Verilog硬件描述语言

实验与创新实践教育中心

Verilog硬件描述语言

- Verilog HDL概述
- 结构化设计与Verilog模块
- 标识符和数据类型
- 运算符及表达式
- 赋值语句和过程块（组合逻辑电路描述）
- 条件语句和条件表达式
- 模块的测试

数字设计的发展

电子系统设计自动化EDA
（Eletronic Design Automation）

计算机辅助工程CAE
（computer Assist Engineering）

计算机辅助设计CAD
（computer Assist Design）

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**：从概念、算法、协议等设计电子系统
- 计算机：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

Description ≠ Design

描述什么：

- 在不同抽象级别上描述电路
- 描述电路的连接
- 描述电路的功能
- 描述电路的时序

- **Verilog HDL**：一种硬件描述语言，以文本形式来描述数字系统硬件的结构和行为的语言，可以表示逻辑电路图、逻辑表达式，也可以表示数字逻辑系统所完成的逻辑功能。

Verilog语言的层次

• 行为描述语言&结构描述语言

系统级：

用高级语言结构实现设计模块的外部性能的模型；

算法级：

用高级语言结构实现设计算法的模型；

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；

门级：

描述逻辑门以及逻辑门之间的连接的模型；

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。

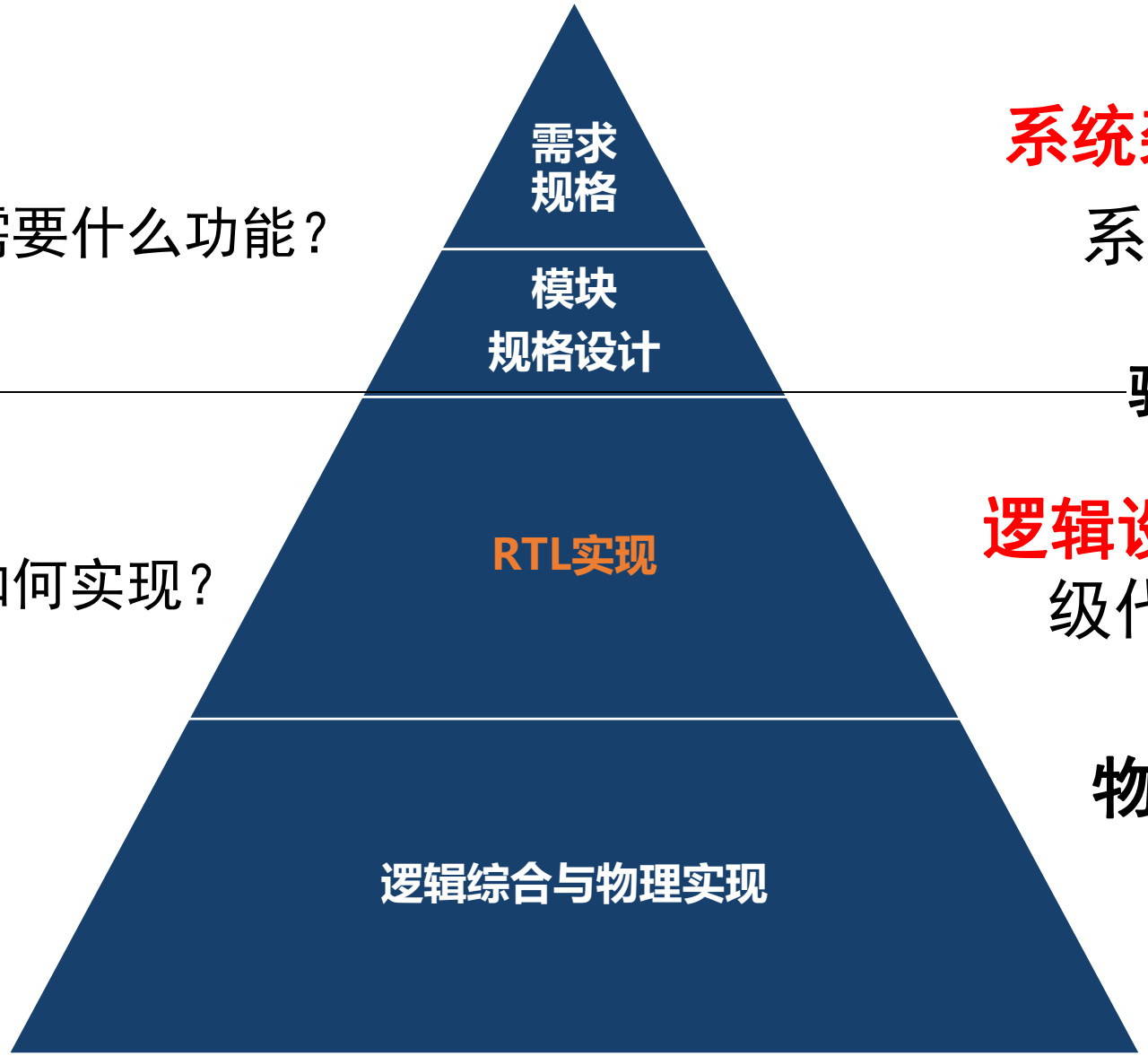
行为级描述

侧重对模块行为功能的抽象描述

结构级描述

侧重对模块内部结构实现的具体描述

设计层次



需要什么功能？

如何实现？

系统架构师：

系统级或算法级，描述系统的规格。

验证工程师：对设计好的电路进行验证

逻辑设计工程师：使用Verilog RTL级代码，可综合，精确到时钟周期。

物理设计工程师：

对门级网表进行布局布线，将其做成实际的芯片。

描述方式

```verilog
module decoder_38_struct(
    input [2:0] data_in,
    input [2:0] en,
    output reg[7:0] data_out
);

    // 内部连线定义
    wire en_out,en0_out,en1_not;

    // 调用基本库元件
    not not0(en0_out, en[0]);
    not not1(en1_out, en[1]);
    and and0(en_out,en0_out,en1_not,en[2]);
endmodule
```

结构化描述

```verilog
module decoder_38_dataflow(
        input   [2:0] data_in,
        input   [2:0] en,
        output  [7:0]data_out
);

assign data_out[5]=data_in[2]&&~data_in[1]
                              &&data_in[0];
// 或 assign data_out[5]=(data_in == 3'd5);

// 其他赋值语句

endmodule
```

