黑金动力社区

FPGA 那些事儿

——Verilog HDL建模设计

杨开陵　徐巧玉　王志慧　王军委　马　瑞　编著

- 模块封装，简化设计
- 实例丰富，注释详与
- 风趣幽默，层次清晰

北京航空航天大学出版社
BEIHANG UNIVERSITY PRESS

源程序下载地址：
http://www.buaapress.com.cn的“下载中心”

FPGA 那些事儿
——Verilog HDL 建模设计

杨开陵　徐巧玉
王志慧　王军委　马　瑞　编著

北京航空航天大学出版社

内 容 简 介

本书总结了使用 Verilog HDL 语言进行 FPGA 设计的方法——建模技巧，语言风趣幽默、层析清晰，力图使读者建立起模块化设计的思想，简化设计。全书共 6 章，第 1 章起到扫盲的作用；第 2 章介绍建模的所有基本单位、应用以及一些需要注意的问题，这是通过 Verilog HDL 语言来设计 FPGA 的基础；第 3 章介绍一些典型实验，如 PS2 解码、UART 串口和 VGA 驱动等；第 4 章讲仿顺序操作，是建模的一个重点，利用建模的优点去弥补 Verilog HDL 在顺序性操作的不足；第 5 章是模块封装，这可用于后期建模；第 6 章是系统建模，使建模模型有一个大体的轮廓和框架，是构建设计的“蓝图”。建模本身的重点就是简化设计，在最大程度上把设计的内容直接地表达出来。

本书适合 Verilog HDL 的初学者、电子相关专业的大学生以及所有对 FPGA 感兴趣的电子工程师。

图书在版编目(CIP)数据

FPGA 那些事儿 ：Verilog HDL 建模设计 / 杨开陵等编著. --北京 ：北京航空航天大学出版社，2013.8

ISBN 978-7-5124-1210-1

Ⅰ.①F… Ⅱ. ①杨… Ⅲ. ①VHDL 语言—程序设计 Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字(2013)第 168003 号

FPGA 那些事儿——Verilog HDL 建模设计

杨开陵　徐巧玉　王志慧　王军委　马　瑞　编著

责任编辑　刘　星

*

北京航空航天大学出版社出版发行

北京市海淀区学院路 37 号(邮编 100191)　http://www.buaapress.com.cn

发行部电话：(010)82317024　传真：(010)82328026

读者信箱：emsbook@gmail.com　邮购电话：(010)82316936

涿州市新华印刷有限公司印装　　各地书店经销

*

开本：710×1 000　1/16　印张：22　字数：469 千字

2013 年 8 月第 1 版　2013 年 8 月第 1 次印刷　印数：3 000 册

ISBN 978-7-5124-1210-1　定价：49.00 元

前　言

一、本书的由来

在学习 Verilog HDL 语言过程中，经常会出现这样的一些人，他们徘徊在学习的边缘，而且心中一直回响着这样的一个问题：

“我在学什么，为什么不管我怎么学，我都没有实感……”

没错，这就是初学 Verilog HDL 语言的心声。

在众多的 Verilog HDL 参考书中，隐隐约约会出现“建模”这样一个词语。建模在 Verilog HDL 的世界里是一个重要的基础，即模块建立的简称。FPGA 的逻辑资源好比乐高的积木，要组合乐高就需要工具，那 Verilog HDL 就是 FPGA 建模的工具。

Verilog HDL 作为 FPGA 建模的一个工具，若是没有技巧地使用它们，则无法很好地发挥它们的优势。读者们，曾几何时有没有为建模的规划而头疼过？有没有为天书般的源码而欲哭无泪？这些心情笔者也拥有过、经历过，这一切的一切都只是一个原因：

“没有建模的技巧……”

网络上常说“学习 Verilog HDL 就是要明白什么是 RTL 级代码，多参考别人写的代码”，但是前提是“你能看懂别人在写什么，别人在设计什么，别人在做什么”。有一句学习 Verilog HDL 的名言“参考别人的代码有如半死不活地受折磨”，当你看懂别人在写什么的时候，估计那时已经身心疲惫，这一切的一切都是：

“没有建模的技巧……”

