


**北京航空航天大学**  
Beijing University of Aero. and Astro.  
( BEIHANG University )

**2020**



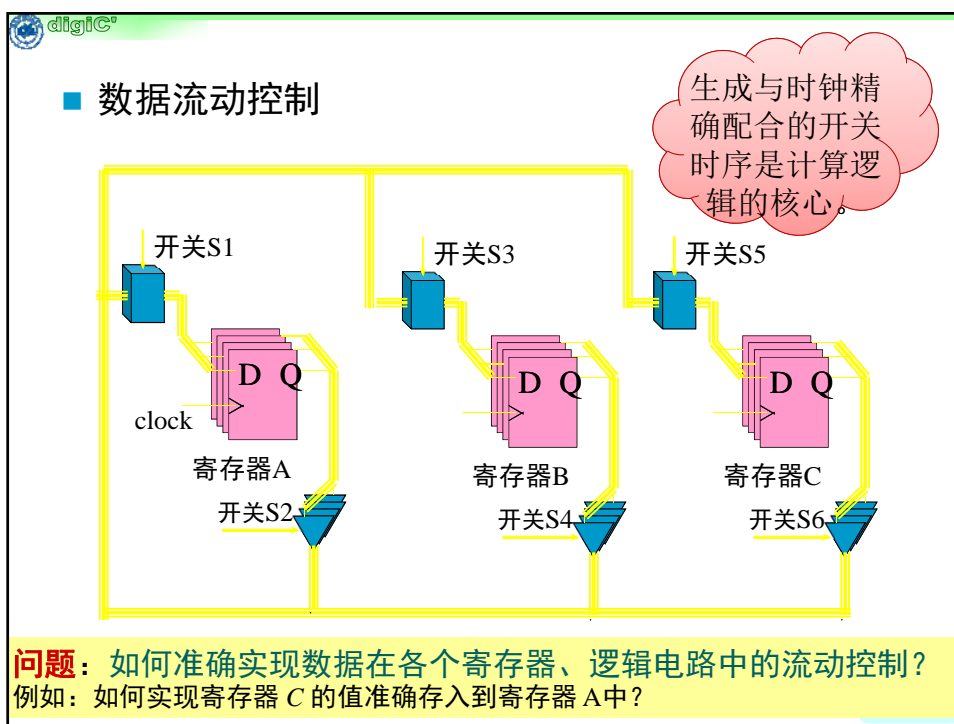
# 数字电路与系统

## 便携式实验

---

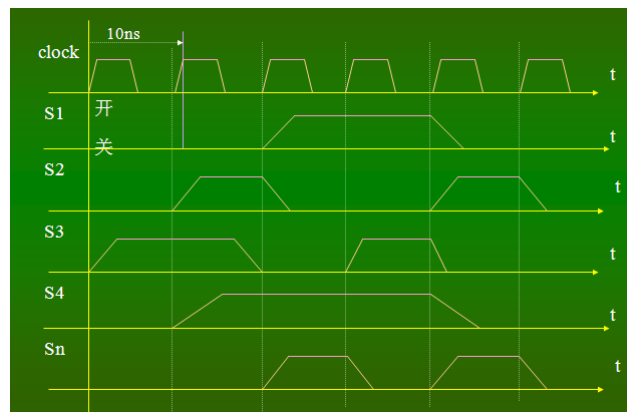
### Verilog硬件描述语言简介 (下)

1





- 如果能严格以时钟跳变沿为前提，按排好的时序，来操作逻辑系统中每一个开关 $S_i$ ，则系统中数据的流动和处理会按同一时钟节拍有序地进行，避免了冒险和竞争现象，时延问题就能有效地加以解决。

