高等学校电子信息类专业
“十三五”规划教材

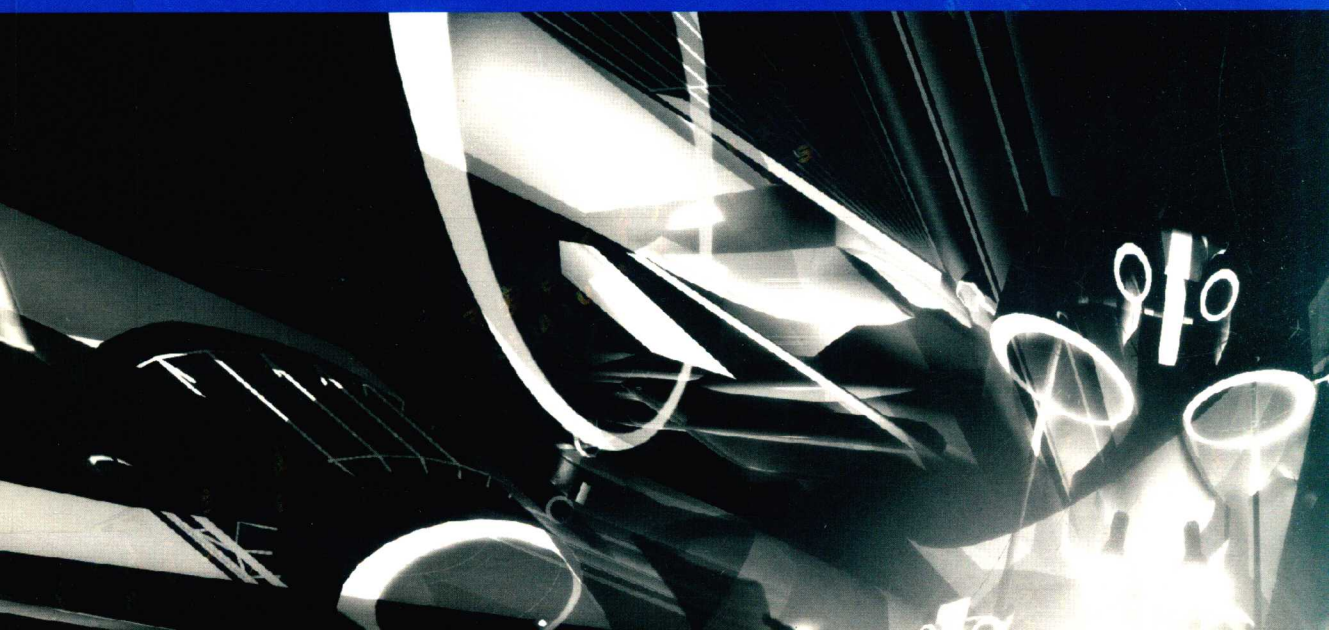
ELECTRONIC
INFORMATION SPECIALTY

Verilog程序设计与EDA

（第二版）

刘　靳　刘笃仁　编著

西安电子科技大学出版社
http://www.xduph.com



高等学校电子信息类专业“十三五”规划教材

Verilog 程序设计与 EDA

(第二版)

刘　靳　刘笃仁　编著

西安电子科技大学出版社

内 容 简 介

本书除绪论外共分 9 章，主要内容包括：Verilog HDL 的基本结构与描述方式、Verilo HDL 的基本要素、Verilog HDL 的基本语句、组合电路设计、时序电路设计、仿真测试程序设计、组合电路设计实例、时序电路设计实例、EDA 开发软件等。书中选用了相当数量的例题、实例，便于读者联系实际，举一反三，学习运用。

本书可作为高等学校通信、电子工程、自动控制、工业自动化、检测技术及电子技术应用等相关电类专业本科和专科生 Verilog HDL、EDA 课程的教材和教学参考书，也可作为相关工程技术人员的学习参考书。

图书在版编目(CIP)数据

Verilog 程序设计与 EDA / 刘靳，刘笃仁编著. —2 版. —西安：西安电子科技大学出版社，2018.12
高等学校电子信息类专业“十三五”规划教材
ISBN 978-7-5606-4924-5