在这里没有攻击他人的意思，笔者始终觉得一个好的设计不仅是自己看得懂，而且还要别人看得懂，设计的表达能力要直接，代码要整齐，建模有结构。

其实在学习 Verilog HDL 的过程中，有段时间笔者也是经常犯错误，困惑重重，一度有放弃学习的想法，直到偶然间在 ourdev 上看见了 FPGA 黑金开发板，于是重新开始了 Verilog HDL 的学习之旅。笔者告诉自己，不要再犯同样的错误，要找出障碍的原因，于是开始测试许多不同的实验，最后发现到一个关键的东西，那就是“建模”。为了证实自己的想法是对的，就开始在网上分享一些关于 Verilog HDL 建模

技巧的设计笔记。

之后，想法越来越多，建模技巧也越来越成熟。

总结经验，分享经验，慢慢地也就有了现在这本书。

二、本书的内容

很多初学 Verilog HDL 和 FPGA 的朋友成为了之前说过的那群徘徊在边缘的人，主要原因就是他们没有掌握好建模技巧，从而成为了他们继续前进的一大阻碍。在这里笔者将自己养成的建模技巧，在书中称为“初级建模”，分享给大家，希望能让更多初学的朋友越过这一段学习的障碍。

“初级建模”是笔者为建模技巧（建模习惯）所起的名字，读者也可以称其为模块化设计、建模设计等。“建模”就是指利用硬件描述语言去建立某个资源模块，使得 Verilog HDL 语言更形象、更具体。“初级”一词是指最基本的、最简单的。在笔者心中，“初级建模”有“万丈高楼从地起”的意思。在本书中，笔者用心地去记录每一个内容，语言幽默且直接，更有一些生活化的语言和比喻，希望读者能够接受，能够用愉快的心情读完本书。

笔者一直觉得建模技巧作为 Verilog HDL 的基本功，它甚至比时序分析、功能仿真来得更重要，需要得到大家的重视。建模技巧的潜能是难以估计的，笔者一直深信拥有建模技巧的建模，Verilog HDL 语言绝对不会亚于其他高级语言，甚至还可以超越它们。

限于笔者的水平和经验，加之时间比较仓促，疏漏或者错误之处在所难免，敬请读者批评指正。有兴趣的朋友可发送邮件到：akuei_2@yahoo.com，与笔者交流；同时可以在黑金动力社区 http://www.heijin.org 与作者进行互动，图书出版后的勘误、后续进展、相关资源等也会在这里进行更新，希望这里能成为与广大读者交流的平台。

另外，书中的所有源程序也可以在北京航空航天大学出版社网站的“下载中心”下载。

三、读者对象

这本书的适合人群：

- 初学者。
- 希望从另一个角度去了解 Verilog HDL 的人们。

这本书确实有点不适合对 Verilog HDL 一点都没有了解的入门者，读者应该先从一些权威的参考书了解 Verlilog HDL 语言和 FPGA 的基本知识后，再来看这本书。

四、致　谢

这本书可以成功出版，首先要感谢黑金动力社区提供一个平台，感谢版主 avic。笔者是一个住在马来西亚的普通人，没有高学历，而且中文又不好，从最开始收到

EDN China 的邀请开始博客之旅，然后又非常微妙地和 avic 合作一起来出版这本书，这一切都让笔者有着说不出的感动和感激。感激生命线人黄娜，感激开门师傅特权（吴厚航），感激贵人 avic，使笔者能够有机会和广大读者分享自己的经验。同时，还要感谢马胜涛、杨晓美、王卫敏、韦孟辉、陈婷婷、马兴伟、李春艳、陈洪国、李艳在书稿校对过程中所做的工作。最后笔者想说一声 Terima Kasih，这是马来文的谢谢，此外还有接受恩惠的意思。

杨开陵（akuei2）
2013 年 5 月

目 录

第1章

Verilog HDL 扫盲文

每一个人都是宝石，只是忘了如何除去身上的尘土而已。
学习审查自己，一点一点地移掉自身的尘土……

在读者还没有进入重要内容之前，笔者有责任帮大家进行简单地扫盲。扫盲的目的如下：一是更进一步加深读者对 Verilog HDL 语言的认识；二则是可以清楚地表达本书所要讨论的范围。

