内容

1. 综述

1.1 范围

此标准提供 IEEE 1800 SystemVerilog 语言的语法和语义的定义, 此语言是统一的硬件设计,规范和验证的语言. 此标准包括了对行为级(behavioral),寄存器传输级(RTL) 和门级(gate-level)硬件描述的支持; 支持 testbench, coverage, assertion, 面向对象(object-oriendted) 和随机约束结构; 以及为其他编程语言提供 API(application programming interfaces).

1.2 目的

此标准旨在满足日益增长的硬件规范, 设计, 验证语言使用需求. 此版本修复了 IEEE Std 1800-2012 中的语言定义的错误. 此版提供了更强的特性以易于设计, 验证, 跨编程语言交互.

1.3 内容简介

该标准为 SystemVerilog 语言的完整标准. 此标准包含以下内容:

- 所有 SystemVerilog 结构的形式化语法和语义.
- 用于仿真的系统 task 和 function, 比如文本输出命令.
- 编译器指令, 如文本替换宏和仿真时间刻度.
- 编程语言接口(Programming Language Interface, PLI)机制
- SystemVerilog 的 Verification Procedual Interface(VPI) 的形式化语法和语义.
- 不包含在 VPI 内的一个 Application Programming Interface(API), 用于覆盖率(coverage)访问.
- Direct programming interface(DPI), 用于与 C 语言互动.
- VPI, API, DPI 的头文件
- concurrent assertion 的形式化语义
- standard delay format(SDF)结构的形式化语法和语义
- 非正式的用例

1.4 特定术语

以下术语贯穿此文档:

- SystemVerilog 3.1a 指 Accellera SystemVerilog 3.1a Language Reference ManualB4
- Verilog 指 IEEE Std 1364-2005
- Language Reference Manual (LRM) 指 Verilog 或 SystemVerilog 的标准文档
- Tool 指读取 SystemVerilog 源码的软件实现, 如逻辑仿真器(Logic Simulator)

1.5 此标准中的约定用法

此标准组织成 clause, 每个 clause 关注此语言的一个特定方面. 在 clause 中还有 subclause 用于讨论各个 construct 和 concept.

以下为贯穿此标准的措辞约定:

- "Shall" 用于表示强制要求, 必须严格遵循此标准, 不能有偏差.
- "Should" 用于表示在多个可能的选项中某个选项更为合适, 但是并不排除其他选项; 或表示某个行为更合适但不强制要求; 或表示不期望出现某个行为, 但并不禁止出现此类行为.
- "May" 用于表示可以在标准允许的范围内的行为.
- "can" 用于陈述.

> 译者注: 使用"必须"替代"Shall", 使用"可以"替代"Can", 另外两个直接使用英文表示.

1.6 语法描述

正文使用以下约定:

- 定义一个术语的时候使用*斜体*
- ***斜体加粗***用于举例, 文件名再, 常量引用, 特别是 0, 1, x, z 值.
- 当指代真正的 SystemVerilog 关键字时使用 **加粗** 字体表示.

> 译者注: "斜体加粗"在原文中是 constant-width, "加粗" 在原文中是 boldface constant-width. 译者使用 Markdown 无法打出这些字体.

使用巴克斯范式(Backus-Naur Form,BNF) 描述 SystemVerilog 的语法. 约定如下:

- 小写单词, 单词间可能有下划线"_", 表示 syntactic category. 例如: module_declaration
- 使用红色字符表示语法中所需的关键词, 操作符, 标点符号,例如: `module => ;`
- 使用垂线 "|"分割不同的可选择的 `items`,例如: unary_operator::= `+` | `-` | `!` | `~` | `&` | `|` | `~|` | `^` | `~^` | `^~` |
- 使用不加粗的中括号"[]"括住可选的 item. 例如: function_declaration::=`function` [lifetime] [function_body_declaration]
- 使用不加粗的大括号"{}"括住一个重复的 item. 此 item 可以出现 0 次或2多次; 与 equivalent left-recursive 规则一样, 从左向右开始重复. 因此, 以下两条规则等价:
  - list_of_param_assignments ::= param_assignment { `,` param_assignment}
  - list_of_param_assignments ::= param_asignment | list_of_param_assignments `,` param_asignment

语法中的*限定性术语*是一个术语, 如 *array_identifier*, 在此术语中, "array" 表示了语义内容, "identifier" 术语表示此限定性术语归约为了此语法中的术语 "identifier", 即 array_identifier 是 identifier. 此语法并没有完整的定义此类限定性术语的语义; 比如通过声明语句可以将一个 identifier 语义上地限定为 array_identifier, 但在使用 array_identifier 的语法中并未显式地描述声明的格式.

