SpinalHDL

Technicaly a Scala library

Scala in short

- General purpose programming language
- Paradigms
  - Imperative
  - Object oriented
  - Functional
  - Statistically typed
- Executed on the top of the JVM

Hello world

```
package myProject

object MyScalaProgram{
    def main(args: Array[String]) {
        println("Hello world")
    }
}
```

val and var

```
package myProject

object MyScalaProgram{
    def main(args: Array[String]) {

        var message : String = null;
        message = "Hello world"
        println(message)

    }
}
```

```
package myProject

object MyScalaProgram{
    def main(args: Array[String]) {

        val message = "Hello world";
        println(message)

    }
}
```

SpinalHDL is a Scala library

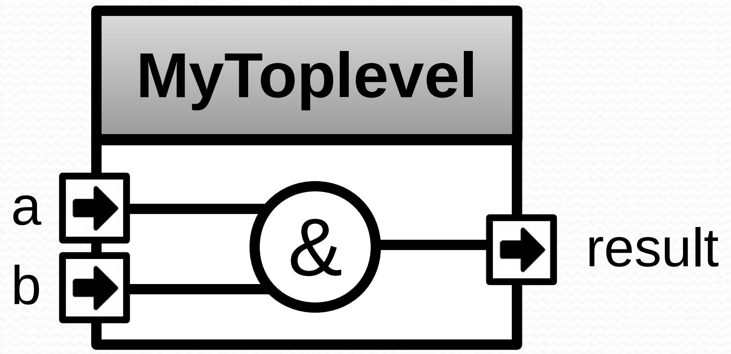
```
import spinal.core._

class MyToplevel() extends Component {
    //...
}

object GenerateToplevel{
    def main(args: Array[String]) {
        SpinalVerilog(new MyToplevel())
    }
}
```

SpinalHDL package structure

- spinal.core._
  - Bool, Bits, UInt, SInt, Component, Area, ...
- spinal.core.sim._
- spinal.lib._
  - Stream
  - bus
  - com
  - ...

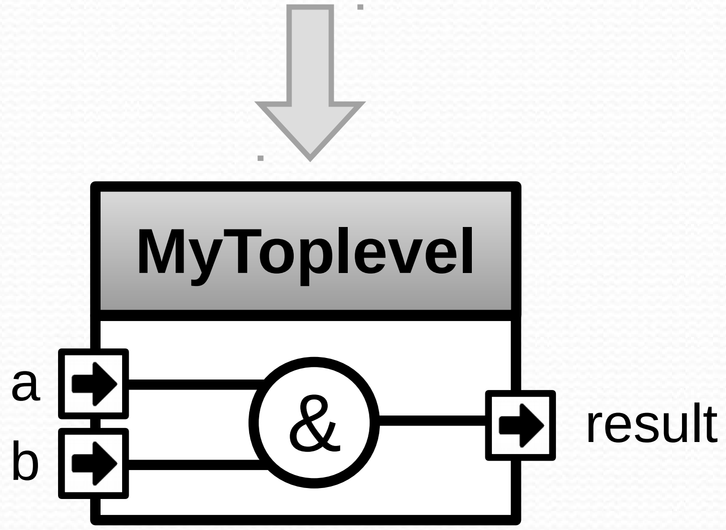
SpinalHDL register your actions at Scala runtime

```
import spinal.core._

class MyToplevel() extends Component {
    val a, b = in Bool()
    val result = out Bool()
    result := a && b
}

object GenerateToplevel{
    def main(args: Array[String]) {
        SpinalVerilog(new MyToplevel())
    }
}
```



A lot of it is makeup

```
class MyToplevel() extends Component {
    val a, b = in Bool()
    val result = out Bool()
    result := a && b
}
```



```
class MyToplevel3() extends Component {
    val in1, in2 = new Bool()
    in1.setName("a")
    in2.setName("b")
    in1.asInput()
    in2.asInput()

    val out1 = new Bool()
    out1.setName("a")
    out1.asOutput()
    out1.assignFrom(in1 && in2)
}
```

Scala control blocks

```
object MyScalaProgram{
    def main(args: Array[String]) {

        val message = "Hello world";
        if(message.contains("Hello")){
             println("all good")
        } else {
            println("Order 66")
        }


    }
}
```

Scala runtime do the hardware elaboration

```
class MyToplevel(doOr : Boolean) extends Component {
  val a, b = in Bool()
  val result = out Bool()
  if(doOr){ // This is a Scala if, not a SpinalHDL one
    result :=  a || b
  } else {
    result := a && b
  }
}

object GenerateToplevel{
  def main(args: Array[String]) {
    SpinalVerilog(new MyToplevel(doOr = true))
  }
}
```

```
module MyToplevel (
  input           a,
  input           b,
  output          result
);

  assign result = (a || b);

endmodule
```