1.1 两种主流 HDL 语言

很多进入 FPGA 世界不久的朋友，第一个要学习的当然是 HDL 语言，目前较为流行的有 Verilog HDL 和 VHDL 两种语言。如果读者是 VHDL 语言的爱好者，且无打算学习 Verilog HDL 语言，那么这本书并不是您的最好选择。在笔者的眼中，VHDL 稍显死板，但并不是说它不好，只是笔者嫌它麻烦而已；反之，Verilog 却像是一个活泼且爱捣乱的小男孩，思想很简单却很俏皮，很难琢磨。

网上有一个很常见的问题："学习 VHDL 好还是学习 Verilog HDL 好"？很多东西都是仁者见仁，智者见智。如果要笔者来回答，自然会推荐学习 Verilog HDL，因为它很有趣，也很好玩。

为什么本书选择 Verilog HDL 语言？

这个问题笔者也很难回答，当时学习的时候并没有考虑那么多，但是后来发现 Verilog HDL 语言有太多的潜能了，笔者不小心就陷入研究它的"陷阱"了。Verilog HDL 语言的语法和格式都比较随意，它不像 VHDL 语言那么严谨，可能这就是选择它的原因吧。事实上，选择 VHDL 语言也好，选择 Verilog HDL 语言也好，都是"萝卜青菜，各有所爱"。笔者自身不喜欢受限制太多，故和 Verilog HDL 语言意气相投，所以最终还是选择了它。

1.2 HDL 语言的层次

有一个很好笑的话题，老师常常都说 HDL 语言的层次是处于汇编语言与 C 语

言之间。假设汇编语言是初级语言，C 语言是高级语言，那么 HDL 语言岂不是不上又不下？这似乎有些不可思议，其实，我们没必要为这个话题浪费太多时间。HDL 语言的英文全称是 Hardware Description Language，中文译名就是硬件描述语言。事实上，无论是汇编语言还是 C 语言，它们的作用就是用来控制处理器；而 HDL 语言的作用则是用来建立一个硬件的模块。

打个比方，假设有一个 C51 单片机的串口硬件，那么既可以使用汇编语言去控制它，也可以使用 C 语言去控制它。但是站在 HDL 语言的角度上，则可以建立一个受控制的串口硬件模块，也可以建立一个不受控制（即自动）的串口硬件模块，当然还可以用 HDL 去驱动串口硬件模块。

从这一点就可以看出，如果归纳 HDL 语言和汇编语言、C 语言分别处于哪个层次的话，其实它们基本上是两种不同的对“层次”的理解和角度，那又何必拿它们来做比较呢？但在有些标准上，HDL 语言却既是硬件语言又是初级语言（凡是涉及硬件的通通都被打入初级语言的“冷宫”）。总而言之，HDL 语言的层次很“暧昧”就是了。

实际上，确实有不同层次级的 HDL 语言，如 SystemVerilog 或者 SystemC，它们都是系统级的 HDL 语言。相比之下，Verilog HDL 语言和 VHDL 语言的层次都称为模块级。但是这些层次的区分一点也不重要，只要把 Verilog HDL 语言掌握得好，那读者什么层次都可以实现。

1.3 RTL 级和组合逻辑级

虽说 Verilog HDL 语言是用来描述硬件的，但它的应用范围远不只硬件描述这么简单。从时钟源这个角度来讲，Verilog HDL 语言建立的硬件模块可以分为有时钟源的模块和无时钟源的模块。

对于有时钟源的模块，以时钟信号作为衡量或控制操作是否被执行的基本依据，也就是说，通过时钟节拍来控制硬件模块的执行。而无时钟源的模块，不需要消耗时钟信号，硬件模块也可以被执行。在一些图书介绍中，有时钟源的模块就是常说的时序逻辑，而无时钟源的模块对应的则是组合逻辑。

由此就有了时序逻辑级和组合逻辑级的建模之分。概括地讲，时序逻辑级建模的基本单元是寄存器，组合逻辑级建模的基本单元是逻辑门。单单用几行文字来讲述和理解时序逻辑级和组合逻辑级的建模实在太抽象了，笔者举个简单的例子：

```
module Add_Module( input [7:0]A, input [7:0]B, input [7:0]C, Output[15:0]);
    wire [7:0]_a = A;
    wire [7:0]_b = B;
    wire [7:0]_c = C;
    assign Output = _a + _b + _c;
endmodule
```