数据流描述

```verilog
module decoder_38(
        input   [2:0] data_in,
        input   [2:0] en,
        output reg[7:0]data_out
    );

    always @( * ) begin  //非完整实现，需要自行补全其他信号处理
        case( data_in )
            3'b000:     data_out = 8'b0000_0001;
            3'b001:     data_out = 8'b0000_0010;
            3'b010:     data_out = 8'b0000_0100;
            3'b011:     data_out = 8'b0000_1000;
            3'b100:     data_out = 8'b0001_0000;
            3'b101:     data_out = 8'b0010_0000;
            3'b110:     data_out = 8'b0100_0000;
            3'b111:     data_out = 8'b1000_0000;
        endcase
endmodule
```

行为描述

结构化描述：通过调用库中的元件或已设计好的模块来完成设计。

数据流描述：主要使用assign连续赋值语句，多用于组合逻辑电路。

行为描述：从电路的功能出发，关注逻辑电路输入、输出的因果关系。

即在何种输入条件下产生何种输出，描述的是一种行为特性。

在较复杂的电路设计中，三种描述方法往往混合使用。

RTL与综合

- **综合：**将**HDL语言**、**原理图**等设计输入翻译成由与、或、非等基本逻辑单元组成的门级连接，并根据设计目标和要求优化所生成的逻辑链接，输出**门级网表文件**。
- **RTL级语言最重要的特性就是可综合。**
- **典型的RTL级设计包括：**

  组合逻辑描述、时序逻辑描述、时钟域描述

学生的总结

- **Verilog是硬件描述语言**，用语言描述的方式进行电路设计，最终要实现出硬件电路。verilog只是简化了电路设计的工作量，本质上就是设计数字电路，永远绕不开电路这点！
- 评价一个verilog代码的好坏是看最终实现的功能和性能。合理的设计方法是首先理解要设计的电路，也就是把需求转化为数字电路，对电路的结构和连接十分清晰，然后再用verilog表达出这段电路。
- 要理解硬件并行性，**搞清时序关系**，一定要摒弃顺序执行的思维。
- 建议采用“**学、用、查**”方式，Verilog“用”着“用”着就会了。

初学Verilog的几点提醒

要知道 verilog 是硬件描述语言，它应该是用于描述电路的。但是它的语法较为宽松，以致于将其用于描述算法时也能符合语法规则。但是如果这么做 vivado 会不开心，因为这样做出来的电路往往是不可综合的。综合，简单的说就是按照代码中你对你所设计的电路的描述，生成你所期望的电路。但是如果你一开始脑海里就没有过任何电路，你用 verilog 描述了一个算法，然后扔给 vivado：“给爷综！”那你也太难为人家了。

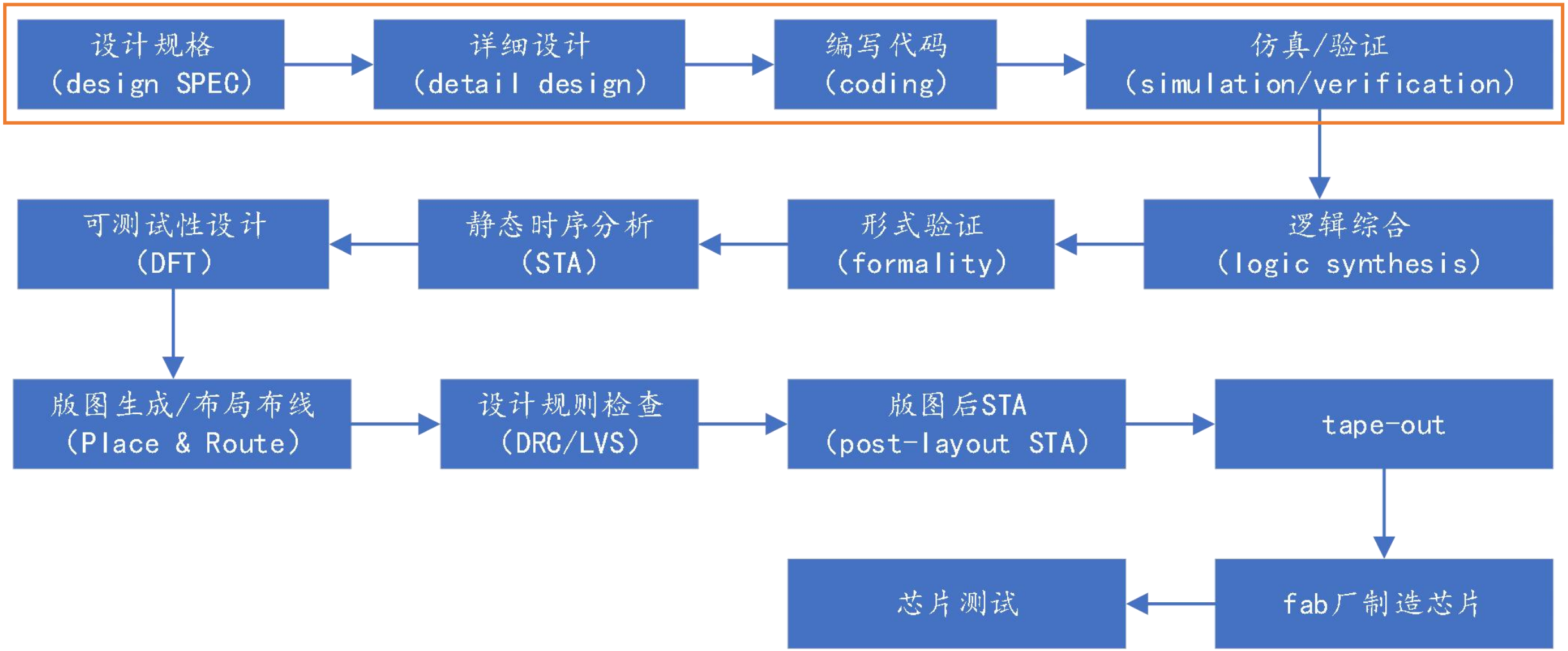
有的读者可能会疑问：电路是什么呢？怎样才叫设计了一个电路呢？这就回归到数字逻辑设计课程理论课部分的内容了。这么说吧，电路就是由逻辑门，锁存器，触发器，导线等组成的一个。。。呃。。。东西。设计电路，就是画出一张电路图，可以用软件画，也可以在纸上画，或者更随便一点，在脑海里画。在哪里画并不重要，重要的是你设计出来的真的是一个电路，而不是算法。

