Chisel 初步探索与实验

PLCT Lab 每周技术分享

熊家辉
浙江工商大学

2024 年 2 月 21 日

目录

第 1 节

Verilog 语言简介

第 1 节

Verilog 语言简介

第 1.1 小节

Verilog 语言

Verilog 语言

- **用途** Verilog 是一种用于描述、设计电子系统（特别是数字电路）的硬件描述语言，主要用于在集成电路设计，特别是超大规模集成电路的计算机辅助设计。
- **标准** Verilog 是电气电子工程师学会（IEEE）的 1800-2009 号标准。
- **语言要素** Verilog 的设计初衷是成为一种基本语法与 C 语言相近的硬件描述语言。

Verilog 代码示例

一段由 Chisel 吐出的 Verilog 代码

```
1 module ModuleSample(
2   input       clock,reset,
3   input  [7:0] io_a,io_b,
4   output [7:0] io_minnum,io_maxnum
5 );
6   wire _io_maxnum_T = io_a <= io_b;
7   assign io_minnum = _io_maxnum_T ? io_a : io_b;
8   assign io_maxnum = _io_maxnum_T ? io_b : io_a;
9 endmodule
```

第 1 节

Verilog 语言简介

第 1.2 小节

运行 Verilog

Verilator 比较

1 Verilator

- 接受 Verilog 或 SystemVerilog。
- 执行 lint 代码质量检查。
- 编译为多线程 C++ 或 SystemC。
- 优于许多闭源商业模拟器，单线程和多线程输出模型。
- 最广泛的使用。

1 iverilog

- 接受 Verilog 或 SystemVerilog （不完全支持）。
- 广泛的兼容性（包括 Windows®）。
- 部分指令支持不完全。（见 Unsupported Constructs）
- 速度较慢。

Verilator 简介

提示

该报告的重点为 Chisel 相关内容，Verilator 相关内容仅供参考。

- **用途** Verilator 是一款开源的支持 Verilog 和 System Verilog 仿真工具，它支持代码质量检查等功能，能够将给定的电路设计翻译成 C++ 或者 System C 的库等中间文件，最后使用 C/C++ 编写 testbench ，去调用前面生成的中间文件，由 C 编译器编译执行，来完成仿真。
- **主要功能** Verilator 的主要功能就是将 Verilog 代码转化为 SystemC 或 C++ 代码。

Verilator 工作原理

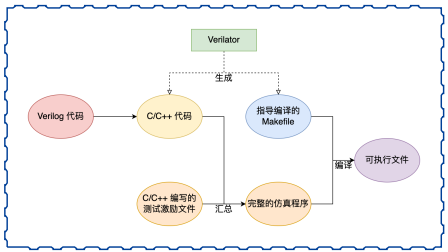

图: Verilator 工作原理

Verilator 工作流程

1. 硬件设计转化为一个处理器模拟器的软件。
2. 使用 C/C++ 编写激励文件。
3. 利用 GCC 等编译器将生成的 C/C++ 文件和我们编写的激励文件编译成成用于仿真的可执行文件。

Verilator 激励代码

节选自 Verilator 官方激励代码示例。

节选

```
1 VerilatedContext* const contextp = new VerilatedContext;
2 contextp->commandArgs(argc, argv);
3 Vtop* const top = new Vtop{contextp};
4 while (!contextp->gotFinish()) {
5     top->eval();
6   }
7 top->final();
```

第 2 节

Scala 简介

第 2 节

Scala 简介

第 2.1 小节

Scala 语言简介

Scala 语言

1. **目的** Scala 是一门多范式的编程语言，设计初衷是要集成面向对象编程和函数式编程的各种特性。
2. **编译** Scala 运行于 Java 平台（Java 虚拟机），并兼容现有的 Java 程序。
3. **特性** Scala是一种纯面向对象的语言、函数式语言、强静态类型语言。