Ⅰ. ① V… Ⅱ. ① 刘… ② 刘… Ⅲ. ① VHDL 语言—程序设计—高等学校—教材 ② 电子电路—电路设计—计算机辅助设计—高等学校—教材 ③ EDA Ⅳ. ① TP312 ② TN702

中国版本图书馆 CIP 数据核字(2018)第 266184 号

策划编辑 云立实
责任编辑 云立实
出版发行 西安电子科技大学出版社(西安市太白南路 2 号)
电 话 (029)88242885 88201467 邮 编 710071
网 址 www.xduph.com 电子邮箱 xdupfxb001@163.com
经 销 新华书店
印刷单位 陕西天意印务有限责任公司
版 次 2018 年 12 月第 2 版 2018 年 12 月第 2 次印刷
开 本 787 毫米×1092 毫米 1/16 印 张 15
字 数 351 千字
印 数 3001～5000 册
定 价 34.00 元
ISBN 978-7-5606-4924-5 / TP

XDUP 5226002-2

前　言

本书是在原《Verilog 程序设计与 EDA》一书(西安电子科技大学出版社 2012 年 9 月出版)的基础上，根据教学改革的需要，结合作者近年来 Verilog 程序设计与 EDA 课程教学、科研实践重新修订而成的。

此次修订在内容上进行了调整，更便于组织教学。Verilog 程序设计与 EDA 课程是实践性很强的课程，在完成前几章的教学任务后，教师可根据实际，选择本书后相关公司的开发软件，以组合电路设计实例、时序电路设计实例章节为参考，组织实验教学内容，以便在学习中实践，在实践中学习。

本书此次出版得到了西安电子科技大学教材建设基金的资助。

本书的编著得到了西安电子科技大学教务处、电子工程学院领导和同行的支持和帮助；西安电子科技大学出版社云立实编辑为之付出了辛勤工作。对于这一切，编者表示诚挚的感谢。此外，对于在学习、写作过程中参考过的文献、资料的作者，编者也深表感谢。

由于作者水平有限，书中难免存在疏漏和不妥之处，敬请各位读者批评指正。

作　者

2018 年 3 月

第一版前言

Verilog HDL 是当今国际上流行的一种硬件描述语言，用于从算法级、门级到开关级的多种抽象设计层次的数字电路与系统建模。Verilog HDL 语言不仅定义了语法，而且对每个语法结构都定义了清晰的模拟、仿真语义。因此，用这种语言编写的模块能够方便地进行仿真、验证。该语言从 C 语言中继承了多种操作符和结构，具有明显的优势。

本书是作者从事 Verilog HDL 程序设计与 EDA 应用及教学的经验总结，主要内容包括：Verilog HDL 的基本要素、Verilog HDL 的基本语句、组合电路设计、时序电路设计、仿真测试程序设计、组合电路设计实例、时序电路设计实例及 EDA 开发软件等。

本书从初学者的角度出发，循序渐进地介绍了 Verilog HDL 程序设计与 EDA 应用，将基本概念、难点疑点分散到各章、各节，甚至贯穿于例题、实例的注释中，以便读者轻松学习，逐步消化。本书相关内容来自教学与科研实践，能使读者快速掌握 Verilog HDL 语言的实质，并能应用其设计实现各种现代数字电路与系统。

书中选用了相当数量的例题、实例，其中还有不少一题多种思路的设计，这种一题多解的方式是为了启发读者从多角度、多思路思考问题。有些看似简洁的程序，有时并不能被最简单的 CPLD 或 FPGA 电路实现，反之亦然。实际中可根据 EDA 开发软件反馈的设计信息或资源利用报告进行分析、确认。

本书的编写得到了西安电子科技大学电子工程学院领导、同行以及西安培华学院电气信息工程学院领导的支持和帮助；西安电子科技大学出版社云立实编辑为之付出了辛勤工作。对于这一切，编者表示诚挚的感谢。此外，对于在学习、写作过程中参考过的文献、资料的作者，编者也深表感谢。

由于编者水平有限，书中难免存在疏漏和不妥之处，敬请各位读者批评指正。

编　者

2012 年 3 月

目　录

绪　论

随着科学技术的发展，硬件电路与系统的设计发生了新的变革，现代硬件电路与系统的设计蓬勃兴起。依据先进的设计思想，利用现代化的设计手段，使用可编程的新型器件来设计现代电路与系统已成为一种趋势。

