

Agile SoC Design with Spinal HDL

Pu Wang @ DatenLord
Jijing Guo @ Alibaba

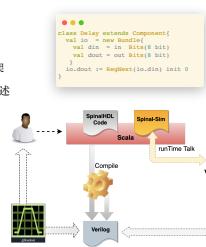


什么是SpinalHDL

-Charles Papen
@Dulu1990



- 一门硬件描述语言，基于Scala上的HDL框架
- Verilog生成器，不是HLS, RTL层面进行描述
- 是传统HDL的一个超集
- 提供更高级别的参数化和组织抽象能力
- 强大的元编程能力(寄生Scala)
- 提供一整套lib和SoC互连方案
- 提供一整套基于开源项目的前端开发方案



传统HDL有什么问题

思考

- 你觉的Verilog/VHDL/SV哪些地方你不能忍受?
- 像让你来改进Verilog, 你会提供那些feature?
- 你希望EDA工具应该增加那部分feature?
- 你有没有用原本或其他工具来生成rtl代码的经历?

问题

- 例化繁琐，大量的手动连线（插件只能解决部分问题）
- 大量的重复声明（无体化的中间信号位宽声明）
- 函数不能带参数（假设）
- 模块参数化能力弱（基于+ parameter不支持函数）
- 错误检测能力弱（需要依赖EDA工具检查错误）
- 重构、增减信号麻烦，即便是依赖脚本，也不通用
- 低级错误积累繁杂，增加验证成本、增长验证收敛周期
- 基础电路重复书写、工程身心疲惫
- 电路实现精度低，并发逻辑跟模块大小成线性关系
- 语言缺乏泛化结构化的元素
- 事件驱动的语式并不是用来RTL描述的，它是用来仿真行为建模

错误检查弱 参数化能力弱 复用程度低



HDL语言的3个演进方向

Super Verilog

优点：语法有延续性

问题：没能根本上解决参数化、错误检查的问题，EDA对新特性支持并不积极

HLS

优点：对于某些C参考平台可以快速生成硬件电路

问题：C代码限制较多，消耗有限性，面向冗余不可控，Debug空虚，ECO困难，高危芯片采用的较少

DSL based

优点：基于RTL级别的描述，参数化能力，高效的生成器，没有冗余

问题：百花齐放，生态割裂，迫切需要一个更强力的语言去领导

Scala

基于SCALA的硬件构筑语言(泛CHISEL)
Constructing Hardware In Scala Embedded Language

CHISEL

spinalHDL



泛CHISEL特性

- 面向对象、根基一切对象，电路对象跟字符串一样
- 类型推断(动态语言的使用感受，静态语言的安全)
- 免去费时的IDE支持
- 大量的基础组件以及可重用IP
- 模式匹配
- 函数式编程
- 支持中缀表达式，非常适合做DSL `dout := din`
- 隐式转换、隐式扩展

- 面向对象、根基一切对象，电路对象跟字符串一样
- 类型推断(动态语言的使用感受，静态语言的安全)
- 免去费时的IDE支持
- 大量的基础组件以及可重用IP
- 模式匹配
- 函数式编程
- 支持中缀表达式，非常适合做DSL `dout := din`
- 隐式转换、隐式扩展



SpinalHDL能解决什么问题



- 静态错误(lint) 100% 覆盖
- 避免繁琐的声明
- 互连集成自定义检查规则

- 最小信息原则
- 验证潜力(SuperUVVM)
- 创建基础lib
- IP参数化设计
- Soc顶层互连简化

Never Generate Broken Verilog

Why not Super SystemVerilog

- 有限关键字
- 没有扩展能力，取决于标准的定义
- 宏的能力有限(`define`, `ifdef`)
- EDA工具的支持

Scal相对于SpinalHDL是它的宏
可以随时随地对创造或者加工电路对象
赋予工程师灵活的手段去丰富电路对象

SpinalHDL能达到什么效果

Lint错误

100% 解决

- 包括但不限于
- Assignment overlapping
- Clock domain
- Hierarchy violation
- Combustional loops
- Latches
- Undriven signals
- Width mismatch
- Unreachable switch statements

