



第七章

数字系统设计基础



7.1 概述

- 传统的真值表、卡诺图、状态转移图等方法设计电路需要凭设计者的经验，而且不适合大规模的数字系统设计。
- 需要一种数字系统的设计方法，突破传统方法的局限性。

7.1.1 数字系统的基本模型

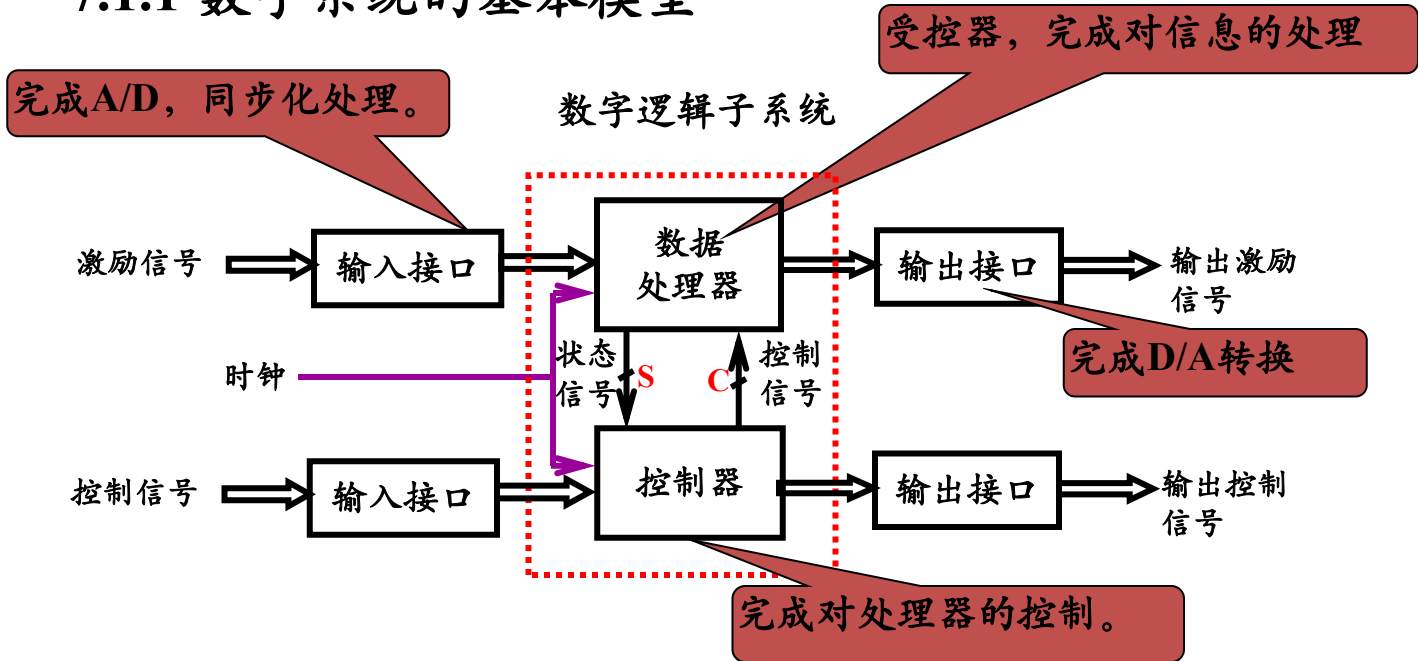


图7.1.1 数字系统的一般模型

注意：**控制器**是区别数字系统和简单的功能部件的标志



1) 数据处理器

(1) 数据处理器结构

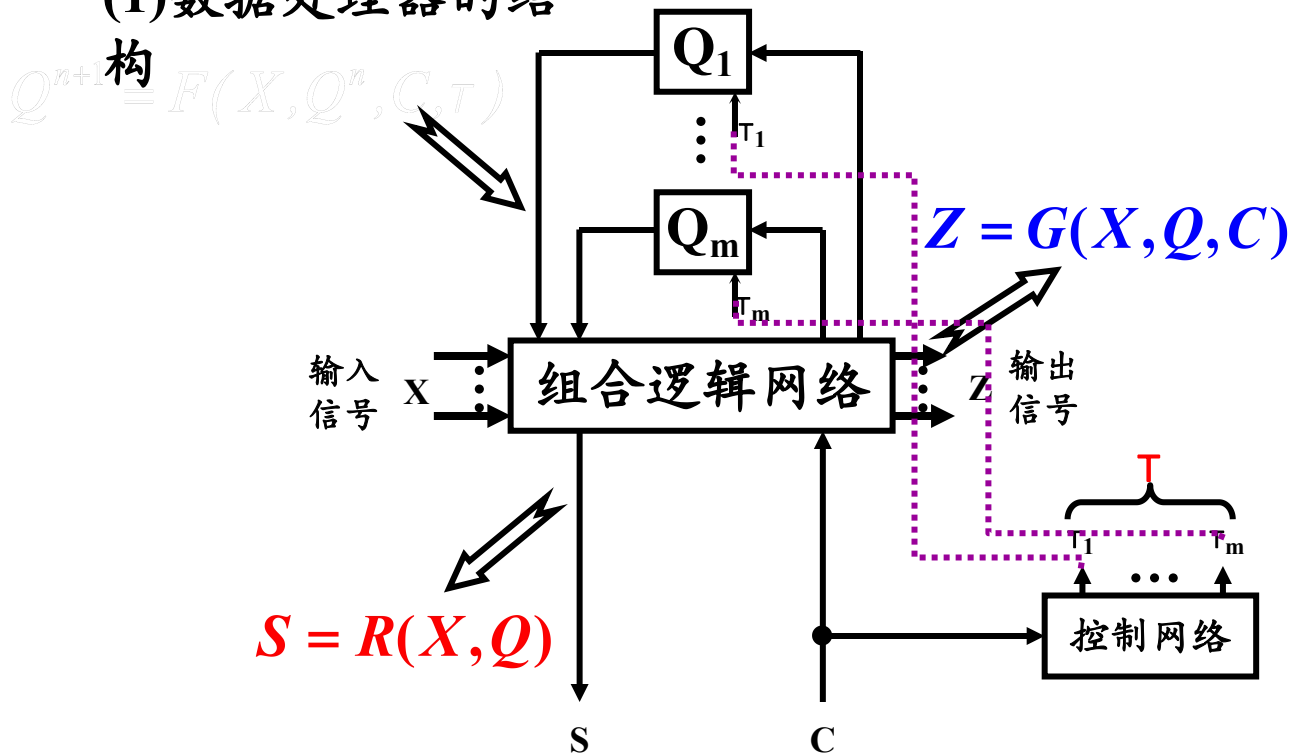


图7.1.2 数据处理器模型



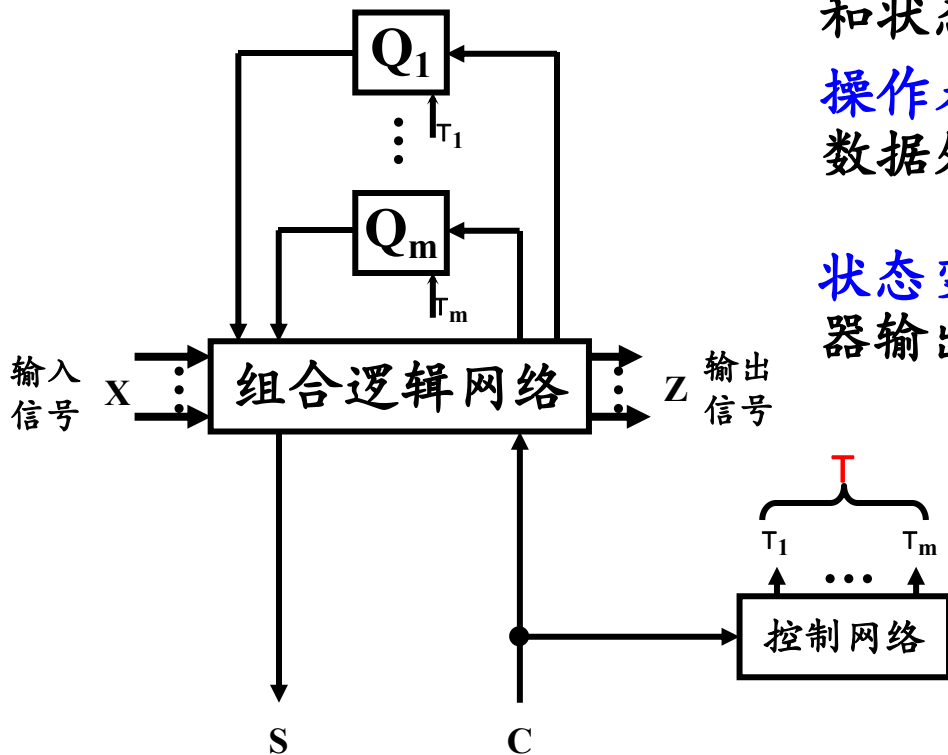
(2) 数据处理器的描述方法

明细表：规定数据处理任务的表格

明细表两个子表：操作表和状态变量表。

操作表：列出在控制信号下，
数据处理器应实现的操作。

状态变量表：定义数据处理器输出的状态变量和信号。



设一个简单数据处理器如图

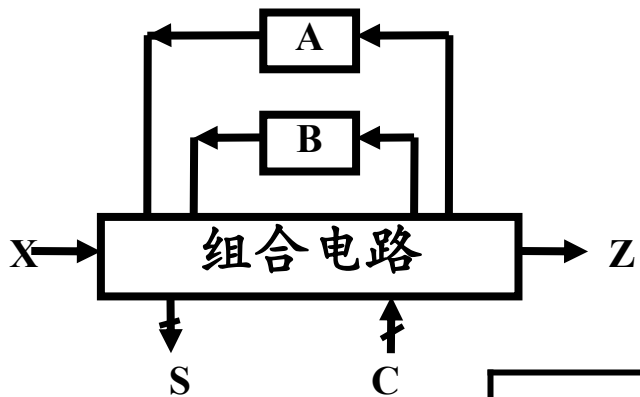


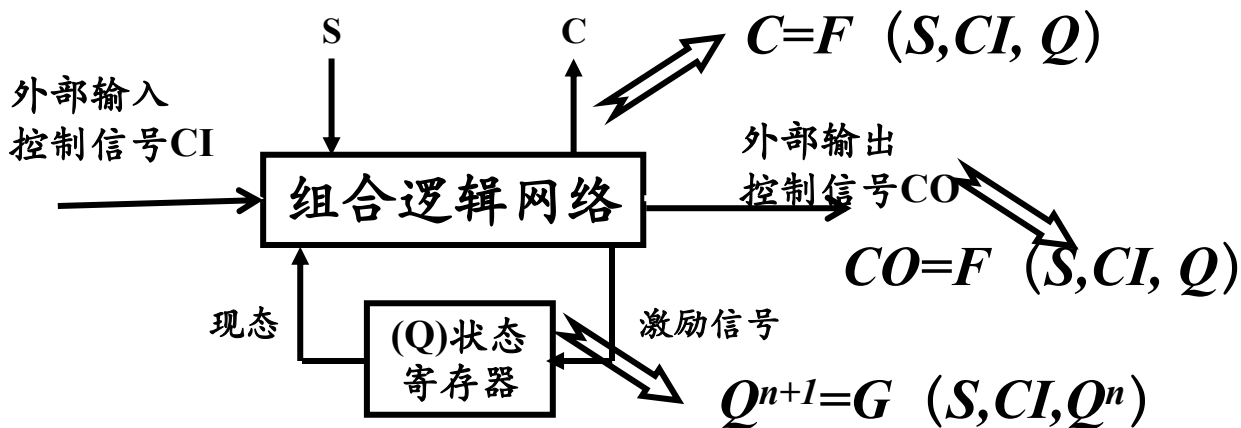
表7.1.1 数据处理
器明细表

操作表		状态变量表	
控制信号	操作	状态变量	定义
NOP	无操作	S_1	$X > 0$
ADDA	$A \leftarrow A + X$	S_2	$X < 0$
ADDB	$B \leftarrow B + X$	输出 $Z = A$	
CLAB	$A \leftarrow 0, B \leftarrow 0$		

2) 控制器

实现一个较复杂的任务，必存在一个算法，控制器就是用来规定算法的步骤。

控制器决定算法步骤，必须有记忆能力，所以它是一个时序电路，应包含存储器



控制器的描述方法：状态转移图或状态转移表

7.1.2 对数字系统的时序的约定

1.同步数字系统

- (1)只有一个系统时钟；
- (2)输入信号都与系统时钟同步；
- (3)系统时钟同时到达所有存储元件的时钟脉冲输入端。

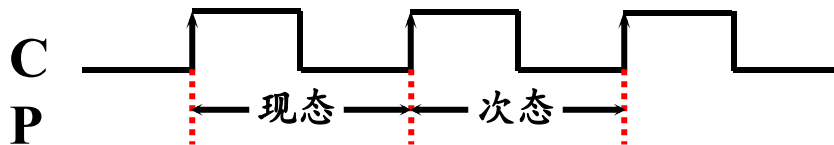
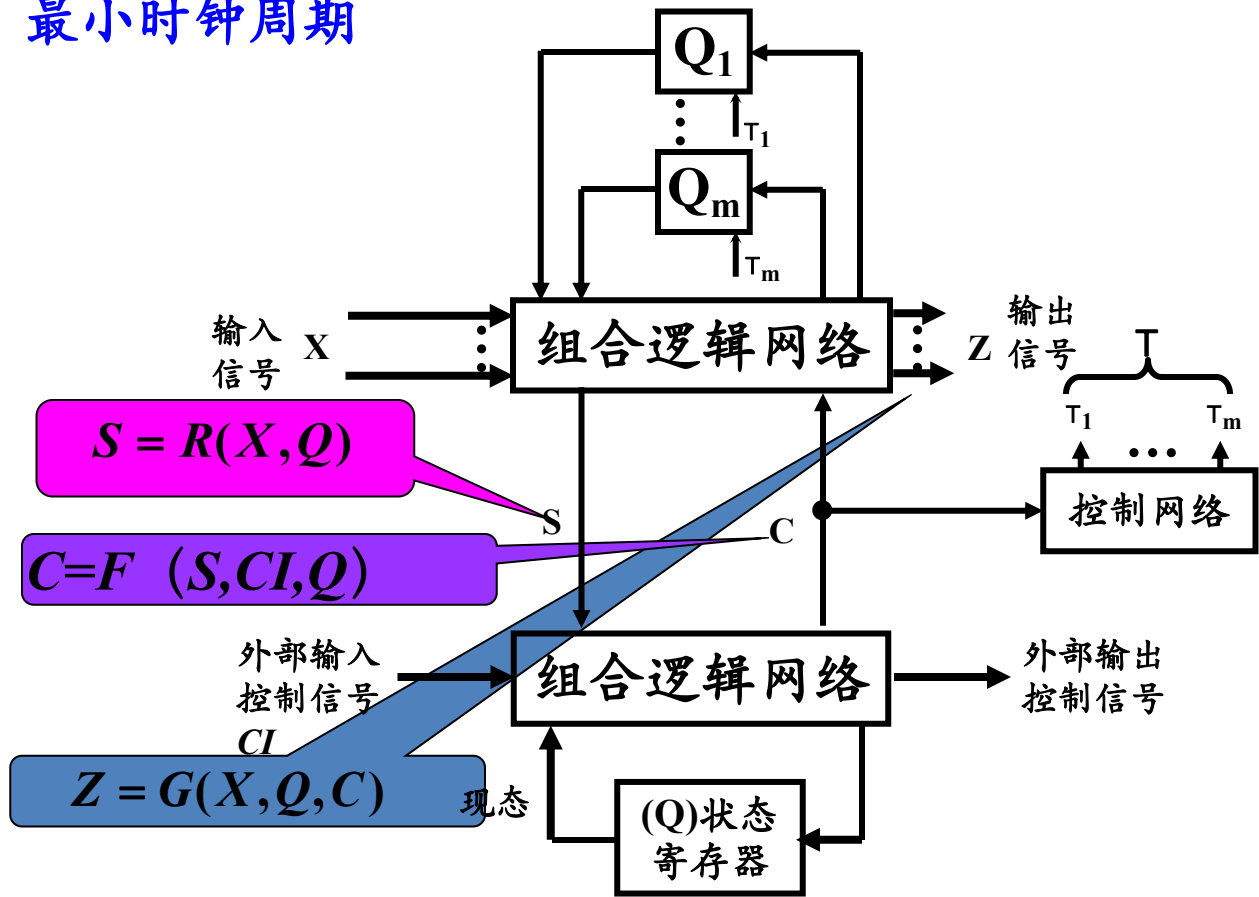
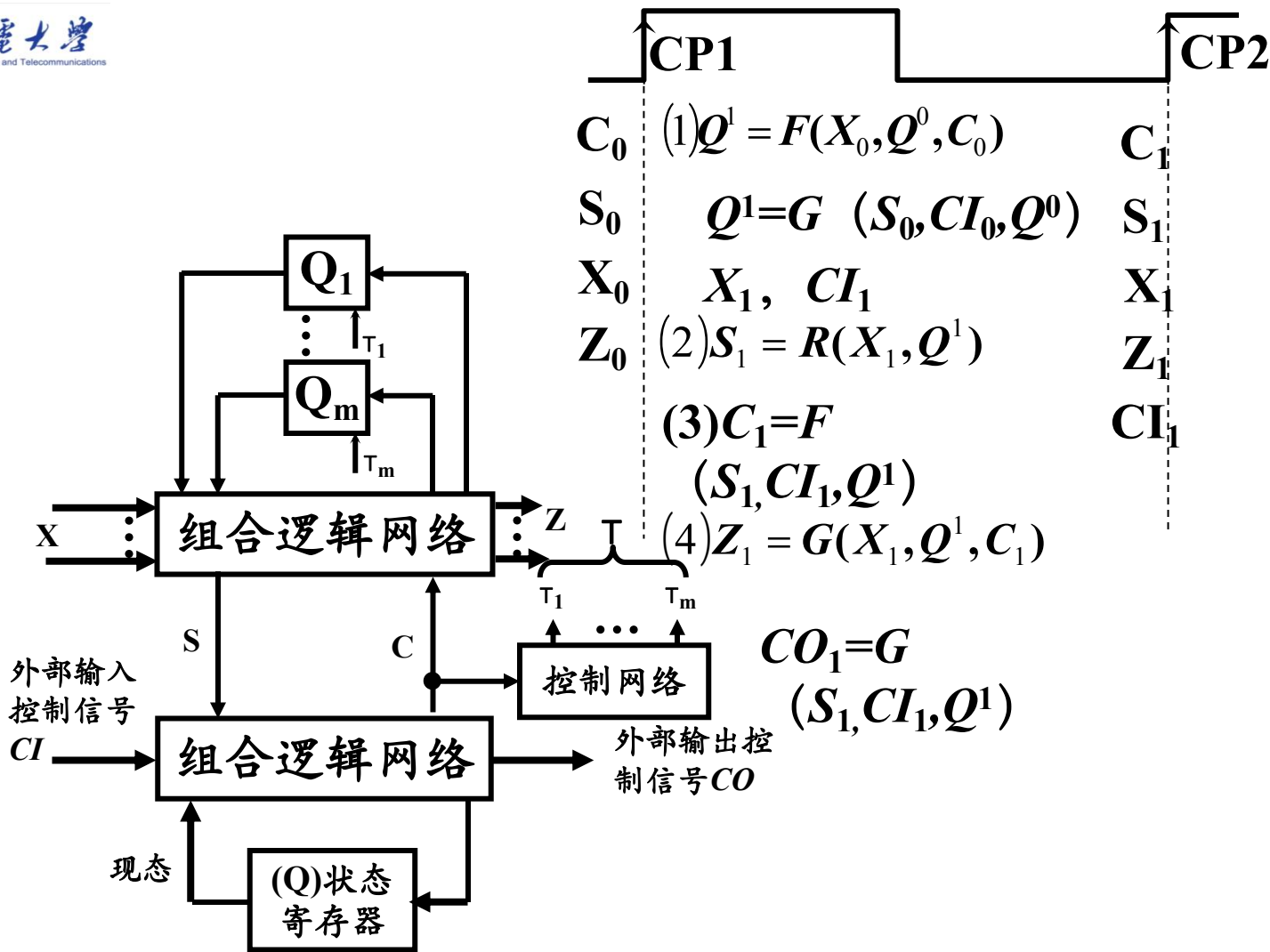


图7.1.4 系统时钟脉冲波形



1) 最小时钟周期







时钟脉冲有效边沿到达之前：

所有与操作有关的信号都应达到稳定值。

时钟脉冲有效边沿到达之后：

(1) 寄存器状态 Q 更新，同时输入信号 X 变化，根据 $S=R(X, Q)$ ，形成新的状态变量 S 。

(2) S 稳定后，控制器根据 $C=F(S, CI, Q)$ （其中 Q 是已更新的寄存器状态）形成控制信号 C 。

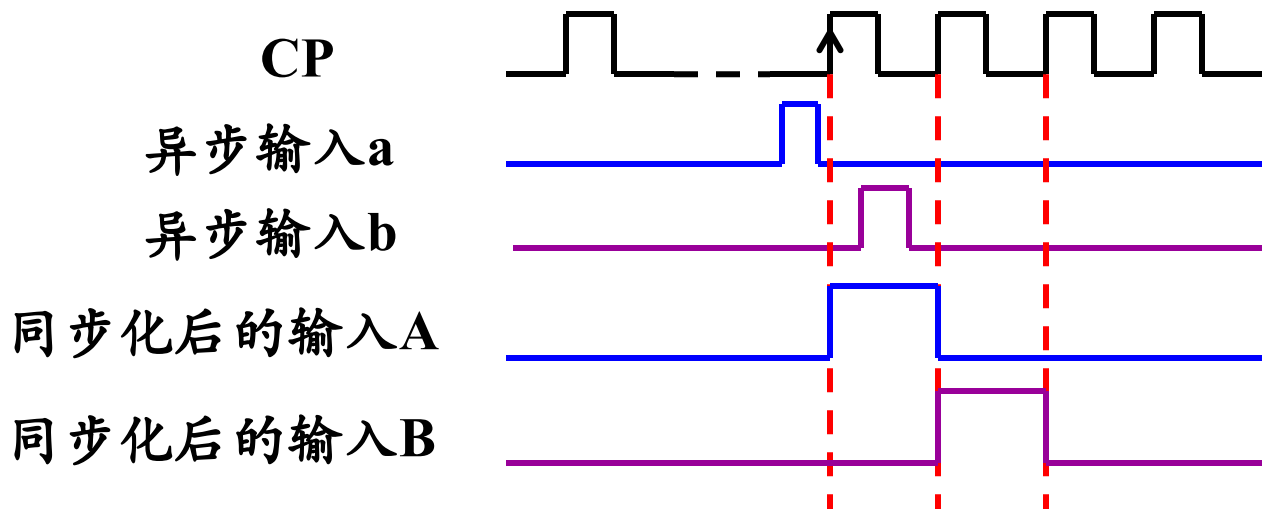
(3) C 稳定以后，建立稳定的 T 信号和电路的输出信号 $Z=G(X, Q, C)$ 。

最小周期由以上操作时间决定。



3. 异步输入信号转换成同步输入信号

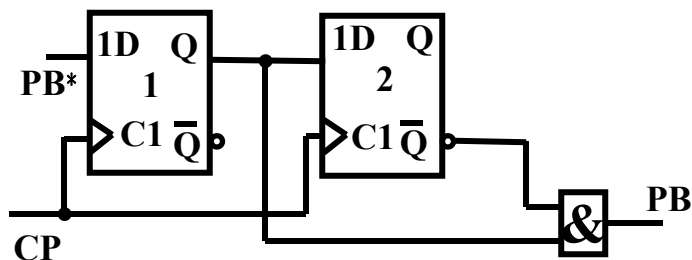
异步输入信号：早于或晚于系统时钟有效沿出现的输入信号。



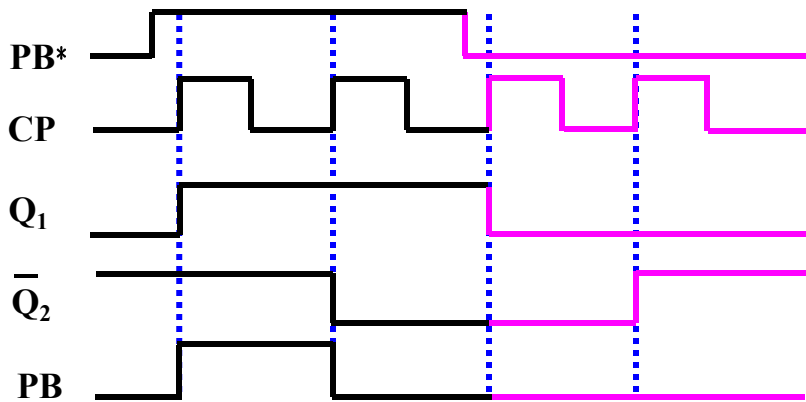
同步化处理的思路:

(1) 将异步输入信号寄存并保留到下一个系统时钟出现为止;