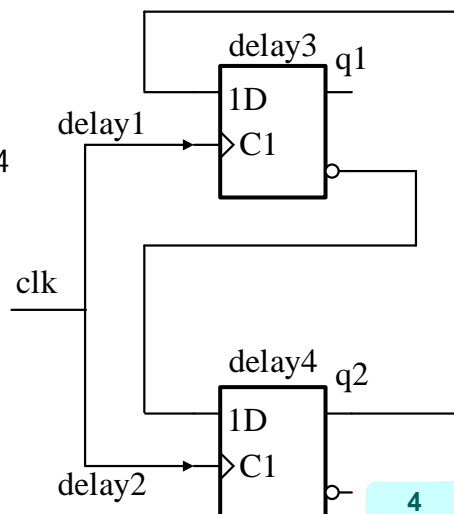


3



### ■ 想象电路实际执行情况：

- 假设  $\text{delay1} > \text{delay2}$
- 假设  $\text{delay2} > \text{delay1}$
- 考虑  $\text{delay1,2} \ll \text{delay3,4}$
- 假设  $\text{delay3} \neq \text{delay4}$
- 稳定吗？



4



## 有限状态机的基本概念

### ■ 有限状态机（Finite State Machine, FSM）

- 是由寄存器组和组合逻辑构成的时序电路，公共时钟信号。
- 状态的改变只可能发生在时钟的跳变沿时。
- 状态是否改变以及如何改变取决于当前状态与输入信号。
- 状态机可用于产生在时钟跳变沿开关的复杂的控制逻辑，是**同步数字逻辑的控制核心**。

