



## EDA讲座3

- 👉 硬件描述语言3
- 👉 EDA作业三

### 如何听？听什么？

- 硬件描述语言 ——语言与电路结构、规定和规范
- EDA作业三——功能模块的划分、外设工作原理

## 目 录

### 一、组合电路（以2选1为例）

1. 电路模块的基本结构
2. 电路模块的描述方式

### 二、测试平台

### 三、时序电路

1. D触发器
2. 状态机

## 同步复位的D触发器

```
module dff_syn (clk,d,q,rst);
    input clk,d,rst;
    output q;
    reg q;

    always @ (posedge clk)
    begin
        if (!rst) q<=0;
        else q<=d;
    end
endmodule
```

### 1. 对时钟上升沿敏感

敏感信号posedge clk启动过程语句

☞ 边沿敏感信号clk必须出现在敏感表中  
always过程结构中clk不再出现

### 2. 在上升沿，有效复位信号清零

☞ 信号rst为时钟的同步控制信号，不出现在敏感表中。

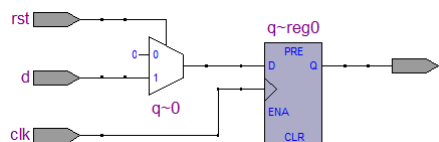
## 同步复位的D触发器

```

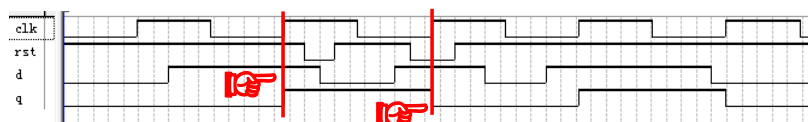
module dff_syn (clk,d,q,rst);
  input clk,d,rst;
  output q;
  reg q;

  always @(posedge clk)
  begin
    if (!rst) q<=0;
    else q<=d;
  end
endmodule

```



启动过程语句



在上升沿复位信号清零

## 异步复位的D触发器

```

module dff_asyn (clk,d,rst,en,q);
  input clk,d,rst,en;
  output q;
  reg q;

  always @(posedge clk or negedge rst)
  begin
    if (!rst) q<=0;
    else if (en) q<=d;
  end
endmodule

```

对时钟上升沿和rst下降沿敏感，敏感信号启动过程语句

☞ 信号rst为时钟的异步控制信号：

1. 必须出现在敏感表中；
2. 在always过程结构中必须表述其逻辑行为且两种表述必须相匹配。

☞ 敏感表中出现边沿敏感的表述，则其他类型的敏感信号不能放置。

如：always@ (posedge clk or rst) ❌

## 异步复位的D触发器

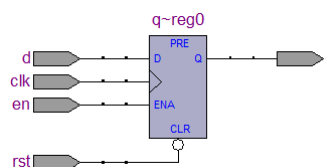
```

module dff_asyn (clk,d,rst,en,q);
input clk,d,rst,en;
output q;
reg q;

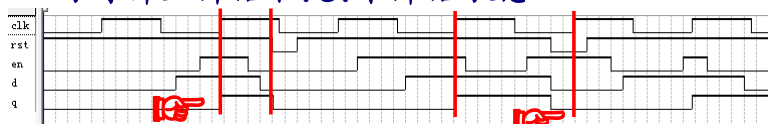
always @(posedge clk or negedge rst)
begin
    if (!rst) q<=0;
    else if (en) q<=d;
end

endmodule

