

## 前言

本讲义是“嵌入式操作系统与微控制器”这门课的配套实验教材，围绕MC9S12DP256/DG128 核心子板和母板共安排了 16 个实验，引导学生从最基本的汇编语句开始，逐步熟悉嵌入式软硬件的设计方法。

我们不是巨人，站在我们的肩膀上能看到的只是最浅显的一个层面。但对于一个对嵌入式领域十分感兴趣却又惴惴不安不敢入内的人来说，我们还是可以起到一个领路人的作用。如果能够一丝不苟地做完一部分实验并切实钻研了其中的奥妙，一个学期下来会有很大的收获。

但一日建不成罗马，没有打破砂锅问到底的精神也成不了“大牛”。需要指出的是，这 16 个实验充其量也就是个敲门砖而已，真正要成为嵌入式设计的高手，需要不间断地吸收新东西且不断地实践之，在疑问中找到成长的路径。

为了利用有限的实验时间获得最大的收益，我们希望每一个做实验的人都能做到以下四点：

1. 做实验之前熟读相关的资料，做到胸有成竹，最好能在课前准备好程序
2. 不放过实验过程中遇到的每一个问题，尽量尝试着自己去解决
3. 不拘泥于本书中的实验要求，敢于按照自己的想法做相关的实验
4. 课后写好实验报告，着重总结实验中遇到的问题和获得的经验。

翻开下一页，迈出千里之行的第一步。

## 目录

### 第一部分：基本实验

实验 1.	熟悉实验环境.....	3
实验 2.	S12 汇编语言基础 .....	4
实验 3.	C 程序编写 .....	5

### 第二部分：S12 I/O 外设模块实验

实验 4.	GPIO 通用输入输出及 IRQ 中断.....	6
实验 5.	SCI 串行口实验 .....	9
实验 6.	用 TIMER 实现精密定时 .....	12
实验 7.	A/D 转换实验 .....	15
实验 8.	SPI 同步串行外设接口 .....	17
实验 9.	I <sup>2</sup> C 总线实验 .....	22
实验 10.	PWM 模块实验 .....	26
实验 11.	MSCAN 模块的自循环模式.....	31
实验 12.	Keyboard 实验.....	35
实验 13.	动态数码管的显示.....	37
实验 14.	Flash 在线编程 .....	38

### 第三部分：uC/OS-II 实验

实验 15.	uC/OS-II 在 S12 上的移植.....	40
实验 16.	综合实验 —— uC/OS-II 下多 I/O 任务实现.....	41

## 实验1. 熟悉实验环境

### 一、实验目的

熟悉从工程建立到调试的整个过程。

### 二、实验任务

用 Codewarrior 新建一个 ASM 工程 (ASM Generic), 将 main.asm 修改成如下内容 (不包括行号):

```
1. portb    equ $0001
2. ddrb     equ $0003

3.         org $c000
4. main:
5.         ldaa #$ff
6.         staa ddrb
7.         ldaa #$f0
8.         staa portb
9.         ldaa #$0f
10.        staa portb
11.        bra *
```

编译连接生成.s19 文件, 通过监控程序把 s19 文件中的机器码下载到 S12 单片机中, 用各项监控指令对该程序进行调试, 观察实验现象 (注意设置断点)。

### 三、预习要求

阅读课本附录 A.2~A.4 明确如下问题:

1. 如何用 Codewarrior 建立一个 ASM 工程?
2. 超级终端该如何设置以便和监控程序建立通信?
3. 程序的起始地址?
4. 如果要分别观察上程序中第 8 行和第 10 行代码的运行结果, 需要使用哪些监控命令?

## 实验2. S12 汇编语言基础

### 一、实验目的

1. 熟悉 S12 汇编语言，掌握 S12 汇编语言对各存储器和寄存器的操作。
2. 对 S12 内部寄存器和存储器的结构分布有一个明确的概念。

### 二、实验任务

1. 编写汇编语言源程序，在\$2000 处存放八位十六进制数\$A2，在\$2100 处存放八位十六进制数\$2B，将它们相加后存放于\$2200 处。
2. 将上一任务的第二个数改为\$FB，重复实验，观察标志寄存器的变化。
3. 将 PORTB 接的 8 个 LED 的状态设成“亮灭亮灭亮灭亮灭”。

### 四、预习要求

阅读课本第三章，明确如下问题：

1. MC9S12DP256 的 RAM 存储区从哪里开始？到哪里结束？DG128 呢？
2. MC9S12DP256 和 DG128 的寄存器都有哪些？各有什么用途？
3. 怎样在\$2000 处定义一个八位十六进制数？
4. 标志寄存器有几个有效位？分别都代表什么含义？
5. 两个数相加的指令是什么？
6. 怎样控制 PORTB 口的状态？（输入输出？各个端口电平高低？）

## 实验3. C 程序编写

### 一、实验目的

了解用 C 语言编写嵌入式软件的方法。

### 二、实验任务

1. 向\$2500~\$251f 的 RAM 区域任意填入 32 个数，找出从指针 ppoint 指向的单元开始的包含 n 个字节的 RAM 区中最大的一个数，将这个数用 LED 显示出来，其中 ppoint 和 n 可以任意指定(但由 ppoint 和 n 构建的查找区域位于\$2500~\$251f 内，比如 ppoint=\$2503 n=4 则表示查找\$2503~\$2506 这段 RAM 区中最大的数并将其用 LED 显示，建议用 PORTB 上接的 8 个 LED)。

2. 编写 SCI 通信子程（参阅实验五关于 SCI 模块的介绍）。

### 三、预习要求

(1) 阅读课本第 4 章和第 8 章明确如下问题：

1. 如何用 C 程序语句和硬件打交道？
2. C 程序和汇编程序相互调用需要注意什么？相应的 C 编译器如何处理参数传递？
3. 代码段如何和相应的存储器绑定？（prm 文件的书写方法）

(2) 设计软件流程，给出设计代码。

## 实验4. GPIO 通用输入输出口及 IRQ 中断

### 一、实验目的

1. 掌握 MC9S12DP256 汇编语言对通用端口的操作指令。
2. 掌握程序中指令循环和跳转的方法。
3. 学会使用程序延时，并会大概估算延迟时间。

### 二、实验任务

1. 在 PORTB 口的小灯上循环显示跑马灯。
2. 将 PORTA 口接八位 DIP 开关，在 PORTB 口的小灯上显示其状态。
3. 利用 IRQ 作为中断源，采用中断方式，按动 IRQ 键后显示跑马灯。
4. 利用 IRQ 作为中断源，采用中断方式，利用八位 DIP 开关控制跑马灯的循环速度。开关的不同状态代表不同的速度，IRQ 按动触发。

### 三、硬件资源

MC9S12DP256/DG128 微控制器有着丰富的端口资源，PORTA、PORTB、PORTE、PORTK、PORTH、PORTJ、PORTM、PORTS、PORTP 和 PORTT 构成了极丰富的端口网络。但由于实验条件和实验时间的限制，我们不可把这些端口都用到。所以在实验板上我们只引出了 PORTA 和 PORTP 两个端口的全部引脚，如果以后的工作学习中大家有其它的引脚要引出时，原理和这两个端口是一样的。另外，PORTB 虽然没有引出接线部分，但在子板的内部已经将其和八个小灯连接起来了。这样的话 PORTB 作为输出端口接小灯的话，编制程序时还是可以利用的。

总线上还引出了 IRQ 和 XIRQ 两个插孔，作为中断信号的输入端。具体的用法请参阅 MC9S12DP256 或 DG128 的 DATASHEET。

IRQ 中断信号产生电路可产生上升沿信号和下降沿信号。

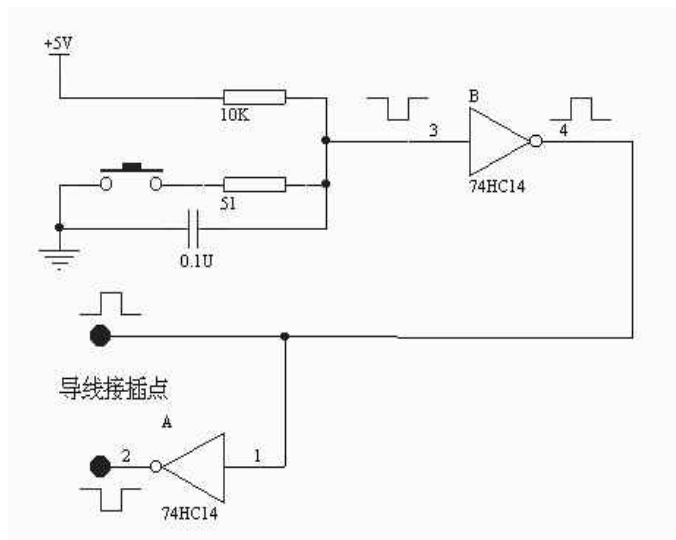


图 4.1 IRQ 中断信号产生电路

八位开关量输入电路，当开关处于 ON(闭合)状态，则输入低电平；当开关处于 OFF(打开)状态，则输入高电平。

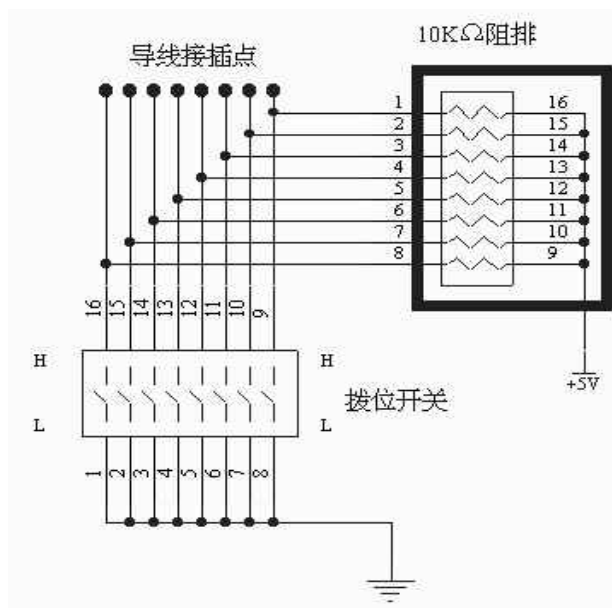


图 4.2 八位开关量数字电路

#### 四、实验说明

实验中每个通用输入输出端口要用到的寄存器都有两个，端口定义寄存器和端口方向寄存器。

以 B 端口为例，端口定义寄存器为 `PORTB` 和端口方向寄存器为 `DDRB`。在 MC9S12DP256 或 DG128 的 DATASHEET 上可以查到这两个寄存器的地址是 \$0001 和，则定义寄存器的语句为

```
PORTB EQU    $0001
DDRB  EQU    $0003
```

将其写在程序的开始处，后面的程序中就可以使用了，`DDRB` 的各位定义了 B 端口相应位的方向，置 0 表示该位为接受输入位，置 1 表示该位为输出位。

在每一种输出状态确定后，要进行一定的延时以保持该状态以便于人眼观察。延时程序一般来说就是对一个寄存器内的值进行不断地加或减的操作，直到预定的值为止。比如说下面的程序段：

```
LDX    #$4000----- (1)
WAIT:----- (2)
DEX----- (3)
CPX    #$0000----- (4)
BNE    WAIT----- (5)
```