(2) 让同步化后的输入与当前系统时钟的有效时刻同时出现, 并保持一个时钟周期。



(a) 电路

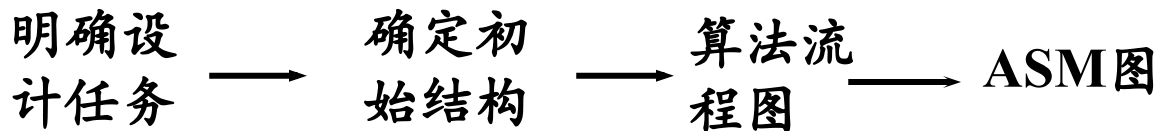


(b) 波形图

图7.1.7 适应于宽脉冲的单脉冲电路

7.1.4 数字系统的设计步骤

1. 系统设计



确定输出和输入之间的关系，找到实现数字系统的设计原理和方法。



划分系统的控制单元和受控单元，确定初始结构框图。

建立算法流程图，表示解决问题的步骤。

根据一定的规则将算法流程图转换成ASM图

2、逻辑设计

当系统中各个子系统（指最低层子系统）或部件的逻辑功能和结构确定后，采用比较规范的形式来描述系统的逻辑功能。

① 数据处理器设计

建立操作明细表

② 控制器设计

建立状态转移表





3、电路设计

选择合理的器件和连接关系，以实现系统逻辑要求。电路设计的结果常采用两种方式来表达：**电路图方式、硬件描述语言方式。**



4、物理设计

印制线路板或可编程器件或ASIC等设计、安装和调试。

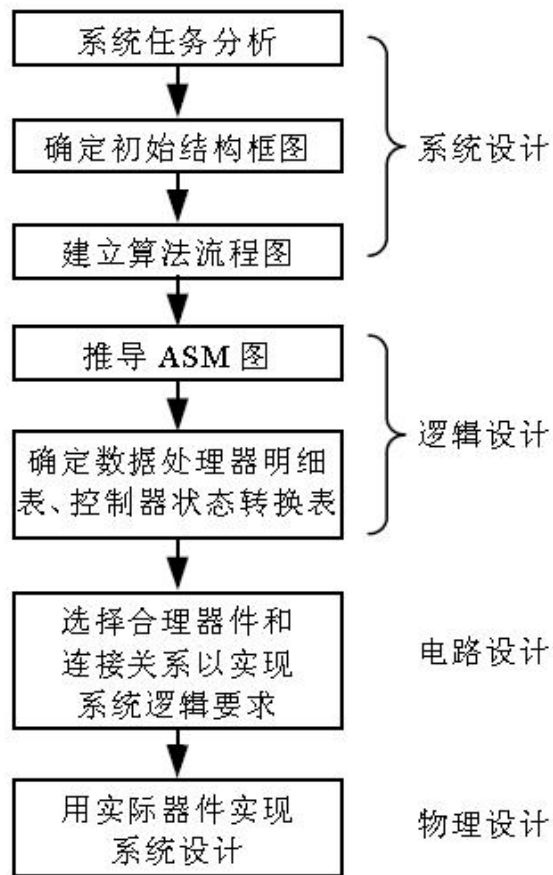


图 7.1.9 数字系统设计步骤



7.2 数字系统的描述工具

7.2.1 寄存器传输语言(RTL)

寄存器操作

对寄存器所存信息的加工和存储

寄存器传输语言

既表示了寄存器传输操作，又和硬件间有个简单的对应关系的一种方便的设计工具。



寄存器具有广义的概念

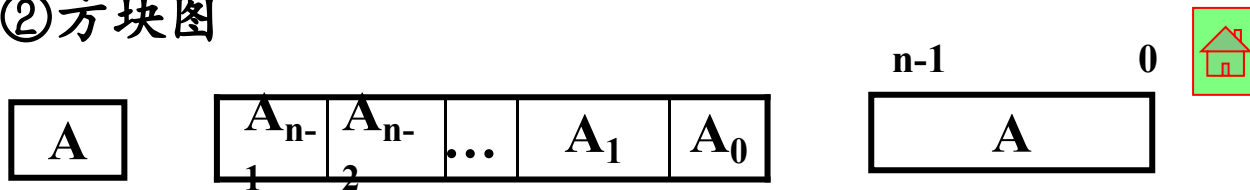
既包括暂存信息的寄存器，也包括移位寄存器、计数器、存储器等

1.传输操作

寄存器的表示方法

①大写英文字母

②方块图



(a)寄存器 A (b)寄存器A的位分布图 (c)寄存器A的位编号

图7.2.1 寄存器方块图表示

传输操作

$$\overline{X} \cdot T_1 : A \leftarrow B$$

↑ 控制函数
 控制函数结束

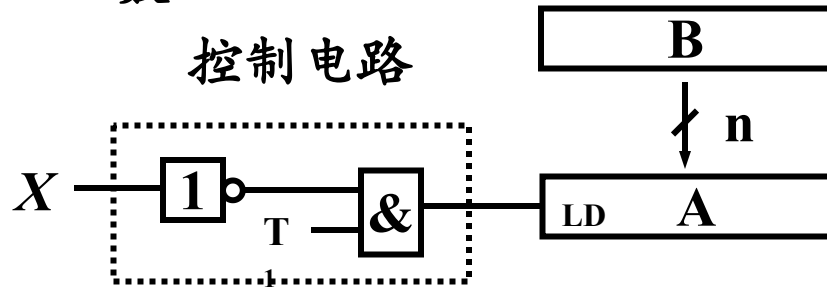


图 7.2.2 实现语句 $\overline{X} \cdot T_1 : A \leftarrow B$ 的电路示意图

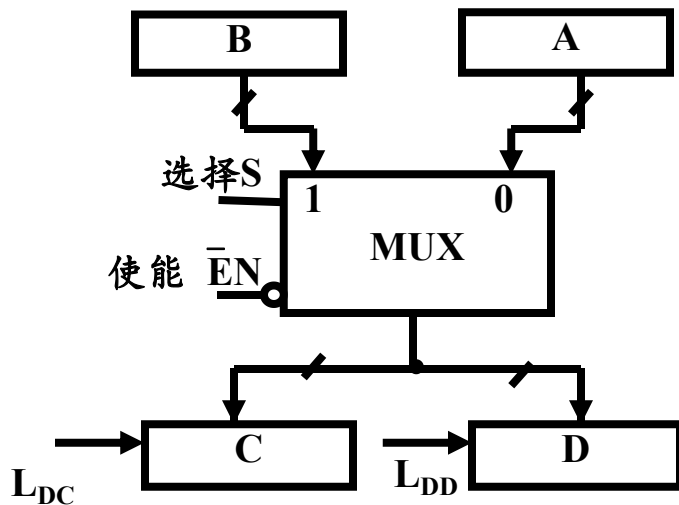


例：设两个源寄存器A、B,两个目标寄存器C、D。
试通过一个如下图所示的二选一数据选择器实现
如下寄存器传输语言：

$T_1: C \leftarrow A$

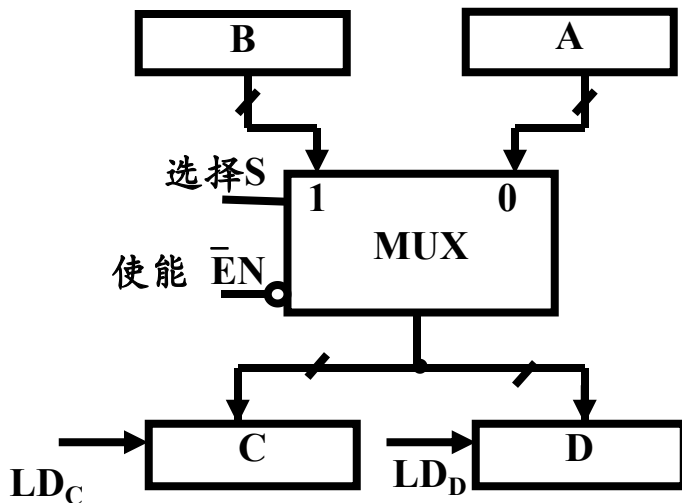
$T_5: C \leftarrow B$

$T_6: D \leftarrow B$ ：画出逻辑图





$T_1: C \leftarrow A$
 $T_5: C \leftarrow B$
 $T_6: D \leftarrow B$
 列真值表:



T_1	T_5	T_6	S	$\bar{E}N$	LD_C	LD_D
0	0	0	Φ	Φ	0	0
0	0	1	1	0	0	1
0	1	0	1	0	1	0
0	1	1	1	0	1	1
1	0	0	0	0	1	0
1	0	1	Φ	Φ	Φ	Φ
1	1	0	Φ	Φ	Φ	Φ
1	1	1	Φ	Φ	Φ	Φ

所以:
 $S = \bar{T}_1$;
 $\bar{E}N = 0$;
 $LD_C = T_1 + T_5$;
 $LD_D = T_6$

2、算术操作

$$T_2 : A \leftarrow A + B$$

$$T_5 : A \leftarrow A + 1$$

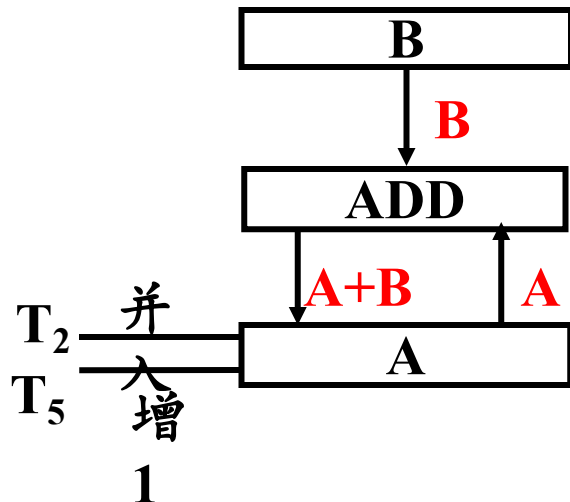


图7.2.5 完成加和增“1”操作的示意图



3、逻辑操作

与运算符“ \wedge ”；或运算符“ \vee ”

——为了与算术运算的符号“ \cdot ”、“ $+$ ”区别。

$T_1+T_2: \underline{A \leftarrow A+B, \quad C \leftarrow D \vee F}$



两个操作同时实现（并行关系）



$$T_1+T_2: A \leftarrow A+B, C \leftarrow D \vee F$$

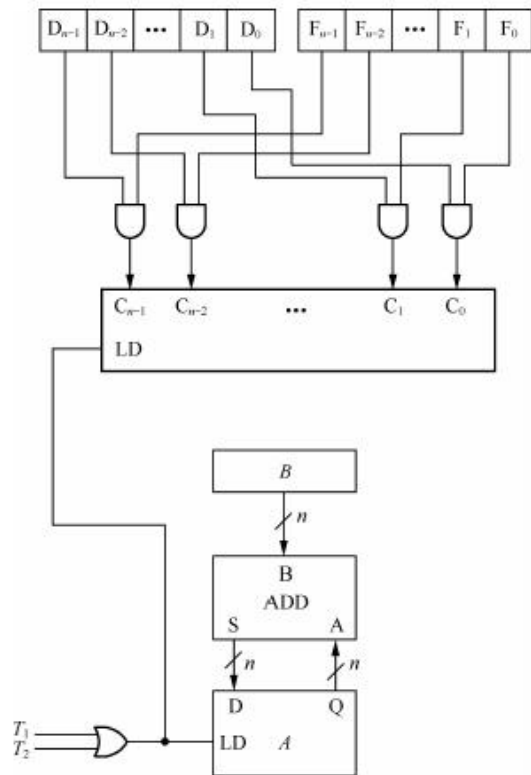
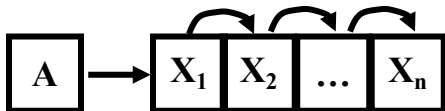


图 7.2.6 $T_1+T_2: A \leftarrow A+B, C \leftarrow D \vee F$ 对应的硬件示意图

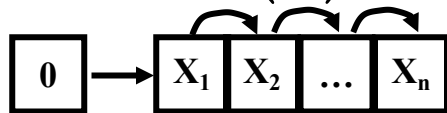


4、移位操作

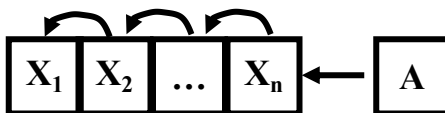
1.右移操作: $X \leftarrow SR(A, X)$



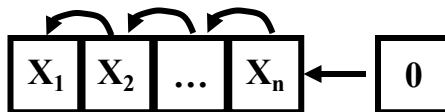
$X \leftarrow SR(X)$



2.左移操作: $X \leftarrow SL(X, A)$



$X \leftarrow SL(X)$





5、条件控制语句

P: IF (条件) Then (微操作1) Else (微操作2)



控制函数

例:

T2: IF (C=0) THEN (F←1) ELSE (F←0) 可以写成两个一般语句:

$\bar{C} \cdot T2: F \leftarrow 1$

$C \cdot T2: F \leftarrow 0。$



小结

一条RTL语句：描述数字系统所处的一个状态。

其操作：说明数据处理器要实现的操作。

控制函数：说明控制器发出的命令。

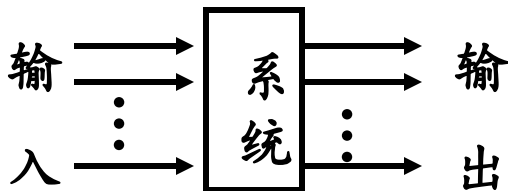
一个RTL语句可以定义一个数字系统。

7.2.2 方框图

1.作用

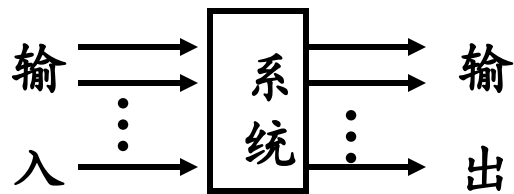
描述数字系统的总体结构。

2.示例

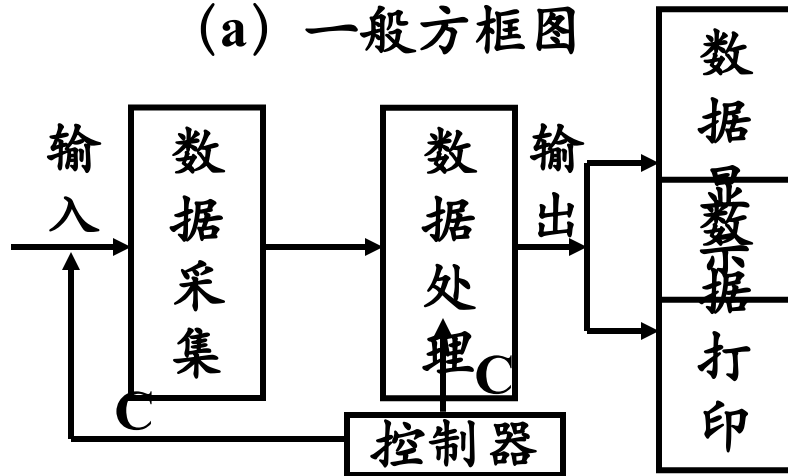


(a) 一般方框图

图7.2.9 一个智能仪表的方框图



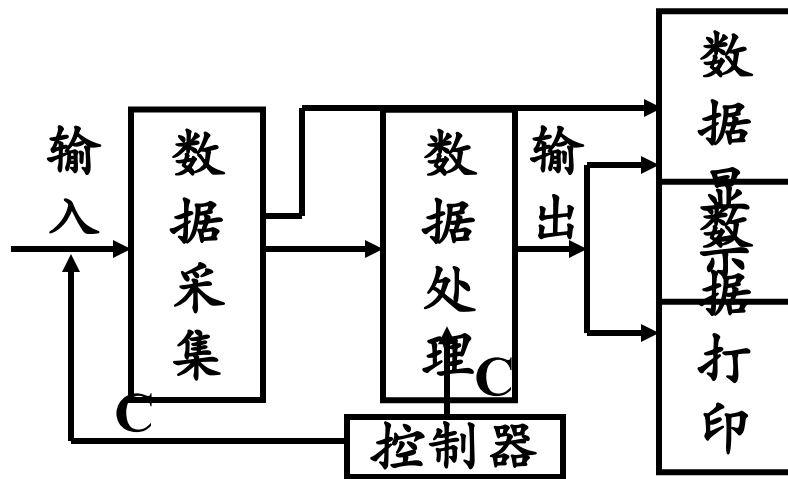
(a) 一般方框图



(b) 系统分解图

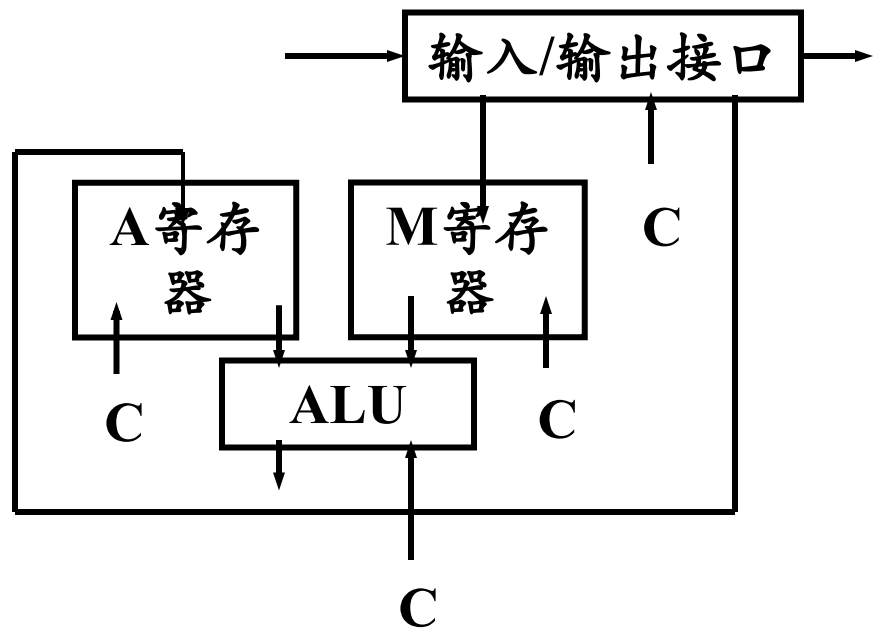
图7.2.9 一个智能仪表的方框图





(C) 进一步细化方案

图7.2.9 一个智能仪表的方框图



(d) 数据处理模块的细化

图7.2.9 一个智能仪表的方框图

7.2.3 算法流程图

1. 作用

描述算法。

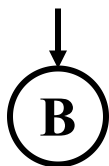
注意：按照事件的先后次序排列的，与电路的时序无严格的对应关系。

2. 基本符号

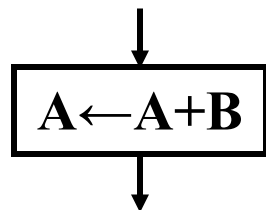
入口点；出口点；传输框；判断框



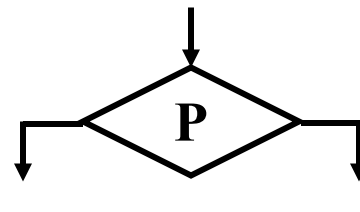
(a) 入口点



(b) 出口点



(c) 传输框



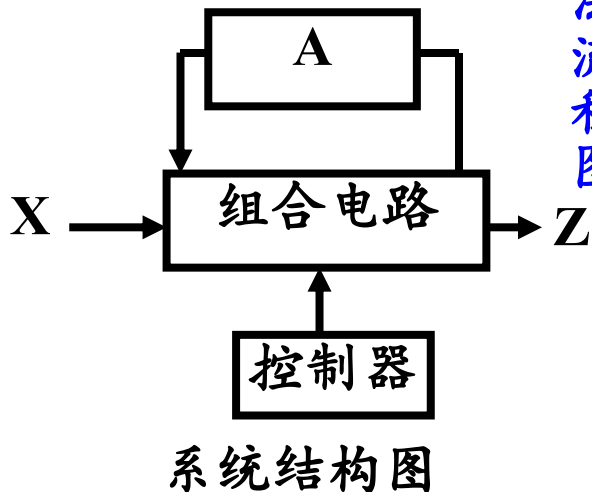
(d) 判断框

图7.2.10 流程图符号

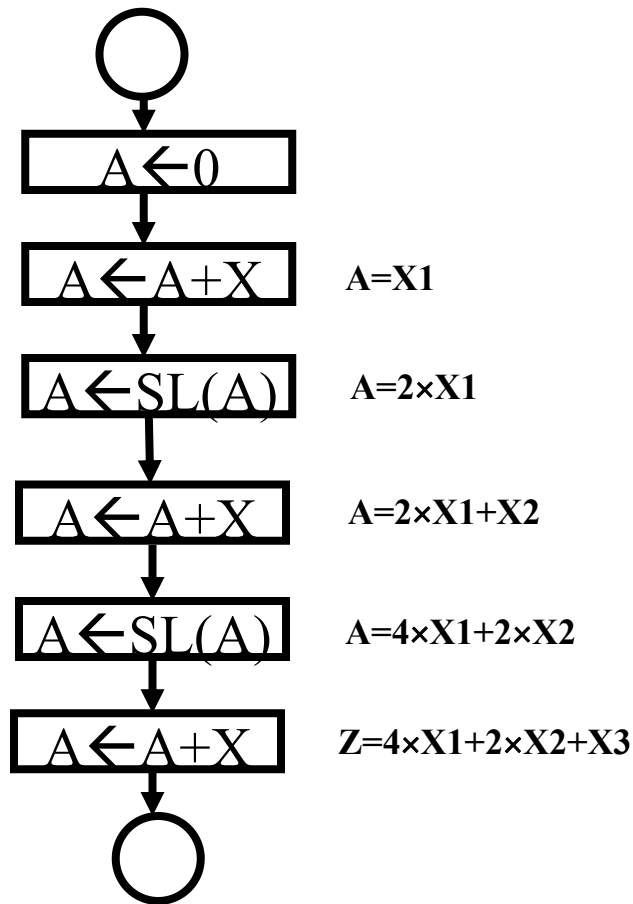


例7.2.2: 函数求值, 计算 $Z=4 \times X_1 + 2 \times X_2 + X_3$ 的值。

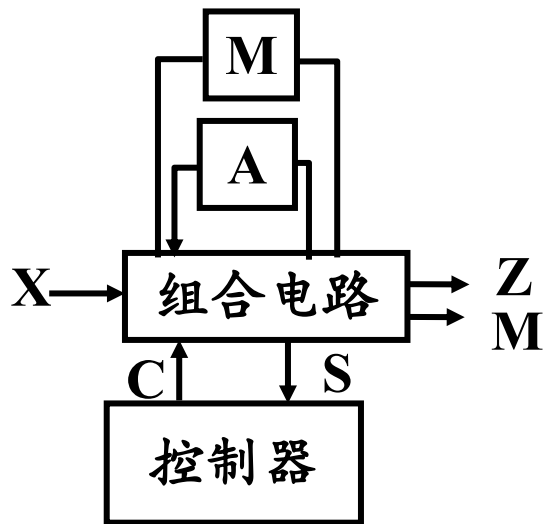
任务: 输入端X串行输入 X_1, X_2, X_3 , 计算完成后, 提供输出Z。



算法流程图

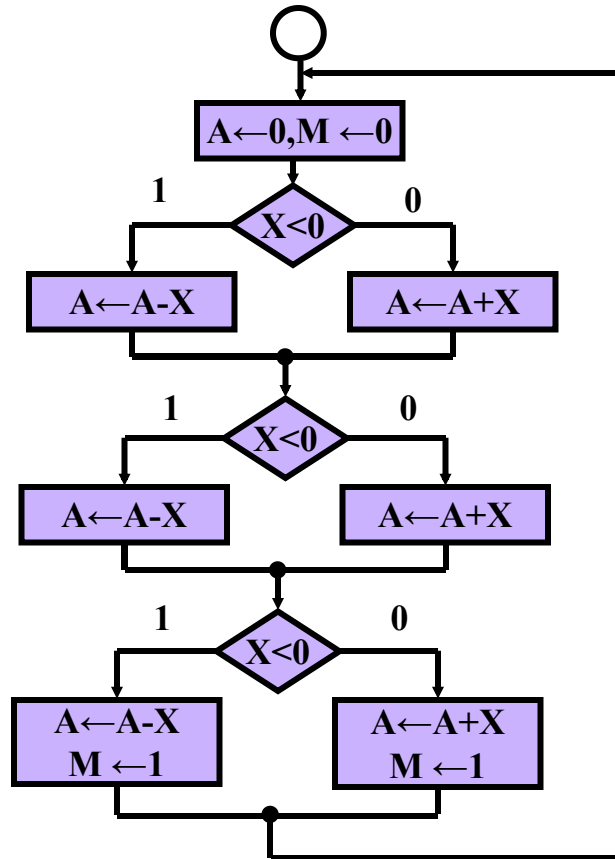


例 7.2.3 绝对值计算，计算 $Z = |X_1| + |X_2| + |X_3|$ 。



例 7.2.3 系统结构图

图 7.2.12 算法流程图



7.2.4 算法状态机图 (ASM图)

