# 控制器1

**刘 鹏**

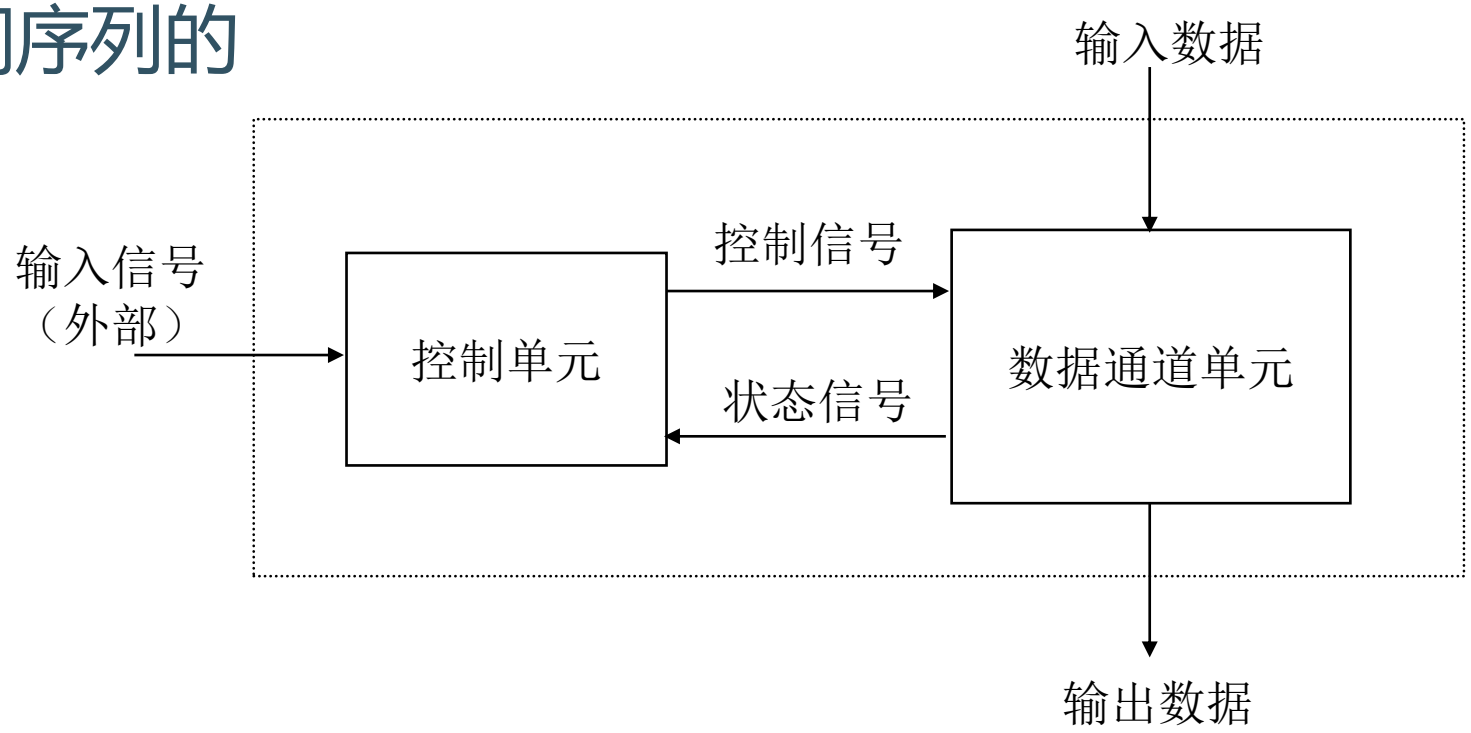
浙江大学信息与电子工程学院

liupeng@zju.edu.cn

Source: 补充讲义

# 数字系统中控制单元和数据通道单元的关系

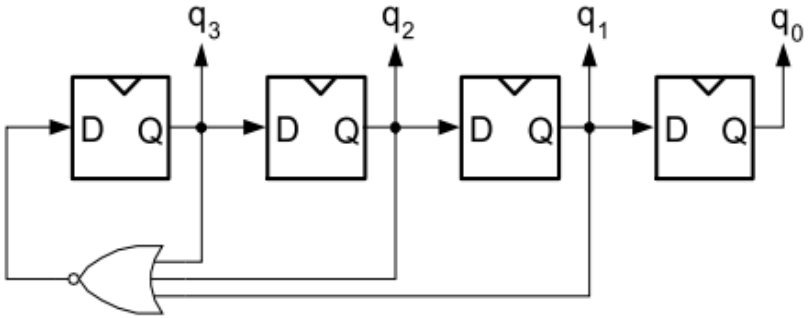
- 数据通道单元的所有微操作都是由控制单元启动，产生微操作控制序列信号的控制单元是一个时序电路，它的各种状态(指内部状态)表示系统的各个控制功能
- 控制单元是为启动数字系统中处理器的微操作提供控制信号时间序列的



# 控制器设计方法

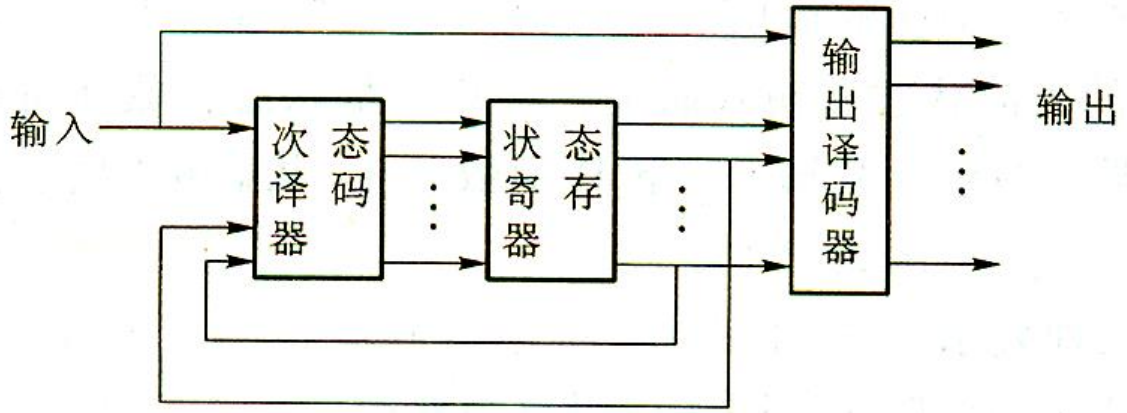
- 控制器是一个时序电路，完全可以按时序电路的设计方法进行设计

- 按照寄存器传送方法，可有4种方法
  - **每个状态一个触发器（one-hot，热位）如右图可自动的one-hot 计数器**
  - **序列寄存器-译码器法**
  - **可编程逻辑阵列PLA控制法（选择器法）**
  - **微程序控制法**