- (1) 是对寄存器 X 赋一个初使值，这个值的确定在后面会提到。
- (2) 一个标号，方便循环的进行。
- (3) 将寄存器 X 的值减 1。
- (4) 将寄存器 X 内的值与#\$0000 相减，但这个结果只影响标志位的值，不对寄存器的值构成影响。
- (5) 如果结果不为 0，就是说寄存器 X 的值不是#\$0000，那么就跳转回 WAIT 标号的地方继续执行。NE 的意思就是 Not Equal

注意不要犯一种简单的错误，就是将 WAIT 标号放到 LDX 指令的上方，那样永远也无

法执行结束了。

像上面那个小例子，循环部分执行了 16384 次（注意十六进制与十进制的转换），基本上就可以认为循环的时间就是延时的时间了。相信大家也明白了，要知道延时的时间，首先要知道的就是每一条指令的执行时间。这个和 CPU 的工作频率还有指令的执行周期直接相关。同学们可以查相关的新产品手册来大概确定一下。如果 CPU 工作频率为 8MHz，程序里每 1 条指令占两个时钟周期，一共三条循环指令，那么循环 16384 次的时间大概就是  $16384 * 3 * 2 / 8000000 \text{ s} = 0.012288 \text{ s}$ 。这个数字还是小了点，有时候得设置双重循环才可以符合要求。

程序中使用中断的方法，首先要查中断向量表中程序所要使用的中断的位置。由于监控程序中对中断向量表作了一些处理，请大家认真阅读教材中关于监控程序中断向量表管理的章节（4.15）。查清楚中断向量表中 IRQ 中断源所在位置，将中断子程序的入口地址写入中断向量表中，完成对中断子程序的设置。注意定义中断子程序的位置要在整个源程序的末尾。例如：

```

    ×××
SUB_NAME:
    ×××
    BRA *

    ORG    $++++      ; +++++为中断向量表中中断源所在位置
    FDB    SUB_NAME

```

## 五、预习要求

1. 查询要使用的各端口的寄存器，A 端口的地址寄存器和方向寄存器各是多少？
2. 在教材中查询有关数据，写一个一秒钟的延时子程序；
3. 关于寄存器的移位指令有哪些？
4. 跑马灯的移位点亮有几种实现方法？
5. 查 XIRQ 和 IRQ 的中断源地址。
6. 你还能想出其它更生动的小灯程序吗、

## 实验5. SCI 串行口实验

### 一、实验目的

1. 掌握 S12 单片机串行口的基本应用方法。
2. 掌握串行口相关寄存器的设定方法。
3. 实现 S12 单片机与 PC 机之间串行通信的方法。

### 二、实验任务

PC 机从通过串口给单片机发送一个 1—8 的数字，点亮相应数量的调试小灯（小灯与 PORTB 相连）；然后让 LED 以跑马灯的形式循环闪烁。

### 三、实验说明

为了实现 PC 机与单片机的通信，首先要对与 SCI 相关的寄存器进行初始化。

以下是 SCI 通信涉及到的寄存器：

SCIBDH:

	7	6	5	4	3	2	1	0
R	0	0	0	SBR12	SBR11	SBR10	SBR9	SBR8
W								
RESET:	0	0	0	0	0	0	0	0

SCIBDL:

	7	6	5	4	3	2	1	0
R	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
W								
RESET:	0	0	0	0	0	1	0	0

SBR12-0: 波特率设置位，代表 BR 的值，波特率的计算公式为：

$$\text{SCI baud rate} = \text{SCI module clock} / (16 \times \text{BR}),$$

当 BR 被设置为 0 时，波特率发生器被禁止。

SCICR1:

	7	6	5	4	3	2	1	0
R	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT
W								
RESET:	0	0	0	0	0	0	0	0

LOOPS: 循环选择位，设置为 1 表示允许循环操作，设置为 0 表示普通状态。

SCISWAI: 等待模式下 SCI 停止位，设置为 1 表示等待模式下 SCI 被禁止，设置为 0 表示仍被允许。

RSRC: 接收源选择位，当 LOOPS 被设置为 1 时，RSRC 设置为 1 表示接收输入端与发送端外部连接，设置为 0 表示接收输入端与发送端内部连接。

下表是 LOOPS 与 RSRC 两位的状态表：

LOOPS	RSRC	Function
0	x	Normal operation
1	0	Loop mode with Rx input internally connected to Tx output
1	1	Single-wire mode with Rx input connected to sci_tx_ind

表 5.1 LOOPS 与 RSRC 状态表

M: 数据格式模式位, 设置为 1 表示有 1 位起始位, 9 位数据位, 1 位停止位, 设置为 0 表示有 1 位起始位, 8 位数据位, 1 位停止位。

WAKE: 唤醒条件位, 设置为 1 表示由 Address mark 唤醒, 设置为 0 表示由 Idle line 唤醒。

ILT: 空闲线模式位, 置 1 表示空闲计数从停止位开始, 置 0 表示空闲计数从起始位开始。

PE: 奇偶校验选择位, 置 1 表示允许插入奇偶校验位, 置 0 禁止。

PT: 奇偶校验形式位, 置 1 表示为奇校验, 置 0 表示为偶校验。

SCICR2:

	7	6	5	4	3	2	1	0
R	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
W								
RESET:	0	0	0	0	0	0	0	0

TIE: 传输中断选择位, 置 1 表示允许中断, 置 0 表示禁止中断。

TCIE: 传输完成中断选择位, 置 1 表示允许中断, 置 0 表示禁止中断。

RIE: 接收寄存器满中断选择位, 置 1 表示允许中断, 置 0 表示禁止中断。

ILIE: 空闲线中断选择位, 置 1 表示允许中断, 置 0 表示禁止中断。

TE: 传输发送允许位, 置 1 表示允许传输发送, 置 0 表示禁止。

RE: 接收允许位, 置 1 表示允许接收, 置 0 表示禁止。

RWU: 接收唤醒位, 设置为 1 表示普通操作, 设置为 0 表示允许唤醒操作且在此期间禁止接收中断请求, 实验时设置为 1 即可。

SBK: 发送断点位, 置 1 表示发送断点特征, 置 0 表示不发送。

SCISR1:

	7	6	5	4	3	2	1	0
R	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
W								
RESET:	1	1	0	0	0	0	0	0

TDRE: 发送数据寄存器空标志位, 状态为 1 表示数据已送到发送寄存器中, 状态为 0 表示发送寄存器为空。

TC: 发送完成位, 状态为 1 表示没有发送进程, 状态为 0 表示正在发送。

RDRF: 接收数据寄存器满标志位, 状态为 1 表示接收寄存器中数据可用, 状态为 0 表示数据寄存器中没有数据。

IDLE: 空闲线标志位, 状态为 1 表示接收端空闲, 状态为 0 表示接收端正在活动。

OR: Overrun 位, 状态为 1 表示有过运行情况发生, 状态为 0 表示没有。实验中用不到。

NF: 噪声标志位, 状态为 1 表示数据中有噪声, 状态为 0 表示没有。

FE: 帧错误标志位, 状态为 1 表示有帧错误发生, 状态为 0 表示没有。

PF: 奇偶错误标志位, 状态为 1 表示有错误发生, 状态为 0 表示没有。

SCISR2:

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	BRK13	TXDIR	RAF
W								
RESET:	0	0	0	0	0	0	0	0

BRK13: 中断传输特征长度位, 置 1 表示 13 或 14 位的长度, 置 0 表示 10 或 11 位的长度。

TXDIR: 单方向状态下 TxD 引脚方向位, 置 1 表示 TxD 作为输出端, 置 0 表示作为输入端。

RAF: 接收器活动标志位, 状态为 1 表示有接收进程, 状态为 0 表示没有接收进程。

SCIDRH:

	7	6	5	4	3	2	1	0
R	R8	T8	0	0	0	0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

SCIDRL:

	7	6	5	4	3	2	1	0
R	R7	R6	R5	R4	R3	R2	R1	R0
W	T7	T6	T5	T4	T3	T2	T1	T0
RESET:	0	0	0	0	0	0	0	0

这两个寄存器是数据寄存器, R 表示接收, T 表示发送, T8, R8 的有效与否与 SCICR1 里的 M 有关, 当 M 置 1 时有效, M 置 0 则无效。

#### 四、预习要求

(1) 参考 datasheet 明确 SCI 各寄存器的作用, 思考如何将通讯设置为波特率 9600, 8 位数据位, 无奇偶校验位, 1 个停止位, 无流量控制?

(2) 设计软件流程, 给出设计代码。

## 实验6. 用 TIMER 实现精密定时

### 一、实验目的

了解 S12 单片机的定时器结构，掌握使用定时器的精密定时方法。

### 二、实验任务

使用定时器，使小灯每两秒改变一次亮灭状态。

### 四、实验说明

在复杂的单片机应用系统中，定时功能往往是必不可少的。根据精度和长短的要求，定时功能的实现方法也有所不同。

#### 1. 软件延时

在定时精度要求不高的场合下，可以使用软件延时的方法，简单的说就是写几条循环指令，根据指令表可以估算循环指令的执行时间，再乘以循环次数，就可以得到总的延时时间，如：

```

ldaa #$08      (1 个指令周期)
loop: deca     (1 个指令周期)
      cmpa #$00 (1 个指令周期)
      bne  loop  (转移: 3 个指令周期; 不转移 1 个指令周期)

```

分析指令我们可以看出，整个循环体执行一次要 5 个指令周期，乘以循环次数 8，整个的延时次数大约是 40 个指令周期。

使用软件延时的方法不需要有单独的硬件支持，实现简单，但是缺点是延时过程 CPU 一直被占用，且计算延时时间较繁琐，不易控制。

#### 2. 硬件定时

使用单独的硬件来完成定时，可以使 CPU 在延时的过程中仍能处理其他程序，我们一般称这样的硬件为“定时器”。由于定时器与 CPU 相独立，互相的通信由中断来完成，在初始化定时器以后，CPU 就可以放心执行其他任务了，定时完成后，由定时器发出中断来通知 CPU。

MC9S12DP256/DG128 中可以使用实时时钟或增强型定时器来完成定时功能，二者是相互独立的。下面分别介绍。

实时时钟的可以通过对外部晶振分频而得到一个定时中断。RTICTL 是实时时钟控制寄存器，向该寄存器写入内容，通过查表会得到一个分频因子，外部晶振除以分频因子就是中断的频率了。（参见教材 191 页，6.4 产生时钟节拍中断一节）

实时时钟使用简便，但受外部晶振和分频因子大小所限，不易实现精密定时，想得到高精度编程的整倍数的时钟节拍中断，可以使用增强型定时器。

下面介绍使用增强型定时器的模数计数器实现精密定时的方法。

模数计数器是一个递减的 16 位计数器，既可以作为一个时基定时地产生中断，也可以用来产生控制信号将输入捕捉寄存器或者脉冲累加器的值锁存到保持寄存器中。访问该计数器，操作应该在一个时钟周期内完成。有关模数计数器的寄存器包括：

模数递减计数器 (MCCNTH、MCCNTL)	\$0076、\$0077
模数递减标志寄存器 (MCFLG)	\$0067
模数递减控制寄存器 (MCCTL)	\$0066

模数计数器由初值递减，递减到 0 时，会产生相应的中断。递减频率由总线时钟经分频得到，为了实现精密定时，初值是可以由用户写入的，默认情况是 \$ffff，在模数模式允许后 (MODMC=1)，向模数递减计数器的写操作会更新预置数的值，模数计数器递减到 0 时，就会装载最新的预置数，装载的操作也可以由置 MCCTL 寄存器中的 FLMC 位为 1 来强制执行。

