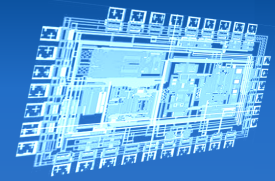
数字电路实验

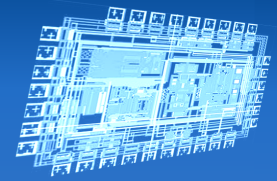
Lecture 2: Verilog 语言与Quartus入门

南京大学 计算机科学与技术系

目录

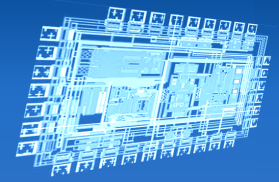


硬件描述语言



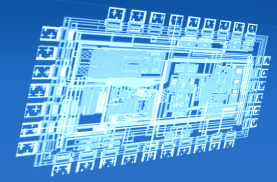
- 硬件描述语言HDL
  (Hardware Description Language)
  协助硬件设计者在较高层次上对电路进行设计、模拟以及综合
- 常见硬件描述语言
  - Verilog HDL
  - VHDL
  - ABEL

VHDL和Verilog HDL



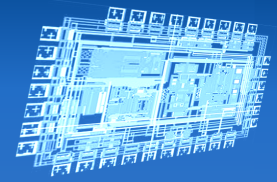
- VHDL最初于1981年由美国军方组织开发。
- Verilog 由Gateway Design Automation公司于1984年作为一个逻辑模拟的语言开发。
- VHDL 和Verilog 共同的特点在于：
  - 能形式化地抽象表示电路的结构和行为；
  - 支持逻辑设计中层次与领域的描述；
  - 可借用高级语言的精巧结构来简化电路行为的描述；
  - 具有电路仿真与验证机制以保证设计的正确性；
  - 支持电路描述由高层到低层的综合转换；
  - 硬件描述与实现工艺无关；
  - 便于文档管理；
  - 易于理解和设计重用。

VHDL和Verilog HDL



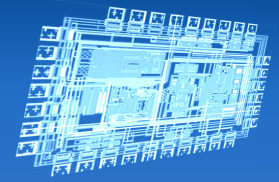
- Verilog HDL的特点
  - 易于学习和掌握，只要有C语言的编程基础，就可以快速掌握并使用
  - 在门级开关电路描述方面比VHDL强
  - 拥有更广泛的应用群体，成熟的资源比VHDL丰富
  - 比较合适作为学习HDL设计方法的入门和基础
- VHDL的特点
  - VHDL在系统级抽象方面比Verilog HDL强
  - VHDL比Verilog HDL的语句冗长且不灵活
  - VHDL较难掌握，需要有Ada编程基础并进行专业培训