ASM图是一种描述时钟驱动的数字系统工作流程的方法。

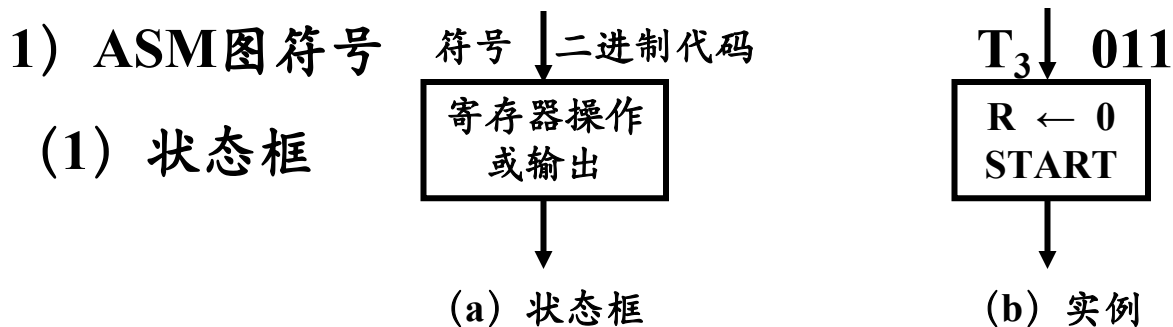
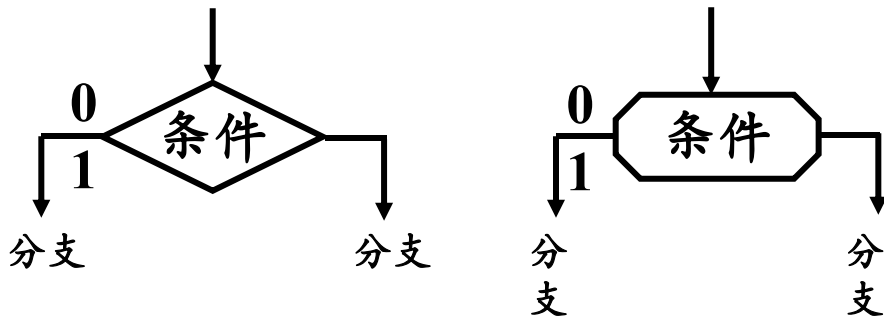


图7.2.13 状态框

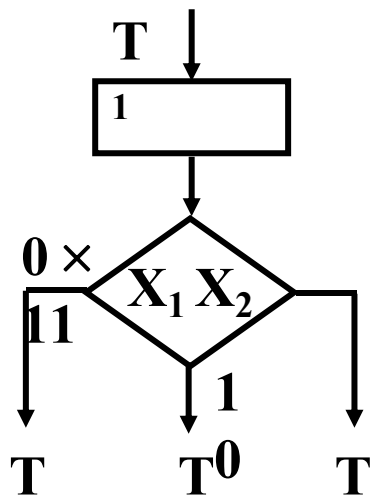
状态框中的操作通常用RTL语言表示，若用助记符代替，则该助记符是控制器发出的控制命令。

(2) 判断框

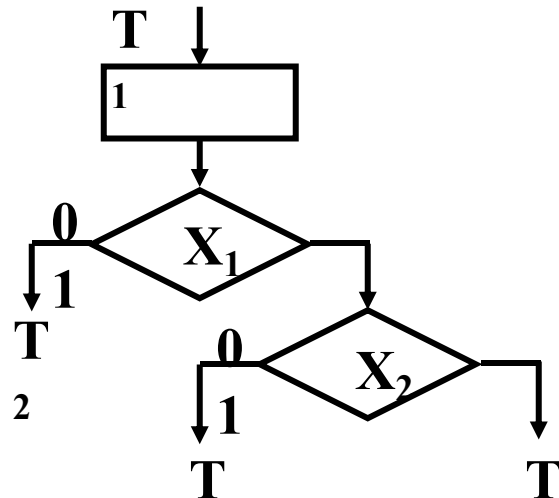


判断框表示判断变量对控制器的影响，因此判断框中的条件通常为处理器的状态信号或外部输入控制信号。





(a) 真值表图解分支表示



(b) 变量优先级分支表示

图7.2.15 判断框3个分支表示

(2) 条件框

条件框为ASM图所特有的，条件框内的操作和输出是在给定条件下，判断条件被满足时才发生的，所以条件框的输入必定与判断框的分支相连。

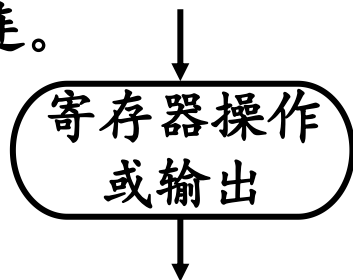


图 7.2.16 条件框

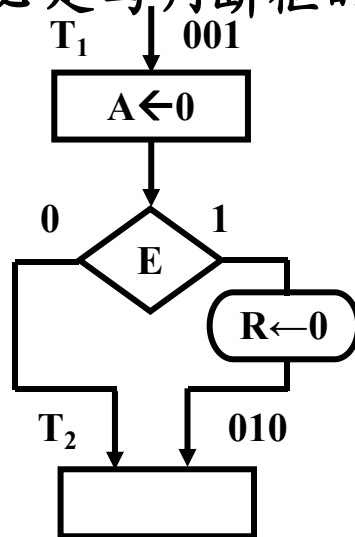
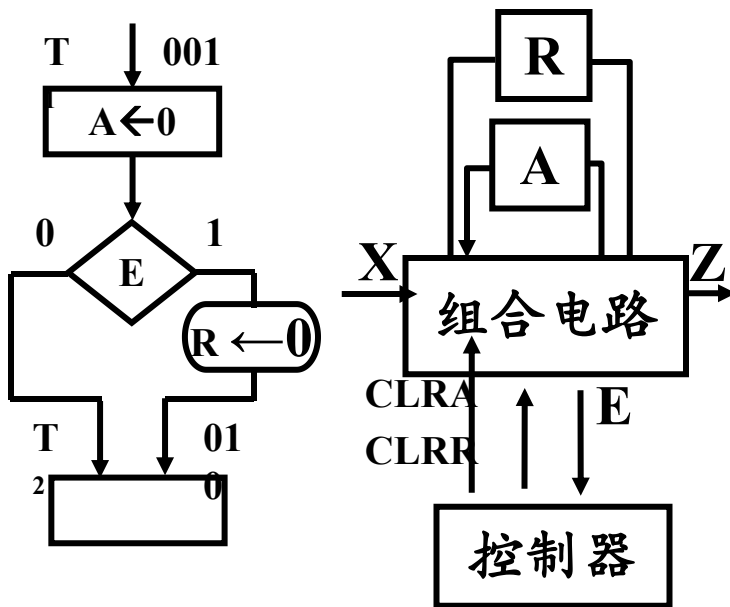


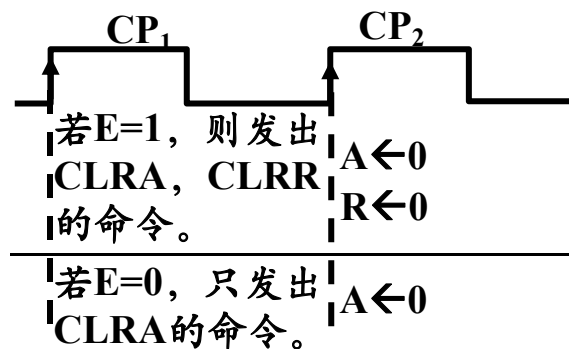
图 7.2.16 条件框举例

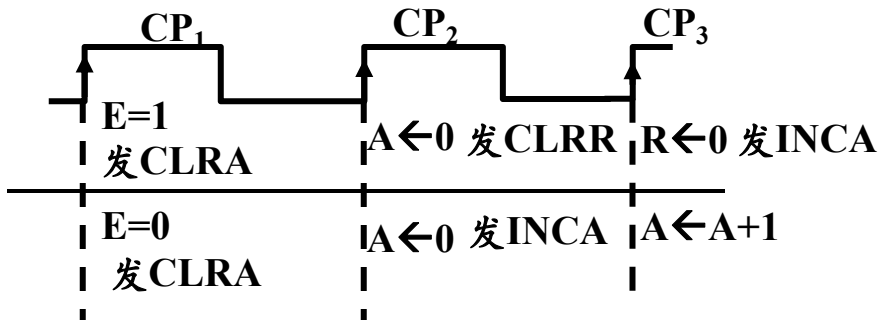
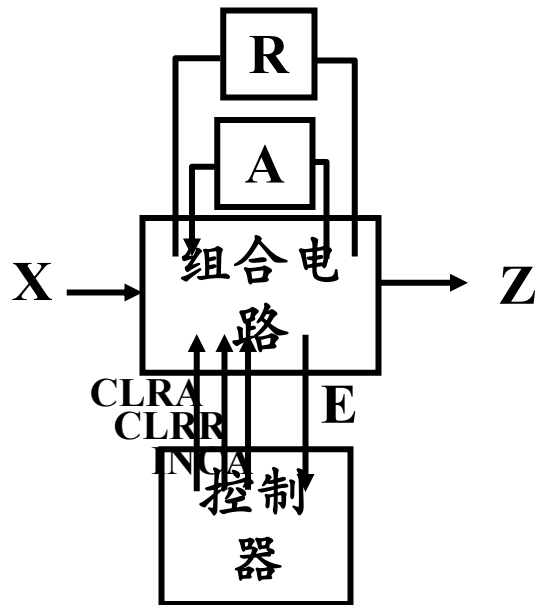
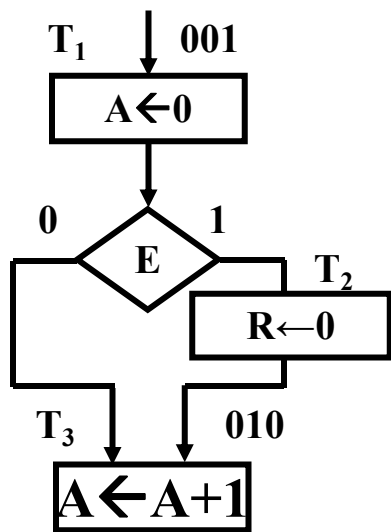
2. ASM块

- 1、一个ASM块表示一个时钟周期内系统的状态；
- 2、一个ASM块由一个状态框和若干与之相连的判断框和条件框组成；
- 3、一个ASM块内所有操作在同一有效时钟沿完成。



$A \leftarrow 0$ 和 $R \leftarrow 0$ 在同一个时钟沿完成。且在进入 T_2 状态的有效时钟沿完成。





(3) ASM图与控制器状态转移图的关系；

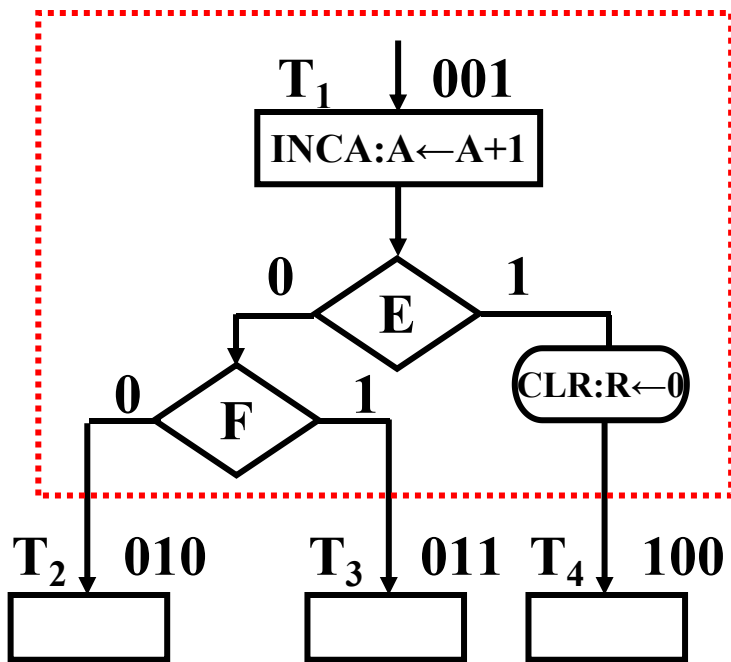


图 7.2.17 ASM块

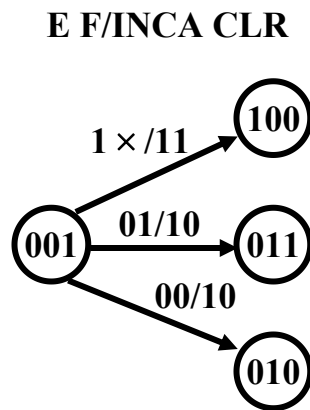


图 7.2.19 等效状态转移图

状态转移图只能表述控制器状态转移情况，但无法表示处理器完成何种操作。



4.各种逻辑框之间的时间关系

例：一个数字系统的数据处理器有2个触发器E和F及1个二进制计数器A，计数器的各个位分别用 A_4 、 A_3 、 A_2 、 A_1 标记， A_4 为最高位， A_1 为最低位。启动信号S使计数器A和触发器F清“0”，从下一个时钟脉冲开始，计数器增1，一直到系统停止工作为止。系统的操作序列由 A_3 和 A_4 之值决定，即：

- ① $A_3=0$ ，触发器E清“0”，并继续计数。
- ② $A_3=1$ ，触发器E置“1”，并检验 A_4 ，若 $A_4=0$ ，继续计数；若 $A_4=1$ ，触发器F置“1”，系统停止计数。



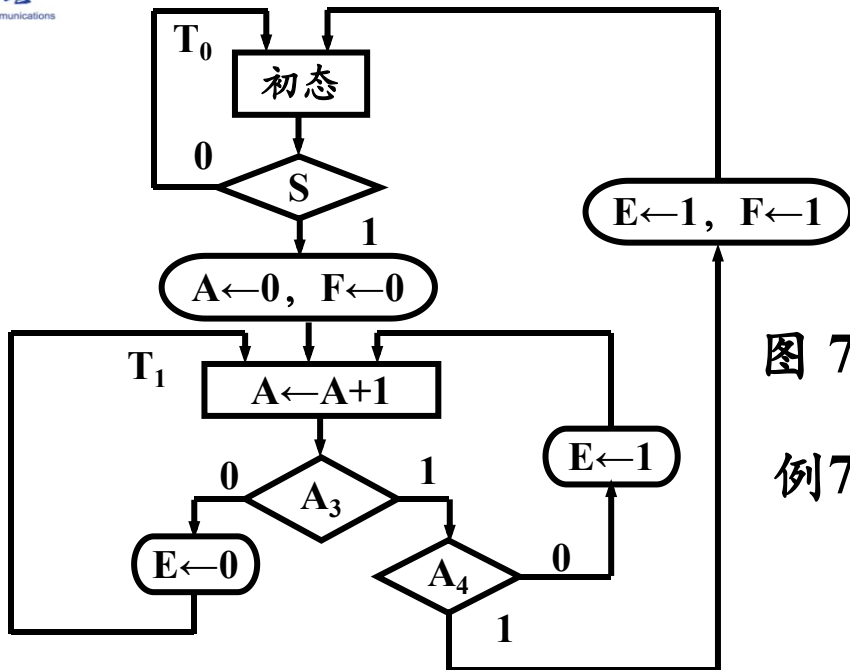


图 7.2.22

例7.2.4 ASM图

A ₄	A ₃	A ₂	A ₁	E	F	条件	状态
0	0	0	0				

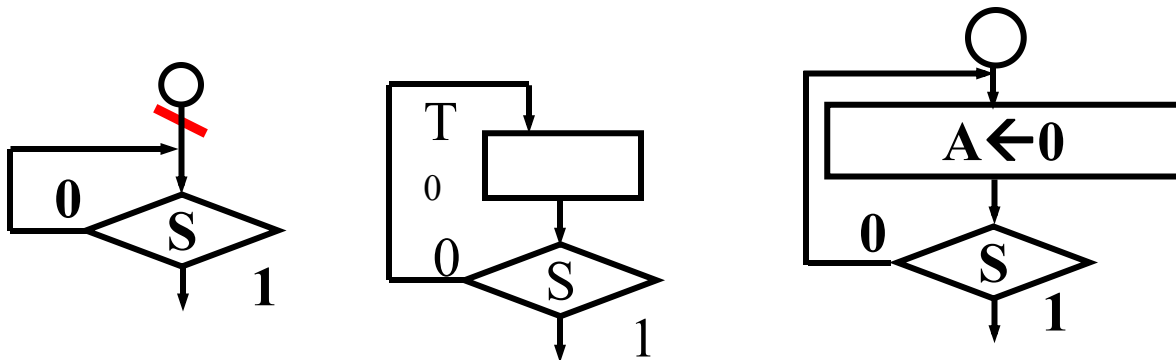
表7.2.4 ASM图的操作序列

A ₄	A ₃	A ₂	A ₁	E	F	条件	状态
0	0	0	0	1	0	A ₃ =0 A ₄ =0	T ₁
0	0	0	1	0	0		
0	0	1	0	0	0		
0	0	1	1	0	0		
0	1	0	0	0	0	A ₃ =1 A ₄ =0	T ₁
0	1	0	1	1	0		
0	1	1	0	1	0		
0	1	1	1	1	0		
1	0	0	0	1	0	A ₃ =0 A ₄ =1	T ₁
1	0	0	1	0	0		
1	0	1	0	0	0		
1	0	1	1	0	0		
1	1	0	0	0	0	A ₃ =1	T ₁
1	1	0	1	1	1	A ₄ =1	T ₀

4.ASM图的建立

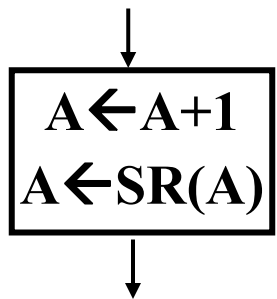
从算法流程图→ASM图

原则1：在算法的起始点安排一个状态；

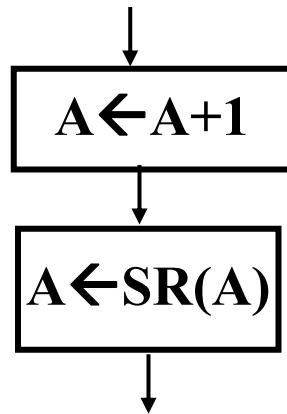


算法流程图 → ASM图 ← 算法流程图

原则2：必须用状态来分开不能同时实现的寄存器传输操作；



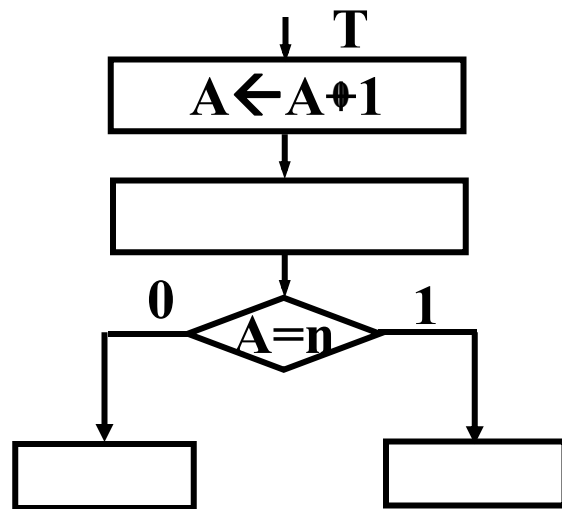
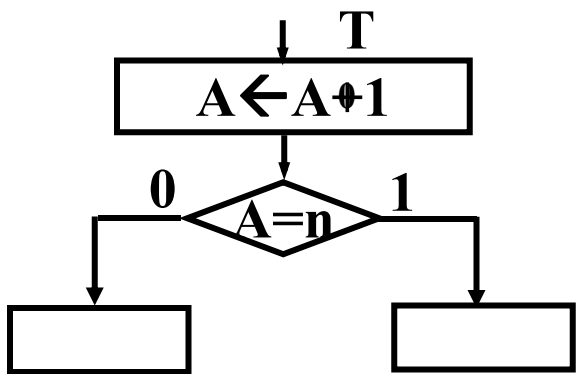
算法流程图



ASM图



原则3: 如果判断框中的转移条件受前一个寄存器操作的影响, 应在它们之间安排一个状态。



算法流程图



ASM图

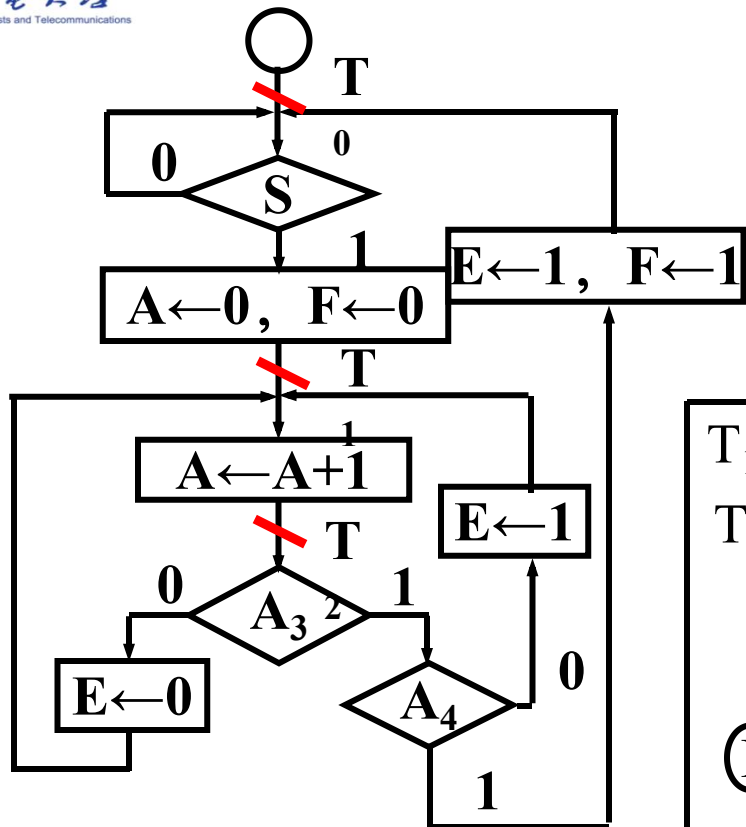


图 7.2.25 算法流程

图

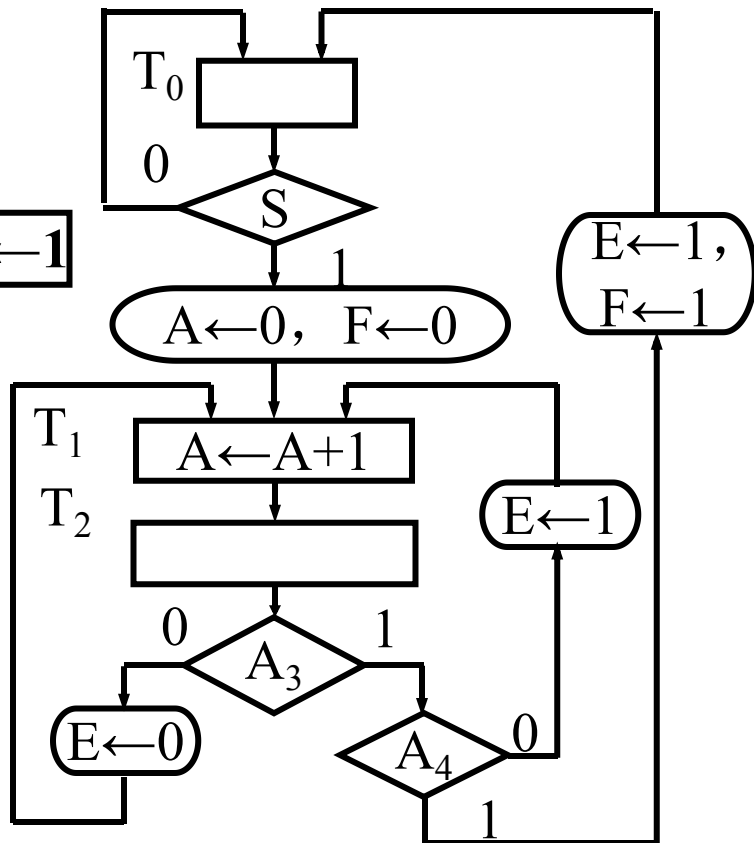
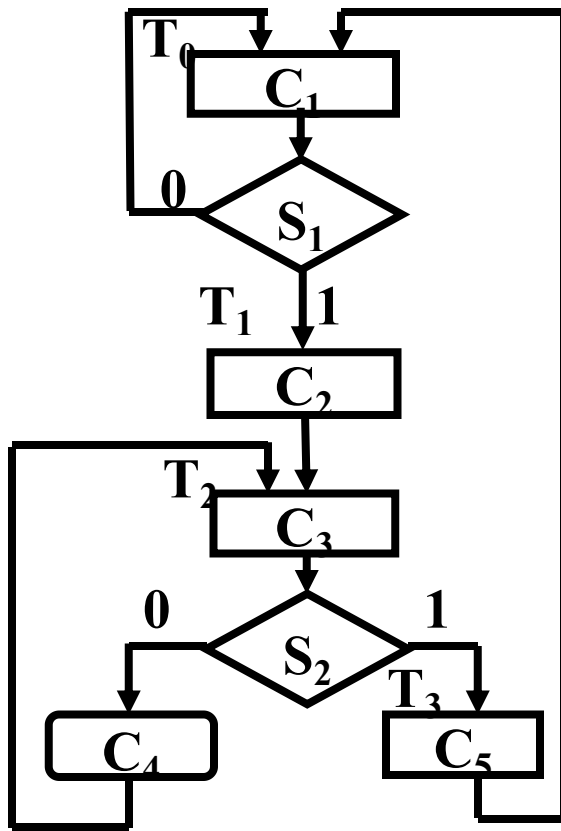
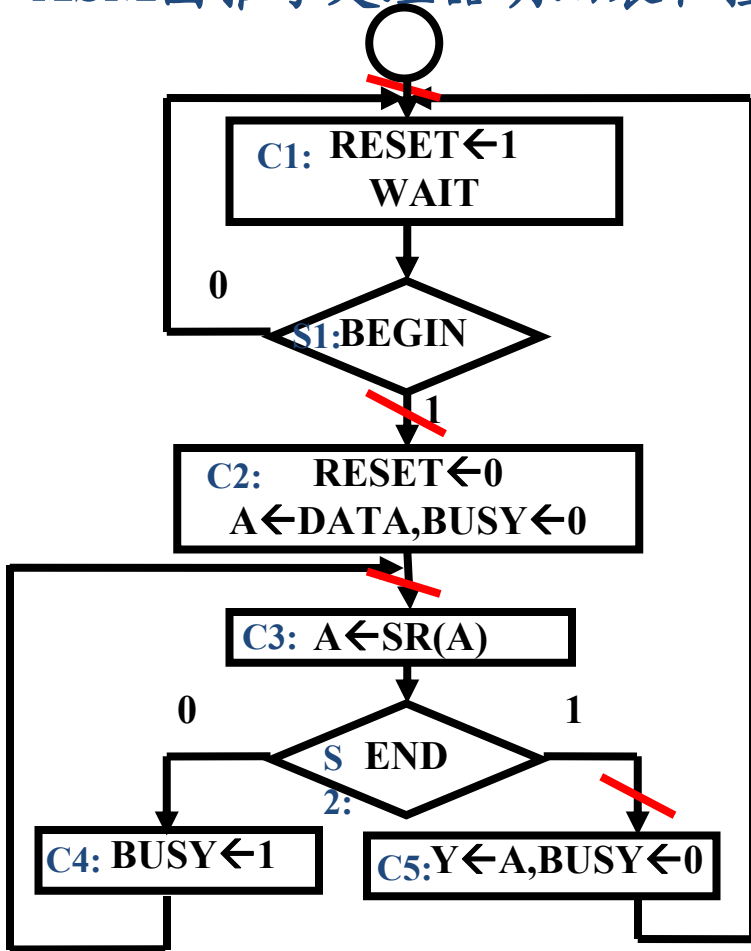
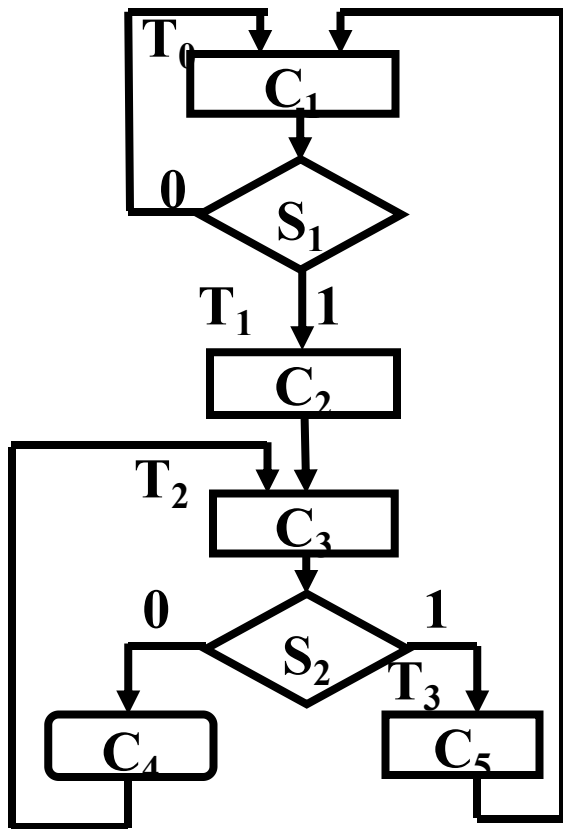


