SPINALHDL

Introduction:

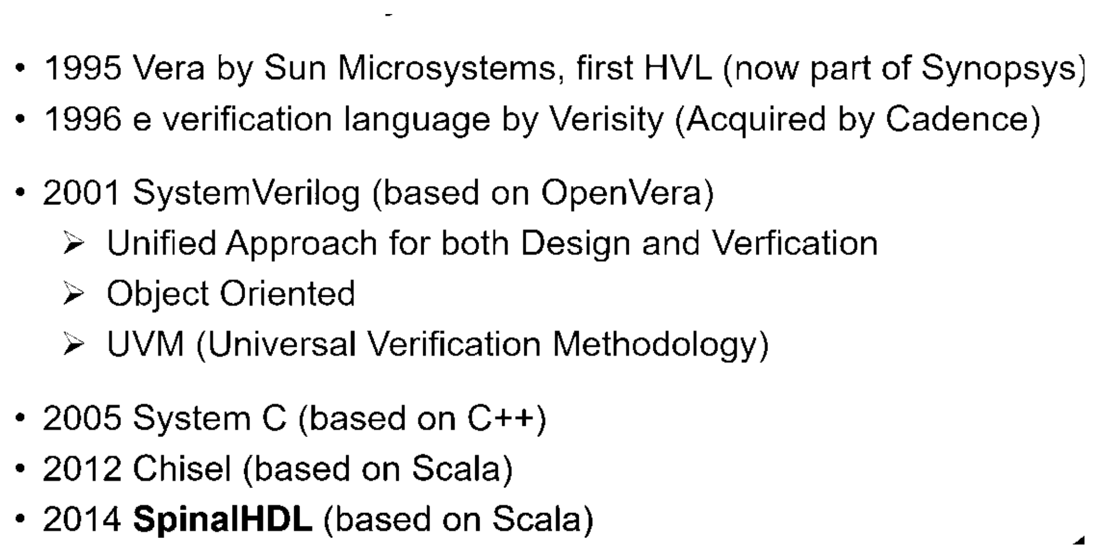
SpinalHDL is a high-level hardware description language (HDL) that is designed to be more modern and user-friendly compared to VHDL. SpinalHDL is integrated with the Scala programming language, which allows hardware designers to take advantage of high-level programming constructs for designing and testing hardware components.

It has its own toolchain for synthesis, simulation, and formal verification, including plugins for popular FPGA synthesis tools.

It is designed to improve productivity and simplify the description of digital circuits. It generates synthesizable Verilog or VHDL code from SpinalHDL source code.

History:

SpinalHDL is a newer language and has a smaller user base, which means that it may have fewer resources, libraries, and tools available.



- 1995 Vera by Sun Microsystems, first HVL (now part of Synopsys)
- 1996 e verification language by Verisity (Acquired by Cadence)
- 2001 SystemVerilog (based on OpenVera)
  - Unified Approach for both Design and Verfication
  - Object Oriented
  - UVM (Universal Verification Methodology)
- 2005 System C (based on C++)
- 2012 Chisel (based on Scala)
- 2014 **SpinalHDL** (based on Scala)

What is scala??

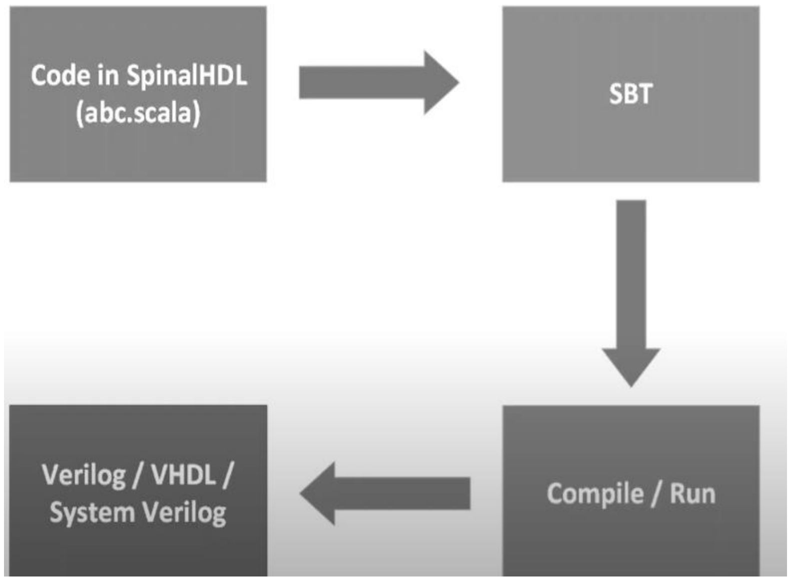
SpinalHDL uses the Scala programming language as its base language. Scala is a statically typed, multi-paradigm programming language that runs on the Java Virtual Machine (JVM).