这是一个简单的组合逻辑级建模而成的 3 路加法器，大家可以看到，里面并没有

时钟信号。再看如下实例：

```
module Add_module
( input CLK, input RSTn, input [7:0]A, input [7:0]B, input [7:0]C, Output[15:0]) ;
    reg [15:0]rTemp;
    alawys @ ( posedge CLK negedge RSTn )
    if( ! RSTn )
        rTemp <= 4'0;
    else
        rTemp <= A + B + C;
    assign Output = rTemp;
endmodule
```

这是一个简单的时序逻辑级建模而成的 3 路加法器，可以看到其中有时钟信号 CLK 的身影。

上面 2 个 3 路加法器的例子，一个是由组合逻辑级建模而成，另外一个是由时序逻辑级建模而成。组合逻辑级建模给人最直接的印象就是模块都不带时钟信号；反之，时序逻辑级建模的最大特征，就是模块都会伴随时钟信号。当然，笔者不可能仅以时钟信号来区分组合逻辑级建模和时序逻辑级建模。

笔者认为，在实际的学习中过于在意区分“什么是组合逻辑级建模，什么是时序逻辑级建模”，对学习没有太大帮助(虽然有些参考书很注重区分)，凡是有关 Verilog HDL 语言的建模，在进行硬件模块的建立时可能都需要用到。话虽如此，但在这本书里笔者还是偏重于时序逻辑级的建模。

1.4 Verilog HDL 语言真得那么难掌握

Verilog HDL 语言容易入门但是不容易掌握，估计这是所有 Verilog HDL 学习者的心声。其实要掌握 Verilog HDL 语言是很简单的，笔者的秘诀就是：掌握 Verilog HDL 语言的思想——建模和时序。建模是 Verilog HDL 语言的结构或者说是地基，而时序是所有模块的活动记录。这些概念，读者目前不明白也不要紧，笔者还会在后续内容中详细讲述。

很多朋友在接触 FPGA 之前都接触过单片机，可能在不知不觉中都有利用学习单片机的方式来学习 FPGA 的想法。笔者也是过来人，对这样的感受非常了解。

在学习单片机的时候，很多人估计都是直接以 C 语言入门。笔者就先说说单片机这个硬件：它们都是各大厂商的产品，一些基本的硬件资源嵌入在这个小小的产品当中。学习单片机，其实质是在学习如何控制单片机里面的各种寄存器。对于一些硬件动作的发生，使用者可以完全不用知道，只要懂得如何配置控制该硬件的寄存器就可以了。

相比之下，如果读者要通过 FPGA 来实现串口，那就要自行建立串口硬件模块。因为，FPGA 不像单片机那样，其内部并没有做好的寄存器等硬件资源，FPGA 可以

称为是赤裸裸的乐高积木。要建立一个串口硬件模块，首先需要明白串口的操作原理，然后根据需要，通过自定义方式或在修改串口硬件动作发生原理的基础上描述该模块。最后还要考虑这个串口硬件模块如何被控制，是否对它进行独立化（见第 5 章接口建模）等，这些就是建模思路。要实现这样的建模，也需要通过某种语言，比如 Verilog HDL 语言。

Verilog HDL 语言虽然有语法，但是没有自身的结构，也没有自己相对标准化或明确规定的一套用法。在网上这种现象很普遍，读者会看到五花八门、百花齐放、各种各样的模块内容。很多时候，这些模块的内容恐怕只有设计者自己能看懂，别人估计要花上几倍的精力才能搞明白“它在描述什么”。

为此，如果要把 Verilog HDL 语言掌握好、用好，第一要提供 Verilog HDL 语言的结构，第二要建立 Verilog HDL 语言的一套用法（使用规则）。对于前者，笔者是用建模来解决的；后者，笔者用仿顺序操作来实现。

1.5 高级语言和 Verilog HDL 语言的区别

在一些高级语言（如 C 语言）中，组成操作行为的基本单位是步骤，举个例子：

```
步骤 1 Sum1 = a + b + c;
步骤 2 Sum2 = d + e + f;
```