图 7.2.25 ASM图



5、ASM图推导处理器明细表和控制器状态转移图

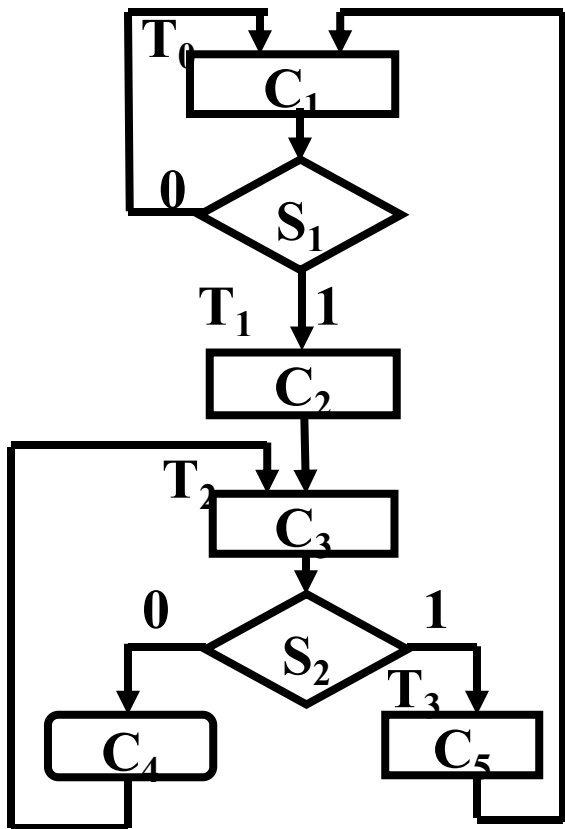




ASM图

操作表		状态变量表	
控制信号	操作	状态变量	定义
C ₁	RESET←1 WAIT	S ₁ S ₂	BEGIN END
C ₂	RESET←0 A←DATA, BUSY←0		
C ₃	A←SR(A)		
C ₄	BUSY←1		
C ₅	Y←A,BUSY←0		

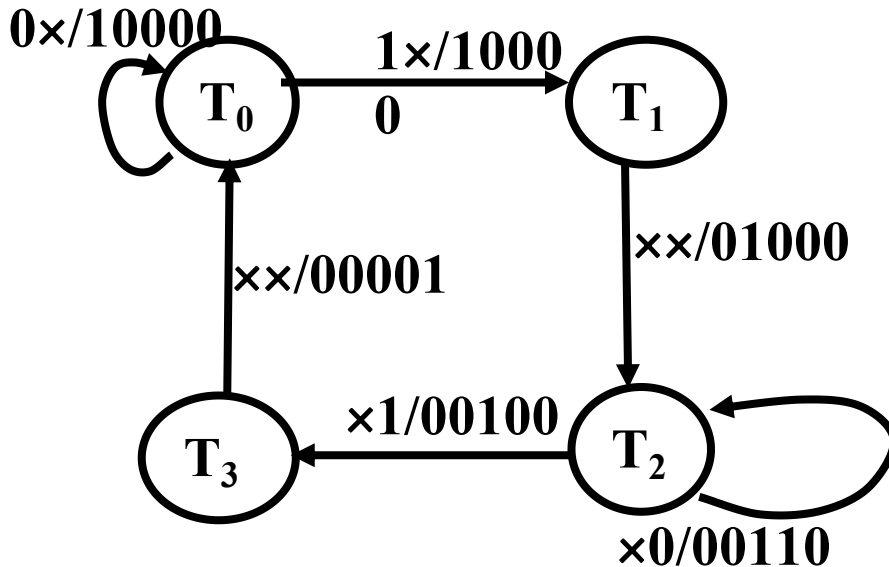
表12.3.20 处理器明细表



控制器的状态转移图

输入/输出:

$S_1S_2/C_1C_2C_3C_4C_5$





7.4 数字系统设计举例

例1 设计一个求两个4位二进制数之积的数字乘法器。被乘数为DX，乘数为DY，求两数之积的命令信号为START，Z为乘积。

一、系统级设计

1) 算法设计

表 7.4.1 乘法
的手算过程

运算过程		算式说明
	1 0 1 0	被乘数
×	1 1 0 1	乘数
	1 0 1 0	第一部分积
	0 0 0 0	第二部分积
	1 0 1 0	第三部分积
+	1 0 1 0	第四部分积
	1 0 0 0 0 1 0	乘积=部分积之和





算法规律:

(I) 两个 r 位的二进制数相乘, 乘积为 $2r$ 位。

(II) 乘数的第 i 位为0时, 第 i 位的部分积为0;

乘数的第 i 位为1时, 第 i 位的部分积是被乘数。

运算过程	算式说明
1 0 1 0	被乘数
× 1 1 0 1	乘数
1 0 1 0	第一部分积
0 0 0 0	第二部分积
1 0 1 0	第三部分积
+ 1 0 1 0	第四部分积
1 0 0 0 0 1 0	乘积=部分积之和

(III) 第 i 位的部分积相对于第 $i-1$ 位的部分积求和时左移一位。

为了便于数字电路实现, 将一次多数相加改为累加实现。累加的和称为部分和, 把它存如累加寄存器A中。

表 7.4.2 累计部分积的乘法过程

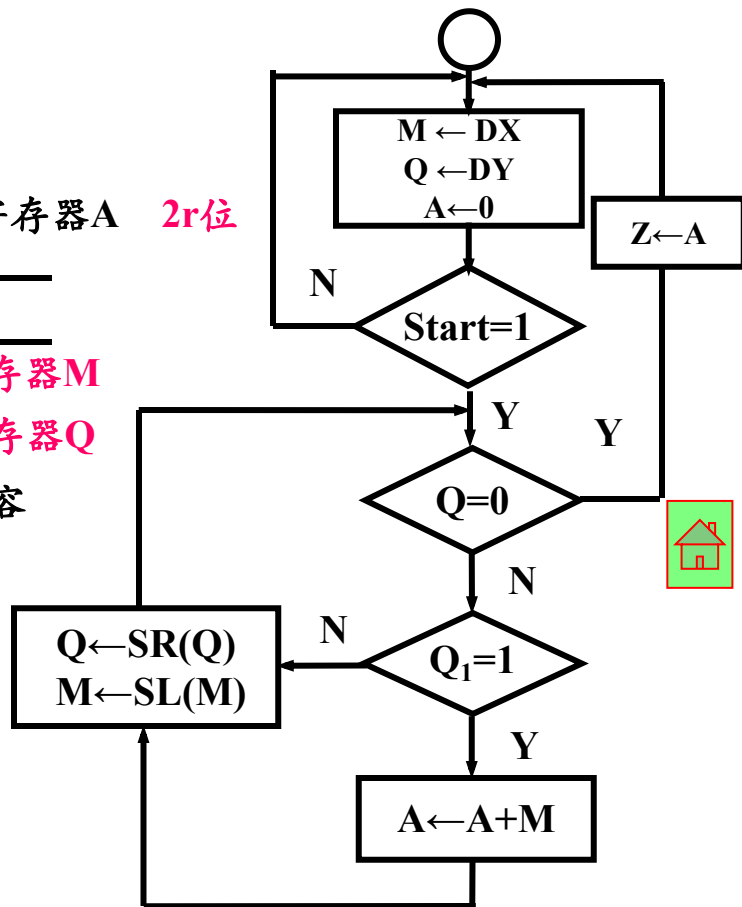
运算过程	算式说明
1 0 1 0	被乘数
× 1 1 0 1	乘数
0 0 0 0 0 0 0 0	累加器初始内容
+ 1 0 1 0	第一部分积
0 0 0 0 1 0 1 0	第一部分和
+ 0 0 0 0	第二部分积
0 0 0 0 1 0 1 0	第二部分和
+ 1 0 1 0	第三部分积
0 0 1 1 0 0 1 0	第三部分和
+ 1 0 1 0	第四部分积
1 0 0 0 0 0 1 0	乘积=第四部分和

- 1、如果乘数第*i*位为0，则累加器不进行任何操作。
- 2、如果乘数第*i*位为1，则将被乘数左移*i*-1位，再与累加器相加。
- 3、为了使算法简单，对乘数第*i*位的判断，可以将乘数寄存器每次右移一位，这样只判断最低位即可。

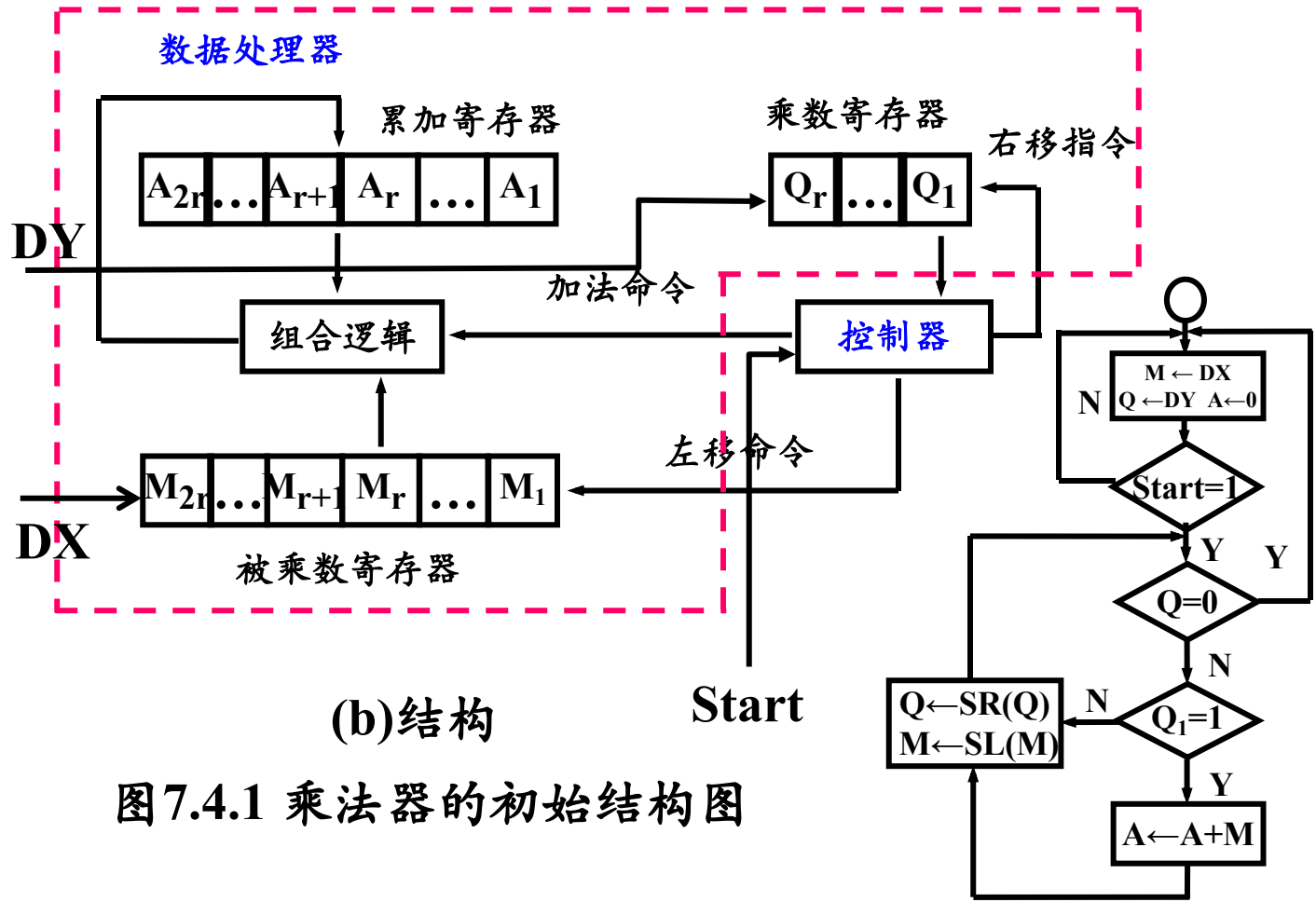
寄存器M存被乘数：2r位

寄存器Q存乘数：r位； 累加寄存器A 2r位

运算过程	算式说明
1 0 1 0	被乘数 寄存器M
× 1 1 0 1	乘数 寄存器Q
0 0 0 0 0 0 0 0	累加器A初始内容
+ 1 0 1 0	第一部分积
0 0 0 0 1 0 1 0	第一部分和
+ 0 0 0 0	第二部分积
0 0 0 0 1 0 1 0	第二部分和
+ 1 0 1 0	第三部分积
0 0 1 1 0 0 1 0	第三部分和
+ 1 0 1 0	第四部分积
1 0 0 0 0 0 1 0	乘积=第四部分和



(a) 算法流程图



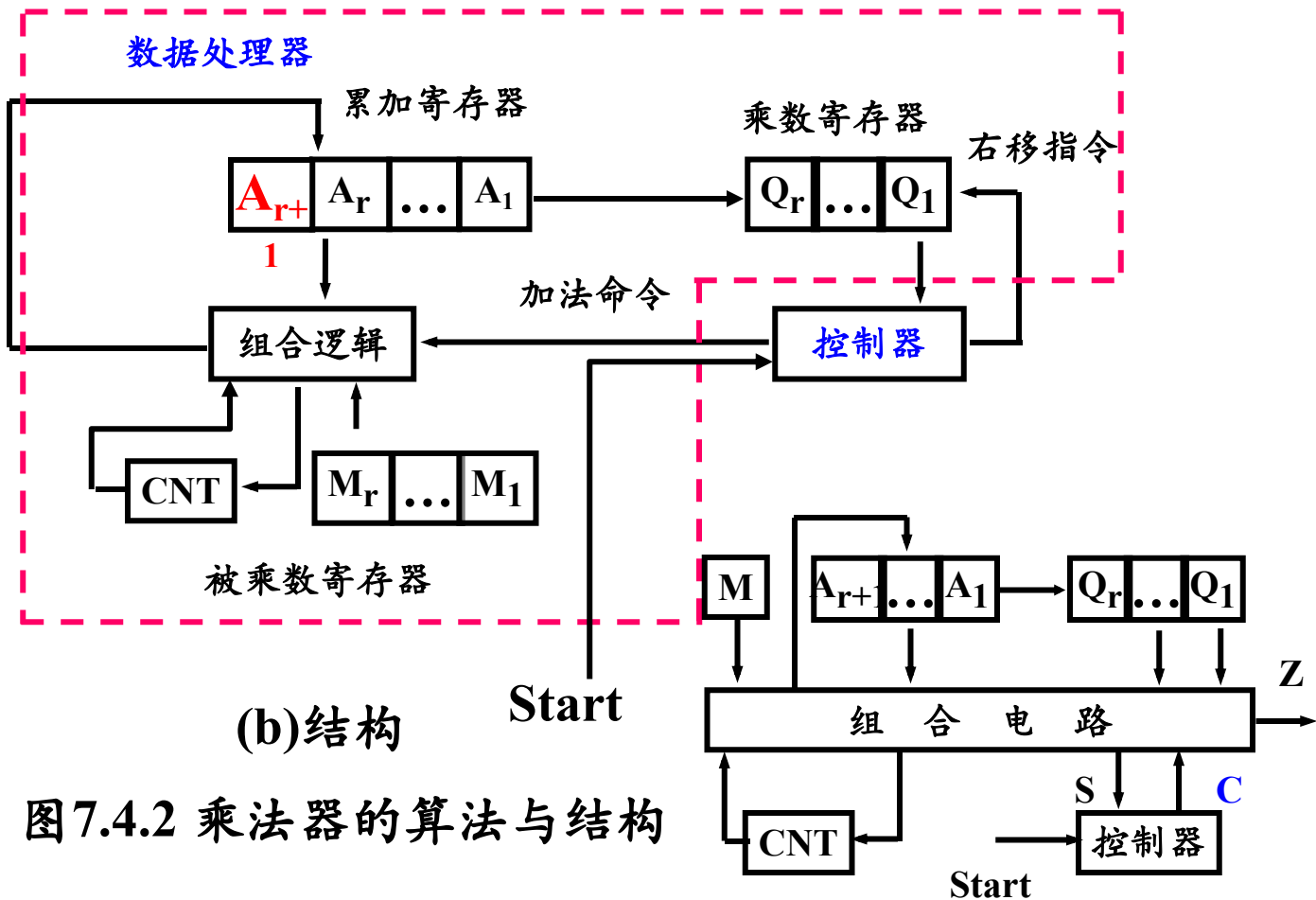
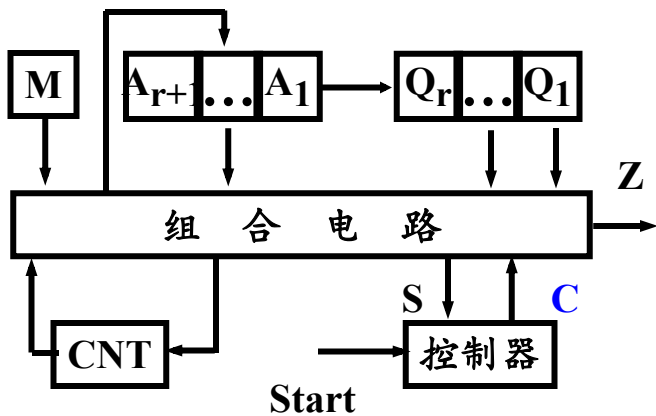
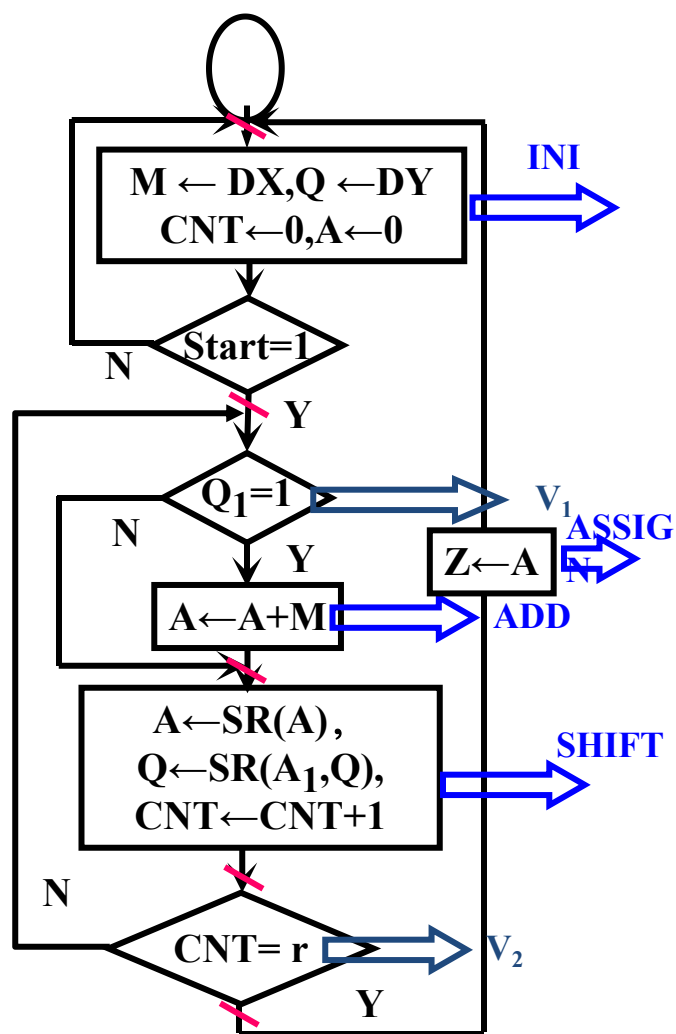
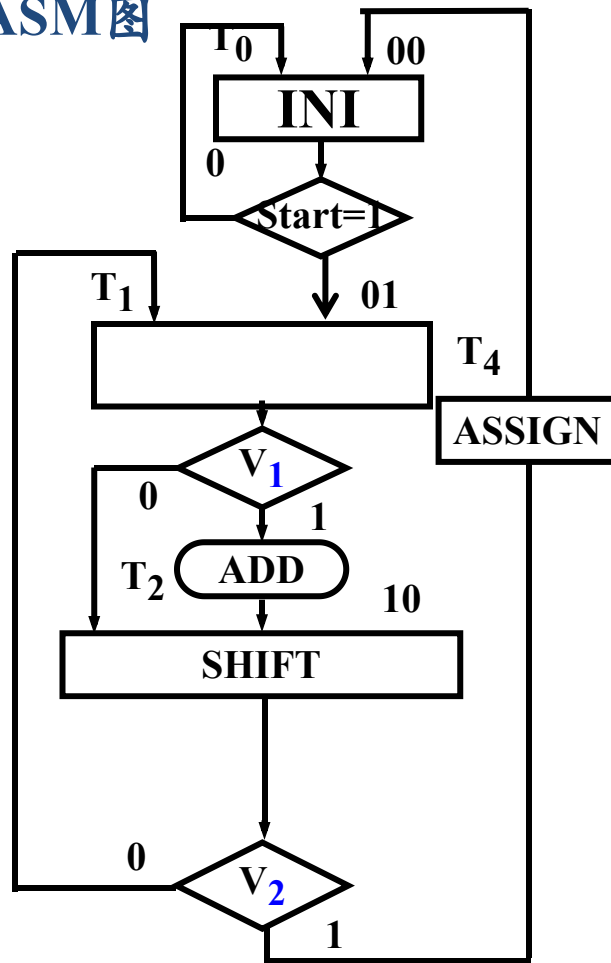
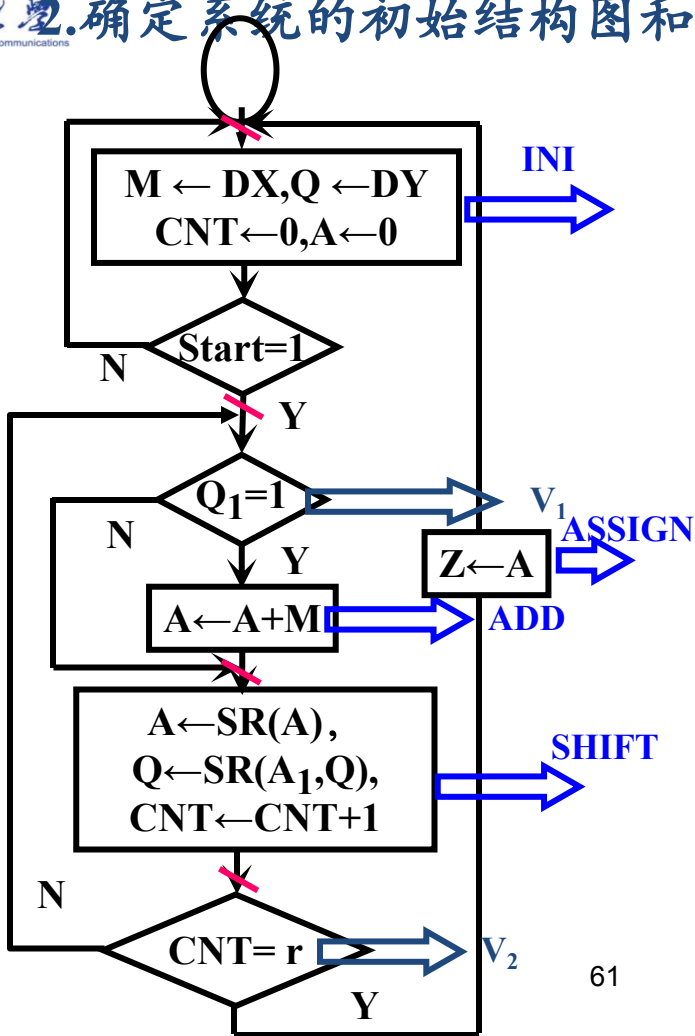


