微机与接口技术实验

北京大学信息科学技术学院

2018 年 2 月

目录

实验一　单片机开发系统的使用

1.1　实验目的

1. 初步了解 MCS-51 系列单片机及其存储器构成、对存储器的管理方法；
2. 初步了解 MCS-51 系列单片机指令系统；
3. 熟悉单片机开发系统及开发工具的基本操作方法；
4. 掌握系统调试和运行程序的基本方法。

1.2　实验原理

1.2.1　MCS-51 简介

单片机是单片微型机的简称，是由单块集成电路芯片组成的处理器系统。单片机内部包含有计算机的基本功能组件，如 CPU、内存、总线控制器、计数器和 IO 等外设模块。由于单片机的集成度高，使用它可以方便的组成一个控制系统，在实际应用中使用非常广泛。MCS-51 系列单片机是 Intel 公司在 1980 年推出的产品，虽然它只是一个 8 位的单片机，但在当时属于技术比较领先的产品，广泛的被工业界所接受，直到今天依然有很多的应用领域。

存储器是单片机系统的重要组成部分，按照存储内容和存取方式不同，单片机系统的存储器可分为两类。一类是程序存储器，用于存放程序代码（有时还有一些常数数据）。通常情况，程序代码和常数数据只能被读取，而不能被任意改写，因而程序存储器是只读（ROM）的。另一类是数据存储器，用于存放程序运行时的工作变量和数据，如原始数据、运算中间/最终结果、数据暂存/缓冲、标志字节/位等，有时也用于存放待调试的程序。数据存储器中的数据可根据需要写入或读出，因而数据存储器是随机存取（RAM）的。

按照存储器物理位置的不同，单片机系统的程序存储器和数据存储器都可有片内和片外之分。片内外程序存储器地址空间统一编址。

MCS-51 系列单片机按照片内部 ROM 配置种类可分为：无 ROM 的 8031 型、有 ROM（只可一次性写入）的 8051 型、有 EPROM（紫外线擦除，可重写入）的 8751 型，以及有 EEPROM（电可擦除，可重写入）的 8951 型。

MCS-51 系列单片机按照片内 ROM、RAM、定时器/计数器、中断源数量等的不同，可分为 51 和 52 两个子系列：就 ROM 而言，51 子系列的有 4K 字节的片内 ROM；而 52 子系列的有 8K 字节的片内 ROM。比较新的 MCS-51 产品在存储器容量和外设组成上都有很多扩展，但受限于 51 单片机的指令集构造，扩展的空间不大，在使用的时候需要根据芯片的数据手册了解扩展部分的使用方法。

MCS-51 系列单片机的寄存器主要有以下几个：

1. 程序计数器 PC（Program Counter）是一个 16 位的寄存器，用来记录下一条需要执行指令的内存地址，当系统复位的时候，PC 的值为 0，其取值范围最大是 0FFFFH，因此程序存储器的寻址范围为 64K。
2. 数据指针 DPTR 是一个 16 位的寄存器，它有两个 8 位寄存器 DPH 和 DPL 组成。DPTR 经常被用来作为间接寻址的指针来对程序存储器或数据存储器进行寻址。

3. 累加器 A（Accumulator）又被记作 ACC，是一个通用 8 位寄存器，可以在很多指令中用来做操作数或保存计算结果。
4. 累加器 B 是专门为乘法和除法设置的寄存器，在除法运算中用来保存乘数或者除数。
5. 工作寄存器 R0∼R7 是程序中常用的寄存器，可以作为指令的操作数或者间接寻址的指针等，它们实际保存在单片机的 RAM 中，具体位置由 PSW（见下）决定
6. 堆栈指针 SP 是一个 8 位寄存器，用来记录堆栈的栈顶位置，系统复位的时候 SP 的值为 07，程序在开始的时候可以给 SP 设置初值，压栈（PUSH）的时候 SP 自动加一，弹栈（POP）的时候 SP 自动减一。
7. 程序状态字 PSW（Program Status Word）是一个 8 位标志寄存器，具体的定义可以参考表 1.1。

