优矽科技
UC TECH IP

Heterodox Realization of Asynchronous Reset Register

Zhanhao Liang, UCTECHIP

优矽科技
UC TECH IP

Outline

- Three realizations of async reset regs
- The reason for choosing the third
- The goal of realization
- Environment of realization
- Process
- Result
- Limitation

Three realizations of async reset regs

- Write them in Verilog and wrap in Chisel BlackBox – common and temporary solution.
- Create async reset registers APIs in Chisel or FIRRTL – real sense of realization, orthodox.
- Just change sync reset to async reset while FIRRTL emitting Verilog – text modification, heterodox.

优矽科技
UC TECH IP

The reason for choosing the third

- Wrapping in Chisel BlackBox is a little bit verbose, and more verbose under parameters' control.
- For the second, has no ability to make it.
- The third way is easy to achieve and easy to use – still using original reg APIs.

优矽科技
UC TECH IP

The goal of realization

- Generate async reset regs in Verilog using original reg APIs.
- Less modification to ensure not changing other Verilog code.
- Easy to apply to Chisel Projects

优矽科技
UC TECH IP

Environment of realization

- Ubuntu 16.04 or 18.04
- Resource
  - firrtl – Repo from Github: https://github.com/freechipsproject/firrtl
  - ChiselProjects – made for test

优矽科技
UC TECH IP

Environment of realization – modify resource

- firrtl Repo commit: 5e23294 2017.09.29

Environment of realization – test resource

- sbt 0.13.16
- Build definition -- build.sbt, comes from chisel-tutorial Repo commit: 1f8d68e
- directory structure from chisel-tutorial

```
ChiselProjects
|--build.sbt
|--Makefile
|--project
|  |--build.properties
|--src
    |--main
    |   |--scala
    |       |--AsyncResetReg
    |           |--AsyncResetReg.scala
    |--test
       |--scala
          |--AsyncResetReg
             |--AsyncResetRegMain.scala
```

Process

- modify file: firrtl/src/main/scala/firrtl/Emitter.scala
- compile and package firrtl code, get a file – firrtl.jar
- make directory – ChiselProjects/lib, place firrtl.jar in it
- make Chisel test code
- generate Verilog code

Process

```
            expr match {
              case m: Mux if canFlatten(m) =>
                val ifStatement = Seq(tabs, "if (", m.cond, ") begin")
                val trueCase = addUpdate(m.tval, tabs + tab)
                val elseStatement = Seq(tabs, "end else begin")
                val ifNotStatement = Seq(tabs, "if (!(", m.cond, ")) begin")
                val falseCase = addUpdate(m.fval, tabs + tab)
                val endStatement = Seq(tabs, "end")

                ((trueCase.nonEmpty, falseCase.nonEmpty): @ unchecked) match {
                  case (true, true) =>
                    ifStatement +: trueCase ++: elseStatement +: falseCase :+ endStatement
                  case (true, false) =>
                    ifStatement +: trueCase :+ endStatement
                  case (false, true) =>
                    ifNotStatement +: falseCase :+ endStatement
                }
              case _ => Seq(Seq(tabs, r, " <= ", e, ";"))
            }
*/
/*****************************for async reset*****************************/
            expr match {
              case m: Mux if canFlatten(m) =>
                val ifStatement = Seq(tabs, "if (", m.cond, ") begin")
                val trueCase = addUpdate(m.tval, tabs + tab)
                val elseStatement = Seq(tabs, "end else begin")
                val ifNotStatement = Seq(tabs, "if (!(", m.cond, ")) begin")
                val falseCase = addUpdate(m.fval, tabs + tab)
                val endStatement = Seq(tabs, "end")

                if (tabs == "    ") {
                  val alwaysStatement = Seq(tab, "always @(posedge ", clk, " or posedge ", m.cond, ") begin")
                  val alwaysEndStatement = Seq(tab, "end")

                  ((trueCase.nonEmpty, falseCase.nonEmpty): @ unchecked) match {
                    case (true, true) =>
                      alwaysStatement +: ifStatement +: trueCase ++: elseStatement +: falseCase :+ endStatement :+
alwaysEndStatement
                    case (true, false) =>
                      alwaysStatement +: ifStatement +: trueCase :+ endStatement :+ alwaysEndStatement
                    case (false, true) =>
                      alwaysStatement +: ifNotStatement +: falseCase :+ endStatement :+ alwaysEndStatement
                  }
                } else {
```