# 状态机

- 状态机是指按有序方式遍历预先确定的状态序列的数字逻辑功能电路
- 状态机是组合逻辑和寄存器的特殊组合，它包括两个主要部分：即**组合逻辑部分和寄存器部分**
- 寄存器用于存储状态机内部状态；组合逻辑部分又可分为状态译码器和输出译码器，状态译码器确定状态机的下一个状态，即确定状态机的激励方程，输出译码器确定状态机的输出，即确定状态机的输出方程

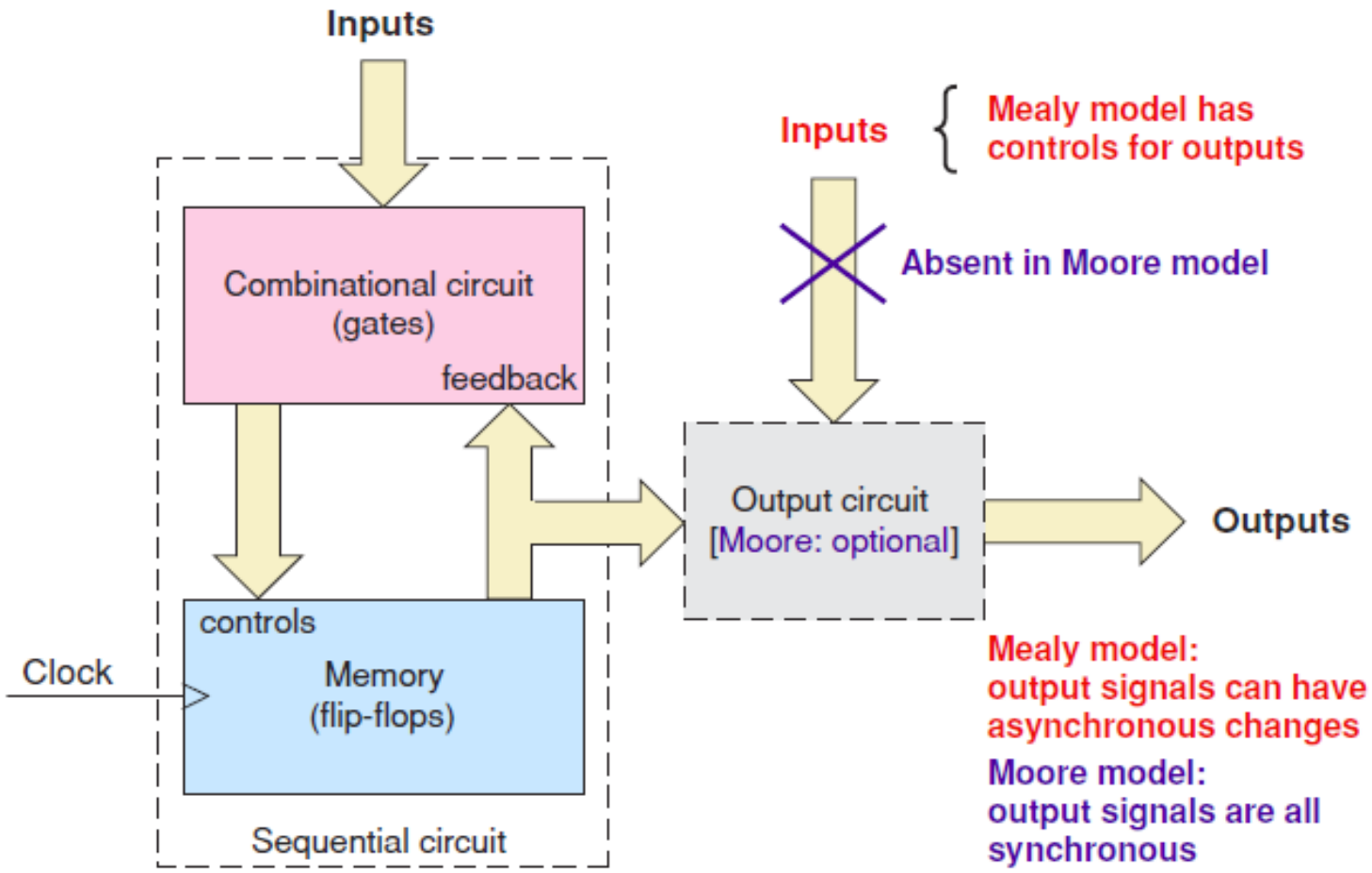


# 状态机主要完成两种基本操作

- 状态机内部状态转换。遍历某一确定的状态序列，其中次态由次态译码器根据现态和输入条件来确定
- 根据状态变化(称为状态转移)产生输出信号。输出译码器根据现态和输入条件可确定输出信号
- 状态机有三种表示方法：
  - **状态图**
  - **状态表**
  - **流程图**