表 1.1: 程序状态字

D7	D6	D5	D4	D3	D2	D1	D0
CY	AC	F0	RS1	RS0	CV		P

CY 进位标志在发生进位和借位的时候被设置，其它算数运算被清零，在有些移位指令中也会改变它的值，程序中通过 C 来访问这个位。

AC 辅助进位标志。如果在加法和减法过程中，低四位向高四位发生进位或者借位，AC 位被设置，否则被清零。

F0 是用户标志位，可由用户来使用

RS1/RS0 寄存器选择位，用来设置工作寄存器的保存位置。

OV 溢出标志位，如果算数运算发生进位、借位或者溢出（超出 ACC 的表示范围）则 OV 位被设置，否则该位被清零。

P 奇偶校验位，如果累加器中 1 的个数为奇数，则 P 位被设置，否则该位被清零。

MCS-51 子系列单片机片内 RAM 共有 256 字节，其低 128 字节全部是用户可读写的，高 128 字节区域为专用寄存器区。低 128 字节的区域都可以按字节为单位进行读写访问。如图 1.1 所示，从 00H∼1FH 字节单元共 32 个字节的区域常可以用作工作寄存器，字节地址为 00H∼07H、08H∼0FH、10H∼17H、18H∼1FH 四个区域分别称为 0、1、2、3 工作寄存器区。每个工作寄存器区的 8 个字节单元分别对应 R0、R1、…R7 共 8 个工作寄存器。每个 R 寄存器可以根据程序状态字寄存器 PSW（如表 1.1）中 RS1/RS0 的设置 00、01、10、11，对应某一工作寄存器区中的一个字节单元。例如当 RS 设置为 00 时，工作寄存器 R0∼R7 对应为 00H∼07H 的字节单元。i

内部数据的地址从 20H∼2FH 字节单元共 16 个字节的区域，可以按位寻址，此区域共有从 00H、01H、…7EH、7FH 共 128 个（16 × 8）可寻址位，使用 SETB 和 CLR 指令访问。

内部数据的地址从 80H∼0FFH 为 SFR（SPecial Function Register）区域，用来保存单片机的特殊寄存器，例如累加器 ACC 就保存在 0E0H 地址上，在实际使用中一般只使用 SFR 的名称来访问，而不必记忆他们的地址。

对于不同的存储位置，MCS-51 采用不同的读写指令，其中访问片内 RAM 使用 MOV 指令，访问片外数据存储器（可读写）的指令用 MOVX，访问程序存储器（只能读）的指令用 MOVC。具体的指令语法可以参考附录中的 MCS-51 系列单片机指令系统。

1.2.2 C8051F020 简介

C8051F020 系列器件使用 Silicon Labs 的专利 CIP-51 微控制器内核。CIP-51 与 MCS-51 指令集完全兼容, 可以使用标准 803x/805x 的汇编器和编译器进行软件开发。

CIP-51 内核具有标准 8052 的所有外设部件, 包括：

- 5 个 16 位的计数器/定时器、

- 两个全双工 UART、256 字节内部 RAM、
- 128 字节特殊功能寄存器 (SFR) 地址空间、
- 8/4 个字节宽的 I/O 端口

CIP-51 采用流水线结构, 与标准的 8051 结构相比指令执行速度有很大的提高。
C8051F020 增加的功能：

- 扩展的中断系统向 CIP-51 提供 22 个中断源 (标准 8051 只有 7 个中断源), 允许大量的模拟和数字外设中断微控制器。
- MCU 可有多达 7 个复位源: 一个片内 VDD 监视器、一个看门狗定时器、一个时钟丢失检测器、一个由比较器 0 提供的电压检测器、一个软件强制复位、CNVSTR 引脚及 /RST 引脚。
- MCU 内部有一个独立运行的时钟发生器，在复位后被默认为系统时钟。如果需要，时钟源可以在运行时切换到外部振荡器，外部振荡器可以使用晶体、陶瓷谐振器、电容、RC 或外部时钟源产生系统时钟。

