绪 论

1.什么是信号处理电路？它通常由哪两大部分组成？

信号处理电路是进行一些复杂的数字运算和数据处理，并且又有实时响应要求的电路。它通常有高速数据通道接口和高速算法电路两大部分组成。

2.为什么要设计专用的信号处理电路？

因为有的数字信号处理对时间的要求非常苛刻，以至于用高速的通用处理器也无法在规定的时间内完成必要的运算。通用微处理器芯片是为一般目的而设计的，运算的步骤必须通过程序编译后生成的机器码指令加载到存储器中，然后在微处理器芯片控制下，按时钟的节拍，逐条取出指令分析指令和执行指令，直到程序的结束。微处理器芯片中的内部总线和运算部件也是为通用目的而设计，即使是专为信号处理而设计的通用微处理器，因为它的通用性也不可能为某一特殊的算法来设计一系列的专用的运算电路而且其内部总线的宽度也不能随便的改变，只有通过改变程序，才能实现这个特殊的算法，因而其算法速度也受到限制所以要设计专用的信号处理电路。

3.什么是实时处理系统？

实时处理系统是具有实时响应的处理系统。

4.为什么要用硬件描述语言来设计复杂的算法逻辑电路？

因为现代复杂数字逻辑系统的设计都是借助于 EDA 工具完成的，无论电路系统的仿真和综合都需要掌握硬件描述语言。

5.能不能完全用 C 语言来代替硬件描述语言进行算法逻辑电路的设计？

不能，因为基础算法的描述和验证通常用 C 语言来做。如果要设计一个专用的电路来进行这种对速度有要求的实时数据处理，除了以上 C 语言外，还须编写硬件描述语言程序进行仿真以便从电路结构上保证算法能在规定的时间内完成，并能通过与前端和后端的设备接口正确无误地交换数据。

6.为什么在算法逻辑电路的设计中需要用 C 语言和硬件描述语言配合使用来提高设计效率？

首先 C 语言很灵活，查错功能强，还可以通过 PLI 编写自己的系统任务，并直接与硬件仿真器结合使用。C 语言是目前世界上应用最为广泛的一种编程语言，因而 C 程序的设计环境比 Verilog HDL 更完整，此外，C 语言有可靠地编译环境，语法完备，缺陷缺少，应用于许多的领域。比较起来，Verilog 语言只是针对硬件描述的，在别处使用并不方便。而用 Verilog 的仿真，综合，查错等大部分软件都是商业软件，与 C 语言相比缺乏长期大量的使用，可靠性较差，亦有很多缺陷。所以只有在 C 语言的配合使用下,Verilog 才能更好地发挥作用。C 语言与 Verilog HDL 语言相辅相成，互相配合使用。这就是即利用 C 语言的完整性又要结合 Verilog 对硬件描述的精确性，来更快更好地设计出符合性能要求的

硬件电路系统，从而来提高效率。

第一部分 Verilog 数字设计基础

第 1 章 Verilog 的基本知识

1.什么是硬件描述语言？它的主要作用是什么？

硬件描述语言是一种用形式化方式来描述数字电路和系统的语言。它的主要作用是：数字电路系统的设计者利用这种语言可以从上层到下层（从抽象到具体）逐步描述自己的设计思想，用一系列分层次的模块来表示极其复杂的数字系统。

2.目前世界上符合 IEEE 标准的硬件描述语言有哪两种？它们各有什么特点？

符合 IEEE 标准的硬件描述语言是 Verilog HDL 和 VHDL 两种。它们的共同特点是：能够形式化地抽象表示电路的行为和结构；支持逻辑设计中层次与范围的描述；可借用高级语言的精巧结构来简化电路行为的描述；具有电路仿真与验证机制以保证设计的正确性；支持电路描述由高层到低层的综合 1 转换硬件描述与实现工艺无关；便于文档管理；易于理解和设计重用。不同点：Verilog HDL 是一种非常容易掌握的硬件描述语言,而 VHDL 掌握起来就比较困难。

3.什么情况下需要采用硬件描述语言的设计方法？

在对逻辑电路及系统的设计的时间要求很短的情况下需要采用硬件描述语言的设计方法。

4.采用硬件描述语言设计方法的优点是什么？有什么缺点？

优点是：与工艺无关性。这使得工程师在功能设计，逻辑验证阶段，可以不必过多考虑门级及工艺实现的具体细节，只需要利用系统设计时对芯片的要求，施加不同的约束条件，即可设计出实际电路。

缺点是：需要相应的 EDA 工具，而 EDA 工具的稳定性需要进一步的在工程中提升。

5.简单叙述一下利用 EDA 工具并采用硬件描述语言的设计方法和流程？

采用自顶向下的设计方法：从系统级开始把系统划分为基本单元，然后再把每个基本单元划分为下一层次的基本单元，一直这样做下去，直到可以直接用 EDA 元件库中的基本元件来实现为止。其基本流程主要由两大主要功能部分组成：(1)设计开发 即从编写设计文件->综合到布局布线->电路生成这样一序列步骤。(2)设计验证 也就是进行各种仿真的一序列步骤，如果在仿真过程中发现问题就返回设计输入进行修改。

5.硬件描述语言可以用哪两种方式参与复杂数字电路的设计？

复杂数字电路的设计和复杂数字电路的仿真验证。

7.用硬件描述语言设计的数字系统需要经过哪些步骤才能与具体的电路相对应？

编写设计文件；功能仿真；优化，布局布线；布线后门级仿真

8.为什么说用硬件描述语言设计的数字逻辑系统下具有很大的灵活性并可以映射到任何工艺的电路上？

硬件描述语言的设计具有与工艺无关性。这使得工程师在功能设计，逻辑验证阶段，可以不必过多考 虑门级及工艺实现的具体细节，只需要利用系统设计时对芯片的要求，施加不同的约束条件，即可设计出实际电路。

9.软核是什么？虚拟器件是什么？它们的作用是什么？

把功能经过验证的，可综合的，实现后电路结构总门数在 5000 门以上的 Verilog HDL 模型称为软核。而把由软核构成的器件称为虚拟器件。

作用：大大缩短设计周期，加快复杂电路的设计。

10.集成电路行业中 IP 的含义是什么？固核是什么？硬核是什么？与软核相比它们各有什么特点？各适合于什么场合？

在集成电路行业中 IP 是知识产权（Intellectual Property）的含义。把在某一现场可编程门阵列器件上实现的经验证是正确的，总门数在 5000 门以上的电路结构编码文件称为固核。把在某一专用集成电路工艺的器件上实现的经验证时正确的总门数在 5000 门以上的门电路结构版图掩膜称为硬核。

在工具实现手段和工艺技术尚未确定的逻辑设计阶段，IP 核具有很大的灵活性，很容易借助 EDA 工具与其他外部逻辑结合为一体。相比之下固核和硬核与其他外部逻辑结合为一体的灵活性要差很多。

11.简述 Top_Down 设计方法和硬件描述语言的关系？

Top_Down 的设计方法是首先从系统设计入手，从顶层进行功能划分和结构设计。系统的总仿真是顶层进行功能划分的总要环节，而该过程需要采用硬件描述语言的方法。

12.System Verilog 与 Verilog 有什么关系？适合于何种设计？

System Verilog 是 Verilog 语言的拓展和延伸。Veril 适合系统级，算法级，寄存器级，逻辑级，门级，电路开关级设计而 System Verilog 更适合于可重用的可综合 IP 和可重用的验证用 IP 设计，以及特大型基于 IP 的系统级设计和验证。

第 2 章 Verilog 语法的基本概念

1.Verilog 语言有什么作用？

·可描述顺序执行和并行执行的程序结构；

·用延迟表达式或事件表达式来明确地控制过程的启动时间；

·通过命名的事件来触发其他过程里的激活行为或停止行为；

·提供了条件如 if-else，case 等循环程序结构；

·提供了可带参数且非零延续时间的任务程序结构；

·提供了可定义新的操作符的函数结构；

·提供了用于建立表达式的算术运算符，逻辑运算符，位运算符；

·Verilog HDL 语言作为一种结构化的语言非常适用于门级和开光级的模型设计；

·提供了一套完整的表示组合逻辑的基本元件的原话；

·提供了双向通路和电阻器件的原话；

·可建立 MOS 器件的电荷分享和电荷衰减动态模型；

·Verilog HDL 的构造性语句可以精确地建立信号的模型；

2.构成模块的关键词是什么？

module，endmodule

3.为什么说可以用 Verilog 构成非常复杂的电路结构？

因为 Verilog 可描述顺序执行和并行执行的程序结构；用延迟表达式或事件表达式来明确地控制过程的启动时间；通过命名的事件来触发其他过程里的激活行为或停止行为；提供了条件如 if-else，case 等循环程序结构；提供了可带参数且非零延续时间的任务程序结构；提供了可定义新的操作符的函数结构；提供了用于建立表达式的算术运算符，逻辑运算符，位运算符；Verilog HDL 语言作为一种结构化的语言非常适用于门级和开光级的模型设计；提供了一套完整的表示组合逻辑的基本元件的原话；提供了双向通路和电阻器件的原话；可建立 MOS 器件的电荷分享和电荷衰减动态模型；Verilog HDL 的构造性语句可以精确地建立信号的模型。

4.为什么可以用比较抽象的描述来设计具体的电路结构？

因为有可以用比较抽象描述设计电路结构的语言，而这种语言是适合数字系统设计的语言。

5.是否任意抽象的符合语法的 Verilog 模块都可以通过综合工具转变为电路结构？

不能。要符合语法，还符合一些基本规则的 Verilog 模块才可以通过综合工具转变为电路结构。

6.什么叫综合？

通过综合工具把行为级描述的模块通过逻辑网表自动转化为门级形式的模块叫综合。

7.综合是由什么工具来完成的？

EDA 工具来完成综合的。

8.通过综合产生的是什么？产生的结果有什么用处？

产生的是由与门，或门和非门组成的加法器，比较器等组合逻辑。产生的模块很容易与某种工艺的基本元件逐一对应起来，再通过布局布线工具自动地转变为某种工具工艺的电路布线结构。