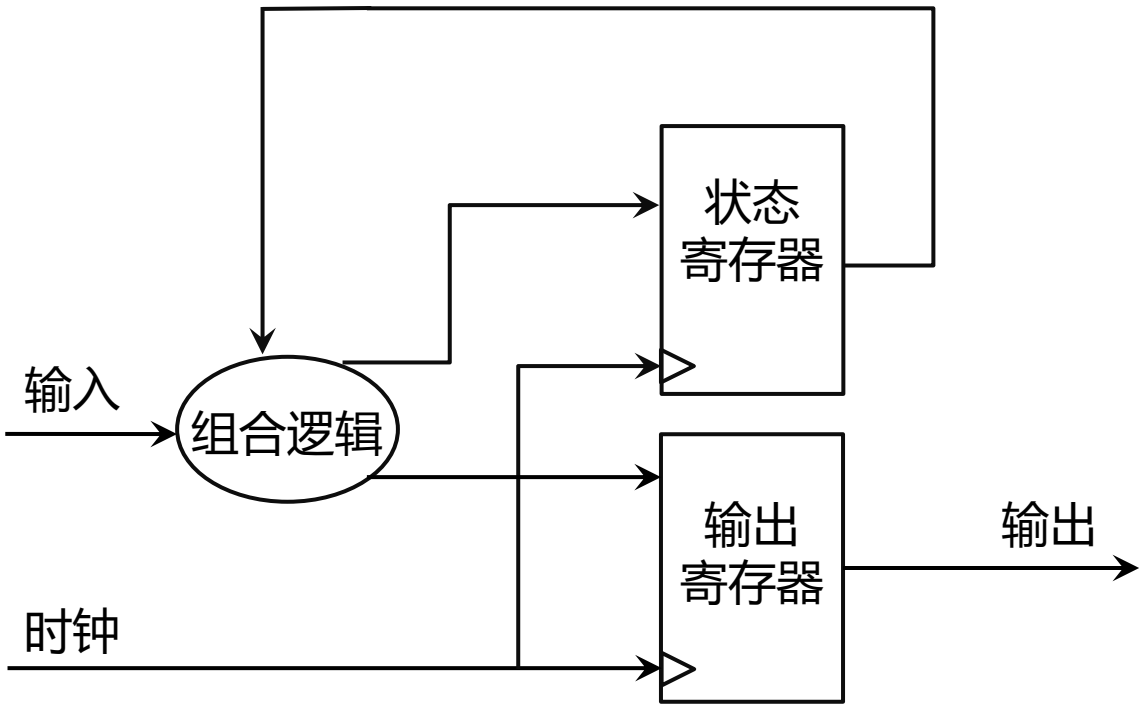
# 状态机的两种基本类型

- Mealy状态机和Moore状态机



# 输出加寄存器消除毛刺

- 对输出加寄存器是一个行之有效的方法
- 通过在时钟边沿取样输出信号，可以极大地消除毛刺带来的影响



# 状态单元

Always blocks are the only way to specify the "behavior" of state elements. Synthesis tools will turn state element behaviors into state element instances.

D-flip-flop with synchronous set and reset example:

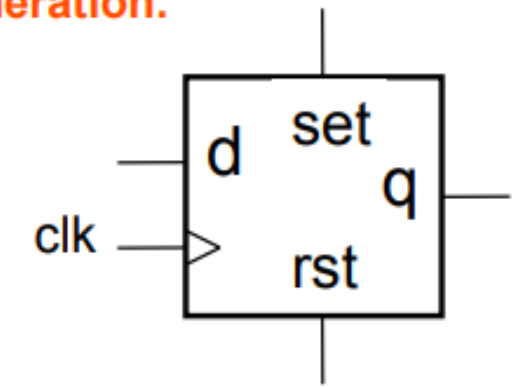
```
module dff(q, d, clk, set, rst);
  input d, clk, set, rst;
  output q;
  reg q;

  always @(posedge clk)
    if (rst)
      q <= 1'b0;
    else if (set)
      q <= 1'b1;
    else
      q <= d;
endmodule
```

keyword

"always @ (posedge clk)" is key to flip-flop generation.

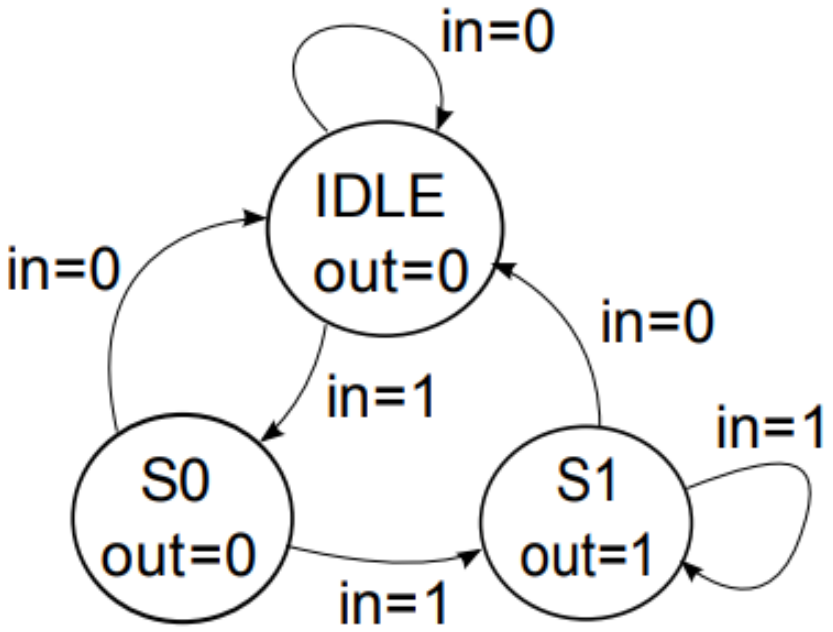
This gives priority to reset over set and set over d.



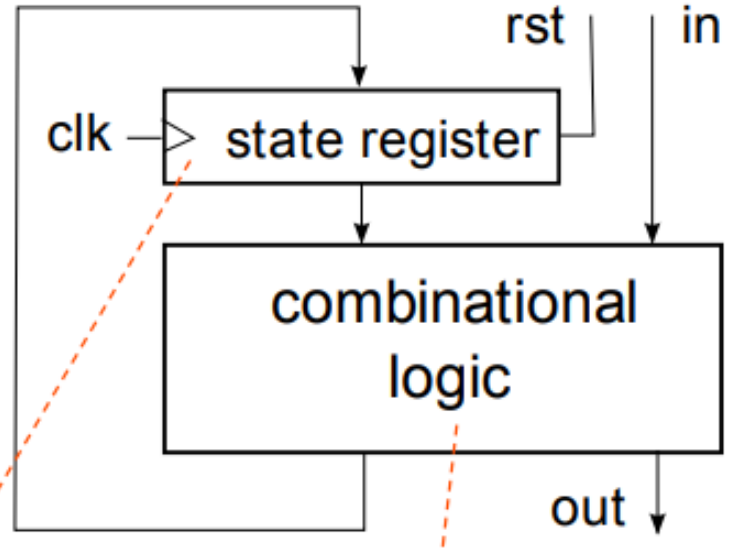

On FPGAs, maps to native flip-flop.

