2.5 EDA技术与HDL语言

2.5.1 EDA技术发展的背景（1）

电路集成度不断提高

- 摩尔定律
- SSI→MSI→LSI→VLSI→ULSI→SOC (System On Chip)
- 集成度用等效门数目（NAND）或晶体管数目来衡量
- 特征尺寸：手机的CPU采用7nm工艺
- PLD和ASIC已经成为当前数字系统设计的主流

2.5.1 EDA技术发展的背景（2）

计算机硬件的发展

– CPU主频加快、内存增大，原来只能在工作站上运行的设计软件可在PC上运行，使设计软件更加普及

计算机软件的发展

– 经历CAD、CAE、EDA三个阶段

– EDA软件功能逐步强大，成为普遍的工具，全方位、大幅度地提高硬件设计效率

数字集成电路水平与计算机软硬件的发展相互促进

2.5.1 EDA技术发展的背景（3）

自下而上的设计方法

- 从最底层（元件）开始，在最高层（系统）结束
- 通用元器件→构成小功能模块→整个硬件系统

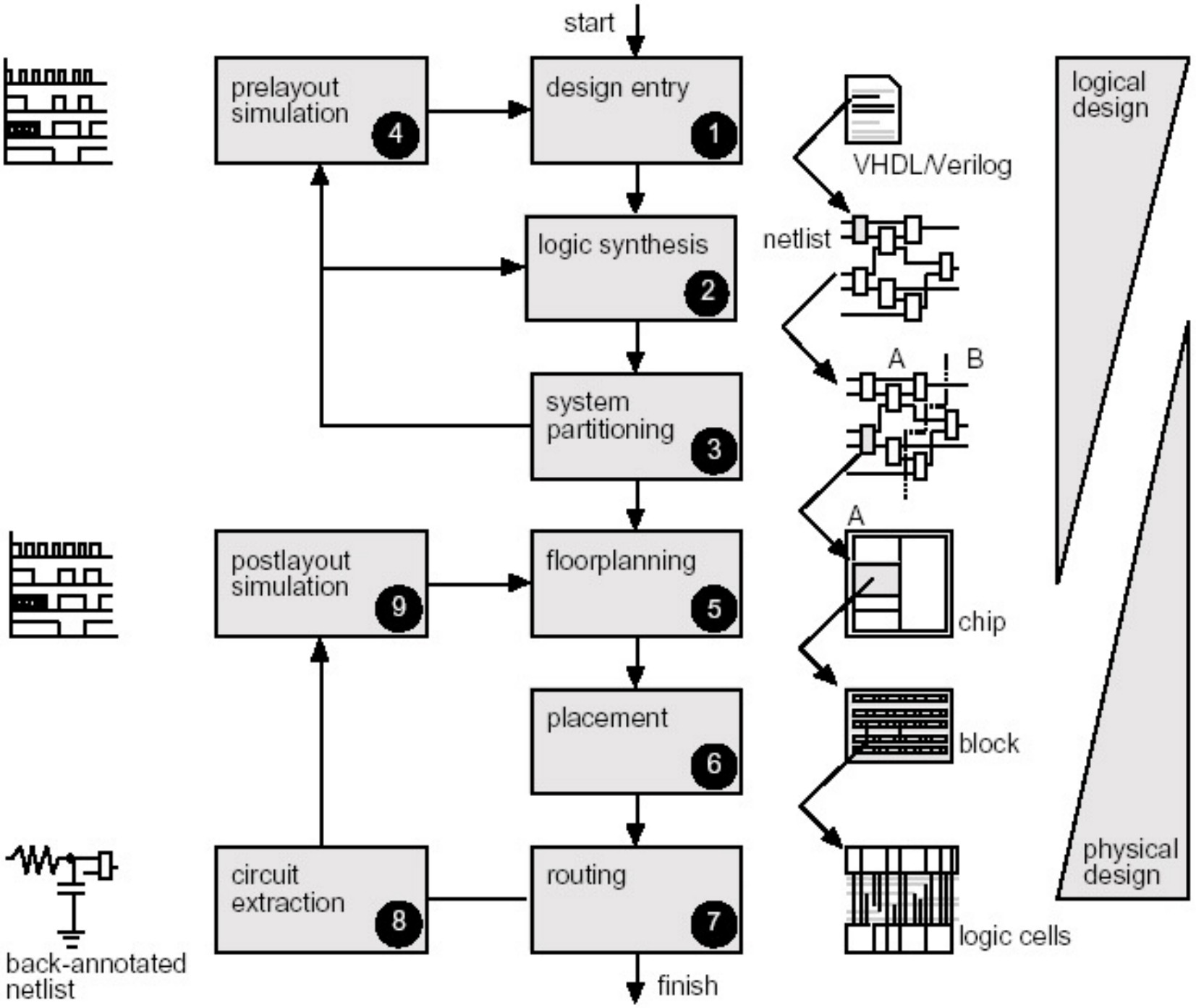
自上而下的设计方法

- 系统总体功能要求→划分若干功能单元→划为基本执行单元→…→EDA元件库中的基本元件
- 多个设计者同时参与
- 功能模块的重用

2.5.1 EDA技术发展的背景（4）

EDA技术

- 采用HDL进行设计
- 高层的综合和优化
- 功能仿真和时序仿真
- 并行工程
- 开放性和标准性
- 包括：系统设计、电路设计、综合、仿真、版图设计、PCB板设计等多方面的功能


start
prelayout simulation
4
design entry
1
logic synthesis
2
system partitioning
3