语言提示

scale 一词指规模、尺度、比例，而 *scala* 没有意思。请在搜索时注意拼写。

Scala 代码示例

Hullo World!

```
1 object HelloWorld extends App {
2    println("Hello, world!")
3  }
```

第 2 节

Scala 简介

第 2.2 小节

Scala 实践

Scale 安装

参考 Installation。

- Java Development Kit：安装 `openjdk-21-jdk` 包。
- SBT：安装 `sbt` 包。

Scala 编译

将代码示例存入 HelloWorld.scala。运行下列命令编译。

编译 Scala 源代码

```
1 scalac HelloWorld.scala // 把源码编译为字节码
2 scala HelloWorld // 把字节码放到虚拟机中解释运行
```

第 2 节

Scala 简介

第 2.3 小节

ScalaTest

ScaleTest 简介

ScalaTest 是 Scala 生态系统中最灵活、最受欢迎的测试工具。
与构建它的 Scala 语言一样，ScalaTest 旨在随着用户的需求而增长：您可以轻松扩展和组合 ScalaTest 的核心组件，以满足您可能有的任何特殊需求。因此，ScalaTest 可以扩展到各种规模的项目，从探索新想法的个人到协作开发关键任务软件的大型团队。
ScalaTest 缩小规模的一种方式是，尽管ScalaTest 具有丰富的功能集，但它很容易上手。基于您从其他测试框架的经验中获得的知识，您可以通过 ScalaTest快速提高工作效率。

ScaleTest 代码示例

在包含了 `"org.scalatest" %% "scalatest" % "3.2.18" % "test"` 库后，可以使用下列代码进行测试。

Hullo ScalaTest!

```
1 class ExampleTest extends AnyFlatSpec with Matchers {
2     "Integers" should "add" in {
3         val i = 2
4         val j = 3
5         i + j should be (5)
6     }
7 }
```

第 3 节

Chisel 简介

第 3 节

Chisel 简介

第 3.1 小节

Chisel 库简介

Chisel 简介

1. **用途** Constructing Hardware in a Scala Embedded Language (Chisel) 是一种开源硬件描述语言，用于在寄存器传输级别描述数字电子设备和电路。
2. **语言** Chisel 将硬件构造添加到 Scala 编程语言中。
3. **目标** Chisel 可以编写复杂的、可参数化的电路生成器，从而生成可综合的 Verilog。
4. **手段** Chisel 可以创建可重用的组件和库，提高了设计的抽象级别，同时保留了细粒度的控制。

Chisel 代码示例

以下是来自 LED blink 的示例。

LED blink

```
1 class Blinky(freq: Int, startOn: Boolean = false) extends Module {
2    val io = IO(new Bundle {val led0 = Output(Bool())})
3    val led = RegInit(startOn.B)
4    val (_, counterWrap) = Counter(true.B, freq / 2)
5    when(counterWrap) {
6        led := ~led
7      }
8    io.led0 := led
9  }
```

Chisel 输出示例

来自 LED blink 的示例可以生成如下 Verilog 代码。

LED blink 的 Verilog 代码节选

```
1 if (reset) begin led <= 1'h0; counterWrap_c_value <= 9'h0; end
2 else begin
3 automatic logic counterWrap = counterWrap_c_value == 9'h1F3; led <=
      counterWrap ^ led;
4 if (counterWrap) counterWrap_c_value <= 9'h0; else counterWrap_c_value
      <= counterWrap_c_value + 9'h1;
5 end
```

第 3 节

Chisel 简介

第 3.2 小节

Chisel 实践

Chisel 安装

参考 Installation。

- Java Development Kit：安装 `openjdk-21-jdk` 包。
- SBT：安装 `sbt` 包。
- Verilator：安装 `verilator` 包。

Chisel 编译

使用 Chisel Project Template 作为模板，可以快速开始设计硬件。
在目录中运行 `sbt test` 可以测试当前安装是否正确。环境正常时，测试应当通过。
在目录中运行 `sbt run` 可以运行主程序。