```

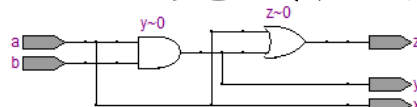


对时钟上升沿和rst下降沿敏感



if-elseif... , 判断的先后次序上隐含优先级关系

## 过程中的两类赋值语句：



### ■ 阻塞式赋值

```
always @(a,b) begin
```

```
    x=a;
```

```
    y=b&x;
```

```
    z=x|y;
```

```
end
```

a	b	x	y	z
0	0	0	0	0
1	1	1	1	1

### ■ 非阻塞式赋值

```
always @(a,b) begin
```

```
    x<=a;
```

```
    y<=b&x;
```

```
    z<=x|y;
```

```
end
```

a	b	x	y	z
0	0	0	0	0
1	1	1	0	0

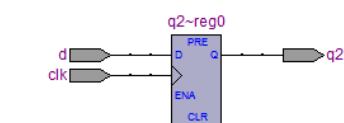


赋值方式不同，综合电路相同，某一时刻仿真不同

## 过程中的两类赋值语句：

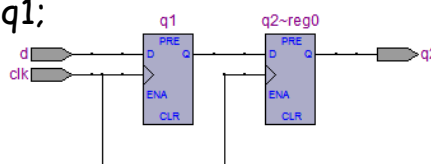
### ■ 阻塞式赋值

```
always @(posedge clk)
begin
  q1=d;
  q2=q1;
end
```



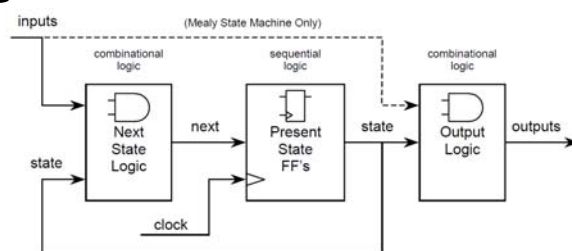
### ■ 非阻塞式赋值

```
always @(posedge clk)
begin
  q1<=d;
  q2<=q1;
end
```



赋值方式不同，综合电路不同

## 状态机



- 按状态机的描述结构有单进程和多进程（双进程或三进程）。

为使代码结构清晰，建议使用多进程的方式描述。

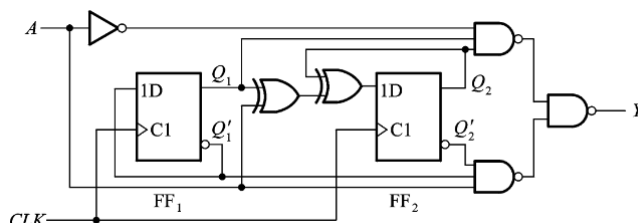
- 可以使用QuartusII编辑器中的状态机模板。

新建Verilog文件界面：

Edit\Insert Template\verilogHDL\Full Designs\State Machines

## 状态机举例

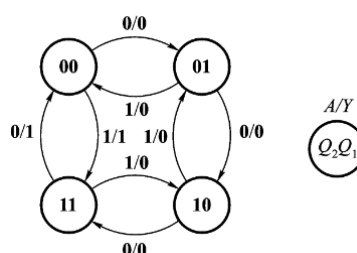
### ■ 例6.2.3



$$\begin{cases} D_1 = Q_1' \\ D_2 = A \oplus Q_1 \oplus Q_2 \end{cases}$$

$$\begin{cases} Q_1^* = D_1 = Q_1' \\ Q_2^* = D_2 = A \oplus Q_1 \oplus Q_2 \end{cases}$$

$$Y = A'Q_1Q_2 + AQ_1'Q_2'$$



## 三进程描述:

```

module fsm3
(
    input clk, A, reset,
    output reg out);
    reg [1:0] current_state, next_state;

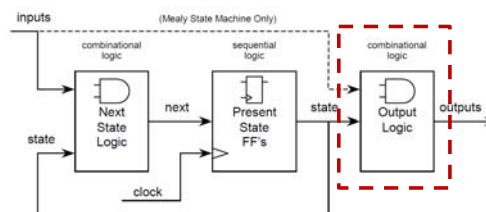
    parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;

    1: always @ (current_state or A) begin
        case (current_state)
            S0:
                if (A) out = 1;
                else out = 0;
            S1:
                if (A) out = 0;
                else out = 0;
            S2:
                if (A) out = 0;
                else out = 0;
            S3:
                if (A) out = 0;
                else out = 1;
        endcase
    end

```

说明部分: 由参数说明关键字 **parameter** 定义各个状态

1. 组合电路部分: 输出方程



**3:**

```

always @ (current_state or A)
begin
  case (current_state)
    S0:
      if (A)
        next_state <= S3;
      else
        next_state <= S1;
    S1:
      if (A)
        next_state <= S0;
      else
        next_state <= S2;
    S2:
      if (A)
        next_state <= S1;
      else
        next_state <= S3;
    S3:
      if (A)
        next_state <= S2;
      else
        next_state <= S0;
  endcase
end
endmodule