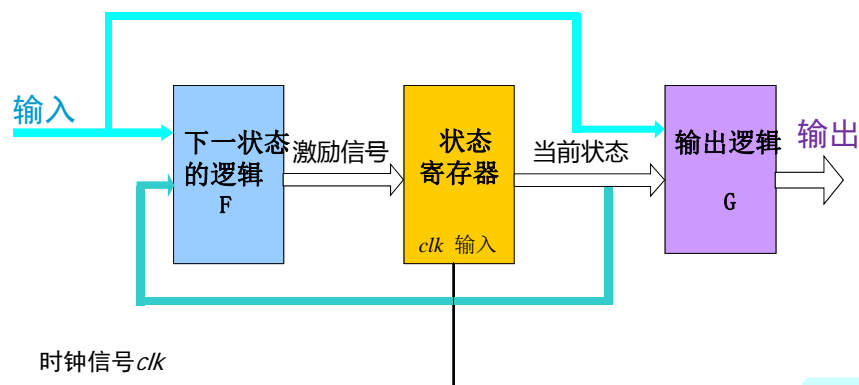
5



## FSM的基本概念（续）

### ■ Mealy状态机

- 下一状态 =  $F$ （当前状态，输入信号）；
- 输出信号 =  $G$ （当前状态，输入信号）；



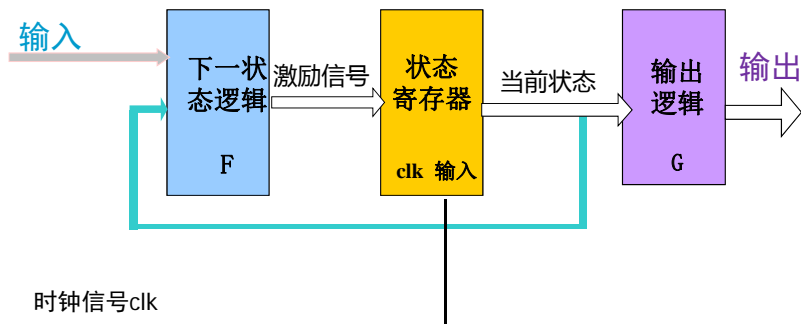
6



## FSM的基本概念（续）

### ■ Moore状态机

- 下一状态=F（当前状态，输入信号）；
- 输出信号=G（当前状态）；



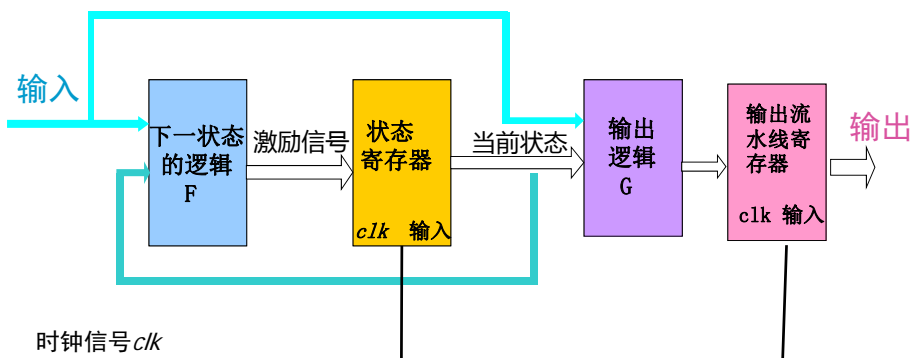
7



## FSM的基本概念（续）

### ■ 带流水线输出的Mealy状态机

- 下一状态=F（当前状态，输入信号）；
- 输出信号=G（当前状态，输入信号）；



8



## 简单的有限状态机设计

- 状态转移图表示
- RTL级可综合的Verilog HDL模块表示

9

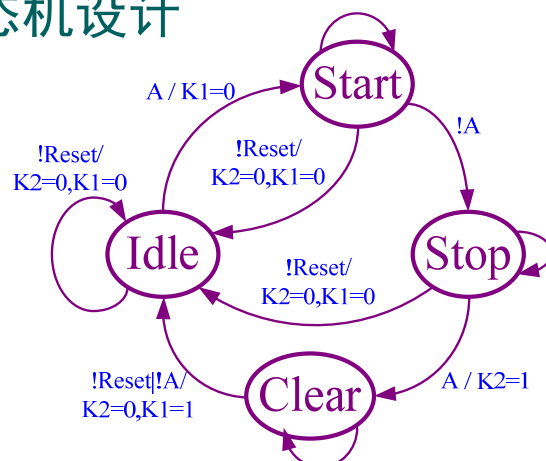


## 简单的有限状态机设计

- 状态转移图表示

图形表示：

- 状态
- 转移
- 条件
- 开关（操作）



10



## 简单的有限状态机设计

### ■ 有限状态机的Verilog HDL描述

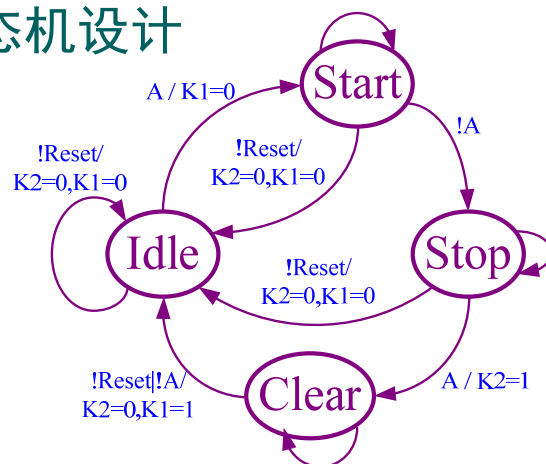
- 1. 定义模块名和输入输出端口
- 2. 定义输入、输出变量或reg变量
- 3. 定义时钟和复位信号
- 4. 定义**状态变量**和**状态寄存器**
- 5. 用时钟边沿触发的always块表示状态转移过程
- 6. 在复位信号有效时给状态寄存器赋初始值
- 7. 描述状态的**转换过程**
- 8. 验证状态转移的正确性，必须完整和全面

11



## 简单的有限状态机设计

### ■ 状态转移图 模块接口定义



```

module MyFSM ( clock, reset_n, a, k1, k2 );
input clock, reset_n, a;           // 定义时钟、复位和输入信号
output k1, k2;                     // 定义输出控制信号的端口
reg k1, k2;                         // 定义输出控制信号的寄存器变量类型
reg [1:0] state ;                  // 定义状态寄存器
  
```

12



## ■ 状态编码

➤ 常用参数定义（也可以采用宏定义，但不方便）

➤ 如：

```
parameter    IDLE  = 2'b00,    START = 2'b01,
              STOP  = 2'b11,    CLEAR = 2'b10; //定义状态变量参数值
```

➤ 又如：独热码（one-hot）

```
parameter    IDLE  = 4'b0001,    START = 4'b0010,
              STOP  = 4'b0100,    CLEAR = 4'b1000; //定义状态变量参数值
```