# 有限状态机 Finite State Machines

State Transition Diagram




Implementation Circuit Diagram




Holds a symbol to keep track of which bubble the FSM is in.

CL functions to determine output value and next state based on input and current state.

out = f(in, current state)

next state = f(in, current state)

What does this one do?

Did you know that every SDS is a FSM?

# Finite State Machines

```
module FSM1(clk, rst, in, out);
input clk, rst;
input in;
output out;

// Defined state encoding:
parameter IDLE = 2'b00;
parameter S0 = 2'b01;
parameter S1 = 2'b10;
reg out;
reg [1:0] state, next_state;

// always block for state register
always @(posedge clk)
      if (rst) state <= IDLE;
      else state <= next_state;
```

Must use reset to force to initial state.

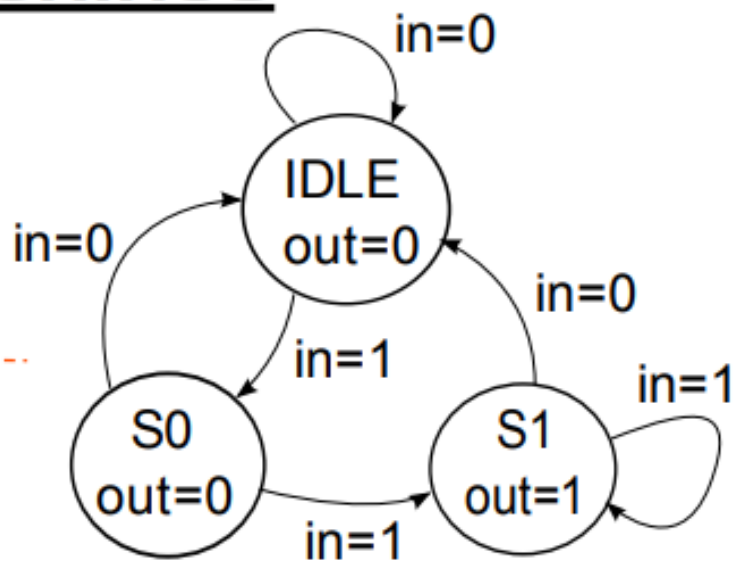
reset not always shown in STD

Constants local to this module.

out not a register, but assigned in always block

Combinational logic signals for transition.

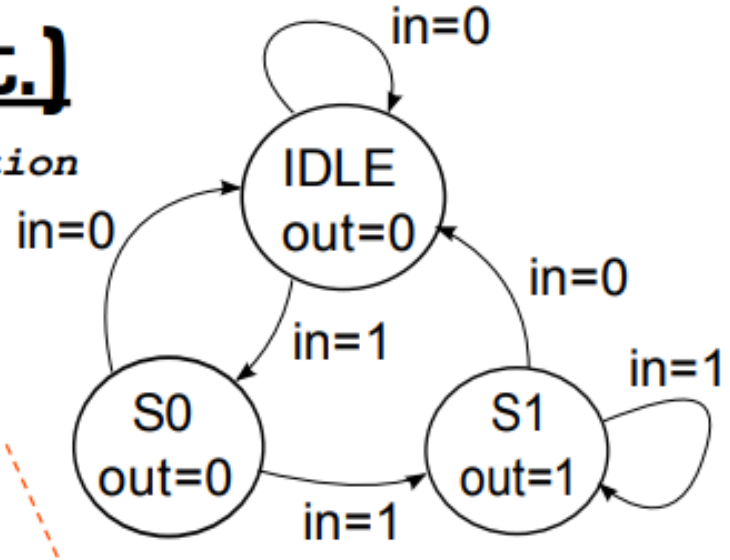
THE register to hold the "state" of the FSM.




A separate always block should be used for combination logic part of FSM. Next state and output generation. (Always blocks in a design work in parallel.)

# FSMs (cont.)

```
// always block for combinational logic portion
always @(state or in)
case (state)
// For each state def output and next
  IDLE    : begin
              out = 1'b0;
              if (in == 1'b1) next_state = S0;
              else next_state = IDLE;
            end
  S0      : begin
              out = 1'b0;
              if (in == 1'b1) next_state = S1;
              else next_state = IDLE;
            end
  S1      : begin
              out = 1'b1;
              if (in == 1'b1) next_state = S1;
              else next_state = IDLE;
            end
  default: begin
              next_state = IDLE;
              out = 1'b0;
            end
endcase
```




Each state becomes a case clause.

For each state define:

Output value(s)

State transition

Use “default” to cover unassigned state.
Usually unconditionally transition to reset state.

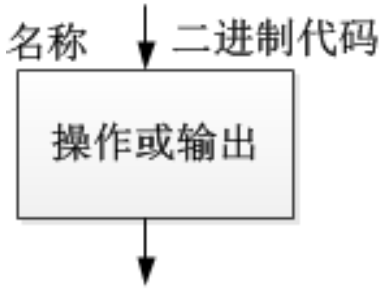
# 算法流程图

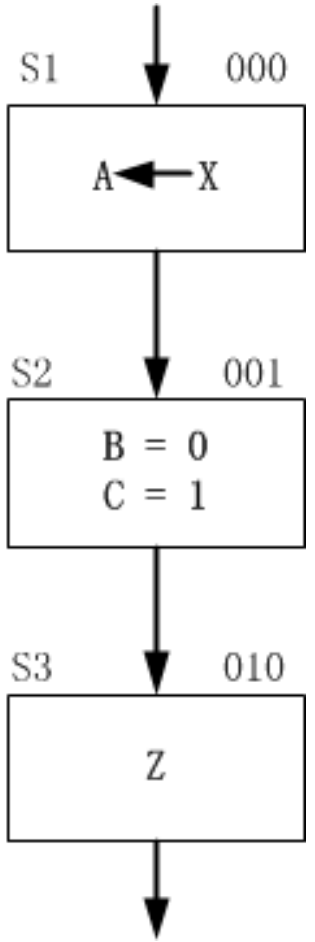
- Algorithmic State Machine Chart，**ASM**
- ASM图用来描述控制器不同时间内应完成的一系列操作，指出控制器状态转换、转换条件，以及控制器的输出
- ASM图又称为**算法状态机图**，它用符号来表示系统的时序操作，**类似于流程图的形式，但又不同于流程图**
- ASM图中**不仅反映了工作顺序，而且还表明了控制器的状态转换顺序**