先进的设计思想是自上而下的设计思想，即先设计顶层、再设计底层。这种一气呵成的设计思想只能借助于现代化的设计手段实现。即应用计算机技术将硬件电路与系统设计要做的许多工作用软件设计来完成，这就是电子设计自动化 EDA(Electronic Design Automation)，也就是硬件设计软件化。要应用计算机做到硬件设计软件化，必须要有计算机和设计者都能识别的硬件描述语言 HDL(Hardware Description Language)及其综合、编译、仿真、下载的 EDA 开发软件。

可编程的新型器件是实现各种现代硬件电路与系统的载体、实体。可编程器件包括可编程逻辑器件 PLD(Programmable Logic Device)、复杂可编程逻辑器件 CPLD(Complex Programmable Logic Device)、现场可编程门阵列 FPGA(Field Programmable Gate Array)、可编程模拟器件 PAD(Programmable Analog Device)、可编程模拟电路 PAC(Programmable Analog Circuit)、通用数字开关 GDS(Generic Digital Switch)、通用数字交叉 GDX(Generic Digital Cross)器件、可编程智能混合器件 SmartFusion 等。

总而言之，现代硬件电路与系统的最终实现，不仅需要先进的设计思想和理念，还需要现代化的设计手段——计算机平台和硬件描述语言程序设计、可编程的新型器件和功能设计工具(EDA 软件开发工具)、连接计算机与可编程器件的下载接口 JTAG 和编程电缆。

本书首先讨论 Verilog 程序设计，接着讨论与 EDA 有关的可编程器件功能设计工具(软件开发工具)应用。

0.1　关于 Verilog HDL

Verilog 是一种用于数字电路与系统设计的硬件描述语言，可以从算法级、门级到开关级的多种抽象设计层次对数字系统建模。使用 Verilog HDL，用户能灵活地进行各种级别的逻辑设计，方便地进行数字逻辑系统的仿真验证、时序分析和逻辑综合。

Verilog 是 1983 年由 GDA(Gateway Design Automation)公司的 Philip R.Moorby 为其模拟器产品开发的硬件建模语言，那时它只是一种专用语言。Philip R.Moorby 本人后来成为 Verilog-XL 的主要设计者和 Cadence(Cadence Design System)公司的第一合伙人。其间，他设计了第一个关于 Verilog-XL 的仿真器，并推出了用于快速门级仿真的 XL 算法。1989 年，

Cadence 公司收购了 GDA 公司，Verilog 归 Cadence 公司所有。两年后，该公司成立了 OVI(Open Verilog International)，决定致力于推广 Verilog OVI 标准，使其成为 IEEE 标准。这一努力最后获得成功，Verilog 语言于 1995 年成为 IEEE 标准，称为 IEEE Std 1364-1995。从此，Verilog HDL 成为一种极具竞争力的用于数字电路与系统设计的硬件描述语言。通过不断实践，Verilog HDL 又增加了 Verilog-2001(称为 IEEE Std 1364-2001)、Verilog-2005(称为 IEEE Std 1364-2005)，使其使用更加方便。

我国《集成电路/计算机硬件描述语言 Verilog》(国家标准编号为 GB/T18349—2001)于 2001 年 10 月 1 日已正式实施。

Verilog HDL 是一种简洁清晰、功能强大、容易掌握、便于学习的硬件描述语言，只要有 C 语言的编程基础，在了解了 Verilog HDL 的基本结构与描述方式、基本要素、基本句法等以后，再辅助上机操作，就能很快掌握 Verilog HDL 程序设计技术。

0.2　关于 EDA

EDA 是电子设计自动化(Electronic Design Automation)的缩写，它是在 20 世纪 80 年代初从计算机辅助设计 CAD(Computer Aided Design)、计算机辅助制造 CAM(Computer Aided Manufacturing)、计算机辅助测试 CAT(Computer Aided Test)和计算机辅助工程 CAE(Computer Aided Engineering)的概念发展而来的。