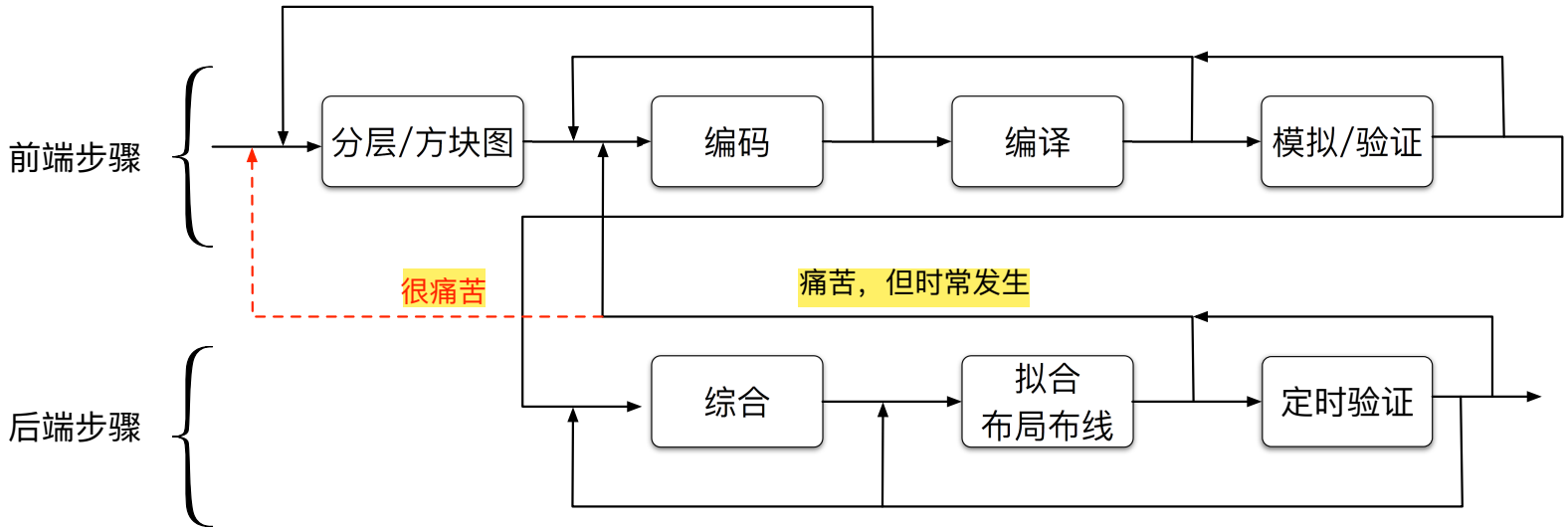
HDL工具组



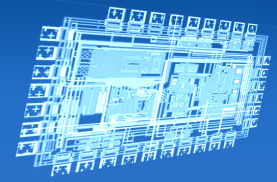
- 基本工具
  - 文本编辑器（Text Editor）
  - 编译器（Compliter）
  - 综合器（Synthesizer）
  - 模拟器（Simulator）：Test Bench及波形编辑器
- 扩展工具
  - 模板生成器（Template generator）
  - 原理图展示器（Schematic Viewer）
  - 翻译器（Translator）
  - 定时分析器（Timing Analyzer）
  - 后插注解器（Back Annotator）

HDL设计过程





Verilog语言入门



- 基本单元– 模块 (module)
  - 端口定义列表：模块通过输入、输出端口与其他模块交互。
    - input：输入到模块内部的信号
    - output：输出到模块外部的信号
    - inout：表示双向信号
  - 内部信号定义：列出模块内需使用的信号
  - 功能语句描述：模块的主体部分，用于描述模块的功能。
- 模块的实例化
  - 模块名 实例标识符(端口关联列表)
  - 端口关联列表：按顺序关联或按端口名字关联

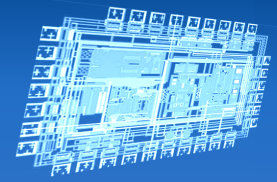
模块定义

```
module 模块名(端口参数1,端口参数2, ···);
      端口参数说明(input, output, inout);
      内部信号定义(wire,reg);

      功能语句描述;

endmodule;
```

示例（禁止门）

```
module VrInhibit (X, Y, Z);
      input X,Y;
      output Z;

      assign Z = X & ~Y;

endmodule
```

Verilog中的标识符

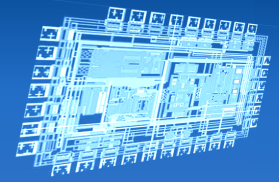


- **标识符**

  以字母或下划线开头，可包含字母、数字、下划线和美元$符号，用来描述“对象”的名称。如模块名、端口名、变量名、常量名、实例名等。

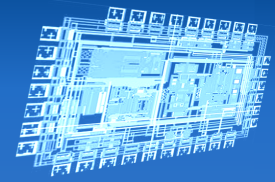
  - 标识符不能与关键字同名。
  - 应采用有意义的名字。
  - 变量名区分大小写。
  - 合法的名字：A_99_Z，Reset，_54MHz_Clock$，Module
  - 非法的名字：123a，$data，module，7seg.v

Verilog中的常量与注释



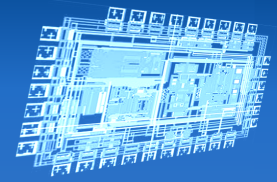
- 整数常量n'Bdd...d, <位宽>'<进制><数字>
  - 位宽(size)为二进制位数，用十进制数表示。
  - 进制(base)为常数的基数，可为二(b)、八(o)、十(d)、十六(h)进制，缺省为十进制。
  - 数字(value)为指定进位制中的任意有效数字，可通过下划线"_"来分隔数字，用于提升可读性。
  - 示例：
    - 10'b1010101010 – 10bit二进制数1010101010
    - 8'd42 – 8bit十进制数42
    - 16'hFFFF – 16bit十六进制数FFFF
- parameter常量:定义模块内部的命名常量
  - `parameter BUS_SIZE=32;`
- 注释
  - 单行注释：//
  - 多行注释：/* ... */

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
  - 可以存储: 0，1，x，z
  - x表示非法值或者不定态
  - z表示浮空值或者高阻态