**总定时时间=预置数×总线周期×中断次数**

实验系统中单片机内部总线频率为 24M，如果需要 2 秒的定时时间，预置数和中断次数分别应该设为多少？请利用上面公式自行计算。

## 四、实验步骤

使用模数计数器实现定时的程序流程图如下：

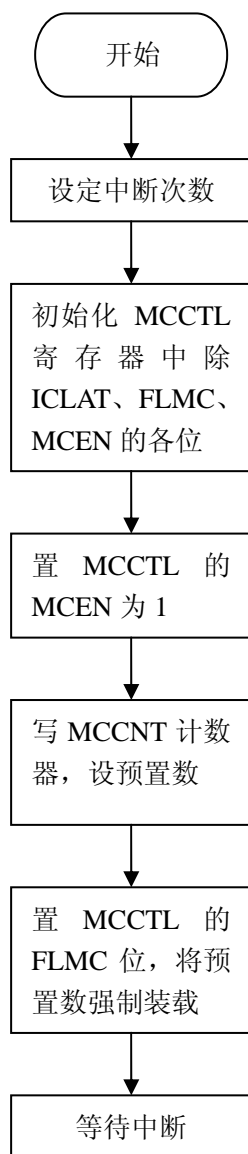


图 6.1 精密定时软件流程图

在中断程序中，用户需要清除相应的中断标志位（MCFLG），并将中断次数减 1，当中断次数减为 0 时，一次延时就完成了。

## 五、预习要求

1. 教材 285 页，9.3 增强型定时器一节，明确使用硬件定时相对于软件延时的好处（从 CPU 工作状态，最短定时等方面进行分析）
2. 编写程序

## 实验7. A/D 转换实验

### 一、实验目的

了解 S12 单片机 ADC 模块的使用方法。

### 二、实验任务

用 S12 的 ADC 模块将一路(或多路)模拟电平转换成数字量,并将转换结果显示在 LED 上,或者通过 SCI 发送到 PC 终端显示出来。

### 三、硬件资源

实验板上留出了 4 个模拟电平输入通道,分别是 AN6, AN7, AN14, AN15。其中 AN6 和 AN7 属于 ADC0, AN14 和 AN15 属于 ADC1,注意设置与采样通道相对应的寄存器。

模拟电平由电源 5V 经变阻器分压产生,范围为 0~5V。实验板上还另加了保护电路,模拟电平最好与保护电路连接后再输入到 ADC 的某一通道。

保护电路和模拟电平产生电路如下:

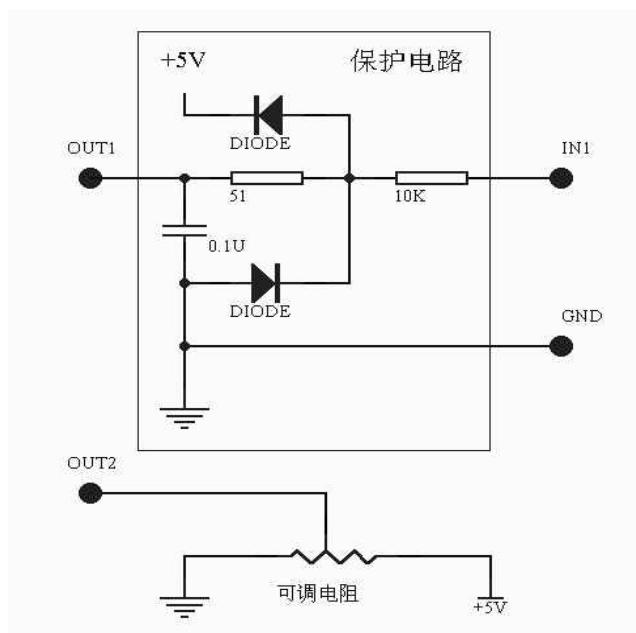


图 7.1 保护电路及模拟电平产生电路

## 四、实验说明

与 S12 的 ADC 模块相关的寄存器如下，各寄存器的详细定义可参阅 datasheet。

**ATDCTL2:** 控制寄存器。主要设置 A/D 标志位清除方式、A/D 采样触发方式、是否允许 A/D 采样完成中断等。

**ATDCTL3:** 控制寄存器。主要设置每次 A/D 转换采样几路电平、采样结果的存储方式等。

**ATDCTL4:** 控制寄存器。主要设置 A/D 转换精度、A/D 转换时钟频率等。

**ATDCTL5:** 控制寄存器。主要设置 A/D 转换结果的对齐方式和数据类型，以及 A/D 的采样模式（连续采样/单词采样，顺序转换/单通道转换等）

**ATDSTAT0:** 状态标志寄存器。包括 A/D 转换完成标志，出错标志、转换结果存储索引等标志位

**ATDTEST1:** 测试寄存器。

**ATDSTAT1:** 标志寄存器。标识一次 A/D 转换中各通道的完成情况。

以上寄存器的具体内容和与其他与 ADC 模块相关的寄存器请参看 datasheet 相应章节。

## 五、预习要求

(1) 参考 datasheet 明确 ADC 各寄存器的作用，主要思考以下问题：

1. A/D 转换的时钟应该是多少？如何设置分频因子？
2. A/D 转换如何启动？有几种启动方式？分别如何设置相关寄存器？
3. 每次 A/D 转换启动那儿路电平采样？采样结果如何存储（注意 FIFO 的 A/D 转换模式）？采样结果的数据类型（8 位/10 位？左对齐/右对齐？有符号数/无符号数？）？
4. 如何判断 A/D 转换是否结束？如何清标志位？

(2) 设计软件流程，给出设计代码。

## 实验8. SPI 同步串行外设接口

### 一、实验目的

1. 掌握同步串行外设接口 SPI 的工作原理，并籍此了解一些串行通讯的知识。
1. 学会使用 74LS164 和 74LS165 这两种芯片，掌握他们的工作原理。

### 二、实验任务

将 SPI 端口作为输入端接八位 DIP 开关，将 DIP 开关表示的八位二进制数在 PORTB 上显示出来；

1. 将 SPI 端口作为输出端接八位小灯，在小灯上输出 01010101 内容。
2. 利用实验板上的现有条件，将两个 SPI 端口分别作为输入输出端，输入端接八位 DIP 开关，输出端接八位小灯。要求将 DIP 开关表示的八位二进制数，在小灯上显示出来。
3. 利用 SPI 端口实现双机通讯：A 机向 B 机发送 ‘A’ 字符，B 机接收并在屏幕上显示出来，接着 B 机向 A 机回馈 ‘B’ 字符，A 机接收并在屏幕上显示出来。
4. 在双机通讯的基础上，将通讯内容改为字符串：“ HOW ARE YOU? ” —— “ FINE THANKS! ”。

### 三、硬件资源

MC9S12DP256 微控制器一共有三个 SPI 接口，实验板上为大家引出了两个，其引脚与 PORTP 复用。PORTP0-3 是 SPI1，PORTP4-7 是 SPI2。

SPI 的原理在实验教材中已经有详细的说明，在此不再赘述。

SPI 是一种串行接口，但我们实际生活中所用的许多的数据量都是并行的，因此需要将其转换为并行数据。在本实验中我们使用了 74LS164 芯片来完成串行数据转换为并行数据的功能，用 74LS165 完成并行数据转换为串行数据的功能。