9.仿真是什么？为什么要进行仿真？

仿真是对电路模块进行动态的全面测试。通过观测被测试模块的输出信号是否符合要求可以调试和验证逻辑系统的设计和结构准确与否，并发现问题及时修改。

10.仿真可以在几层面上进行？每个层面的仿真有什么意义？

分别为;前仿真，逻辑网表仿真，门级仿真和布线后仿真；

前仿真，逻辑网表仿真，门级仿真;可以调试和验证逻辑系统的设计和结构准确与否，并发现问题及时修改。

布线后仿真：分析设计的电路模块的运行是否正常。

11.模块的端口是如何描述的？

用“.”表示被引用模块的端口。

12.在引用实例模块的时候，如何在主模块中连接信号线？

用小括号中来表示本模块中与之连接的模块。

13.如何产生连续的周期性测试时钟？

用 always 语句来产生连续的周期性测试模块。

14.如果不用 initial 块，能否产生测试时钟？

不能，没有 initial 块，就不知道时钟信号的初始值

15.从本讲中的简单例子，是否能明白 always 块与 initial 块有什么不同？

Initial 块只执行一次，而 always 块执行无数次。

16.为什么说 Verilog 可以用来设计数字逻辑电路和系统？

因为 Verilog 可描述顺序执行和并行执行的程序结构；用延迟表达式或事件表达式来明确地控制过程的启动时间；通过命名的事件来触发其他过程里的激活行为或停止行为；提供了条件如 if-else，case 等循环程序结构；提供了可带参数且非零延续时间的任务程序结构；提供了可定义新的操作符的函数结构；提供了用于建立表达式的算术运算符，逻辑运算符，位运算符；Verilog HDL 语言作为一种结构化的语言非常适用于门级和开光级的模型设计；提供了一套完整的表示组合逻辑的基本元件的原话；提供了双向通路和电阻器件的原话；可建立 MOS 器

件的电荷分享和电荷衰减动态模型；Verilog HDL 的构造性语句可以精确地建立信号的模型。

第 3 章 模块的结构、数据类型、变量和基本运算符号

1.模块由几个部分组成？

由描述接口和描述逻辑功能两部分组成。

2.端口分为几种？

三种：输出口，输入口，输入/输出口

3.为什么端口要说明信号的位宽？

因为如果不说明信号的位宽可能会在信号发生改变时发生错误，不容易看出接收到的信号的数据宽度，就很难进行数据的处理。

4.能否说模块相当于电路图中的功能模块，端口相当于功能模块的引脚？

可以那样说，每个模块都有特定的功能，而功能的实现就必须依靠具体的电路得以实现，端口是信号传递的通道，可以说是功能模块的引脚。

5.模块中的功能描述可以由哪几类语句或语句块组成？它们出现的顺序会不会影响功能的描述？

用 assign 语句声明，用实例元件，用 always 块。它们出现的顺序不会影响到功能的描述。

6.这几类描述中哪一种直接与电路结构有关？

用实例元件直接与电路结构有关。

7.最基本的 Verilog 变量有哪几种类型？

wire 型、reg 型、memory 型

8.reg 型和 wire 型变量的差别是什么？

reg 型变量是寄存器型变量，wire 型变量是连线型变量。两者根本性的差别在于 reg 型变量有个寄存器来存放变量，这个值只有变量发生改变时才会改变否则保证原来的值不变，wire 型变量的值不是确定的值。

9.由连续赋值语句（assign）赋值的变量能否是 reg 类型的？

可以是 reg 类型的变量。

10.在 always 模块中被赋值的变量能否是 wire 类型的？如果不能是 wire 类型，那么必须是什么类型的？它们表示的一定是实际的寄存器吗？

不能。必须是 reg 类型的变量，它们表示不一定是实际的寄存器。

11.参数类型的变量有什么用处？

参数类型的变量的好处是可以提高程序的可读性和可维护性。

12.Verilog 语法规定的参数传递和重新定义功能有什么直接的应用价值？

可以用于定义延迟时间和变量宽度。

13.逻辑比较运算符小于等于“<=”和非阻塞赋值大于等于“<=”的表示是完全一样的，为什么 Verilog 在语句解释和编译时不会搞错？

因为逻辑比较时“<=”两边是两个操作数，此时“<=”是双目运算符，而在非阻塞赋值时“<=”的右边是操作数，此时“<=”单目运算符。

14.是否可以说实例引用的描述实际上就是严格意义上的电路结构描述？

不能实例引用的描述是在门级电路上加以描述的，和严格意义上的电路结构描述还是有点差距的。

第 4 章 运算符、赋值语句和结构说明语句

1.逻辑运算符与按位逻辑运算符有什么不同，它们各在什么场合使用？

用逻辑运算符运算时是两个操作数进行逻辑运算，而按位逻辑运算符运算时是两个操作数对应的每一位进行逻辑运算。逻辑运算符多用于条件的判断，按位逻辑运算符用于信号的运算和检测。

2.指出两种逻辑等式运算符的不同点，解释书上的真值表。

两种逻辑运算符有很大的区别。”===“要求两个比较数完全一样，无论高阻还是未知，只要每位完全相同即可；而“==”只有在两个操作数每位都已知即 1 或 0，在这种前提下两个操作数每位相同结果才为真，如果不是在这个前提那么其结果始终为 x;

3.拼接符的作用是什么?为什么说合理地使用拼接符可以提高程序的可读性和可维护性？拼接符表示的操作其物理意义是什么？

拼接符的作用是把两个或多个信号的某些位拼接起来进行运算操作。因为借助拼接符可以用一个符号名来表示由多位信号组成的复杂信号。其物理意义是将多个信号结合成一个信号。

4.如果都不带时间延迟，阻塞和非阻塞赋值有什么不同？举例说明它们的不同点？

阻塞和非阻塞赋值的区别在阻塞是顺序执行而非阻塞是并行执行。

以下面的语句举例

非阻塞赋值

```
always@(posedge clk)
begin
b<=a;
c<=b;
end
```

阻塞赋值

```
always@(posedge clk)
begin
b=a;
c=b;
end
```

两种不同的赋值方式结果是不同的，非阻塞赋值 b<=a;c<=b;两条语句是同时执行的，而阻塞赋值 b=a;c=b;两条语句先执行 b=a 后执行 c=b.

5.举例说明顺序块和并行块的不同？

```
begin
#50   r=’h35;
#50   r=’hE2;
#50   r=‘h00;
#50   r=’hF7;
#50   ->end_wave;
end

fork
#50   r=’h35;
#100   r=’hE2;
#150   r=‘h00;
#200   r=’hF7;
#250   ->end_wave;
join
```

上面两个块执行起来效果是完全一样的，第一个模块是按顺序执行，而第二个模块是每个语句同时执行的。

6.如果在顺序块中，前面有一条语句是无限循环，下面的语句能否进行？

下面的语句不能执行。

7.如果在并行块中，发生上述情况，会如何呢？

下面的语句能够执行。

第 5 章 条件语句、循环语句、块语句与生成语句

1.为什么建议在编写 Verilog 模块程序时，如果用到 if 语句建议大家把配套的 else 情况也考虑在内？

因为如果没有配套的 else 语句，在不满足 if 条件语句时，将会保持原来的状态不变，从而在综合时会产生一个锁存器，而这是设计不想要的结果。

2.用 if 语句；elseif 语句；elseif 语句；...else 语句和用 case endcase 表示不同条件下的多个分支是完全相同的，还是有什么不同？

不是完全相同。（1）与 case 语句中的控制表达式和多分支表达式这种比较相比，if_else_if 结构中条件表达式更为直观些。（2）对于那些分支表达式中存在不定值 x 和高阻值 z 的位时 case 语句提供了处理这种情况的手段。

3.如果 case 语句的分支条件没有覆盖所有可能的组合条件，定义了 default 项和没有定义 default 项有什么不同？

定义了 default 项则会使电路描述的更加的清楚，综合的时候不会产生不想要的结果，没用定义 default 则会使在综合是产生一个锁存器。

4.仔细阐释 case、casex 和 casez 之间的不同。

表 5.1 case,casez 和 casex 的真值

case	0	1	x	z	casez	0	1	x	z	casex	0	1	x	z
0	1	0	0	0	0	1	0	0	1	0	1	0	1	1
1	0	1	0	0	1	0	1	0	1	1	0	1	1	1
x	0	0	1	0	x	0	0	1	1	x	1	1	1	1
z	0	0	0	1	z	1	1	1	1	z	1	1	1	1

case、casex、casez 对应的真值表如上，可以看出 case 无论是 0,1,还是 x 高阻都能够比较，而 casez 不将高阻进行比较，在其它情况都进行比较；而 casex 不将高阻和 x 进行比较，在其它情况进行比较。

5.forever 语句如果运行了，在它下面的语句能否运行？它位于 begin end 和位于 fork join 块有什么不同？

不能运行。位于 begin end，由于 begin and 是顺序块，所以只要执行到 forever 则将不能运行下面的程序；而位于 fork join，它是并行块，执行了 forever 还是能够执行 forever 下面的语句。

6.forever 语句 repeat 语句能否独立于过程块而存在，即能否不在 initial 或 always 块中使用？

forever 不能独立于过程块中，而 repeat 能够独立于过程块中。

7.用 for 循环为存储器许多单元赋值时是否需要时间？为什么如果不定义时间延迟，它可以不需要时间就把不管多大的储存器赋值完毕？

如果定义了时间延迟则需要时间，否则不需要时间。因为循环的边界是确定的，那么在综合时该循环语句被认为是重复的硬件结构。

8.for 循环是否可以表示可以综合的组合逻辑？请举例说明。

可以表示综合的组合逻辑。例如用 for 循环实现的乘法器

9.在编写测试模块时用什么方法可以使 for 循环按照时钟的节拍运行？请比较图 5.3 所示程序段。

```
always @(posedge clk)
  begin
    for ( i=0; i<=1024; i=i+1)
        mem[i] = i;
  end
这样写能不能按照时钟节拍来对
mem[i]赋值？右边框内的程序呢？
```

```
initial
  begin
    for ( i=0; i<=1024; i=i+1)
        begin
          mem[i] = i;
          @(posedge clk)
    end
  end
```

图 5.3　两种程序段

可以在 for 循环的最后嵌套时钟节拍运行的信号。第一种程序不能按照时钟节拍来对 mem[i]赋值，而第二种程序可以。

10.声明一个为 oscillate 的寄存器变量并将它初始化为 0，使其每 30 个时间单位进行一次取反操作，不要使用 always 语句(提示：使用 forever 循环)。

```
reg oscillate;
initial
  begin
  oscillate=0;
  forever
  #30 oscillate=！Oscillate;
  end
```