Verilog语言中只有很少一部分是用于设计电路的

Verilog硬件描述语言

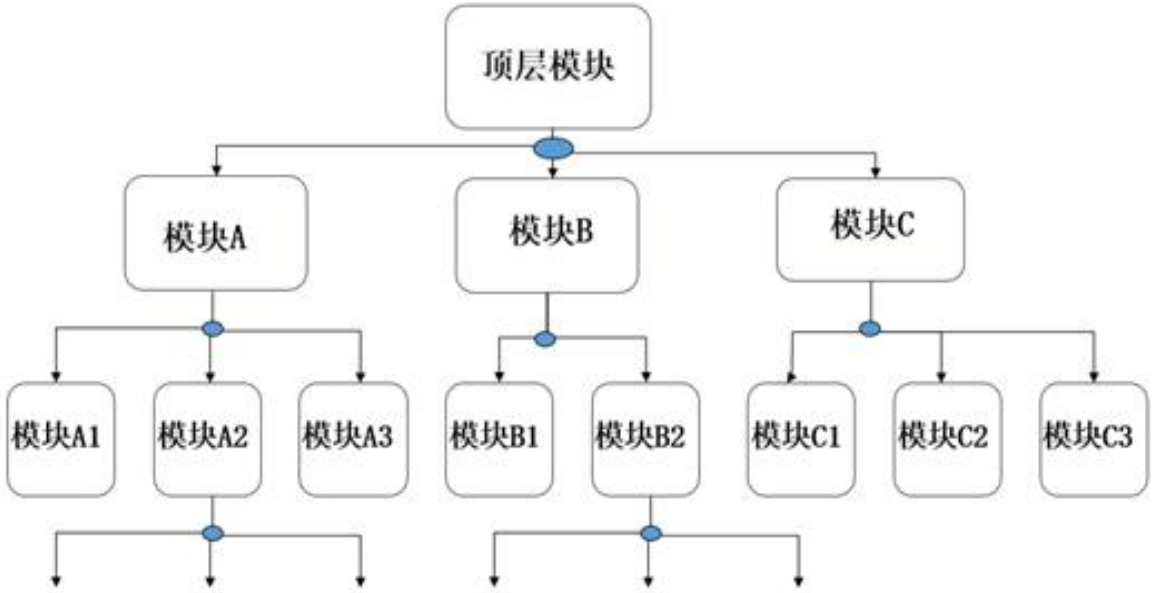
- 概述
- 结构化设计与Verilog模块
- 标识符和数据类型
- 运算符及表达式
- 赋值语句和过程块（组合逻辑电路描述）
- 条件语句和条件表达式
- 模块的测试

数字集成电路开发流程


设计规格
(design SPEC)
详细设计
(detail design)
编写代码
(coding)
仿真/验证
(simulation/verification)
逻辑综合
(logic synthesis)
形式验证
(formality)
静态时序分析
(STA)
可测试性设计
(DFT)
版图生成/布局布线
(Place & Route)
设计规则检查
(DRC/LVS)
版图后STA
(post-layout STA)
tape-out
fab厂制造芯片
芯片测试

Verilog的设计方法-自顶向下的结构化设计

- 从顶层模块开始，先确定好顶层模块的输入输出和内部的逻辑功能，然后逐层分解成更小的功能模块，总体的设计步骤就是“**自顶向下**、**模块划分**、**逐层细化**”，直到子模块的功能较为纯粹、单一。最终采用Verilog语言直接描述硬件行为，由逻辑综合工具自动完成从HDL到门级电路的转换。

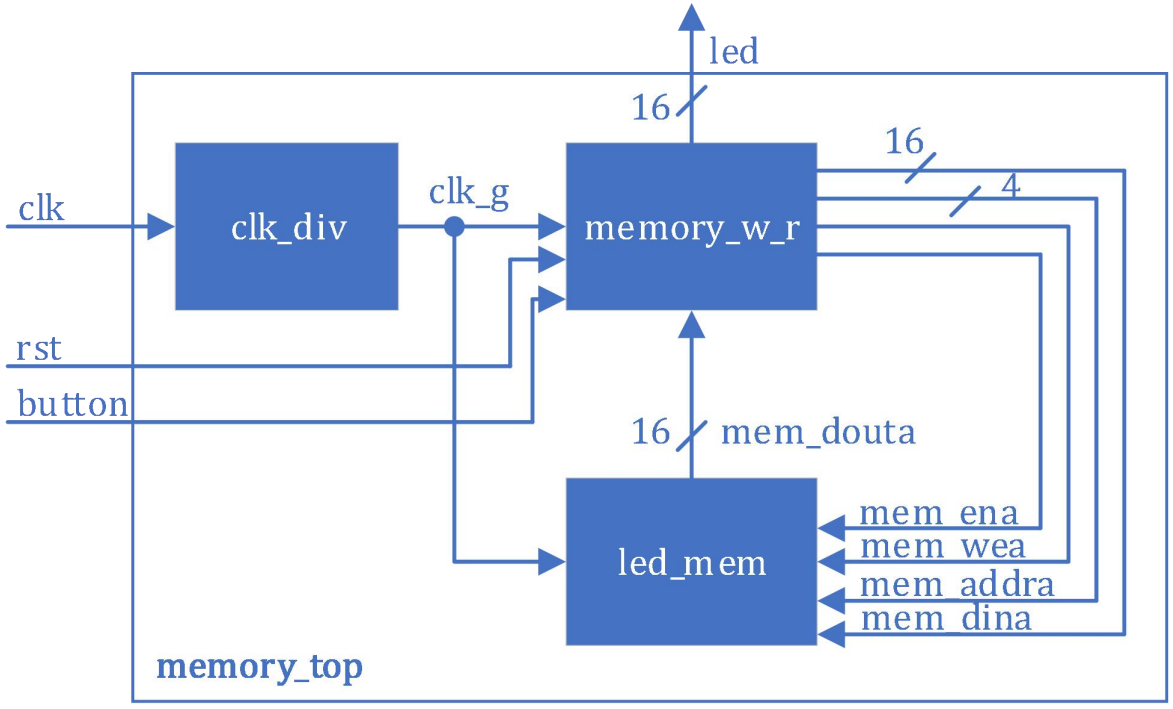


- **模块划分**
- **功能说明**

  描述实现的功能，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（包括必要的说明）。
- **设计代码**

