

数字设计实践与 Verilog硬件描述语言

高翠芸

School of Computer Science

gaocuiyun@hit.edu.cn

主要内容

- 数字系统设计
- Verilog HDL基本知识
- Verilog HDL语法
- 方法：
 - 讲义 → 自学 → 实践

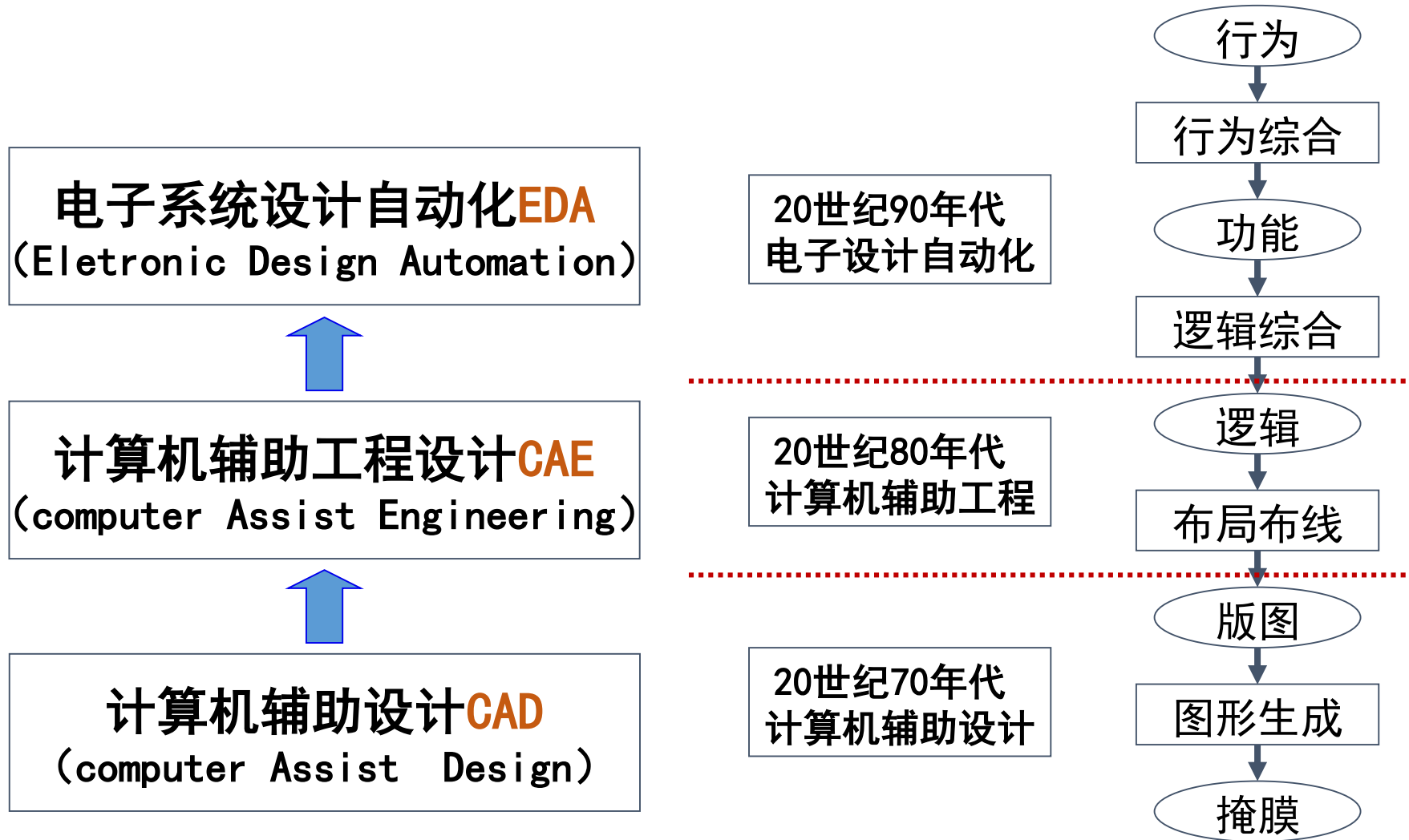
数字系统概念

- **数字系统**：是指对数字信息进行存储、传输、处理的电子系统。它的输入和输出都是数字量。
- 通常把门电路、触发器等称为**逻辑器件**；
- 将由逻辑器件构成，能执行某单一功能的电路，如计数器、译码器、加法器等，称为**逻辑功能部件**；
- 把由逻辑功能部件组成的能实现复杂功能的数字电路称**数字系统**。

复杂的数字系统

- 嵌入式微处理机系统
- 数字信号处理系统
- 高速并行计算逻辑
- 高速通信协议电路
- 高速编码/解码、加密/解密电路
- 复杂的多功能智能接口
- 门逻辑总数超过几万门达到几百甚至达几千万门的数字系统

数字设计的发展

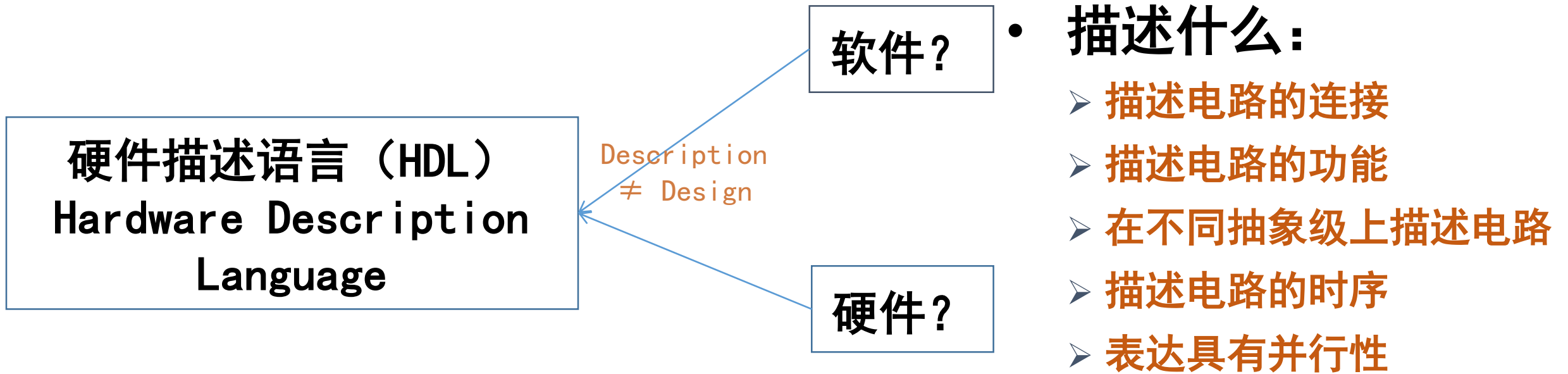


EDA技术

- EDA (Electronic Design Automation) 电子设计自动化
- 以计算机为工具，在EDA软件平台上，用硬件描述语言完成设计文件，然后由计算机自动地完成逻辑编译、化简、分割、综合、优化、布局、布线和仿真，直至对于特定目标芯片的适配编译、逻辑映射和编程下载等工作。
- 设计者的工作——从概念、算法、协议等设计电子系统
- 计算机的工作——电路设计、性能分析到设计出IC版图或PCB版图

