系统任务

Verilog 中还提供了很多系统任务，类似于 C 中的库函数，使用这些系统任务可以方便地进行测试。由于计组实验中用到的系统任务相对较少，所以在此仅对 `$display`，`$monitor`，`$readmemh` 进行介绍。

输出信息

格式：`$display(p1, p2, ..., pn)`；

这个系统任务的作用是用来输出信息，即将参数 p2 到 pn 按参数 p1 给定的格式输出。用法和 C 语言中的 printf 类似。下面用一个例子简单介绍其用法。

例如：

```
module disp;
    reg[4:0] a;
    reg[4:0] b;
    initial begin
        a = 10;
        b = 20;
        $display("a = %d,b = %d\n",a,b);
    end
endmodule
```

其输出结果为： a = 10,b = 20

其中 `%d` 表示以十进制的形式输出，`\n` 为换行符。

在此说明几种常用的输出格式：

输出格式	说明
%h 或 %H	以十六进制数的形式输出
%d 或 %D	以十进制数的形式输出
%b 或 %B	以二进制数的形式输出
%c 或 %C	以 ASCII 码字符的形式输出

输出格式	说明
%s 或 %S	以字符串的形式输出

监控变量

格式：

- `$monitor(p1, p2, ..., pn);`
- `$monitor;`
- `$monitoron;`
- `$monitoroff;`

任务 `$monitor` 提供了监控和输出参数列表中的表达式或变量值的功能。其参数列表中输出控制格式字符串和输出列表的规则和 `$display` 中的一样。当启动带有一个或多个参数的 `$monitor` 任务时，仿真器则建立一个处理机制，使得每当参数列表中变量或表达式的值发生变化时，整个参数列表中变量或表达式的值都将输出显示。如果同一时刻，两个或多个参数的值发生变化，则在该时刻只输出显示一次。

`$monitoron` 和 `$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。其中 `$monitoroff` 任务用于关闭监控标志，停止监控任务 `$monitor` ， `$monitoron` 则用于打开监控标志，启动 `$monitor` 监控任务。 `$monitor` 与 `$display` 的不同处还在于 `$monitor` 往往在 `initial` 块中调用，只要不调用 `$monitoroff` ， `$monitor` 便不间断地对所设定的信号进行监视。

读取文件到存储器

格式：

- `$readmemh("<数据文件名>", <存储器名>);`
- `$readmemh("<数据文件名>", <存储器名>, <起始地址>);`
- `$readmemh("<数据文件名>", <存储器名>, <起始地址>, <结束地址>);`

功能： `$readmemh` 函数会根据绝对/相对路径找到需要访问的文件，按照 ASCII 的解码方式将文件字节流解码并读入容器。文件中的内容必须是十六进制数字 0~f 或是不定值 x，高阻值 z（字母大小写均可），不需要前导 0x，不同的数用空格或换行隔开。假设存储器名为 arr，起始地址为 s，结束地址为 d，那么文件中用空格隔开的数字会依次读入到 arr[s],arr[s+1]... 到 arr[d]。假如数字的位数大于数组元素的位数，那么只有低位会被读入，剩下的高位会被忽略。

此系统任务用来从文件中读取数据到存储器中，类似于 C 语言中的 fread 函数。

例如：

```
module im;
    reg [31:0] im_reg [0:2047];
    initial begin
        $readmemh("code.txt", im_reg);
    end
endmodule
```

仿真后即可将 code.txt 中的内容读入 `im_reg` 存储器中。

一处仿真工具间的差异

需要特别注意的是，表示存储器数量的中括号内的范围如果是从高到低的，例如 `reg [31:0] img_reg [2047:0]` ，则采用 `$readmemh("<数据文件名>", <存储器名>, <起始地址>);` 这种格式进行读入会在 ISE 和 VCS 上会得到不同的结果：在 ISE 中读入的数据从起始地址依次向低下标填充，而 VCS 中读入的数据从起始地址依次向高下标填充。

例如以下代码片段，读取文件后按下标顺序输出存储器的内容：

```
reg [7:0] mem [3:0];
integer i;

initial begin
    $readmemh("code.txt", mem, 1);
    for (i = 0; i < 4; i = i + 1) begin
        $display("%1d: %02x", i, mem[i]);
    end
end
```

数据文件 `code.txt` 内容为：

```
01
02
```

这段代码在 ISE 和 VCS 中分别仿真会得到不同的结果，其中 ISE 仿真输出为：

```
0: 02
1: 01
2: xx
3: xx
```

而 VCS 仿真输出为：

```
0: xx
1: 01
2: 02
3: xx
```

要避免上述差异带来的影响，请按从低到高的范围声明存储器的数量，例如上述代码修改为 `reg [7:0] mem [0:3]` ；或在使用从高到低范围时显式给出 readmemh 的结束地址，例如 `$readmemh("code.txt", mem, 1, 3)` ，其中结束地址可以超过实际填充了数据的地址，只需大于起始地址。

fsdb 相关参数（仅 VCS）

在 VCS 中，默认情况下，为了加快仿真速度，仿真器不会记录任何波形信号。我们需要使用 `$fsdbDumpvars` 系统任务，告知仿真器我们需要记录哪些信号。

该命令最简单的用法，只需一行，告知系统我们需要所有波形。我们计组的设计较小，导出所有波形也不会影响仿真速度，因此使用这个命令就足够。

```
initial begin
    $fsdbDumpvars();
end
```

如果设计较复杂（例如有数百、数千个模块），记录所有波形会消耗较多性能，拖慢仿真速度。我们可以为该系统任务增加参数，只记录部分模块的波形，语法如下（其中括号内的竖线表示左右的写法选择一种，三个参数均为可选参数）：

```
$fsdbDumpvars(
    [depth, | "level=",depth_var,]
    [instance, | "instance=", instance_var]
    [, "option"| ,"option, option_var"]
);
```

第一个参数指定要导出的深度：

- depth=0: 导出所有信号
- depth=1: 导出当前模块的信号
- depth=n: 导出当前模块和向下 n-1 级的信号

第二个参数指定要导出的模块名称；第三个参数是额外的选项。例如下面三种写法都是可以的：

```
initial begin
    $fsdbDumpvars(0, system);
    $fsdbDumpvars(0, system, "+fsdbfile+novas.fsdb");
    $fsdbDumpvars(1, top.dut.u1);
end
```

除了 `$fsdbDumpvars` 指定要记录波形的范围外，我们也可以使用 `$fsdbDumpon` 和 `$fsdbDumpoff` 动态开关波形的记录。例如：

```
initial begin
    $fsdbDumpvars();

    $fsdbDumpoff();
    #200;
    $fsdbDumpon();
    #300;
    $fsdbDumpoff();
    #100;
    $fsdbDumpon();
end
```