实验板上还提供了八位 DIP 开关和八位小灯，可以作为二进制数的输入输出部分。

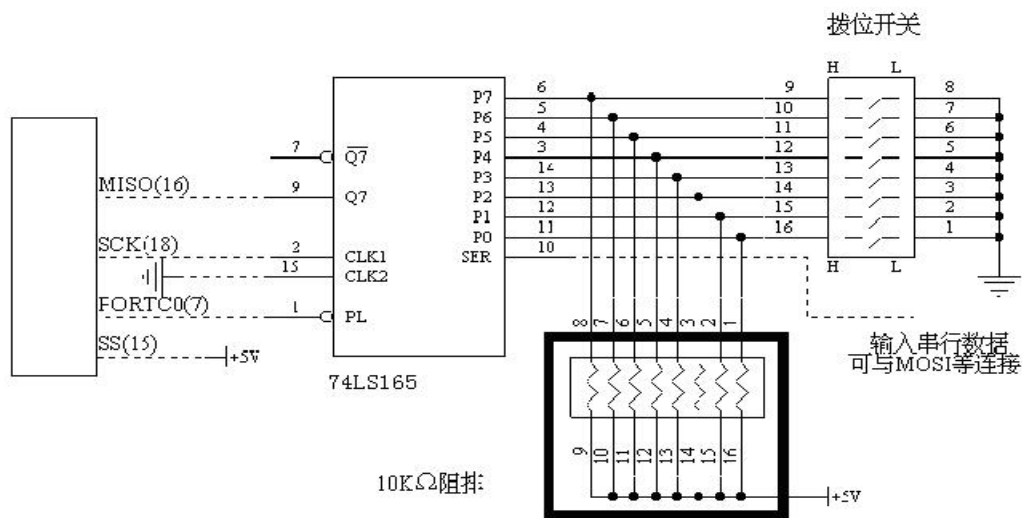


图 8.1 SPI 读操作实验电路

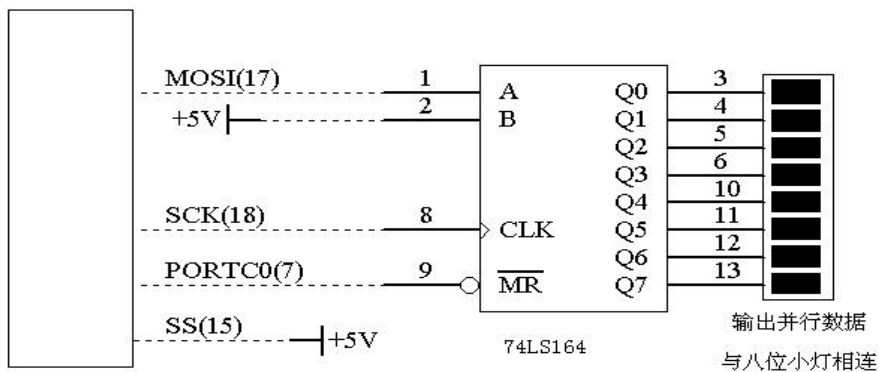


图 8.2 SPI 写操作实验电路

八位数字量显示模块。排阻起到限流保护的功能。接入电平为高时，相应的 LED 不发光；接入电平为低时，相应的 LED 发光。

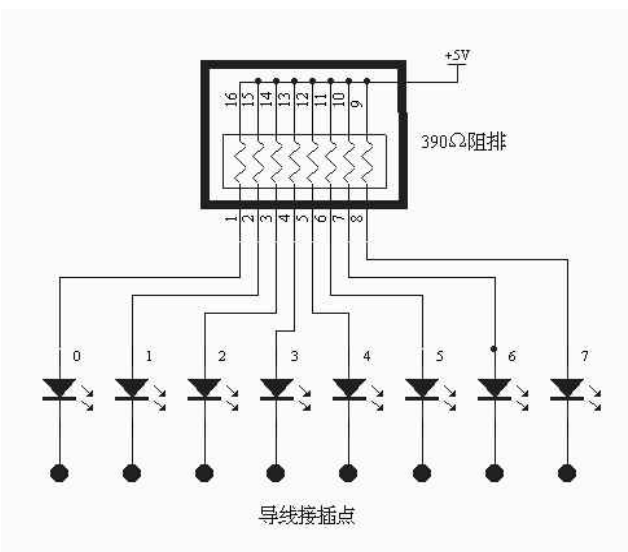


图 8.3 八位数字量显示模块

**74LS165:**

这是一款将数据转换为串行数据输出的芯片，下面是它的外部引脚图以及输入输出的对应关系。

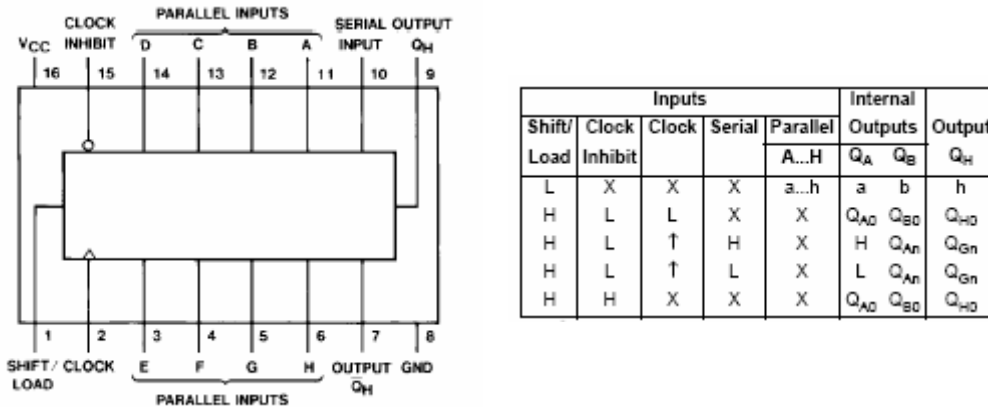


图 8.4 74LS165 引脚图及真值表

实验中用到的是并行输入串行输出的方式，参照输入输出的对应关系 Parallel 部分，当 Shift/Load 设置为低电平时会读入数据，高电平时会保持数据。这样的话，我们在发送的时候就可以先将 Shift/Load 置低读入数据，然后置高保持，发送至 MCU。注意在 Shift/Load 置低时最好有一个延时以保证数据已经被完全地读入，否则有可能发生错误。Shift/Load 置高后经过一个时钟周期数据就会以串行的形式发送到 MCU，然后按照上面的判断方式判断其是否发送完成。

下面给出 74LS165 的时序图以供参考：

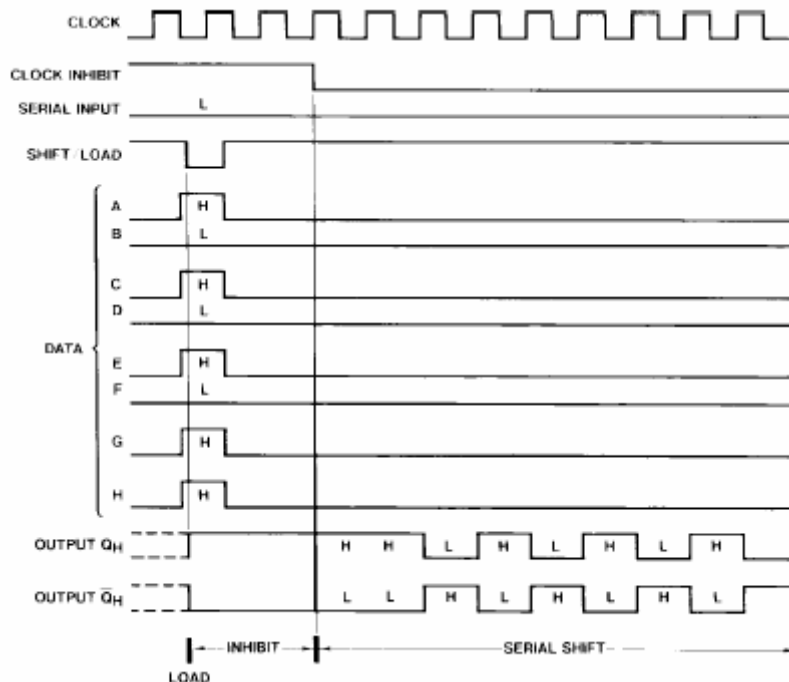


图 8.5 74LS165 时序图

**74LS164:**

这是一款将串行数据转换为并行数据输出的芯片，下面是它的外部引脚图以及输入输出的对应关系。

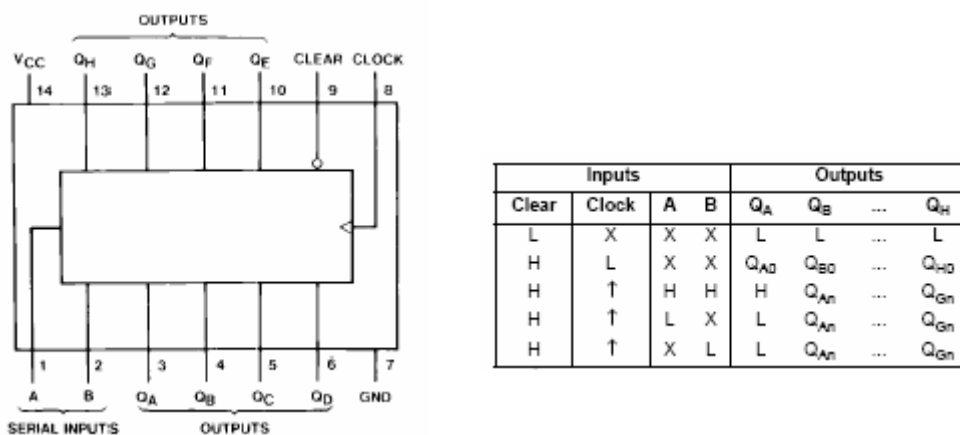


图 8.6 74LS164 引脚图及真值表

参照输入输出状态的对应图可以看出，Clear 端为低电平时会将输出端清零。Clear 端为高电平时，B 端也为高电平时，Clock 每经过一个上升沿，A 端的数据就会发出一位，8 个时钟周期后数据就会发送完成。具体的经过请大家自己分析。

编程时，先将 Clear 端置低清零，然后置高。将数据送入数据寄存器后就开始发送了，发送过程中不断地查询状态寄存器 SPISR 的 SPRF 位，待其发送完成。下面给大家 74LS164 的时序图以供参考：

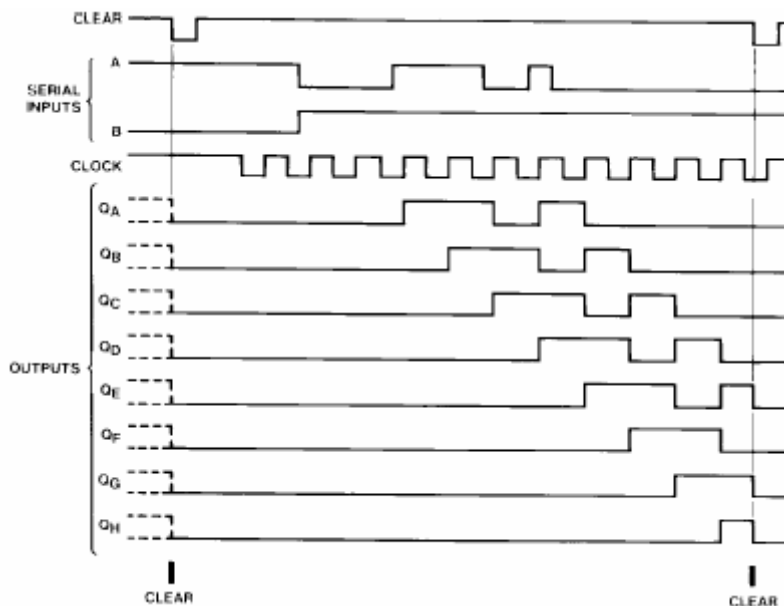


图 8.7 74LS164 时序图

#### 四、实验说明

包括保留寄存器在内，每一个 SPI 端口都有 8 个控制或状态寄存器。本实验中初始化时只需要初始化其中的 3 个，另外有 1 个作为状态寄存器用来查询 SPI 端口的状态，有 1 个作为数据寄存器作为发送或接收数据的缓存。

要用到的寄存器分别有控制寄存器 SPICR1、SPICR2，SPI 波特率寄存器 SPIBR，SPI 状态寄存器 SPISR，SPI 数据寄存器 SPIDR。具体在用的时候要注意是用 SPI1 或是 SPI2。相应的表示为：SPI\*DR，SPI\*SR……，\*表示 1 或 2。

SPICR1:

Bit 7	6	5	4	3	2	1	Bit 0
SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE

该寄存器主要用于初始化 SPI 端口的中断方式、模式选择、时钟极性、时钟相位等。

SPICR2:

	Bit 7	6	5	4	3	2	1	Bit 0
R	0	0	0	MODFEN	BIDIROE	0	SPISWAI	SPC0
W								
Reset:	0	0	0	0	0	0	0	0

该寄存器主要用于初始化 SPI 端口的双工模式、等待模式下时钟的状态等。

**SPIBR: SPI 波特率选择寄存器**

	Bit 7	6	5	4	3	2	1	Bit 0
R	0	SPPR2	SPPR1	SPPR0	0	SPR2	SPR1	SPR0
W								
Reset:	0	0	0	0	0	0	0	0

共分为两组，SPPR0-2 和 SPR0-2，最后的分频结果为原始时钟频率除以

$$(SPPR + 1) \cdot 2^{(SPR + 1)}$$

以上三个寄存器就是初始化时使用的寄存器。

**SPISR:**

	Bit 7	6	5	4	3	2	1	Bit 0
R	SPRF	0	SPTEF	MODF	0	0	0	0
W								
Reset:	0	0	1	0	0	0	0	0

**SPRF:** 状态为 1 表示可以接收或发送新的数据，为 0 表示传输尚未完成。

**SPTEF:** SPI 数据寄存器是否为空。状态为 1 表示数据寄存器为空，为 0 表示数据寄存器中有数据。

**SPIDR:** 数据寄存器。可以写入或读出。

	Bit 7	6	5	4	3	2	1	Bit 0
R	Bit 7	6	5	4	3	2	2	Bit 0
W								
Reset:	0	0	0	0	0	0	0	0

**五、预习要求**

1. 参考 MC9S12DP256 或 DG128 的 DATASHEET，掌握 SPI 各寄存器的使用方法。
2. 实验中将 SPI 作为主机使用，怎样正确地设置 SPI 的工作模式？
3. SPI 通讯的时钟波特率应设为多少？怎样计算实际得到的时钟波特率？
4. 芯片 74LS164 接收串行数据时，需要在哪些端口的输出信号时序上做准备工作？
5. 芯片 74LS165 发送串行数据时，需要做哪些准备工作？
6. 发送数据时，如何启动数据发送
7. 接收数据时，如何判断数据是否已接收完成？

## 实验9. I<sup>2</sup>C 总线实验

### 一、实验目的

1. 通过对 PCF8574 的读写操作，学习 I<sup>2</sup>C 总线的基本原理。
2. 熟悉 S12 系列单片机内部的 I<sup>2</sup>C 总线控制器。

