ScalaHDL

Yao Li

Contents

- Current Main.scala
- Expected Simulator Workflow

Current Main.scala

```
object Main extends ScalaHDL {
  sync('clk is 1)
  defMod.logic('d, 'q, 'clk) {
    'q := 'd
  }

  delay(10)
  defMod.clkGen('clk) {
    cycle('clk)
  }

  sync('clk is 0)
  defMod.stimulus('d, 'clk) {
    'd := Random.nextInt(2)
  }
```

```
  def main(args: Array[String]) {
    val q = Signal(0)
    val d = Signal(1)
    val clk = Signal(0)
    val sim = new Simulator(this)
    sim.simulate(
      module('logic, q, d, clk),
      module('clkGen, clk),
      module('stimulus, d, clk))
  }
}
```

Changes since last version

- now a HDL module is defined via defMod instead of module
- sync and delay "annotations" are added
- Int can be assigned to a HDL identifier directly
- there's a new class Simulator

Expected Simulator Workflow

- There's an instance of HDLModule class for each HDL module defined.
- Every statements inside a HDL module definition is stored in a way similar to AST.
- There will be a Waiter class. Each instance of them wraps a HDLModule.
- When a Waiter is called, it will call exec method on every statement of related HDLModule, and switch back to the caller's context.

The Waiter

```
class Waiter(m: HDLModule) {
  def next(wl: Seq[Waiter]) {
    val stmts = moduleStmts(m.name).reverse
    Iterator[List[Waiter]] {
      while (true) {
        yld(List())
        for (stmt <- stmts) stmt.exec
      }
    }
  }
}
```

- The Iterator and yld methods are from: http://stackoverflow.com/questions/2201882/implementing-yield-yield-return-using-scala-continuations
- **NOTE:** There're still bugs in this piece of code now and I'm trying to solve them.

Any Questions?

Thanks!