什么是硬件描述语言

- 具有特殊结构能够对**硬件逻辑电路的功能**进行描述的一种高级编程语言

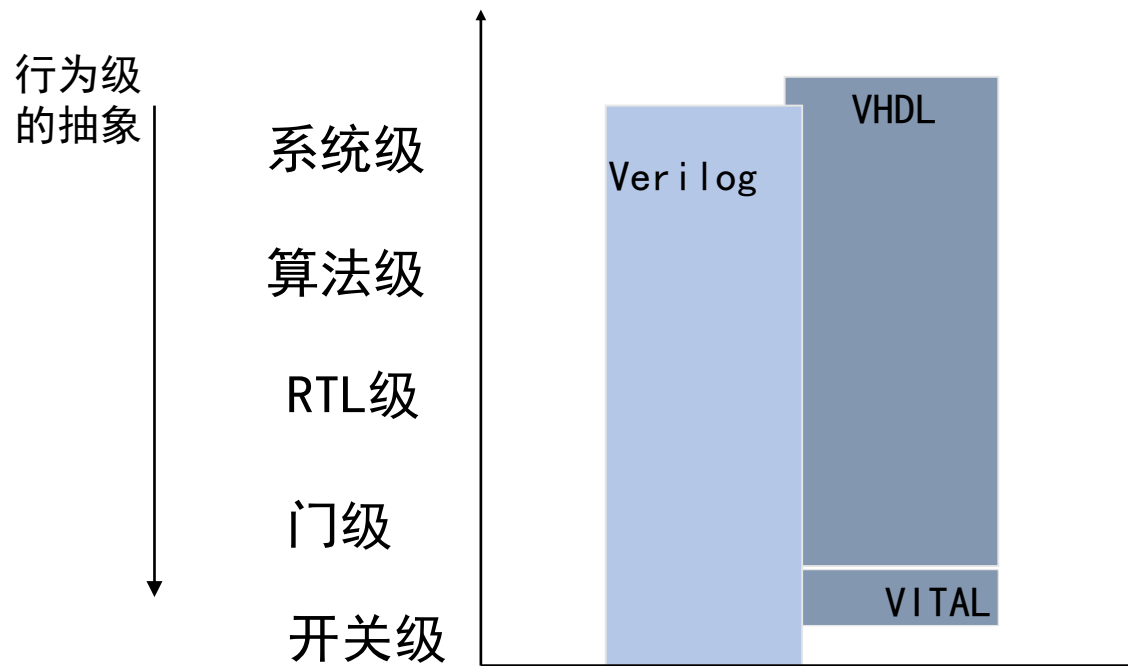


为什么要使用硬件描述语言

- 电路的逻辑功能容易理解；
- 便于计算机对逻辑进行分析处理；
- 把逻辑设计与具体电路的实现分成两个独立的阶段来操作；
- 逻辑设计与实现的工艺无关；
- 逻辑设计的资源积累可以重复使用；
- 可以由多人共同更好更快地设计非常复杂的逻辑电路（几十万门以上的逻辑系统。

VHDL vs. Verilog

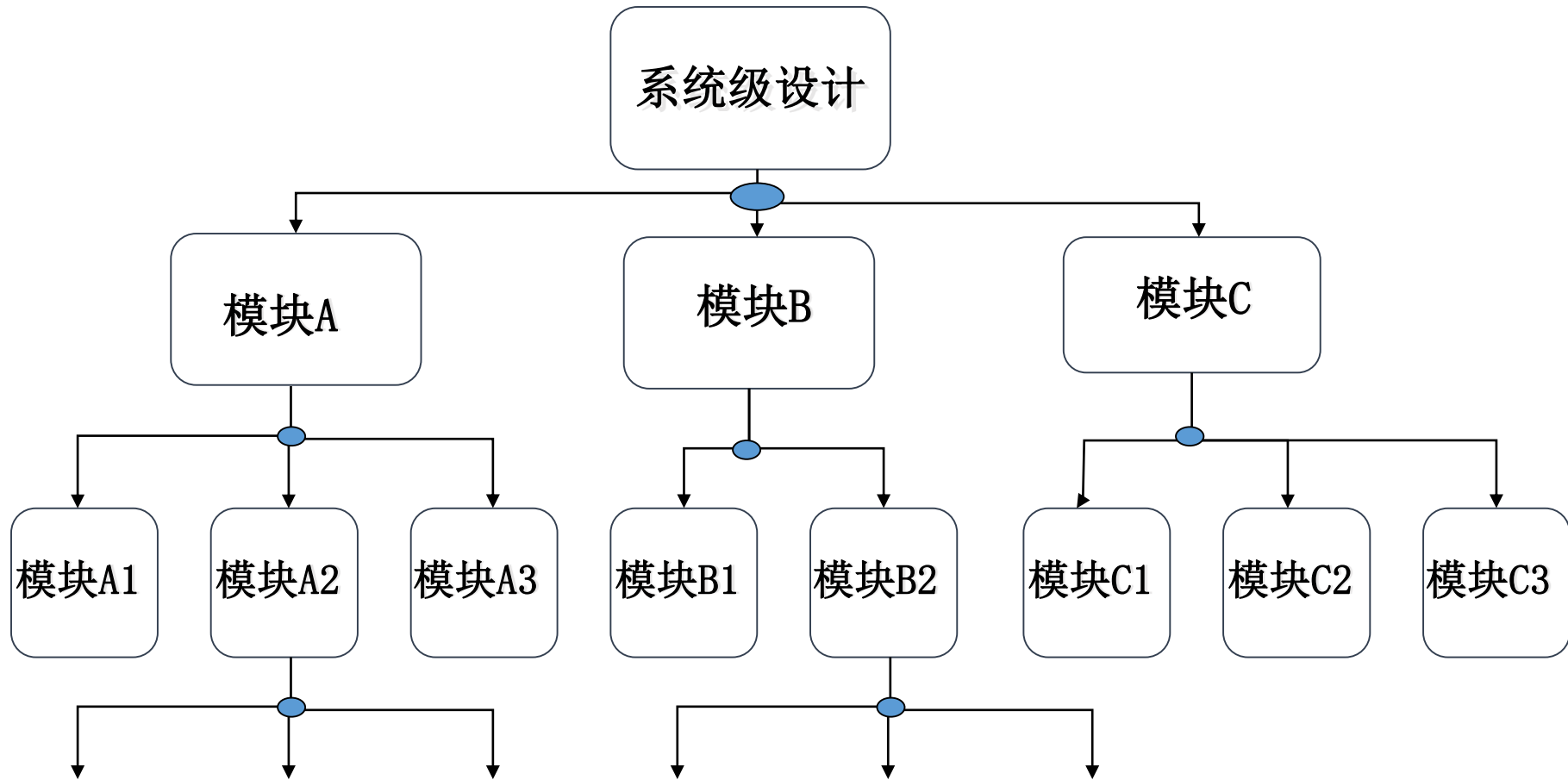
- VHDL:
 - 起源于ADA语言
 - 侧重于系统级描述
 - 含有大量的内置数据类型和用户自定义类型
- Verilog:
 - 起源于C语言
 - 侧重于电路级描述
 - 数据类型由语言本身定义，含有专门描述连线等的类型