# ASM图中采用的符号和规则

- ASM图：**状态框**

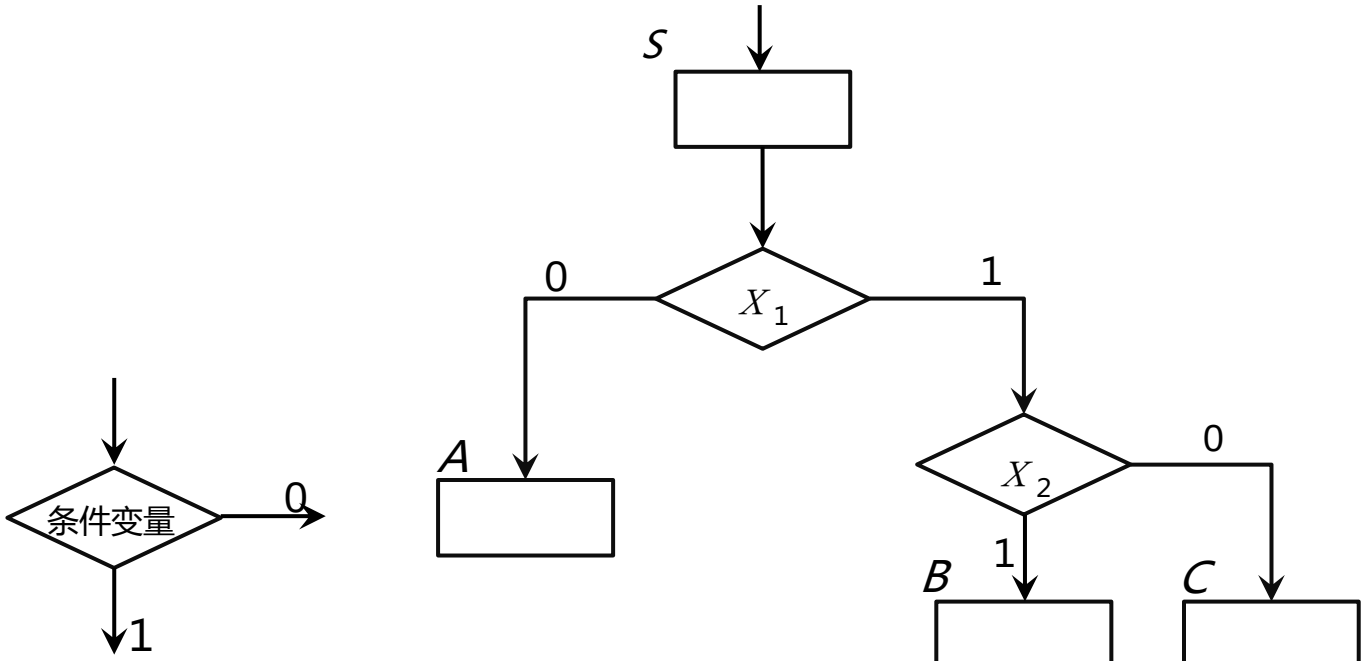
- 数字系统控制序列的状态用状态框表示，状态框的形状是一个矩形，框内标出在此状态下实现的寄存器传输操作或输出，状态的名称置于状态框的左上角，分配给状态的二进制代码置于状态框的右上角



# 判断框

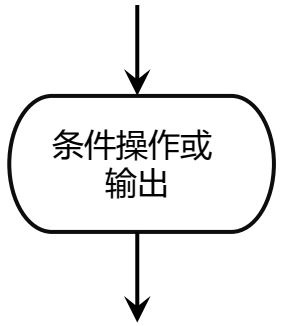
- **菱形框**内填写条件变量的判断条件，经判断框后状态转移出现两个或多个分支，如图 (a)所示。若条件是真，选定一个分支，若条件是假，选定另一个分支。图(b)是由两个判断框构成ASM图的实例



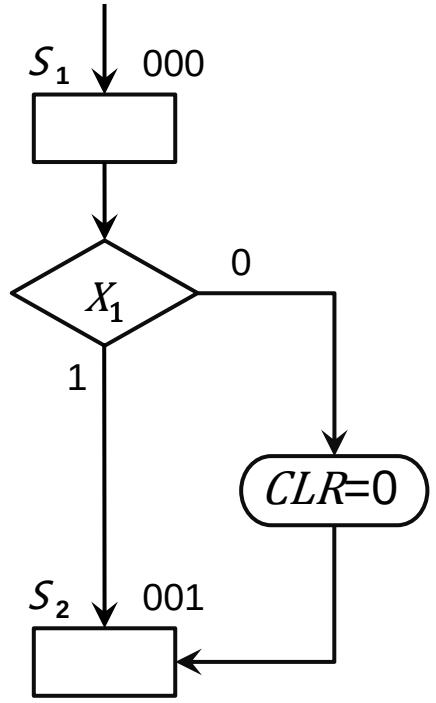

（a）判断框符号　　（b）判断框构成的ASM图实例

# 条件框

- 条件框的形状为椭圆形如图(a)所示，框内填写数据子系统进行的条件操作，框外填写必需的条件输出，条件框的输入通道必定来自判断框的分支，即条件框的操作或输出必须是在同时满足状态与条件的情况下才进行
- 如图(b)所示。当系统处于状态S1时，如果条件X1=0，那么CLR被清“0”，否则CLR保持不变，同时不论X1为何值，系统的下一状态都是S2




