Chisel @ CS250 – Ready/Valid Lecture 11b

Jonathan Bachrach

EECS UC Berkeley

October 3, 2013
Many large digital designs are divided into local synchronous pipelines, or units, connected via decoupling FIFOs.

- Approx. 10K-100K gates per unit

Decoupled units may have different clocks.

- In which case, need asynchronous FIFOs.
Ready / Valid

![Diagram of Ready/Valid](image)
Ready / Valid + Queue

Producer

Queue

Consumer

bits T
valid Bool
ready Bool

bits T
valid Bool
ready Bool

bits T
valid Bool
ready Bool

bits T
valid Bool
ready Bool
Ready / Valid Transfer

Clock

Valid

Ready

transfer
Two Ready / Valid Transfers

Clock

Valid

Ready

transfer transfer
No Ready / Valid Transfer Both Low

Clock

Valid

Ready
No Ready / Valid Transfer Valid Low

Clock

Valid

Ready
No Ready / Valid Transfer Ready Low

Clock

Valid

Ready
No Ready / Valid Transfer Out of Phase

Clock

Valid

Ready
No Ready / Valid Combinational Loops

Combinational Logic

Producer

bits
T
valid
Bool
ready
Bool

Consumer

bits
T
valid
Bool
ready
Bool

Combinational Logic
producer is valid regardless of whether consumer is ready
consumer is ready regardless of whether producer is valid

but how do you know when to move on?
cmd.ready = cmd.x & !xcmd.ready
xcmd.valid = cmd.valid & cmd.x
xcmd.bits = cmd.bits
cmd.ready =
  (!cmd.x | xcmd.ready) &
  (!cmd.y | ycmd.ready)
xcmd.valid = cmd.valid & cmd.x &
  (!cmd.y | ycmd.ready)
ycmd.valid = cmd.valid & cmd.y &
  (!cmd.x | xcmd.ready)
xcmd.bits = ycmd.bits = cmd.bits
x_blocked = cmd.x & !xcmd.ready
y_blocked = cmd.y & !ycmd.ready
cmd.ready = !x_blocked & !y_blocked
xcmd.valid = cmd.valid & cmd.x & !y_blocked
ycmd.valid = cmd.valid & cmd.y & !x_blocked
xcmd.bits = ycmd.bits = cmd.bits
cmd.ready =
(!cmd.x | xcmd.ready) &
(!cmd.y | ycmd.ready) &
(!cmd.op | (op.valid &
  (!cmd.x | xop.ready) &
  (!cmd.y | yop.ready)))
xcmd.valid =
cmd.valid & cmd.x &
(!cmd.op | (op.valid & xop.ready)) &
(!cmd.y | (ycmd.ready &
  (!cmd.op | (op.valid & yop.ready))))
ycmd.valid = ...

Harder Ready / Valid Example
cmd.ready =
(!cmd.x | xcmd.ready) &
(!cmd.y | ycmd.ready) &
(!cmd.op1 | (op1.valid &
  (!cmd.x | xop1.ready) &
  (!cmd.y | yop1.ready))) &
(!cmd.op2 | (op2.valid &
  (!cmd.x | xop2.ready) &
  (!cmd.y | yop2.ready)))

xcmd.valid =
  cmd.valid & cmd.x &
  (!cmd.op1 | (op1.valid & xop1.ready)) &
  (!cmd.op2 | (op2.valid & xop2.ready)) &
  (!cmd.y | (ycmd.ready &
    (!cmd.op1 | (op1.valid & yop1.ready)) &
    (!cmd.op2 | (op2.valid & yop2.ready)))))

ycmd.valid = ...
transaction() {
    c = cmd.deq
    if (c.x)
        xcmd.enq(c)
    if (c.y)
        ycmd.enq(c)
    if (c.op1) {
        o1 = op1.deq
        if (c.x)
            xop1.enq(o1)
        if (c.y)
            yop1.enq(o1)
    }
    if (c.op2) {
        o2 = op2.deq
        if (c.x)
            xop2.enq(o2)
        if (c.y)
            yop2.enq(o2)
    }
}
- no ready in terms of its own valid
- no valid in terms of its own ready
- ready/valid must make sure that all data that should move can move
Using Decoupled in Chisel (Old)

**producer**

```scala
def Producer(): Unit = {
  val results = Decoupled(UInt(width = 64))
  val result = Reg(UInt(width = 64))

  results.valid := Bool(false)
  results.bits := UInt(0)

  ... when (isResult && results.ready) {
    // enq
    results.valid := Bool(true)
    results.bits := result
  }
}
```

**consumer**

```scala
def Consumer(): Unit = {
  val cmds = Decoupled(UInt(width = 32)).flip
  val cmd = Reg(UInt(width = 32))

  cmds.ready := Bool(false)
  ...

  ... when (cmds.valid) {
    // deq
    cmds.ready := Bool(true)
    cmd := result
  }
}
```

- Intuitive
- In general will create combinational loop
- If though queue between producer and consumer ok
Using Decoupled in Chisel (New)

**producer**

```scala
val results = Decoupled(UInt(width = 64))
val result = Reg(UInt(width = 64))
results.valid := isResult
results.bits := result
isResult := Reg(results.ready) & haveNewResult
```

**consumer**

```scala
val cmds = Decoupled(UInt(width = 32)).flip
val cmd = Reg(UInt(width = 32))
cmds.ready := amReady
cmd := result
amReady := Reg(cmds.valid) & isConsumedCmd
```

- Ready and valid never function of each other
- Hard to design
- Never will create combinational loop
“Ready / Valid” based on Chris Fletcher’s CS150 Writeup which is based on Greg Gibeling’s Writeup

Chris Patten’s mother of all decoupled examples told to me by Yunsup Lee