Verilog与VHDL建模能力的比较

Verilog 应用较广泛、起步更容易！

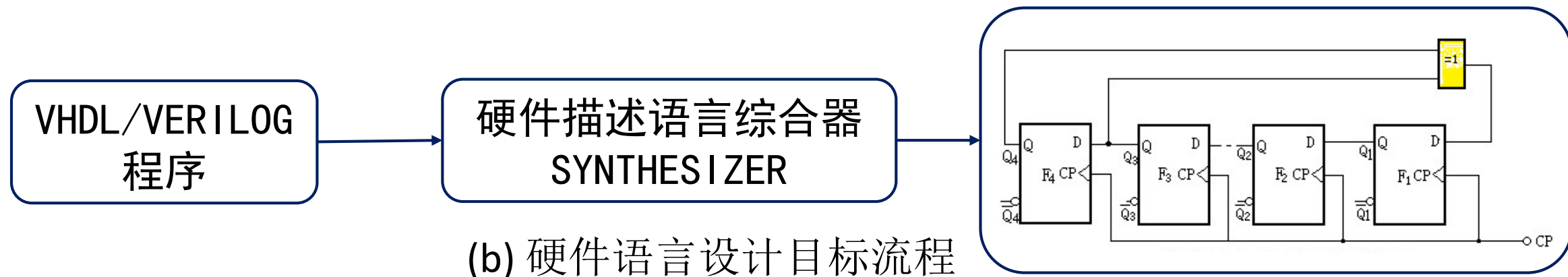
Top-Down设计思想



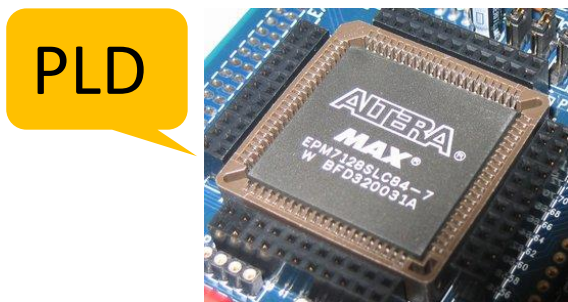
软件描述语言和硬件描述语言的区别



(a) 软件语言设计目标流程

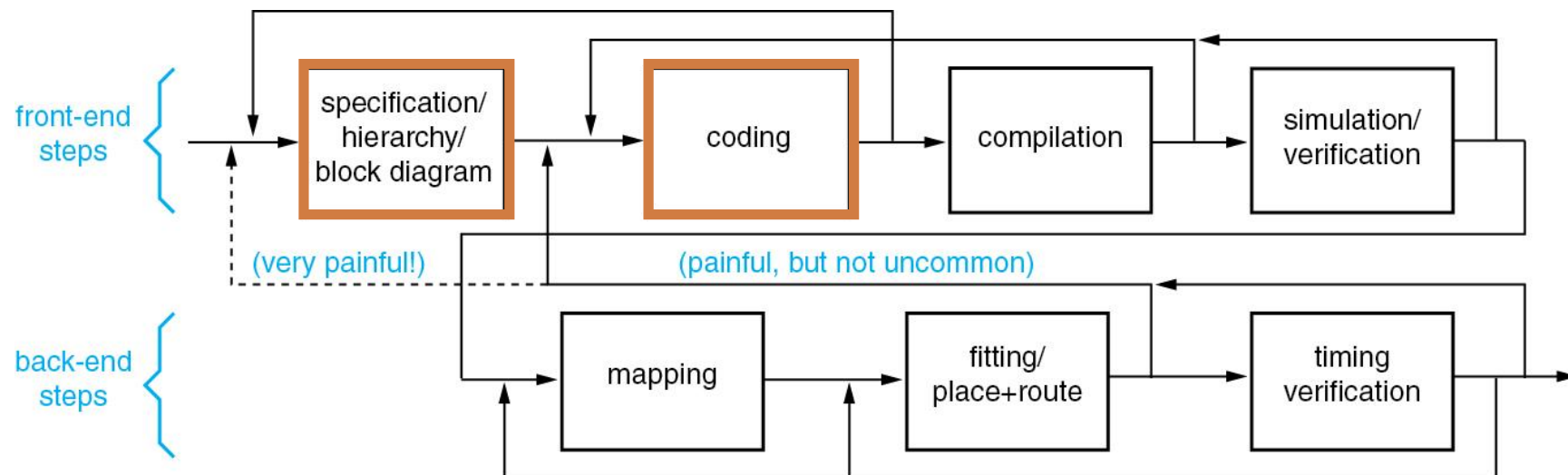


(b) 硬件语言设计目标流程



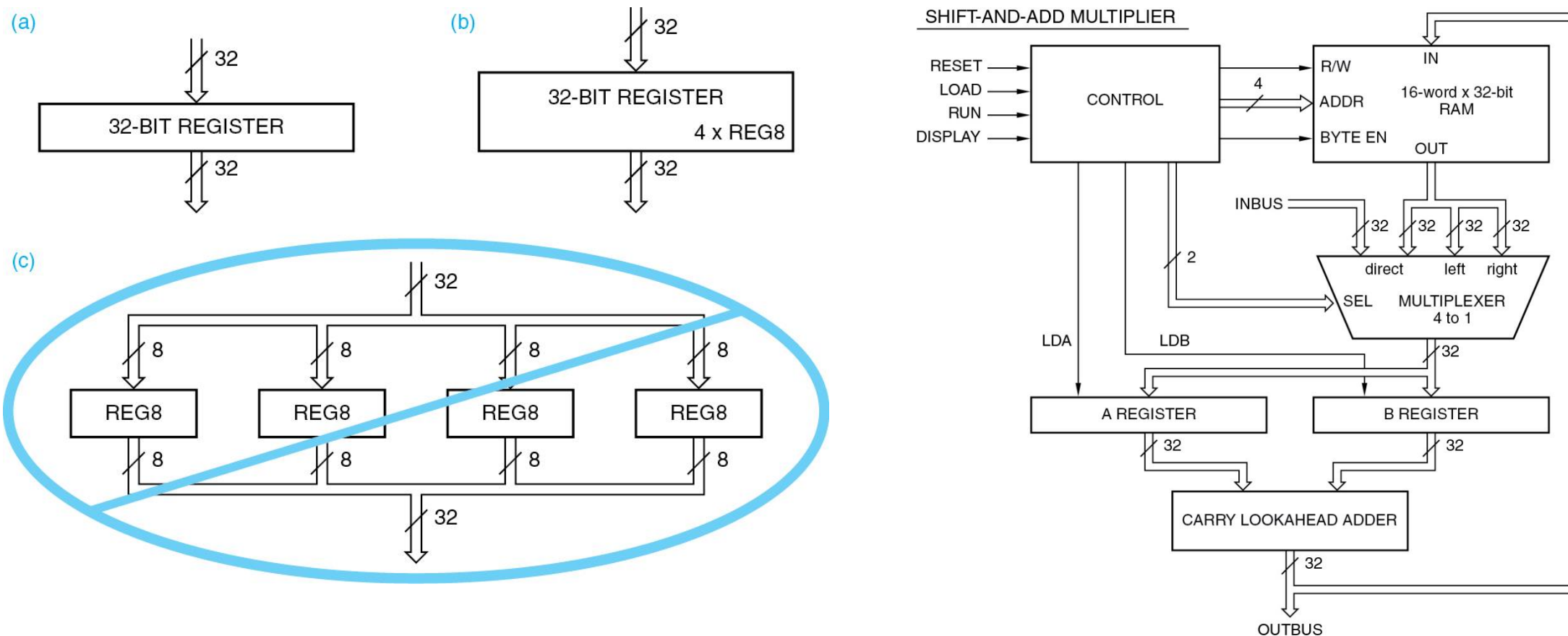
基于HDL的设计流程

- 功能说明：准确地描述电路或系统应该做什么，包括所有输入和输出（“接口”）的描述及实现的功能。
- 方框图：展示系统的输入、输出、功能模块、内部数据通路和重要的控制信号。
- 设计代码：用硬件描述语言实现电路或系统设计