步骤 1，将 a+b+c 的结果赋予 Sum1；步骤 2，将 d+e+f 的结果赋予 Sum2。在使用者眼中，C 语言只要两个步骤就可以完成操作。但是，使用者看不见的是，步骤 1 和步骤 2 总和所生成的指令估计会超过 8 个，而单片机一个时钟只能执行一个指令，故完成步骤 1 和步骤 2 至少需要超过 8 个时钟。在用 C 语言编写代码时，使用者关心的主要是通过什么步骤可以实现想要的操作，而不会过于关心“一共被执行了多少指令”和“消耗了多少个时钟周期”。

然而，对于 Verilog HDL 语言来说，尤其是 RTL 级的建模，时钟代表了模块执行所要消耗的单元。看一个例子：

```
case( i )
    0:
    begin Sum1 <= a + b + c; Sum2 <= d + e + f; i <= i + 1'b1; end
```

在一个时钟之内 a+b+c 与 d+e+f 同时赋值给 Sum1 和 Sum2。

再看以下实例：

```
case( i )
    0:
    begin Sum1 <= a + b + c; i <= i + 1'b1; end
    1:
    begin Sum2 <= d + e + f; i <= i + 1'b1; end
```

在第一个时钟之内，a+b+c 赋值给 Sum1；在第二个时钟之内，d+e+f 赋值

给 Sum2。

通过上面两段代码可以发现，Sum1 和 Sum2 可以在一个时钟周期内求得，也可以分开两个时钟周期完成赋值。换句话说，理论上 Verilog HDL 语言在一个时钟之内既可以完成“很多很多”的操作，又可以只完成单一操作。因为它的操作是由逻辑资源组成的，只要 FPGA 的逻辑资源允许，读者要在一个时钟内完成多少个操作都没有问题。这个事实也从某种角度体现了 Verilog HDL 语言是并行的这一性质。

表 1.5.1 总结了 C 语言和 Verilog HDL 语言之间的区别。有关高级语言和 Verilog HDL 语言的区别，就讨论到这里，建议读者不要过于深入区分，如此纠结对学习没有什么好处，更多的认识和体会，当读者深入学习以后，自然就会了解。

表 1.5.1　C 语言和 Verilog HDL 语言之间的区别

比较属性	C 语言	Verilog HDL 语言
最小单位	指令	逻辑资源
单个时钟可以执行的操作	一个指令(简单指令)	理论上是无限个操作(并行操作的性质)
结构	有固定的结构	没有固定结构
使用规则	有固定的使用规则	没有规定使用规则
语法的约束强度	软	偏硬

1.6　什么是 Verilog HDL 语言的时序

如前所述，时序是 Verilog HDL 语言的中心思想之二。有些书上将时序图等效于逻辑波形图，在某种程度上这种解释没有任何错误。但是，这种解释有些片面了。时序，可以称为是模块的活动记录，又或者说：把每一个时钟周期的模块活动记录都链接起来的话，就可以形成俗称的波形图。

从另一个角度看，在长长的时序图里，还包含了模块的活动规则、沟通记录等一些重要的信息。对于 Verilog HDL 语言中的时序概念，初学者确实很难掌握。因为在传统的印象中，时序给人的感觉是既有延时又存在亚稳态现象等。虽然这也是关于时序的一种解释，但这是发生在物理上的。在 Verilog HDL 语言中的时序是理想化的概念，不存在任何瑕疵。我们知道，驱使(RTL 级)模块行动的最小单位就是时钟，即使发生延时也只是延时几个时钟周期，绝对不会出现类似于延时 2～3 ns（物理路径延时）等问题。

1.7　Verilog HDL 的综合语言

Verilog HDL 语言有两个部分，即综合语言和验证语言。本书不涉及验证语言的内容，至于综合语言，因为是建模最常用的，笔者习惯称之为建模语言。

综合语言常用的关键字不多，如表 1.7.1 所列。

表 1.7.1 综合语言常用的关键字举例

属 性	关键字
模块建立	module…endmodule
输入/输出声明	input,output,inout
判断	if…elseif…else,case()…endcase
资源声明	reg,wire
基本操作	always@(),posedge,negedge,assign,begin…end
常量声明	parameter