(a)条件框符号



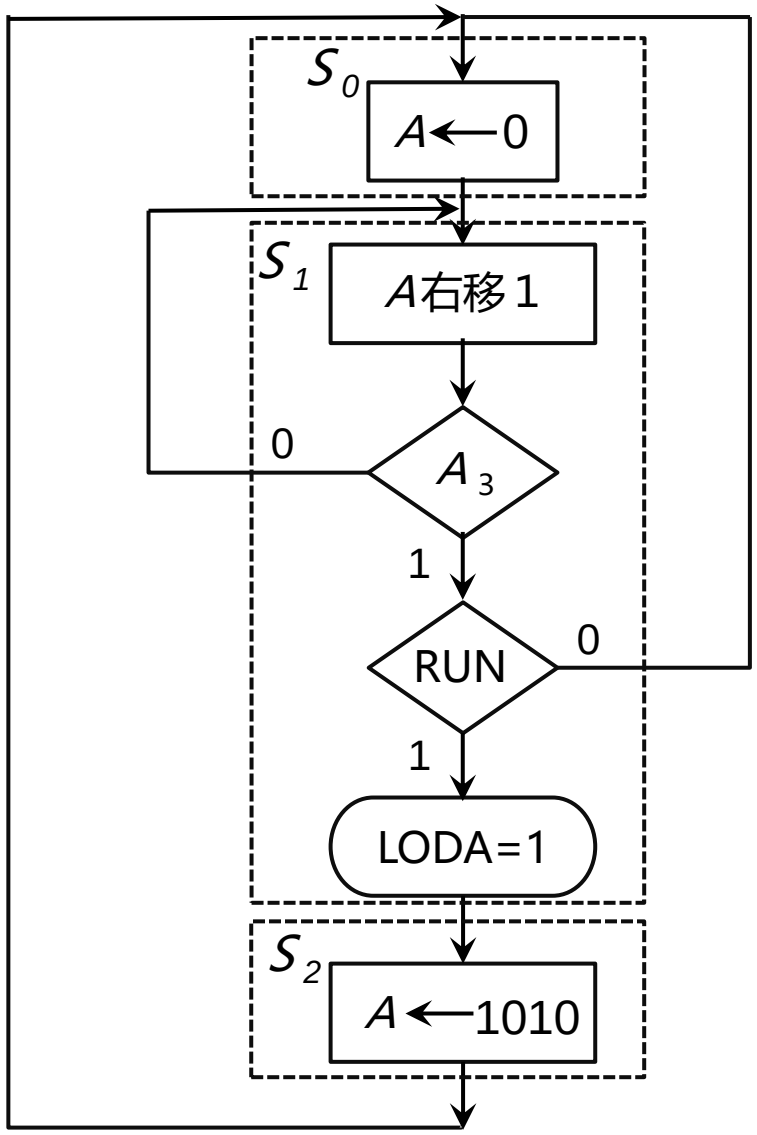

(b)条件框构成的ASM图实例

# ASM图的时间划分

- **ASM**块描述了一个时钟周期内系统的工作情况，它包括**数据子系统**和**控制器**两个方面，即在当前状态及条件下，数据子系统所完成的各种操作以及控制器转换的后续状态

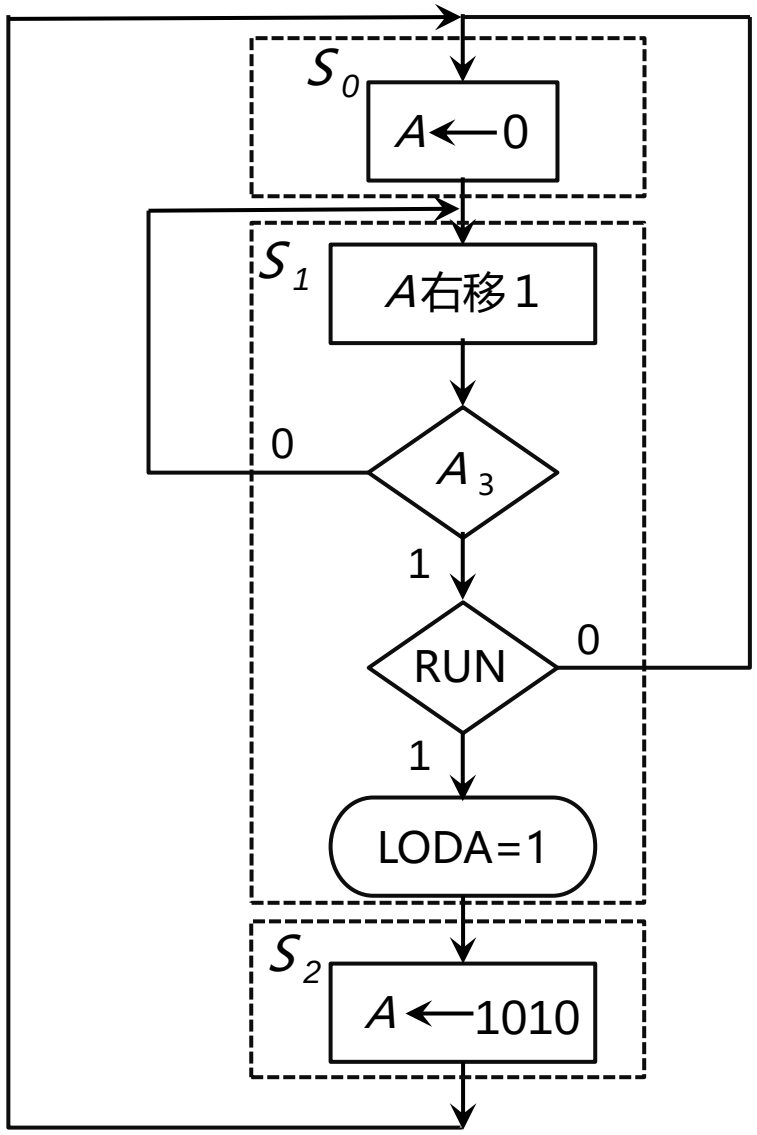
- 所有的操作和状态转换都发生在时钟的同一个跳变边沿

- **ASM**图是按时钟的节拍描述整个数字系统的操作。系统的主时钟不仅作用到数据子系统的寄存器上，而且也作用到控制器的触发器上



# 4位移位寄存器的ASM

- 图中A是一个四位移位寄存器，同步清零和移位置数
- 其中**A3**为**A**的最高位，**RUN**为外部输入的异步变量，**LODA**为移位置数变量，为条件输出，即 **LODA=S1·A3·RUN**
- 注意**A0A1A2A3**(顺序)