方框图

明确展示最重要的系统元件，以及这些元件怎样共同工作



Verilog HDL语言介绍

什么是Verilog HDL

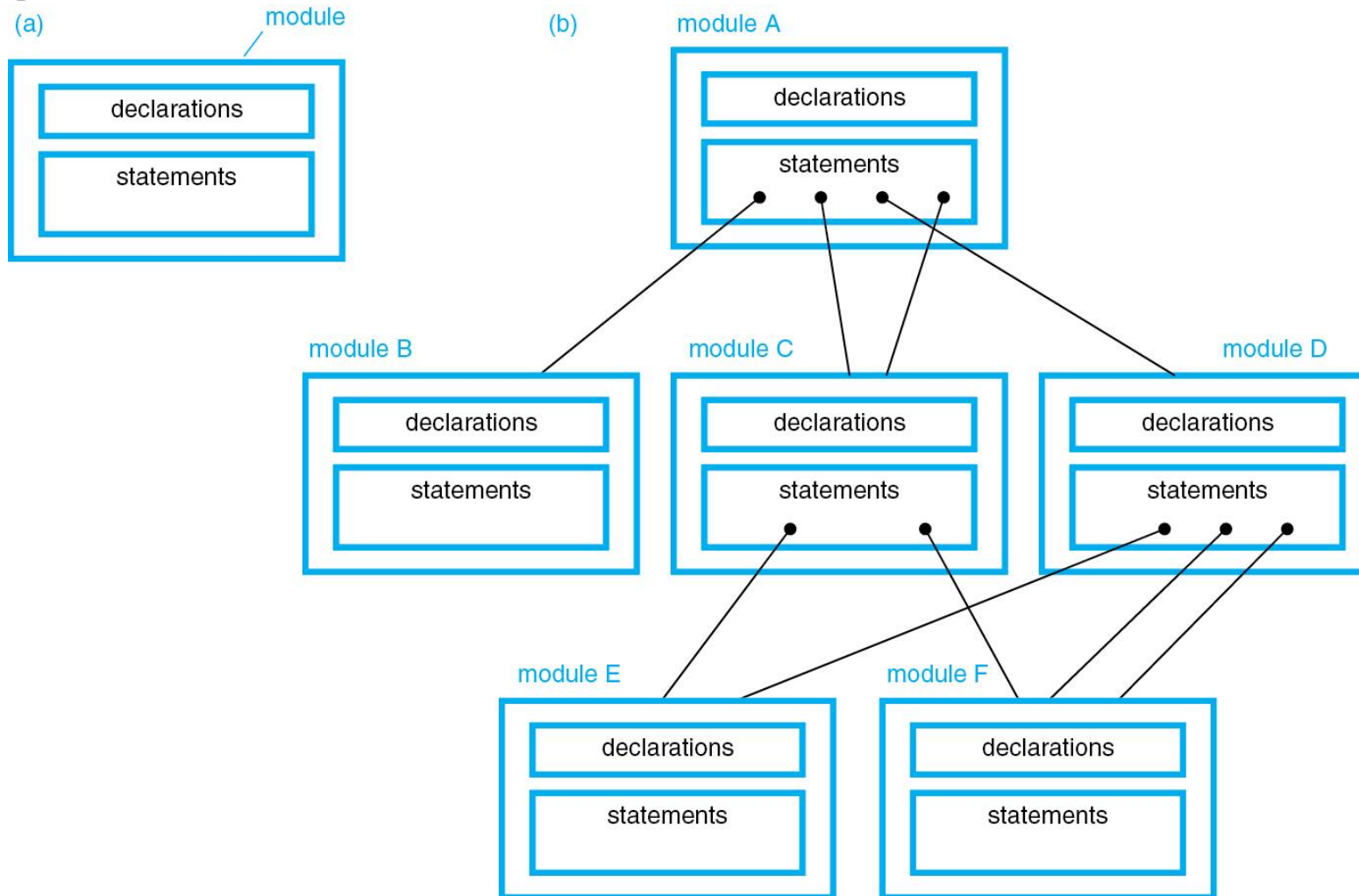
- Verilog HDL是硬件描述语言的一种，用于数字电子系统设计。设计者可以用它进行各种级别的逻辑设计，可用它进行数字逻辑系统的仿真验证、时序分析、逻辑综合。它是目前应用最广泛的一种硬件描述语言。
- 与VHDL相比：
 - 有较多的第三方工具的支持
 - 语法结构比VHDL简单
 - 学习起来比VHDL容易
 - 仿真工具比较好使
 - 测试激励模块容易编写

应用情况和适用的设计

- Verilog较为适合系统级(System)、算法级(Algorithm)、寄存器传输级(RTL)、逻辑(Logic)、门级(Gate)和电路开关级(Switch)的设计。

Verilog模型和模块

■ Verilog设计的基本单元是模块（module）



Verilog HDL的特点

- 形式化地表示电路的行为和结构；
- 从C编程语言中继承了多种操作符和结构；
- 可在多个层次上对所设计的系统加以描述，语言对设计规模不加任何限制；
- 具有混合建模能力：一个设计中的各子模块可用不同级别的抽象模型来描述；
- 基本逻辑门、开关级结构模型均内置于语言中，可直接调用。
- 易学易用，功能强大

Verilog HDL描述方式

- 行为型描述



只描述行为特征，不涉及电路的实现，一种高级语言描述方式。

- 数据流型描述



给出逻辑方程，通过assign连续赋值实现组合逻辑功能。

- 结构型描述



调用Verilog语言已定义的基元（实体）描述逻辑电路的方式。

Verilog HDL的抽象级别

• 行为描述语言&结构描述语言

行为级描述

系统级：

用高级语言结构实现设计模块的外部性能模型；

算法级：

用高级语言结构实现设计算法模型；

RTL级（寄存器传输级）：

描述数据在寄存器之间流动和如何处理这些数据的模型；

侧重对模块
行为功能的
抽象描述

结构级描述

门级：

描述逻辑门以及逻辑门之间的连接模型；

开关级：

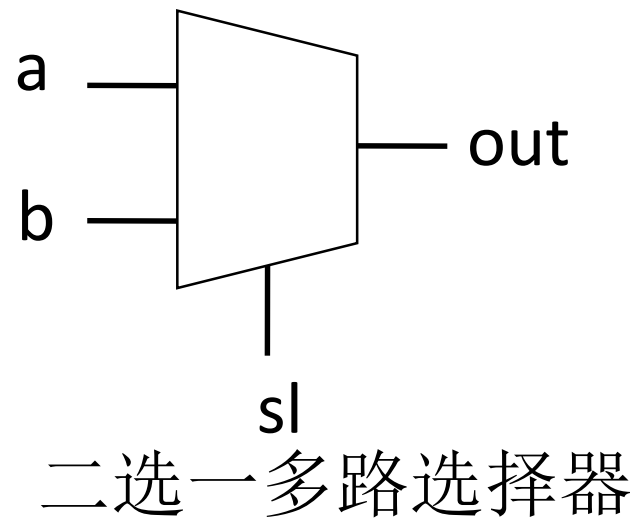
描述器件中三极管和存储节点以及它们之间连接的模型。

侧重对模块
内部结构实现
的具体描述

行为级Verilog HDL

- 在行为级模型中，逻辑功能描述采用高级语言结构，如@、while、wait、if、case。
- RTL模块是可综合的，它是行为模块的一个子集合。
- 行为级Verilog

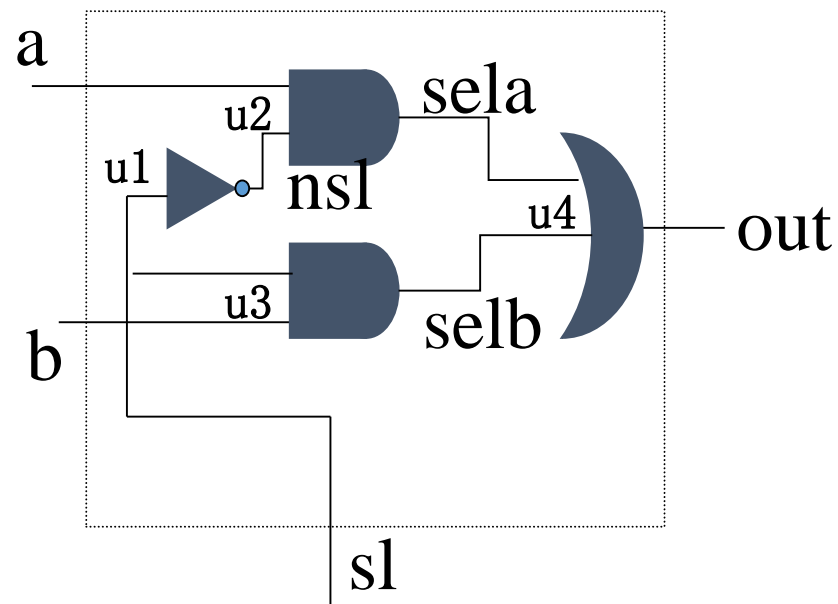
```
module muxtwo (out, a, b, sl); //二选一多路选择器
    input a, b, sl;    //输入信号名
    output out;        //输出信号名
    reg out;
    always @( sl or a or b)
        if (! sl) out = a; //控制信号sl为非，输出与输入信号a一致
        else out = b;      //控制信号sl为非，输出与输入信号b一致
endmodule
```