Chisel 基础语法 I

GCD.scala

```
1  class ModuleSample extends Module {
2    val io = IO(new Bundle {
3      val a = Input(UInt(8.W))
4      val b = Input(UInt(8.W))
5      val minnum = Output(UInt(8.W))
6      val maxnum = Output(UInt(8.W))
7    })
8    io.minnum := Mux(io.a <= io.b, io.a, io.b)
9    io.maxnum := Mux(io.a <= io.b, io.b, io.a)
10 }
```

Chisel 基础语法 II

上述代码可以运行 `sbt run` 生成对应的 SystemVerilog 代码。

GCD.scala 对应 SystemVerilog 代码节选

```
1 module ModuleSample(
2   input        clock,reset,
3   input  [7:0] io_a,io_b,
4   output [7:0] io_minnum,io_maxnum
5 );
6   wire _io_maxnum_T = io_a <= io_b;
7   assign io_minnum = _io_maxnum_T ? io_a : io_b;
8   assign io_maxnum = _io_maxnum_T ? io_b : io_a;
9 endmodule
```

第 3 节

Chisel 简介

第 3.3 小节

Chisel 设计验证

Chisel 设计验证

参见 Design Verification。

- svsim 是 Chisel 的轻量级测试库，包含在此存储库中。
- chiseltest（适用于 Chisel 5.0 及之前版本）是用于基于 Chisel 的 RTL 设计的包含电池的测试和形式验证库，并且是以前的 PeekPokeTester 的替代品，提供相同的基本构造，但具有简化的接口和并发支持 fork 和 join 和 Verilator 集成进行模拟。

svsim I

svsim 是 Chisel 的轻量级测试库，包含在 Chisel 中。它是一个用于编译和控制 SystemVerilog 模拟的低级库，目前以 Verilator 和 VCS 作为后端。
Chisel 中可以使用 expect 来进行断言。通过对输入进行操作，然后让时钟前进一个周期即可获取结果。

svsim II

GCDSpec.scala

```
1 class GCDSpec extends AnyFreeSpec with Matchers {
2   "min and max check" in {
3     simulate(new ModuleSample) { dut =>
4       var a,b : Int =100;
5       dut.io.a.poke(a);dut.io.b.poke(b);dut.clock.step()
6       dut.io.minnum.expect(min(a,b),"Min Value test failed.");dut.io.
            maxnum.expect(max(a,b),"Max Value test failed.")
7     }
8   }
9 }
```

svsim III

Chisel 中可以使用 println 输出任何东西。

svsim IV

GCDSpec.scala

```
1 class GCDSpec extends AnyFreeSpec with Matchers {
2   "min and max check" in {
3     simulate(new ModuleSample) { dut =>
4       var a,b : Int =100;
5       dut.io.a.poke(a);dut.io.b.poke(b);dut.clock.step()
6       println("Result: " +a.toString+" & "+b.toString+" = "+dut.io.minnum
            .peek().toString)
7     }
8   }
9 }
```

Chiseltest I

版本支持

Chisel 6.0.0 在 2024 年 1 月 19 日发布。根据 Chisel Project Versioning 的约定，主要版本更新不会提供向前的兼容性。而根据 ucb-bar/chiseltest#699，该库尚未对 Chisel 6.0.0 提供支持。下列内容未经验证，仅供参考。

Chiseltest 是用于基于 Chisel 的 RTL 设计的包含电池的测试和形式验证库。
Chiseltest 强调测试的轻量级（最小化样板代码）、易于读写（可理解性）和组合（为了更好的测试代码重用）。同时，它提供更加复杂的多线程测试等。
以下代码为队列测试的示例，见QueueTest.scala。

Chiseltest II

GCDSpec.scala

```
1  it should "work with a combinational queue" in {
2    test(new PassthroughQueue(UInt(8.W))) { c =>
3      c.in.initSource();c.out.initSink()
4      fork {
5        c.in.enqueueSeq(Seq(42.U, 43.U, 44.U))
6      }.fork {
7        c.out.expectDequeueSeq(Seq(42.U, 43.U, 44.U))
8      }.join()
9    }
10 }
```