```

**2:**

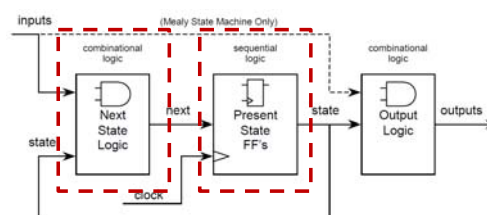
```

always @ (posedge clk or posedge reset) begin
  if (reset)
    current_state <= S0;
  else
    current_state <= next_state;
end

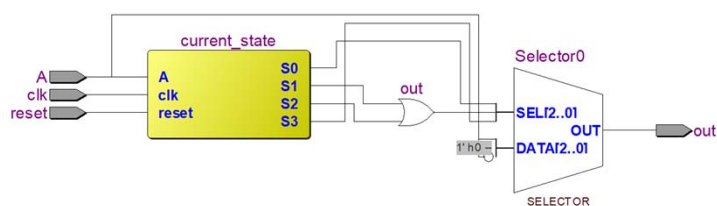
```

☞ **2.时序电路部分：状态方程**

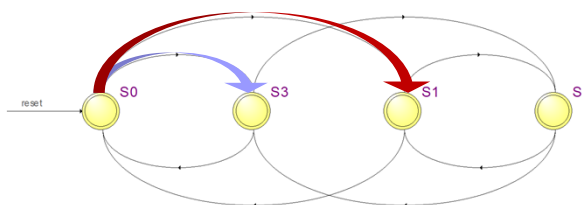
☞ **3.组合电路部分：驱动方程**



Tools → Netlist Viewers → RTL Viewer



Tools → Netlist Viewers → State Machine Viewer 或双击current\_state模块



	Source State	Destination State	Condition
1	S0	S3	(A)
2	S0	S1	(!A)
3	S1	S0	(A)
4	S1	S2	(!A)
5	S2	S3	(!A)
6	S2	S1	(A)
7	S3	S0	(!A)
8	S3	S2	(A)



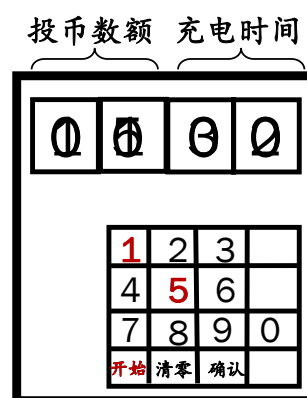
# EDA作业三

——功能模块的划分、外设工作原理

## 投币式手机充电仪

可以实现投币、实时显示投币数额和充电时间等功能。

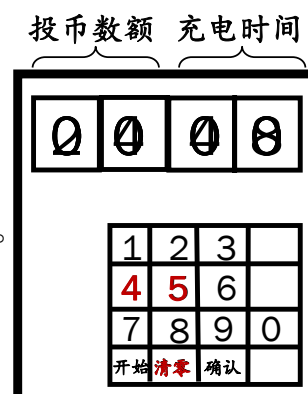
- 刚上电即“初始状态”，数码管显示全灭。
- 按“开始”键后进入准备投币状态，数码管显示“0000”。
- 矩阵键盘直接输入投币数额1~20角
- 2倍于投币数额的允许充电时间也实时显示





