CH3 Systemverilog HDL

一、引言

1. systemverilog程序

$.sv \rightarrow Synthesizer$（综合器）→硬件电路网表文件

2. 逻辑综合

RTL描述→翻译→未经优化的中间表示→逻辑优化→实现与布局布线（设计约束、工艺库）→优化的门级网表

3. Systemverilog HDL程序的基本结构

```
module mux2(input logic D0, D1, sel, output logic y);
endmodule
```

4. 端口

```
input [类型] [位宽] 端口名1, 端口名2, ...;
output [类型] [位宽] 端口名1, 端口名2, ...;
input [类型] [位宽] 端口名1, 端口名2, ...;
```

5. 内部变量

用于对模块内部的各个子模块或逻辑门之间进行连接

```
[类型] [位宽] [名字]
logic [7:0] a;
```

6. 逻辑模块

> 行为建模：描述输入和输出之间的因果关系
>
> - 持续赋值语句
> - 过程块
>
> 结构建模：调用其他已经定义过的模块对整个电路的功能进行描述

7. 程序模板

```
module module_name(port1, port2, ...)
    端口类型说明(optional);
    内部变量定义(optional);

    实例化引用低层次模块;
    持续赋值语句;
    过程块语句;
endmodule
```

二、语法要素

1. 间隔符和注释

间隔符：\b \t \n以及换页符

不出现在字符串中的间隔符直接忽略。

注释：// /**/

2. 标识符和关键字

标识符：[a-z|A-Z|_]+[a-z|A-z|0-9|$|_]*

关键字：又称保留字，无法作为标识符使用

3. 逻辑值

0、1、X(x)、Z(z)

4. 常量

整数型常量：<+/-> <位宽>'<进制><数值>，除了数值，其余都可以缺省。

不带负号：EDA工具按无符号数处理（即电路中的位表示）[指定位宽和进制时]

带负号：EDA工具按有符号补码处理

没有给出位宽时，按照当前变量位宽处理。多的部分忽略，少则高位补0

没有给出进制，默认按十进制处理，且常量有符号数。

5. 数据类型

变量：

变量类型：单源驱动，持续驱动（综合成连线）/值保持（综合成寄存器）

线网类型：可以多源驱动，综合成连线。必须持续驱动。

主要变量类型：

> logic(1), bit(1), byte(8), shortint(16), int(32)等（除logic，其余变量的二进制单元非0即1）
> logic [signed] [位宽] 信号1，.....
>
> logic可以进行全选或域选
>
> byte, shortint, int无法进行域选
>
> wire（被弃用，但端口信号默认为wire）
>
> tri（多源驱动信号）

6. 运算符

```
~ & | ^ ~& ~| ~^ ~^ // 除~，其余都为双目运算

& | ^ ~^             // 缩减运算，对操作数的所有位以此进行位运算，结果只有1位

{a, {a, b}, 10{a}, c[1:0], 2'b01};  // 拼接

```

```
+ - * / %    // 算术运算

<< >> <<< >>>    // 逻辑、算术移位运算

< > <= >= == != // 相等运算

! && || // 逻辑运算

? :      // 条件运算

++       // 支持自增运算
```

三、建模方法

1. 行为建模

- 持续赋值语句（assign）
- 过程块（always initial）

持续赋值语句：assign [#延迟量] 信号名 = 表达式（延迟量用于仿真，不可综合！！）

过程块建模：always_comb begin ... end

使用阻塞赋值（按顺序执行），可以使用if else和case语句。（分支尽量完整，避免出现锁存器，解决办法：default）

循环结构：for, repeat（指定循环次数）, while, forever（无限循环）（for可综合）

2. 结构化建模

也被称为**层次化建模**。

位置关联法：module_name object_name (port1, port2, ...)

名称关联法：module_name object_name (.P1(port1), .P2(port2), ...)

二者不可混用。

3. 门级建模

使用封装的**基本逻辑元件**来描述电路。（实例名可以忽略）

关键字：and, nand, or, nor, xor, xnor, buf, not, bufif1, bufif0, notifi1, notif0

4. 参数化建模

```
`define WIDTH 8          // 无需分号结尾
`include "define.sv"
parameter WIDTH = 8;
```

四、测试程序

1. 模板

```
module test_bench_name();        // 无需参数表
    // 信号定义
    // 模块实例化
    // 添加激励信号
    // 显示输出结果
endmodule
```

2. 激励信号

- initial
- always
- 文件

3. initial过程块

```
initial begin
   // 赋值    #10
   // 赋值    #15
   // ...
end
```

多个initial块并行执行，不能在多个initial块中，在同一仿真时刻对同一信号进行赋值。

4. 文件激励

```
logic [2 : 0] stim [7 : 0];
logic a, b, c;
initial begin
    $readmemb("data.txt", stim);    // 将所有激励读入数组
    for(int i = 0; i < 10; i ++) begin
        {a, b, c} = stim[i];    #10
    end
end
```

5. 输出响应

- 获取仿真时间：$time
- 显示信号值：$display
- 结束/中断仿真：$finish, $stop
- 文件输入：$readmemb, $readmemh
- 文件输出：$fopen, $fclose, $fdisplay, $fmonitor

6. 获取仿真时间

使用`timescale所设置的时间单位。

- $time：返回64位整数时间值
- $stime：返回一个32位整数时间值
- $realtime：返回一个实数时间值

$monitor($time, "a = %b b = %b c = %b y = %b", a, b, c, y);

7. 显示信号值

$display("format_string", port1, port2, ...);　　执行到该语句时才打印

$montitor("format_string", port1, port2, ...);　监视器，输出变量列表中某个变量改变时打印

8. 设置时间格式

$timeformat(units_number, precision_number, suffix_string, minimum_field_width);

units_number：0到-15之间的整数值，0表示秒，-3表示毫秒，-6表示微秒，-9表示纳秒。

precision_number：打印时间的保留位数（小数点后）。

suffix_string：打印时间值后面打印的一个后缀字符串

minimum：是时间值与后缀字符串合起来的字符串的最小长度，默认20.

$timeformat(-9, 2, "ns", 14);

9. 结束/终止仿真

$finish;

$finish(n);

$stop();

$stop(n);

n : "0"→不输出任何信息，"1"→给出仿真时间

10. 文件输入

$readmemb：读2进制数据

$readmemh：读16进制数据

$readmemb("file_name", array, begin_addr, end_addr); 起始地址可缺省

使用空格或换行区分的那个数据。

文件中：@hex_addr data（十六进制地址，数据）可将数据读入特定地址中

11. 文件输出

$fopen：打开文件，返回文件句柄

$fclose：关闭文件，传入文件句柄

$fmonitor(file_handler, "format_string", port1, ...)（fdisplay同理）

五、FPGA

1. 概念

一种基于**查找表**而不是逻辑门来实现组合逻辑电路的高度**可重构（可编程）**阵列。

2. 特点

- 可编程器件，半定制电路，“万能芯片”
- 可实现包括组合逻辑电路、时序逻辑电路在内几乎所有类型的数字逻辑电路
- 设计灵活，成本低

3. FPGA开发流程

- 逻辑综合
- 仿真验证