图7.4.2 乘法器的算法与结构



(b) 算法流程图





乘法器的ASM图

确定系统的初始结构图

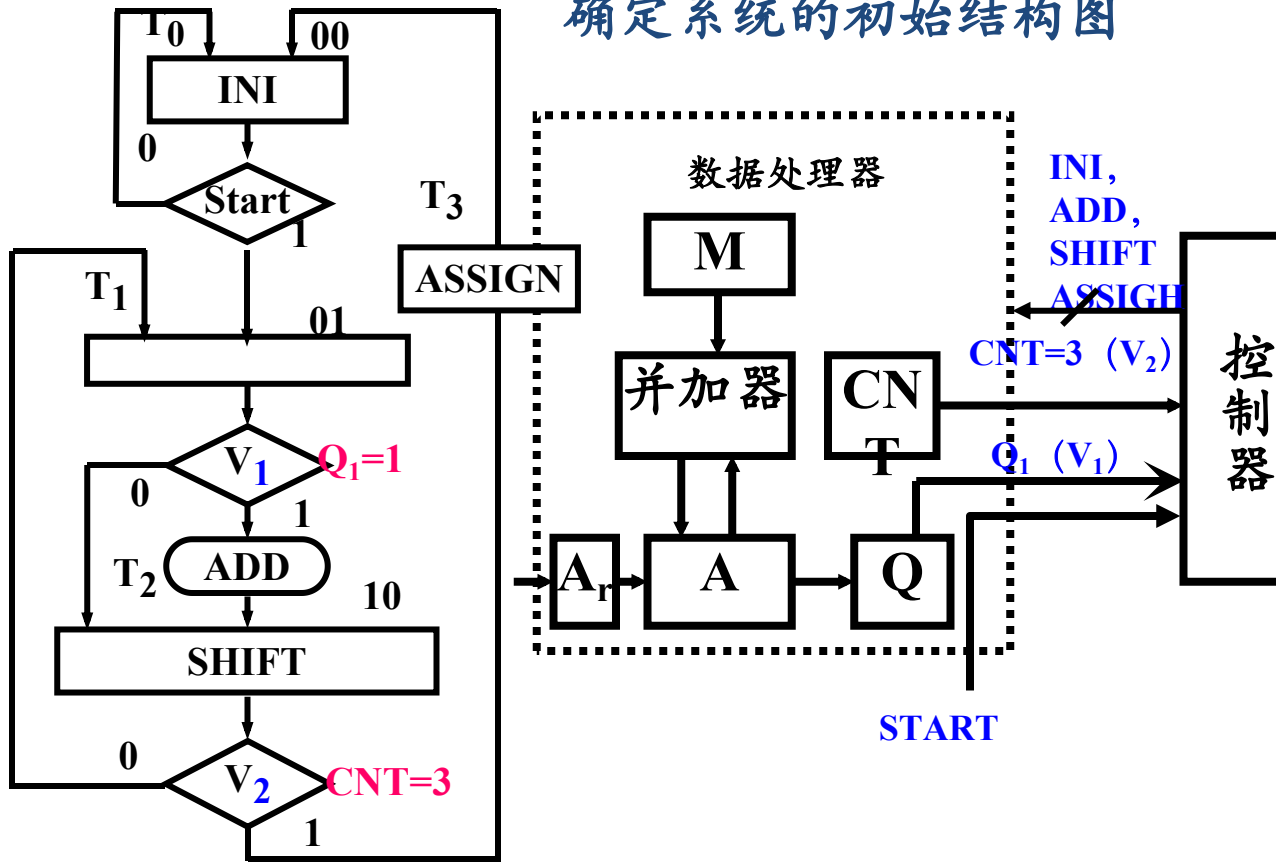
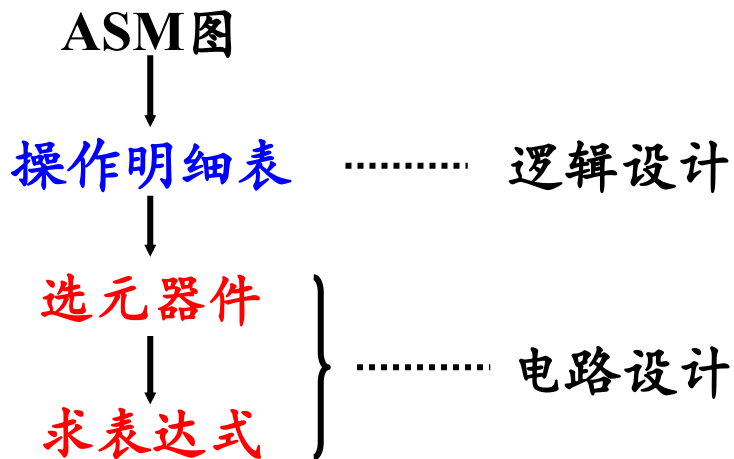


图7.4.3 乘法器的ASM图



3. 电路设计

1) 数据处理器的实现



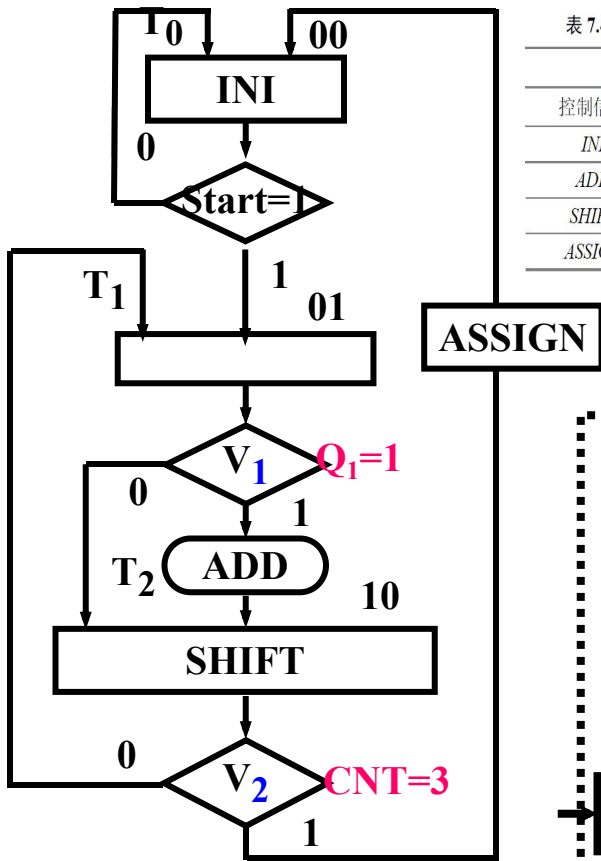


表 7.4.3

数据处理器明细表

操作表		变量表	
控制信号	操作	变量	定义
INI	$M \leftarrow DX, Q \leftarrow DY, CNT \leftarrow 0, A \leftarrow 0$	V_1	$Q_1=1$
ADD	$A \leftarrow A+M$	V_2	CNT=3
SHIFT	$A \leftarrow SR(A), Q \leftarrow SR(A_0, Q), CNT \leftarrow CNT+1$		
ASSIGN	$Z \leftarrow A$		

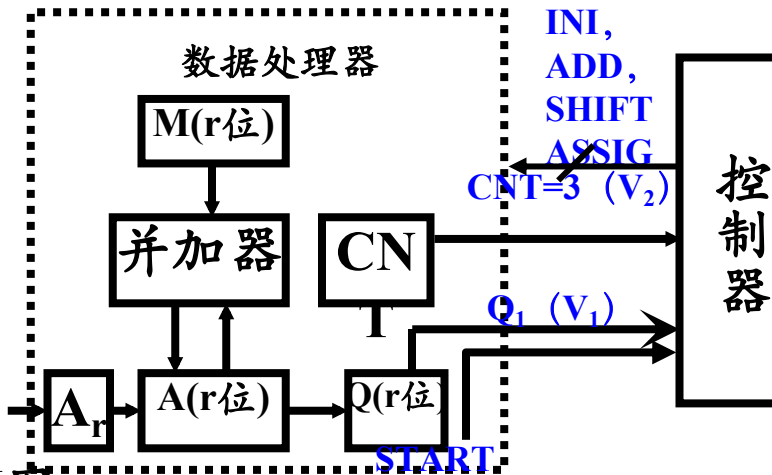


图7.4.3 乘法器的ASM图



(1) 寄存器A的实现

表 7.4.3 数据处理器明细表

操作表		变量表	
控制信号	操作	变量	定义
<i>INI</i>	$M \leftarrow DX, Q \leftarrow DY, CNT \leftarrow 0, A \leftarrow 0$	V_1	$Q_1 = 1$
<i>ADD</i>	$A \leftarrow A + M$	V_2	$CNT = 3$
<i>SHIFT</i>	$A \leftarrow SR(A), Q \leftarrow SR(A_0, Q), CNT \leftarrow CNT + 1$		
<i>ASSIGN</i>	$Z \leftarrow A$		

- 以累加寄存器A为目标的寄存器操作有清“0”，置数，右移，保持。注意清“0”是同步操作
- 选用四位移位寄存器74194实现
- 填写74194功能控制端 M_1 、 M_0 的真值表



表 7.4.4 74194功能表

M_1	M_0	功能
0	0	保持
0	1	右移
1	0	左移
1	1	置数

表 7.4.3

数据处理器明细表

操作表		变量表	
控制信号	操作	变量	定义
INI	$M \leftarrow DX, Q \leftarrow DY, CNT \leftarrow 0, A \leftarrow 0$	V_1	$Q_i = 1$
ADD	$A \leftarrow A + M$	V_2	CNT = 3
SHIFT	$A \leftarrow SR(A), Q \leftarrow SR(A_0, Q), CNT \leftarrow CNT + 1$		
ASSIGN	$Z \leftarrow A$		



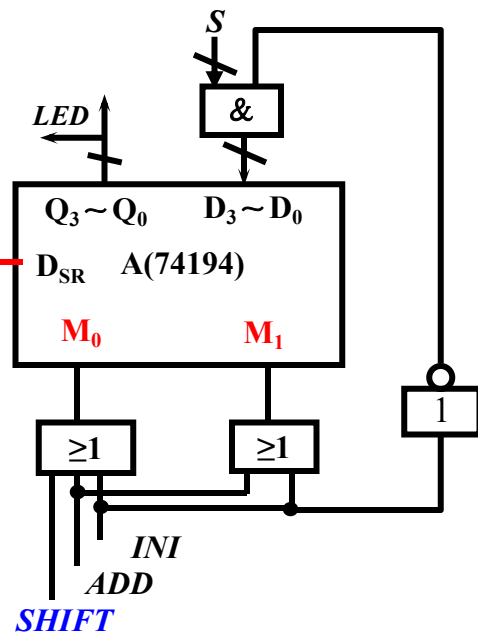
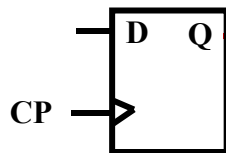
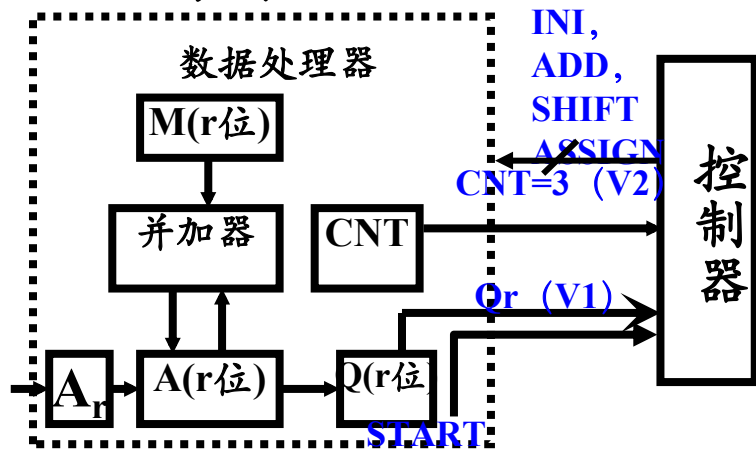
INI	ADD	SHIFT	M_1	M_0
0	0	0	0	0
1	0	0	1	1
0	1	0	1	1
0	0	1	0	1

$$M_1 = INI + ADD$$

$$M_0 = INI + ADD + SHIFT$$

$$D_i = \overline{INI} \cdot S_i$$

其中 S_i 为并行加法器输出



$$M_1 = \overline{INI} + ADD$$

$$M_0 = \overline{INI} + ADD + SHIFT$$

$$D_i = \overline{INI} \cdot S_i$$

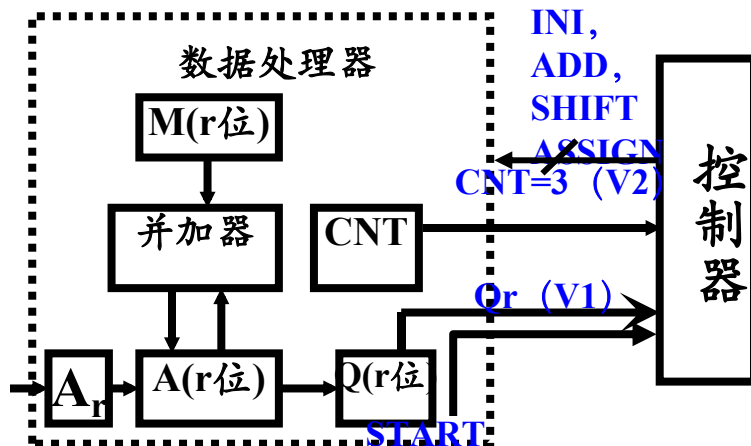


(2) 乘数寄存器Q的实现

表 7.4.3

数据处理器明细表

操作表		变量表	
控制信号	操作	变量	定义
INI	$M \leftarrow DX, Q \leftarrow DY, CNT \leftarrow 0, A \leftarrow 0$	V_1	$Q_1=1$
ADD	$A \leftarrow A+M$	V_2	CNT=3
SHIFT	$A \leftarrow SR(A), Q \leftarrow SR(A_0, Q), CNT \leftarrow CNT+1$		
ASSIGN	$Z \leftarrow A$		



❖ 以乘数寄存器Q为目标的
的操作有：移位、置数
和保持。

❖ 为减少集成电路的种
类，乘数寄存器Q也选
用74194实现



乘数寄存器Q的实现

表 7.4.3

数据处理器明细表

操作表		变量表	
控制信号	操作	变量	定义
<i>INI</i>	$M \leftarrow DX, Q \leftarrow DY, CNT \leftarrow 0, A \leftarrow 0$	V_1	$Q_1=1$
<i>ADD</i>	$A \leftarrow A+M$	V_2	$CNT=3$
<i>SHIFT</i>	$A \leftarrow SR(A), Q \leftarrow SR(A_0, Q), CNT \leftarrow CNT+1$		
<i>ASSIGN</i>	$Z \leftarrow A$		

表7.4.4 74194功能表

M_1	M_0	功能
0	0	保持
0	1	右移
1	0	左移
1	1	置数

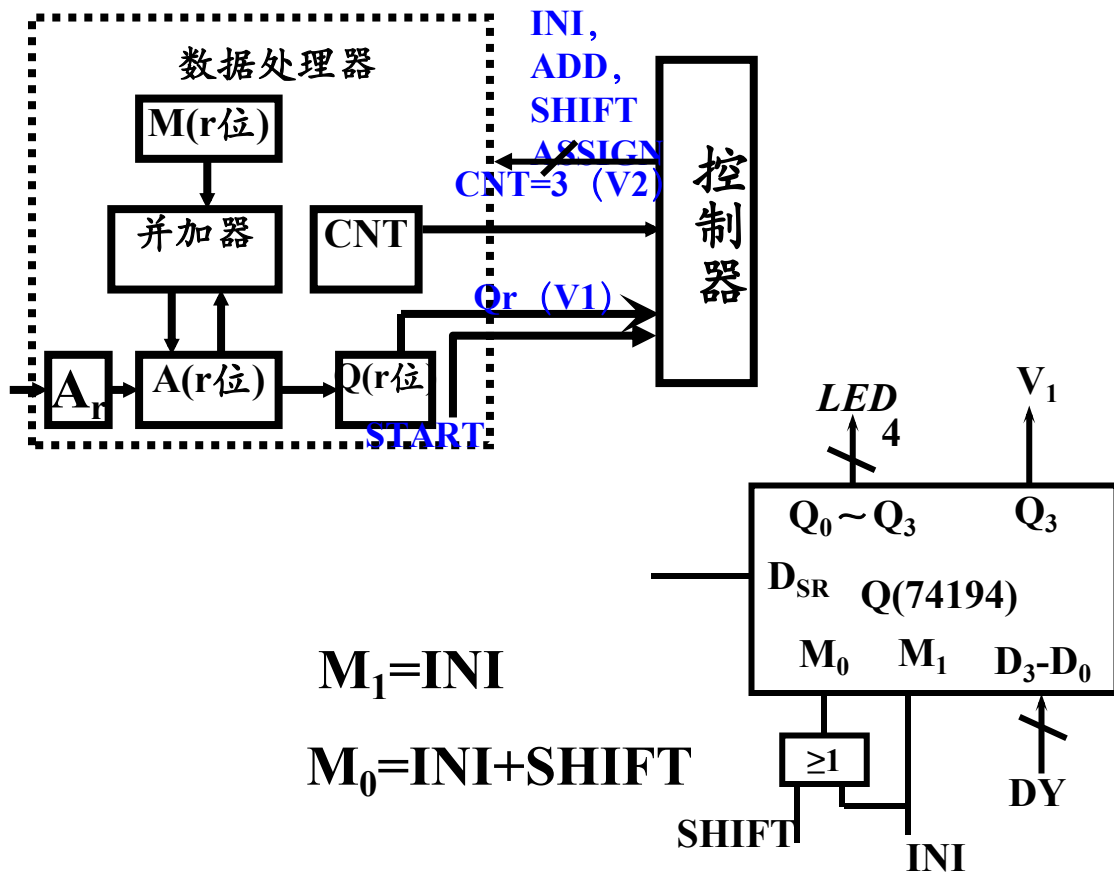
INI	SHIFT	M_1	M_0
0	0	0	0
1	0	1	1
0	1	0	1

$$M_1 = INI$$

$$M_0 = INI + SHIFT$$



乘数寄存器Q的电路实现





(3) 被乘数计数器M的实现

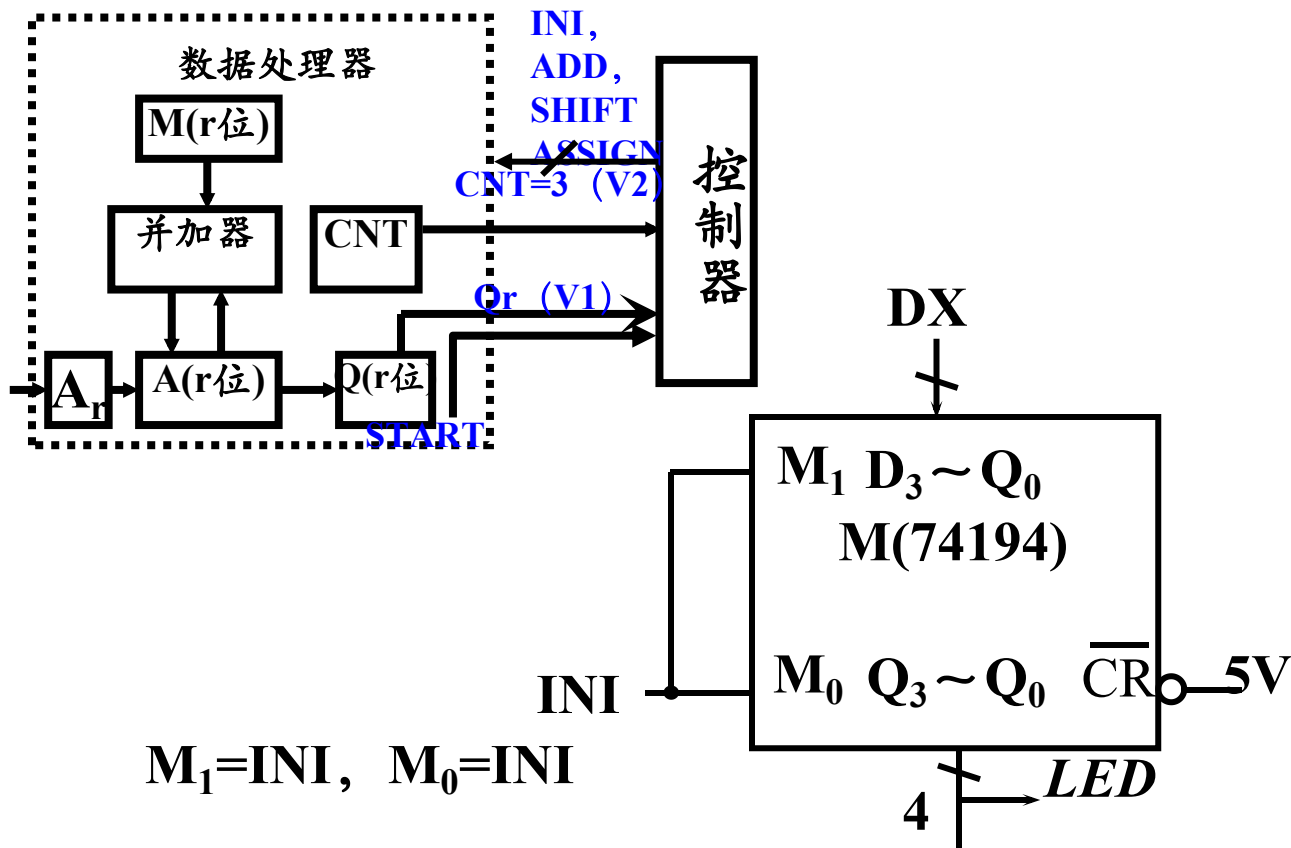
表 7.4.3 数据处理器明细表

操作表		变量表	
控制信号	操作	变量	定义
<i>INI</i>	$M \leftarrow DX, Q \leftarrow DY, CNT \leftarrow 0, A \leftarrow 0$	V_1	$Q_1 = 1$
<i>ADD</i>	$A \leftarrow A + M$	V_2	$CNT = 3$
<i>SHIFT</i>	$A \leftarrow SR(A), Q \leftarrow SR(A_0, Q), CNT \leftarrow CNT + 1$		
<i>ASSIGN</i>	$Z \leftarrow A$		

- 没有任何操作，只是置数和保持功能，INI有效时，置数 $M_1M_0=11$ ，其余时间保持， $M_1M_0=00$ 。
- 因此 $M_1=INI, M_0=INI$



被乘数寄存器M的电路实现



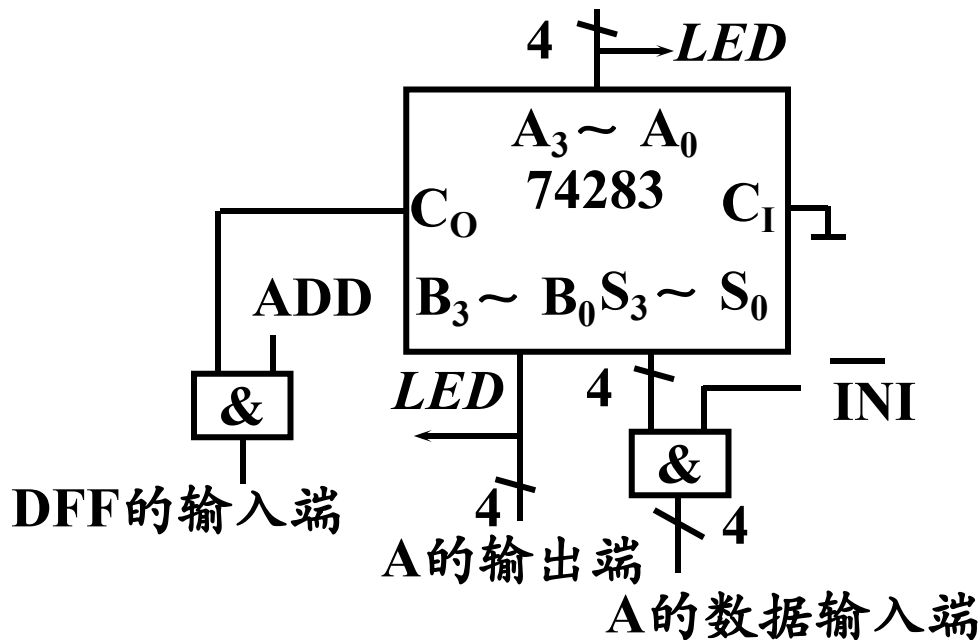


(4) 加法器的实现

表 7.4.3

数据处理器明细表

操作表		变量表	
控制信号	操作	变量	定义
INI	$M \leftarrow DX, Q \leftarrow DY, CNT \leftarrow 0, A \leftarrow 0$	V_1	$Q_1=1$
ADD	$A \leftarrow A+M$	V_2	$CNT=3$
SHIFT	$A \leftarrow SR(A), Q \leftarrow SR(A_0, Q), CNT \leftarrow CNT+1$		
ASSIGN	$Z \leftarrow A$		





