

数字电路实验

Lecture 2: Verilog 语言与Quartus入门

南京大学 计算机科学与技术系

2020年秋季

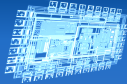


目录



① Verilog HDL 简介

② Quartus 入门



- 硬件描述语言HDL
(Hardware Description Language)

协助硬件设计者在较高层次上对电路进行设计、模拟以及综合

- 常见硬件描述语言

- Verilog HDL
- VHDL
- ABEL



● 基本工具

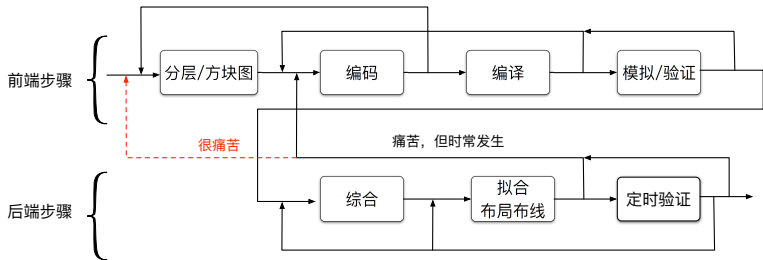
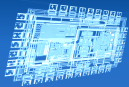
- 文本编辑器 (Text Editor)
- 编译器 (Compiler)
- 综合器 (Synthesizer)
- 模拟器 (Simulator) : Test Bench及波形编辑器

● 扩展工具

- 模板生成器 (Template generator)
- 原理图展示器 (Schematic Viewer)
- 翻译器 (Translator)
- 定时分析器 (Timing Analyzer)
- 后插注解器 (Back Annotator)



HDL 设计过程





Verilog语言入门



- 基本单元—模块 (module)
- 模块的实例化

模块定义

```
module 模块名(端口参数1,端口参数2, ...);  
    端口参数说明(input, output, inout);  
    局部参数说明(wire,reg);
```

模块功能语句;

```
endmodule;
```

示例（禁止门）

```
module VrInhibit (X, Y, Z);  
    input X,Y;  
    output Z;  
  
    assign Z = X & ~Y;  
  
endmodule
```



Verilog中的信号



- 网格(net): 类似物理连线
 - wire
 - tri
 - ...
- 变量(Variable): 程序执行中存储数据, 无物理意义
 - reg: 不是触发器
 - integer: 整数, 取决于模拟器字长
- 信号宽度
 - 定义: `wire [msb:lsb] identifier;`
 - 位选择: `a[1:8]` – a 的左 8bit



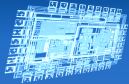
Verilog中的常量



- 定义方式: $n'Bdd\dots d$, $\langle\text{位宽}\rangle'\langle\text{进制}\rangle\langle\text{数字}\rangle$
 - $10'b1010101010$ – 10bit二进制数1010101010
 - $8'd42$ – 8bit十进制数42
 - $16'hFFFF$ – 16bit十六进制数FFFF
- **parameter常量**: 定义模块内部的命名常量
 - `parameter BUS_SIZE=32;`



Verilog 的逻辑系统与操作



逻辑系统

信号	含义
0	逻辑0或假
1	逻辑1或真
x	未知逻辑值
z	三态中高阻

逐位布尔操作

操作符	含义
&	与
	或
^	异或
~, ~^	同或
~	非

算术移位操作

操作符	含义
+	加
-	减
*	乘
/	除
%	取模
<<	向左移位
>>	向右移位



● 条件操作符 ?

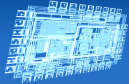
- 形式: $X? Y: Z$
- 例: $f = (A > B)? A: B;$
 -- 取 $\max(A, B)$

逻辑操作

操作符	含义
&&	逻辑与
	逻辑或
!	逻辑非
==	逻辑相等
!=	逻辑不等
>	大于
>=	大于等于
<	小于
<=	小于等于



Verilog数组及连接操作符



● 数组

- 定义

```
reg identifier [start:end];
```

- 例如:

```
reg [7:0] mem_a[1:16];
```

16个8bit reg型变量

```
mem_a[12][3:0]
```

mem_a的12号单元的低四位

● 连接操作符: {}

- 当需要将若干个向量或者位合并成新的向量时，可以使用连接操作符

- {x,...,y}

将{x,...,y}各向量首尾相接

- {n{x}}

将x重复n次



Verlog基本设计模式

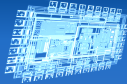


- 结构式设计
等效于描述电路逻辑原理图
- 数据流式设计
用连续赋值语句来描述电路功能
- 行为式设计
用always程序块来实现过程式语句

always程序块外的语句是并行执行的！



结构式程序设计



- 实例化组件，描述组件之间的连接
- 内置门: and, nand, xor, or, nor, not, ...