# 4位移位寄存器状态转换表

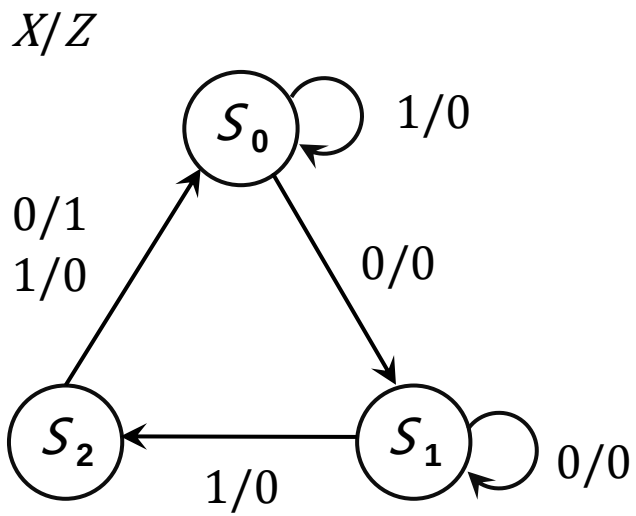
| 状态机现态 | 条件变量 | | 移位寄存器内容 | | | | 状态机次态 |
|---|---|---|---|---|---|---|---|
| | $A_3$ | RUN | $A_0$ | $A_1$ | $A_2$ | $A_3$ | |
| $S_0$ | Ø | Ø | 0 | 0 | 0 | 0 | $S_1$ |
| $S_1$ | 0 | Ø | 1 | 0 | 0 | 0 | $S_1$ |
| $S_1$ | 0 | Ø | 1 | 1 | 0 | 0 | $S_1$ |
| $S_1$ | 0 | Ø | 1 | 1 | 1 | 0 | $S_1$ |
| $S_1$ | 0 | Ø | 1 | 1 | 1 | 1 | $S_1$ |
| $S_1$ | 1 | 0 | 1 | 1 | 1 | 1 | $S_0$ |
| $S_0$ | Ø | Ø | 0 | 0 | 0 | 0 | $S_1$ |
| $S_1$ | 1 | 1 | 1 | 1 | 1 | 1 | $S_2$ |
| $S_2$ | Ø | Ø | 1 | 0 | 1 | 0 | $S_0$ |

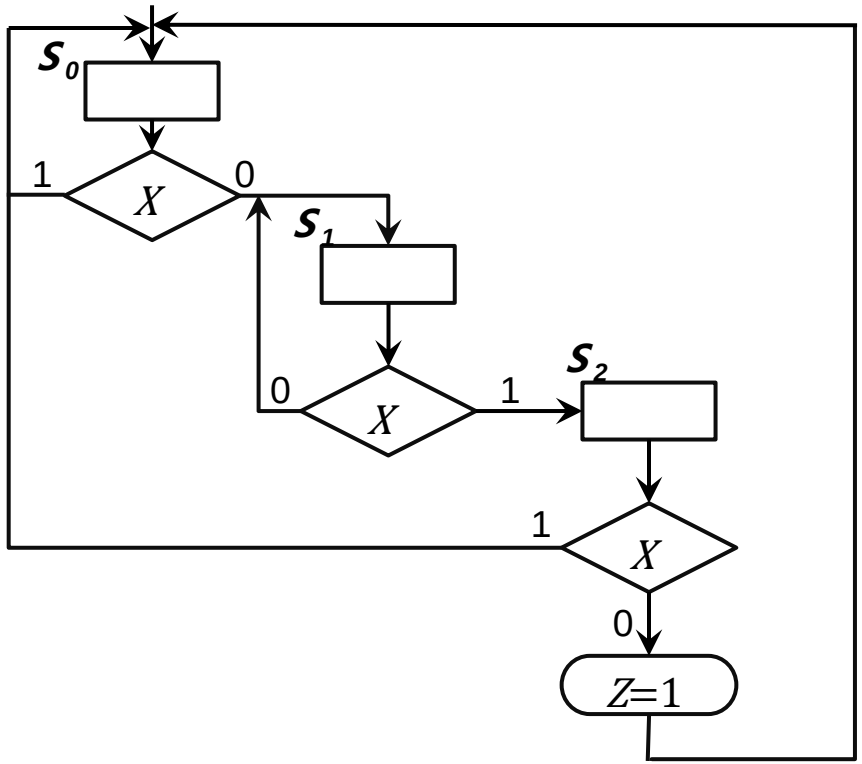
# ASM图的建立原则

- 流程图中的工作块基本上对应了ASM图中的状态框
- 如果工作块的操作不能在一个CLK内同时进行，在ASM图中就必须将其分为几个状态框，在这几个状态之间实现无条件转移
- 流程图中的判断块基本上对应了ASM图中的判断框
  - 如果判断条件是上个操作的结果，那么在ASM图中应在此判断框前增加一个状态框
  - 如果不增加一个状态框，则判断条件对应于前一个CLK的工作块的操作结果
- 在ASM图的最上层加一个起始状态

# 例1

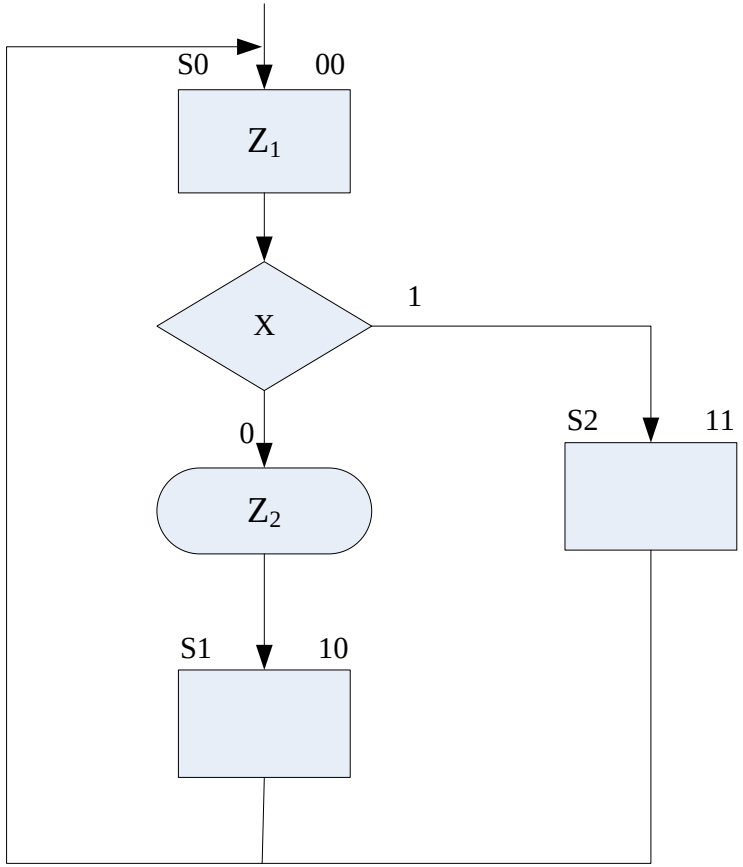
- 串行数据序列是每个时钟周期传送一个数据0或1的数据流。设X为输入的串行数据序列。当检测到数据流中出现所需的010数据时，使检测器的输出Z为1。试画出其ASM图



