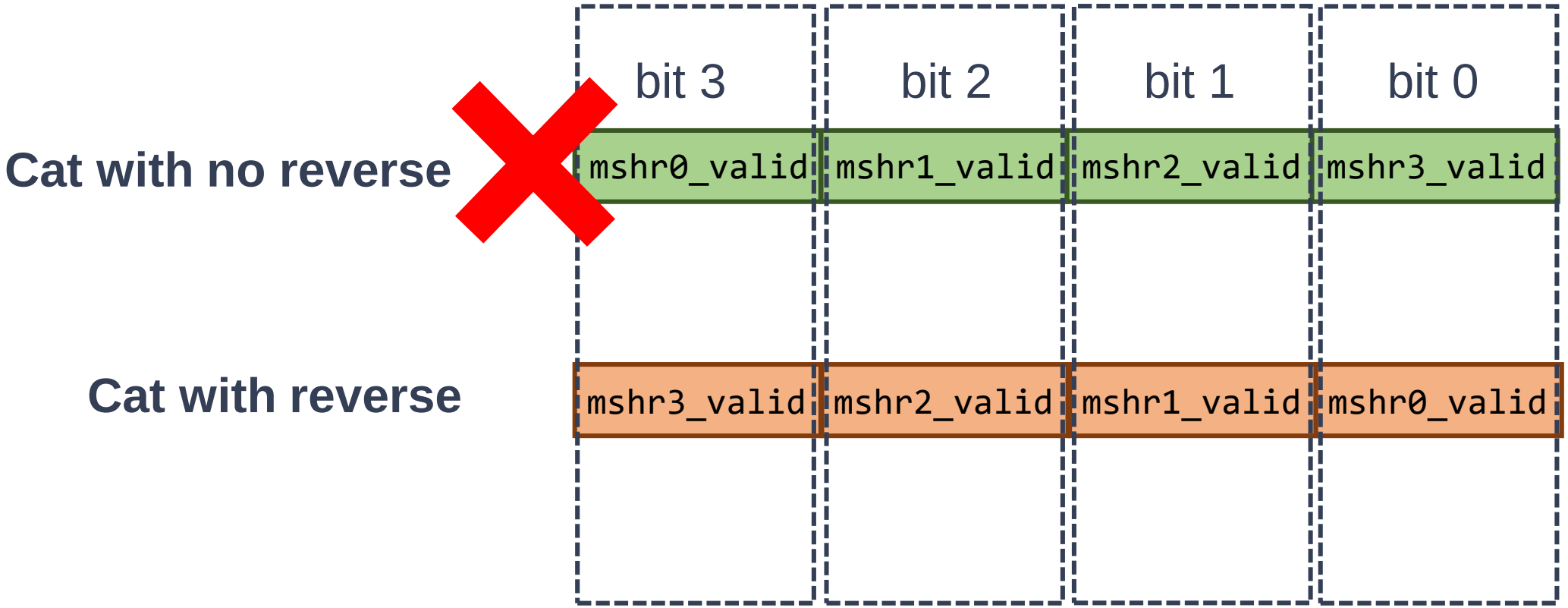
Cat-map problem

```
// Is there an MSHR free for this request?
val mshr_validOH = Cat(mshrs.map(_.io.status.valid).reverse)
val mshr_free = ((~mshr_validOH).asUInt & prioFilter).orR()
```



Cat-map problem

```
// Is there an MSHR free for this request?
val mshr_validOH = Cat(mshrs.map(_.io.status.valid).reverse)
val mshr_free = ((~mshr_validOH).asUInt & prioFilter).orR()
```