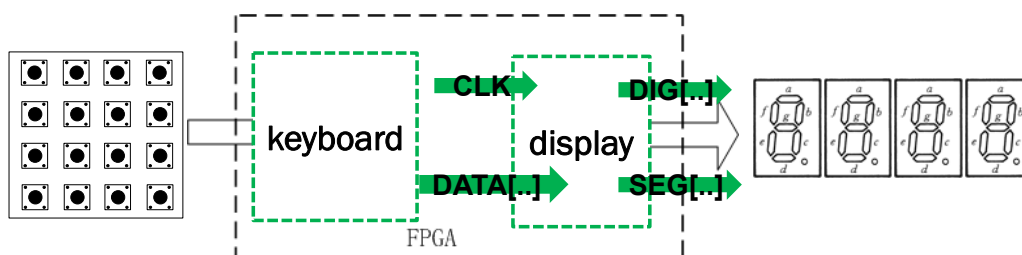
## 投币式手机充电仪

- ☞ 最大投币20角；即最大时间显示40
- ☞ 未确认充电之前可随时清零。清零回至“开始状态”，10秒无动作回到“初始状态”。
- ☞ 确认充电后，充电时间开始倒计时，  
投币数额仍保持显示；  
当时间计至0时，投币数额**同时**归0。  
回到“开始状态”，10秒后回到“初始状态”。



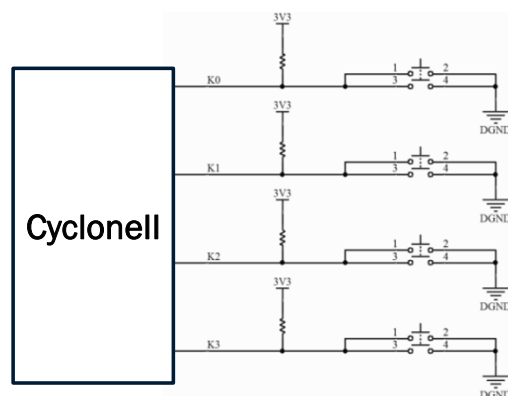
## 任务分解

- 利用模块化设计方法，“自顶向下”划分功能模块
  - ☞ 键盘输入电路、数码管显示电路、分频器...



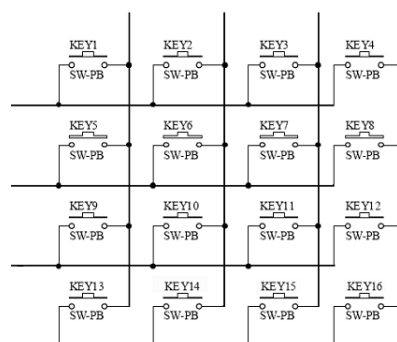
## 1. 独立按键

- ☞ 原理图：单个按键，接到FPGA引脚
- ☞ 特点：工作原理简单；按键较多时，占用I/O资源



## 2. 矩阵键盘

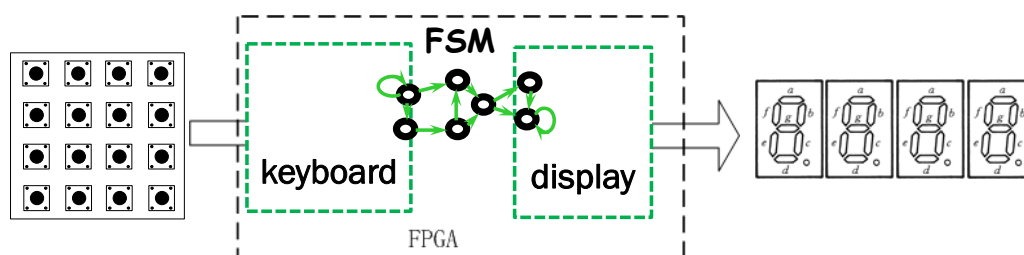
- 以行列方式排列，按键位于行列线的交叉点上
- 按键较多时，节省I/O资源
- 工作原理比独立式复杂





## 任务分解

- 利用模块化设计方法，“自顶向下”划分功能模块
- 控制电路（状态机电路）——协调各模块之间的操作  
遵循时序电路的设计方法：逻辑抽象、画出状态转换图...



## 选做：

充电结束播放一段音乐

- 工作原理
  - ☞ 每个音符的发音频率（音高）
  - ☞ 每个音符的持续时间（速度、拍子）

- 网络学堂资源
  - ☞ EDA视频：Verilog描述状态机
  - ☞ 验收要求
- 其他资源：答疑？开放时间？
- 第13周讲解验收要求、验收键盘电路的仿真与下载  
键盘仿真：防抖、长按键
- 第14周验收状态机电路仿真、状态图和整体功能  
状态机仿真：模拟正常一次流程