结构级Verilog HDL

- Verilog内部带有描述基本逻辑功能的基本单元(primitive)，如and门。
- 综合产生的结果网表通常是结构级的。
- 结构级Verilog适合开发小规模元件

```
module muxtwo (out, a, b, sl); //二选一多路选择器
    input a, b, sl;
    output out;
    not u1 (nsl, sl );      //nsl=~sl
    and #1 u2 (sela, a, nsl); //sela=a&nsl
    and #1 u3 (selb, b, sl); //selb=b&sl
    or  #2 u4 (out, sela, selb); //out=sela|selb
endmodule
```



二选一多路选择器

学习Verilog HDL的要点

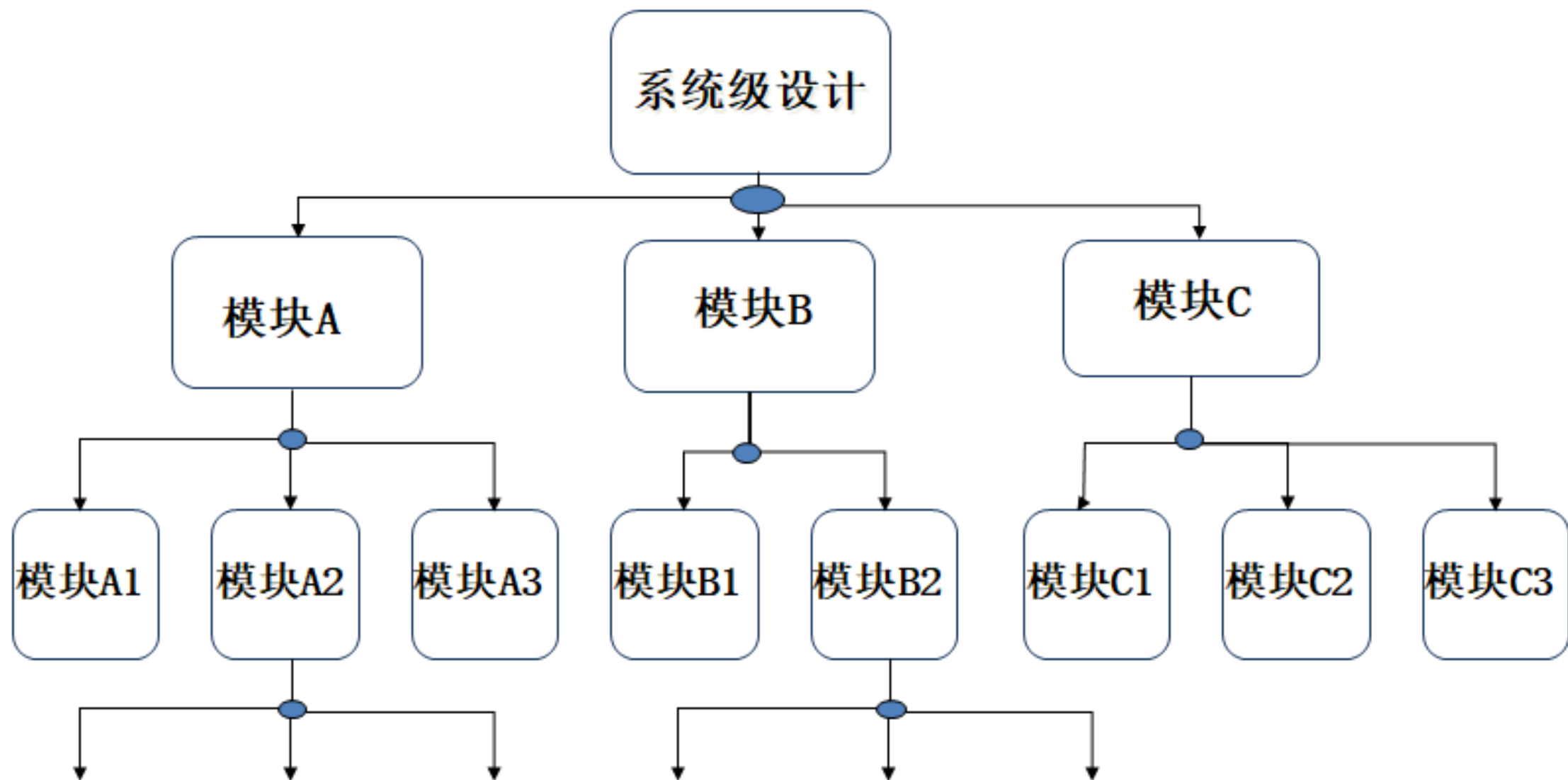
- 一定要对电路图和电路的时序图有深刻的认识
 - 编写Verilog代码的目的是生成实际硬件电路。
 - 而电路一般都不是串行执行的，很多时候都是并行工作的。
- 衡量Verilog代码的唯一标准：就是在代码正确与清晰的前提下，可以生成结构尽可能简单、功能却非常强大的电路！
- 不是所有的Verilog代码都能转换成实际电路(“可综合”)。
- 另外，即使用可综合的代码去编写，如果你描述的电路实际上无法实现，也是无法综合的！

Verilog HDL语法

Verilog HDL语法

- 模块的结构
- 标识符和数据类型
- 运算符及表达式
- 赋值语句和块语句
- 条件语句和循环语句
- 模块的调用
- 模块的测试

Verilog 的模块



模块基本结构

模块声明:

module module_name (port_list); //模块名 (端口声明列表)

端口定义:

input[信号位宽]; //输入声明

output [信号位宽]; //输出声明

...

数据类型说明:

reg [信号位宽]; //寄存器类型声明

wire [信号位宽]; //线网类型声明

parameter; //参数声明

...

功能描述: //主程序代码

assign $a=b+c$

always@(posedge clk or negedge reset)

function

task

...

endmodule

模块声明

module module_name (port_list); //模块名（端口声明列表）

- “模块名”是模块唯一的标识符，区分大小写。
- “端口列表”是由模块各个输入、输出和双向端口组成的列表。
(input,output,inout)
- 端口用来与其它模块进行连接，括号中的列表以“,”来区分，列表的顺序没有规定，先后自由。

```
module muxtwo (out, a, b, sl);
```

```
module muxtwo (out, a, b, sl); //二选一多路选择器
    input a, b, sl;
    output out;
    reg out;
    always @( sl or a or b)
        if (! sl) out = a;
        else out = b;
endmodule
```

端口定义

input[信号位宽]; //输入声明

output [信号位宽]; //输出声明

inout[信号位宽]; //输入/输出端口

input a, b, sl; //输入信号名
output out; //输出信号名

```
module muxtwo (out, a, b, sl); //二选一多路选择器
    input a, b, sl;   //输入信号名
    output out;   //输出信号名
    reg out;
    always @( sl or a or b)
        if (! sl) out = a;
           //控制信号sl为非, 输出与输入信号a一致
        else out = b;
           //控制信号sl为非, 输出与输入信号b一致
endmodule
```

- 输入端口：模块从外界读取数据的接口，是连线类型
- 输出端口：模块向外界传输数据的接口，是连线或寄存器型
- 输入输出端口：可读取数据也可接收数据的端口，数据是双向的，是连线型
- 端口定义也可以写在端口声明的位置：

```
module module_name(input port1,input port2,...output port1,...);
```

