Memory Black Box Tips

yportne13

Dec 10, 2023

https://github.com/yportne13/SpinalMemBist

Tools

typst: https://github.com/typst/typst

polylux: https://github.com/andreasKroepelin/polylux

StyleTTS2: https://github.com/yl4579/StyleTTS2

yi-34b-chat: https://huggingface.co/01-ai/Yi-34B-Chat

Memory

what we get:

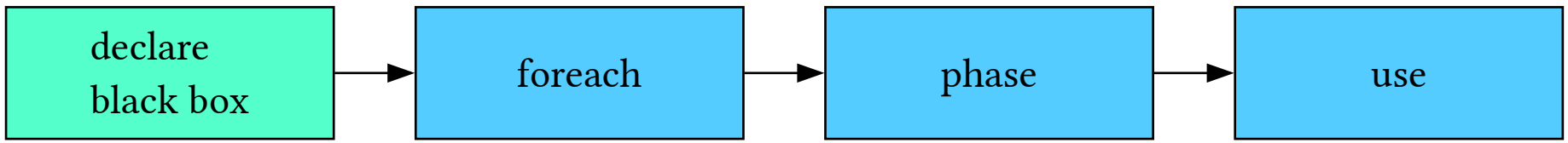
```
reg [7:0] mems_0 [0:11];
always @(posedge clk) begin
  if(io_we) begin
    mems_0[io_waddr] <= io_win;
  end
end
always @(posedge clk) begin
  if(io_ren) begin
    io_rout <= mems_0[io_raddr];
  end
end
```

Memory

what we need:

```
Ram_1w_1rs #(
  .wordCount(12),
  .wordWidth(8),
  ...
) mems_0 (
  .wr_clk  (clk               ), //i
  .wr_en   (io_we             ), //i
  .wr_addr (io_waddr[3:0]     ), //i
  .wr_data (io_win[7:0]       ), //i
  .rd_clk  (clk               ), //i
  .rd_en   (io_ren            ), //i
  .rd_addr (io_raddr[3:0]     ), //i
  .rd_data (io_rout[7:0]      )  //o
);
```

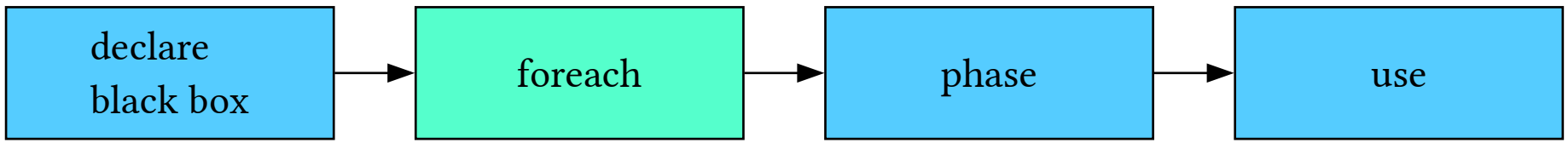
SpinalHDL Origin Example



Phase.scala

```scala
class Ram_1w_1rs(
  val wordWidth       : Int,
  ...
) extends BlackBox {
  val io = new Bundle {
    ...
  }
  ...
}
```

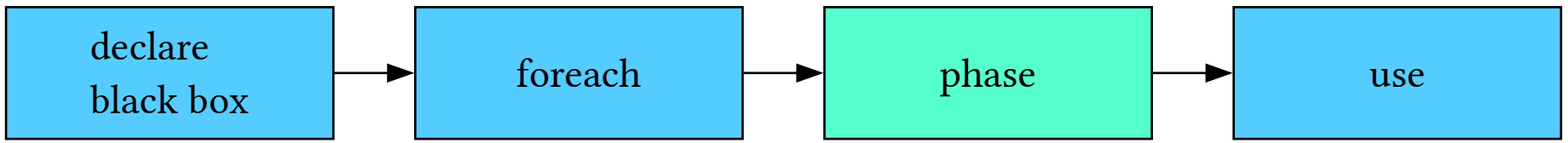
SpinalHDL Origin Example



Phase.scala

```
walkBaseNodes{
  case mem: Mem[_] => mems += mem
  case ec: ExpressionContainer =>
    ec.foreachExpression{
      case port: MemPortStatement => ...
      case _ =>
    }
  case _ =>
}
```

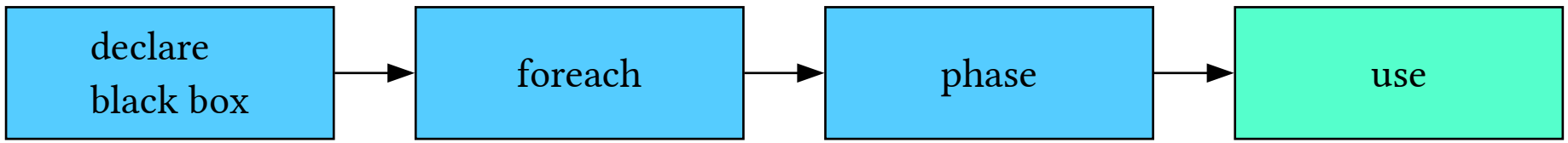
SpinalHDL Origin Example



Phase.scala

```
876 class PhaseMemBlackBoxingDefault(policy: MemBlackboxingPolicy) extends
    PhaseMemBlackBoxingWithPolicy(policy){
877   ...
878   mem.component.rework {
879     val ram = new Ram_1w_1rs(...)
880     ...
881     removeMem()
882   }
883 }
```