可编程器件的诞生，使现代硬件电路与系统的电子设计技术发生了革命性的变化。现代硬件电路与系统的 EDA 技术以计算机为工具，设计者在 EDA 开发软件平台上，用硬件描述语言 HDL 完成硬件电路的源文件设计，然后通过计算机自动地完成编译、化简、分割、综合、优化、布局、布线和仿真，直至完成对选定的目标芯片(可编程器件)的适配、逻辑映射、布局布线和下载等工作。EDA 技术的出现，极大地提高了电路设计的效率和可靠性，有效地减轻了设计者的劳动强度，缩短了产品的设计周期，加快了产品的上市时间。从 20 世纪 70 年代起，国际上电子和计算机技术较先进的国家，一直在积极探索新的电子电路设计方法，并在设计方法、集成器件、开发工具等方面进行了彻底的变革，取得了巨大成功。在电子技术设计领域，可编程逻辑器件(如 CPLD、FPGA)的应用，已得到广泛的普及，这些器件为数字系统的设计带来了极大的灵活性。可编程模拟器件、可编程数字开关及互联器件(如 ispPAC、ispGDS、ispGDX)的研制成功，为各种现代电路与系统硬件的设计注入了新的活力。这些器件可以通过软件编程对其硬件结构和工作方式进行重构、重组态，从而使得硬件的设计可以如同软件设计那样方便快捷。这一切极大地改变了甚至是颠覆了传统的电路与系统设计方法、设计过程和设计观念，促进了 EDA 技术的迅速发展。

现代 EDA 的应用相当广泛，包括机械、电子、通信、航空航天、航海、交通、化工、矿产、生物、医学、军事等各个领域。例如在飞机、汽车制造过程中，从设计、性能测试及特性分析直到飞机试飞、汽车试车模拟，都可能涉及 EDA 技术。

本书所指的 EDA 技术，是针对电子电路与系统设计的电子设计自动化，且主要讨论现代数字电路与系统的硬件技术。

第 1 章　Verilog HDL 的基本结构与描述方式

1.1　基 本 结 构

模块是 Verilog 的基本描述单位。

模块用于描述某个电路设计的功能或结构以及与其他模块通信的外部端口。每个模块与其他模块之间通过端口联系，模块自身通过端口与其描述的内容联系。

一个硬件电路用 Verilog 设计的基本结构组成一个模块，它是实现硬件电路的源文件。

一个硬件电路用 Verilog 程序设计的过程叫做编写源文件或建模。

一个模块又可以在另一个模块中使用。

图 1.1 是一位全加器的电路。该电路的函数表达式为

$$sum=a \oplus b \oplus cin$$

$$cout=(a \oplus b)cin+ab$$

其中：a、b 分别为两个一位二进制的加数、被加数；cin 为低位向本位的进位；sum 为本位和；cout 是本位向高位的进位。

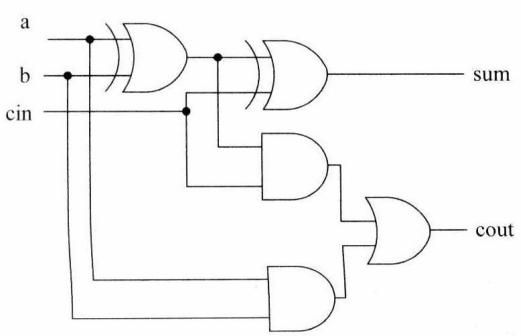


图 1.1　一位全加器的电路

当一个完成某一功能的电路图或函数表达式已知时，可依据电路图或函数表达式用 Verilog 设计其源文件。

例 1.1　下面给出了一个简单的 Verilog 源文件的基本结构。它是一个带进位的一位二进制加法器电路的源文件设计实例。

```
module Fulladder(sum, cout, a, b, cin);
    input cin;
    input a, b;
    output cout;
```

```
    output sum;
    assign sum=(a^b)^cin;
    assign cout=((a^b)&cin)|(a&b);
endmodule
```

其中，符号 ^、&、| 在 Verilog 中分别表示异或、与、或。

当要设计的某一电路功能简单、关系清楚时，可直接用 Verilog 相关语句设计源文件。

例 1.2　下面给出一个简单的 Verilog 源文件的基本结构。它是一个带进位的 4 位二进制加法器电路的源文件设计实例。

```
module Fulladder4(sum, cout, a, b, cin);
    input cin;
    input [3:0] a, b;          //[3:0]是简略写法，表示“a3, a2, a1, a0”、“b3, b2, b1, b0”
    output cout;
    output [3:0] sum;          //[3:0]是简略写法，表示“sum3, sum2, sum1, sum0”
    assign {cout, sum}=a+b+cin;        //{,}是 Verilog 中的连接符
endmodule
```