13



## 三种有限状态机的编写风格（摘自“特权同学”）

### ■ 一段式状态机：

➤ 所有逻辑（输入、输出、状态）都在一个always块中解决

### ■ 两段式状态机：

- 时序逻辑 和 组合逻辑 划分开来
- 时序：当前 和 下一状态 的切换
- 组合：各个输入、输出以及状态判断

### ■ 三段式状态机：

- trivial** 一、初始状态设置；下一时钟边沿，状态变量 $\leq Q^{n+1}$
- 二、根据输入信号的激励和 $Q$ ，计算 $Q^{n+1}$ ，纯组合逻辑
  - 三、对输入信号和 $Q^{n+1}$ 译码，得到输出信号，并在时钟边沿变化

简记为：  $Q$  : current state ;  $Q^{n+1}$  : next state

14

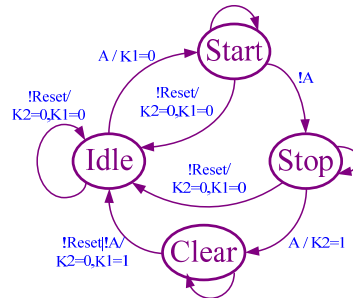


## 简单的有限状态机设计——一段式

```

always @( posedge clock )
    if ( !reset_n ) begin //定义复位后的初始状态和输出值
        state <= IDLE ; k2 <= 0 ; k1 <= 0 ;
    end
    else
        case (state)
            IDLE: begin
                if (a) begin
                    state <= START;
                    k1 <= 0;
                end
                else state <= IDLE;
            end
            START: begin
                if ( !a ) state <= STOP;
                else state <= START;
            end

```



15

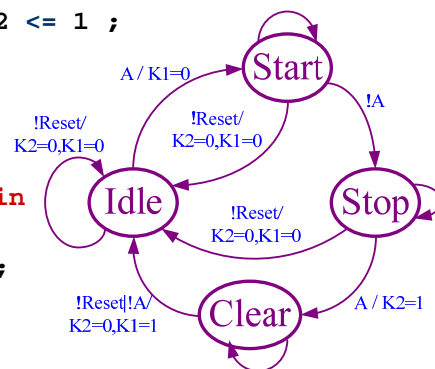


(续)

```

STOP: begin //符合条件进入新状态，否则留在原状态
    if ( a ) begin
        state <= CLEAR ; k2 <= 1 ;
    end
    else state <= STOP ;
end
CLEAR: begin
    if ( !a | !reset_n ) begin
        state <= IDLE ;
        k2 <= 0 ; k1 <= 1 ;
    end
    else state <= CLEAR ;
end
default: state <= IDLE ;
/* 对于无效的状态编码（如：独热码产生的多余状态） *
* 有些状态不可达，增加 default项，确保最后回到IDLE状态 */
// 思考，一定要回到IDLE状态吗？
endcase

```



16





## 简单的有限状态机设计——二段式

### ■ 有限状态机的Verilog HDL描述：二段式

- 思路：把状态的变化与（决定下一状态+输出）分成两部分来考虑
  - 时序逻辑 + 组合逻辑
- 为了扩展和调试方便：除了将分支语句的判断以**always**过程块实现组合逻辑函数，输出信号也**分别**以**always**过程块实现组合逻辑函数
- **优点**：在调试多输出状态机时，比较容易发现问题和改正模块编写中出现的问题。

17



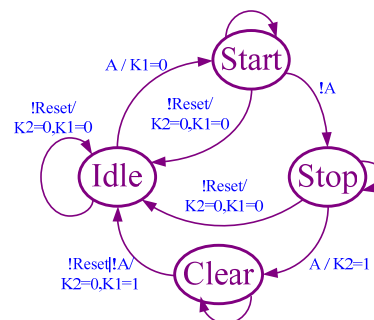
## 简单的有限状态机设计——二段式

### ■ 有限状态机的Verilog HDL描述：二段式

```

module MyFSM (clock, reset_n, a, k2, k1);
input clock, reset_n, a;
output k2, k1;
reg k2, k1;
reg [3:0] state, nextstate ;

parameter
IDLE      = 4'b0001,
START     = 4'b0010,
STOP      = 4'b0100,
CLEAR     = 4'b1000;
  
```



18



## 简单的有限状态机设计——二段式

### ■ 有限状态机的Verilog HDL描述

```
//----- 每一个时钟沿产生一次可能的状态变化-----
always @(posedge clock)
begin
    if (!reset_n)
        state <= IDLE;
    else
        state <= nextstate;
end
//-----
```