### 二、实验任务

1. 使用查询方式，完成单片机 I<sup>2</sup>C 模块对 PCF8574 的读操作，将读取的 PCF8574 并口输入反映在 PORTB 的 LED 小灯上。
2. 使用查询方式，完成单片机 I<sup>2</sup>C 模块对 PCF8574 的写操作，将 PCF8574 的并口输出接到 LED 小灯上反映结果。
3. 根据芯片手册的中断程序流程图，编写 I<sup>2</sup>C 模块的中断程序，利用中断方式完成两个基本任务。

### 三、硬件资源

在实验板上，通过 I<sup>2</sup>C 总线 I/O 扩展芯片 PCF8574 将 MC9S12DP256 的 I<sup>2</sup>C 总线的的数据转换为并行的 I/O 口数据，依此为基础完成 I<sup>2</sup>C 的总线读写实验。

#### 1. I<sup>2</sup>C 总线写操作

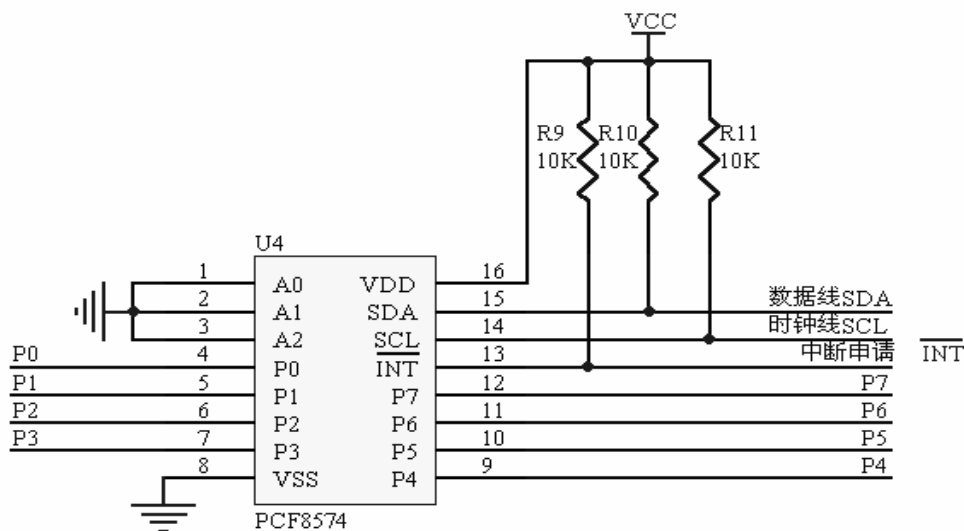


图 9.1 I<sup>2</sup>C 总线写操作实验电路

实验中，为了简单起见，将 3 个地址引脚 A0、A1、A2 直接接地。所以，在 I<sup>2</sup>C 总线中，该器件的地址为“0100000”。实验中，我们首先将 PCF8574 用作输出口，由于 PCF8574 可以直接驱动 LED，我们将 8 个并行 I/O 引脚直接与 8 个 LED 相连，由 I<sup>2</sup>C 总线接口来控制 LED 的亮灭。

## 2. I<sup>2</sup>C 总线读操作

在将 PCF8574 用于输入口时,将 8 个 I/O 口与板上的 8 位数字量开关相连,通过 PCF8574 将并行的数字量,通过 I<sup>2</sup>C 总线接口读入微控制器。

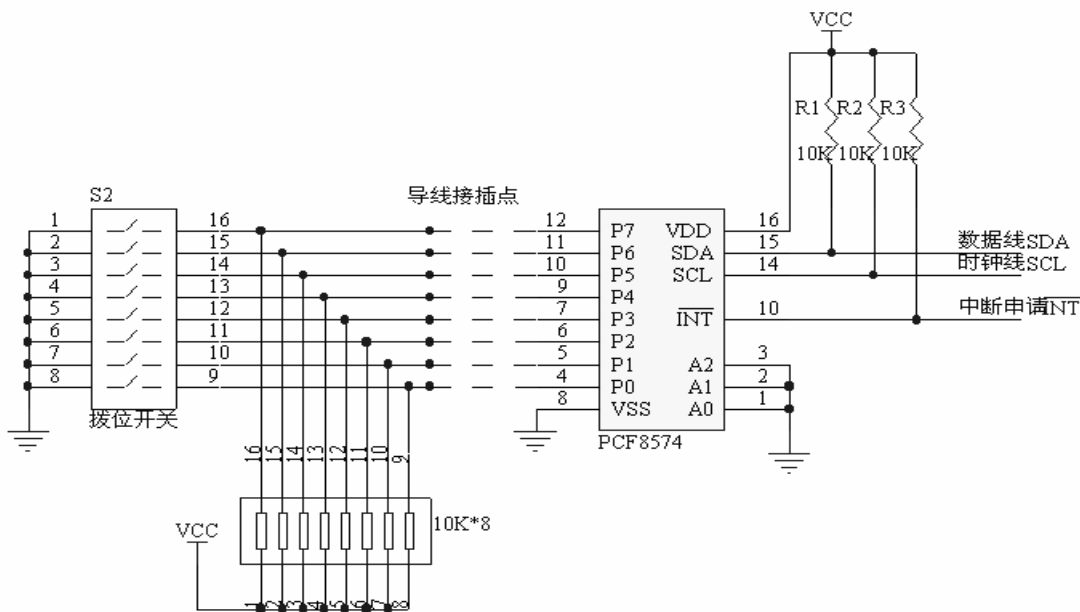


图 9.2 I<sup>2</sup>C 总线读操作实验电路

## 四、实验说明

本次实验涉及的是对一种串行总线的学习,因此预习中不仅要仔细阅读相关芯片手册,了解寄存器的设置,还要查阅相关资料,学习 I<sup>2</sup>C 总线的原理。

在本次实验中,需要事先了解的资料如下:

### 1. I<sup>2</sup>C 总线协议:

作为一种串行总线协议, I<sup>2</sup>C 的工作原理是比较复杂的,并不是区区几页讲义能够阐述清楚的,因此这部分内容,这里就不详细介绍了,读者可以翻阅相关书籍或者上网来查找相关资料。

### 2. I<sup>2</sup>C 总线扩展 I/O 接口芯片 PCF8574:

PCF8574/8574A 是 I<sup>2</sup>C 总线上广泛使用的扩展 I/O 芯片,可以将串行的 I<sup>2</sup>C 总线接口转成并行的 I/O 接口。PCF8574 与 PCF8574A 唯一区别在地址上, PCF8574 的器件地址是“0100”,而 PCF8574A 的器件地址是“0111”。

PCF8574 (A) 的特性概括如下:

- 工作电压在 2.5V 到 6V 之间
- 备用电流小于 10μA
- 中断输出引脚为开漏结构
- 8 为准双向 I/O 口,可直接驱动 LED
- 3 个地址引脚,最多可挂接 8 个同类器件(加上 PCF8574A 可达 16 个)

在 I<sup>2</sup>C 总线上, PCF8574 (A) 总是处于被控节点状态,所有读写操作只能由主控器件进行控制。

当主控器件进行写操作时, PCF8574 (A) 工作在输出状态。由 I<sup>2</sup>C 总线主控器件发送的数据将在 PCF8574 (A) 应答位之后,在时钟信号的低电平周期内出现在 I/O 端口。

当主控制器进行读操作时，PCF8574 (A) 工作在输入状态。在 I<sup>2</sup>C 总线主控制器发出寻址字节后，在 PCF8574 (A) 的应答位所对应的时钟脉冲的上升沿将 I/O 口的电平锁存到输入锁存寄存器中，由主控制器读取。

PCF8574 (A) 的中断申请引脚-INT 在输入口的电平发生跳变时发出中断申请，在主控制器对 PCF8574 (A) 进行读写操作时复位。读操作在 SCL 的上升沿复位，写操作在 SCL 的下降沿复位。

#### 使用注意事项:

PCF8574(A)的 I/O 口为准双向 I/O 口,作为输入口时,应事先由主控制器对 PCF8574 (A) 进行读操作,使输入锁存器置 1。

中断申请引脚-INT 为开漏结构,使用时需接上拉电阻。由于是“线与”逻辑,可同时挂接多个中断申请引脚。

PCF8574 (A) 最多可在一个 I<sup>2</sup>C 总线中使用 16 个同类型 I/O 扩展器件。但是 PCF8574A 的器件地址与 LED 驱动器 SAA1064 的器件地址相同,如果系统中有 SAA1064 时,要注意 PCF8574A 的地址不能与 SAA1064 重叠。

### 3. 单片机内部 I<sup>2</sup>C 模块

MC9S12DG128 内部集成了一个 I<sup>2</sup>C 总线接口模块,简要介绍如下:

#### 1. I<sup>2</sup>C 模块特点:

- 兼容标准 I<sup>2</sup>C 总线协议
- 支持总线多主机操作
- 可编程设置 256 种串行总线时钟
- 可以软件选择是否使用应答位
- 以字节为单位的数据传输中断
- 总线仲裁失败可以产生中断,并且切换运行模式
- 处在从机模式时,可以地址认证成功中断
- 可以生成/监测时序中起始和结束信号
- 可以重复产生起始信号
- 可探测总线空闲

#### 2. I<sup>2</sup>C 模块相关引脚:

SCL ----I<sup>2</sup>C 时钟总线

SDA ----I<sup>2</sup>C 数据总线

#### 3. I<sup>2</sup>C 模块寄存器:

I<sup>2</sup>C 模块主要有如下 5 个寄存器。

地址	寄存器	访问
0xE0	IBAD—总线地址寄存器	可读/写
0xE1	IBFD—总线频率寄存器	可读/写
0xE2	IBCR—总线控制寄存器	可读/写
0xE3	IBSR—总线状态寄存器	可读/写
0xE4	IBDR—总线数据寄存器	可读/写

寄存器的具体使用请参见芯片手册“S12IICV2.pdf”。

## 五、预习要求

1. 掌握主从机工作的流程，熟悉 I<sup>2</sup>C 总线上数据传输的时序；熟悉一个完整数据包的传输格式；了解总线寻址，总线仲裁。
2. 了解 PCF8574 芯片各引脚定义，查找 PCF8574 的时序要求，以此为根据，计算总线频率寄存器（IBFD）的取值要求。
3. 仔细阅读单片机的 I<sup>2</sup>C 模块手册，掌握各寄存器的使用，熟悉 I<sup>2</sup>C 总线读写的流程。

## 实验10. PWM 模块实验

### 一、实验目的

1. 学习使用 PWM 模块。
2. 了解 PWM 波用作 D/A 转换的原理。

### 二、实验任务

使用单片机内部 PWM 模块调制产生不同脉宽的方波，观察方波经过低通滤波器的 D/A 转换效果。

### 三、硬件资源

PWM 波用作 D/A 输出时，最简单的方法就使 PWM 波通过一个一阶低通滤波器（如图 R1、C1 构成一阶低通滤波器），为了使电压保持，要是输出阻抗尽可能的大，为此，在低通滤波器后再加上一级电压跟随器（图中由 LM324 和 R2 构成）。