当要求电路完成某一特定功能时，可通过分析其工作过程，直接用 Verilog 相关语句设计源文件。

例 1.3　下面给出一个稍复杂的 Verilog 源文件的基本结构。它是一个具有两个 3 位二进制乘法器电路的源文件设计实例。此例的设计思路具有一定难度，等学习深入后就会理解。

设被乘数 a[2:0]={a2, a1, a0}，乘数 b[2:0]={b2, b1, b0}，乘积为{p5, p4, p3, p2, p1, p0}。分析乘积过程如下：

			a2	a1	a0
	×		b2	b1	b0
			a2b0	a1b0	a0b0
		a2b1	a1b1	a0b1	
	a2b2	a1b2	a0b2		
p5	p4	p3	p2	p1	p0

```
module plus3(a2, a1, a0, b2, b1, b0, p5, p4, p3, p2, p1, p0);
input a2, a1, a0, b2, b1, b0;
output p5, p4, p3, p2, p1, p0;
reg p5, p4, p3, p2, p1, p0;
reg[5:0] result;
reg[2:0] a, b;
integer bindex;
always@(a2 or a1 or a0 or b2 or b1 or b0)
    begin
        a={a2, a1, a0};
        b={b2, b1, b0};
```

```
    result=0;
    for(bindex=0; bindex<3; bindex=bindex+1)
            if(b[bindex])
        result=result+(a<<bindex);
    {p5, p4, p3, p2, p1, p0}=result;
end
endmodule
```

通过上面的三个例子，我们初步看到了 Verilog 实现硬件电路的源文件，即模块的基本结构：

(1) Verilog 程序是由模块构成的。每个模块中包含不同的内容，这些内容被安排在 module 和 endmodule 两个关键字之间。

(2) 每一个模块完成一个特定的功能。模块是可以进行层次嵌套的。一个大型的数字电路与系统可以分割成若干个实现一定功能的子模块，然后通过顶层模块调用子模块，完成系统的整体功能。

(3) 在每一个模块中要对端口进行定义，说明输入、输出口，并对模块的功能进行逻辑描述。

(4) 模块是 Verilog 的基本设计单位。一个模块由两部分组成：一部分描述接口；另一部分描述逻辑功能。这种描述实际上就是说明输入是如何影响输出的。

下面给出了一个通用的模块或源文件的基本结构。

```
module module name(port list);
Declarations:
input, output, inout;
reg, wire, parameter;
function, task, …;
Statements:
initial statement;
always statement;
module instantiation;
Gate instantiation;
UDP instantiation;
Continuous assignment;
endmodule
```

在一个程序模块中，第一行是模块的端口(或叫接口)定义，其格式为

```
module　模块名(端口 1，端口 2，端口 3，…);
```

它声明了模块的输入、输出端口。

接下来的 Declarations 说明部分用于定义不同的项，例如模块 I/O(输入/输出)的信号流向，哪些是输入、哪些是输出；定义描述中使用的数据类型、寄存器和参数等。I/O 说明的格式为

```
input　输入端口名 1，输入端口名 2，…，输入端口名 N;
```

output　输出端口名 1，输出端口名 2，…输出端口名 N;

当它们有规律地排列时，可使用简略写法。如上述 4 位二进制加法器电路源代码中第三行、第五行和 3 位二进制乘法器电路源代码中第五行、第六行那样。

我们也可以把 I/O 说明直接放在端口定义语句里，其格式为

module module name(输入端口名 1，输入端口名 2，…，输入端口名 N，输出端口名 1，输出端口名 2，…，输出端口名 N);