Verilog中的向量和数组



- 向量

  将多个1位信号组成一个整体进行操作，向量的长度称为位宽，通过有序界[MSB:LSB]来定义。

  - `wire[7:0] a; // 一根位宽为8的连线a`
  - `reg[31:0] rdata, wdata; // 位宽为32的寄存器rdata和wdata`
  - 位选择: `rddata[15:8]` – rddata中间的8bit

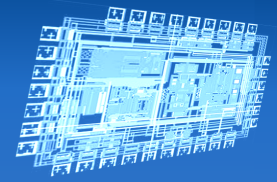
- 数组

  具有相同数据类型的有序集合，通过索引访问各元素

  定义格式：数据类型 数组名[first_addr : last_addr]

  - `wire b[2:0]; //数组b包含3根位宽为1的连线`
  - `reg r[9:0]; //数组r包含10个位宽为1的寄存器`
  - 可用reg[MSB : LSB] [first_addr : last_addr]定义存储器
    - `reg[15:0] mem[1023:0]: //1K×16 b的存储器mem`
  - 数组变量不能整体引用，也不能子界范围引用，只能引用单个元素

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

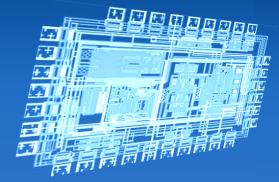
逐位布尔操作

操作符	含义
&	与
|	或
^	异或
~^ , ^~	同或
~	非

算术与移位操作

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

逻辑与关系操作

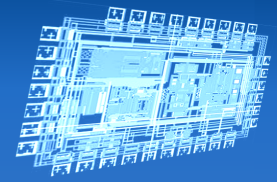


- 条件操作符？
  - 形式：X? Y: Z
  - 例：f= (A>B)? A:B;
    -- 取max(A,B)

逻辑与关系操作

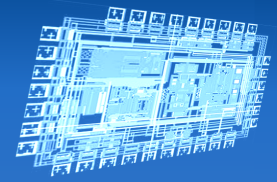
操作符	含义
&&	逻辑与
||	逻辑或
!	逻辑非
==	逻辑相等
!=	逻辑不等
>	大于
>=	大于等于
<	小于
<=	小于等于

归约运算符



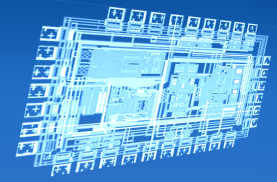
- 归约运算符
  - 包括“&”、“~&”、“|”、“~|”、“^”和“^~”（或“~^”），分别表示与归约、与非归约、或归约、或非归约、异或归约和同或归约运算。
  - 无论操作数的位宽是多少，归约运算结果的位宽均为1。
  - 将操作数最低位依次与前面各位进行与、或、异或等运算。
    - 若信号a的位宽为4，则`&a = ((a[0] & a[1]) & a[2]) & a[3]`
  - 综合出一个输入端口数量与操作数位宽相同的门电路

Verilog的连接操作符



- 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
  - 是唯一一个操作数数量可变的运算符，操作数之间用“,”分开。
  - 位拼接运算结果的位宽为所有操作数的位宽之和。
  - 位拼接运算符的行为相当于信号集线器，无需综合出逻辑门器件。

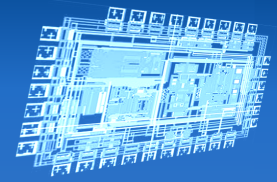
运算符优先级



为提高程序的可读性，建议使用括号来控制运算的优先级！

类 别	运 算 符	优先级
逻辑、位运算符	! ~	高
算术运算符	* / %	↓
	+ -	↓
移位运算符	<< >>	↓
关系运算符	< <= > >=	↓
等式运算符	= = != === !==	↓
归约、位运算符	& ~&	↓
	^ ^~或~^	↓
	| ~|	↓
逻辑运算符	&&	↓
	||	低
条件运算符	? :	

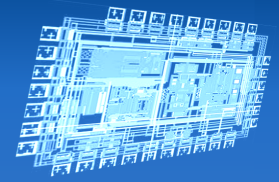
Verlog基本设计模式



- **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

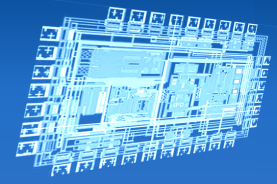


- 实例化组件，描述组件之间的连接
- 内置门: and，nand，xor，or，nor，not，...

