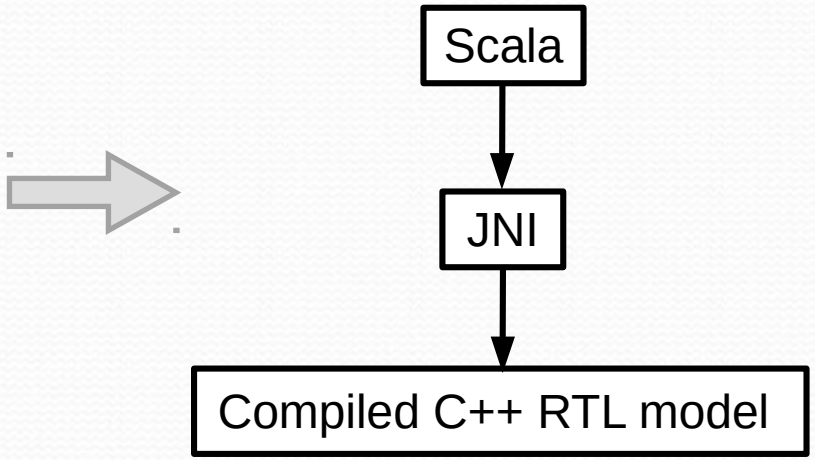
# SpinalHDL

## Simulation

# Flow

- Setup
  - Scala to Verilog  (via SpinalHDL)
  - Verilog to C++ (via Verilator)
  - C++ to shared object (via GCC)
  - Shared object binding into Scala (via JNI)




- Advantages
  - It test the generated Verilog
  - C++ model from Verilator are realy fast, free, open source and accurate
  - You can use all the Scala world to interract with your DUT (TCP, GUI, ...)

# API

- The Simulation API provide the following primitives :
  - read/write DUT interface signals
  - sleep a given time
  - waitUntil a given condition
  - fork/join threads
  - Scala for everything else
- It behave like a regular event-driven HDL simulation
  - It manage delta cycles
  - When you write a dut signals, the changed value will only be readable after the end of the current delta cycle
  - It remain deterministic

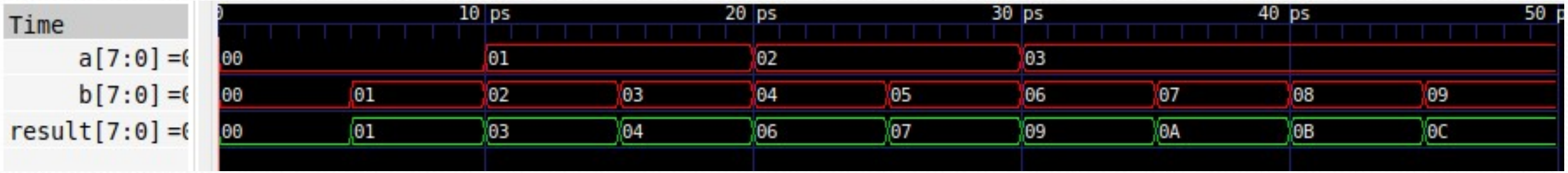
# Testbench template

```
object DutTestbench extends App{
  val compiled = SimConfig.withWave.compile(rtl = new Dut)

  compiled.doSim("test1"){ dut =>
    //Testbench code
  }

  compiled.doSim("test2"){ dut =>
    //Testbench code
  }
}
```

# Threads

```
SimConfig.withFstWave.compile(new Dut).doSimUntilVoid{dut =>
    fork{
        dut.a #= 0; sleep(6)
        dut.a #= 1; sleep(6)
        dut.a #= 2; sleep(6)
        dut.a #= 3; sleep(6)
    }
    fork{
        for(i <- 0 until 10){
            dut.b #= i
            sleep(i)
        }
    }
}
```

```
class Dut extends Component{
    val a,b = in UInt(8 bits)
    val result = out(a + b)
}
```



# Simple example

```
import spinal.core.sim._

object DutTestbench extends App{
    SimConfig.withWave.compile(new Dut).doSim{ dut =>
        dut.clockDomain.forkStimulus(period = 10)

        var idx = 0
        var resultModel = 0
        while(idx < 100) {
            dut.io.a #= Random.nextInt(256)
            dut.io.b #= Random.nextInt(256)
            dut.io.c #= Random.nextInt(256)
            dut.clockDomain.waitSampling()
            assert(dut.io.result.toInt == resultModel)
            resultModel = (dut.io.a.toInt + dut.io.b.toInt - dut.io.c.toInt) & 0xFF
            idx += 1
        }
    }
}
```

```
class Dut extends Component {
    val io = new Bundle {
        val a, b, c = in UInt (8 bits)
        val result = out UInt (8 bits)
    }
    io.result := RegNext(io.a + io.b - io.c) init(0)
}
```

# Example : JTAG TCP server

```
// Scala side thread used to provide a TCP connection
val server = new Thread    {
    override def run() = {
        val socket = new ServerSocket(7894)
        println("WAITING FOR TCP JTAG CONNECTION")
        while (true) {
            val connection = socket.accept()
            connection.setTcpNoDelay(true)
            outputStream = connection.getOutputStream()
            inputStream = connection.getInputStream()
            println("TCP JTAG CONNECTION")
        }
    }
}
server.start()
```

```
// Variables (shared)
var inputStream: InputStream = null
var outputStream: OutputStream = null

// Simulation thread used to read/execute the TCP stream
fork {
    while (true) {
        sleep(jtagClkPeriod * 200)
        while (inputStream != null && inputStream.available() != 0) {
            val buffer = inputStream.read()
            jtag.tms #= (buffer & 1) != 0;
            jtag.tdi  #= (buffer & 2) != 0;
            jtag.tck  #= (buffer & 8) != 0;
            if ((buffer & 4) != 0) {
                outputStream.write(if (jtag.tdo.toBoolean) 1 else 0)
            }
            sleep(jtagClkPeriod / 2)
        }
    }
}
```

# Questions ?

- Online documentation :
  - https://spinalhdl.github.io/SpinalDoc-RTD/SpinalHDL/Simulation/index.html
- Ready to use project :
  - https://github.com/SpinalHDL/SpinalTemplateSbt