示例（禁止门）

```
module VrInh (in, invin, out);  
    input in, invin;  
    output out;  
    wire notinvin; //中间连线  
  
    not U1 (notinvin, invin); //实例化非门U1，第一参数为输出，第二参数为输入  
    and U2 (out, in, notinvin); //实例化与门U2  
  
endmodule
```



• assign 语句

- assign语句只能给网格信号(如wire)赋值, 不能给变量 (如reg) 赋值
- 并行执行, 顺序无关, 连续执行
- 可以用条件赋值

示例 (素数检测器)

```
module Vrprimd (N, F);  
  input [3:0] N;  
  output F;  
  wire N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0; //中间连线  
  
  assign N3L_N0 = ~N[3] & N[0];  
  assign N3L_N2L_N1 = ~N[3] & ~N[2] & N[1];  
  assign N2L_N1_N0 = ~N[2] & N[1] & N[0];  
  assign N2_N1L_N0 = N[2] & ~N[1] & N[0];  
  assign F= N3L_N0 | N3L_N2L_N1 | N2L_N1_N0 | N2_N1L_N0;  
  
endmodule
```



过程式程序设计



- **always** 程序块
 - **always @** (信号1 or 信号2 or ...)
 - 程序块内顺序执行，可以是begin – end 程序块
- **灵敏度列表 – (信号1 or 信号2 or ...)**
 - 确定**always**程序块何时执行
 - 可以是上升沿 **posedge** 或下降沿 **negedge**
 - 可以缺省设置 **always @ ***
- **always** 内的赋值
 - 只可赋值给变量(如**reg**), 不可赋值给网格信号(如**wire**);
 - 阻塞式赋值 **=**, 立即赋值, 对后续语句有影响
 - 非阻塞式赋值 **<=**, 等整个程序块执行完后赋值

示例 (上升沿触发的D触发器)

```
module VrposDff (CLK, D, Q);  
    input CLK, D;  
    output Q;  
    reg Q;  
    always @ (posedge CLK)  
        Q <= D;  
  
endmodule
```



过程式程序设计 – if, case



if 语句

```
module Vrprimei (N, F);
  input [3:0] N;
  output F;
  reg F;
  parameter OneIsPrime=1;
  always @ (N)
    if (N == 1) F = OneIsPrime;
    else if ( (N % 2) == 0)
      begin
        if (N == 2) F = 1; else F = 0 ;
      end
    else if ( N <= 7) F = 1;
    else if ( (N == 11) || (N == 13) ) F = 1;
    else F = 0;
endmodule
```

case 语句

```
module Vrprimecs (N, F);
  input [3:0] N;
  output F;
  reg F;

  always @ (N)
    case (N)
      1, 2, 3, 5, 7, 11, 13: F = 1;
      default : F = 0;
    endcase
endmodule
```

保证变量在每个路径都赋值，避免推断锁存器



过程式程序设计 –for



- 可以使用for循环
- 需要注意系统是否可以综合
- Verilog 支持 repeat, while, forever; 但建议尽量避免使用



Test Bench设计



- 测试平台用于产生激励，对待测模块进行测试
- 可以使用函数function或任务task
- 系统任务和函数
 - \$display
 - \$write
 - &monitor
 - \$stop
- 时间尺度 `'timescale time-unit/ time-precision`
- initial程序块，模拟开始时执行
- 时延: `#` 延迟数值



Test Bench 示例



```

`timescale 1 ns / 10ps

module Vrprime_tbc() ;
    reg [3:0] Num;
    wire Prime;

    task Check;
        input expect;
        if (Prime != expect)
            $display ("Error: N = %b, expect %b, got %b", Num, expect, Prime);
    endtask

    Vrprimedly UUT( .N(Num), .F(Prime) );

    initial begin : TB
        integer i;
        parameter OneIsPrime = 1;
        for ( i = 1; i <= 15; i + 1); begin
            Num = i;
            case (Num)
                4'd1: Check(OneIsPrime);
                4'd2, 4'd3, 4'd5, 4'd7, 4'd11, 4'd13: Check(1);
                default: Check(0);
            endcase
        end
    end

endmodule

```



Quartus 基本操作



请按照Quartus使用入门文档进行操作

- 1 新建工程
- 2 输入设计文件，并编译
- 3 创建Test Bench进行仿真
- 4 添加引脚约束，生成sof文件
- 5 下载到FPGA进行实际验证