floorplanning
5
placement
6
routing
7
circuit extraction
8
postlayout simulation
9
back-annotated netlist
finish
VHDL/Verilog
netlist
A
B
A
chip
block
logic cells
logical design
physical design

2.5.2 EDA设计流程（1）

逻辑设计

- 设计输入（Design Entry）：原理图和HDL语言两种方式
- 逻辑综合（Logic Synthesis）：生成网表文件，描述逻辑单元及其之间的连线
- 系统划分（System Partition）：将一个大的系统分成多个模块，每个模块由不同的ASIC芯片完成
- 布局前仿真（Prelayout Simulation）：也叫**功能仿真**，验证设计的功能是否正确

2.5.2 EDA设计流程（2）

物理设计

- 设计输入平面规划：在芯片上安排放置网表中的模块
- 布局（Placement）：确定每个模块中每个单元的位置
- 布线（Routing）：连接模块和单元
- 参数提取（Extraction）：确定连线的电阻、电容参数
- 布局后仿真（Postlayout simulation）：也叫**时序仿真**，在加入布局/布线所增加的各种电学参数后，再次检查系统仍能否正常工作

2.5.2 EDA设计流程——输入方法

原理图输入

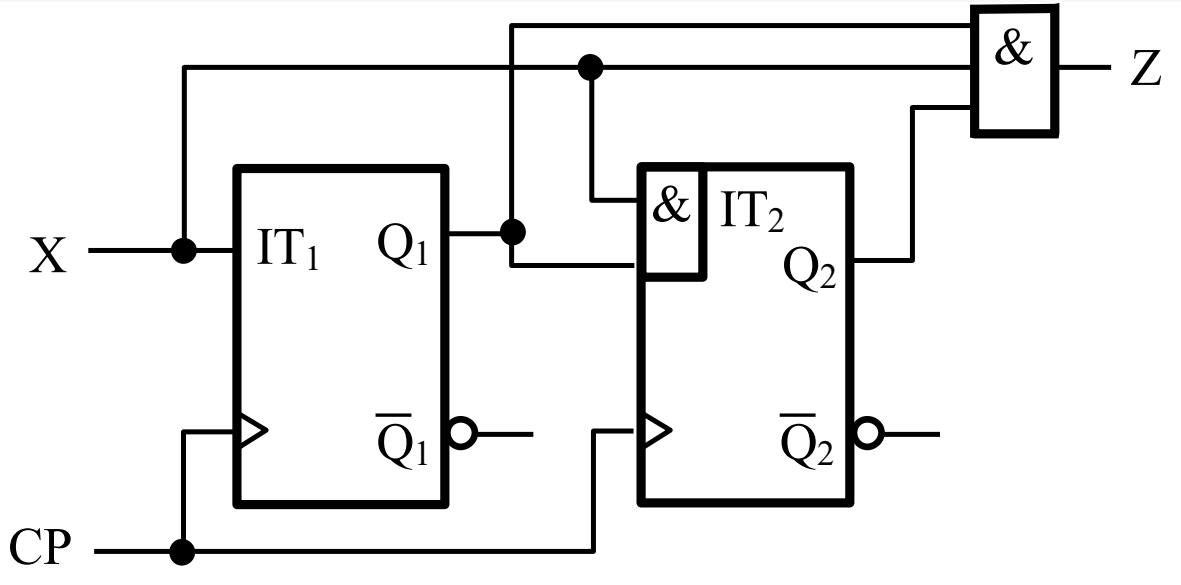
- 使用元件符号和连线等描述
- 综合的效率高
- 可读性不强、移植性不强、设计大规模的数字系统时显得繁琐
- 一般用于顶层设计时各模块之间的连接通用