至于综合语言常用的操作符也不多，如表 1.7.2 所列。

表 1.7.2 综合语言常用的操作符举例

属 性	操作符	属 性	操作符
赋值	<= , =	位操作	{}, <<, >>
逻辑判断	> , < , <= , >=, ! = , ==	数学运算	+, −, * ,/, %
下标	[]	其他	?:(三目), *(声明组合逻辑用)

当读者看了表 1.7.1 和表 1.7.2 给出的综合语言以后，是不是很惊讶？难道就这么简单？不要怀疑，只要懂这些东西就足够了。先谈谈几个比较有趣的关键字和操作符。

1.7.1 reg 和 wire 的尴尬

如何区分 reg 和 wire？如果站在 RTL 级建模的角度上，reg 就是寄存器，用来暂存内容并提供操作空间；wire 就是连线，作用仅此而已。但是站在组合逻辑级建模的角度，reg 和 wire 就让人尴尬地分不清楚了，举个例子：

```
module omg_module ( output [7:0]Q );
wire [3:0]A = 4'd0;
wire [3:0]B = 4'd2;
wire [3:0]C = 4'd3;
assign Q = A + B + C;
endmodule
```

以上的一段代码，请问 wire 的作用是连线还是寄存内容？这样的使用方法没有错。在这里只是给出一个有趣的例子而已，对于一些初学者来说可能会非常疑惑，尤其是那些习惯“面向对象”思维的读者。

如果换成以下写法的话，是不是觉得更有道理呢？

```
module omg_module ( output [7:0]Q );
reg [3:0]A;
```

```
reg [3:0]B;
reg [3:0]C;
always @ ( * )
    begin
        A = 4'd0;
        B = 4'd2;
        C = 4'd3;
    end
assign Q = A + B + C;
endmodule
```

always (*)的使用暗示了这是一个组合逻辑，寄存器 A 的值是 4'd0，寄存器 B 的值是 4'd2，寄存器 C 的值是 4'd3，Q 的驱动输出是寄存器 A、B、C 值的综合。

更有趣的是，经过编译，两段代码所生成的 RTL 视图都一样，如图 1.7.1 所示。在这里只是想通过这个例子告诉读者一个信息，在建模时，解读能力的优先级往往要高过内容精简性。也就是说，宁可多写几行代码以增强代码的解读能力，也不要为了精简而少写几行代码。虽然这两段代码都没有错，而且结果一致，但有一个潜在的问题是，人们一般都习惯把东西分类以便管理和理解，既然 wire 在字面上就是连线的意思，那干脆就把它当连线来使用。

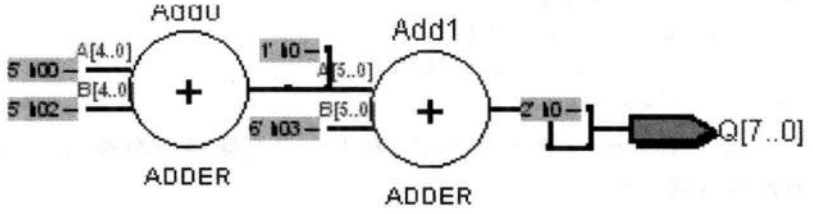

图 1.7.1 生成的 RTL 视图

1.7.2 always @ ()的多样性

always @ ()的用法很多，但是用得最多的是以下的第一种和第二种：

```
1. always @ (posedge CLK or negedge RSTn) //当 CLK 和 RSTn 变化的时候
2. always @ ( * )  //什么时候都变化，即默认为组合逻辑
3. always @ ( A )  //当 A 变化的时候
```

第一种用法，表示了在 always @ ()语句之下的动作，对每一个 CLK 的上升沿或者 RSTn 的下降沿都被执行。给出一个比较简单的例子：

```
always @ ( posedge CLK or negedge RSTn )
if( ! RSTn )
    Counter <= 8'd0;
    else
    Counter <= Counter + 1'b1;
```

这是一个简单的计数器实例，当 RSTn 产生下降沿 Counter 就清零，反之在每一