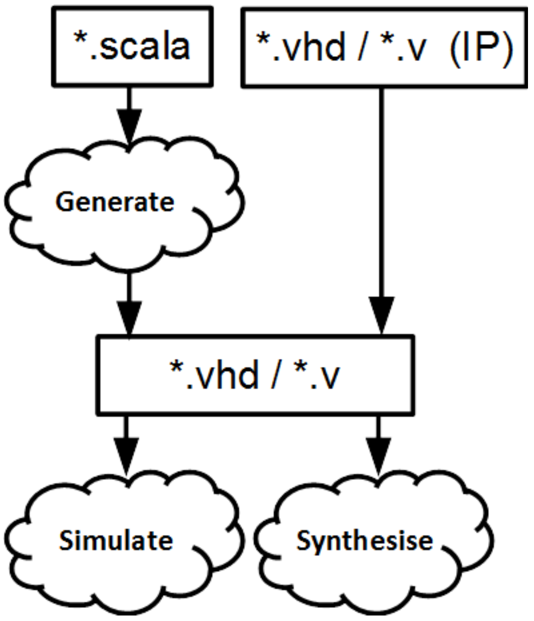
SpinalHDL leverages the features and syntax of Scala to provide a high-level hardware description language for designing digital circuits and systems.

Flowchart:

Once spnialhdl code has been written ,the tool can generate VHDL, Verilog codes and can give it to any simulator or synthesis tool. We can simulate this in any simulator such as Verilator or vivado etc.

SpinalHDL is interoperable with VHDL and (System)Verilog, We can do both instantiate SpinalHDL IPs in these language (using generated code) and instantiate IPs in these languages in SpinalHDL.


Code in SpinalHDL (abc.scala)
SBT
Compile / Run
Verilog / VHDL / System Verilog



About Verilator:

Verilator is an open-source tool used in digital design and hardware description languages (HDLs) for simulating and synthesizing digital designs written in languages like Verilog and VHDL. It primarily focuses on simulation and is known for its high-speed simulation capabilities, making it a popular choice. When designers want to verify the correctness and performance of their designs before moving on to synthesis and implementation on hardware platforms such as FPGAs or ASICs.

Verilator basically compiles the HDL code into optimized C++ code and simulates the behavior of the digital circuit. It does not appear to have native Windows support. However, Verilator works in WSL or in other Linux-compatible environments.

How it is different from Vivado:

Verilator and Vivado are two different tools used in the field of digital design and hardware development.

Vivado provides tools for the entire design flow, starting from designing digital circuits using HDLs (such as Verilog, VHDL, and SystemVerilog) to synthesis, place-and-route, bitstream generation, and programming FPGAs.

We might use Verilator to verify the functionality of your design in the early stages, and then use Vivado when you are ready to implement the design on specific hardware.