HDL输入

- 逻辑描述功能强，可读性强、国际标准，便于移植
- 综合效率相对较低

HDL与原理图的关系类似于高级语言与汇编语言

2.5.2 EDA设计流程——输入方法




```
if(clk'event and clk='1') then
    if(x=1) then count4 := count4 + 1;
    else count4 := count4 - 1;  end if;
    if(count4 = 3) then       z := '1';
    else        z := '0';    end if;
end if;
```

2.5.2 EDA设计流程——输入方法

HDL的特点

- 良好的可读性，容易理解。
- 抽象表示电路的行为和结构；
- 与硬件独立，一个设计可用于不同的硬件结构，不必了解过多的硬件细节；
- 支持设计的重复使用；
- 有丰富的软件支持HDL的综合和仿真，提高设计效率；
- 更方便地向ASIC过渡。

2.5.2 EDA设计流程——HDL与计算机语言的区别

运行基础

- 计算机语言是在CPU + RAM构建的**平台上运行的指令**
- HDL设计的是由逻辑门、触发器组成的**数字电路**

执行方式

- 计算机语言基本上以**串行**的方式执行
- HDL在总体上是以**并行**方式工作

验证方式

- 计算机语言重点关注于**变量、寄存器值的变化**
- HDL要实现严格的**时序逻辑关系**

2.5.3 Verilog的基本语法（1）—语言要素

间隔符

- 分隔文本，可以使文本错落有致，便于阅读与修改
- 包括空格符（\b）、TAB 键（\t）、换行符（\n）及换页符

注释符

- 改善程序的可读性，编译时不起作用
- 多行注释符： /* --- */
- 单行注释符：以//开始到行尾结束为注释文字

标识符

- 给对象（如模块名、电路的输入与输出端口、变量等）取名所用的字符串。**命名规范与C语言要求相同。**
- clk、counter8、_net、bus_A

2.5.3 Verilog的基本语法（1）—语言要素

关键词

- Verilog语言规定的特殊字符串，用来定义语言的结构，约100个
- 如：module、endmodule、input、output、wire、reg、and等。关键词都是**小写**，关键词不能作为标识符使用符。

逻辑值

- 表示数字逻辑电路的逻辑状态作用
- 4种基本逻辑状态：0、1、**X(x)、Z(z)**

2.5.3 Verilog的基本语法（1）—语言要素

常量

- 整数型、实数型、字符串型
- 整数型，可用二进制、八进制、十进制和十六进制表示，表示形式：<位宽><进制><数字>符。
  - 如： 8’b01010011，8’h53，83(默认32位的十进制数)
- x与z：每个字符所代表的位宽取决于所用进制
- 下划线：用来分隔数的表达以提高程序可读性
  - 16’b0011101001010001
  - 16’b0011_1010_0101_0001
- 用**paramter**关键词定义常量

2.5.3 Verilog的基本语法（1）—语言要素

变量

- **wire型**
  - 用**assign赋值**的组合逻辑信号，输入输出信号**默认为wire型**
- **reg型**
  - 在**always块内被赋值**的任何信号都必须定义成reg型
  - **注意：**reg型并不一定就是触发器的输出
- 总线变量的定义方式
  - **wire/reg** [n-1:0] 变量名1，变量名2，…，变量名m
  - **wire/reg**[7:0] Data，wire[3:0] Addr;

2.5.3 Verilog的基本语法（1）—语言要素

运算符

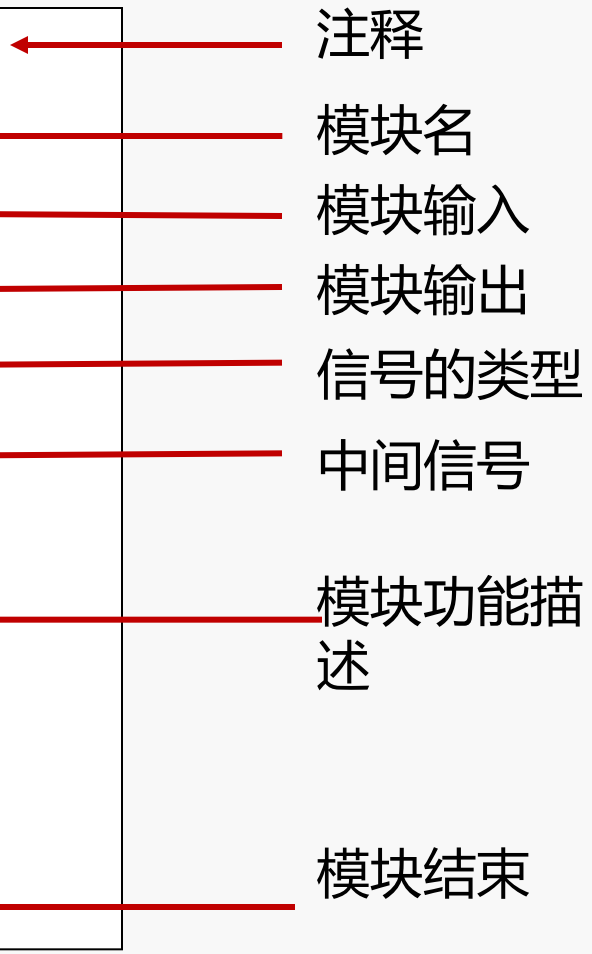
- 算术运算符：+、 -、*、/、%
- 赋值运算符：<=、=
- 关系运算符：>、<、>=、<=
- 等式运算符：==、!=、===、!==
- 逻辑运算符：&&、||、！
- 位运算符： &、|、~、^（异或）、^~（同或）
- **缩减运算符：&、~&、|、~|、^、~^**
- 条件运算符：?
- 移位运算符：<<、>>，用0来填补空出的位。
- **拼接运算符：{}，**将多个信号拼接起来进行操作
- **优先级别**