11.设计一个周期为 40 个时钟单位的时钟循环，其占空比为 25%，使用 always 和 initial 块进行设计，将其在仿真 0 时刻的值初始化为 0。

```
 initial
    begin
      clock=0;
      always
      begin
```

```
    #30 clock=0;
    #10 clock=1;
    end
  end
```

12.给定下面含有阻塞过程赋值语句的 initial 块，每个语句在什么仿真时刻开始执行？a，b，c 和 d 在仿真过程中的中间值和仿真结束时的值是什么?

```
initial
begin
  ①  a=1'b0;
  ②  b=#10 1'b0;
  ③  c=#5   1'b0;
  ④  d=#20{a,b,c};
end
```

第一条语句在仿真开始时就执行，第二句在仿真 10 个时钟单元后执行，第三句在仿真 15 个时钟信号单元后执行，第四句在仿真 35 个时钟单元后执行。在中间仿真过程中 a=0,b,c,d 为不确定值结束时 abcd 的值是 a=1'b0,b=1'0,c=1'0,d=3'b000。

13.在第 12 题中，如果 initial 块中包含的是非阻塞过程赋值语句，那么各个问题的答案是什么？

如果是非阻塞过程赋值则答案是;第一条语句在仿真开始时就执行，第二句在仿真 10 个时钟单元后执行，第三句在仿真 5 个时钟信号单元后执行，第四句在仿真 20 个时钟单元后执行。在中间仿真过程中 a=0,b,c,d 为不确定值结束时 abcd 的值是 a=1'b0,b=1'0,c=1'0,d=3'b000。

14.下面例子中 d 的最终值是什么？

```
initial
begin
b=1'b1;
c=1'b0;
#10   b=1'b0;
end
initial
begin
d=#25(b|c);
end
```

D 的最终值 0。

15 使用带同步清零端的 D 触发器（清零高电平有效，在时钟下降沿执行清零操作）设计下一个下降沿触发的 D 触发器，只能使用行为语句(提示：D 触发器的输出 q 应当声明为寄存器变量)。使用设计出的 D 触发器输出一个周期为 10 个时间单位的时钟信号。

```
module D_FF(CLR,CLK,D,Q);
input CLR,CLK,D;
output Q;
reg Q;
always @(posedge CLR or necedge CLK)
begin
if(CLR)    Q=0;
else
#10 Q<=D;
end
endmodule
```

16.使用带有异步清零端的 D 触发器设计第 15 题要求的 D 触发器(在清零端变为高电平后立即执行清零操作，无须等待下一个时钟下降沿)，并对这个 D 触发器进行测试。

```
module D_FF(CLR,CLK,D,Q);
input CLR,CLK,D;
output Q;
reg Q;
always @(posedge CLR )
begin
Q<=0;
end
always @(necedge CLK)
begin
#10 Q<=D;
end
endmodule
```

17.使用 wait 语句设计一个电平敏感的锁存器，该锁存器的输入信号为 d 和 clock，输出为 q，其功能是当 clock=1 时 q=d；

```
module L_FF(d,clock,q);
  input    d,clk;
```

```
 output    q;
 reg    q;
 always
 begin
   wait(clock==1)
   q=d;
 end
endmodule
```

18.使用条件语句设计[例 5.18]中的四选一多路选择器，外部端口必须保持不变。

```
module mux4_to_1(out,i0,i1,i2,i3,s1,s0);
output out;
input i0,i1,i2,i3;
input s1,s0;
reg out;
always @(s1 or s0 or i0 or i1 or i2 or i3)
begin
    if(s1==0&&s0==0)
       out=i0;
    else if(s1==0&&s0==1)
       out=i1;
    else if(s1==1&&s0==0)
       out=i2;
    else if(s1==1&&s0==1)
       out=i3;
    else
       out=1'bx;
end
endmodule
```

19.使用 case 语句设计八功能的算术运算单元(ALU),其输入信号 a 和 b 均为 4 位，还有功能选择信号 select 为 3 位，输出信号为 out(5 位)，算术运算单元 ALU 所执行的操作与 select 信号有关，具体关系如 5.1 所列(忽略输出结果中的上溢和下溢的位)。

表 5.1　select 信号的功能

select 信号	功　能
3 'b 000	out = a
3 'b 001	out = a+b
3 'b 010	out = a−b
3 'b 011	out = a/b
3 'b 100	out=a%b（余数）
3 'b 101	out = a << 1
3 'b 110	out = a >> 1
3 'b 111	out = a > b（大小比较）

```
module ALU(a,b,select,out);
input[3:0]    a,b;
input[2:0]    select;
output[4:0] out;
reg[4:0] out;
always @(select)
begin
  case(select)
       3'b000: out=a;
       3'b001: out=a+b;
       3'b010: out=a-b;
       3'b011: out=a/b;
       3'b100: out=a%b;
       3'b101: out=a<<1;
       3'b110: out=a>>1;
       3'b111: out=a>b;
       default:out=5'bx;
    endcase
end
endmodule
```

20.使用 while 循环设计一个时钟信号发生器。其时钟信号的初值为 0，周期为 10 个时间单元。

```
initial
begin
clk=0;
while(1)
   #10 clk=!clk;
end
```

21.使用 for 循环对一个长度为 1024(地址从 0~1023)、位宽为 4 的寄存器类型数组 cache_var 进行初始化，把所有单元都设置为 0.

```
begin
 reg[3:0]   cache_var[1023:0];
 intiger i;
 for(i=0;i<1024;i++)
    cache_var[i]=0;
end
```

22.使用 forever 循环设计一个时钟信号，周期为 10，占空比为 40%，初值为 0.

```
initial
begin
 clk=0;
 forever
  begin
    #6 clk=0;
    #4 clk=1;
   end
end
```

23.使用 repeat 将语句 a=a+1 延迟 20 个时钟上升沿之后再执行。

```
parameter delay=20;
intiger   i;
reg a;
begin
repeat(delay)
```

```
 always @(posedge clk)
 begin
    i++;
    if(i==20)
    a=a+1;
 end
end
```

24.下面是一个内嵌顺序块和并行块的块语句。该块的执行结束时间是多少？事件的顺序是怎样的？每条语句的仿真结束时间是多少？

```
 initial
 begin              //  顺序执行
  x=1'b0;
 #5    y=1'b1;         .....5
    fork                 //并行执行
     #20 a=x;          .....25
     #15 b=y;          .....20
    join
 #40    x=1'b1;    //  顺序执行    .....65
    fork                //并行执行
     #10 p=x;             .....75
     begin          //  顺序执行
      #10 a=y;            .....75
      #30 b=x;            .....105
     end
     #5 m=y;              .....70
    join
 end
```

该块的执行结束时间是 5+20+40+40=105 个时钟单位

每条语句执行的时间如上图所标

25.用 forever 循环语句，命名块(named block)和禁用(disabling of) 命名块来设计一个八位计数器。这个计数器从 count=5 开始计数，到 count=67 结束计数。每个时钟正跳变计数器加一，时钟的周期为 10，计数器的设计只用到了一次循环，然后就被禁用了(提示：使用 disable 语句)。

```
reg[7:0]    count;
initial
  begin
    count =5;
  end
begin:mame block
    forever
      begin: disabling of
        always @(posedge clk)
            begin
              if(count<67)
              #10 count=count+1;
              disable: disabling of;
            end
      end
end
```

第 6 章 结构语句、系统任务、函数语句和显示系统任务

1.怎样理解 initial 语句只执行一次的概念?

在仿真开始时，initial 语句只执行一次，但 initial 语句里面的语句可能不执行一次，因为如果是 while 循环，虽然 initial 语句是执行一次，但只要进了 while 循环，则会执行到仿真结束。

2.在 initial 语句引导的过程中是否可以有循环语句？如果可以，是否与思考题 1，互相矛盾？

可以，并不互相矛盾。initial 语句确实是执行了一次，但并不意味着 initial 语句过程中就不能是循环语句，两者并不矛盾。

3.怎样理解由 always 语句引导的过程块是不断活动的？

always 语句的过程块是不断活动的，在仿真过程中 always 块始终在循环的活动着，检查 always 语句后面的信号是否发生相应改变，这是 always 活动的实质，如果 always 语句后面没有检查的信号则将会进入一个循环，将会使仿真器锁死。

4.不断活动与不断执行有什么不同？

不断活动是 always 语句不断活动检查是否满足条件（如某个信号发生改变），不断执行时 always 语句引导的过程中的语句不断的执行着。

5.怎样理解沿触发和电平触发的不同？

沿触发是在某个信号在上升沿或下降沿到来时，触发执行过程块。电平触发是在某个信号发生改变时就会触发执行过程块。

6.是不是可以说沿触发是有间隔的，在一定的时间区间里只需要注意有限的点，而电平触发却需要注意无穷多个点？

不是。沿触发是信号的上升沿或下降沿进行触发，而电平触发是在某个信号发生改变时进行触发，并不需要注意无穷多个点。

7.沿触发的 always 块和电平触发的 always 块各表示什么类型的逻辑电路的行为？为什么？

沿触发的 always 块常表示时序逻辑电路，因为其和时序有很关。电平触发的 always 块常表示组合逻辑电路，因为其和只和电平有关。

8.简单叙述任务和函数的不同点

（1）函数只能与主模块共用同一个仿真时间单元，而任务可以定义自己的仿真时间单位；

（2）函数不能启动任务，而任务能启动其他任务和函数；

（3）函数至少要有一个输入变量，而任务可以没有或有多个任何类型的变量；

（4）函数返回一个值，而任务则不返回值。

9.简单叙述$display、$write 和$strobe 的不同点。

$display 自动地在输出后进行换行，$write 则不进行自动换行，其它都非常相似。

如果许多其他语句与$display 任务在同一个时间单位执行，那么这些语句与$display 任务的执行顺序是不确定的。如果使用$strobe，该语句总是在同时刻的其他赋值语句执行完成之后执行。