Process

```
/*
        at_clock.getOrElseUpdate(clk, ArrayBuffer[Seq[Any]]()) ++= {
          val tv = init
          val fv = netlist(r)
          if (weq(tv, r))
            addUpdate(fv, "")
          else
            addUpdate(Mux(reset, tv, fv, mux_type_and_widths(tv, fv)), "")
        }
*/
/*****************************for async reset*******************************/
        at_clock.getOrElseUpdate(clk, ArrayBuffer[Seq[Any]]()) ++= {
          val tv = init
          val fv = netlist(r)
          if (weq(tv, r))
            addUpdate(fv, "    ")
          else
            addUpdate(Mux(reset, tv, fv, mux_type_and_widths(tv, fv)), "    ")
        }
      }
/*****************************for async reset*******************************/
```

优矽科技
UC TECH IP

Process

```
/*
      def update(e: Expression, value: Expression, clk: Expression, en: Expression) {
         if (!at_clock.contains(clk)) at_clock(clk) = ArrayBuffer[Seq[Any]]()
         if (weq(en,one))
            at_clock(clk) += Seq(e," <= ",value,";")
         else {
            at_clock(clk) += Seq("if(",en,") begin")
            at_clock(clk) += Seq(tab,e," <= ",value,";")
            at_clock(clk) += Seq("end")
         }
      }
*/
/*****************************for async reset*****************************/
      def update(e: Expression, value: Expression, clk: Expression, en: Expression) {
         if (!at_clock.contains(clk)) at_clock(clk) = ArrayBuffer[Seq[Any]]()
         if (weq(en,one)) {
            at_clock(clk) += Seq("")
            at_clock(clk) += Seq("/**********Memory**********/")
            at_clock(clk) += Seq("")
            at_clock(clk) += Seq(tab, "always @(posedge ", clk, ") begin")
            at_clock(clk) += Seq(tab, tab, e," <= ",value,";")
            at_clock(clk) += Seq(tab, "end")
            at_clock(clk) += Seq("")
            at_clock(clk) += Seq("/**********Memory**********/")
            at_clock(clk) += Seq("")
         } else {
            at_clock(clk) += Seq("")
            at_clock(clk) += Seq("/**********Memory**********/")
            at_clock(clk) += Seq("")
            at_clock(clk) += Seq(tab, "always @(posedge ", clk, ") begin")
            at_clock(clk) += Seq(tab, tab, "if(",en,") begin")
            at_clock(clk) += Seq(tab, tab, tab, e," <= ",value,";")
            at_clock(clk) += Seq(tab, tab, "end")
            at_clock(clk) += Seq(tab, "end")
            at_clock(clk) += Seq("")
            at_clock(clk) += Seq("/**********Memory**********/")
            at_clock(clk) += Seq("")
         }
      }
/*****************************for async reset*****************************/
```