1.7 此标准中使用的颜色

此标准中只使用少量颜色以增强阅读性. 颜色并非必要的, 也不影响阅读黑白版本的标准文档. 在以下情况使用颜色:

- 超链接到此标准的其他部分使用${\color{blue}{蓝色}}$, 如 B4
- 在形式化语言定义中的语法关键字和 token 使用红色
- 某些图片使用了少量的颜色以增强阅读性

1.8 此标准的内容

略略略略略略略略

1.9 弃用的 clause

Annex C 中列出了之前版本的 IEEE Std 1364 或 IEEE Std 1800 中存在但现已弃用的 construct. Annex C 也列出了此标准中出现了的 construct, 但是这些 construct 在此标准的未来版本中可能弃用.

1.10 例子

此标准中有少量的 SV 代码例子. 这些是非正式的例子. 它们用于展示 SV construct 在一个简单的上下文中的用法, 而非定义完整的语法.

预备知识

此标准中的部分 clause 假定您有一定的 C 语言编程经验.

2. Normative references

略略略略略略略略

3. 设计和验证的构造块

3.1 概述

此 clause 讨论以下内容:

- module, programs, interface, checkers, primitives 的用途
- subroutine 概述
- packages 概述
- configurations 概述
- 设计的层级结构 概述
- compilation 和 elaboration 的定义
- Declaration name spaces
- 仿真时刻, 仿真单位和仿真精度

此 clause 定义了 SV 中许多重要的术语和概念, 且贯穿此文档. 此 clause 提供了用于硬件设计和仿真环境的建模块的用途和用法.

3.2 设计元素

一个*设计元素*是一个 SV module, program, interface, checker, package, primitive 或 configuration. 使用以下关键字引用上述结构: **module**, **program**, **interface**, **checker**, **package**, **primitive**, 和 **config**.

设计元素是用于建模和构建一个设计和验证环节的基本构造块. 这些构造块可以包含声明和 procedual 代码.

此 clause 描述了这些构造块的用途. 它们的完整语法和语义定义在后文给出.

3.3 Modules

SV 中的基本构造块是 module, 使用关键字 **module** 和 **endmodule**包围. modules 主要用于表示设计块, 但是也可以包含验证代码以及在验证块和设计块之间通信. module 可以包含以下结构:

- port
- 数据声明, 如 nets, variable, structure 和 union
- 常量(constant)声明

- 用户自定义类型定义
- Class 定义
- 从 package 中引入声明
- 定义 subroutine
- 例化其他 module, program, interface, checker, primitive
- 例化 class 对象
- continuous assignment
- procedual 块
- generate 块
- specify 块

在此标准的后文中陆续讨论上述结构.

注意--上述并未列出所有结构. module 可以包含其他的结构, 这些结构也会在后文中讨论.

下述是一个使用 module 表示 2 选 1 多选器 (multiplexer) 的简单例子:

```
module mux2to1 (
  input wire a, b, sel,
  output logic y
);
always_comb begin
  if(sel) y = a;
  else    y = b;
end
endmodule: mux2to1
```

关于 module 的更多细节见23. 模块和层级结构 以及 3.11 节.

3.4 Programs

program 构造块使用关键字 **program** .. **endprogram** 包围. 此结构用于建模 testbench 环境. 尽管 module 可以很好的描述硬件, 但是对于 testbench 来说, 重点并不是硬件级的细节, 比如 wire, 结构层级, 互联, 而是在于建模完整的用于验证设计的环境.

program 快服务于以下三个目的:

- 提供 testbench 的执行入口点
- 创建一个 scope, 在此 scope 中封装 program-wide data, task 和 function.
- 提供一个语法上下文, 用于指定 Reaction 区域中的调度.

program 结构是 testbench 和设计之间明确的分界线, 更重要的是其指定了专门的仿真执行语义. 协同14. 时钟块 (**clocking**) 为设计和 testbench 之间提供了不会产生竞争的互动方式, 且可以使用 cycle 和 transaction 级的抽象.

program 块可以包含数据声明, 类定义, subroutine 定义, 对象实例, 一个或多个 initial 或 final procedure.
program 不可以包含 always procedure, primitive 实例, module 实例, interface 实例, 其他 program 实例.

SV 中的抽象和建模结构简化了 testbench 的创建和维护. 实例化和单独地连接不同 program 实例的能力使得它们能够用作通用模型.

声明 program 的例子:

```
program test (input clk, input [16:1] addr, inout [7:0] data);
  initial begin
    ...
endprogram
```

在 24. Programs 对其进行更细致的讨论.