设计框图

- 系统的输入、输出、功能模块及其描述、内部数据通路及重要的控制信号。



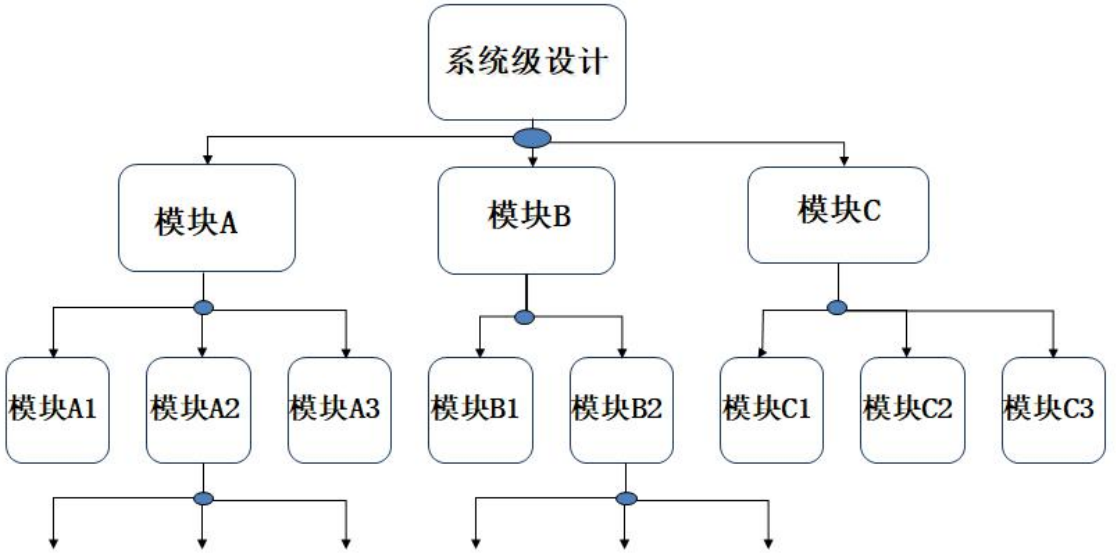
流水灯系统控制框图

- 系统分为3个模块：
- clk_div：用于时钟分频；
- led_mem：用于存储LED灯依次显示的序列；
- memory_w_r：用于控制存储器的读写，以及将读取回来的序列显示到LED上；

从“模块”开始搭电路-模块定义

- Verilog的基本设计单元是模块（module），模块的实例化建立了模块描述的层次，通过模块端口的连接，把下层模块连接到了上层模块中。
- 模块包括接口描述和逻辑功能描述两部分。（电路外特性）
- 模块定义包括4个主要部分（在module和endmodule 之间）

  端口定义、I/O说明、内部信号声明和功能定义



从“模块”开始搭电路-模块定义

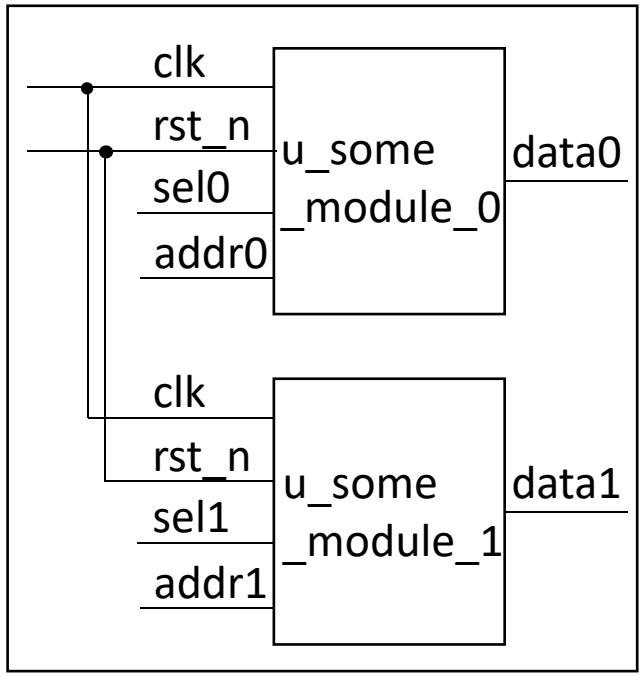
```
module module_name(
        input wire [width-1:0] port_name1,
        input wire [width-1:0] port_name2,
        output wire/reg [width-1:0] port_name3
    ); //在模块名后面的()中做端口定义和I/O说明，行末用逗号而不是分号。


    //内部信号声明
    wire[width-1:0] name1;
    reg [width-1:0] name2;

    //功能定义
    //用assgin语句和always块描述

endmodule
```

```
module some_module (
        input  wire        clk  ,
        input  wire        rst_n,
        input  wire        sel  ,
        input  wire [3:0]  addr ,
        output reg         data
);

//模块代码

endmodule
```

从“模块”开始搭电路-模块实例化

- 一个模块在另外一个模块中实例化，等效于实际电路中加入了被实例化的电路。
- 模块实例应用u_x_x表示（多次例化用序号0、1、2等表示）；

```
module_name instance_name(
.port_name1(data_name1),
.port_name2(data_name2),
……);
```




```
wire        sel0 ;
wire        sel1 ;
wire [3:0]  addr ;
wire        data0;
wire        data1;

some_module u_some_module_0 (
    .clk    (clk  ),
    .rst_n  (rst_n),
    .sel    (sel0 ),
    .addr   (addr ),
    .data   (data0)
);

some_module u_some_module_1 (
    .clk    (clk  ),
    .rst_n  (rst_n),
    .sel    (sel1 ),
    .addr   (addr ),
    .data   (data1)
);
```

Verilog硬件描述语言

- 概述
- 结构化设计与Verilog模块
- 标识符和数据类型
- 运算符及表达式
- 赋值语句和过程块（组合逻辑电路描述）
- 条件语句和条件表达式
- 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。
- 注释有两种：
  - 以“/*”开头，以“*/”结束。　　/*这是注释*/
  - 以“//”开头到本行结束。　　//这也是注释

数据类型

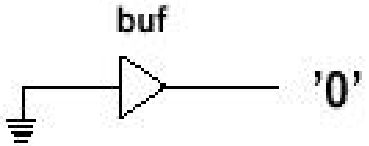
- 共有19种数据类型，分为物理数据类型和抽象数据类型
- 物理数据类型：与实际硬件电路有明显的映射关系
  - wire(连线型)、reg(寄存器型)、memory(存储器型)等
- 抽象数据类型：用于进行辅助设计和验证的数据类型
  - 整型integer、时间型time、实型real、参数型parameter等
- 数据类型还可分为常量和变量
  - 常量：数字、参数型parameter
  - 变量：wire(连线型)、reg(寄存器型)、(memory)存储器型等

