ScalaHDL

Contents

This time let's talk about some of the specifications of ScalaHDL, by going through our original proposal.

- **Proposal:** Implement 3 basic types: bool, signed, and unsigned, with these last two types having configurable bit-lengths; these values can also have an initialization value that will automatically be assigned when reset.
- `Bool(value: Int), Signed(value: Int, size: Int), Unsigned(value: Int, size: Int)`. (with "string" type to support **x** and **z**)
    - 2 options: `Signed(3)` means 2'b11 or a number of 3 bits?
- `init('a := 18)` or `init('a := 0x12)`
    - but **NOT** `init('a := 5b10010)`.

- **Proposal:** Implement all the basic logic operators: not, or, and, nand, xor, nor.

- Logical operators not, or, and, nand, xor, nor.

  - these can be used both as functions or modules, i.e. `not(‘a)` or `‘not(‘a)`.

  - the result could be 0, 1, x.

Equality Problem

- Despite the equality problem we discussed last time, things could get more complicated with **`x`** and **`z`**.
- `a == b`
    - what’s the result of `4bx001 == 4bx001`?

- **Proposal:** Allow the declaration of modules that can have both a number of inputs, and outputs, based on the basic types described above; these modules will be able to contain multiple synchronous or asynchronous blocks of logic and contain internally defined signals and registers.

- See next slide.

- In MyHDL:

```
def foo(args…):
   @always(…)
   def bar(args…):
      …
```

- In current ScalaHDL:

```
sync(arg is 1)
defMod.foo(args…) {
   …
}
```

- A solution:

```
defMod.foo(args…) {
    sync(arg is 1).bar {
        …
    }
}
```

- **Proposal:** Allow these modules to be interconnected manually or through recursive functions, with wiring automatically inferred.

- Two ports will be automatically wired if they are used by the same symbol in given environment.

- A module can use another module like a function call.

- A self-recursive module will produce “generate” of Verilog. (more details to be considered)

- If module A calls module B, while module B calls module A, things will get rather complicated…

- **Proposal:** Provide sugar syntax to minimize the designers effort in defining synchronous and asynchronous blocks, declaration of signals, sets of signals, instantiate modules, etc.

- `sync()`, `async()` functions.

- Signal declaration would be:

  - `‘a := Bool(0)`, `‘a` will be automatically added into symbol table if `‘a` has never been declared in given environment

  - or `val a = Bool(0)` (`a` doesn’t need to be `var` because `Bool` is mutable)

- **Proposal:** Output synthesizable HDL (e.g. Verilog or VHDL).

- Verilog for now.

- **Proposal:** Be able to be simulated such that values can be test programmatically using unit-tests and test-benches.

- We will be using Scala's test library with a ScalaHDL trait mixed in.

  - We use a trait instead of a class here thus users can choose their own favourite test library (e.g. scalatest, JUnit, etc.).

- **Proposal:** Such simulation need to be contained within a Scala instance and be able to be driven and drive Scala-defined functions.

- Simulation would be like what we have seen before.

- You can use Scala-defined functions both in modules for simulating, or inside a Scala function which calls the simulator.

- **Proposal:** Output traces of the internal signals so that they can be examined using a Value Change Dump (VCD) format (e.g. using GTKWave).
- As we have seen before.

- **Proposal:** ScalaHDL will support annotations to be used.
- No longer the case for now. We will be using normal function call.

- **Proposal:** ScalaHDL will provide more intuitive and comfortable syntax, like the example given below, and will be similar to another open source project, MyHDL.

- We will keep following this principle in designing more detailed specifications.

Any Questions?

Thanks!