SpinalHDL origin example



use memory as normal

```
val mem = Mem(Bits(8 bits), 64)
mem.write(io.waddr, io.win, io.we)
mem.readSync(io.raddr)
```

```
SpinalConfig(
  memBlackBoxers = ArrayBuffer(new PhaseMemBlackBoxingDefault())
).generateVerilog(new toplevel)
```

Implementations based on size

if size is small, use reg

Memlib.scala

```
161 def removeMem(): Unit ={
162   super.removeMem(mem)
163 }
164
165 // use blackbox mem or not
166 val useBlack = mem.getWidth * mem.wordCount > 10*8
167 if(useBlack)
168 if (mem.initialContent != null) {
169   return "Can't blackbox ROM"  //TODO
```

```
1 reg [7:0] mems_0 [0:11];
```

Implementations based on size

if size is big, divide into multi mems

```
67 class Multi_Ram_Wrapper(...) extends Component {
68   ...
69   this.parent.rework {
70     io.wr.clk := wrClock.readClockWire
71     io.rd.clk := rdClock.readClockWire
72   }
73
74   val sepBy = 32
75   val memNum = 1 << log2Up(wordCount/sepBy)
76   require(wordCount/memNum*memNum == wordCount)
77   val mems = (0 until memNum).map(idx => {
78     val mem = new Ram_1w_1rs(...)
79     ...
80   })
81 }
```

Connect to Top

Memlib.scala line 59

```
58 class Ram_1w_1rs_bist() extends BlackBox {
59   val bist_en = in Bool()
60   ...
61 }
```

Memlib.scala line 268

```
266 mem.component.rework {
267   ...
268   val toplevel = mem.component.parents().headOption.getOrElse(mem.component)
269   ram.io.bist_en := toplevel.asInstanceOf[Component{
270     val io: Bundle{val bist_en: Bool}}].io.bist_en.pull()
271 }
```

Extend Signal

Target:

```
val mem = Mem(Bits(8 bits), 64)
mem.write(io.waddr, io.win, io.we)
mem.readSync(io.raddr)
mem.bist(io.bist_en)
```

```
package object MemLib {
  implicit class MemBistEnPort[T <: Data](mem: Mem[T]) {
    def bistEn(that: Bool) {
      that.addTag(MemBistEn(mem))
    }
  }
}
```

Extend Signal

```
168 mem.component.dslBody.walkStatements{
169   case s: Bool => {
170     s.getTags().foreach{
171       case MemBistEn(m) => {
172         if(mem == m){
173           ram.io.bist_en := s
174         }
175       }
176       case _ =>
177     }
178   }
179   case _ =>
180 }
```

Ctrl Module

example.scala line 8

```
8  class Bist extends BistCtrl {
9    val io = new Bundle {
10     val en = in Bool()
11   }
12
13   override def tasks: Unit = {
14     mems.groupBy(x => (x._1, x._2))
15       .foreach{case ((cnt, width), mems) =>
16         val wr_addr = Counter(cnt, io.en)
17         mems.foreach{case (count, width, bundle, memcomponent, task) =>
18           this.rework {...}
19         }
20       }
21   }
22 }
```

Ctrl Module

MemBist.scala line 199

```
199 val toplevel = mem.component.parents().headOption.getOrElse(mem.component)
200 val bistctrl = toplevel.children.find(c => c.isInstanceOf[BistCtrl])
201 bistctrl match {
202   case Some(bistctrl) => {
203     val connect = (bundle: BistBundle, memcomponent: Component) => {
204       ... all the connection logics
205     }
206     bistctrl.asInstanceOf[BistCtrl].mems = (
207       mem.wordCount, mem.getWidth, cloneOf(ram.io.bist), mem.component,
208       connect(_, _)
209     ) :: bistctrl.asInstanceOf[BistCtrl].mems
210   }
211   case None => return "bist ctrl module not found"
212 }
```

Resize Lint

resize.scala line 13

```
13 pc.topLevel.walkComponents(c => {
14   c.dslBody.foreachStatements{
15     case as: AssignmentStatement => {
16       as.source match {
17         case s: BaseType => {
18           if(s.hasTag(tagAutoResize))
19           as.target match {
20             case t: BaseType => {
21               if(t.getBitsWidth < s.getBitsWidth)
                   PendingError(s"INVALID RESIZE (${t.getBitsWidth} bits <- $
22 {s.getBitsWidth} bits) on ${as.toStringMultiLine} at \n$
   {as.getScalaLocationLong}")
23             }
```

Thank You

QA: https://github.com/yportne13/SpinalMemBist