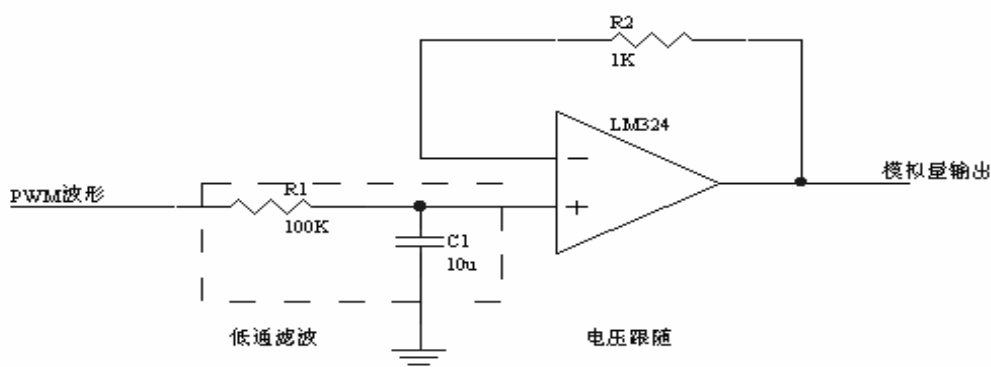


图 10.1 PWM 用作 D/A 转换接口电路

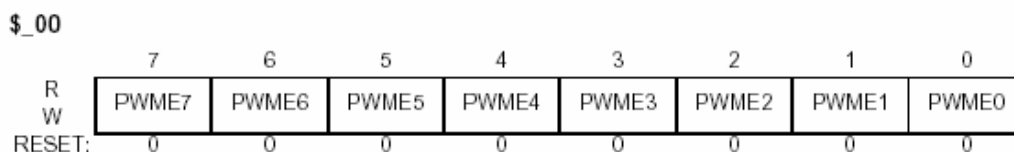
得到的模拟量就和 PWM 波的占空比有关。比如，如果占空比是 1:1，那么得到的电压值就是 1/2 的最大输出电平（5v），即 2.5v。

实验板上为用户提供了这样一个这样的接口电路。同时，为了使用户能够更加直观的看到模拟电平的变化，又提供了一个串有电阻的发光二极管，用户可将该电平接到发光二极管上。当该电平大于 2v 时，随着电平的增高，发光二极管的亮度就会随之增加。

#### 四、实验说明

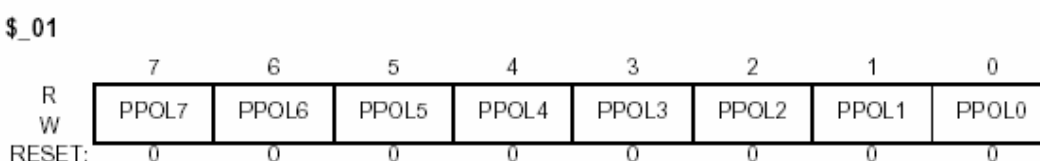
PWM 模块共有 28 个寄存器，其中 8 个为系统保留寄存器，具体介绍如下：

##### PWM 启动寄存器 (PWME)



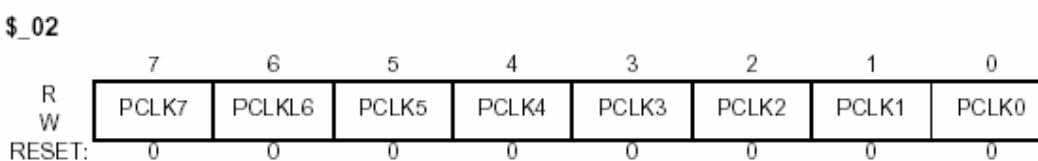
本寄存器的 8 个 bits 分别用来开关 8 路 PWM 的通道。

##### PWM 极性寄存器 (PWMPOL)



本寄存器的 8 位 bits 分别用来设定 8 路 PWM 通道输出波形的起点电平。

##### PWM 时钟选择寄存器 (PWMCLK)



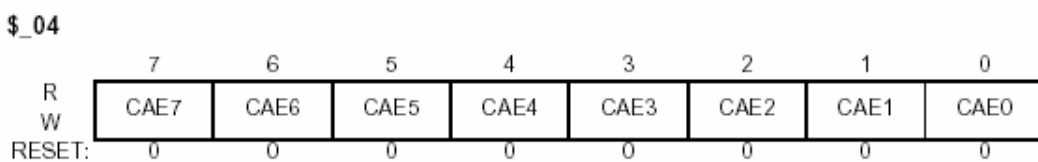
本寄存器的 8 位 bits 分别用来设定 8 路 PWM 通路的时钟来源。

##### PWM 预分频寄存器 (PWMPRCLK)



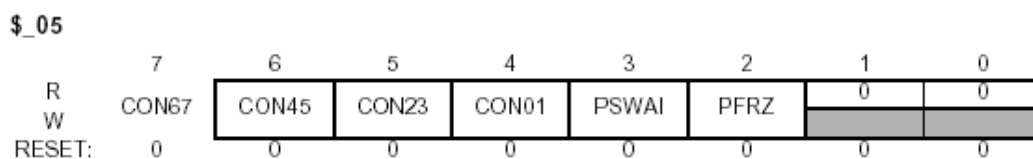
本寄存器用来设定 ClockA 和 ClockB 的预分频因子。

##### PWM 波形对齐寄存器 (PWMCAE)



本寄存器的 8 位 bits 用来分别选择 8 路 PWM 通路输出是中心对称还是左对称。

PWM 控制寄存器 (PWMCTL)



本寄存器提供 PWM 模块操作时的几个控制位。

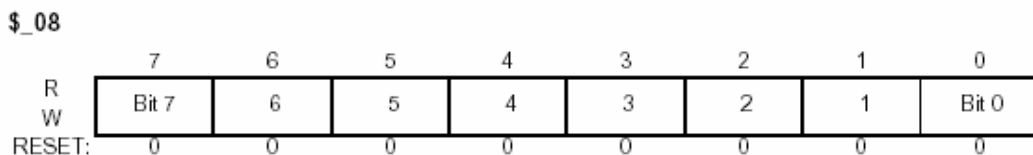
保留寄存器 (PWTST)



保留寄存器 (PWMPRSC)



ClockA 分频寄存器 (PWMSCLA)



本寄存器用来设定 ClockSA 的频率：

$$\text{Clock SA} = \text{Clock A} / (2 * \text{PWMSCLA})$$

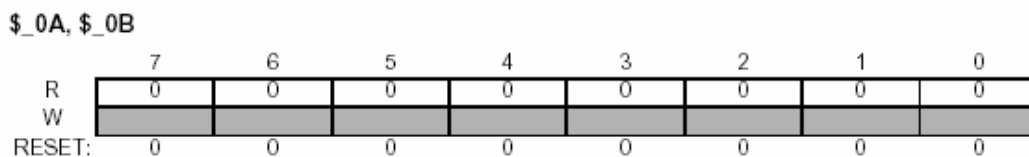
ClockB 分频寄存器 (PWMSCLB)



本寄存器用来设定 ClockSB 的频率：

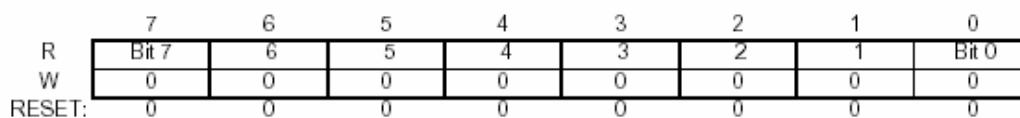
$$\text{Clock SB} = \text{Clock B} / (2 * \text{PWMSCLB})$$

保留寄存器(PWMSCNTx)



PWM 通道计数寄存器 (PWMCNTx)

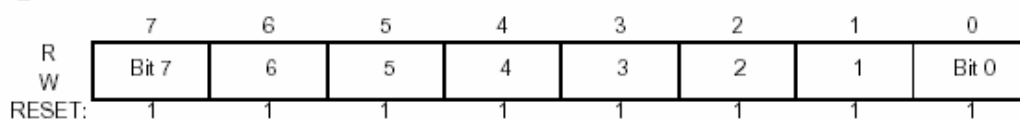
- \$\_0C = PWMCNT0
- \$\_0D = PWMCNT1
- \$\_0E = PWMCNT2
- \$\_0F = PWMCNT3
- \$\_10 = PWMCNT4
- \$\_11 = PWMCNT5
- \$\_12 = PWMCNT6
- \$\_13 = PWMCNT7



此 8 个寄存器分别为 8 个通道的波形输出计数器。

PWM 通道周期寄存器(PWMPERx)

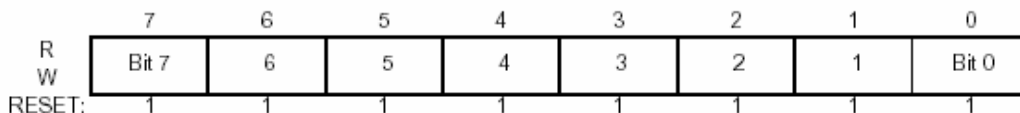
- \$\_14 = PWMPER0
- \$\_15 = PWMPER1
- \$\_16 = PWMPER2
- \$\_17 = PWMPER3
- \$\_18 = PWMPER4
- \$\_19 = PWMPER5
- \$\_1A = PWMPER6
- \$\_1B = PWMPER7



此 8 个寄存器分别为 8 个通道设定方波的周期。

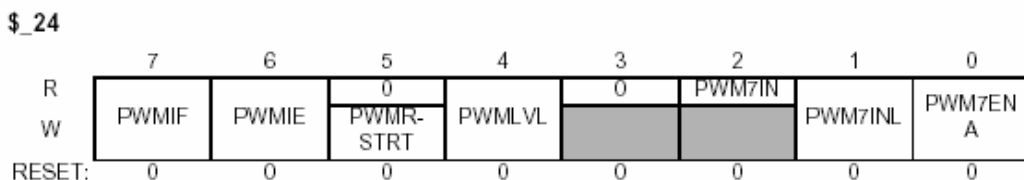
PWM 通道脉宽寄存器(PWMDTYx)

\$\_1C = PWMDTY0  
 \$\_1D = PWMDTY1  
 \$\_1E = PWMDTY2  
 \$\_1F = PWMDTY3  
 \$\_20 = PWMDTY4  
 \$\_21 = PWMDTY5  
 \$\_22 = PWMDTY6  
 \$\_23 = PWMDTY7



此 8 个寄存器分别为 8 个通道设定脉宽。

PWM 关闭寄存器(PWMSDN)



本寄存器用来设定紧急情况下 PWM 的自关闭功能。

注:

1. 以上寄存器地址均为寄存器的相对地址，需加上模块基地址才可使用。
2. 寄存器具体介绍，请参照 S12PWM\_8B8CV1.pdf。

五、预习要求

- (1) 仔细阅读相应芯片手册，思考以下问题：
  - 1.如何根据脉宽调制周期要求设定各分频寄存器？
  - 2.PWM 模块波形输出有几种对齐方式？掌握各种对齐方式。
  - 3.如何设定输出波形的脉宽比例？
- (2) 设计软件流程，给出设计代码。

## 实验11. MSCAN 模块的自循环模式

### 一、实验目的

初步了解 CAN 总线，熟悉其通信协议，并掌握 S12 单片机 MSCAN 模块的使用方法。

### 二、实验任务

利用 MSCAN 模块的 Loop Back 模式，自发自收数据。

### 三、实验说明

#### 1. 报文滤波

CAN 总线允许用户设定一组标识符放在报文中，接收节点可以根据报文的标识符来决定是否接收该报文，我们称之为“报文滤波”。

CAN 总线中传送信息是以打包方式传送的，单元叫做报文或帧，一共有 4 中不同的帧格式：数据帧、远程帧、出错帧、超载帧。本实验中用户需要了解数据帧的帧结构（图 1）。