10.简单叙述 Verilog1364-2001 版语法规定的电平敏感列表的简化写法。

关键词“or”也可以使用“,”来代替。

11.如何在 Verilog 测试模块中，利用文件的读写产生预定格式的信号，并记录有测试价值的信号？

Verilog 提供了系统任务来选择要转储的模块实例或模块实例信号（dumpvars），选择 VCD 文件的名称($dumpfile),选择转储过程的起点和终点($dumppon，$dumpoff),选择生成检测点(Sdunpall)，其使用方法如下

```
initial
$dumpfile("myfile.dmp");   //仿真信息转储到 myfile.dmp 文件
initial
$dumpvars;        //没有指定变量范围，把设计中全部信号都转储
initail
$dunpvars(1,top);   //转储模块实例投票中的信号
                    //数 1 表示层次的等级只转储 top 下第一层信号
                    //即转储 top 模块中的变量，而不转储在 top 中引用
initial
$dumpvars(2,top.m1);   //转储 top.m1 模块下两层的信号
initial
$dumpvars(0,top.m1);   //数 0 表示转储 top.m1 模块下面各个层的所有信号
initial
begin
$dumpon;              //启动转储过程
#100000 $dunpoff;   //过了 100000 个仿真单位后，停止转储过程
end
initial
//生成一个检查点，转储所有 VCD 变量的现行值
$dumpall;
```

第 7 章 调试用系统任务和常用编译预处理语句

1.为什么在多个模块调试的情况下$monitor 需要配合$monitoron 和$monitoroff 来工作？

$monitoron 和$monitoroff 任务的作用是通过打开和关闭监控标志来控制监控任务$monitor 的启动和停止，这样使得程序员可以很容易地控制$monitor 何时发生。$monitoron 则用于打开监控标志，启动监控任务$monitor。通常在通过调用$monitoron 来启动$monitor 时不管$monitor 参数列表中的值是否发生改变，总是立刻输出显示当前时刻参数列表中的值，这用于在监控的初始时刻设定初始比较值。在默认情况下，控制标志在仿真的起始时刻就已经打开了。在多模式调试的情况下，许多模块中都调用了$monitor，因为任何时刻只能用一个$monitor 起作用，因此需配合$monitoron 与$monitoroff 使用，把需要监视的模块用$monitor 打开，在监视完毕后及时用$monitoroff 关闭，以便把$monitor 让给其他模块使用。

2.请用$random 配合求模运算编写：

(1)用于测试的跳变沿抖动为周期 1/10 的时钟波形。

(2)随机出现的脉宽随机的窄脉宽。

```
module random_pulse(dout);
output[9:0]    dout;
reg[9:0] dout;
integer delay;
initial
begin
  #10 dout=0;
    for(k=0;k<100;k=k+1)
      begin
         delay={$random}%10;
         #delay dout=1；
         #delay dout=0；
      end
  end
endmodule
```

3.Verilog 的编译预处理与 C 语言的编译预处理有什么不同？

Verilog 的编译处理，在编译处理命令之前要以‘`’开头。

4.请仔细阐释`timescale 编译预处理的作用？

`timescale 命令用来说明跟在该命令后的模块的时间单位和时间精度。使用`timescale 命令可以在同一个设计里包含采用了不同的时间单位的模块。

5.不同`timescale 定义的多模块仿真测试时需要注意什么？

如果在同一个设计里，多个模块中用到的时间单位和时间精度单位不同，需要用到以下的时间结构：

（1）用`timescale 命令来声明本模块中所用到的时间单位和时间精度；

（2）用系统任务$printtimescale 来输出显示一个模块的时间单位和时间精度；

（3）用系统函数$time 和$realtime 及%t 格式声明来输出显示 EDA 工具记录的时间信息。

6.为什么说系统任务$readmen 可以用来产生用于算法验证的极其复杂的测试用数据流？

在 Verilog HDL 程序中有两个系统任务$readmemb 和$readmemh，并用来从文件中读取数据到存储器中，这两个系统任务可以在仿真的任何时刻被执行使用。复杂数据可以用 C 语言产生，存在文件中，用$readmem 取出存入存储器，在按节拍输出，这在验证算法逻辑电路时特别有用。

7.为什么熟练地使用条件编译命令可以使源代码有更大的灵活性，可以使用于不同的实现对象，如不同工艺的 ASIC 或速度规模不同的 FBGA 或 CPLD,从而为软核的商品化创造条件？

合理的使用条件编译和条件执行预处理可以使测试程序适应不同的编译环境，也可以把不同的测试过程编写到一个统一的测试程序中去，可以简化测试的过程，对于复杂设计的验证模块的编写很有实用价值。

第二部分　设计和验证部分

第 9 章 Verilog HDL 模型的不同抽象级别

1.Verilog HDL 的模型共有哪几种类型(级别)?

有 5 种类型，系统级，算法级，RTL 级，门级，开关级。

2.每种类型的 Verilog HDL 各有什么特点？主要用于什么场合？

系统级，算法级和 RTL 级是属于行为级，门级是属于结构级的。

系统级：用高级语言结构实现设计模块的外部性能的模型。

算法级：用高级语言结构实现设计算法的模型。

RTL 级：描述数据在寄存器之间流动和如何处理这些数据的模型。

门级：描述逻辑门以及逻辑门之间的连接的模型。

开关级：描述器件中三极管和存储节点以及它们之间连接的模型。

3.不可综合成为电路的 Verilog 模块有什么好处？

描述比较直观

4.为什么说 Verilog HDL 的语言结构可以支持构成任意复杂的数字逻辑系统？

通过 Verilog 语言中的模块实例引用，可以构成任何复杂结构的电路，这种以结构方式所建成的 Verilog 模型不仅是可以以仿真的，而且也是可以综合的，其本质是表示电路的具体结构，也可以说这种 Verilog 文件也是一种结构网表。

5.什么是综合？是否任何符合语法的 Verilog HDL 程序都可以综合？

综合是通过综合器把 HDL 程序转化成标准的门级结构网表。不是任何复合语法的 Verilog HDL 程序都可以综合。

6.综合生成的是不是真实的电路？若不是，还需要哪些步骤才能真正成为具体的电路？

不是，真实的电路还需要利用 ASIC 和 FPGA 制造厂商的布局布线工具，根据综合生成的标准的门级结构网来产生。

7.为什么综合以后还可以用 Verilog 进行仿真？

通过综合以后产生的是门级结构，而门级结构再经过 Verilog 仿真测试验证其正确性

8.同一物理电路的行为模块仿真验证与结构模块的仿真验证在意义上有什么不同？

行为模块的仿真首先要转换为结构模块再进行仿真验证，结构模块是直接进行仿真的。

9.为什么说前端逻辑设计必须包含结构仿真验证，只有行为验证是远远不够的？

只有行为仿真远远不够的，因为 Verilog HDL 各种建模的方法，发挥各自在不同类型电路描述中的长处，而且要在层次管理工具的协调下把各个既独立又相互联系的模块组和，才能有效的设计出高质量的数字电路来。

10.什么是 TOP-DOWN 设计方法？通过什么手段来验证系统分块的合理性。

通过行为建模把一个复杂的系统分解成可操作的若干个模块，每个模块之间的逻辑关系通过行为建模的仿真加以验证。

11.编写两路每路为 1 位信号的二选一多路器的行为模块，再编写它的结构模块，然后编写测试模块分别对这两个模块进行测试，观测仿真运行的结果，编写实验报告。

行为模块

```
module sel2_1(a,b,s,f);
input a,b,s;
```

```
 output f;
 reg f;
 always @(a,b,s)
    begin
    if(s==0)    f=a;
      else
      f=b;
      end
 endmodule
```

结构模块

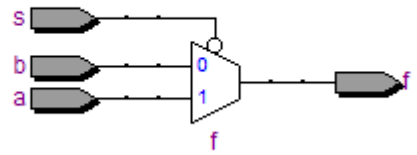
```
   module sel2_1(a,b,s,f);
   input a,b,s;
   output f;
   wire s_not,and1,and2;
   not U1(s_not,s);
   and U2(and1,b,s),
         U3(and2,a,s_not);
   or    U4(f,and1,and2);
   endmodule
```

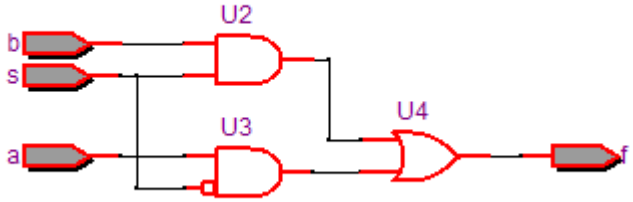
12.如果让你编写两路每路为 8 位信号的二选一多路器的结构模块是不是感觉麻烦？编写行为模块是不是很方便？

是，编写行为模块会很方便，只需要将输入输出的一位改为八位。

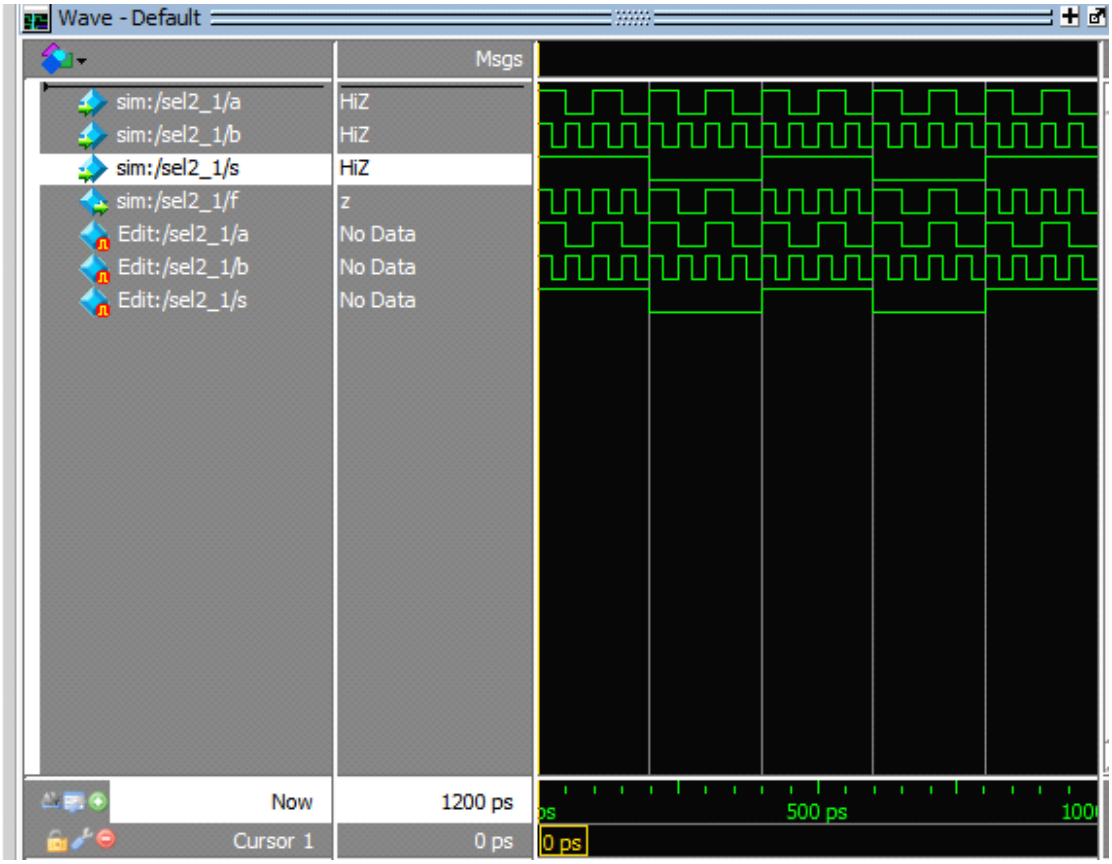
13.用什么方法可以把行为模块转换为结构模块？用你掌握的综合器，把 10 题和 11 题从行为模块转换为 Verilog 网表，仔细阅读自动生成的网表文件。