常量—整数

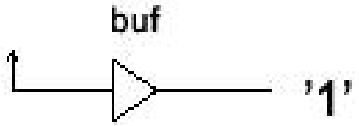
- 二进制(b或B)、十进制(d或D)、十六进制(h或H)、八进制(o或O)
- 表达方式：<位宽><进制><数字>
  - 8'b10101100 //位宽为8的数的**二进制**表示，**'b表示二进制**
    8'ha2 //位宽为8的数的**十六进制**表示，**'h表示十六进制**
  - 没有数字位宽采用默认位宽
    - 由具体的机器系统决定，但至少32位
  - 在<数字>这种描述方式中，采用默认进制（十进制）

四种数值类型（逻辑值）

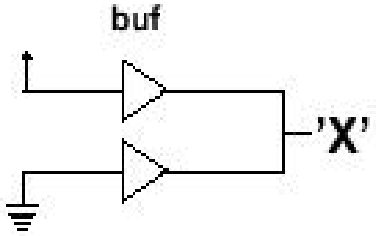
- Verilog语言规定了 4种基本的**数值类型**



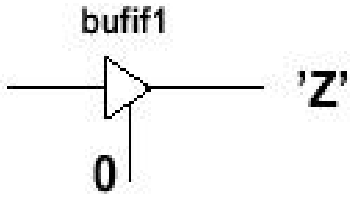
低电平、逻辑0、“假”、接地



高电平、逻辑1、“真”



不确定或未知的逻辑状态



高阻态（一般情况，电路分析时可看作开路）

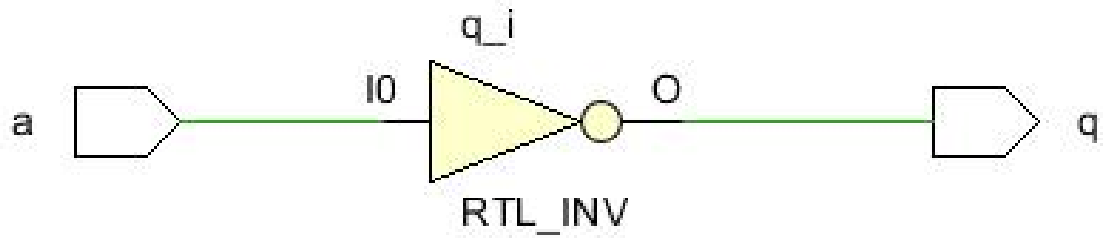
参数型（parameter）

- 定义格式：
  - parameter 参数名1=表达式1，…，参数名n=表达式n;
  - parameter length=32,weight=16;　　//定义了两个参数
  - 其中:表达式k既可以是常数，也可以是表达式。
  - 参数定义完后，程序中所有的参数名将被替换为相应的表达式。
- 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。

wire型

- 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。
- 不存贮逻辑值，必须由器件驱动。
- 通常由assign进行赋值。

  如 assign q = ~ a;



- 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。

reg型

- reg型的变量具有状态保存的作用，保持最后一次赋值。
- reg型的变量只能在initial或always过程语句的内部被赋值。
- 在always块内被赋值的每一个信号都必须定义成reg型，综合后常常是寄存器或触发器的输出，但不一定总是这样，比如，采用always块描述的组合逻辑仍然会为综合为组合逻辑的输出。
- reg类型要在电路复位时赋初值，没赋值情况下默认为不定态“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

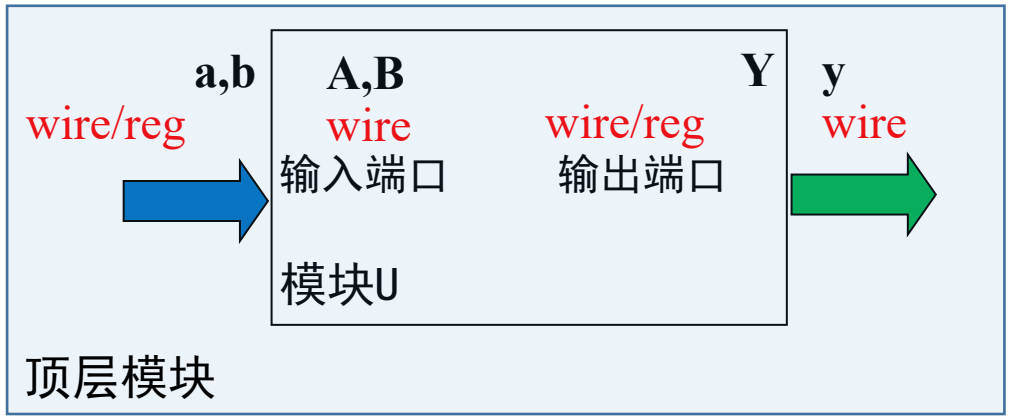
数据类型声明

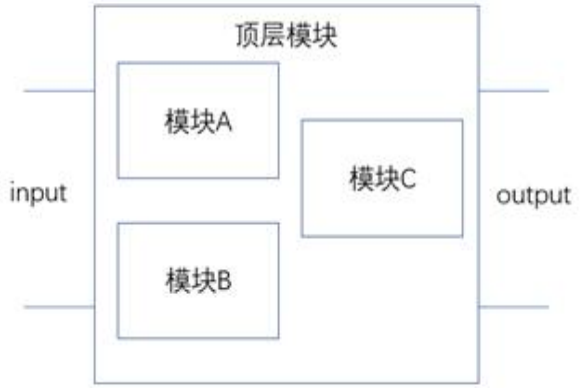
```
reg [width-1:0] 变量名1,变量名2;        //reg型声明
wire [width-1:0] 变量名1,变量名2;       //wire型声明
parameter 参数名1=表达式1,参数名2=表达式2 ;     //参数声明
```

- 模块中用到的所有信号都必须进行数据类型的声明。
- 在Verilog HDL中变量和参数只要在使用前声明即可。
- 声明变量的数据类型后，不能再更改（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类型一致。

如何选择正确的数据类型？

- **端口：（input、output）**
- 可以由reg或wire连接驱动，但它本身只能驱动wire连接。



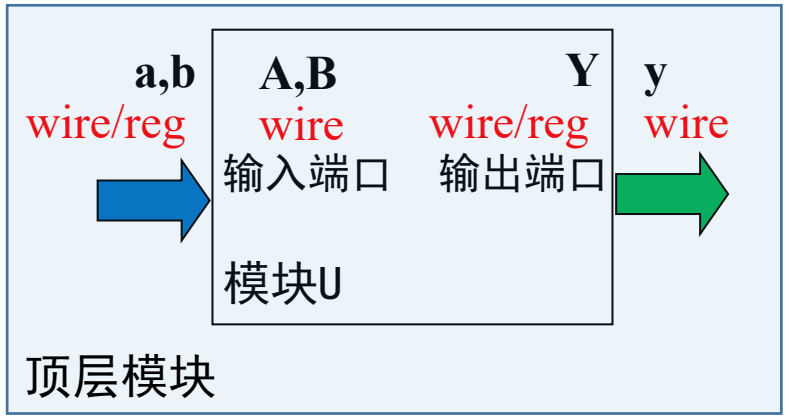


- **变量：**
- assign赋值语句左侧数据类型，必须是wire类型；
- 过程块（always块 或 initial块）中被赋值的左侧数据类型，必须把它声明为reg类型变量。