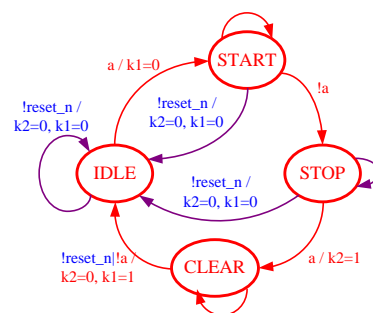
19



### ■ 有限状态机的Verilog HDL描述：二段式（续）

//----- 产生下一状态的组合逻辑

```
always @( state or a )
case (state)
IDLE:
    if ( a ) nextstate = START;
    else nextstate = IDLE;
START:
    if ( !a ) nextstate = STOP;
    else nextstate = START;
STOP:
    if ( a ) nextstate = CLEAR;
    else nextstate = STOP;
CLEAR:
    if ( !a ) nextstate = IDLE;
    else nextstate = CLEAR;
default: nextstate = IDLE;
endcase
```



好处：专心致志

➤ 输入+原态 → 次态

➤ 没reset\_n捣乱

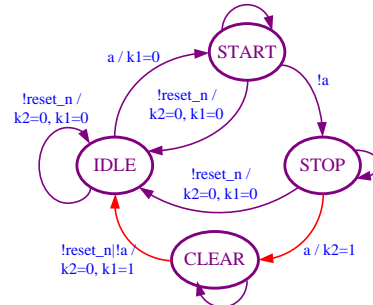
20



## ■ 有限状态机的Verilog HDL描述：二段式（续）

```
//---- 产生输出k1的组合逻辑
always @ ( state or reset_n or a )
  if ( !reset_n ) k1 = 0;
  else
    if ( state == CLEAR && !a )
      // 从CLEAR 转向 IDLE
      k1=1;
    else k1= 0;

//---- 产生输出k2的组合逻辑
always @ ( state or reset_n or a )
  if ( !reset_n ) k2 = 0;
  else
    if ( state == STOP && a )
      // 从STOP 转向 CLEAR
      k2 = 1;
    else k2 = 0;
```



### 开放性讨论：

- 有否疑问？
- 如何解决

另：对于k1的代码可能还有错

21



## 说明

- one-always, two-always, three-always, 但是：

- “所谓的一段式、二段式、三段式写法不能单纯从几个**always**语句来区分，必须清楚它们不同的逻辑划分”

—— 摘自 特权同学

22



## 简单的有限状态机设计

### ■ 有限状态机的Verilog HDL描述

- 三段式——特点：触发器延迟输出

其中：

- 时序逻辑  
——每一个时钟沿产生一次可能的状态变化 部分
- 组合逻辑  
——产生下一状态 部分
- 与“二段式——含有多个组合逻辑语句块”中相应的代码完全相同

不同在于➔（第“三”个语句块）

23



### ■ 有限状态机的Verilog HDL描述：（续）

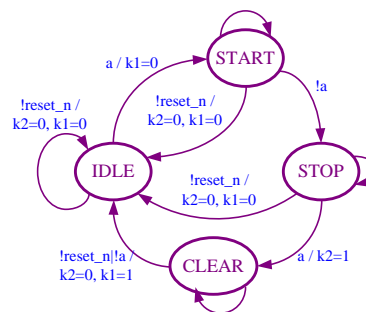
- 三段式——特点：触发器延迟输出

```
always @ ( posedge clock ) begin
    if ( !reset_n && state != CLEAR ) k1 <= 0 ;
    // 从CLEAR到IDLE的弧是例外

    else
        case ( nextstate )
            IDLE : if( !a ) k1 <= 1 ;
                   else k1 <= 0;

            CLEAR : k1 <= 1;
            // 如果理解CLEAR状态k1不变就对
            // 如果理解k1只保持一个周期就错

            default: k1 <= 0 ;
        end
end
// 其实并不太容易理解，根据自身习惯慎用
```