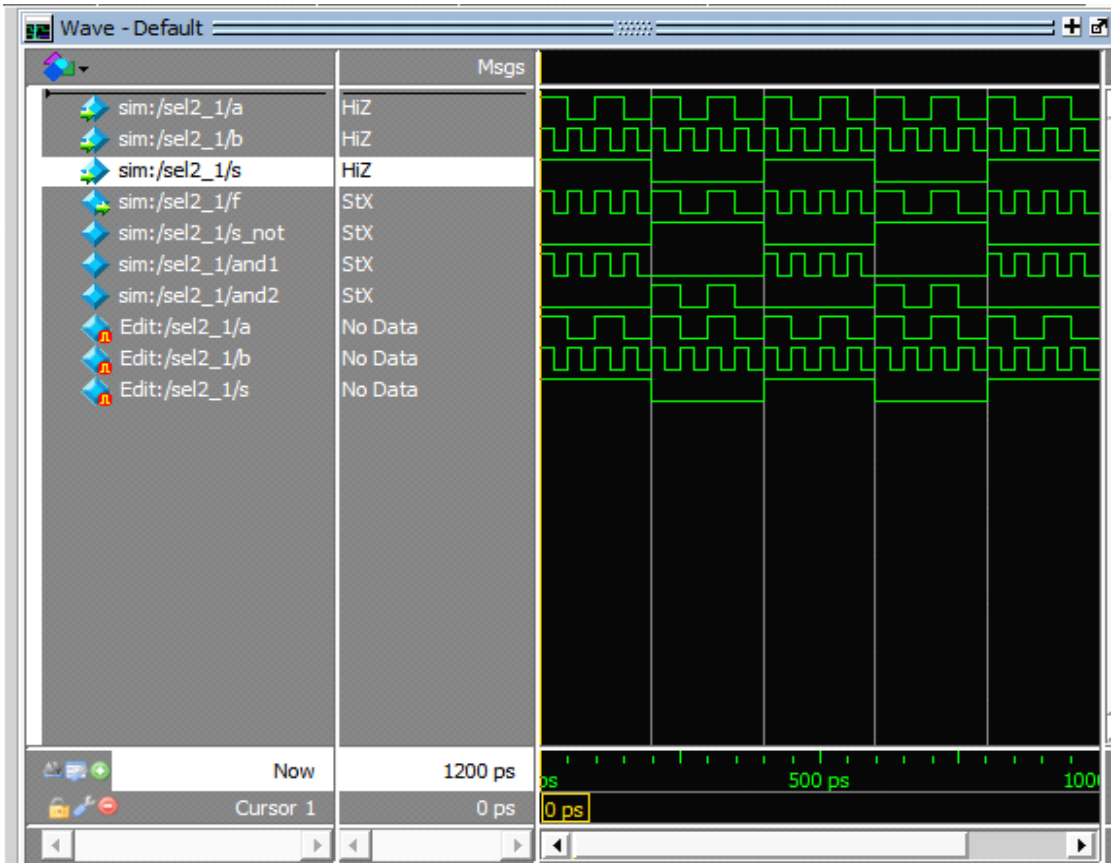
用 EDA 综合工具可以把行为模块转换为结构模块。





14.编写测试模块分别对行为的和自动生成的 Verilog 网表进行测试，比较仿真结果细微的不同，分析为什么不同。


Wave - Default
Msgs
sim:/sel2_1/a
sim:/sel2_1/b
sim:/sel2_1/s
sim:/sel2_1/f
Edit:/sel2_1/a
Edit:/sel2_1/b
Edit:/sel2_1/s
HiZ
HiZ
HiZ
z
No Data
No Data
No Data
Now
1200 ps
Cursor 1
0 ps
500 ps
0 ps


Wave - Default
Msgs
sim:/sel2_1/a
sim:/sel2_1/b
sim:/sel2_1/s
sim:/sel2_1/f
sim:/sel2_1/s_not
sim:/sel2_1/and1
sim:/sel2_1/and2
Edit:/sel2_1/a
Edit:/sel2_1/b
Edit:/sel2_1/s
HiZ
HiZ
HiZ
StX
StX
StX
StX
No Data
No Data
No Data
Now
1200 ps
Cursor 1
0 ps
500 ps
0 ps

第 10 章 如何编写和验证简单的纯组合逻辑

1.写出八位加法器和八位乘法器的逻辑表达式，比较用超前进位逻辑和不用超前进位逻辑的延迟。

八位加法器：

```
module add8（a,b,sum,cout）;
input[7:0] a,b;
output[7:0] sum;
output cout;
assign {count,sum}=a+b;
endmodule
```

八位乘法器

```
module add8（a,b,m）;
input[7:0] a,b;
output[15:0] m;
assign m=a*b;
endmodule
```

用超前进位逻辑可以减少由于逐位进位信号的传递所造成的延时。

2.为什么用算法操作符号表示的加法器和乘法器能通过综合器转变为逻辑电路？除了用算法操作符的表达式实现加法器和乘法器外，是否可以直接引用可配置的参数化实例来实现算术操作电路？

因为库中已经存在着可配置的参数化加法器乘法器的电路结构和相应行为模型。可以直接引用可配置的参数化实例来实现算术操作电路。

3.提高复杂运算组合逻辑运算速度有哪些办法？

采用流水线的设计方法，用总线的方式实现数据流通。

4.如何用 Verilog HDL 模块来描述总线的操作？为什么总线的操作必须有严格的时序控制？

各运算部件和数据寄存器可以通过带控制的三态门与总线的连接，通过对控制端电平的控制来确定在某一段时间内，总线归哪两个或哪几个部件使用。因为使总线连接模块能正常工作的最重要的因素是与其他模块的配合，控制信号的互相配合由同步状态机控制的开关阵列控制。

5.详细解释为什么采用流水线的办法可以显著提高层次多的复杂组合逻辑的运算速度。

采用流水线技术可以在相同的半导体工艺的前提下通过电路结构的改进大幅度地提高重复多次使用的复杂组合逻辑计算电路的吞吐量，从而来提高层次多的复杂组合逻辑的运算速度。如果某个组合逻辑设计的处理流程可以分为若干个步骤，而且整个数据处理过程是单向的即没有反馈或者迭代运算，前一个步骤的输出是下一个步骤的输入，则可以考虑采用流水线设计方法提高系统的数据处理频率即吞吐量。把组合逻辑分成延迟时间相等的小块，每块完成一定的组合逻辑功能都用寄存器暂存保存组合逻辑输出的数据值，只要小块的组合逻辑的延迟小于时钟周期，整个组合逻辑的输入值每个时钟就可以变化一次不会由于组合逻辑的延迟引起输出值的错误，若没有这些寄存器来暂时保存局部组合逻辑的输出值，则为了保证整个组合逻辑的输出正确，输出端信号的变化周期必须大于整体

逻辑的延迟时间。数据处理的吞吐量受到限制，采用流水线方法，虽然第一次输出有较长的延迟，但过了若干个周期后，每个时钟周期可以输出值一次，数据处理的频率，即吞吐量大大增加了。

第 11 章 复杂数字系统的构成

1.利用数字电路的基本知识解释，为什么说即使组合逻辑的输出端的所有信号同时变化，其输出端的各个信号不可能同时达到新的值？各个信号变化快慢有什么决定？

由于逻辑门和布线有延迟，因此没有办法使实际电路的输出与理想的布尔方程计算完全一致，可以说实际组合逻辑电路输出的瞬间不确定性是无法避免的。所以说即使组合逻辑的输出端的所有信号同时变化，其输出端的各个信号不可能同时达到新的值。各个信号变化的快慢与逻辑门和布线造成的传输延时有关。

2.如果组合逻辑的输入端信号变化非常快，其输出端的逻辑关系能否正确？变化快到什么程度以后，就没有正确的输出？如果还有正确输出，但时间片段很少，有什么办法可以加长正确输出的时间片？

其输出端的逻辑关系不能确定是否正确。当快到比确定下一个状态所使用的组合电路的延迟都快，就没有正确的输出。在输出端后加一个寄存器，有一个时钟控制，时钟周期宽度尽可能大些。

3.为使运算组合逻辑有一个正确的输出，为什么必须在复杂运算组合逻辑的输入端和输出端增加寄存器组来存放数据？

由于逻辑门和布线有延迟，因此没有办法使实际电路的输出与理想的布尔方程计算完全一致，可以说实际组合逻辑电路输出的瞬间不确定性是无法避免的。如果能使组合逻辑电路的输入端稳定一段时间，即所有的输入信号在一段相对较长的时间段里不再发生变化，虽然在稳定时间片段的刚一开始由于冒险竞争现象会产生与理想情况不一致的毛刺或输出不确定的情况，但只要稳定时间片段大于最长的路径延迟，就可以取得组合逻辑电路的理想输出。如果能躲开输出不确定片段，在理想值稳定输出的片刻把该输出端存入寄存器组，则寄存器组中保留的就是该组合逻辑电路的理想输出。