如何选择正确的数据类型？

模块U：

```
module U(
output wire Y,
input wire A, B);

//功能描述

endmodule
```



顶层模块：

```
module top(
端口定义
) ;
wire y;
reg a, b;

U u1(
.Y(y),
.A(a),
.B(b)
);

//其他逻辑
endmodule
```

模块实例化不是参数传递

模块端口与外部信号按照其名字进行连接

Verilog硬件描述语言

- 概述
- 结构化设计与Verilog模块
- 标识符和数据类型
- 运算符及表达式
- 赋值语句和过程块（组合逻辑电路描述）
- 条件语句和循环语句
- 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：　==，！=，===，！==

运算符—续

- 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；
  - 进行取模运算时（%，亦称求余运算符），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，如有一个操作数为不确定值x，则结果也为不确定值x。
- 逻辑运算符： && ， || ， !
  - &&和||是双目运算符，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0，两个操作数**按位**进行操作。

运算符—续

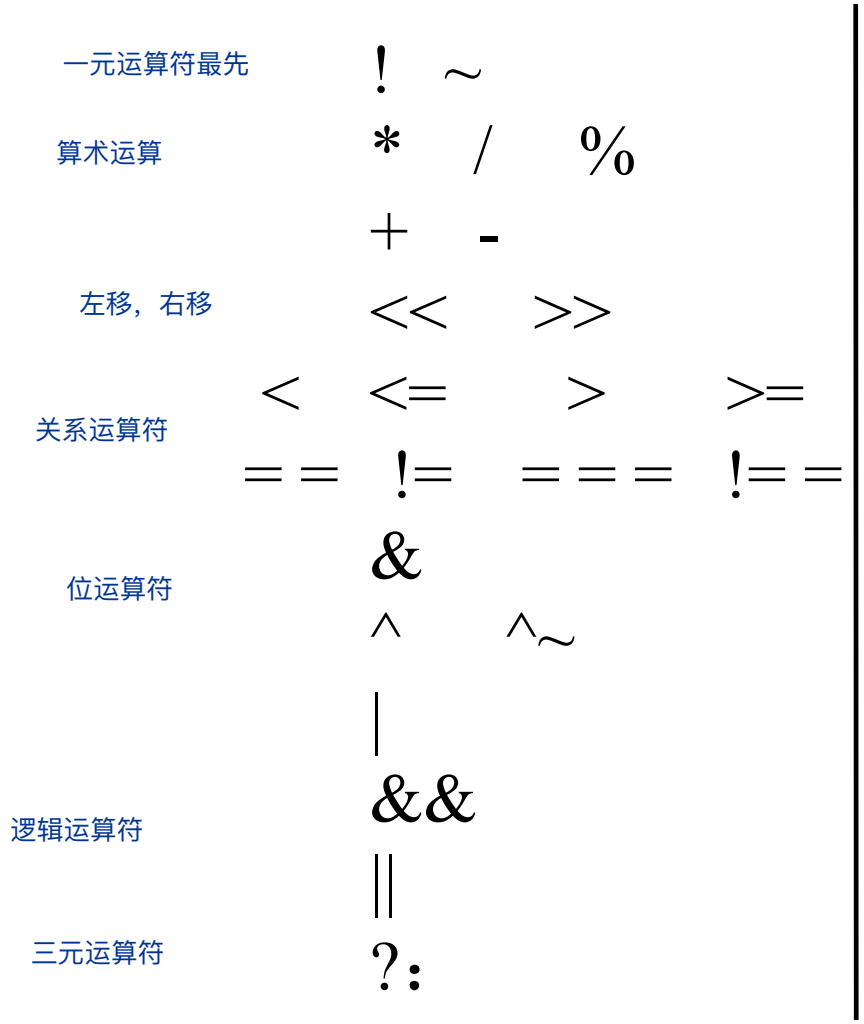
- 关系运算符：＞ ， ＜ ， ＞= ， ＜=
  - 如果关系运算是假的，则返回值是0，如果关系是真的，则返回值是1。
  - 关系运算符的优先级别低于算数运算符。如：a<size-1等同于a<(size-1)
  - 如果某个操作数值不定，则关系是模糊的，返回值是不定值。
- 移位运算符：＜＜ ， ＞＞
  - a>>n其中a代表要进行移位的操作数，n代表要移几位。这两种移位运算都用0来填补移出的空位。如果操作数已经定义了位宽，则进行移位后操作数改变，但是其位宽不变。
- 拼接运算符 ： { }
  - {信号1的某几位，信号2的某几位，.....信号n的某几位} 将某些信号的某些为列出来，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。
  - {a，b[3:0],w}　　//等同于{a,b[3],b[2],b[1],b[0],w}

运算符—续

- 等式运算符：==，！=，===（全等），！==
  - ==，!=：操作数中有X或Z时，结果为X
  - ===，!==：操作数相同结果为1，不同为0，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



高 优 先 级 别

低 优 先 级 别

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 | B ;	F=A+B
非门	F = ~A ;	F=A′
与非门	F = ~ (A & B);	F=(AB)′
或非门	F = ~ (A | B);	F=(A+B)′
与或非门	F = ~((A & B) | (C & D));	F=(AB+CD)′
异或门	F = (A ^ B); F = (~A&B) | (A& ~B);	F=A⊕ B F=A′B+AB′
同或门	F = (A ^~ B); F = ~ (A ^ B); F = (~A&~B) | (A& B);	F=(A⊕ B)′ F=A′B′+AB

Verilog硬件描述语言

- 概述
- 结构化设计与Verilog模块
- 标识符和数据类型
- 运算符及表达式
- 赋值语句和过程块（组合逻辑电路描述）
- 条件语句和循环语句
- 模块的测试

赋值语句

- 在Verilog中，变量**不能随意赋值**，需要使用连续赋值语句或过程赋值语句。
- assign称为**连续赋值**；
- 过程块initial或always中的赋值称为**过程赋值**。

连续赋值语句

- 语法格式：　assign 变量名（wire类型）=赋值表达式;

```
例　　wire  [7:0]  a , b , c, d;
      assign  a = 8'b00001111;
      assign  b = {a[3:0],a[7:4]};
      assign  c[3:0] = d[3:0];
      assign  d = ~a&b;
```

