第5章 仿真测试文件

CHAPTER 5

测试文件用来在仿真软件环境下测试设计电路的功能是否符合设计要求。编写测试文件最主要的工作就是设计激励信号，也就是需要测试电路模块输入端子的信号，通过给被测试电路输入端子特定的激励信号，观察被测电路的输出是否满足设计要求，以此来辅助和验证电路设计。这里介绍 ModelSim 仿真软件支持下的测试文件的设计。

5.1 测试文件结构

由 3.2.3 节的介绍可知，Quartus Prime 软件可以辅助进行测试程序的编写。在电路设计综合完成后，通过菜单命令 Processing→Start→Start testbench template writer，可以生成一个与项目顶层文件同名的 testbench 测试文件模板。

可以看到，此测试电路模块没有外部端子，模块名为 myexam_vlg_tst，内部包含了 3 个主要部分：信号定义、实例化、施加激励。施加激励通过 initial 模块和 always 模块实现，设计者需要根据测试需求，设计需要的激励信号，其中 initial 模块用于产生执行一次的激励信号，如复位信号、非周期性输入信号等；always 模块用于产生由敏感事件列表触发的信号，如时钟信号、周期性输入信号等。

```
`timescale 1 ps/ 1 ps
module <测试电路模块名>( );                    //测试电路没有外部端子
constant                                       //常量说明
reg                                            //通用寄存器信号说明
wire                                           //net 型变量说明
reg     //被测电路输入信号说明,输入信号数据类型定义为 reg

<被测电路模块名> <被测电路模块例化名>(                    //被测电路模块实例化
//端口映射表
    .被测模块端口  (测试模块信号),
    .被测模块端口  (测试模块信号),
    …
    .被测模块端口  (测试模块信号)
);

//一次性激励信号产生模块 initial
initial
begin
```

```
    //只需要执行一次的程序代码
    $display("Running testbench");
end

//周期性激励信号产生模块 always
always @(敏感信号列表)                    //敏感信号列表是否存在可选
begin
    //需要在敏感信号激励下执行的程序代码
end
assign                                    // 功能描述语句
endmodule
```

5.2 `timescale 指令

该指令用于指定测试程序仿真时的时间单位和时间精度,时间单位是模拟时间和延迟值等时间值的度量单位。如果测试程序中没有该指令,则 ModelSim 仿真器采用默认的时间单位 1ps。可以通过系统任务 $printtimescale 显示测试程序当前使用的时间单位和时间精度。

例 5.1 时间单位仿真验证

```
`timescale 10 ns / 1 ns
module test;
reg set;
initial begin
    #1 set = 0;              //经过 1 个时间单位,即 10ns,set 值赋为 0
    #2 set = 1;              //经过 2 个时间单位,即 20ns,set 值在 30ns 处赋为 0
    $printtimescale;
end
endmodule
```

此段代码的执行结果如图 5-1 所示。

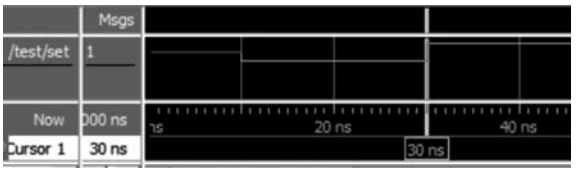

图 5-1 时间单位仿真验证

同时在 ModelSim 的 Transcript 窗口中显示如下信息:

```
# Time scale of (test) is 10ns / 1ns
```

5.3 initial 语句

initial 语句面向模拟仿真过程,不能被综合。initial 语句通常用来对信号进行初始化,或者产生一次性变化的信号。initial 语句格式如下:

```
initial
    begin
      语句 1;
      语句 2;
      …
    end
```

例如，

```
initial
    begin
    data = 0;
    for(addr = 0;addr < size;addr = addr + 1)
        memory[addr] = 0;
end
```

这段代码用 initial 语句在仿真开始时对各变量进行了初始化，data 赋值为 0，并且初始化了一个空间为 size 大小的存储器 memory。

通常可以用 initial 语句生成一系列特定的激励信号，如例 5.2 所示。

例 5.2　激励信号生成

```
module test;
reg[3:0] a;
initial
    begin
        a = 4'h0;
        #10 a = 4'h1;
        #10 a = 4'h2;
        #20 a = 4'h3;
        #10 a = 4'h0;
    end
endmodule
```

实现效果如图 5-2 所示，生成了一个随时间变化的信号 a，且 begin-end 块内每条语句的延迟是累加的。由此可知，块内的语句是按照顺序执行的，只有上面一条语句执行完后下面的语句才能执行；每条语句的延迟时间都是相对于前一条语句的仿真时间而言的；直到最后一条语句执行完，程序流程控制才跳出该语句块。因此我们称 begin-end 块为顺序块。