数据类型说明

```
reg [信号位宽]; //寄存器类型声明
wire [信号位宽]; //线网类型声明
parameter;      //参数声明
```

```
reg out; //输出信号reg类型
```

```
module muxtwo (out, a, b, sl); //二选一多路选择器
    input a, b, sl; //输入信号名
    output out; //输出信号名
    reg out;
    always @( sl or a or b)
        if (! sl) out = a;
            //控制信号sl为非, 输出与输入信号a一致
        else out = b;
            //控制信号sl为非, 输出与输入信号b一致
endmodule
```

- 模块中用到的所有信号都必须进行数据类型的定义。
- 声明变量的数据类型后, 不能再进行更改
- 在VerilogHDL中只要在使用前声明即可
- 声明后的变量、参数不能再次重新声明
- 声明后的数据使用时的配对数据必须和声明的数据类型一致

Verilog HDL语法

- 模块的结构
- 标识符和数据类型
- 运算符及表达式
- 赋值语句和块语句
- 条件语句和循环语句
- 模块的调用
- 模块的测试

标识符

- 赋给对象的唯一名称，可以是字母、数字、下划线和符号“\$”的组合，且首字符只能是字母或者下划线。
- 大小写敏感。
- 关键词用小写字母定义，如：always,and,assign等
- 注释有两种：
 - 以“/*”开头，以“*/”结束。
 - 以“//”开头到本行结束。

数据类型

- 共有19种数据类型，分为物理数据类型和抽象数据类型
- 物理数据类型：抽象数据程度比较低，与实际硬件电路的映射关系明显。
 - 连线型wire、寄存器型reg、存储器型memory 等
- 抽象数据类型：用于进行辅助设计和验证的数据类型
 - 整型integer、时间型time、实型real、参数型parameter等
- 数据类型分为常量和变量，分别属于以上类型。
 - 常量：数字、参数型parameter
 - 变量：连线型wire、寄存器型reg、存储器型memory 等

数据类型

- 分成常量和变量
- 最基本最常用的4种：
 - 寄存器型 `reg`
 - 线网型 `wire`
 - 整形 `integer`
 - 参数型 `parameter`
- 其余的包括: `large\ medium\ scalared\ time\ small\ tri\ trio\ tri1\ triand\ trior\ trireg\ vectored\ wand\ wor` 型，主要与基本单元库有关，设计时很少使用

常量

- 在程序运行中，其值不能被改变的量叫常量
- 两类最基本的常量：数字型常量和参数（parameter）
 - 数字型常量：整型数可以按如下两种方式书写
 - 简单的十进制数格式
 - 基数格式

数字

- 整数:

- 二进制(b或B)、十进制(d或D)、十六进制(h或H)、八进制(o或O)
- 表达方式:

- [位宽][']进制 值, 这是一种全面的描述方式
- 位宽是按照二进制数来计算的。进制可以为b或B(二进制)、o或O(八进制)、d或D(十进制)、h或H(十六进制)。值是基于进制的数字序列。如:

3'b001, 3'B110 三位二进制

6'o12, 6'O12 六位八进制

4'd9, 4'D9 十位十进制

8'hBF, 8'HBF 八位十六进制

8'b10101100 //位宽为8的数的二进制表示, 'b表示二进制

8'ha2 //位宽为8的数的十六进制表示, 'h表示十六进制

数字

- 基数格式计数形式的数通常为无符号数。
- 这种形式的整型数的长度定义是可选的。如果没有定义一个整型型的长度，数的长度为相应值中定义的位数。
- 下面是两个例子：
 - 'o721 9位八进制数
 - 'hAF 8位十六进制数
- 没有数字位宽采用默认位宽（这由具体的机器系统决定）
- 在< 数字>这种描述方式中，采用默认进制（十进制）

负数：

位宽表达式前加一个减号，减号必须写在数字定义表达式最前面。

-8'd5 //-5 8'd-5 //非法格式

数字

- 如果定义的长度比为常量指定的长度长，通常在左边填0补位。但是如果数最左边一位为x或z，就相应地用x或z在左边补位。
 - 例如：10'b10 左边添0占位, 0000000010
 - 10'bx0x1 左边添x占位, xxxxxx0x1
- 如果长度定义得更小，那么最左边的位相应地被截断。
 - 例如：
 - 3'b1001_0011与3'b011 相等
 - 5'H0FFF 与5'H1F 相等
- 下划线可以用来分割数的表达式以提高程序的可读性，但不能用在位宽和进制处，只能用于具体的数字之间，比如：
 - 16'b1010_1011_11111_000 //合法格式
 - 8'b_0011_1011 //非法格式

参数型 (parameter)

- 定义格式:

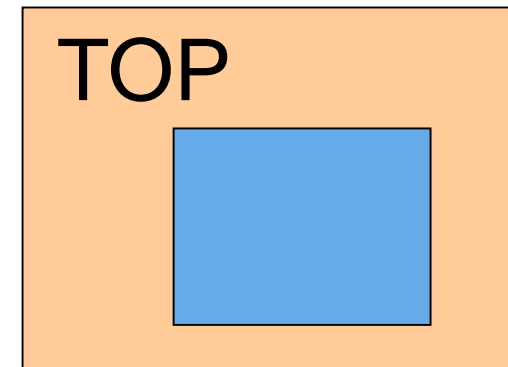
parameter 参数名1=表达式1, ..., 参数名n=表达式n;

- 其中, 表达式既可以是**常数**, 也可以是**表达式**。参数定义完以后, 程序中出现的所有的参数名都将被替换为相对应的表达式。
- parameter length=32,weight=16;//定义了两个参数
- 属于**常量**, 常用来定义**延迟时间和变量的位宽**。
- 在模块或实例引用时, 可通过参数传递改变在**被引用模块或实例中已定义的参数**。
 - 参数定义语句 (**defparam**) ;
 - 带参数值的模块引用。

参数型 (parameter)

```
module TOP (NewA, NewB, NewS, NewC);  
input  NewA, NewB;  
output NewS, NewC;  
defparam Ha1.OR_DELAY=5, //实例Ha1中的参数XOR_DELAY。  
        Ha1.AND_DELAY=2; //实例Ha1中参数的AND_DELAY。  
HA Ha1(NewA, NewB, NewS, NewC);  
endmodule
```

```
module HA(A,B,S,C);  
input A,B;  
output S,C;  
parameter AND_DELAY=1,  
        OR_DELAY=2;  
assign #OR_DELAY  S=A+B;  
assign #AND_DELAY C=A&B;  
endmodule
```