4.对每一个寄存器组来说，上一个时钟的正跳沿是为置数做准备，下一个时钟正跳沿是把本寄存器组置数（并为下一级运算组合逻辑送去输入信号），则为下一级寄存器组的置数做准备的先决条件是什么？

确定下一个状态所使用的组合电路的延迟和时钟到各触发器的差值必须小于一个时钟周期的宽度。

5.Verilog 语法中使用了哪一种赋值符号刻意表示与硬件寄存器组实现完全一致的赋值方式？

用"<="赋值符号刻意表示与硬件寄存器组实现完全一致的赋值方式。

6.一个带使能端的寄存器组被赋入一个正确的输入值需要哪三个条件？

（1）启用同步时序逻辑；

（2）下一个状态所使用的组合电路的延迟和时钟到各触发器的差值必须小于一个时钟周期的宽度；

（3）使能端被使能。

7.为什么建议大家采用同步时序来设计数字逻辑电路，异步逻辑有什么不好？

用 Verilog HDL 设计的可综合模块，必须避免使用异步时序逻辑，这不但是因为许多综合器不支持异步时序逻辑的综合，而且因为用异步时序逻辑确实很难控制由组合逻辑和延迟所产生的冒险和竞争。当电路的复杂度增加时，异步时序逻辑无法调试。工艺的细微变化也会造成异步时序逻辑电路的失效。因为异步时

序逻辑中触发条件很随意，任何时刻都有可能发生，所以记录状态的寄存器组的输出在任何时刻都能发生变化。而同步时序逻辑中的触发输入至少可以维持一个时钟后才会发生第二次触发。这是一个非常重要的差别，因为可以利用这一时间段，即在下一个触发信号来到之前为电路状态的改变创造一个稳定可靠地条件。

8.简单叙述不同时钟域模块之间数据准确传送的方法。

使用 RAM(DPRAM)、FIFO 缓冲的方法完成异步时钟域之间的数据传递。在输入端口使用前级时钟写数据，在输出端口使用本机时钟读数据，并有缓冲器空或满的控制信号来管理数据的读写，以避免数据的丢失，可以非常方便准确地完成异步时钟域之间的数据交换。

第 12 章 同步状态机的原理、结构和设计

1.举例说明状态分配对状态机电路的复杂度和速度的影响。

例 12.2 与 12.1 的状态分配不同，例 12.2 采用独热编码而例 12.1 则采用 Gray 编码，究竟采用哪一种编码好看看 1 具体情况而定。对于用 FBGA 实现的有限状态机建议采用独热码，因为虽然独热编码多用了两个触发器，但所用组合电路可省些，因而使用电路的速度和可靠性有显著提高，而总的单元数并无显著增加。采用独热编码后有了多余的状态，就有一些不可到达的状态，为此可以用默认项表示不可达到的状态。

2.分别说明和解释例 12.1 和例 12.4 中两种不同赋值（即非阻塞赋值“<=”和阻塞赋值“=”）的用法，和逻辑关系符号“==”的含义。

例 12.1 用“<=”赋值，表示在过程块结束以后才进行赋值是并行结构。例 12.4 用“=”赋值，表示是立即进行赋值是顺序结构。两个例子中作用是一样的，例 12.4 用了多个 always 块，而例 12.1 只用到了一个 always 块，故两种赋值的方式才会不一样。逻辑关系符号“==”是表示判断的符号，用到判断信号是否为符合条件的信号，如 if(a==2'b10)则表示 a 是否为 2'b10.

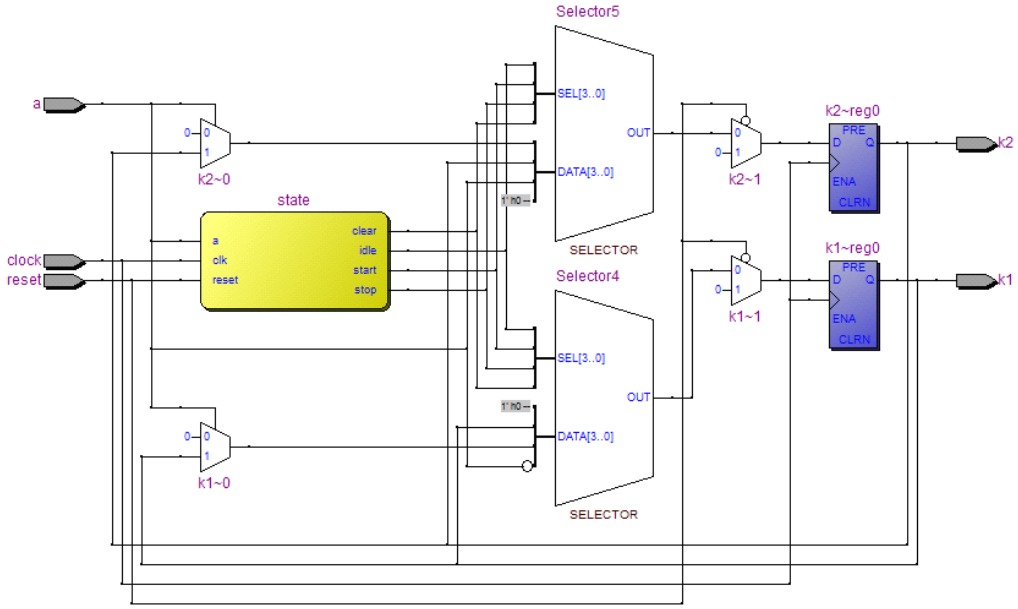
3.一般情况下状态机中的状态变量是用来干什么的？是否可以把状态变量中的某些位指定为状态机的输出，直接用来控制逻辑开关？这样做有什么好处？有什么缺点？

用来表示状态机的几种状态，可以用来对不同状态进行转换。可以直接把状态变量中的某位指定为状态机的输出，直接用来控制逻辑开关。这样做可以提高输出信号的开关速度并节省电路器件。但这种方法的缺点就是开关的维持时间必须与状态机的时间一致。

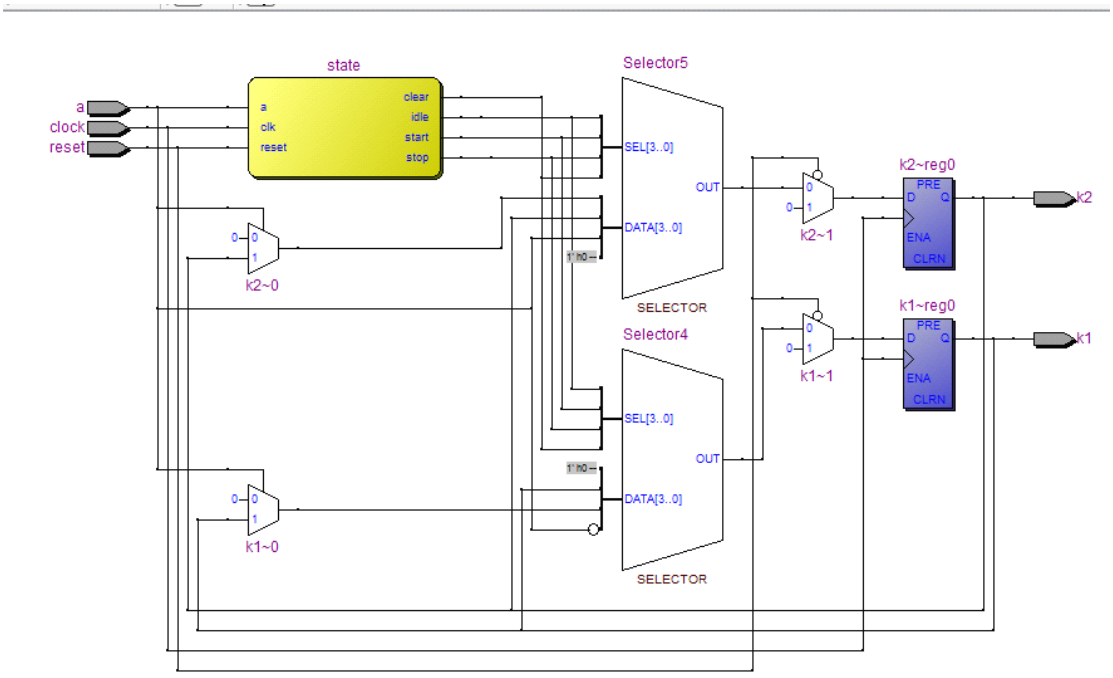
4.分析例 12.1~例 12.4 中用 Verilog 编写的状态机模块，经综合后产生的电路结构中，哪个属于 Mealy 状态机？哪个属于 Moore 状态机？请认真分析及综合出来的电路结构后，给出正确的答案。