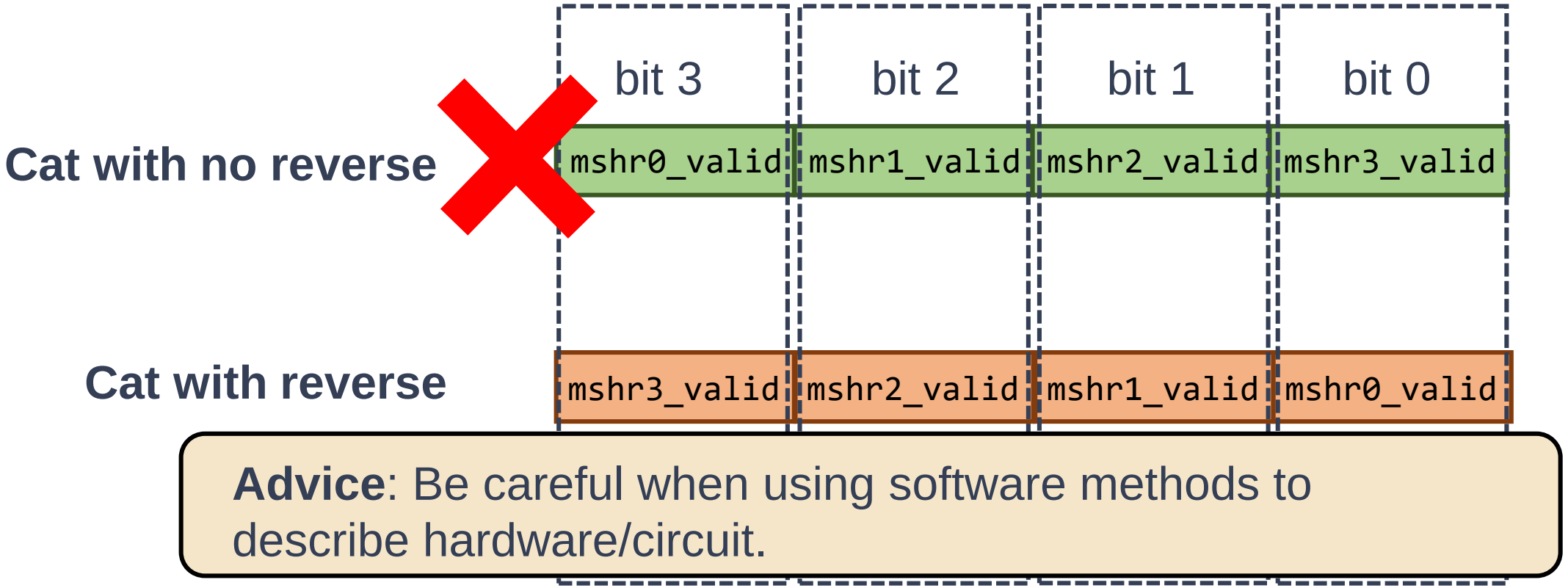


Cat-map problem

```
// Is there an MSHR free for this request?
val mshr_validOH = Cat(mshrs.map(_.io.status.valid).reverse)
val mshr_free = ((~mshr_validOH).asUInt & prioFilter).orR()
```

Cat → Hardware thinking

map → Software thinking



Advice: Be careful when using software methods to describe hardware/circuit.

Width! Width! Width!

```
val validMeta = Cat((0 until nWays).map{w => v
alidArray(Cat(s2_idx, w.U))}.reverse).asUInt
```

Width! Width! Width!

```
val validMeta = Cat((0 until nWays).map{w => v
alidArray(Cat(s2_idx, w.U))}.reverse).asUInt
```

An I$ with very high miss rate

Width! Width! Width!

```
val validMeta = Cat((0 until nWays).map{w => v
alidArray(Cat(s2_idx, w.U))}.reverse).asUInt
```