Statements 语句的定义部分则用于定义设计的功能和结构，它是模块的核心。

说明部分和语句定义部分可以散布在模块中的任何地方，但是变量、寄存器、线网和参数等的说明部分必须在使用前出现。

为了使模块描述清晰和具有良好的可读性，最好将所有的说明部分放在语句前。

1.2 描 述 方 式

在设计的源文件即模块中，可以采用下述四种描述方式：

(1) 数据流描述方式。

(2) 行为描述方式。

(3) 结构化描述方式。

(4) 混合描述方式(前三种方式的混合)。

下面通过实例分别说明这四种描述方式。

1.2.1 数据流描述方式

用数据流描述方式对要求实现的硬件电路建模或设计源文件，最基本的方法就是使用连续赋值语句。在连续赋值语句中，若干个值(或数据)被指定为线网变量。线网变量是数据类型的一种。

连续赋值语句的格式为

assign [delay] LHS_net=RHS_expression;

等号右边表达式中的操作数无论在什么时候发生变化，该表达式都会重新计算，并在设定的时延[delay]后将变化了的值赋予等号左边的线网变量。

时延[delay]定义了等号右边表达式中的操作数变化与赋值给等号左边之间的持续时间。[delay]是可选的，如果没有定义它的值，缺省时延为 0。

例 1.4　用数据流描述方式，对如图 1.2 所示一位全减器电路建模(编写源文件)。

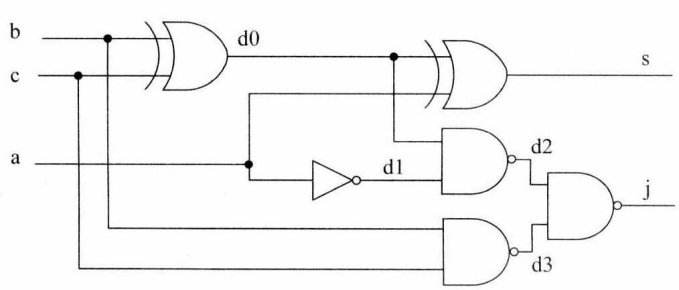


图 1.2　一位全减器电路

该一位全减器的输入为 a、b、c，输出 s 代表差，j 代表向高位借位，c 代表低位向本位借位。其真值表如表 1.1 所示。

表 1.1　一位全减器真值表

a	b	c	s	j
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

下面给出了用数据流描述方式对图 1.2 所示一位全减器电路编写的源文件。

```
module ywqj(s, j, a, b, c);
input a, b, c;
output s, j;
wire d0, d1, d2, d3;
assign d0=c^b;
assign s=d0^a;
assign d1=~a;
assign d2=~(d0&d1);
assign d3=~(b&c);
assign j=~(d2&d3);
endmodule
```

上述模块中，线网类型连线(wire)说明了 4 个连线型变量 d0、d1、d2、d3(连线型类型是线网类型的一种)。模块中包含了 6 个连续赋值语句。

连续赋值语句是并发执行的，与语句的先后顺序无关。

一位全减器电路设计的另一种简单思路如例 1.5 所示。

例 1.5

```
module suber1(a, b, c, s, j);
    input a, b;
    input c;
    output s;
    output j;
    assign {j, s}=a-b-c;        /*有效利用了连接符{ , }，将用逗号分隔的向高位借位 j 与差 s
                                  按位连接在一起*/
endmodule
```

例 1.6　用数据流描述方式，对七段译码器建模(编写源文件)。

输入 A、B、C 经译码器使输出 a1、b1、c1、d1、e1、f1、g1 驱动七段数码管，显示相应的数字 0、1、2、3、4、5、6、7。七段数码管各段的定义如图 1.3 所示，其真值表如表 1.2 所示。

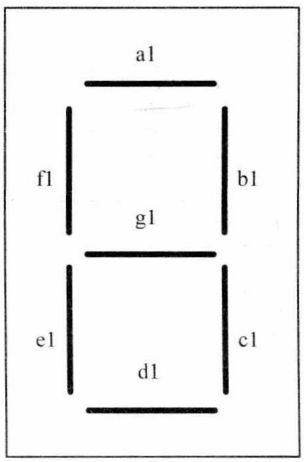


图 1.3　七段数码管各段的定义

表 1.2　七段数码管显示真值表

A	B	C	a1	b1	c1	d1	e1	f1	g1	显示
0	0	0	1	1	1	1	1	1	0	0
0	0	1	0	1	1	0	0	0	0	1
0	1	0	1	1	0	1	1	0	1	2
0	1	1	1	1	1	1	0	0	1	3
1	0	0	0	1	1	0	0	1	1	4
1	0	1	1	0	1	1	0	1	1	5
1	1	0	1	0	1	1	1	1	1	6
1	1	1	1	1	1	0	0	0	0	7