从综合后产生的电路结构，可知例 12.1，例 12.2 都属于 Mealy 状态机而例 12.3，例 12.4 属于 Moore 状态机

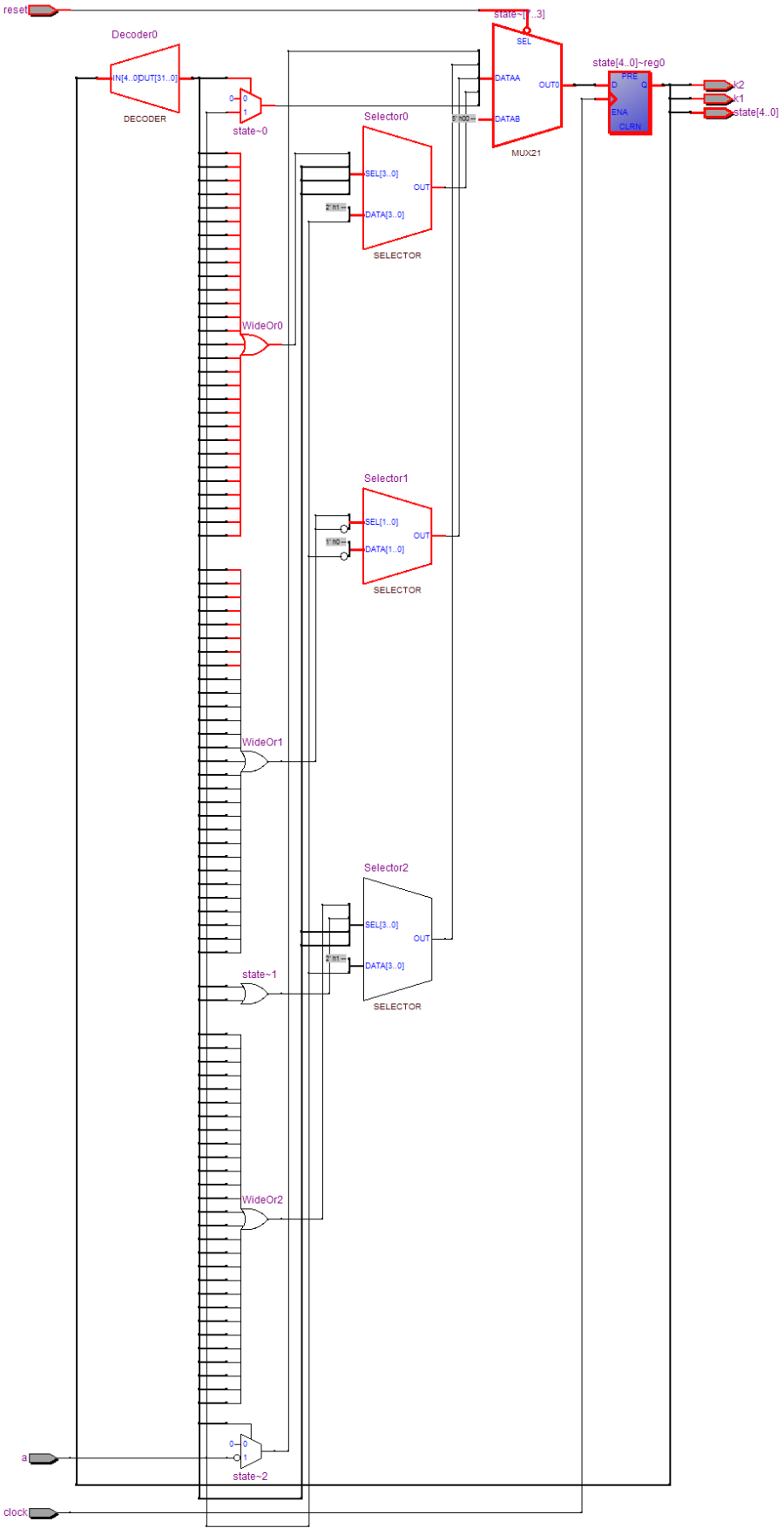
例 12.1



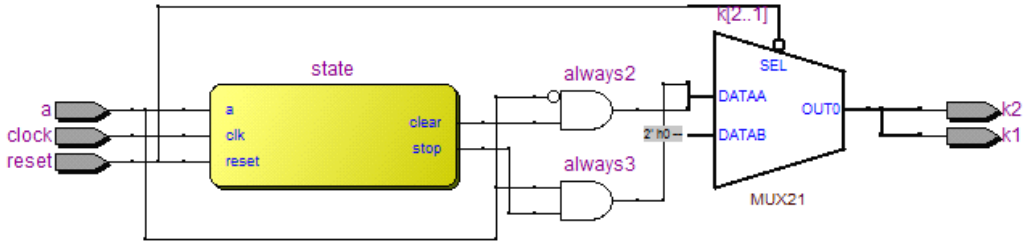
例 12.2



例 12.3


reset
Decoder0
IN[4..0]
OUT[31..0]
DECODER
state~0
state~[1..3]
SEL
DATAA
DATAB
OUT0
MUX21
state[4..0]~reg0
PRE
D
Q
ENA
CLRN
k2
k1
state[4..0]
Selector0
SEL[3..0]
DATA[3..0]
OUT
SELECTOR
WideOr0
Selector1
SEL[1..0]
DATA[1..0]
SELECTOR
WideOr1
Selector2
SEL[3..0]
DATA[3..0]
SELECTOR
state~1
WideOr2
a
state~2
clock

例 12.4



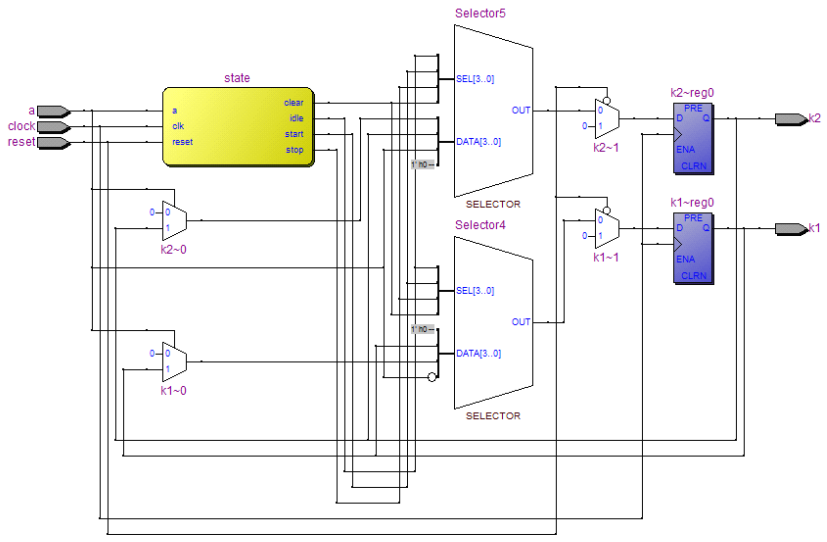
5.如果需要设计带流水线输出的 Mealy 状态机，其 Verilog 模块应该如何编写？请你编写一下，并通过综合器产生电路结构，分析其电路结构和时序。

Verilog 模块：

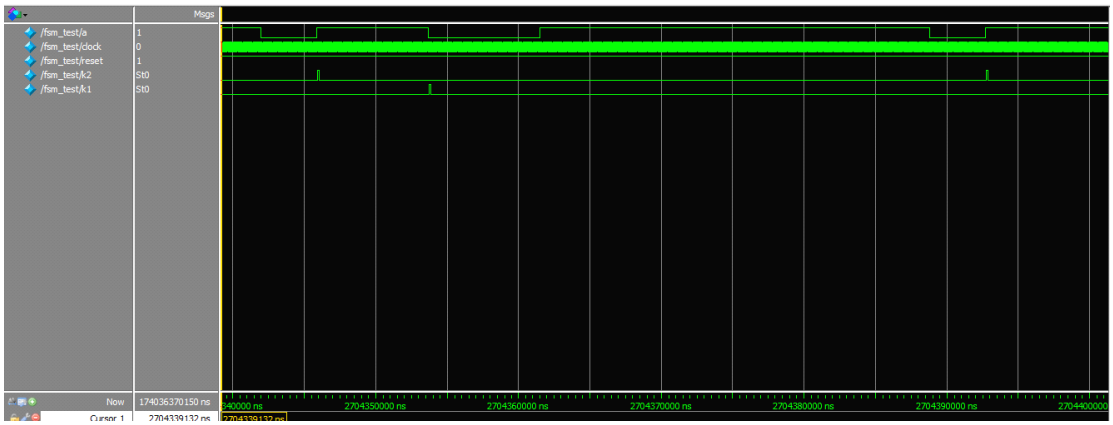
```
module fsm(clock,reset,a,k2,k1);
input clock,reset,a;
output k1,k2;
reg k1,k2;
reg[3:0] state;
parameter idle= 4'b1000,
          start=4'b0100,
            stop= 4'b0010,
            clear=4'b0001;
            always @(posedge clock)
            if(!reset)
              begin
                 state<=idle;
                 k2<=0;
                 k1<=0;
              end
            else
              case(state)
                idle:if(a)
                       begin
                         state<=start;
                         k1<=0;
                       end
                       else
                         begin
                           state<=idle;
                           k2<=0;
                           k1<=0;
                       end
```

```
start:if(!a) state<=stop;
             else state<=start;
stop:if(a) begin
                    state<=clear;
                     k2<=1;
                     end
                   else
                    begin
                       state<=stop;
                       k2<=0;
                       k1<=0;
                    end
clear:if(!a)
             begin
           state<=idle;
           k2<=0;
           k1<=1;
         end
          else
              begin
            state<=clear;
                  k2<=0;
                  k1<=0;
              end
   default:state<=idle;
         endcase
endmodule
```

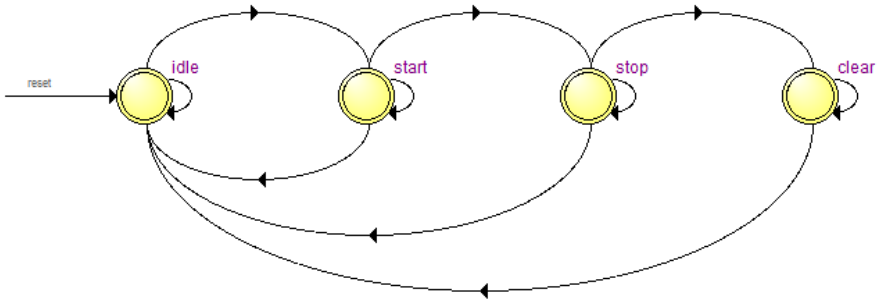
电路结构：



时序：



状态转换图



电路结构有数据选择器构成，输出通过寄存器输出，可构成流水线。通过仿真其时序如上图所示，各状态的转换如图，可知符合要求。

6.在状态机的测试模块中，最后面的 initial 块语句有什么作用，若测试模块中没有最后的 initial 语句块能不能进行仿真？如果能需要注意什么？本测试模块还有什么地方没有测试到？应该如何改进？

和其它模块一起有并行的作用，起到是仿真结束的作用。若测试模块中没有最后的 initial 语句能进行仿真。需要注意仿真是各个信号要对应所测试的模块。本测试模块没有测试到当状态不属于四个状态时，将会怎样。在初始化时可以先将 reset 信号为高阻或 xx 而后再按照该测试模块进行仿真。

第 13 章 设计可综合的状态机的指导原则

1.是不是只要符合 Verilog 语法仿真行为正确的模块都可以综合成电路结构？

不是，异步状态机不能够综合成电路结构。

2.为什么在用 Verilog 设计方法时不采用异步的状态机，采用异步状态机有什么问题不好解决？

因为大多数综合器不能综合采用 Verilog HDL 描述的异步状态机转换为电路网表。异步状态机是没有确定时间的状态机，它的状态转移不是由唯一的时钟跳变沿所触发。采用异步状态机不容易判别触发脉冲是正常的触发还是冒险竞争产生的毛刺。