优矽科技
UC TECH IP

Process

```
/*
      def simulate(clk: Expression, en: Expression, s: Seq[Any], cond: Option[String]) {
        if (!at_clock.contains(clk)) at_clock(clk) = ArrayBuffer[Seq[Any]]()
        at_clock(clk) += Seq("`ifndef SYNTHESIS")
        if (cond.nonEmpty) {
          at_clock(clk) += Seq(s"`ifdef ${cond.get}")
          at_clock(clk) += Seq(tab, s"if (`${cond.get}) begin")
          at_clock(clk) += Seq("`endif")
        }
        at_clock(clk) += Seq(tab,tab,"if (",en,") begin")
        at_clock(clk) += Seq(tab,tab,tab,s)
        at_clock(clk) += Seq(tab,tab,"end")
        if (cond.nonEmpty) {
          at_clock(clk) += Seq(s"`ifdef ${cond.get}")
          at_clock(clk) += Seq(tab,"end")
          at_clock(clk) += Seq("`endif")
        }
        at_clock(clk) += Seq("`endif // SYNTHESIS")
      }
*/
/*****************************for async reset*******************************/
      def simulate(clk: Expression, en: Expression, s: Seq[Any], cond: Option[String]) {
        if (!at_clock.contains(clk)) at_clock(clk) = ArrayBuffer[Seq[Any]]()
        at_clock(clk) += Seq(tab, "always @(posedge ", clk, ") begin")
        at_clock(clk) += Seq(tab, tab, "`ifndef SYNTHESIS")
        if (cond.nonEmpty) {
          at_clock(clk) += Seq(tab, tab, s"`ifdef ${cond.get}")
          at_clock(clk) += Seq(tab, tab, tab, s"if (`${cond.get}) begin")
          at_clock(clk) += Seq(tab, tab, "`endif")
        }
        at_clock(clk) += Seq(tab, tab, tab, tab, "if (", en ,") begin")
        at_clock(clk) += Seq(tab, tab, tab, tab, tab, s)
        at_clock(clk) += Seq(tab, tab, tab, tab, "end")
        if (cond.nonEmpty) {
          at_clock(clk) += Seq(tab, tab, s"`ifdef ${cond.get}")
          at_clock(clk) += Seq(tab, tab, tab, "end")
          at_clock(clk) += Seq(tab, tab, "`endif")
        }
        at_clock(clk) += Seq(tab, tab, "`endif // SYNTHESIS")
        at_clock(clk) += Seq(tab, "end")
      }
/*****************************for async reset*******************************/
```

Process

```
        for (clk_stream <- at_clock if clk_stream._2.nonEmpty) {
//          emit(Seq(tab, "always @(posedge ", clk_stream._1, ") begin"))
//          for (x <- clk_stream._2) emit(Seq(tab, tab, x))
//          emit(Seq(tab, "end"))
/*****************************for async reset*******************************/
          emit(Seq(""))
          for (x <- clk_stream._2) emit(Seq(x))
          emit(Seq(""))
/*****************************for async reset*******************************/
        }
        emit(Seq("endmodule"))
      }
```

Result

before

- one module one “always @(posedge clock) begin”.

after

- each reset register, register set and SYNTHESIS macro has a “always @” sentence.
- “always @(posedge clock or posedge reset) begin” for reset registers
- register set and SYNTHESIS macros keep“always @(posedge clock) begin”.

Result

```
  always @(posedge clock or posedge reset) begin
    if (reset) begin
      clk0BBunReg_h_1_y <= 2'h0;
    end else begin
      if (io_en) begin
        clk0BBunReg_h_1_y <= io_bBunIn_h_1_y;
      end
    end
  end
  always @(posedge clock or posedge reset) begin
    if (reset) begin
      clk0BBunReg_h_1_z <= 3'sh0;
    end else begin
      if (io_en) begin
        clk0BBunReg_h_1_z <= io_bBunIn_h_1_z;
      end
    end
  end

/**********Memory**********/

  always @(posedge clock) begin
    if(clk0Mem__T_377_en & clk0Mem__T_377_mask) begin
      clk0Mem[clk0Mem__T_377_addr] <= clk0Mem__T_377_data;
    end
  end

/**********Memory**********/



  always @(posedge io_clk1 or posedge io_rst1) begin
    if (io_rst1) begin
      clk1BoolReg <= 1'h0;
    end else begin
      if (io_en) begin
```

Limitation

- Can not generate any sync reset regs.
- Can not use no reset regs any more otherwise there will be no “always @” sentences or wrong “always @” sentences, such as “always @(posedge clock or posedge enable) begin”.

优矽科技
UC TECH IP

Thanks for Listening!