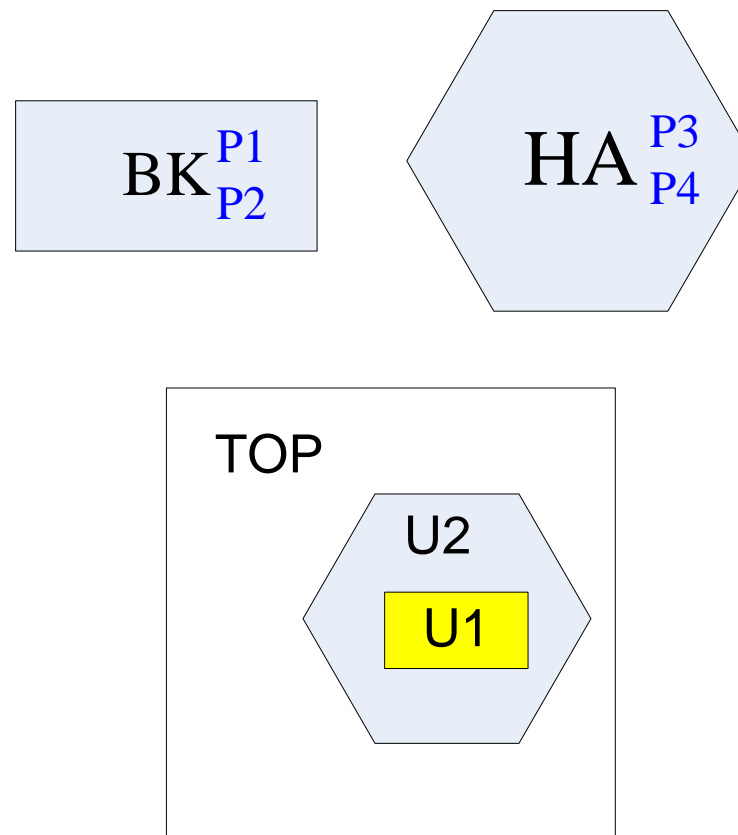


参数型 (parameter)

```
module TOP3(NewA,NewB,NewS,NewC);  
input NewA, NewB;  
output NewS,NewC;  
HA #(5,2) Ha1 (NewA, NewB, NewS, NewC);  
//第1个值5赋给参数AND_DELAY, 该参数在模块HA中说明。  
//第2个值2赋给参数XOR_DELAY, 该参数在模块HA中说明。  
endmodule
```

参数型 (parameter)

- ❖ 假定一个模块为BK,内部两个参数P1, P2; 另外一个模块为HA,内部参数P3, P4
- ❖ 其在TOP模块中实例化调用的名称分别为U1, U2
- ❖ 请尝试在top模块中用两种不同的方法改变其参数值
p1-p4分别为1,2,3,4



变量

□ 在Verilog中有两大主要数据类型：线网类型(wire)、寄存器类型(reg)。

□ 线网类型

包含下述不同种类的线网子类型。

wire, tri 用于连线的最常见的线网类型

wor, trior 线或

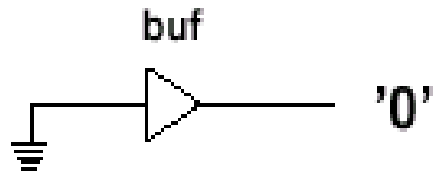
wand, triand 线与

triereg 此线网存储数值，用于电容节点的建模

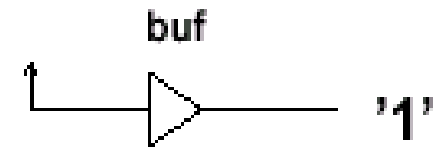
tri1, tri0 用于线逻辑的建模，上拉或下拉驱动

supply0, supply1 supply0用于对“地”建模，supply1对电源建模

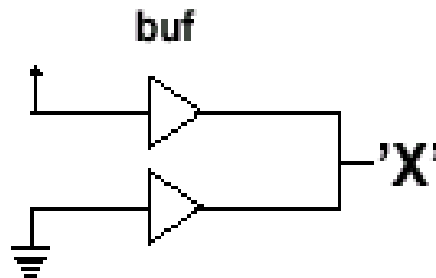
四种逻辑值



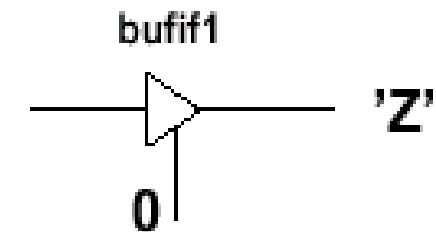
低电平、逻辑0、“假”、接地



高电平，逻辑1、“真”



不确定或未知的逻辑状态



高阻态

线网类型（wire）

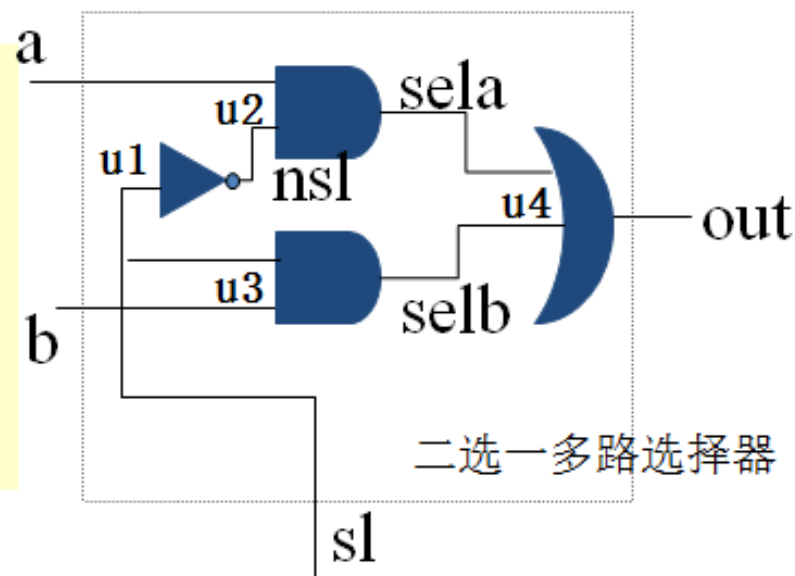
硬件电路中元件之间实际连线的抽象。（如器件的管脚，内部器件如与非门的输出等）。驱动端信号的改变会立刻传递到输出的连线上。

- 不存贮逻辑值，必须由器件驱动。通常由assign 进行赋值。

如 assign $Y = \sim (A \& C)$;

- 当一个wire 类型的信号没有被驱动时，缺省值为Z（高阻）。
- 信号没有定义数据类型时，缺省为 wire 类型。

```
module muxtwo (out, a, b, sl); //二选一多路选择器
input a, b, sl;
output out;
    not u1 (nsl, sl);           //nsl=~sl
    and #1 u2 (sela, a, nsl);   //sela=a&nsl
    and #1 u3 (selb, b, sl);    //selb=b&sl
    or  #2 u4 (out, sela, selb); //out=sela|selb
endmodule
```



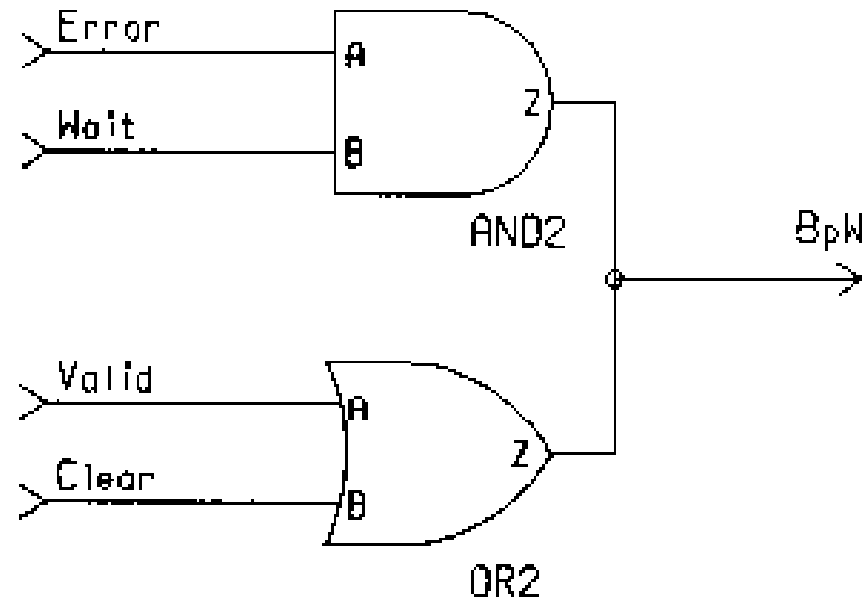
线网类型（wire）

例:

```
wire BpW;
```

```
assign BpW=Error & Wait;
```

```
assign BpW=Valid | Clear;
```