图 5-2　begin-end 顺序块生成激励信号

例 5.3　激励信号生成

```
module test;
reg[3:0] a;
initial
```

```
    fork
        a = 4'h0;
        #10 a = 4'h1;
        #10 a = 4'h2;
        #20 a = 4'h3;
        #30 a = 4'h0;
    join
endmodule
```

实现效果如图 5-3(a)所示,可见 fork-join 块内每条语句的延迟是独立的。由此可知,块内的语句是并行关系,块内每条语句的延迟时间都是相对于程序流程进入块内的时刻。因此,称 fork-join 块为并行块。当对同一信号赋值时,若延迟时间不同,则语句的顺序可以随意调换;但是若对同一信号赋值,且延迟时间相同,则实际起作用的是最后一条信号赋值语句。例如,对 fork-join 语句做如下修改,则实现效果如图 5-3(b)所示,信号 a 在 10ns 时的值由 4'h2 变为了 4'h1。

```
fork
    a = 4'h0;
    #30 a = 4'h0;
    #10 a = 4'h2;
    #10 a = 4'h1;
    #20 a = 4'h3;
join
```

(a) (b)

图 5-3 fork-join 并行块生成激励信号

5.4 always 语句

always 语句在测试文件中用于产生在一定条件下变化的输入信号,具体描述方式与 2.4.3 节中介绍的相同。这里主要介绍如何用 always 语句生成时钟信号。always 语句生成时钟信号的描述如下:

```
always #20 clk = ~clk;
```

此语句设置每隔 20 个时间单位,信号 clk 翻转一次。若时间单位为 1ns,则 clk 信号的周期为 40ns。

5.5 系统函数

Verilog HDL 的系统任务和函数分为 10 个类别:显示任务、文件输入/输出任务、时标任务(timescale tasks)、仿真控制任务、PLA 建模任务、随机分析任务、仿真时间函数、转换

函数、概率分布函数、命令行输入。这里只介绍常用的任务和函数。

1. $display

$display 用于输出信息，信息显示在 ModelSim 的 Transcript 窗口中。$display 任务的格式如下：

```
$display(p1,p2,…,pn);
```

其中，参数 p1 用于格式控制，格式控制符如表 5-1 所示。参数 p2～pn 为要显示的信息，又称输出列表，即将参数 p2～pn 按参数 p1 给定的格式输出。

表 5-1　格式控制符

格　　式	含　　义
%h 或%H	以十六进制的形式输出
%d 或%D	以十进制的形式输出
%o 或%O	以八进制的形式输出
%b 或%B	以二进制的形式输出
%s 或%S	以字符串的形式输出
%t 或%T	以当前的时间格式的形式输出
%c 或%C	以 ASCII 码字符的形式输出

$display 输出显示后自动换行，如：

```
$display("a = %d, b = %h", 1, 2);
$display("%b", 3);
```

执行结果如图 5-4 所示，默认数据长度为 32bit。

```
# a=           1,b=00000002
# 00000000000000000000000000000011
```

图 5-4　显示结果

除 $display 任务外，还有如下 3 个任务：$displayb、$displayh、$displayo，分别表示以二进制、十六进制和八进制显示数据。如“$displayb(1,2);”的执行结果为：

```
# 00000000000000000000000000000001000000000000000000000000000000010
```

2. $write

$write 和 $display 的功能相同，也是用于在 ModelSim 的 Transcript 窗口中显示信息。$write 任务的格式如下：

```
$write(p1,p2,…,pn);
```

其中，参数 p1 用于格式控制，参数 p2～pn 为要显示的信息。$write 也有另外 3 个任务：$writeb、$writeh、$writeo，分别表示以二进制、十六进制和八进制显示数据。

$write 和 $display 的区别在于输出显示后不会自动换行，例如，

```
$write("a = %d, b = %h", 1, 2);
$write("%b", 3);
```

执行结果为：

```
a =      1,b = 0000000200000000000000000000000000000011
```

3. $fopen

$fopen 的功能是打开文件。$fopen 使用格式有两种：

(1) $fopen("<文件名>");

(2) <文件句柄>= $fopen("<文件名>", < type >);

type 为打开文件的类型，如表 5-2 所示。type 类型可以省略，默认以“w”的方式打开文件。注意，以此方式打开文件，若文件原来并不存在，则 $fopen 会先创建此文件再打开；若文件存在，则文件会被清空。