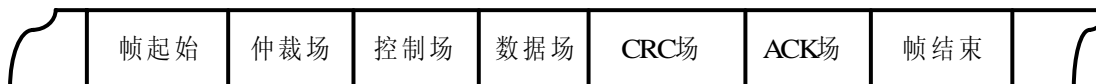


图 11.1 数据帧结构

其中仲裁场包含一段标识符，在最新的 CAN2.0B 的协议中根据标识符的长度将数据帧分成标准帧和扩展帧，标准帧中的标识符长度为 11 位，而扩展帧的标识符长度达到 29 位。编程时，用户可以设定选用标准帧或是扩展帧，设定的寄存器在发送缓冲区中（IDR1）。

接收节点需要预先设定可以接收的标识符（CANIDAR0~7），根据标识符长度不同可以设定为 2 个 32 位的标识符、4 个 16 位的标识符、8 个 8 位的标识符或是直接将该滤波器关闭。关闭后，节点就不再接收任何帧了。用户也可以选择忽略标识符滤波，通过设定标识符屏蔽寄存器(CANIDMR0~7)，可以按位选择哪一位可以忽略。

#### 2. 发送缓冲区和接收缓冲区

在发送缓冲区的设计上，MSCAN 内置了三个具有局部优先级的发送器，某一时刻只有一个缓冲区处于前台状态，当一次发送结束之后，模块会通过一个特定的算法（见教材 285 页）来选定下面将哪一个缓冲区推入前台。这样可以允许 MSCAN 总是将优先级最高的数据帧先发送出去。

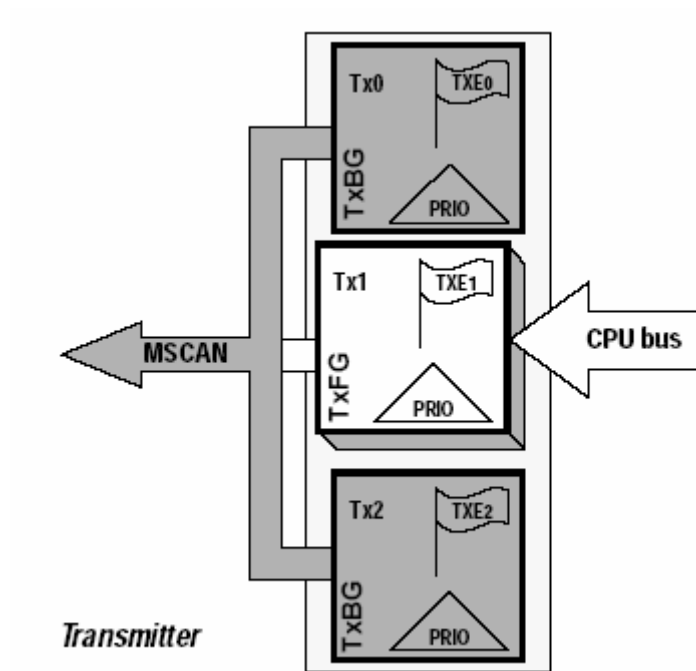


图 11.2 MSCAN 发送缓冲区结构

MSCAN 的接收缓冲区是一个 4 级先进先出 (FIFO) 缓冲区，四级缓冲共享一个地址。

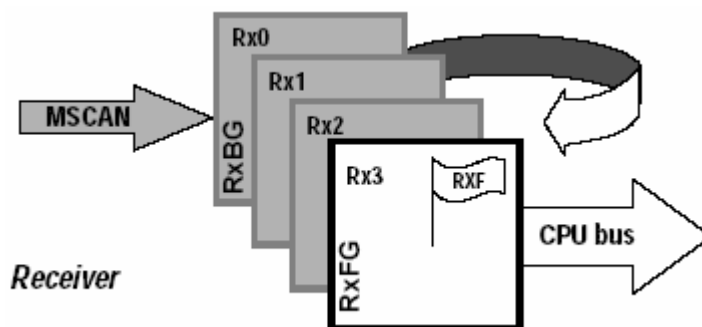


图 11.3 MSCAN 接收缓冲区结构

### 3. 自发自收模式

为了验证发送器和接收器功能正常，MSCAN 设置了一个自发自收模式 (LOOP BACK)，在这种模式下，发送管脚和接收管脚自动在芯片内部短接，发送的内容直接送到接收器，实验中正是使用了这种模式来验证发送和接收程序的正确。需要说明，这种模式除了没有物理层驱动电路外，与实际应用情况完全一样。

#### 四、实验步骤

在开始接收和发送之前，首先要对 MSCAN 模块初始化，MSCAN 模式中提供了一个初始化模式，有些寄存器只能在初始化模式下才能写。下面是初始化流程图：

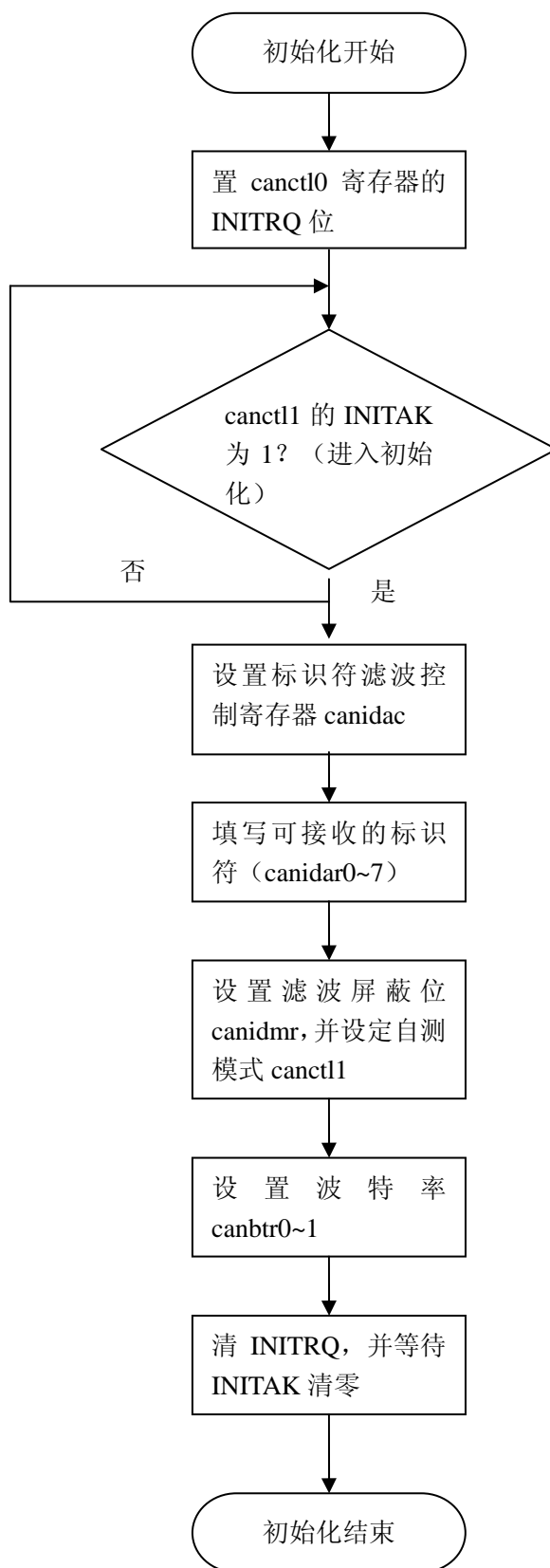


图 11.4 MSCAN 初始化流程图

初始化工作做完，就可以准备发送数据了。由于是自发自收，用户需要在发送数据前先设定接收的模式，本实验中采用中断接收方式。发送数据的过程如下面的流程图所示：

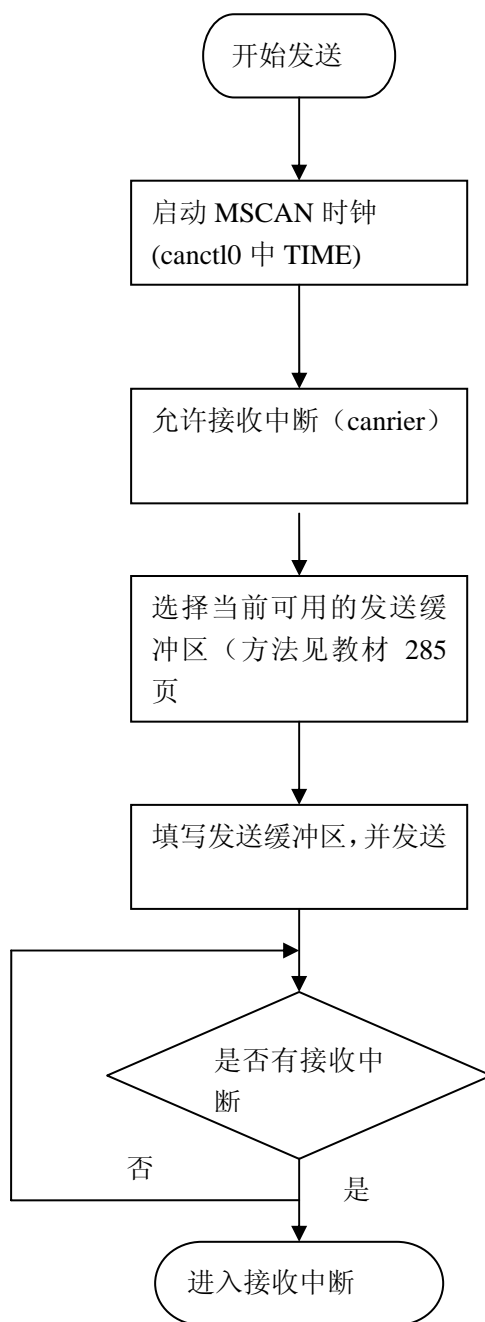


图 11.5 MSCAN 发送流程图

用户收到数据后，可以打印到串口上，用以验证和调试。

#### 四、预习要求

(1) 阅读教材 9.2.3 一节，思考如下问题：

1. 同为总线型接口，CAN、SPI 和 IIC 都是如何辨别属于自己节点的数据的？
2. CAN 总线共有哪几种帧格式，它们各自的作用是什么？
3. 为什么 MSCAN 的发送器采用三个缓冲区的设计？