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
```

数据流式程序设计

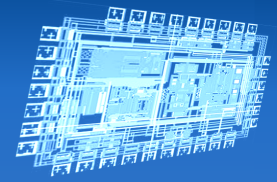


- assign 语句
  - assign语句只能给网格信号(如wire)赋值, 不能给变量（如reg）赋值。但是右边数据类型可以是reg或wire
  - 并行执行，顺序无关，连续执行
  - 可以用条件赋值
  - 相同的wire不能进行重复赋值
  - 描述的电路不能出现组合回路

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

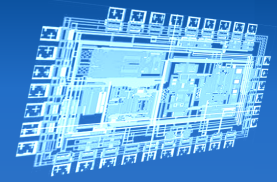


- always程序块
  - always @ (信号1 or 信号2 or . . .)
  - 程序块内顺序执行，可以是begin – end 程序块
- 灵敏度列表 – (信号1 or 信号2 or . . .)
  - 确定always程序块何时执行
  - 可以是上升沿 posedge 或下降沿 negedge
  - 可以缺省设置 always @ *
- always内的赋值
  - 只可赋值给变量(如reg),不可赋值给网格信号(如wire);
  - 阻塞式赋值 =, 立即赋值，对后续语句有影响
  - 非阻塞式赋值 <=, 等整个程序块执行完后赋值

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

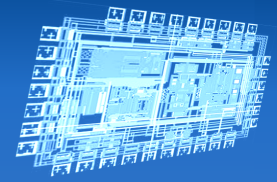
```
module Vrprimei (N, F);
      input [3:0] N;
      output F;
      reg F;
      parameter OneIsPrime=1;
      always @ (N)
        if (N == 1) F = OneIsPrime;
        else if ( (N % 2) == 0)
          begin
            if (N == 2) F = 1; else F = 0 ;
          end
        else if ( N <= 7) F = 1;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

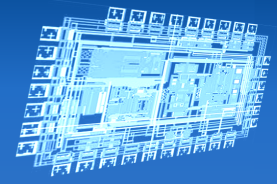
保证变量在每个路径都赋值，避免推断锁存器

过程式程序设计 –for



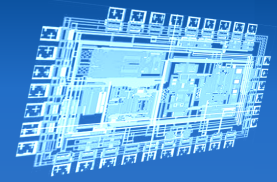
- 可以使用for循环
- 需要注意系统是否可以综合
- Verilog支持 repeat, while, forever；但建议尽量避免使用

语法提示



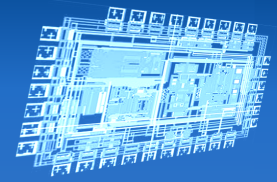
- wire和reg的区别
  - reg不能被assign赋值，只能在单个always块内被=或<=赋值
  - wire只能被assign赋值一次，且不能在always块内被赋值
  - reg和wire都可以被用于各类赋值的右值
- 模块调用接口
  - 子模块的输入只能定义为wire型，但是调用的父模块可以用wire或者reg驱动
  - 子模块的输出可以是reg或wire型，但是调用的父模块一定要用wire型来接受输出
  - 模块端口可以定义为向量，但是不能是数组
- 其他注意事项
  - 允许关联信号的位宽与端口位宽不同，但会报警。
  - 允许端口保持未关联状态，其缺省值为高阻态，但被综合成0，可能会导致非预期结果，故应避免未关联。
  - 复合语句记得前后加上begin, end

语法提示



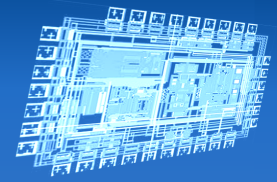
- 注意代码的可综合性
  - 加减乘除运算会综合出对应的加减法器及乘除法器
  - 位运算及逻辑运算综合出对应的门电路
  - 关系运算综合出减法器，等式综合为比较器
  - 移位运算综合为线路拼接或者桶形移位器
  - 条件运算综合出选择器
- 部分不可综合的电路
  - ===判断
  - 复杂的循环
  - 对数组或RAM的任意赋值或读取
  - 多个沿同时驱动一个always块

Test Bench设计



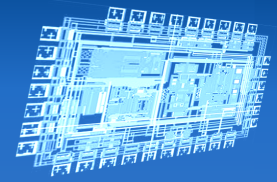
- 测试平台用于产生激励，对待测模块进行测试
- 可以使用函数function或任务task
- 系统任务和函数
  - $display
  - $write
  - &monitor
  - $stop
- 时间尺度 `‘timescale time-unit/ time-precision`
- initial程序块，模拟开始时执行
- 时延: # 延迟数值

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