根据真值表，可写出七段数码管译码器的逻辑方程：

```
a1=(~A&~B&~C) | (~A&B&~C) | (~A&B&C) | (A&~B&C) | (A&B&~C) | (A&B&C);
b1=(~A&~B&~C) | (~A&~B&C) | (~A&B&~C) | (~A&B&C) | (A&~B&~C) | (A&B&C);
c1=(~A&~B&~C) | (~A&~B&C) | (~A&B&C) | (A&~B&~C) | (A&~B&C) | (A&B&~C) | (A&B&C);
d1=(~A&~B&~C) | (~A&B&~C) | (~A&B&C) | (A&~B&C) | (A&B&~C);
e1=(~A&~B&~C) | (~A&B&~C) | (A&B&~C);
f1=(~A&~B&~C) | (A&~B&~C) | (A&~B&C) | (A&B&~C);
g1=(~A&B&~C) | (~A&B&C) | (A&~B&~C) | (A&~B&C) | (A&B&~C);
```

下面给出了用数据流描述方式对七段数码管译码器编写的源文件。

```
module ymq(A, B, C, a1, b1, c1, d1, e1, f1, g1);
input A, B, C;
output a1, b1, c1, d1, e1, f1, g1;
wire a1, b1, c1, d1, e1, f1, g1;
assign a1=(~A&~B&~C) | (~A&B&~C) | (~A&B&C) | (A&~B&C) | (A&B&~C) | (A&B&C);
assign b1=(~A&~B&~C) | (~A&~B&C) | (~A&B&~C) | (~A&B&C) | (A&~B&~C) | (A&B&C);
assign c1=(~A&~B&~C) | (~A&~B&C) | (~A&B&C) | (A&~B&~C) | (A&~B&C) | (A&B&~C)
        | (A&B&C);
assign d1=(~A&~B&~C) | (~A&B&~C) | (~A&B&C) | (A&~B&C) | (A&B&~C);
assign e1=(~A&~B&~C) | (~A&B&~C) | (A&B&~C);
assign f1=(~A&~B&~C) | (A&~B&~C) | (A&~B&C) | (A&B&~C);
assign g1=(~A&B&~C) | (~A&B&C) | (A&~B&~C) | (A&~B&C) | (A&B&~C);
endmodule
```

注意：在 assign 语句中，左边变量的数据类型必须是 wire 型。

1.2.2　行为描述方式

一个设计用行为描述方式给出其行为功能，可使用如下语句：

(1) initial 语句：该语句只执行一次，通常多用于仿真。

(2) always 语句：该语句总是循环执行，或者说该语句重复执行。

需要特别注意的是：① 只有寄存器类型数据能够在这两种语句中被赋值；② 寄存器类型数据在被赋给新值前保持原值不变；③ 所有的初始化语句 initial 和总是语句 always 在 0 时刻并发执行。

例 1.7　下面给出了用行为描述方式对前述七段译码器编写的源文件。

```
module cnt_7(a1, b1, c1, d1, e1, f1, g1, A, B, C);
output a1, b1, c1, d1, e1, f1, g1;
input A, B, C;
reg a1, b1, c1, d1, e1, f1, g1;
always  @(A or B or C)
begin                          //begin…end 顺序过程，或叫顺序块
case({A, B, C})                //case 语句在第 3 章介绍
3'd0:{ a1, b1, c1, d1, e1, f1, g1}=7'b1111110;
3'd1:{ a1, b1, c1, d1, e1, f1, g1}=7'b0110000;
3'd2:{ a1, b1, c1, d1, e1, f1, g1}=7'b1101101;
3'd3:{ a1, b1, c1, d1, e1, f1, g1}=7'b1111001;
3'd4:{ a1, b1, c1, d1, e1, f1, g1}=7'b0110011;
3'd5:{ a1, b1, c1, d1, e1, f1, g1}=7'b1011011;
3'd6:{ a1, b1, c1, d1, e1, f1, g1}=7'b1011111;
3'd7:{ a1, b1, c1, d1, e1, f1, g1}=7'b1110000;
```