- 对应到电路中 —> 导线;
- 左侧数据类型必须是wire型数据;
- 描述组合电路，每条assign赋值语句相当于一个逻辑单元;
- 右侧任何信号的变化都会立即执行该语句一次，所有右值都是敏感信号;
- 各个assign赋值语句之间是并发的关系;
- 在过程块（initial/always）外面（不能写在过程块中）

连续赋值语句

```
wire [7:0] wire_a;
wire [7:0] wire_b;
reg  [7:0] reg_a;
wire [7:0] wire_y1;
wire [7:0] wire_y2;
wire [7:0] wire_y3;
wire [7:0] wire_y4;
wire [7:0] wire_y5;
wire [7:0] wire_y6;

assign wire_b = wire_a;
assign wire_a = reg_a;
assign wire_y1 = ~wire_a;    //取反
assign wire_y2 = wire_a & wire_b;       //与
assign wire_y3 = wire_a | wire_b; //或
assign wire_y4 = wire_a ^ wire_b; //异或
assign wire_y5 = ~(wire_a & wire_b); //与非
assign wire_y6 = ~(wire_a | wire_b);  //或非
```

//一些常用的基础逻辑门

过程赋值语句

- 只能出现在过程块中（initial/always）；
- 过程赋值语句中：没有关键词“assign”；
- 左侧数据类型必须是reg类型的变量；
- 每条过程赋值语句之间是顺序执行的关系。
- 包括阻塞赋值（运算符=）和非阻塞赋值（运算符<=）

阻塞赋值与非阻塞赋值

- 阻塞(Blocking)赋值方式(符号“=”，如b=a;)
  - 在同一个always块中，一条阻塞赋值语句如果没有执行结束，那么该语句后面的语句就不能被执行，即被“阻塞”
  - b的值在赋值语句执行完后立刻就改变;
  - 用在触发事件为电平敏感信号的always块中，描述组合逻辑电路
- 非阻塞赋值(Non-Blocking)赋值方式(符号“<=”，如b<=a;)
  - 用在触发事件为时钟边沿的always块中，描述时序逻辑电路
  - b的值并不是立刻就改变;

过程块

- 过程块有两种：
  - always块
    - 循环执行。
    - 满足触发条件则执行。
    - 一个模块中有多个always块时，可以并行运行。
  - initial块
    - 只能执行一次。
    - 不带触发条件。
    - 通常用于仿真模块。

```
always @(*)begin
   sum = a + b;
   //其他代码
end
```

```
initial begin
   c=1'b0;
   //其他代码
end
```

always块

- 格式如下：

```
always @(<敏感信号列表>)
    begin
        //过程赋值
        //if-else、case选择语句
        //for、while等循环块
    end
```

```
module reg_adder (
        input wire [2: 0] a, b,
        output wire [3: 0] out
        );
        always @(a or b)
            sum = a + b;//a或b发生变化，执行
endmodule
```

- always语句通常带触发条件，触发条件被写在敏感信号列表中，只有当触发条件满足条件或发生变化时，其后的”begin-end”块语句才能执行。
- 敏感信号分为两种：电平敏感、边沿敏感；
- 敏感信号列表中可以有多个信号，用关键字or连接；
- 既可以描述组合逻辑电路也可以描述时序逻辑电路。

Verilog硬件描述语言

- 概述
- 结构化设计与Verilog模块
- 标识符和数据类型
- 运算符及表达式
- 赋值语句和过程块（组合逻辑电路描述）
- 条件语句和条件运算符
- 模块的测试

条件语句(if,case)

- 必须在过程块中使用

if语句：

```
if（条件表达式1）
        语句块1;
else if(条件表达式2)
        语句块2;
……
else
        语句块n;
```

语句块超过1条语句使用begin…end

case语句：

```
 case（条件表达式）
        分支1：语句块1;
        分支2：语句块2;
        ……
        default：语句块n;
 endcase
```

条件语句if…else与case的区别

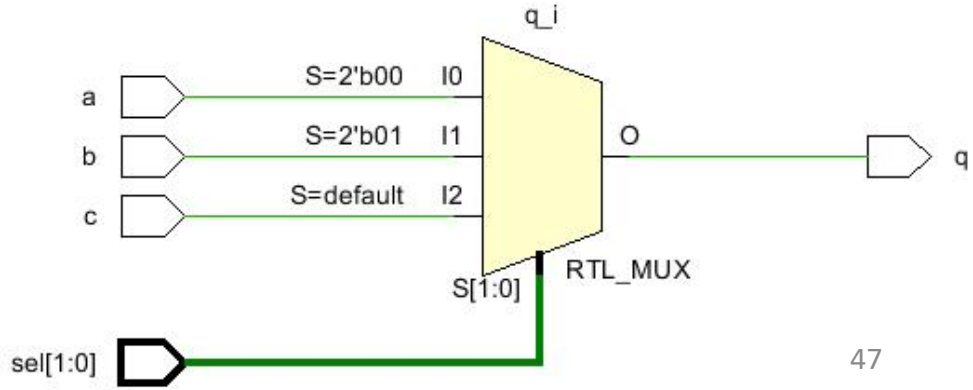
- if生成的电路是串行，是有优先级的编码逻辑；

```
always @ * begin
    if(sela)
        q = a;
    else if(selb)
        q = b;
    else
        q
end
```

- case生成的电路是并行的，各种判定情况的优先级相同。

```
always @ * begin
    case(sel)
        2'b00:  q = a;
        2'b01:  q = b;
        default:    q = c;
```

- if生成的电路延时较大，占用硬件资源少；case生成的电路延时短，占用硬件资源多。



条件运算符

- <条件表达式>?<条件为真>:<条件为假>;

```
assign  q = (sela==1'b1)?a:((selb==1'b1)?b:c);
```

```
always @ * begin
    if(sela)
        q = a;
    else if(selb)
        q = b;
    else
        q = c;
end
```

组合逻辑的Verilog描述-always块

- 使用触发事件为电平敏感信号的always块
- 使用always块描述组合逻辑电路时，需用阻塞赋值
- 将always模块中使用到的所有输入信号和条件判断信号都列在敏感信号列表中（建议使用*）

```
//3-8译码器
always @ * begin
    if (enable) begin
        case (switch)
            3'h0 : decoder_38 = 8'hfe;
            3'h1 : decoder_38 = 8'hfd;
            3'h2 : decoder_38 = 8'hfb;
            3'h3 : decoder_38 = 8'hf7;
            3'h4 : decoder_38 = 8'hef;
            3'h5 : decoder_38 = 8'hdf;
            3'h6 : decoder_38 = 8'hbf;
            3'h7 : decoder_38 = 8'h7f;
            default : decoder_38 = 8'hff;
        endcase
    end
    else decoder_38 = 8'hff;
end
```