3.用 always 块语句如何编写纯组合逻辑电路？在哪些情况下会生成不想要的锁存器？

用 always 块设计纯组合逻辑电路时，在生成组合逻辑的 always 块中参与赋值的所有信号都必须有明确地值，然后只要在 always 块中进行赋值就可以了。如果在赋值表达式右端引用了敏感电平列表中没有列出的信号，那么在综合时将会为该没有列出的信号隐含的产生一个透明锁存器。

4.请用清晰的语言把标准的可综合的带同步复位端的同步状态机的样板模块表达出来。

```
 always @(posedge clk)
  begin
     if(reset)
       begin
       /*置输出为 0*/
       end
      else
       begin
         /* 与时钟同步的逻辑*/
       end
end
```

5.请用清晰的语言把标准的可综合的带异步复位端的同步状态机的样板模块表达出来。

```
always @(posedge clk or posedge reset)
  begin
     if(reset)
       begin
       /*置输出为 0*/
       end
      else
       begin
         /* 与时钟同步的逻辑*/
       end
End
```

6.这两种不同的同步状态机有什么不同？如果输入的复位脉冲很窄，哪种状态机

不能可靠复位？

异步复位端的同步状态机的复位与时钟无关。当复位时他们立即置触发器的输出为 0，不需要等到时钟沿到来才复位。带同步复位端的同步状态机的复位只有在时钟的有效跳变沿时刻复位，信号才能使触发器复位。如果输入的复位脉冲很窄，同步复位端的同步状态机不能可靠复位。

7.为什么说，掌握数字电路基础和计算机体系结构这两门学科的真谛是 Verilog 数字系统设计的基础？

当系统比较复杂时，需要通过仔细的分析，把一个具体的系统分解为数据流和控制流，构想哪些部分用组合逻辑，哪些部分的资源可以共享而不影响系统的性能，需要设置哪些开关逻辑来控制数据的流动，需要一个或几个同步有限状态机来正确有序地控制这些开关逻辑，以便有效地利用有限的硬件资源，才能编写出真正有价值的 RTL 级源代码，从而综合出有实用价值的高性能的数字逻辑电路系统。因此，认真的学习并掌握数字电路基础和计算机体系结构这两门学科的真谛是 Verilog 数字系统设计的基础。

8.如果一定要设计异步触发的计数电路，用 Verilog 描述有什么办法？能否综合？仿真时要注意什么问题？

编写另一个模块，在那个模块中使用另外一个时钟；然后用实例引用的方法在另一个模块中把它连接起来。能够综合。仿真时应尽量使这两个状态机的时钟有一定的联系。

9.把本章中的例题编写，没有编写测试模块的编写相应的测试模块，并在 Verilog 仿真环境下运行，一全面验证设计的正确。

第 14 章 深入理解阻塞和非阻塞赋值的不同

1.用带电平敏感列表触发条件的 always 块表示组合逻辑时，应该用哪一种赋值？

应该用阻塞赋值

2.用带时钟沿触发条件的 always 块表示时序电路时，应该用哪一种赋值？

应该用非阻塞赋值

3.为什么不能在多个 always 块中为同一变量赋值？

多个 always 块中为同一个变量赋值可能会导致竞争冒险即使使用非阻塞赋值也可能产生竞争冒险。

4.为什么不能用$display 系统任务来显示用非阻塞赋值的变量值？

因为非阻塞语句的赋值在所有的$display 命令执行以后才更新数据。

5.$strobe 和$display 这两个显示用系统任务有什么不同？各用于什么场合？

$display 命令的执行是安排在活动事件队列中，但排在非阻赋值赋值数据更新事件之前；$strobe 命令的执行是排在非阻塞赋值数据更新事件之后。$display 适合用来显示阻塞语句的赋值，$strobe 适合用来显示非阻塞语句的赋值。

6.仿真器在处理阻塞和非阻塞赋值操作队列过程中有什么不同？

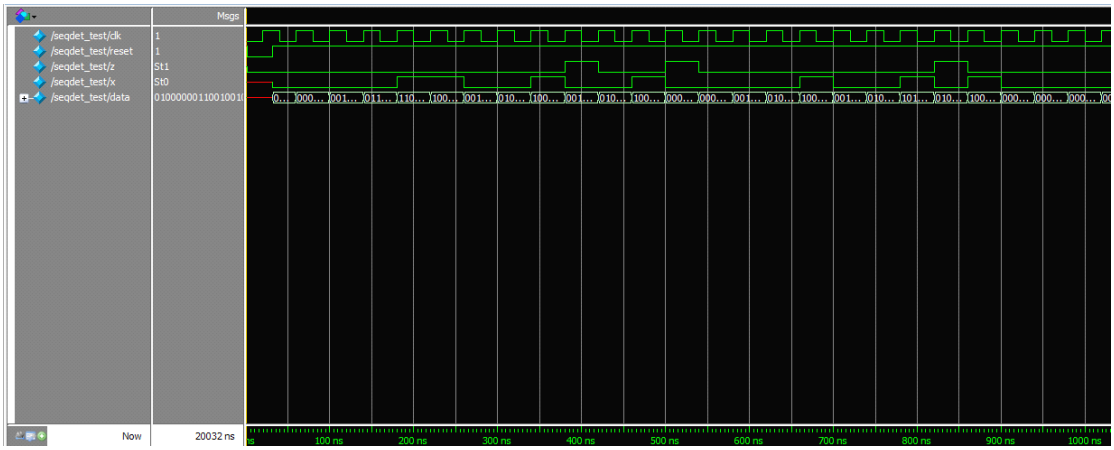
阻塞赋值是由动态事件队列调度的，而非阻塞赋值不是由动态事件队列调度。

7.为什么在可综合 Verilog 模块的设计中，必须注意并遵守本章的 8 条原则？

因为遵循本章的 8 条原则有助于正确的编写可综合硬件，并且可以消除 90%~100%在仿真时可能产生的竞争冒险现象。

第 15 章 较复杂时序逻辑电路设计实践

1. 在计算机上对例 15.1 进行 RTL 级仿真和综合后的门级 Verilog 网表仿真，观测两种层次的仿真输出波形有什么不同，试着说明为什么不同，分析实际电路的波形应该如何？



2. 把例 15.1 改写成能对 111000101 序列进行检测，在接下去的检测中不再考虑已经检出序列中的任何位，编写完整的 Verilog 程序，并进行综合后的门级 Verilog 网表仿真，当出现以上序列时让 Z 的高电平维持两个节拍。

3. 把例 15.2 改写成能对两位并行数据处理的电路，并把并行数据转换为符合协议的串行数据，来控制 4 条输出线的电平。编写完整的 Verilog 程序，并进行综合后的门级 Verilog 网表仿真，验证设计的正确性。

4. 把例 15.2 改写成能对三位并行数据处理的电路，把三位并行数据转换为符合协议的串行数据，来控制 8 条线的电平，但要求并转换模块能在 rst 信号有效后，发出 ack 信号到信号模块，然后由信号源模块发出数据(data[2:0])，编写完整的 Verilog 程序，并进行综合后的门级 Verilog 网标仿真，验证设计的正确性，并编写出实验总结报告。

5. 例 15.2 中的 out16hi 模块布局布线后仿真如果要输出正确结果，为什么必须使从上一个模块 ptosda 输出的 sda 高阻状态变为一个固定的输出后才有可能得到结果，否则出没有正确的电平输出，而 RTL 级仿真和综合后的 Verilog 网表仿真却能得到正确的答案？请从该仿真运行中发现的问题写出必须进行布线后仿真的几个重要因素。

第 16 章 较复杂时序逻辑电路设计实践

1.什么是同步状态机？
2.设计有限同步状态机的一般步骤是什么？
3.为什么说把具体问题抽象成嵌套的状态机的思考方式可以处理极其复杂的逻辑关系？
4.为什么要用同步状态机来产生数据流动的开关控制系列？
5.什么是 HDL RTL 级的描述方式？它与行为描述方式有什么不同？
6.什么是综合？为什么要编写可综合模块？
7.在设计中可综合模块和行为模块的作用分别是什么？
8.可综合的 Verilog HDL RTL 级的描述方式的样板是什么？
9.用 RTL 级描述方式的 Verilog HDL 模块是否都能综合？保证能综合的要点是什么？
10.可综合的 Verilog HDL RTL 级模块的编写中，用阻塞赋值和非阻塞赋值的原则是什么？
11.保证可综合模块 RTL 和布线后仿真一致性的关键是什么？
12.读懂本讲中的例子，如 EEPROM 读写器的设计，并改写 Verilog 模块，使得它不知能随机读写还能进行连续的页面读写，还改写成其他虚拟模块进行 RTL，门级网表和布线后仿真。

第 17 章　简化的 RISC_CPU 设计

1.请叙述设计一个复杂数字系统的步骤。
2.综合一个大型的数字系统需要注意什么？
3.改进本章中的 RISC_CPU 系统，把指令数增至 16，寻址空间降为 4KB，并书写设计报告，实现三个层次的仿真运行。
4.什么叫软硬件联合仿真？为什么说 Verilog 语言之处软硬联合设计？

第 18 章 虚拟器/接口、IP 和基于平台的设计方法及其在大型数字系统设计中的作用

1.为什么要设计虚拟模块？
2.虚拟模块有几种类型？
3.为什么在 ASIC 设计中要尽量利用商业化的虚拟模块和 IP 技术？
4.为什么说编写完整精确地虚拟模块，编写者不但需要全面熟悉地掌握 Verilog 语言，还要需要高度的责任性，并且需要有一个严格的质量保证体系来确保与工艺的电路的一致性？
5.什么是基于平台的设计方法学，为什么该设计方法具有最高的设计效率？

后记

由于最后四章篇幅比较大，就没有一一作答，如有疑问可与作者联系。本文档仅是代表作者的观点，由于作者水平和精力有限，本文档难免有错误和疏忽之处，欢迎大家指正。转载请说明出处，谢谢！

Email：caoxiaoliangzdh@sina.com

天之蓝电子工作室
曹晓亮
2012-8-29