An I$ with very high miss rate

```
wire [6:0] _T_1 = {s2_idx,1'h0}; // @[Cat.
wire [255:0] _T_2 = validArray >> _T_1; //
wire  lo_lo = _T_2[0]; // @[ICache.scala 3
wire [6:0] _T_3 = {s2_idx,1'h1}; // @[Cat.
wire [255:0] _T_4 = validArray >> _T_3; //
wire  lo_hi = _T_4[0]; // @[ICache.scala 3
wire [7:0] _T_5 = {s2_idx,2'h2}; // @[Cat.
wire [255:0] _T_6 = validArray >> _T_5; //
wire  hi_lo = _T_6[0]; // @[ICache.scala 3
wire [7:0] _T_7 = {s2_idx,2'h3}; // @[Cat.
wire [255:0] _T_8 = validArray >> _T_7; //
wire  hi_hi = _T_8[0]; // @[ICache.scala 3
```

Width! Width! Width!

```
val validMeta = Cat((0 until nWays).map{w => v
alidArray(Cat(s2_idx, w.U))}.reverse).asUInt
```

An I$ with very high miss rate

```
wire [6:0] _T_1 = {s2_idx,1'h0}; // @[Cat.
wire [255:0] _T_2 = validArray >> _T_1; //
wire  lo_lo = _T_2[0]; // @[ICache.scala 3
wire [6:0] _T_3 = {s2_idx,1'h1}; // @[Cat.
wire [255:0] _T_4 = validArray >> _T_3; //
wire  lo_hi = _T_4[0]; // @[ICache.scala 3
wire [7:0] _T_5 = {s2_idx,2'h2}; // @[Cat.
wire [255:0] _T_6 = validArray >> _T_5; //
wire  hi_lo = _T_6[0]; // @[ICache.scala 3
wire [7:0] _T_7 = {s2_idx,2'h3}; // @[Cat.
wire [255:0] _T_8 = validArray >> _T_7; //
wire  hi_hi = _T_8[0]; // @[ICache.scala 3
```

```
val validMeta = Cat((0 until nWays).map{w => val
idArray(Cat(s2_idx, w.U(log2Ceil(nWays).W)))}
.reverse).asUInt
```

Width! Width! Width!

```
val validMeta = Cat((0 until nWays).map{w => v
alidArray(Cat(s2_idx, w.U))}.reverse).asUInt
```

An I$ with very high miss rate

```
wire [6:0] _T_1 = {s2_idx,1'h0}; // @[Cat.
wire [255:0] _T_2 = validArray >> _T_1; //
wire  lo_lo = _T_2[0]; // @[ICache.scala 3
wire [6:0] _T_3 = {s2_idx,1'h1}; // @[Cat.
wire [255:0] _T_4 = validArray >> _T_3; //
wire  lo_hi = _T_4[0]; // @[ICache.scala 3
wire [7:0] _T_5 = {s2_idx,2'h2}; // @[Cat.
wire [255:0] _T_6 = validArray >> _T_5; //
wire  hi_lo = _T_6[0]; // @[ICache.scala 3
wire [7:0] _T_7 = {s2_idx,2'h3}; // @[Cat.
wire [255:0] _T_8 = validArray >> _T_7; //
wire  hi_hi = _T_8[0]; // @[ICache.scala 3
```

```
val validMeta = Cat((0 until nWays).map{w => val
idArray(Cat(s2_idx, w.U(log2Ceil(nWays).W)))}
.reverse).asUInt
```

Advice: In `Cat` operation, you must specify the bit width for the immediate value of `UInt` type

Width! Width! Width!

Perhaps we need some syntactic salt?

```
[warn] value 'w.U' in Cat has
no width information!
```

```
val validMeta = Cat((0 until nWays).map{w => v
alidArray(Cat(s2_idx, w.U))}.reverse).asUInt
```