3.5 Interfaces

使用关键字 **interface**...**endinterface** 包围 interface 结构. interface 封装了用于设计块之间, 设计块和验证块之间的通信, 允许从抽象的系统级设计通过连续的细化以平滑的迁移到低级别的寄存器传输和设计的结构视图上. 通过封装块之间的通信, 有助于设计重用.

在最低级, interface 是一组命名的 net 和 variable, 在设计中例化的 interface 可以连接其他例化了的 module, interface, program. 可以通过端口访问 interface, 可以在需要的时候引用 interface 中的 net 和 variable. 设计中经常包含端口列表和端口连接列表, 这些列表只是不断重复的名字. 使用单个名字替换掉一组名字的能力可以极大的降低设计描述的大小以及增强可维护性.

interface 的另外一个能力是封装功能, 使得在一个接口更像一个类模板. interface 可以有参数, 常量, 变量, 函数和任务. 可以声明 interface 中元素的类型, 或者是通过参数传递元素的类型. 使用 interface 实例的名字对成员变量和函数进行引用. 因此, 通过 interface 连接的 module 可以简单地调用 interface 的 subroutine 成员驱动通信. 因此, 可以通过替换拥有相同成员的不同接口(接口的实现的抽象级别冉)以改变通信的抽象级别和/或通信协议的粒度. 而通过此接口连接的 module 无需做出任何改变.

为给 module 的端口提供方向信息以及控制特定模块使用的 subroutine, 提供了 **modport** 结构. 就像其名字, 从 module 的视角定义方向信息.

除了 subroutine, interface 可以包含 process (例, nitial 或 always procedure) 以及 continuous assignment (对 testbench 和 系统级建模很有用). 这样, interface 就可以包含自己的协议检查器, 以自动地验证通过 interface 连接的所有模块是否符合指定协议. 还能实现 functional coverage 记录和报告, 协议检查, 断言.

interface 定义和使用的一个例子:

```
interface simple_bus(input logic clk); // 定义 interface
  logic req,gnt;
  logic [7:0] addr, data;
  logic [1:0] mode;
  logic start, rdy;
endinterface

module memMode(simple_bus a); // simple_bus interface 端口
  logic avail;
  // 当在顶层模块实例化 memMod时, a.req 是"simple_bus" interface 的
  // 实例 sb_intf 的 req 信号
  always @(posedge a.clk) a.gnt <= a.req & avail;
endmodule
```

```
module cpuMod(simple_bus b);
  ...
endmodule

module top;
  logic clk = 0;

  simple_bus sb_intf(.clk(clk)); // 实例化 interface

  memMod mem(.a(sb_intf)); // 连接 interface 到 module 实例
  cpuMod cpu(.b(sb_intf)); // 连接 interface 到 module 实例
endmodule
```

25. Interfaces给出了 interface 的完整描述.

3.6 Checkers

略

3.7 Primitives

略

3.8 Subroutines

略

3.9 Packages

略

3.10 Configurations

略

3.11 Overview of hierarchy

略

3.12 Compilation and elaboration

略

3.13 Name spaces

略

3.14 Simulation time units and precison

4. 调度语义

5. 词法约定

6. 数据类型
7. 聚合数据类型
8. 类
9. Processes
10. Assignment statement
11. 操作符和表达式
12. Procedual programming statements
13. 任务和函数(子任务)
14. 时钟块
15. 跨 Process 的同步和通信
16. Assertions
17. Checkers
18. 产生受约束的随机值
19. 功能覆盖率
20. Utility 系统任务和系统函数
21. 输入/输出系统任务和函数
22. Compiler directives
23. 模块和层级结构
24. Programs
25. Interfaces
26. Packages
27. Generate 结构
28. 门级和开关级建模
29. 用户原语定义
30. Specify blocks
31. Timing checks

32. 使用 Standard delay format 进行反标

33. 配置一个设计的内容

34. 受保护的 Envelopes

35. DPI

36. PLI

37. VPI

38. VPI 调用定义

39. 断言 API

40. 代码覆盖率控制和 API

41. 数据读取 API

B4

Annex A (正式的) 形式化语法

Annex B (正式的) 关键字

Annex C (正式的) 已弃用的 clause

Annex D (非正式的) 可选的系统任务和系统函数

Annex E (非正式的) 可选的 compiler directives

Annex F (正式的) 并发断言的形式化语义

Annex G (正式的) Std package

Annex H (正式的) DPI C layer

Annex I (正式的) svdpi.h

Annex J (正式的) Inclusion of foreign language code

Annex K (正式的) vpi_user.h

Annex L (正式的) sv_compatibility.h

Annex M (正式的) sv_vpi_user.h

Annex N (正式的) 概率分布函数的算法

Annex O (非正式的) 加密/解密 flow

Annex P (非正式的) 术语表

Annex Q (非正式的) 参考书籍