加法器的实现说明

- 用74283来实现。
- 被加数 $A_4A_3A_2A_1$ 接被乘数寄存器M输出，加数 $B_4B_3B_2B_1$ 接累加器A输出。输出 $S_4S_3S_2S_1$ 送累加器A的数据端。
- 由于A在INI信号到时，需要置0，所以将 $S_4S_3S_2S_1$ 先与INI的非相与，再送至A的数据端。
- 74283的CO输出应送DFF的D端，但送时受ADD信号控制，所以应先与ADD信号相与再送给D。



(5) 计数器CNT的实现

表 7.4.3

数据处理器明细表

操作表		变量表	
控制信号	操作	变量	定义
INI	$M \leftarrow DX, Q \leftarrow DY, CNT \leftarrow 0, A \leftarrow 0$	V_1	$Q_1=1$
ADD	$A \leftarrow A+M$	V_2	$CNT=3$
SHIFT	$A \leftarrow SR(A), Q \leftarrow SR(A_0, Q), CNT \leftarrow CNT+1$		
ASSIGN	$Z \leftarrow A$		

- CNT的操作：增1和同步清零。所以采用四位二进制同步计数器74163来实现

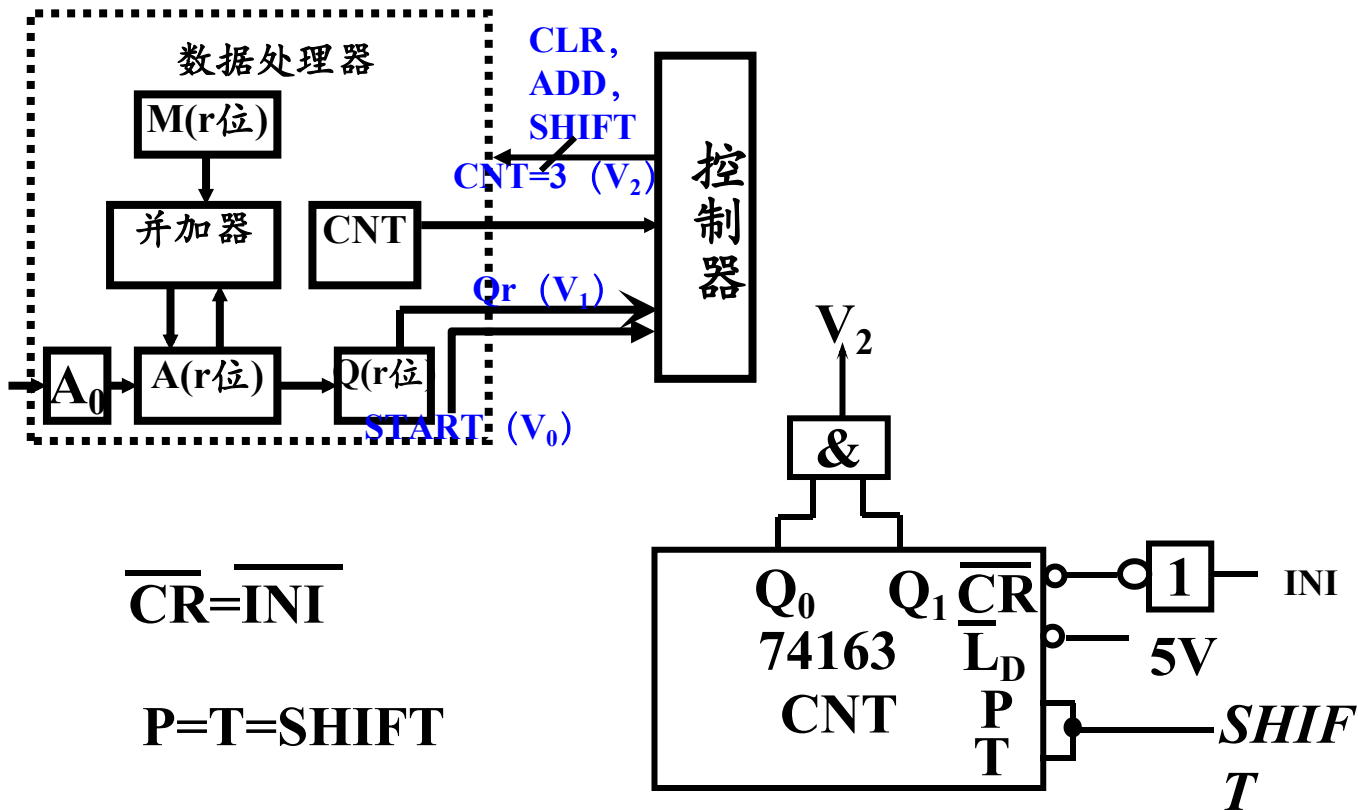
CR	L_D	P	T	CP	功能
0	\emptyset	\emptyset	\emptyset	\uparrow	清0
1	0	\emptyset	\emptyset	\uparrow	并入
1	1	0	\emptyset	\emptyset	保持
1	1	\emptyset	0	\emptyset	保持
1	1	1	1	\uparrow	计数

$$\overline{CR} = \overline{INI}$$

$$P = T = \text{SHIFT}$$

表 7.4.7 74163功能表

计数器CNT的实现电路



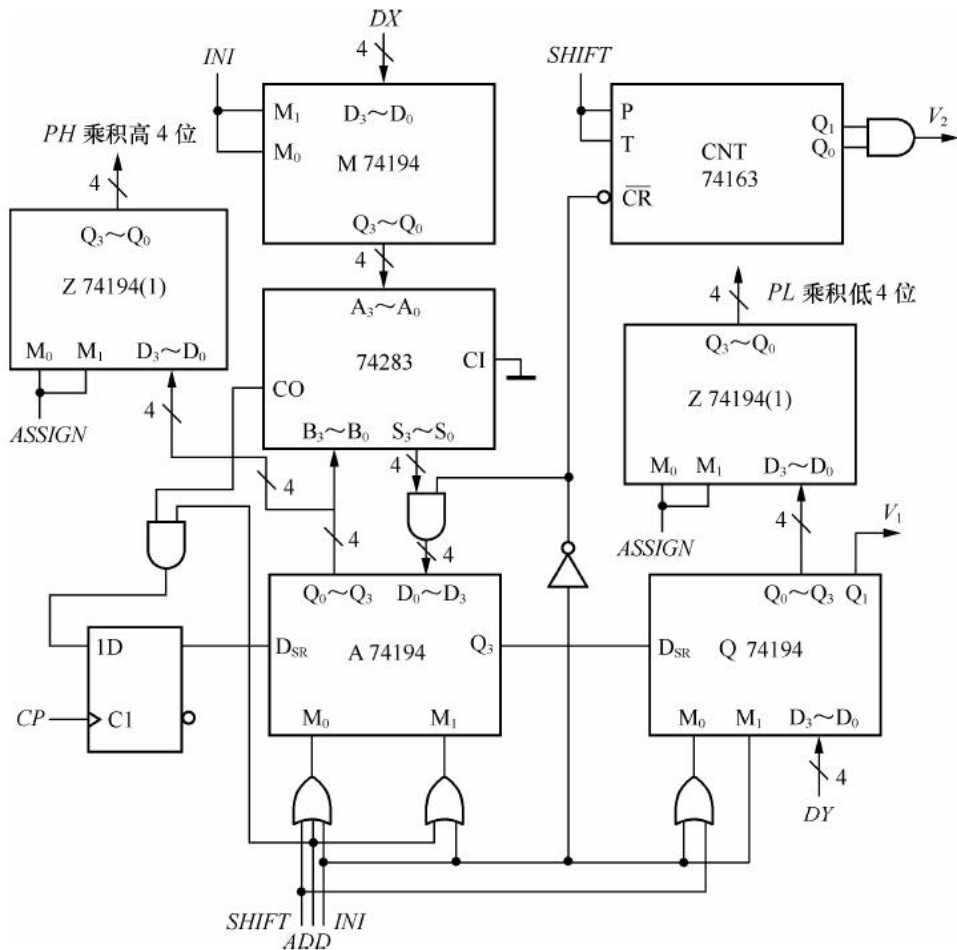


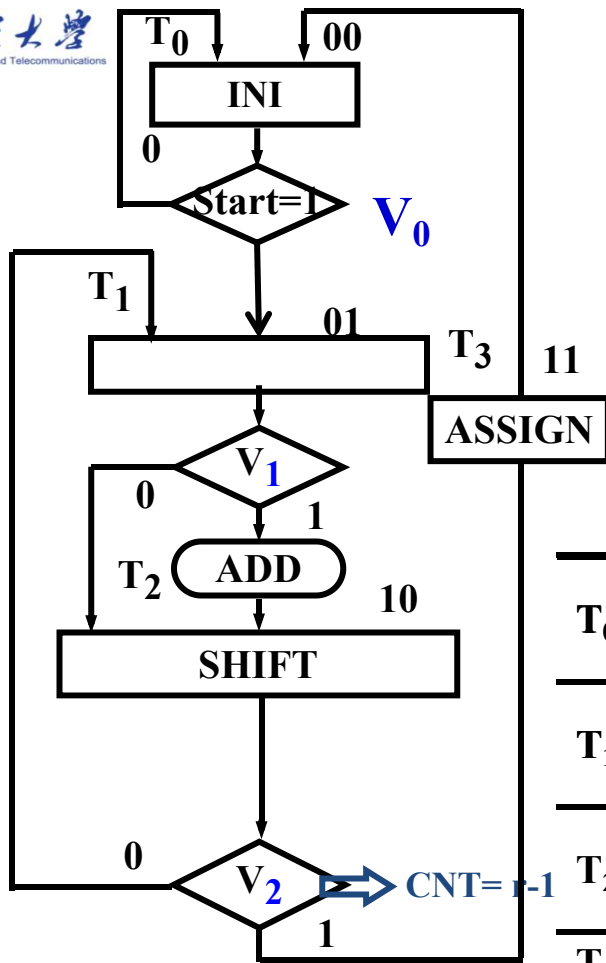
图 7.4.4 乘法器的数据处理器逻辑框图



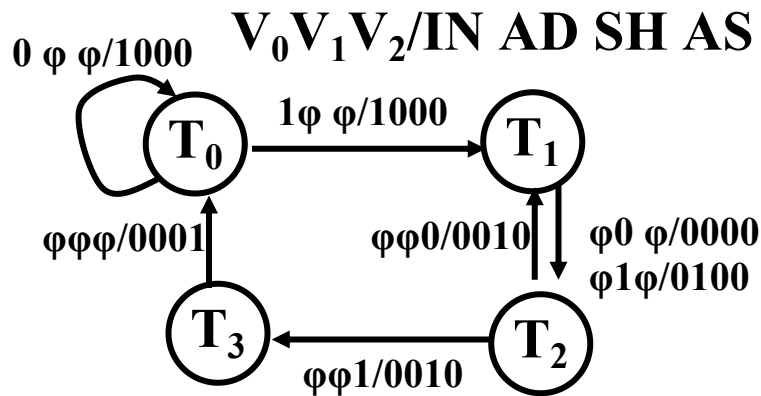
2) 控制器的实现

(1) 传统设计方法

该控制器有4个状态 T_0 、 T_1 、 T_2 、 T_3 ，所以必须选用2个DFF触发器 Q_2Q_1 ，设编码分别为00, 01, 10, 11（标注在ASM图上）。另外有3个输入条件 V_0, V_1, V_2 。所以可得乘法器控制器状态转移图为：



乘法器的ASM图



	现 态		输 入			次 态		输 出
	Q ₁	Q ₀	V ₀	V ₁	V ₂	Q ₁	Q ₀	
T ₀	0	0	0	∅	∅	0	0	INI
	0	0	1	∅	∅	0	1	
T ₁	0	1	∅	0	∅	1	0	ADD
	0	1	∅	1	∅	1	0	
T ₂	1	0	∅	∅	0	0	1	SHIFT
	1	0	∅	∅	1	1	1	
T ₃	1	1	∅	∅	∅	0	0	ASSIN

乘法器的
状态转移、
输出表

	现 态		输 入			次 态		输 出
	Q ₁	Q ₀	V ₀	V ₁	V ₂	Q ₁	Q ₀	
T ₀	0	0	0	∅	∅	0	0	INI
	0	0	1	∅	∅	0	1	
T ₁	0	1	∅	0	∅	1	0	ADD
	0	1	∅	1	∅	1	0	
T ₂	1	0	∅	∅	0	0	1	SHIFT
	1	0	∅	∅	1	1	1	
T ₃	1	1	∅	∅	∅	0	0	ASSIGN



	Q ₀	1
Q ₁	0	1
	1	V ₂

(a) D₁

	Q ₀	0	1
Q ₁	0	V ₀	0
	1	1	0

(b) D₀

$$D_1 = \bar{Q}_1 Q_0 + Q_1 \bar{Q}_0 V_2$$

$$D_0 = \bar{Q}_1 \bar{Q}_0 V_0 + Q_1 \bar{Q}_0$$



乘法器的状态
转移、输出表

	现 态		输 入			次 态		输 出
	Q ₁	Q ₀	V ₀	V ₁	V ₂	Q ₁	Q ₀	
T ₀	0	0	0	∅	∅	0	0	INI
	0	0	1	∅	∅	0	1	
T ₁	0	1	∅	0	∅	1	0	ADD
	0	1	∅	1	∅	1	0	
T ₂	1	0	∅	∅	0	0	1	SHIFT
	1	0	∅	∅	1	1	1	
T ₃	1	1	∅	∅	∅	0	0	ASSIGN



$$INT = \bar{Q}_1^n \bar{Q}_0^n$$

$$SHIFT = Q_1^n \bar{Q}_0^n$$

$$ADD = \bar{Q}_1^n Q_0^n V_1$$

$$ASSIGN = Q_1^n Q_0^n$$

(2) 用MSI电路设计

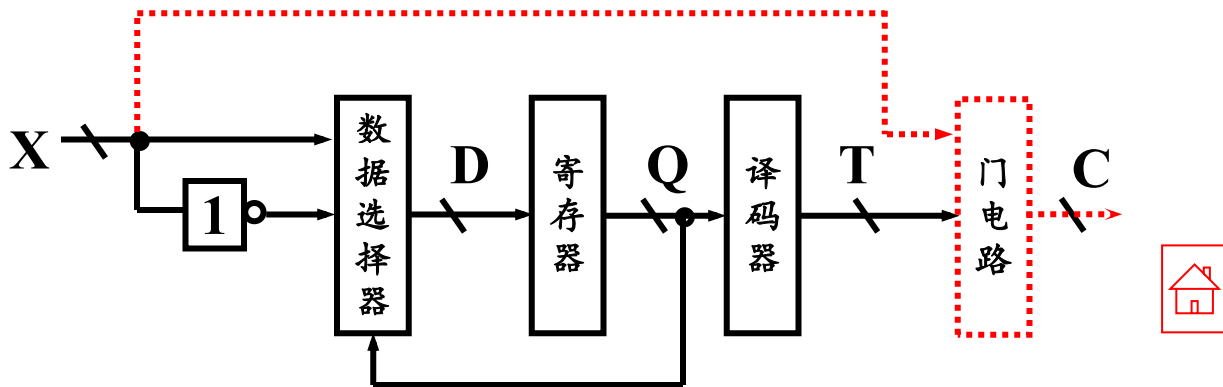


图 7.3.3 利用数据选择器和译码器的控制逻辑框图



	Q_0	0	1
Q_1	0	0	1
1	V_2	V_0	0

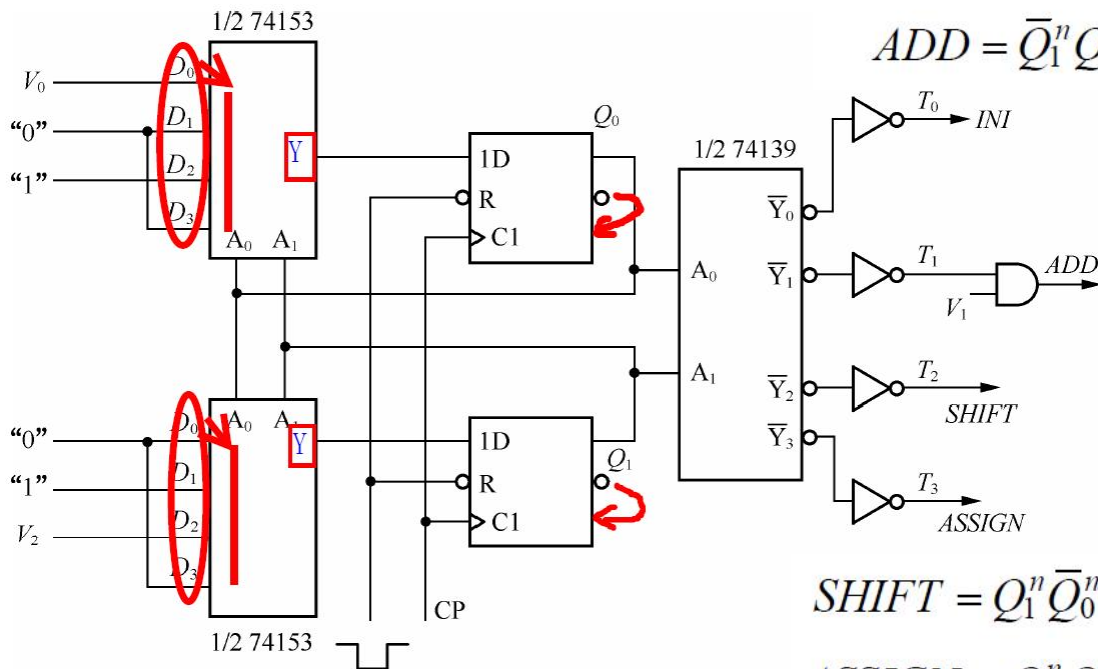
	Q_0	0	1
Q_1	0	V_0	0
1	1	1	0

$$D_1 = \bar{Q}_1 Q_0 + Q_1 \bar{Q}_0 V_2$$

$$D_0 = \bar{Q}_1 \bar{Q}_0 V_0 + Q_1 \bar{Q}_0$$

$$INT = \bar{Q}_1^n \bar{Q}_0^n$$

$$ADD = \bar{Q}_1^n Q_0^n V_1$$



$$SHIFT = Q_1^n \bar{Q}_0^n$$

$$ASSIGN = Q_1^n Q_0^n$$

(3)用每态一个触发器的方法 (用SSI)

控制器有多少状态就有多少触发器，每一个状态对应一个触发器，某一触发器出1表示进入该状态，相当于4个状态分别编码为0001，0010，0100，1000。

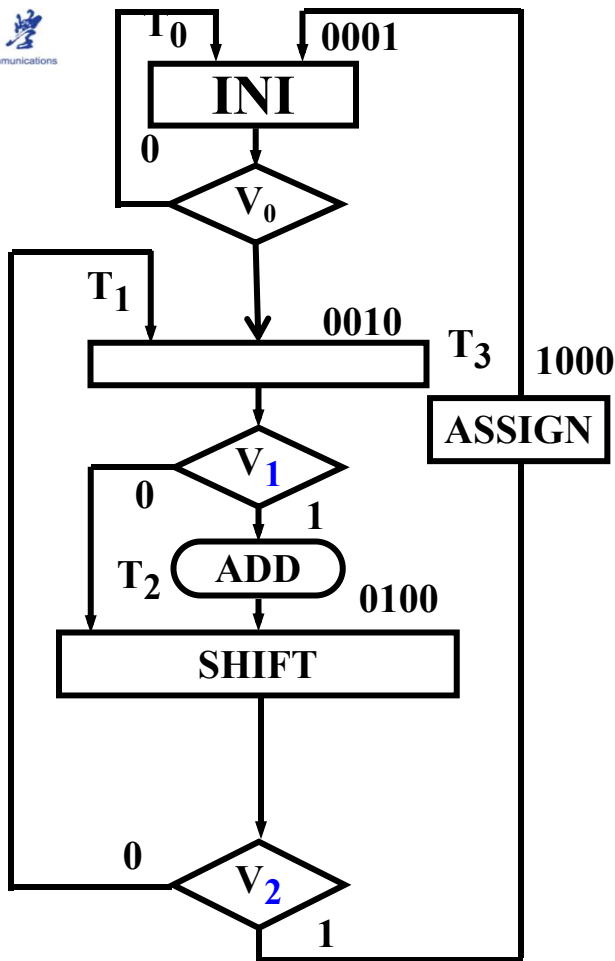


优点：

a、无须分配状态。

b、控制器的逻辑图易于读懂，调试维护方便，只要根据哪个触发器输出1，就知道进入哪个状态。

c、不用列状态转移表，直接根据ASM图求得触发器的激励函数；



乘法器的ASM图

$$D_0 = T_0 \bar{V}_0 + T_3$$

$$D_1 = T_0 V_0 + T_2 \bar{V}_2;$$

$$D_2 = T_1 \bar{V}_1 + T_1 V_1 = T_1$$

$$D_3 = T_2 V_2$$

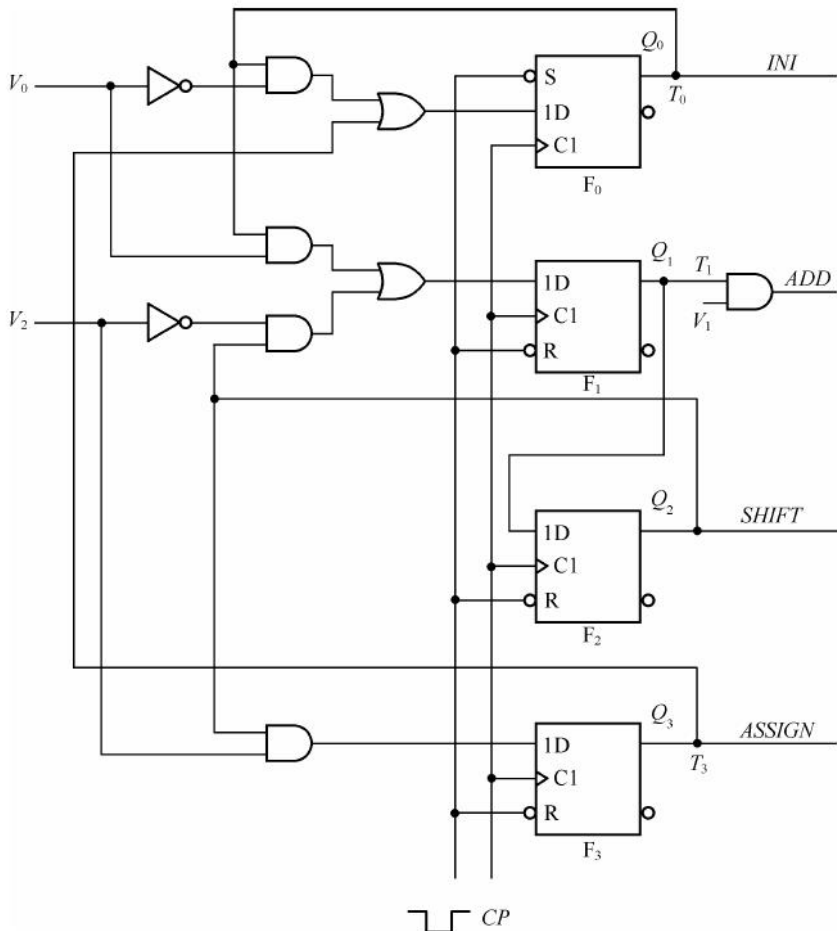
$$INI = T_0$$

$$ADD = T_1 V_1$$

$$SHIFT = T_2$$

$$ASSIGN = T_3$$





$$D_0 = T_0 \bar{V}_0 + T_3$$

$$D_1 = T_0 V_0 + T_2 \bar{V}_2;$$

$$D_2 = T_1 \bar{V}_1 + T_1 V_1 = T_1$$

$$D_3 = T_2 V_2$$

$$INI = T_0$$

$$ADD = T_1 V_1$$

$$SHIFT = T_2$$


$$ASSIGN = T_3$$

图 7.3.5 用每态一个触发器实现的控制器逻辑图

例12.5.1：设计一个交通灯管理系统。其功能如下：

(1)乡间公路上无车时，主干道绿灯亮，乡间公路红灯亮；

(2)乡间公路上有车时，传感器输出 $C=1$ ，且主干道通车时间超过最短时间，主干道交通灯由绿 \rightarrow 黄 \rightarrow 红，乡间公路交通灯由红 \rightarrow 绿；

(3)乡间公路上无车，或有车，且乡间公路通车时间超过最长时间，则主干道交通灯由红 \rightarrow 绿，公路交通灯由绿 \rightarrow 黄 \rightarrow 红；

(4)假设乡间公路绿灯亮的最长时间等于主干道绿灯亮的最短时间，都为16秒，若计时到 $E=1$ ；黄灯亮的时间设为4秒，若计时到 $F=1$ 。当启动信号 $S=1$ 时，定时器开始计时。