Scala runtime do the hardware elaboration

```scala
class MyToplevel() extends Component {
    val a,b,c,d = in UInt(8 bits)
    val result = out UInt(8 bits)

    val inputs = List(a,b,c,d) // Scala list
    var accumulate = U(0, 8 bits)
    for(idx <- 0 to inputs.size-1){
        accumulate = accumulate + inputs(idx)
        accumulate.setName(s"accumulate_$idx")
    }
    result := accumulate
}
```

```verilog
module MyToplevel (
    input      [7:0]    a,
    input      [7:0]    b,
    input      [7:0]    c,
    input      [7:0]    d,
    output    [7:0]    result
);
    wire       [7:0]    accumulate;
    wire       [7:0]    accumulate_0;
    wire       [7:0]    accumulate_1;
    wire       [7:0]    accumulate_2;
    wire       [7:0]    accumulate_3;

    assign accumulate = 8'h0;
    assign accumulate_0 = (accumulate + a);
    assign accumulate_1 = (accumulate_0 + b);
    assign accumulate_2 = (accumulate_1 + c);
    assign accumulate_3 = (accumulate_2 + d);
    assign result = accumulate_3;

endmodule
```

Scala runtime do the hardware elaboration

```
class MyToplevel() extends Component {
  val a,b,c   = in Bool()
  val result = out Bool()

  def bufferThenOr(that : Bool) = {
    val reg = RegNext(that)
    when(reg){
      result := True
    }
  }

  result := False
  bufferThenOr(a)
  bufferThenOr(b)
  bufferThenOr(c)
}
```

```
module MyToplevel (
  input           a,
  input           b,
  input           c,
  output reg        result,
  input           clk,
  input           reset
);
  reg            a_regNext;
  reg            b_regNext;
  reg            c_regNext;
```

```
always @ (*) begin
  result = 1'b0;
  if(a_regNext)begin
    result = 1'b1;
  end
  if(b_regNext)begin
    result = 1'b1;
  end
  if(c_regNext)begin
    result = 1'b1;
  end
end

always @ (posedge clk) begin
  a_regNext <= a;
  b_regNext <= b;
  c_regNext <= c;
end

endmodule
```

How to preserve signal names

```
class MyToplevel() extends Component {
  val a = Bool()              //a.setName("a")
  val logic1 = new Area{
    val tmp = Bool()      // tmp.setCompositeName(this, "tmp")
  }                           // logic1.setName("logic1")
  def func1() = {
    val tmp = Bool()     //Nothing
  }
  def func2() = new Area {
    val tmp = Bool()    // tmp.setCompositeName(this, "tmp")
  }
  val logic2 = func2()  // logic2.setName("logic2")
}
```

```
module MyToplevel (
);
  wire          a;
  wire          logic1_tmp;
  wire          logic2_tmp;


endmodule
```

Function syntax (all func are the same)

```
def func(x : Int, y : Int) : Int = {
  val result = x + y
  return result
}

def func(x : Int, y : Int) = {  //Return type can by inferred
  val result = x + y
  result  //Last statement of the block is used as return value implicitly
}

def func(x : Int, y : Int) = {
  x + y
}

def func(x : Int, y : Int) = x + y
```

SBT

- Meaning : Scala Build Tool
- Kind of Makefile/CMAKE but for Scala
- Command example
  - clean
  - compile
  - runMain myPackage.MyMain
  - testOnly myPackage.MyTestUnit
- How to run a command
  - cd project_root
  - sbt #Wait to console to load, then type the command
  - sbt [command]*

SBT root file (build.sbt)

```
name := "SpinalTemplateSbt"
version := "1.0"
scalaVersion := "2.11.12"
val spinalVersion = "1.4.3"

libraryDependencies ++= Seq(
  "com.github.spinalhdl" % "spinalhdl-core_2.11" % spinalVersion,
  "com.github.spinalhdl" % "spinalhdl-lib_2.11" % spinalVersion,
  compilerPlugin("com.github.spinalhdl" % "spinalhdl-idsl-plugin_2.11" % spinalVersion)
)

fork := true
```

https://github.com/SpinalHDL/SpinalTemplateSbt/blob/master/build.sbt

SBT

- Meaning : Scala Build Tool
- Kind of Makefile/CMAKE but for Scala
- Command example
  - clean
  - compile
  - run-main myPackage.MyMain
- How to run a command
  - cd project_root
  - sbt #Wait to console to load, then type the command
  - sbt [command]*

SBT

- Meaning : Scala Build Tool
- Kind of Makefile/CMAKE but for Scala
- Command example
  - clean
  - compile
  - run-main myPackage.MyMain
- How to run a command
  - cd project_root
  - sbt #Wait to console to load, then type the command
  - sbt [command]*