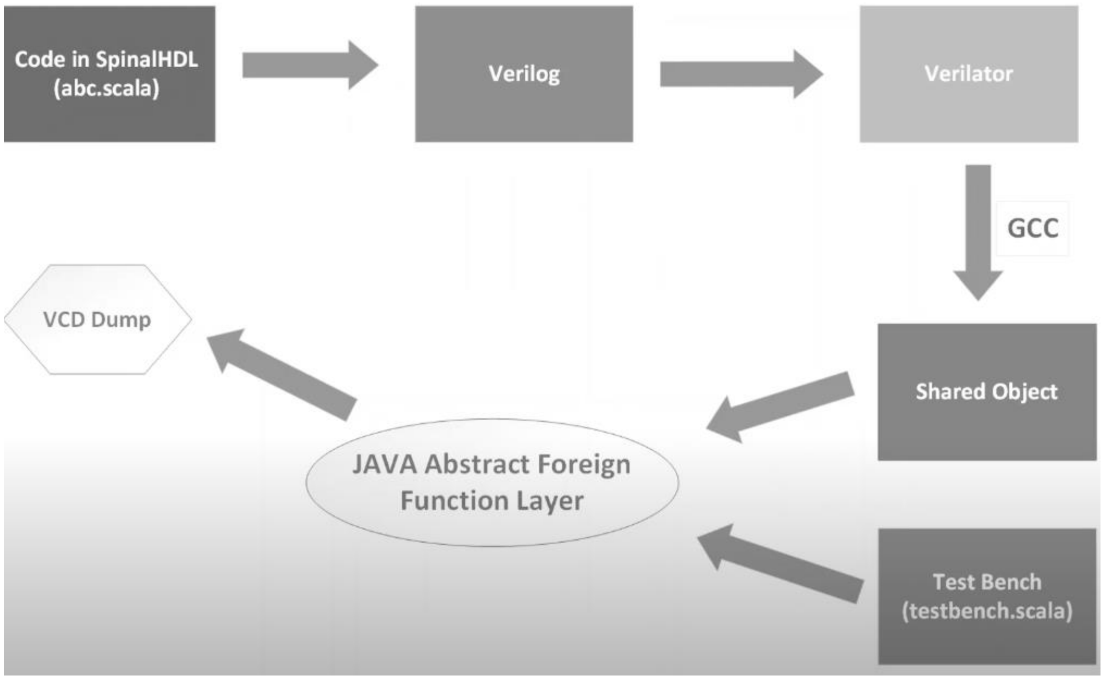
-->>Vivado is for mapping your design to the specific resources and architecture of Xilinx FPGAs.

->>Verilator's primary strength is in simulation, not synthesis.

General guide on how to use Verilator:

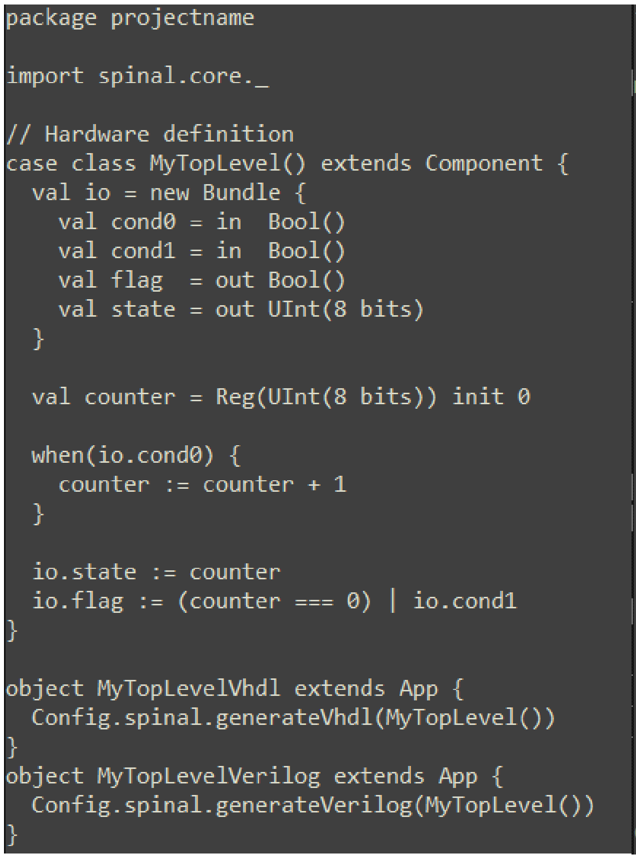
- Install Verilator
- Prepare Your Verilog Code
- Create a Testbench
- Compile Your Design
- After compiling your design, Verilator generates C++ source code that you can use to build a simulation executable.
- Run the Simulation, Analyze and Debug