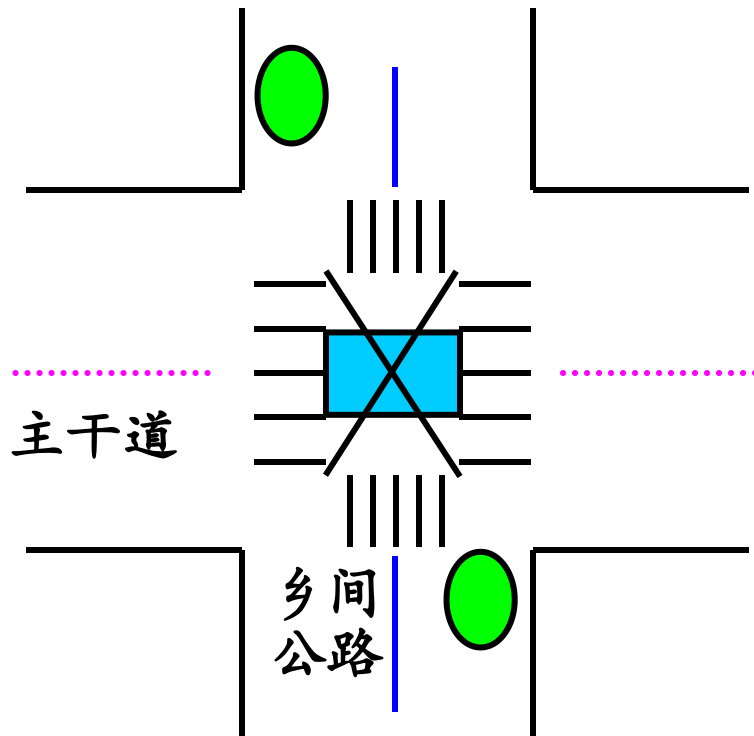


图12.5.1 十字路口交通灯和传感器示意图

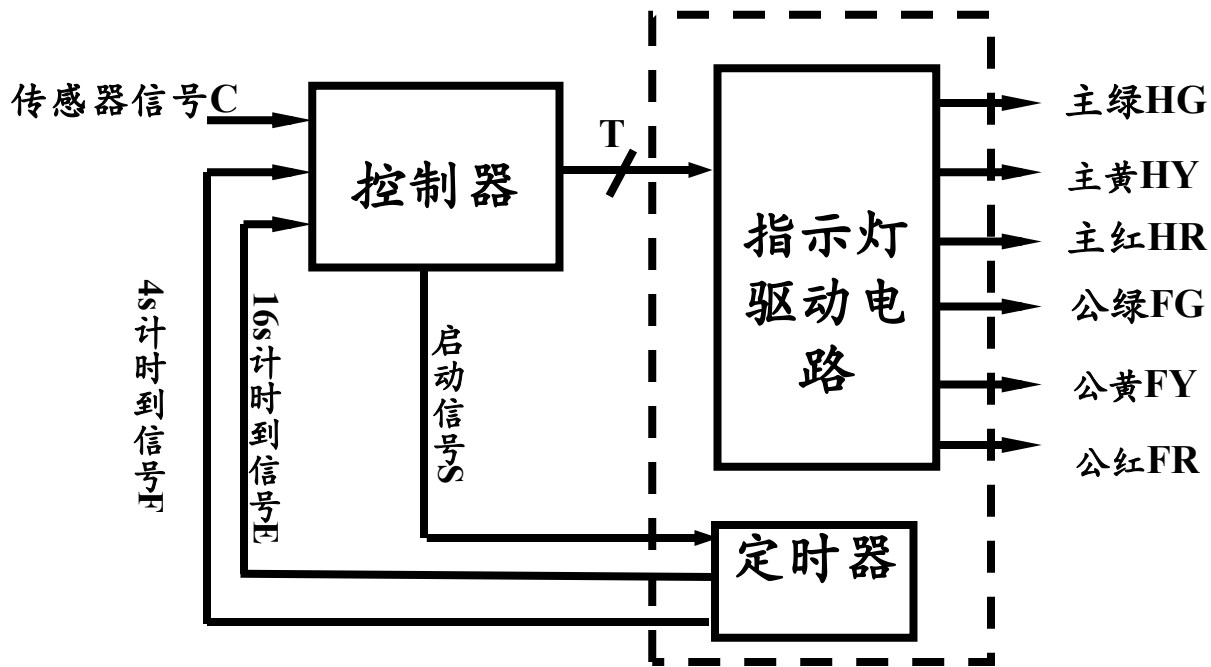


图12.5.2 系统初始结构框图

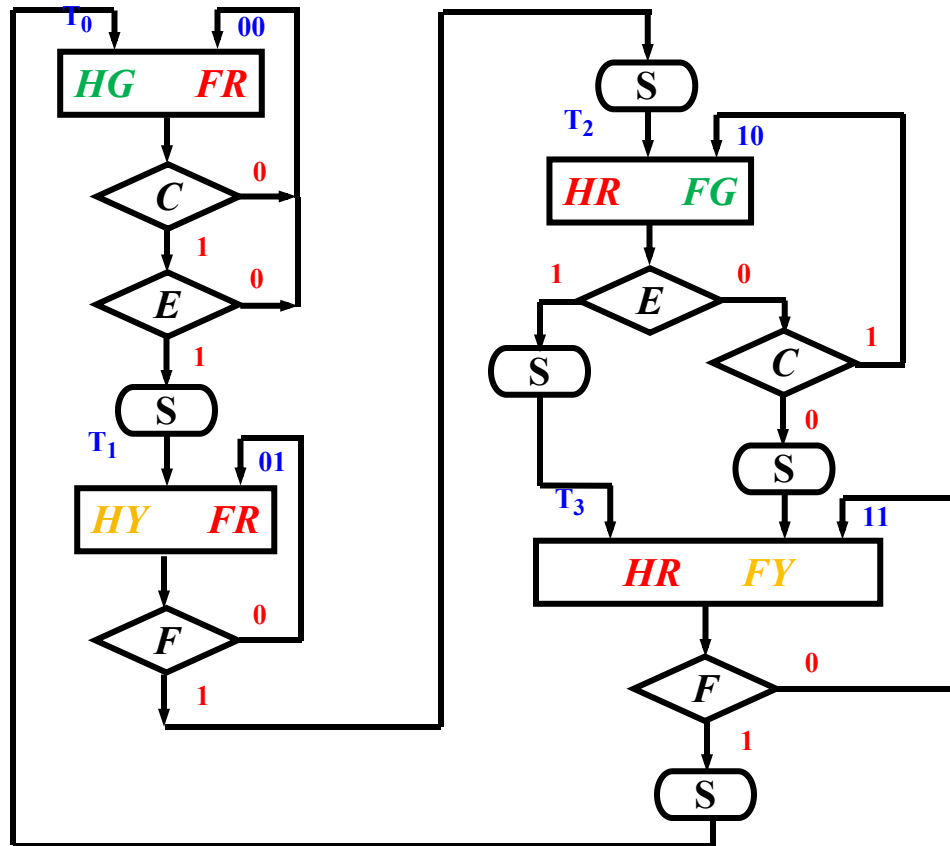


图12.5.3 交通灯管理系统ASM图

处理器设计

- 根据ASM图列出处理器明细表，如表12.5.1。

操作表		状态变量表	
控制信号	操作	状态变量	定义
T ₀	HG=1,FR=1	E F	E=1
T ₁	HY=1,FR=1		F=1
T ₂	HR=1,FG=1		
T ₃	HR=1,FY=1		
S	启动定时器		



根据处理器明细表可知指示灯驱动电路为一组合电路，可根据明细表列出其真值表：

输入	输出					
	HG	HY	HR	FG	FY	FR
T_0	1	0	0	0	0	1
T_1	0	1	0	0	0	1
T_2	0	0	1	1	0	0
T_3	0	0	1	0	1	0

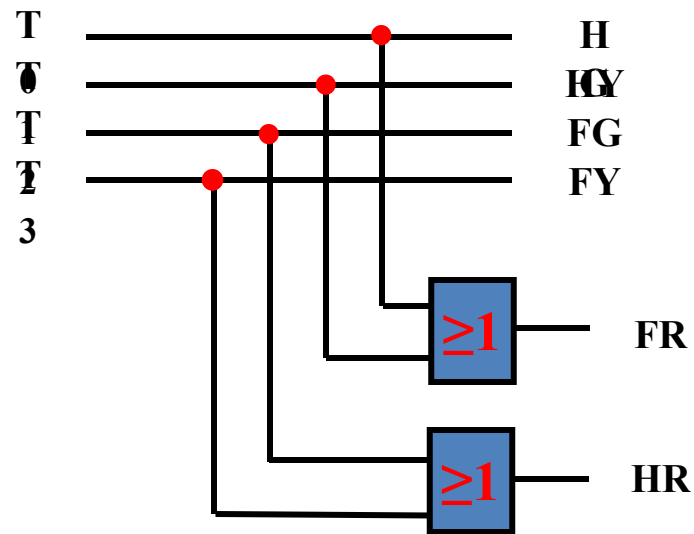


图12.5.4 指示灯电路图

定时电路设计

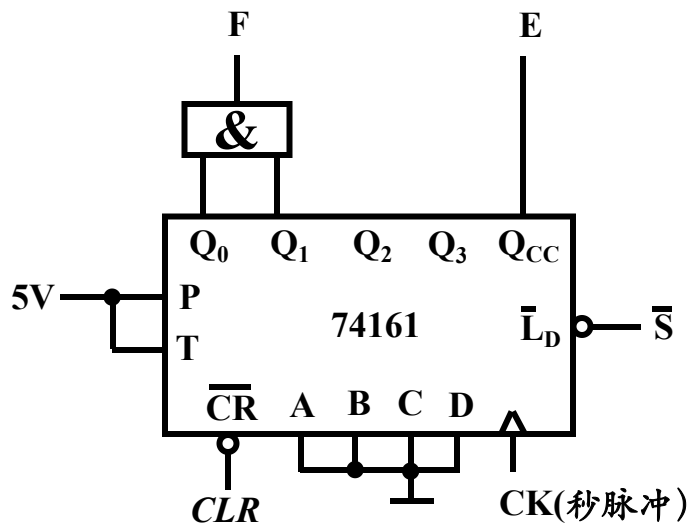


图2 定时电路图

S是启动定时器的信号，E、F为定时器的输出信号，标志16S和4S的计时到

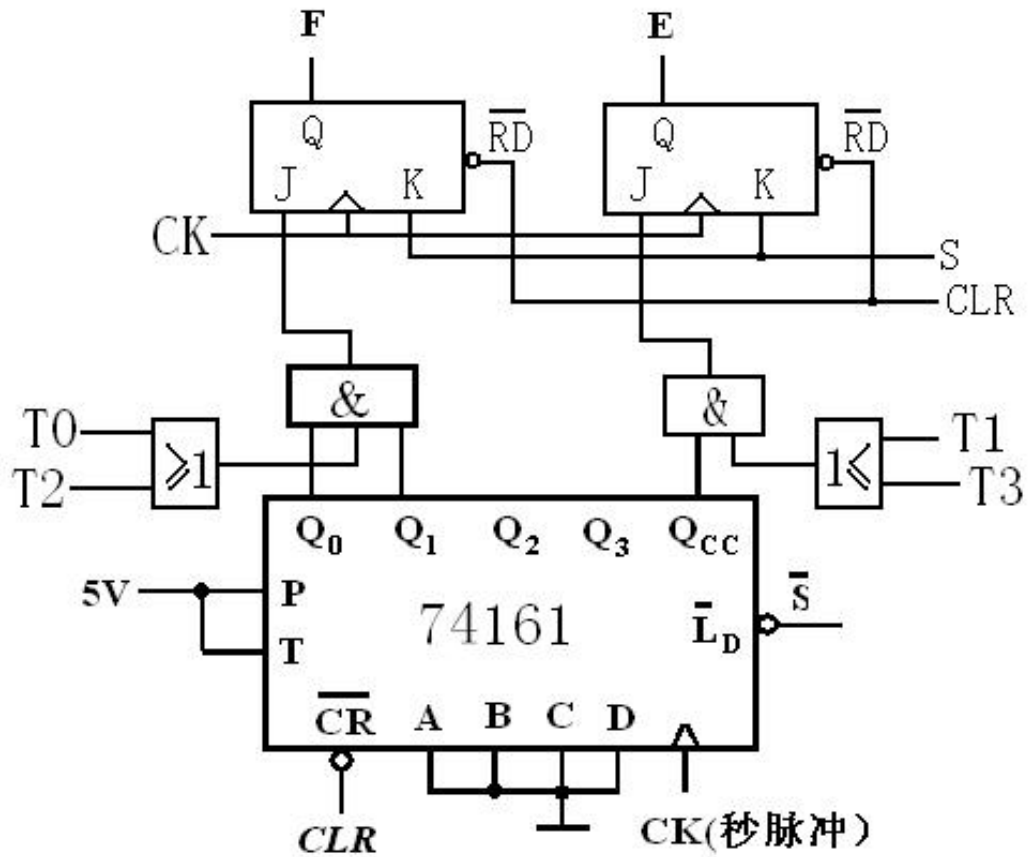


图12.5.5 定时器电路逻辑图

控制器的设计

根据每态一个触发器的方法实现控制器。由4个DFF对4个状态T0、T1、T2、T3进行编码。根据ASM图直接推导激励函数为：

$$D_0 = T_0(\bar{C} + \bar{E}) + T_3F$$

$$D_1 = T_0CE + T_1\bar{F}$$

$$D_2 = T_1F + T_2\bar{E}C$$

$$D_3 = T_2(E + \bar{C}) + T_3\bar{F}$$



根据ASM图，产生S信号的条件为：

$$S = ECT_0 + FT_1 + (E + \bar{C})T_2 + FT_3$$

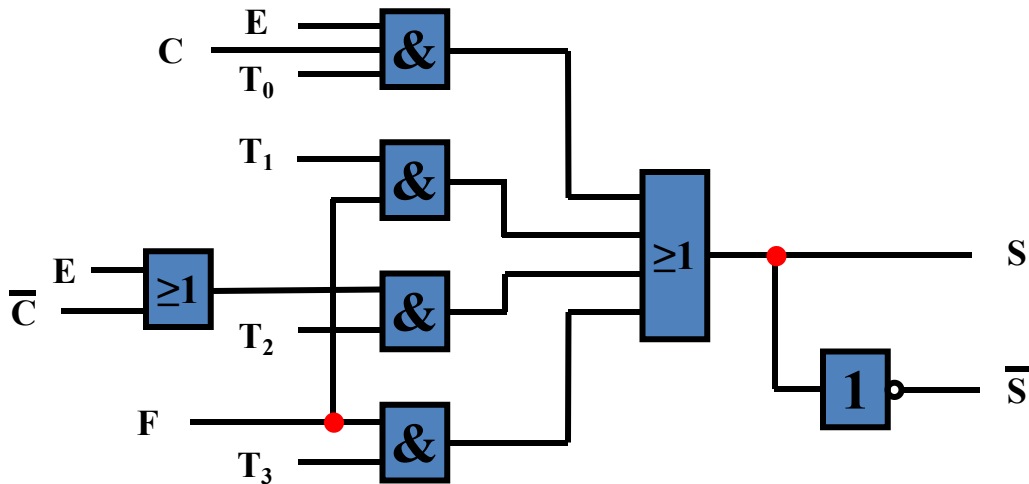


图5 S信号产生电路

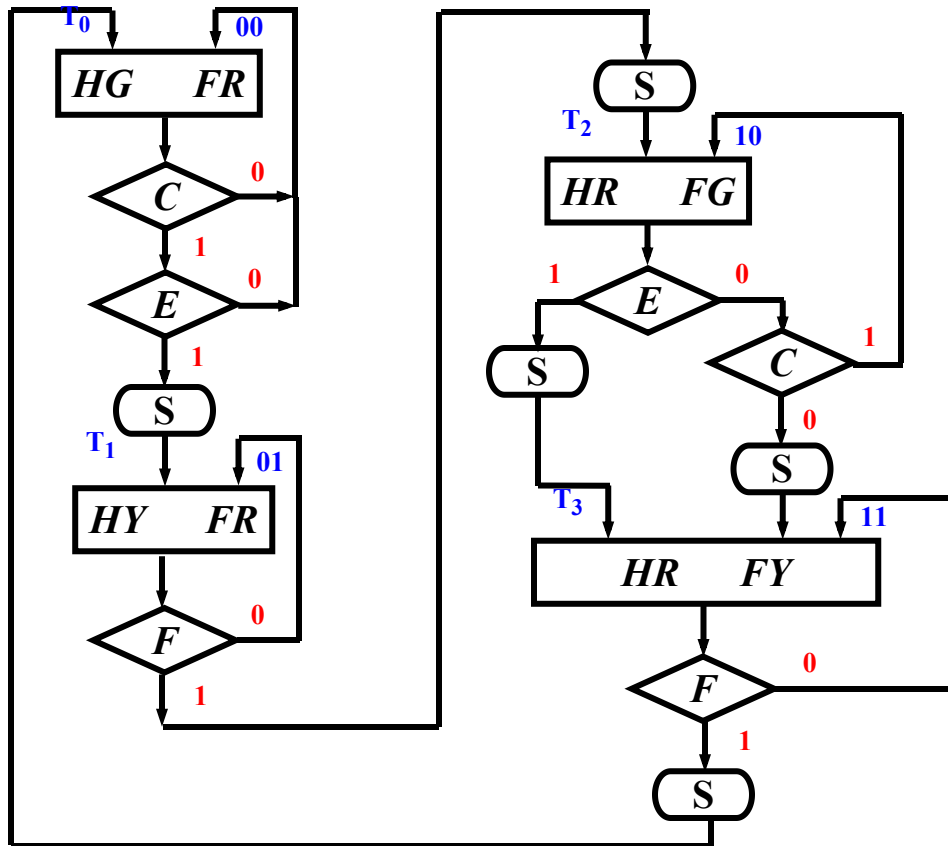
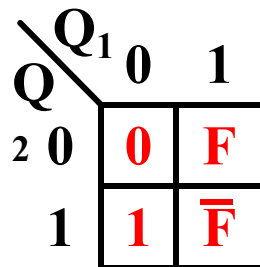


图12.5.3 交通灯管理系统ASM图

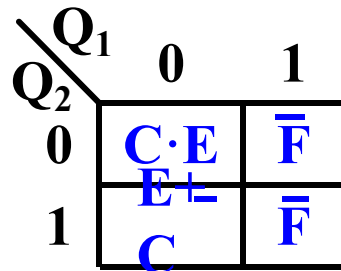
用数据选择器、时序寄存器、译码器的方法实现

控制器状态转移表

	现 态		次 态		转 换 条 件
	Q_2	Q_1	Q_2	Q_1	
T ₀	0	0	0	0	$\overline{C \cdot E}$
	0	0	0	1	$C \cdot E$
T ₁	0	1	0	1	\overline{F}
	0	1	1	0	F
T ₂	1	0	1	0	$\overline{E} \cdot C$
	1	0	1	1	$E + \overline{C}$
T ₃	1	1	1	1	\overline{F}
	1	1	0	0	F



(a) Q_2 次态图



(b) Q_1 次态图

图3 次态图

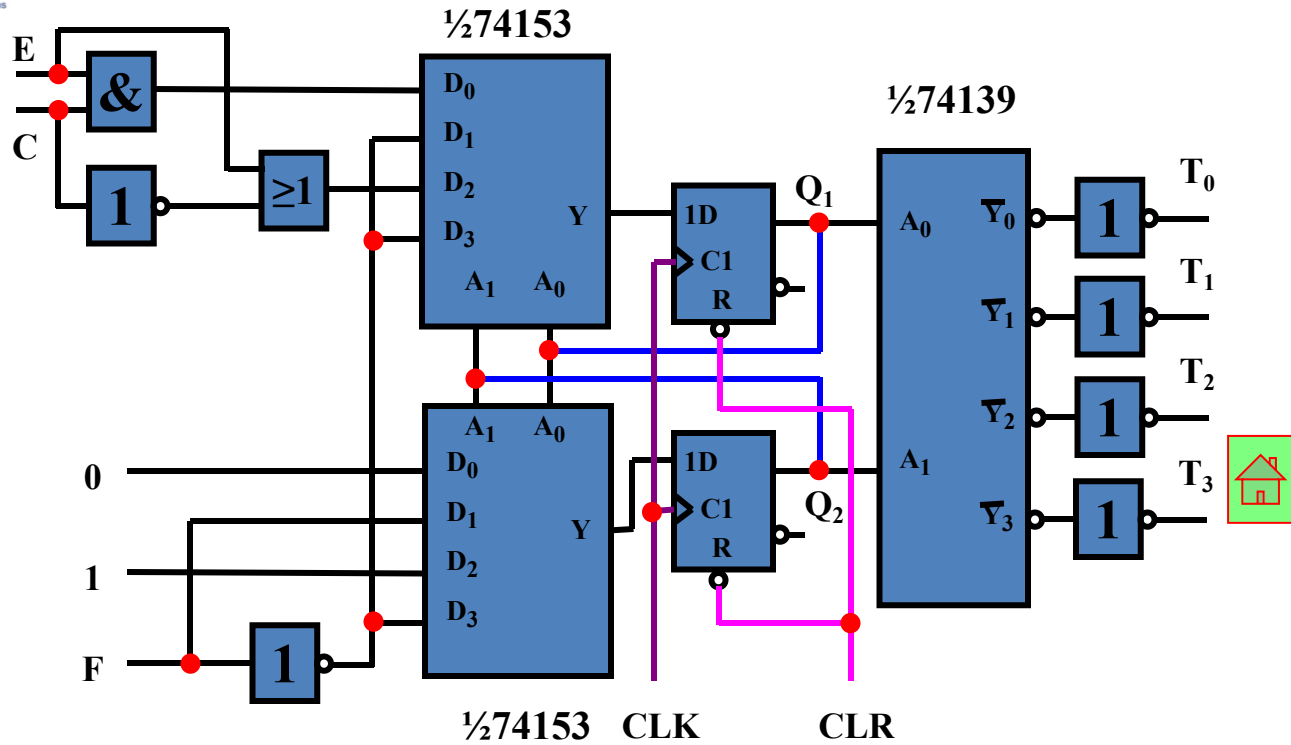


图4 控制器逻辑图



作业

7.1

7.4

7.5

7.9



★ 用PLD器件实现系统

使用PLD器件的优点

- (1)减小系统的硬件规模；
- (2)提高系统的可靠性；
- (3)提高系统的工作速度；
- (4)提高系统的灵活性；
- (5)缩短设计周期；
- (6)降低设计成本；
- (7)增加系统的保密性能。





一、系统级设计

1. 系统初始结构图

2. 导出系统ASM图

二、子系统级、部件级、元件级设计

根据ASM图，列出状态转移表，书写源程序；
用71163实现定时器。



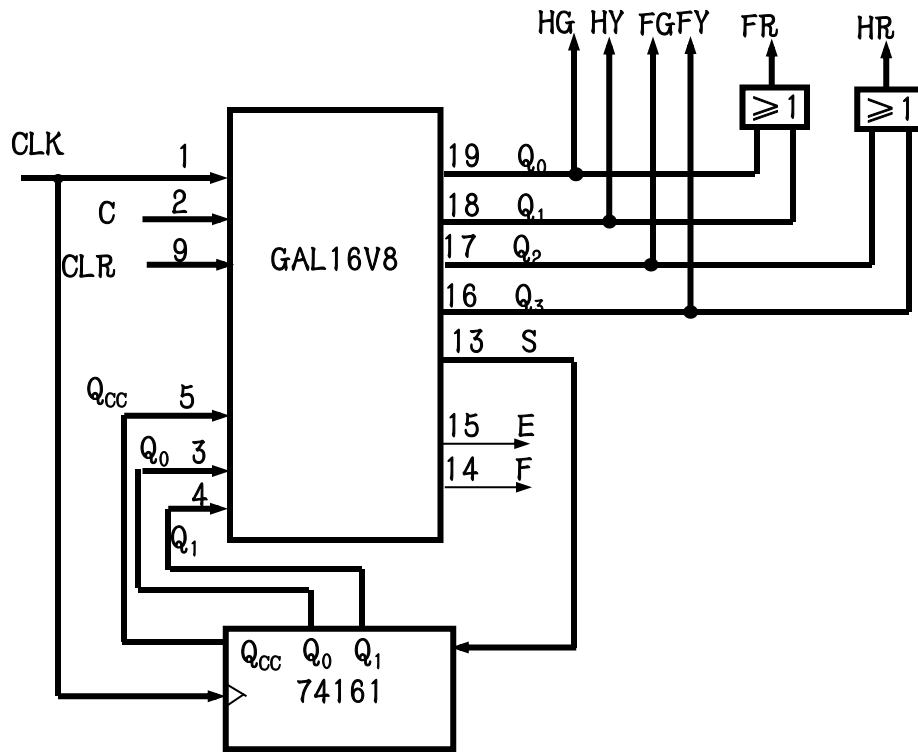


图12.5.8 初始结构框图



编写的CUPL程序

```
NAME TRAFFICLIGHT;  
PARTNO;  
REV V1.0;  
DATE X/X/X;  
DESIGNER XXX;  
COMPANY NUPT;  
ASSEMBLY;  
LOCATION;  
/* INPUT PIN*/  
PIN[1,2,3,4,5,9]=[CLK,C,QA,QB,QCC,CLR];  
/* OUTPUT PIN*/  
PIN[19,18,17,16,15,14,13,11]=[Q0,Q1, Q2, Q3, E, F,S,OE];
```