# 例2

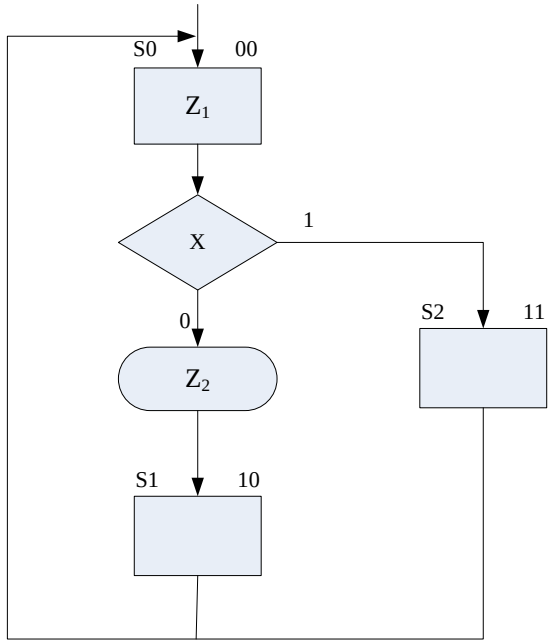
- 某计数器型控制器的ASM图,请根据图中的状态分配设计出对应的电路




- 系统有一个外输入X，两个输出命令Z2和Z1，三个状态S2、S1和S0，即需要两个触发器来设置两个状态变量Q2Q1
- 触发器可采用JK型或D型触发器，此处采用D型触发器

# 状态转换表

| 现态 | | | 次态 | | 转换条件 | |
|---|---|---|---|---|---|---|
| $Q_2$ | $Q_1$ | X | $Q_2^{n+1}$ | $Q_1^{n+1}$ | $Z_2$ | $Z_1$ |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | Ø | 0 | 0 | 0 | 0 |
| 1 | 0 | Ø | 0 | 0 | 0 | 0 |
| 1 | 1 | Ø | 0 | 0 | 0 | 0 |




该表为简化状态转换表，因为10和11状态与输入X无关，所以对应于该两行X值可作为任意项Ø处理

触发器的驱动方程

$$Q_2^{n+1} = \bar{Q}_2\bar{Q}_1$$

$$Q_1^{n+1} = \bar{Q}_2\bar{Q}_1 X$$

输出方程

$$Z_2 = \bar{Q}_2\bar{Q}_1\bar{X}$$

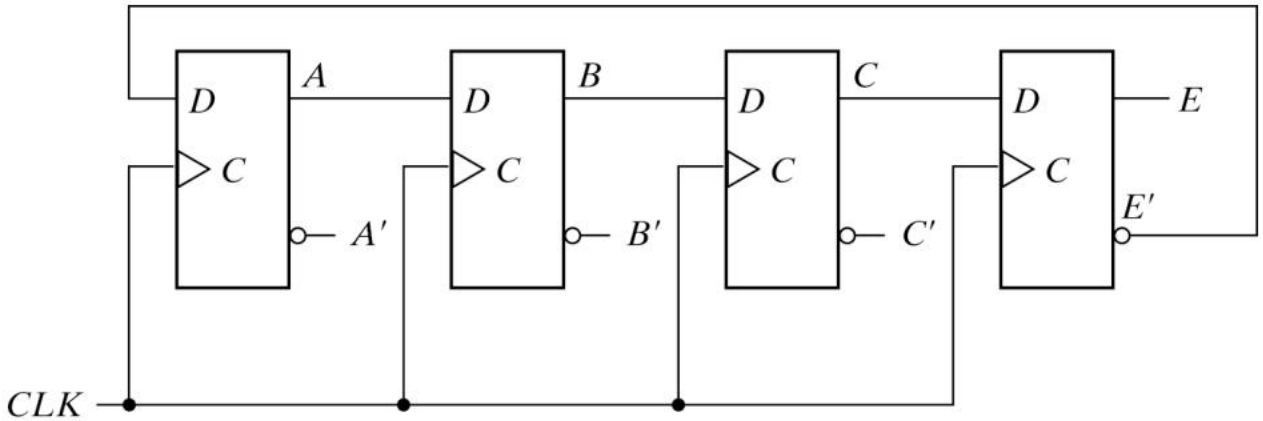
$$Z_1 = \bar{Q}_2\bar{Q}_1$$

设01的次态为00，以保证一旦出现01状态后（电路自启动），经过一个时钟周期可以自动回到有用状态循环

# 状态机设计

- 实际应用问题进行分析和归纳，以确定控制系统的任务及要实现的功能
- 列出采用的**状态机全部可能的状态**，并对每一个状态进行状态编码及定义相应的状态转换条件
- 根据状态图(或把状态图转化为状态表，并对状态图和状态表进行必要的简化处理)和输出函数，画出**状态转移图**
- 建立激励函数和输出函数，画出逻辑电路

# 约翰逊环形计数器 Johnson Ring Counter




(a) Four-stage switch-tail ring counter

| Sequence number | Flip-flop outputs A | B | C | E | AND gate required for output |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | $A'E'$ |
| 2 | 1 | 0 | 0 | 0 | $AB'$ |
| 3 | 1 | 1 | 0 | 0 | $BC'$ |
| 4 | 1 | 1 | 1 | 0 | $CE'$ |
| 5 | 1 | 1 | 1 | 1 | $AE$ |
| 6 | 0 | 1 | 1 | 1 | $A'B$ |
| 7 | 0 | 0 | 1 | 1 | $B'C$ |
| 8 | 0 | 0 | 0 | 1 | $C'E$ |

(b) Count sequence and required decoding

**In fact, for *any size* Johnson counter, the decoding gates will have only two inputs.**

Johnson counters represent a middle ground between ring counters and binary counters. A Johnson counter requires fewer FFs than a ring counter but generally more than a binary counter; it has more decoding circuitry than a ring counter but less than a binary counter.