24



## 一些题外话

“特权同学”认为这种“三段式”的优点是消除输出的毛刺

我们认为：并不是说这种“三段式”可以消除组合逻辑的竞争冒险，而是说它相当于把输出又用D-Flip-Flop延迟输出了一个时钟周期

注意到：“第三段”实际是时序逻辑了

- 不过，随着FPGA内部资源的增多，三段式方法的应用也很普及

25



## 状态机设计总结

- 有限状态机设计的一般步骤：

➤ 1.逻辑抽象，得出状态转换图

➤ 😊 状态化简

➤ 3.状态分配

采用Verilog HDL来描述有限状态机，可以充分发挥硬件描述语言的抽象建模能力

在触发器资源丰富的FPGA或ASIC设计中采用**独热编码 (one-hot)**

**可由计算机自动完成，简化了电路设计** 多的优势，也可以采取**输出编码的状态指定**来简化电路结构，并提高状态机的运行速度。

➤ 😊 选定触发器的类型并求出状态方程、驱动方程和输出方程。

➤ 😊 按照方程得出逻辑图

26

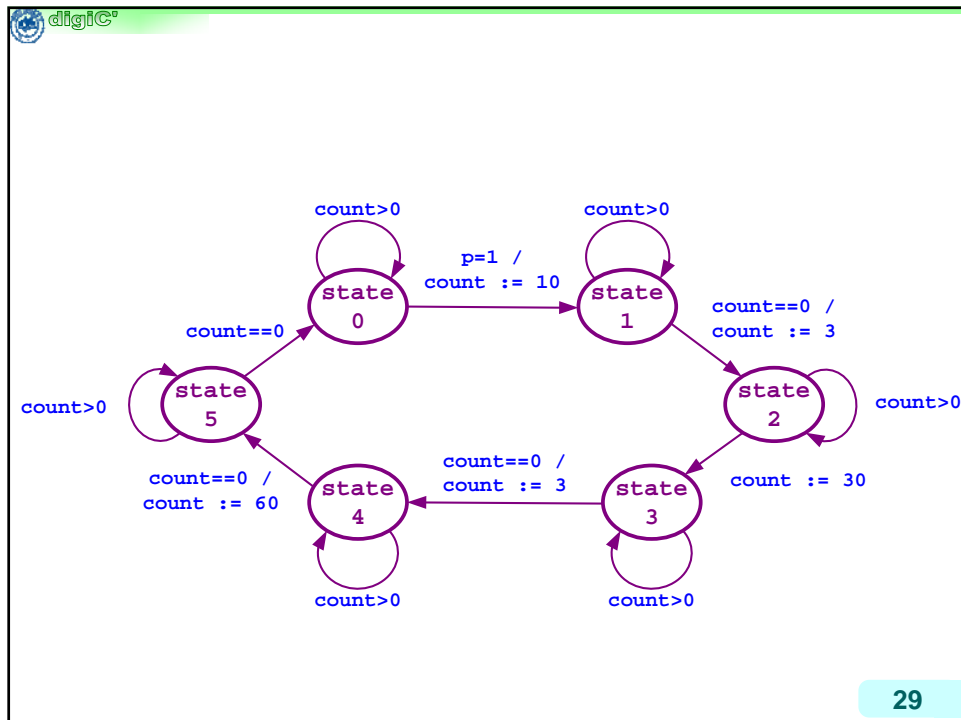


## 交通灯例子

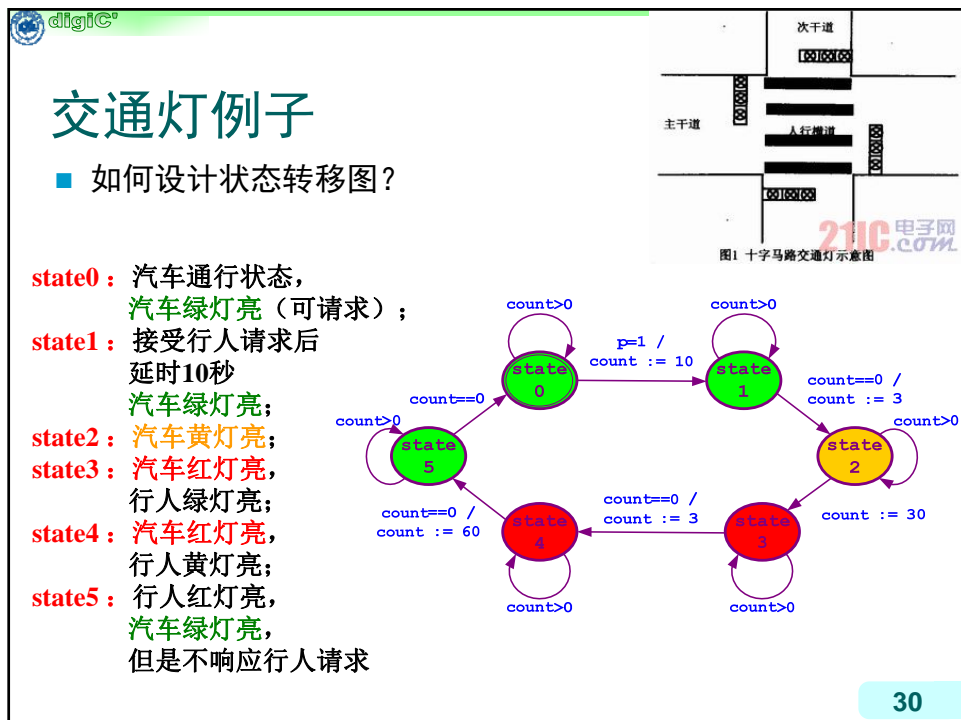
- 正常情况**主干道绿灯**，汽车通行，人行横道红灯，禁止通行；
- 有行人要过马路，进行人行请求，10秒后，**主干道绿灯转黄灯**；
- **主干道黄灯**亮3秒后转**红灯**，同时人行横道红灯转绿灯，行人通行；
- 行人通行30秒后，人行横道绿灯转黄灯；
- 人行横道黄灯亮3秒后转红灯，同时**主干道红灯转绿灯**，并在60秒内不再响应人行请求；
- 60秒后，行人可以继续进行人行请求过马路。

图1 十字路口交通灯示意图

28



29



30



## 交通灯例子

### ■ 如何设计行为模型？

//行人过街交通灯控制器

```
module traffic3(rst,clk,press,lights);
```

```
input      rst;      //异步复位信号输入，高电平复位
```

```