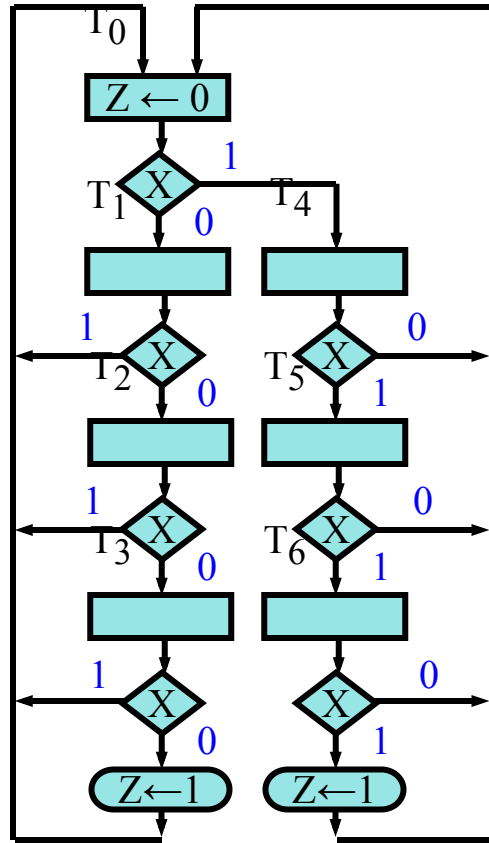


```
/*REGISTER INPUT SIGNAL*/  
Q0.D=!CLR#(CLR&(Q0&!C#Q0&!E#Q3&F));  
Q1.D=CLR&(Q0&C&E#Q1&F);  
Q2.D=CLR&(Q1&F#Q2&C&!E));  
Q3.D=CLR&(Q2&E#Q2&!C#Q3&!F));  
/*START SIGNAL*/  
S=Q0&C&E#Q1&F#Q2&E#Q2&!C#Q3&F;  
/*TIMING*/  
F=Q1&QA&QB&!E#Q3&QA&QB&!E#!S&E;  
E=Q0&QCC&!F#Q2&QCC&!QF#!S&F;  
/*END*/
```

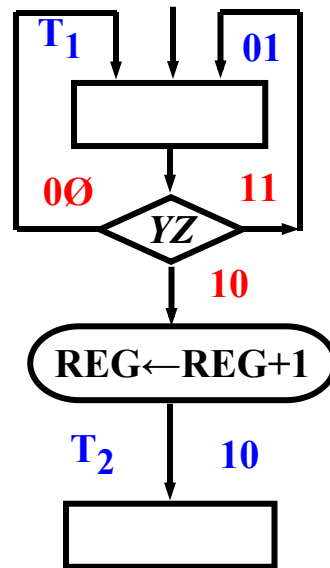


例3 设电路的输入为 X ，输出为 Z ，当 X 在连续的四个时钟周期内输入全“0”或全“1”时，输出为“1”，否则输出为“0”，试画出该电路的ASM图。



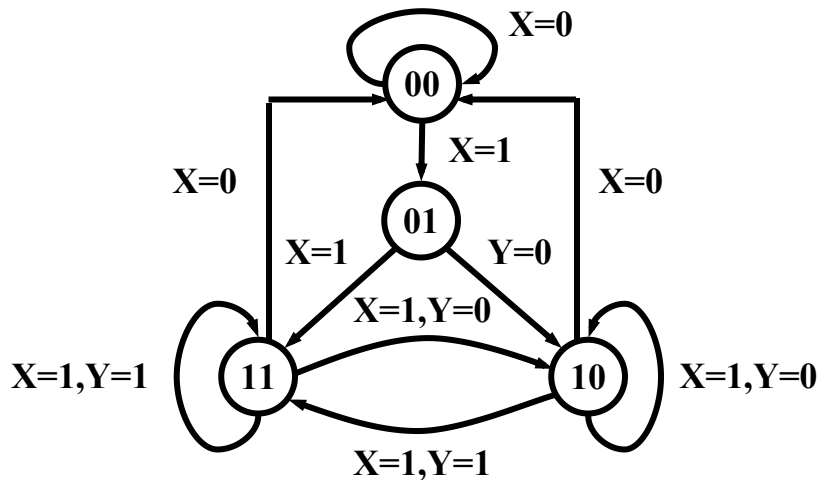


例4 在 T_1 状态下，如果控制输入Y和Z分别等于1和0，系统实现条件操作——寄存器增1，并切换到状况 T_2 。试按上述条件画出一个部分ASM图。

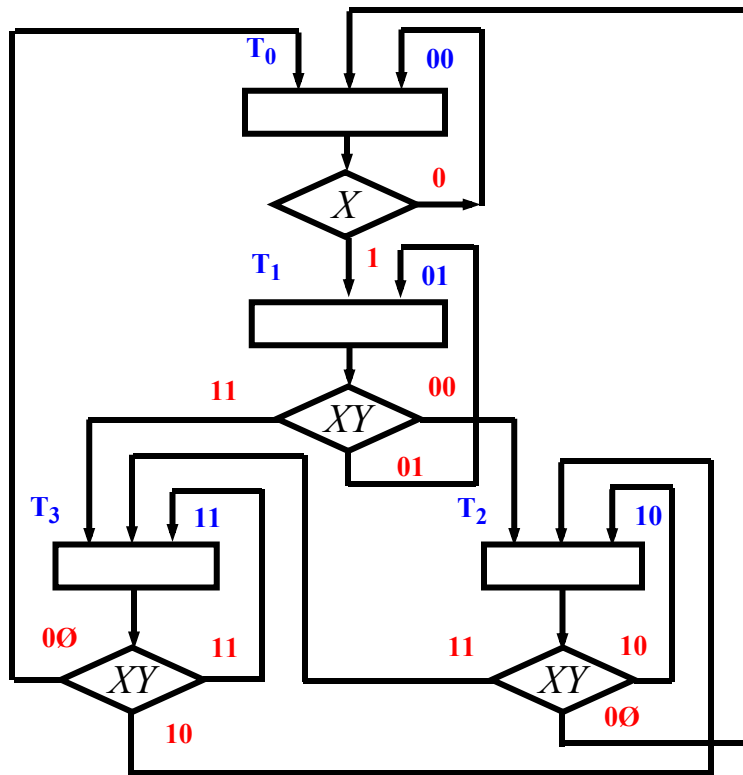
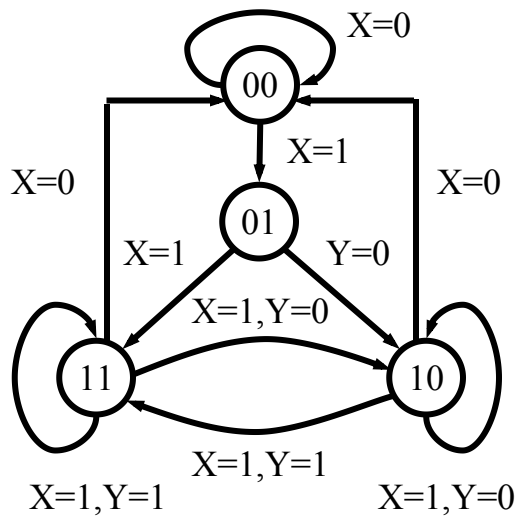


例5 控制器的状态图示于下图，它有四个状态和两个输入端。请完成下列问题：

- (1) 试画出等效的ASM图（状态框是空的）；
- (2) 用数据选择器实现控制器。



(1) 试画出等效的ASM图（状态框是空的）；



(2) 用数据选择器实现控制器。

将二进制代码00, 01, 10, 11分别分配给 T_0 、 T_1 、 T_2 、 T_3 ，控制器的状态转移表如下：

	现 态		输 入		次 态	
	Q_2	Q_1	X	Y	Q_2	Q_1
T ₀	0	0	0	∅	0	0
	0	0	1	∅	0	1
T ₁	0	1	0	0	1	0
	0	1	0	1	0	1
	0	1	1	0	∅	∅
	0	1	1	1	1	1

	现 态		输 入		次 态	
	Q_2	Q_1	X	Y	Q_2	Q_1
T ₂	1	0	0	∅	0	0
	1	0	1	0	1	0
	1	0	1	1	1	1
T ₃	1	1	0	∅	0	0
	1	1	1	0	1	0
	1	1	1	1	1	1



控制器有4个状态（ T_0 、 T_1 、 T_2 、 T_3 ），用2个触发器、2个4选1数据选择器实现。 Q_2 、 Q_1 的次态卡诺图如下：

		Q_1	
		0	1
Q_2	0	0	$x+y$
	1	x	x

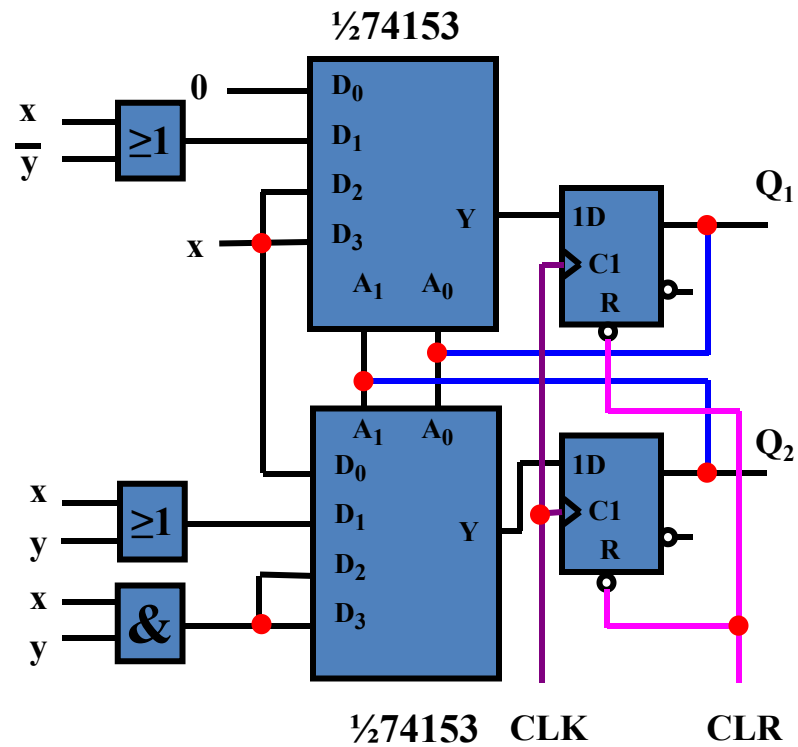
(a) Q_2 次态图

		Q_1	
		0	1
Q_2	0	x	$x+y$
	1	xy	xy

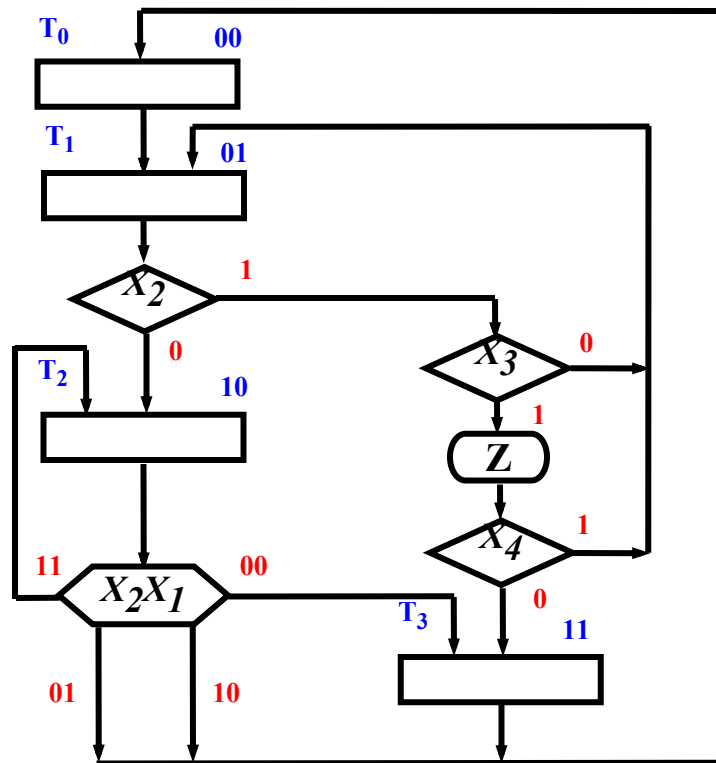
(b) Q_1 次态图



电路图如下:



例6 分别用每态一个触发器法，
多路选择器—时序寄存器
器—译码器法设计右图
所示ASM图的控制器的。



(1) 用每态一个触发器法:

控制器有4个状态 (T_0 、 T_1 、 T_2 、 T_3)，需4个触发器，采用DFF，并分别用F0~F3表示，由ASM图知:

$$D_0 = T_3 + T_2(\bar{X}_2X_1 + X_2\bar{X}_1) = T_3 + T_2\bar{X}_2X_1 + T_2X_2\bar{X}_1$$

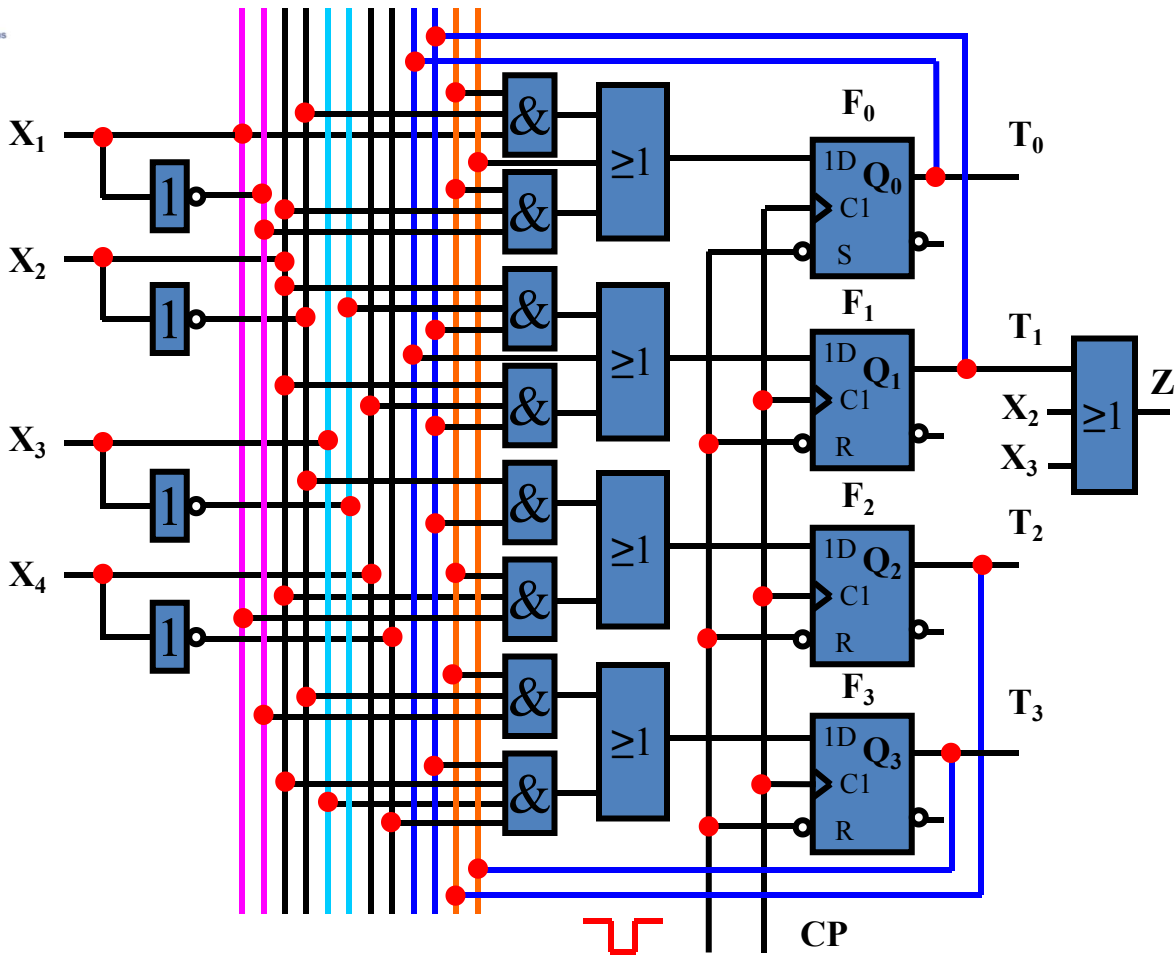
$$D_1 = T_0 + T_1(X_2\bar{X}_3 + X_2X_3X_4) = T_0 + T_1X_2\bar{X}_3 + T_1X_2X_4$$



$$D_2 = T_1\bar{X}_2 + T_2X_2X_1$$

$$D_3 = T_2\bar{X}_2\bar{X}_1 + T_1X_2X_3\bar{X}_4$$

电路图如下:



(2) 用多路选择器—时序寄存器—译码器法:

控制器有4个状态 (T_0 、 T_1 、 T_2 、 T_3)，需2个触发器，采用DFF，并分别用F0、F1表示，需2个四选一多路选择器，1个2—4线译码器:



将二进制代码00，01，10，11分别分配给 T_0 、 T_1 、 T_2 、 T_3 ，控制器的状态转移表如下:



	现 态		输 入				次 态		输 出
	Q ₂	Q ₁	X ₁	X ₂	X ₃	X ₄	Q ₂	Q ₁	Z
T ₀	0	0	∅	∅	∅	∅	0	1	0
	0	1	∅	0	∅	∅	1	0	0
T ₁	0	1	∅	1	0	∅	0	1	0
	0	1	∅	1	1	0	1	1	1
	0	1	∅	1	1	1	0	1	1
T ₂	1	0	0	0	∅	∅	1	1	0
	1	0	0	1	∅	∅	0	0	0
	1	0	1	0	∅	∅	0	0	0
	1	0	1	1	∅	∅	1	0	0
T ₃	1	1	∅	∅	∅	∅	0	0	0



Q_2 、 Q_1 的次态卡诺图如下：

	Q_1	0	1
Q_2	0	0	$\bar{x}_2 + x_3 \bar{x}_4$
	1	$x_1 \odot x_2$	0

(a) Q_2 次态图

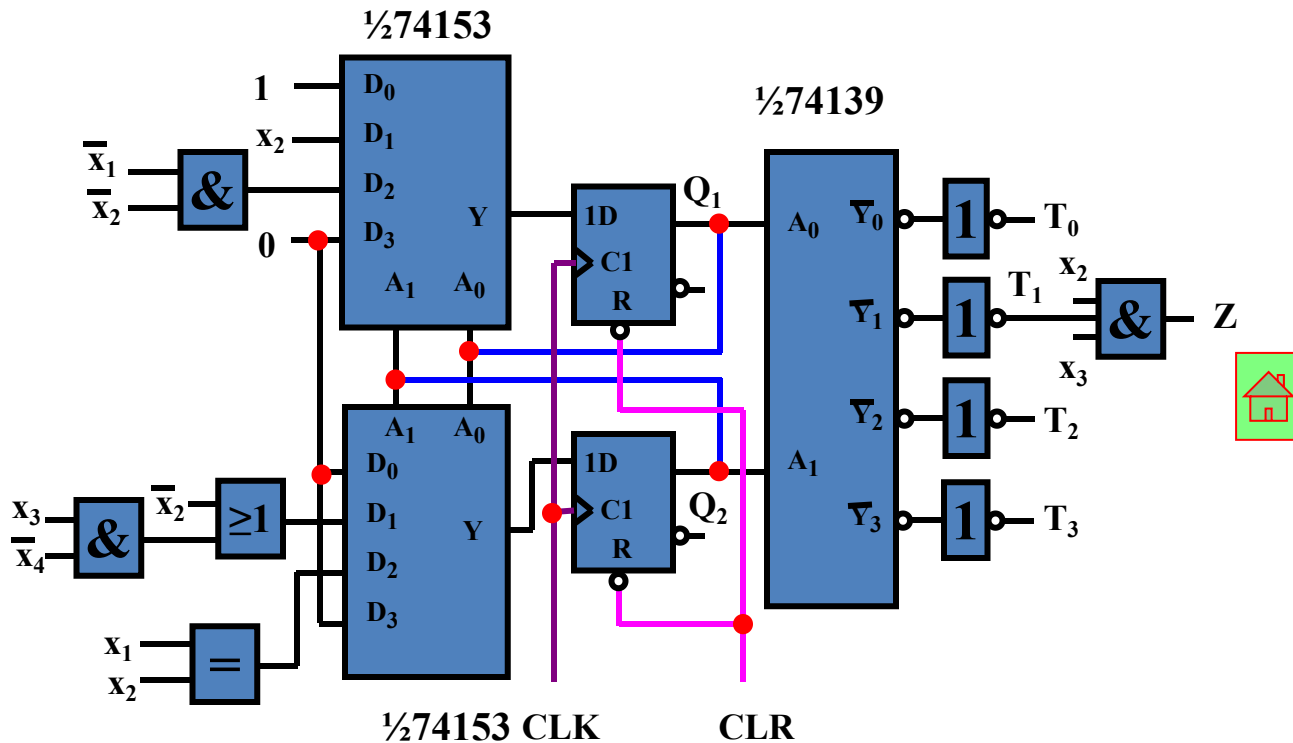
	Q_1	0	1
Q_2	0	1	x_2
	1	$\bar{x}_1 \bar{x}_2$	0

(b) Q_1 次态图





电路图如下:





进行分解时，须遵循下列原则，才能得到一个系统化的、清晰易懂的、可靠性高、可维护性好的设计：

(1) 正确性和完备性原则

检查指标所要求的各项功能是否都实现了，且留有必要的余地，最后还要对设计进行适当的优化。

(2) 模块化、结构化原则

每个子系统、部件或子部件应设计成在功能上相对独立的模块，而且对某个模块内部进行修改时不应影响其他的模块。



(3) 问题不下放原则

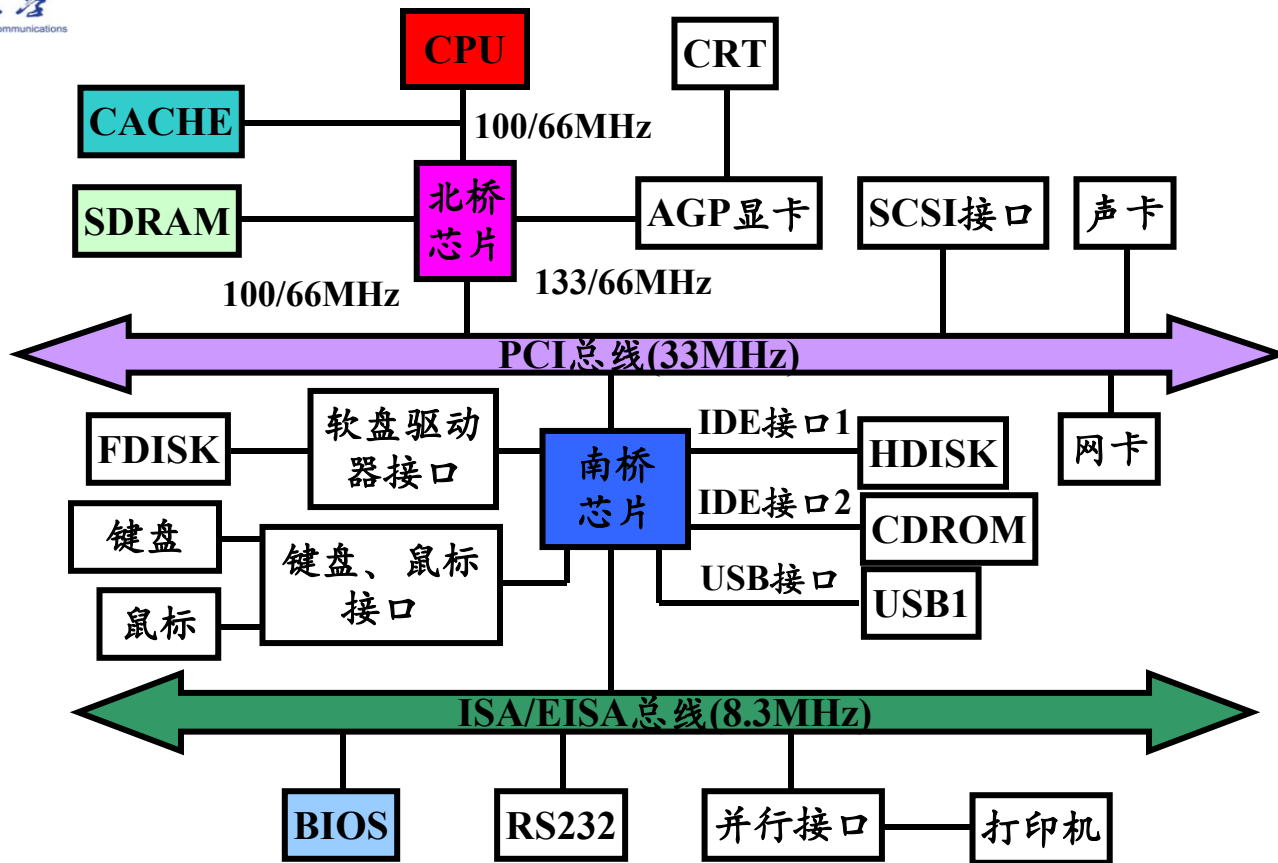
在某一级的设计中如遇到问题时，必须将其解决了才能进行下一级的设计。

(4) 高层主导原则

在底层遇到的问题找不到解决办法时，必须退回到它的上一级去甚至再上一级去，通过修改上一级的设计来减轻下一级设计的困难。

(5) 直观性、清晰性原则

不主张采用难以理解的诀窍和技巧，应当在设计中和文档中直观、清晰地反映出设计者的思路。



Pentium 个人计算机系统