An I$ with very high miss rate

```
wire [6:0] _T_1 = {s2_idx,1'h0}; // @[Cat.
wire [255:0] _T_2 = validArray >> _T_1; //
wire  lo_lo = _T_2[0]; // @[ICache.scala 3
wire [6:0] _T_3 = {s2_idx,1'h1}; // @[Cat.
wire [255:0] _T_4 = validArray >> _T_3; //
wire  lo_hi = _T_4[0]; // @[ICache.scala 3
wire [7:0] _T_5 = {s2_idx,2'h2}; // @[Cat.
wire [255:0] _T_6 = validArray >> _T_5; //
wire  hi_lo = _T_6[0]; // @[ICache.scala 3
wire [7:0] _T_7 = {s2_idx,2'h3}; // @[Cat.
wire [255:0] _T_8 = validArray >> _T_7; //
wire  hi_hi = _T_8[0]; // @[ICache.scala 3
```

```
val validMeta = Cat((0 until nWays).map{w => val
idArray(Cat(s2_idx, w.U(log2Ceil(nWays).W)))}
.reverse).asUInt
```

Advice: In `Cat` operation, it is recommended to specify the bit width for the immediate value of `UInt` type

Care about your `DontCare`

```
val bundleA <> DontCare
// lots of code
bundleA.signal1 := true.B
bundleA.signal2 := 1.U
bundleA.signal4 := 2.U
//...
bundleA.signalx := 3.U
// bundleA.signal3 is set to false.B

//forget...
when(bundleA.signal1 && bundleA.signal3){
  //...
}
```

Care about your `DontCare`

```
val bundleA <> DontCare
bundleA.signal1 := true.B
bundleA.signal2 := 1.U
bundleA.signal4 := 2.U
//...
bundleA.signalx := 3.U
// bundleA.signal3 is set to false.B

//forget...
when(bundleA.signal1 && bundleA.signal3){
  //...
}
```

Advice: Do not use `DontCare` for make a temporary assignment to a bundle.

Care about your `DontCare`

```
val bundleA <> DontCare
bundleA.signal1 := true.B
bundleA.signal2 := 1.U
bundleA.signal4 := 2.U
//...
bundleA.signalx := 3.U
// bundleA.signal3 is set to false.B

//forget...
when(bundleA.signal1 && bundleA.signal3){
  //...
}
```

Perhaps we need some syntactic salt?

```
[warn]signal3 in bundleA has been used but was connected to DontCare
```

Advice: Do not use `DontCare` for make a temporary assignment to a bundle.

Competitive assignment

Defination: Reassignment of the same variable (a wire/register) in the same cycle

Competitive assignment

Defination: Reassignment of the same variable (a wire/register) in the same cycle

Explicit competition assignment

```
when (cond) {
    reg := val_a
    reg := val_b
}
```

Competitive assignment

Defination: Reassignment of the same variable (a wire/register) in the same cycle

Explicit competition assignment

```
when (cond) {
    reg := val_a
    reg := val_b
}
```

Less obvious competitive assignments

```
when (cond) {
    reg.valid := true.B
    reg       := val_b
}
```

Competitive assignment

Defination: Reassignment of the same variable (a wire/register) in the same cycle

Explicit competition assignment

```
when (cond) {
    reg := val_a
    reg := val_b
}
```

Less obvious competitive assignments

```
when (cond) {
    reg.valid := true.B
    reg       := val_b
}
```

Implicit competition assignment

```
for (x <- xs) {
    val idx = f(x)
    somevec(idx) := g(x)
}
```

Competitive assignment

Defination: Reassignment of the same variable (a wire/register) in the same cycle

Explicit competition assignment

```
when (cond) {
    reg := val_a
    reg := val_b
}
```

Less obvious competitive assignments

```
when (cond) {
    reg.valid := true.B
    reg       := val_b
}
```

Implicit competition assignment

```
for (x <- xs) {
    val idx = f(x)
    somevec(idx) := g(x)
}
```

In an implicit competition assignment, f(x) is not a one-to-one function, that is, f(x1) = f(x2) but x1 != x2.