input      clk;      //状态机时钟输入，设1 tick = 1s
```

```
input      press;    //过街按键信号输入，高电平有效
```

```
output [5:0] lights; //交通灯控制输出
```

```
reg [5:0] lights;    //交通灯控制输出
```

//内部信号

```
reg [5:0] count;    //秒计数，最大60秒
```

```
reg [2:0] state;    //状态值
```

```
parameter state0=3'd0, state1=3'd1, state2=3'd2, state3=3'd3,
           state4=3'd4, state5=3'd5;
```

```
parameter lights1=6'b001_100, lights2=6'b010_100,
           lights3=6'b100_001, lights4=6'b100_010;
```



31



### ■ 如何设计行为模型？

```
always @(posedge rst or posedge clk)
```

```
begin
```

```
if(rst)
```

```
begin
```

```
count<=6'd0;
```

```
state<=state0;
```

```
end
```

```
else
```

```
begin
```

```
case(state)
```

```
state0: //汽车通行状态，等待按键
```

```
begin
```

```
if ( press )
```

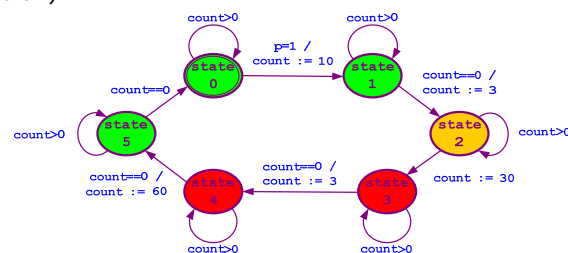
```
begin
```

```
state <= state1; //如果请求键按下，转到状态1
```

```
count <= 6'd10; //同时将count初值给定为10，灯光控制不变
```

```
end
```

```
end
```