第 3 节

Chisel 简介

第 3.4 小节

Chisel 和 VS Code

VS Code 对 Chisel 的支持

Chisel 是一个 Scala 库。因此 Chisel 主要使用 Scala 的相关插件。为支持 Chisel 的，VS Code 主要使用 3 个插件。

- scalameta.metals: Scala 感知和 sbt 相关支持。
- scala-lang.scala: Scala 语法高亮
- aaronduino.chisel: Chisel 语法和代码块支持。

VS Code 的 Chisel 效果

图: VS Code 的 Chisel 智能感知和静态检查演示

第 3 节

Chisel 简介

第 3.5 小节

测试 Chisel

测试 Chisel I

参考 Installation，首先安装所需的依赖项才能在本地构建 Chisel，安装 `openjdk-21-jdk` 包、安装 `sbt` 包、安装 `verilator` 包。
运行下列代码克隆并构建 Chisel。

构建 Chisel

```
1 git clone https://github.com/chipsalliance/chisel.git
2 cd chisel
3 sbt compile
```

为了运行单元测试，PATH 中应包含 verilator、yosys 和 espresso。

测试 Chisel II

如果编译成功并且安装了上述依赖项，则可以通过调用以下命令来运行包含的单元测试。

构建 Chisel

```
1 sbt test
```

第 4 节

Chisel 和 RISC-V

第 4 节

Chisel 和 RISC-V

第 4.1 小节

在 openEuler RISC-V 上运行 Chisel

运行 Verilator I

注意到 openEuler RISC-V 尚未提供 `verilator` 包。参阅 Git Quick Install 的说明。

构建 Verilator

```
yum install -y git help2man perl python3 make autoconf g++ flex bison
    ccache numactl
git clone https://github.com/verilator/verilator --depth=1
cd verilator
autoconf
./configure
make -j `nproc`
sudo make install
```

运行 Verilator II

按照有关提示，运行 `make test` 进行有关测试，部分测试可以通过，部分测试由于缺少包而无法通过或跳过测试。这说明 Verilator 能够在该系统上运行。

Verilator 验证结果

```
1 ==SUMMARY: Passed 1 Failed 0 Time 2:55
2 ==SUMMARY: Passed 1 Failed 0 Time 2:55
```

运行 Scale

参考 17 安装并测试 Hullo World 程序，能够运行通过。

运行 Chisel

参考 Did it work? 的有关说明，在目录中运行 `sbt -Dsbt.override.build.repos=true test --debug`，得到如下结果。猜想问题为 JVM 虚拟机内部实现可能不符合预期，需进一步排查或对 Chisel 进行进一步测试。

构建 Verilator

```
1 scala.MatchError: false (of class scala.reflect.internal.Trees\$Literal)
2         at scala.tools.nsc.typechecker.Macros\$MacroImplBinding\$.
              unpickleAtom(Macros.scala:118)
3         at scala.tools.nsc.typechecker.Macros\$MacroImplBinding\$\
              $anonfun\$7.apply(Macros.scala:186)
4         at scala.tools.nsc.typechecker.Macros\$MacroImplBinding\$\
              $anonfun\$7.apply(Macros.scala:186)
```

第 4 节

Chisel 和 RISC-V

第 4.2 小节

使用 Chisel 的 RISC-V 项目

使用 Chisel 的 RISC-V 项目

- The Sodor Processor Collection: RV32I 处理器的简易实现
- NutShell: OSCPU（大学开源芯片项目）团队开发的处理器，目前支持 RV64IMAC 和 RV32IMAC。
- Rocket Chip Generator: 一款开源片上系统设计生成器，可生成 RTL。
- 香山: 一款开源的高性能 RISC-V 处理器，架构为 RV64GCBK。
- BOOM: Christopher Celio 的 RV64 乱序处理器实现。
- BottleRocket: RV32IMC 微处理器。

谢 谢