



Verilog数字系统设计 课程概述

刘学

13604181486

QQ: 672297876

liuxue@cse.neu.edu.cn

Verilog数字系统设计

1

- 1) 在描述组合逻辑的 always 块中使用阻塞赋值, 则综合组合逻辑的电路结构;
- 2) 在描述时序逻辑的 always 块中使用非阻塞赋值, 则综合时序逻辑的电路结构。

语法要点

- 1) `always`里面赋值左边必须申明成`reg`
- 2) `assign`表达式左边必须申明成`wire`
- 3) 阻塞赋值用`=`
- 4) 非阻塞赋值用 `=>`

- 1) 边沿触发生成寄存器的时序逻辑
- 2) 电平触发条件完整，生成组合逻辑
- 3) 电平触发，条件不完整，生成锁存器的时序逻辑

结论：

申明成`reg`，不一定得到寄存器。

申明成`reg`，也可能得到锁存器

Coding要点

- 如果是边沿触发的逻辑

比如 `always@(posedge clk)`, 里面一律使用 `=>` 赋值

- 如果是电平触发的逻辑，一律使用 `=` 赋值

逻辑简单用 `assign`语句；逻辑复杂，用`always` 语句
分支条件写完整，防止出现锁存器

以下语法只用于仿真

3.6.4 系统任务和系统函数

- 标准输出任务: \$display等
- 文件管理任务: \$fopen等
- 仿真控制任务: \$monitor等
- 时间函数: \$timeformat等
- 时间显示函数: \$time等
- 其它: \$random等

always 语句块里面: 电平触发的是组合逻辑, 边缘触发的是时序逻辑。

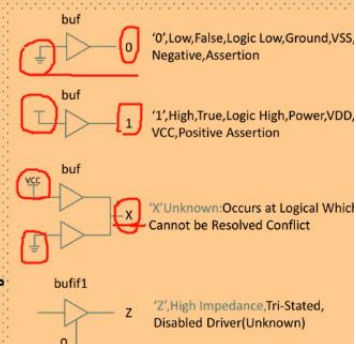
逻辑值

逻辑 0: 表示低电平, 也就对应我们电路 GND;

逻辑 1: 表示高电平, 也就是对应我们电路的 VCC;

逻辑 X: 表示未知, 有可能是高电平, 也有可能是低电平;