若文件名中有路径，则文件路径间隔为“/”而不是“\”。例如，

```
h1 = $fopen("D:/example/file1.txt");
```

表 5-2 文件打开类型

参 数	描 述
"r"或"rb"	以读的方式打开文件
"w"或"wb"	将文件清空或创建文件，并以写的方式打开文件
"a"或"ab"	以追加写的方式打开文件；若文件不存在，则先创建文件
"r+"、"r+b"或"rb+"	打开以更新(读/写)文件
"w+"、"w+b"或"wb+"	将文件清空或创建文件，以进行更新(读/写)
"a+"、"a+b"或"ab+"	打开或创建一个以追加写方式的文件

4. $fdisplay

$fdisplay 将数据写入指定的文件中，使用格式如下：

```
$fdisplay(<文件句柄>,"写入内容");
```

例如，

```
integer h1;
h1 = $fopen("file1.txt");                    //取一个文件的句柄
$fdisplay(h1,"This is a test.");             //将数据写入文件
```

如果写入内容为双引号括起来的信息，则作为字符串写入，否则作为数据写入。

除 $fdisplay 任务外，还有如下 3 个任务：$fdisplayb、$fdisplayh、$fdisplayo，分别表示将整型数以二进制、十六进制和八进制的方式写入文件，且数据长度为 32bit，对整型数以外的数据无效。例如：

```
$fdisplayb(h1,10);
$fdisplayb(h1,10.2);
```

写入文件中的内容显示如下：

```
00000000000000000000000000001010
10.2
```

5. $fwrite

$fwrite 将数据写入指定的文件中，使用格式如下：

```
$fwrite(<文件句柄>,"写入数据");
```

$fwrite 和 $fdisplay 的区别在于 $fdisplay 写完就会自动换行，$fwrite 不会换行。

6. $fclose

$fclose 的功能是关闭文件，使用格式如下：

```
$fclose(<文件句柄>);
```

7. $readmemb

$readmemb 用于读取二进制数据文件内容到存储器，此系统任务常用来初始化一个内存空间。使用格式有如下 3 种：

(1) $readmemb("<数据文件名>",<存储器名>);

(2) $readmemb("<数据文件名>",<存储器名>,<起始地址>);

(3) $readmemb("<数据文件名>",<存储器名>,<起始地址>,<终止地址>);

例 5.4 假设文件 file1.dat 的内容如下：

```
01010000
1000_1010
1010 zzxx11xx
```

若要将此文件的内容读至存储器 memory 中，程序代码如下：

```
`timescale 10ns/1ns
module test;
reg[7:0] memory[0:3];                    //申请 4 个 8bit 空间的存储单元
integer n;
initial
    begin
        $readmemb("file1.dat",memory);   //读取 file1.dat 中的内容到 memory
    for(n = 0;n < = 3;n = n + 1)         //显示读取到的 4 个存储单元的内容
        $display(" % b",memory[n]);
    end
endmodule
```

程序执行后，Transcript 窗口的显示内容如下：

```
# 01010000
# 10001010
# 00001010
# zzxx11xx
```

可见，$readmemb 是以分行符或空格为间隔符读取文件数据，且忽略下画线，并以地址从低到高的顺序存储至存储器中的。若读取的数据位数少于存储单元的长度，则高位自动补零。

8. $readmemh

$readmemh 用于读取十六进制数据文件内容到存储器，使用格式有如下 3 种：

(1) $readmemh("<数据文件名>",<存储器名>);

(2) $readmemh("<数据文件名>",<存储器名>,<起始地址>);

(3) $readmemh("<数据文件名>",<存储器名>,<起始地址>,<终止地址>);

9. $time、$realtime

$time用于返回整数型的仿真时间，$realtime用于返回实数型的仿真时间。具体返回值取决于`timescale的设置，如下例所示。

例5.5 仿真时间获取

```
`timescale 10ns/1ns
module test;
initial
begin
    #10.2 ;
    $display("time_end = %t", $realtime);
    $display("time_end = %t", $time);
    end
endmodule
```

程序执行后，Transcript窗口的显示内容如下：

```
# time_end =              102
# time_end =              100
```

本例中，时间单位为10ns，时间精度为1ns，当延迟数为10.2时，$realtime返回的值为：10.2×10ns，$time返回值的获得则要将10.2转换为整数10，再乘以时间单位，即：10×10ns。

10. $random

$random函数用于产生随机整数，使用格式有如下两种：

(1) $random%b　　//产生(−b+1)～(b−1)范围内的随机数

(2) {$random}%b　　//产生0～(b−1)范围内的随机数

其中，b为十进制整数，例如，

```
    reg [7:0] rand;
rand = { $random} % 60;
```

可以获得8bit宽的随机数，且数值在0～59范围内。