Simulation flowchart:


Code in SpinalHDL (abc.scala)
Verilog
Verilator
GCC
Shared Object
Test Bench (testbench.scala)
JAVA Abstract Foreign Function Layer
VCD Dump

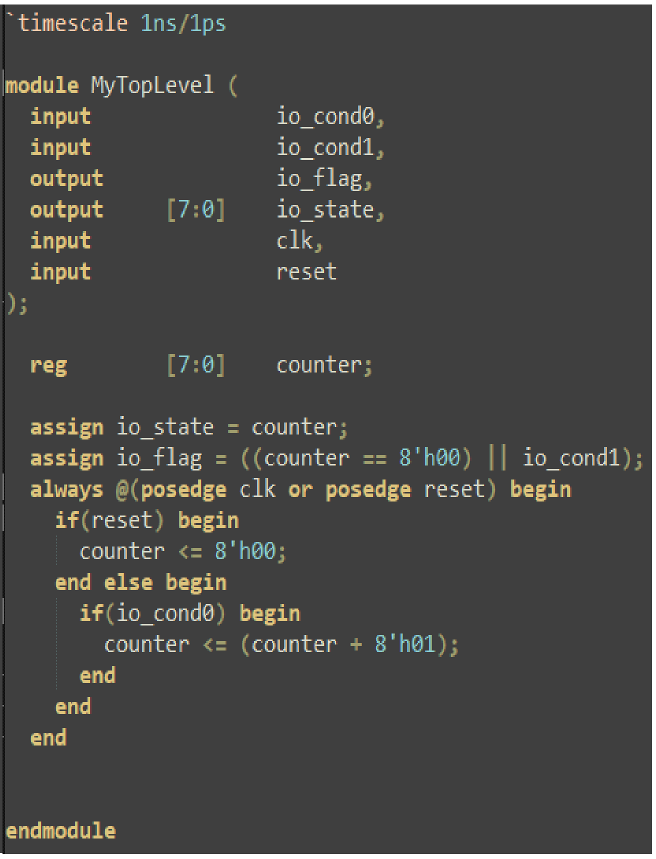
Output of the toolchain:

Below is the hardware Description Language of SpinalHDL based on scala which demonstrates a basic 8 bit up-counter. And it generates Verilog codes of that basic up-counter:



```scala
package projectname

import spinal.core._

// Hardware definition
case class MyTopLevel() extends Component {
  val io = new Bundle {
    val cond0 = in  Bool()
    val cond1 = in  Bool()
    val flag  = out Bool()
    val state = out UInt(8 bits)
  }

  val counter = Reg(UInt(8 bits)) init 0

  when(io.cond0) {
    counter := counter + 1
  }

  io.state := counter
  io.flag := (counter === 0) | io.cond1
}

object MyTopLevelVhdl extends App {
  Config.spinal.generateVhdl(MyTopLevel())
}
object MyTopLevelVerilog extends App {
  Config.spinal.generateVerilog(MyTopLevel())
}
```

SpinalHdl



```verilog
`timescale 1ns/1ps

module MyTopLevel (
  input                  io_cond0,
  input                  io_cond1,
  output                 io_flag,
  output     [7:0]       io_state,
  input                  clk,
  input                  reset
);

  reg        [7:0]       counter;

  assign io_state = counter;
  assign io_flag = ((counter == 8'h00) || io_cond1);
  always @(posedge clk or posedge reset) begin
    if(reset) begin
      counter <= 8'h00;
    end else begin
      if(io_cond0) begin
        counter <= (counter + 8'h01);
      end
    end
  end


endmodule
```

Verilog