(2) 编写程序

## 实验12. Keyboard 实验

### 一、实验目的

1. 了解键盘结构，学习扫描法读取按键的方法。
2. 了解用 8 个 I/O 设计更多按键的键盘的方法。

### 二、实验任务

在实验板上设计并连接小键盘接口的连线，识别小键盘上按下的各键，并将其对应的按键状态用八个发光二极管来显示。

### 三、相关硬件资源

参考如下电路图：

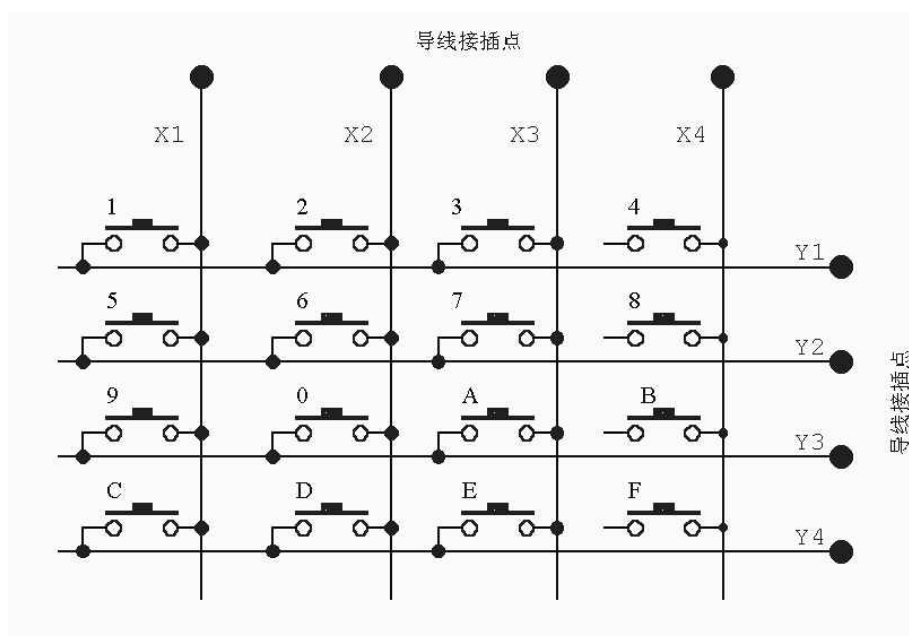


图 12.1 4x4 行列式键盘原理图

PORTA 0->3 接 Y1->Y4; PORTA 4->7 接 X1->X4; 可用 PORTB 接的八个发光二极管作为按键状态指示灯（即哪个键被按下）。

### 四、实验说明

1. 识别键盘上的闭合键，通常采用行扫描法或行反转法。

为描述方便，将与行线相接的端口称为行线端口，与列线相连的端口称为列线端口。

2. 采用行扫描法识别闭合按键时，使行线端口工作在输出方式，列线端口工作在输入方式。

先将键盘的第 0 行输出低电平，其余各行输出高电平，然后读取列值，如果列值中某位为低电平，则表明行列交叉处的键被按下，若列值全为高电平则继续扫描下一行，知道扫描到某列值有低电平或扫描完所有的行为止。可据扫描的结果判断是否有键按下和按下的时哪一行哪一列的交点处的按键。

3. 采用行反转法识别闭合按键时，需在识别过程中改变行线端口和列线端口的工作状态。

首先使行线端口工作在输出方式，列线端口工作在输入方式，通过输出口向行线全部输出低电平，然后通过输入口读取列线值，如果此时有某一键按下，则必定会有某一列线值为低电平。然后改变两端口的工作状态，使行线端口工作在输入方式，列线端口工作在输出方式，将前一次读得的列线值通过列线端口输出，再通过行线端口读取行线值，其中闭合键所在行线上的值必定为低电平。当一个键被按下时，必定可读取到一对唯一的列值和行值。据这一对列值和行值就可以判断按下的是哪一行哪一列交点处的键。

4. 为消除由于键内部的机械簧片再键按下和释放时产生的抖动，保证 MCU 对键的一次闭合仅作一次处理，可在程序中加入一定延时，延时到键状态稳定后再读取键的状态。

## 五、预习要求

1. 将 PORTA 0->3 接 Y1->Y4；PORTA 4->7 接 X1->X4；PORTB 接八个发光二极管，请给出相应的程序。
2. 如果 PORTA 0->3 接到 Y4->Y1 上，程序应该怎么写？
3. 思考如何利用 8 个 IO 口设计多于 16 个键的方法。

在本实验中给出一种 24 键的参考设计：24 键的键盘中，其中 16 个键的连线方法为 PORTA 0->3 接 Y1->Y4；PORTA 4->7 接 X1->X4；剩余的 8 个按键，每个按键的一个节点接地，另外一个节点分别接 PORTA 0->7。在程序设计时，采用行反转法识别闭合按键。请思考如何用程序来实现。

## 六、注意事项

1. 严禁带电操作，接插导线时须断电，并注意防止导线堵塞插孔。
2. 不要急于求成，当实验结果不对时，仔细分析程序的流程是否合理。

## 实验13. 动态数码管的显示

### 一、实验目的

掌握动态数码管的使用方法。

### 二、实验任务

在 8 位动态数码管上显示任意小数。

### 三、实验说明

关于动态数码管的原理请参阅教材 293~294 页。需要说明 8 位数码管的第 8 位是小数点，显示小数时，只需将个位的小数点点亮就可以了。

由于数码管的管脚很多，如果直接用 I/O 口驱动，会比较麻烦，建议用户配合 SPI 的输出功能来完成数码管实验，具体原理图参见教材 269 页。

源程序可参考教材 269~272 页。

### 四、实验预习

1. 芯片 74HC07 的作用是什么？
2. 如何理解“动态数码管”中“动态”的概念？

## 实验14. Flash 在线编程

### 一、实验目的

学习并掌握 S12 单片机 Flash 编程的一般方法。理解在线编程的概念，并掌握其原理。

### 二、实验任务

将事先保存在 RAM 中的数据复制到 Flash 中，源地址和目标地址可以任意设定。

### 三、实验说明

#### 1. 与 FLASH 编程相关的寄存器

FLASH 操作时钟分频寄存器 FCLKDIV (\$0100)

FLASH 配置寄存器 FCNFG (\$0103)

FLASH 命令寄存器 FCMD (\$0106)

FLASH 状态寄存器 FSTAT (\$0105)

FLASH 保护寄存器 FPROT (\$0104)

FLASH 加密寄存器 FSEC (\$0101)

#### 2. FLASH 在线编程的概念及流程

所谓“在线编程”，就是指带有 Flash 的单片机用片内程序去擦写自身 Flash，换句话说，单片机可以自己“更新”自己。由于之前的 ROM 并不能轻易改写，程序一旦运行，整个 ROM 就只读了，所以，“在线编程”这一概念是随着 Flash 的普及而产生的。有了在线编程功能，单片机可以自己“开发”自己，开发的手段就灵活多了。

Flash 的编程，或者叫做写入，是以一个字（Word）为单位。图 14.1 给出了整个写入的流程。

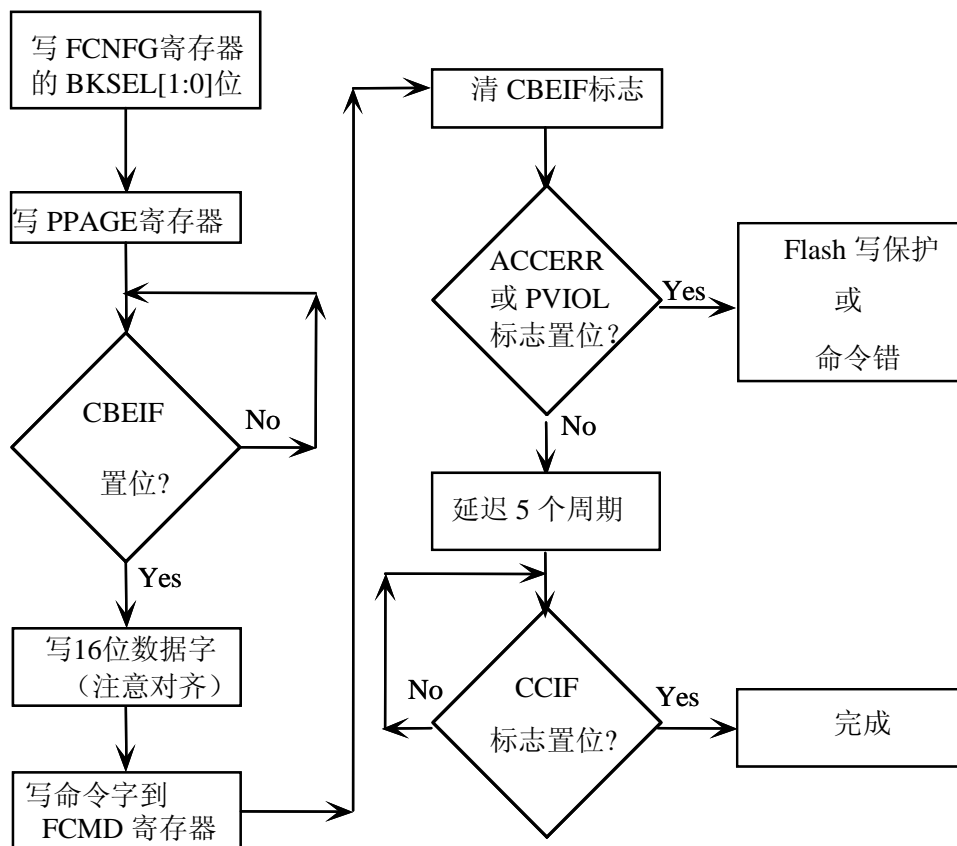


图 14.1 Flash 写入流程图

有关 Flash 编程的具体流程，可以参见教材 127~128 页。

需要特别注意，由于在 FLASH 的擦除和写入过程中，FLASH 是不能读的，故擦除和写入 FLASH 的程序要放在 RAM 中，也就是说，在 FLASH 的擦除或写入之前，要把擦除或写入的可执行代码复制到 RAM 中去，并让程序在 RAM 中执行。

#### 四、预习要求

阅读 4.13 节，明确 FLASH 编程的原理和方法，编写程序。

## 实验15. uC/OS-II 在 S12 上的移植

### 一、实验目的

掌握在 S12 单片机中移植 uC/OS-II 的方法，进而熟悉 uC/OS-II 在各类型 CPU 上移植的一般步骤。

### 二、实验任务

将 uC/OS-II 移植到 S12 单片机中，实现简单的任务调度（控制 PORTB 上不同的 LED 闪烁）。

### 三、预习要求

（1）阅读课本第 6 章和《嵌入式实时操作系统 uC/OS-II》第 13、14 章，明确 uC/OS-II 的代码结构和 uC/OS-II 移植的一般方法。

（2）理清楚如何测试 uC/OS-II 移植是否成功？uC/OS-II 的任务调度如何体现出来？为实现正确的任务调度需要做哪些设置？

（3）思考：按照课本第 6 章和配套光盘上的移植步骤移植 uC/OS-II 有何风险（注意 Codewarrior 的“智能抛弃”功能）？

## 实验16. 综合实验 —— uC/OS-II 下多 I/O 任务实现

### 一、实验目的

在掌握 uC/OS-II 移植方法的基础上实现自己的多任务系统，对 uC/OS-II 的任务管理、进程通信等形成全面的认识，初步学会用 uC/OS-II 来管理各 IO 模块。

### 二、实验任务

将之前做过的 IO 外设模块实验程序做成任务的形式，用 uC/OS-II 来管理各个外设（从如 GPIO, SCI, SPI, IIC, PWM, ADC, Keyboard, CAN 等中任选若干），注意合理利用信号量、邮箱、消息队列等来实现进程间的通信和任务管理。

比如可做一个仿真手机，着重解决如下问题：

1. 收发短信时有呼入信号如何处理？
2. 接听电话时有呼入信号如何处理？
3. 有没有可能做一个多人电话会议系统？（即可以依次发言，参与的每个人都可以听到发言人讲的话）
4. 怎样才能让手机显得功能更强大？

### 三、预习要求

（1）阅读《嵌入式实时操作系统 uC/OS-II》第 7~11 章，明确 uC/OS-II 中用于进程通信和同步的各种方法。

（2）设计软件流程，给出设计代码。