Competitive assignment

```
//exu write back, update some info
for((wb,i) <- io.exuWriteback.zipWithIndex) {
    val wbIdx = wb.bits.redirect.ftqIdx.value
    // ...
    when(cfiUPdate.taken && offset < cfiIndex_vec(wbIdx).bits){
        cfiIndex_vec(wbIdx).valid := true.B
        cfiIndex_vec(wbIdx).bits  := offset
        cfiIsCall(wbIdx) := wb.bits.uop.cf.pd.cfiIsCall
        cfiIsRet(wbIdx) := wb.bits.uop.cf.pd.isRet
        //...
    }
}
```

Competitive assignment

```
//exu write back, update some info
for((wb,i) <- io.exuWriteback.zipWithIndex) {
    val wbIdx = wb.bits.redirect.ftqIdx.value
    // ...
    when(cfiUPdate.taken && offset < cfiIndex_vec(wbIdx).bits){
        cfiIndex_vec(wbIdx).valid := true.B
        cfiIndex_vec(wbIdx).bits  := offset
        cfiIsCall(wbIdx) := wb.bits.uop.cf.pd.cfiIsCall
        cfiIsRet(wbIdx) := wb.bits.uop.cf.pd.isRet
        //...
    }
}
```

The wbIdx of different writeback ports may be the same !

Competitive assignment

```
//exu write back, update some info
for((wb,i) <- io.exuWriteback.zipWithIndex) {
    val wbIdx = wb.bits.redirect.ftqIdx.value
    // ...
    when(cfiUPdate.taken && offset < cfiIndex_vec(wbIdx).bits){
        cfiIndex_vec(wbIdx).valid := true.B
        cfiIndex_vec(wbIdx).bits  := offset
        cfiIsCall(wbIdx) := wb.bits.uop.cf.pd.cfiIsCall
        cfiIsRet(wbIdx) := wb.bits.uop.cf.pd.isRet
        //...
    }
}
```

The wbIdx of different writeback ports may be the same !

```
Wb_1 : cfiIndex_vec(0).valid := true.B
Wb_2 : cfiIndex_vec(3).valid := false.B
Wb_3 : cfiIndex_vec(0).valid := false.B
```

Competitive assignment

```
//exu write back, update some info
for((wb,i) <- io.exuWriteback.zipWithIndex) {
    val wbIdx = wb.bits.redirect.ftqIdx.value
    // ...
    when(cfiUPdate.taken && offset < cfiIndex_vec(wbIdx).bits){
        cfiIndex_vec(wbIdx).valid := true.B
        cfiIndex_vec(wbIdx).bits  := offset
        cfiIsCall(wbIdx) := wb.bits.uop.cf.pd.cfiIsCall
        cfiIsRet(wbIdx) := wb.bits.uop.cf.pd.isRet
        //...
    }
}
```

The wbIdx of different writeback ports may be the same !

```
Wb_1 : cfiIndex_vec(0).valid := true.B
Wb_2 : cfiIndex_vec(3).valid := false.B
Wb_3 : cfiIndex_vec(0).valid := false.B
```

Advice: Ensure that your assignment in the loop will not cause implicit competitive assignment.

3. A developer’s perspective for Chisel

- Excellent HDL
- A lot of syntactic sugar
- Free developers from dirty work
- Make more developers willing to write hardware

3. A developer’s perspective for Chisel

- Excellent HDL
- A lot of syntactic sugar
- Free developers from dirty work
- Make more developers willing to write hardware

3. A developer's perspective for Chisel

- Excellent HDL
- A lot of syntactic sugar
- Free developers from dirty work
- Make more developers willing to write hardware

- Some software thinking may cause problems
- Be familiar to Scala before using Chisel
- Ensure that the hardware (verilog) is consistent with what you wrote with Chisel

Summary

Advice: Use `def` to declare methods, and `val` to declare constants or objects

Advice: Be careful when using software methods to describe hardware/circuit.

Advice: In `Cat` operation, it is recommended to specify the bit width for the immediate value of `UInt` type

Advice: Do not use `DontCare` for make a temporary assignment to a bundle.

Advice: Ensure that your assignment in the loop will not cause implicit competitive assignment.

THANKS!

Q & A