32



**交通灯例子**

■ 如何设计行为模型？

```

state1: // 汽车通行状态,
// 但是请求键已经按下, 延时10秒后起作用
begin
  if (count>6'd0)
  begin
    count<=count-1; // 计数器值也是状态, 一般在时序语句块中操纵计数器
  end
else
  begin
    states<=state2; // 计数结束, 转换状态,
                    // 同时为下一个状态计数器赋初值3
    count<=6'd3;
  end
end
end

```

33

**交通灯例子**

■ 如何设计行为模型？

```

state2: // 汽车通道黄灯
begin
  if ( count>6'd0 )
  begin
    count<=count-1;
  end
else // 计数结束, 转移到下一个状态,
    // 同时为计数器赋初值
  begin
    states<=state3;
    count<=6'd27; // 行人绿灯27秒
                  // 黄灯3秒
  end
end
end

```

34

交通灯例子

■ 如何设计行为模型？

```

state3: //汽车等待，行人通行，时间30秒
begin
  if(count>6'd0)
  begin
    count<=count-1;
  end
else
  begin
    states<=state4;
    count<=6'd3;
  end
end
end

```

35

交通灯例子

■ 如何设计行为模型？

```

state4: //行人通行黄灯，时间3秒
begin
  if(count>6'd0)
  begin
    count<=count-1;
  end
else
  begin
    states<=state5;
    count<=6'd60;
  end
end
end

```

36

## 交通灯例子

### ■ 如何设计行为模型?

```

state5: //汽车通行, 不响应按键的状态
begin
  if(count>6'd0)
  begin
    count<=count-1;
  end
else
  begin
    states<=state0;//等待结束回到状态0
    count<=6'd0;
  end
end
default:
begin
  count<=6'd30; //
  states<=state3; //行人绿灯
end
endcase
end
end

```

37

## 交通灯例子

### ■ 如何设计行为模型?

```

//根据当前状态输出灯光控制
always @ (negedge clk)
begin
  case(state)
    state0: lights<=lights1;
    state1: lights<=lights1;
    state2: lights<=lights2;
    state3: lights<=lights3;
    state4: lights<=lights4;
    state5: lights<=lights1;
    default: lights<=lights3;
  endcase
end
endmodule

```

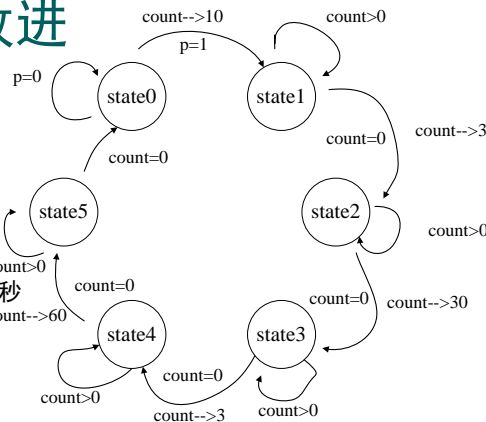
38



## 讨论 交通灯例子改进

- 在行人通行的state3中，考虑到一次通行行人较多的情况，在本状态中，如果有行人继续按压行人请求按钮，则自动增加人行绿灯30s的时间（最多增加一次），请根据改进逻辑进行状态机修改。

```
state3: //汽车等待，行人通行，时间30秒
begin
  if(count>6'd0)
    begin
      count<=count-1;
    end
  else
    begin
      states<=state4;
      count<=6'd3;
    end
  end
end
```



**注意：** 行人通行请求信号为电平信号，有行人按压时，则其信号一直为高水平，在状态机改造中注意行人一直按压的情况。



## 开发的附件（硬件）

- 摆脱了编程器
- 两种USB电缆
  - AM / BM 方头 （下载编程）
  - Micro USB （与实验板串口通信）



## 开发的附件（软件）

- 为了使用USB串口（UART）所必须
  - CP210xVCP
  - USB to UART bridge
- 串口客户端工具
  - 原则上可任意选择好用的串口调试工具
  - 这里提供一款简单的：AccessPort137

41



## 课程信息

- 教师：何锋
  - 新主楼 F-712
  - e-Mail: [fenghe@buaa.edu.cn](mailto:fenghe@buaa.edu.cn)  
（可预约答疑）
- 课程资料
  - 如果有解压密码，约定就是**buaa123456**