组合逻辑的Verilog描述

- 组合逻辑电路是任意时刻的输出仅取决于该时刻的输入，与电路原来状态无关，电路中不包含记忆（存储）元件。
- 通常采用两种常用RTL 级方式来描述组合逻辑电路行为。

  使用assign关键字描述的赋值语句

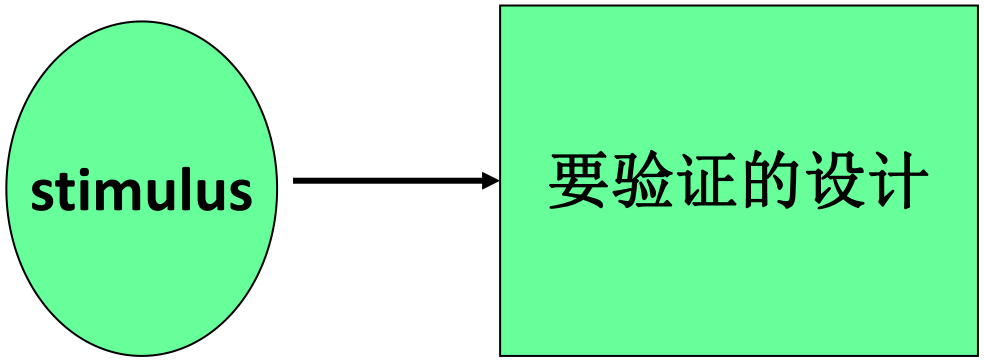
  使用触发事件为电平敏感信号的always块

Verilog硬件描述语言

- 概述
- 结构化设计与Verilog模块
- 标识符和数据类型
- 运算符及表达式
- 赋值语句和过程块（组合逻辑电路描述）
- 条件语句和条件运算符
- 模块的测试

Testbench

- 测试平台（test bench）是一个无输入，有输出的顶层调用模块。
- 一个简单的测试平台包括：
  - 产生激励信号；
  - 实例化待测模块，并将激励信号加入到待测模块中。



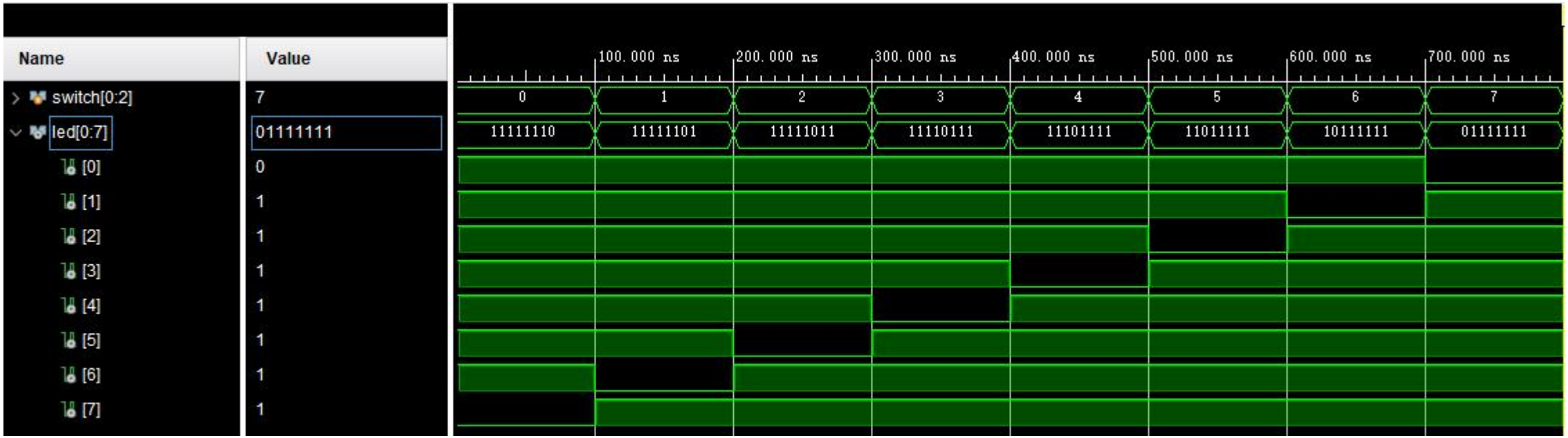
简单的TestBench

Testbench-组合逻辑

```
`timescale 1ns/1ps   //1ns表示延时单位，1ps表示时间精度
module decoder_38_sim();
    reg [2:0] data_in;     // 3位二进制输入
    reg [2:0] en;          // 3位控制信号输入
    wire [7:0] data_out;   // 8位输出

    decoder_38 d_38(
        .data_in(data_in),     // 结合自己的实现完成实例化
        .en(en),
        .data_out(data_out)
    );

    initial
    begin
        #5 begin en = 3'b100;data_in = 3'b000; end  // 构造输入激励信号
        #5 begin en = 3'b100;data_in = 3'b001; end
        #5 begin en = 3'b100;data_in = 3'b010; end
        #5 begin en = 3'b100;data_in = 3'b011; end
        #5 begin en = 3'b100;data_in = 3'b100; end
        #5 begin en = 3'b100;data_in = 3'b101; end
        #5 begin en = 3'b100;data_in = 3'b110; end
        #5 begin en = 3'b100;data_in = 3'b111; end
        #5 begin en = 3'b101;data_in = 3'b000; end   //使能端无效
        #5 $stop ;  // 立即结束仿真
    end
    endmodule
```

- 仿真模块没有输入、输出端口
- 激励信号数据类型要求为reg，以便保持激励值不变，直至执行到下一跳激励语句为止
- 输出信号数据类型要求为wire，以便能随时跟踪激励信号的变化
- initial块只能执行一次，不带触发条件。

仿真分析



100ns时，switch 信号由1'b000变为1'b001，led输出为8'b11111110(低电平有效)；

200ns时，switch 信号由1'b001变为1'b010，led输出为8'b11111101(低电平有效)；

……

700ns时，switch 信号由1'b110变为1'b111，led输出为8'b11111110(低电平有效)；

小结

- Verilog语言概述
  - 行为级描述、结构级描述
  - RTL与可综合
- **模块定义与模块实例化**
- **组合逻辑的Verilog描述**
- **Testbench模块**