逻辑 Z: 表示高阻态, 外部没有激励信号, 是一个悬空状态。



数字进制格式

Verilog数字进制格式包括二进制、八进制、十进制和十六进制。
一般常用的为二进制、十进制和十六进制。

二进制表示如下：4'b0101 表示4位二进制数字0101

十进制表示如下：4'd2 表示4位十进制数字2（二进制0010）

十六进制表示如下：4'ha 表示4位十六进制数字a（二进制1010）

32'd

16'b1001_1010_1010_1001 = 16'h5AA5

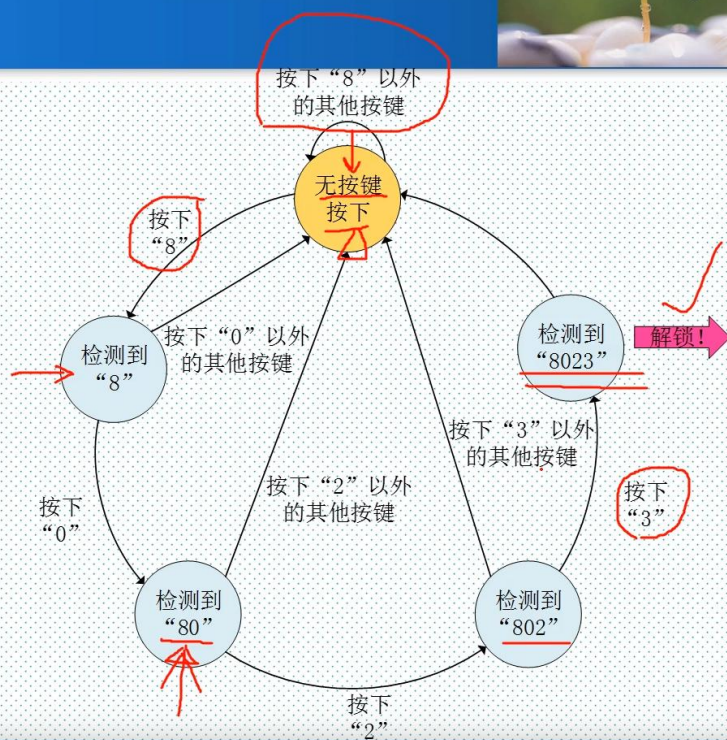
100

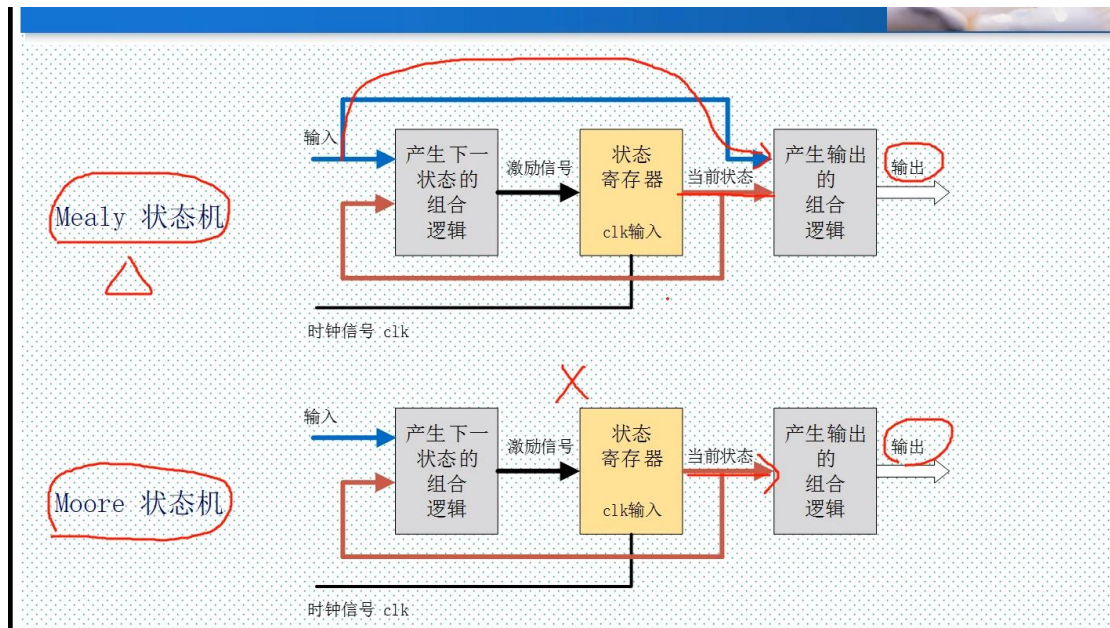
状态机

状态机概念



密码：8023





状态机设计 " 四段论 "

1. 定义状态空间: 列举所有可能出现的状态, 定义当前状态和下一个状态的寄存器

```
//define state space
parameter SLEEP = 2'b00;
parameter STUDY = 2'b01;
parameter EAT = 2'b10;
parameter AMUSE = 2'b11;

// internal variable
reg [1:0] current_state;
reg [1:0] next_state;
```

状态空间可以有多种编码方式: 二进制编码, one-hot 编码等


```
//define state space
parameter SLEEP = 4'b1000;
parameter STUDY = 4'b0100;
parameter EAT = 4'b0010;
parameter AMUSE = 4'b0001;

// internal variable
reg [3:0] current_state;
reg [3:0] next_state;
```

独热码：每个状态只有一个寄存器置位，译码逻辑简单

时序逻辑要用非阻塞赋值，组合逻辑要用阻塞赋值。

阻塞赋值：这一行语句没执行完，不会执行下一行，串行

非阻塞复制：这一行语句是否执行完不影响下一行语句的执行，并行

2. 状态跳转

在时钟上升沿或者下降沿到达时进行状态转换

2 状态跳转 (时序逻辑)

```
// transition
always @ (posedge clk or negedge rst_n) begin
    if (!rst_n)
        current_state <= SLEEP;
    else
        current_state <= next_state;
end
```

敏感列表：
时钟信号以及复位信号边沿的组合

使用非阻塞赋值

锁存器：电平触发的存储器

触发器：边缘触发的存储器

3. 下一个状态的判断

根据当前状态和输入来判断下一个状态应该是什么

3 下个状态判断 (组合逻辑)

```
// next state decision
always @(current state or input signals) begin
    case (current state)
        SLEEP: begin
            if (clock_alarm)
                next_state = STUDY;
            else
                next_state = SLEEP;
            end
        SYUDY: begin
            if (lunch time)
                next_state = EAT;
            else
                next_state = SYUDY;
            end
        EAT: ... ;
        AMUSE: ... ;
        default: ... ;
    endcase
end
```

敏感信号表：
所有的右边表达式中的变量以及if、case条件中的变量

使用阻塞赋值

If/else要配对以避免latch的产生

Case语句要写完整，否则也会出现锁存器

4. 各个状态下的动作

4 各个状态下的动作

```
// action
wire read_book;
assign read_book = (currentn_state == STUDY) ? 1'b1 : 1'b0;
```

```
always @(currentn_state) begin
    if (currentn_state == STUDY)
        read_book = 1;
    else
        read_book = 0;
end
```

使用阻塞赋值