可定制的IP

一次开发
10s生成

- 包含但不仅限于
- 轴连接
- 桥接
- 译码器
- 仲裁器
- RAM
- 开关机

SOC层面

可插拔的框架

- 包含但不仅限于
- 轴连接
- 桥接
- 译码器
- 仲裁器
- RAM
- 开关机

团队的层面

丰富可重用的基础组件

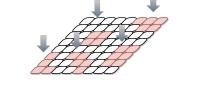
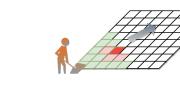
- 包含但不仅限于
- 轴连接
- 桥接
- 译码器
- 仲裁器
- RAM
- 开关机

形成规模效应

项目线的人力扩展不再线性增长，边际成本摊薄



ABC Verilog VS SpinalHDL



- 代码冗余，错误检查弱
- 边界需要手动处理
- 牵一发动全身
- 扁平、复用低
- 人力堆砌，规模和人力往往线性关系
- 简单明了
- EDA工具直接支持

- 更少的代码量
- 安全可插拔
- 边界自动适配，安全检查
- 高效的复用
- 芯片规模跟人力有望指数关系
- 警惕过度设计，复杂化



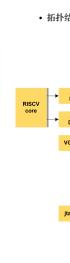
如何用定制PinSel



iCache配置



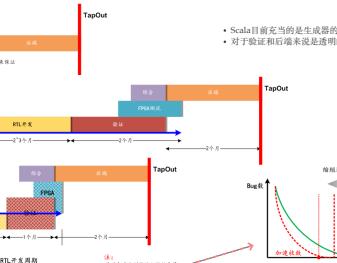
总线路由



拓扑结构

理想

在IC开发流中的位置



Agile development



关于敏捷开发



IC敏捷不捷



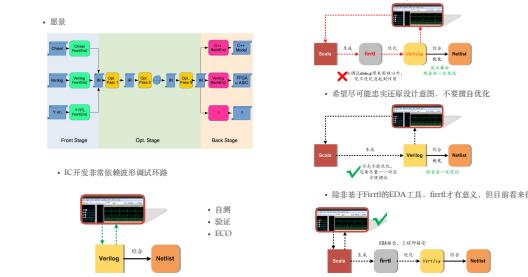
Chisel



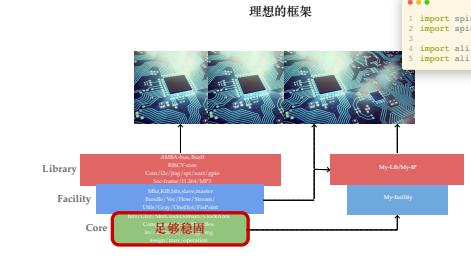
SpinalHDL

- Firrtl非常争议，对Flow使徒增困难，并没带来实质的好处
- Verilog代码风格较差，可读性差，不支持case表达式
- chisel的基本类型设计过于复杂
- chisel不支持异步复位(早期版本)，没有clockDomain
- 做一些不必要的强相关(IO, Module, Wire)
- RocketChip的隐式参数非常丑陋，设计扁平
- ioTest没实用意义(好在iotest2慢慢步入正轨)

Firrtl现状讨论



理想的框架



问题回顾

- 1. Verilog/SV不用够，SpinalHDL/chisel真的有必要吗？
- 2. Scala生成的代码有没有冗余，时序会不会变差？
- 3. 用Scala开发我怎么去debug硬件波形
- 4. Scala的代码可读性怎么样，谁来维护？
- 5. 怎么传统的IC团队和Flow兼容？
- 6. 有了SpinalHDL，是不是一个月的工作几天就能完成？
- 7. SpinalHDL适合大项目，高性能系统的开发吗？
- 8. Chisel和SpinalHDL该如何选择？