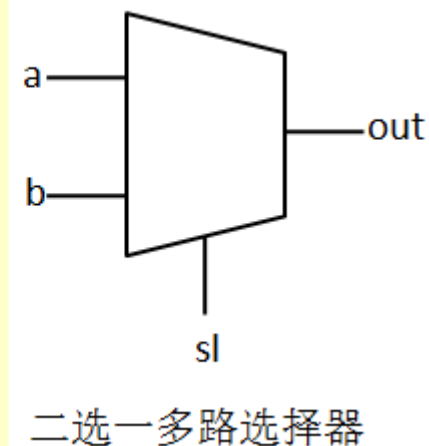


BpW电路图

寄存器类型（reg）

- reg型的变量具有状态保存的作用。综合后常常是寄存器或触发器的输出，但不一定总是这样。
- 在过程块“always”块内被赋值的每一个信号都必须定义成reg型。
- reg型的变量只能在initial或always过程语句的内部被赋值。
- 没有赋值情况下默认为不定值x。
- reg型和wire型的区别：
 - reg型保持最后一次赋值
 - wire型需要持续的驱动

```
module muxtwo (out, a, b, sl); //二选一多路选择器
    input a, b, sl; //输入信号名
    output out; //输出信号名
    reg out;
    always @(sl or a or b)
        if (!sl) out = a;
            //控制信号sl为非，输出与输入信号a一致
        else out = b;
            //控制信号sl为非，输出与输入信号b一致
endmodule
```



如何选择正确的数据类型？

- **输入端口（input）**：可以由寄存器或线网连接驱动，但它本身只能驱动线网连接。
- **输出端口（output）**：可以由寄存器或线网连接驱动，但它本身只能驱动线网连接。
- **输入/输出端口（in/out）**：只可以由线网连接驱动，但它本身只能驱动线网连接。
- 如果信号变量是在过程块（`initial`块 或 `always`块）中被赋值的，必须把它声明为寄存器类型变量

如何选择正确的数据类型？

```
module U(Y, A, B_);
```

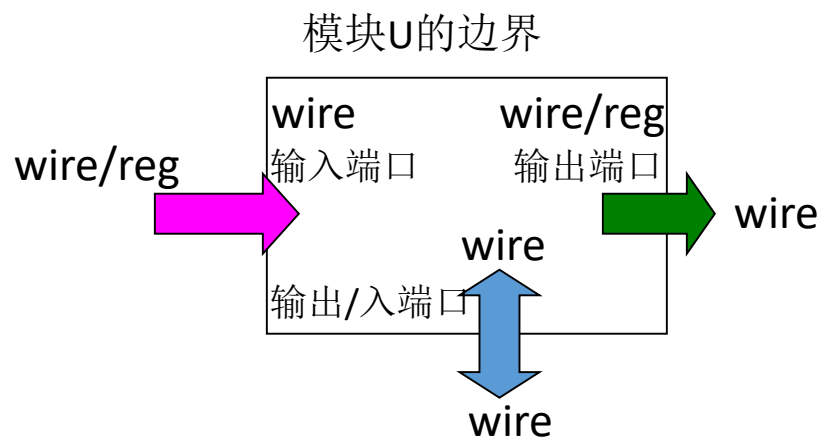
```
output Y;
```

```
input A,B;
```

```
wire Y, A, B;
```

```
and (Y, A, B);
```

```
endmodule
```



```
module top;
```

```
wire y;
```

```
reg a, b;
```

```
U u1(y,a,b);
```

```
initial
```

```
begin
```




```
    a = 0; b = 0;
```

```
    #10 a =1; ....
```

```
end
```

```
endmodule
```

如何选择正确的数据类型？

- 在过程块（**always initial**）中对变量赋值时，忘了把它定义为寄存器 类型（**reg**）或已把它定义为连接类型了（**wire**）
 - 把实例的输出连接出去时，把它定义为寄存器类型
 - 把模块的输入信号定义为寄存器类型。
- 这是经常犯的三个错误！！！！

Verilog HDL语法

- 模块的结构
- 标识符和数据类型
- 运算符及表达式
- 赋值语句和块语句
- 条件语句和循环语句
- 模块的调用
- 模块的测试

运算符

- 按功能分为以下几类：

- ① 算术运算符：+，-，*，/，%
- ② 逻辑运算符：&&，||，!
- ③ 位运算符：~，|，^，&，^~
- ④ 赋值运算符：=，<=
- ⑤ 关系运算符：>，<，>=，<=
- ⑥ 条件运算符：?:
- ⑦ 移位运算符：<<，>>
- ⑧ 拼接运算符：{}
- ⑨ 等式运算符：==，!=，===，!==
- ⑩ 其他

运算符

- **算术运算符：** +, -, *, /, %

- 在进行整数的除法运算时，结果要略去小数部分，只取整数部分；而进行取模运算时（%，亦称作求余运算符）结果的符号位采用模运算符中第一个操作数的符号。
- 在进行算术运算时，如果**某一个操作数有不确定的值x**，则整个结果也为不确定值x。

- **逻辑运算符：** && , || , !

- 其中&&和||是双目运算符，其**优先级别低于关系运算符**，而！高于算术运算符。

- **位运算符：** ~, |, ^, &, ^~

- 在不同长度的数据进行位运算时，系统会自动的将两个数右端对齐，位数少的操作数会在相应的高位补0，一时的两个操作数**按位进行操作**。

运算符

- **关系运算符：** $>$, $<$, $>=$, $<=$
 - 如果关系运算是假的，则返回值是0，如果关系是真的，则返回值是1。
 - 关系运算符的优先级别低于算数运算符。如： $a < \text{size} - 1$ 等同于 $a < (\text{size} - 1)$
 - 如果某个操作数值不定，则关系是模糊的，返回值是不定值。
- **移位运算符：** $<<$, $>>$
 - $a >> n$ 其中 a 代表要进行移位的操作数， n 代表要移几位。这两种移位运算都用0来填补移出的空位。如果操作数已经定义了位宽，则进行移位后操作数改变，但是其位宽不变。
- **拼接运算符：** $\{ \}$
 - $\{ \text{信号1的某几位}, \text{信号2的某几位}, \dots, \text{信号n的某几位} \}$ 将某些信号的某些为列出来，中间用逗号分开，最后用大括号括起来表示一个整体的信号。在位拼接的表达式中不允许存在没有指明位数的信号。

运算符

- 等式运算符： $==$ ， $!=$ ， $===$ ， $!==$
 - $==$ ， $!=$ ：X和Z进行比较时为X
 - $===$ ， $!==$ ：操作数相同结果为1，常用于case表达式的判别。

$===$	0	1	x	z		$==$	0	1	x	z
0	1	0	0	0		0	1	0	x	x
1	0	1	0	0		1	0	1	x	x
x	0	0	1	0		x	x	x	x	x
z	0	0	0	1		z	x	x	x	x

运算符优先级别表

优 先 级 别	
<div>! ~ * / % + - << >> < <= > >= == != === !== & & ^ ~ && ?:</div>	<div>高 优 先 级 别</div> <div>↓</div> <div>低 优 先 级 别</div>

逻辑门的描述

	Verilog描述	逻辑表达式
与门	$F = A \& B;$	$F=AB$
或门	$F = A \mid B;$	$F=A+B$
非门	$F = \sim A;$	$F=A'$
与非门	$F = \sim (A \& B);$	$F=(AB)'$
或非门	$F = \sim (A \mid B);$	$F=(A+B)'$
与或非门	$F = \sim((A \& B) \mid (C \& D));$	$F=(AB+CD)'$
异或门	$F = (A \wedge B);$ $F = (\sim A \& B) \mid (A \& \sim B);$	$F=A \oplus B$ $F=A'B+AB'$
同或门	$F = (A \sim \wedge B);$ $F = \sim (A \wedge B);$ $F = (\sim A \& \sim B) \mid (A \& B);$	$F=(A \oplus B)'$ $F=A'B'+AB$