2.5.3 Verilog的基本语法（2）—基本结构

```
//test module is a ……
module TestMod(a, b, c, d, F);
input a, b, c, d;
output F;
wire a, b, c, d, F;
wire tmp1, tmp2;

assign tmp1 = a&b;
assign tmp2 = ~(c&d);
assign  F = ~(tmp1 | tmp2);

endmodule
```



2.5.3 Verilog的基本语法（2）—基本结构

- 模块由两部分组成
  - 模块的接口
  - 模块的功能描述
- 模块的端口
  - **module** 模块名（口1，口2，口3，…）
- 端口的I/O形式
  - 输入口：**input**
  - 输出口：**output**
  - 双向口（输入/输出口）：**inout**
  - **input**[信号宽度-1：0] 端口名1，端口名2；

2.5.3 Verilog的基本语法（2）—基本结构

有3种方式描述逻辑功能：

- **数据流**方式描述——用**assign**语句

```
assign a=b&c;
```

- **行为**方式描述——用**always**语句

```
always @ ( b or c)
begin
    a = b&c;
end
```

- **结构化**描述——用**元件例化**

```
and ul(c,a,b);
```

2.5.3 Verilog的基本语法（3）—逻辑功能描述

1、数据流方式描述

```
//test module is a ……
module TestMod(a, b, c, d, F);
input a, b, c, d;
output F;
wire a, b, c, d, F;
wire tmp1, tmp2;

assign tmp1 = a&b;
assign tmp2 = ~(c&d);
assign  F = ~(tmp1 | tmp2);

endmodule
```

注释

模块名

模块输入

模块输出

信号的类型

中间信号

模块功能描述

模块结束

2.5.3 Verilog的基本语法（3）—逻辑功能描述

1、数据流方式描述（简化）

```
// TestMod is a ……
 module TestMod(a, b, c, d, F);
 input a, b, c, d;
 output F;
 //wire a, b, c, d, F;   默认为wire型，可以不需要
 //wire tmp1, tmp2;   省却中间变量，功能描述可简写为一行

 //assign tmp1 = a&b;
 //assign tmp2 = ~(c&d);
 //assign  F = ~(tmp1 | tmp2);
 assign F = ~((a&b) | (~(c&d)));
 endmodule
```

2.5.3 Verilog的基本语法（3）—逻辑功能描述

2、行为方式描述

```
// TestMod is a ……
 module TestMod(a, b, c, d, F);
 input a, b, c, d;
 output F;
 reg tmp1, tmp2;
 reg F;

 always@(a or b or c or d)
 begin
     tmp1 = a&b;
     tmp2 = ~(c&d);
     F = ~(tmp1 | tmp2);
 end
 endmodule
```

简化

```
// TestMod is a ……
 module TestMod(a, b, c, d, F);
 input a, b, c, d;
 output F;
 reg F;

 always@(a or b or c or d)
    F = ~((a&b) | (~(c&d)));

 endmodule
```

2.5.3 Verilog的基本语法（3）—逻辑功能描述

3、元件例化方式描述

```
// TestMod is a ……
 module TestMod(a, b, c, d, F);
 input a, b, c, d;
 output F;
 wire tmp1, tmp2;

 and ul(tmp1, a, b);          //assign tmp1 = a&b;
 nand u2(tmp2, c, d);         //assign tmp2 = ~(c&d);
 nor u3(F, tmp1, tmp2);   //assign  F = ~(tmp1 | tmp2);

 endmodule
```