C8051F020 的内存布局情况参考图 1.1。可以看出其内存分成三部分，程序存储器是 FLASH 结构，有 64K 的寻址空间。内部数据区共有 256 字节的寻址空间，当作为 RAM 使用时，低 128 字节既可以直接寻址也可以间接寻址，而高 128 字节只能间接寻址（与直接寻址的 SFR 区域相区别）。外部数据区有 64K 的寻址空间，其中片内实现了 4K 字节，可以作为 RAM 使用（只能使用 MOVX 指令访问，缺省是打开状态，可以通过特殊寄存器的设置来使用外部的地址空间）。

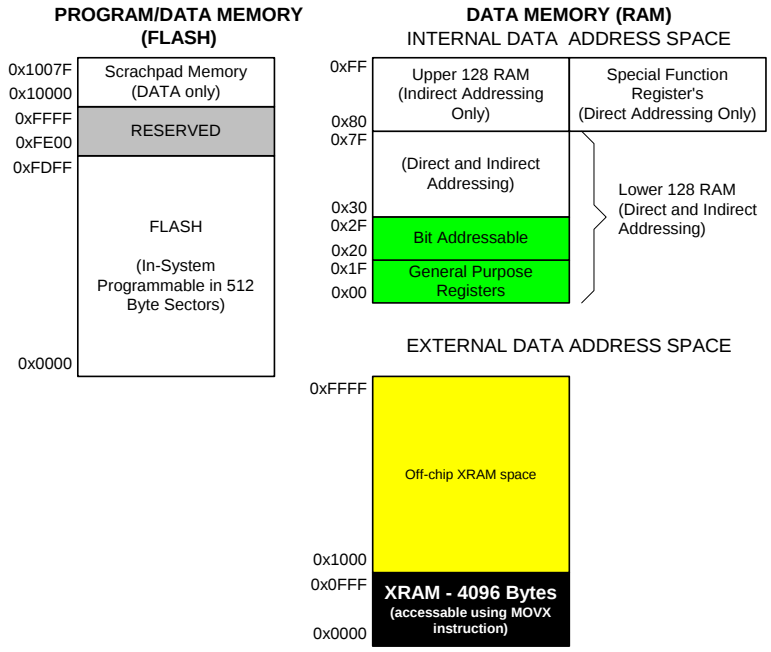


图 1.1: C8051F020 的内存布局

1.3 实验内容

1.3.1 Keil 软件使用

软件的基本使用过程如下：

1. 打开 Keil uVision4 软件，熟悉软件的界面操作

2. 在 Project 菜单中选择 New uVision Project 建立新的工程文件。在选择器件的界面中选择 “Silicon Laboratories, Inc.” 的 C8051F020。在询问是否加载缺省的启动代码命令中选择否。
3. 使用 File 菜单中的 New 命令建立新的源代码文件，保存为扩展名为 A51 的汇编文件，输入汇编代码
4. 右击 “Source Group 1” 选择添加文件到这个组，并把上面保存的文件加入进来，然后选择 Project 菜单里面的 Build Target 对代码进行编译，如果有错误就要修改代码，直到编译通过
5. 鼠标选择 “Target 1”，然后在 Project 菜单中可以修改工程的设置（Options for Target ‘Target 1’），注意在 Debug 页上面可以选择是使用模拟器（Use Simulator）还是使用硬件调试。如果使用硬件调试，要选择 “Silicon Labs C8051Fxxx Driver”，并在驱动设置中选择 “USB Debug Adapter”
6. Project 菜单中选择 Build 来对项目进行编译，注意软件提示窗口中的信息，如果显示 “0 Error(s)” 就表示编译没有问题，可以下载调试了
7. Debug 菜单中的 “Start/Stop Debug Session” 可以用来启动调试和退出调试。启动调试检测程序是否正常执行，如果在调试状态对代码进行了修改需要退出调试状态重新编译代码后再进入调试
8. Debug 菜单中的 “Run” 可以启动程序的运行，“Stop” 命令用来停止运行，“Step” 命令可以用来进行单步调试，“Insert/Remove Breakpoint” 命令用来设置断点，这些都是常用的调试手段。
9. View 菜单中的 Memory Windows 可以打开最多四个内存的观察窗口，在地址位置输入 0x500 就可以查看相应地址的数据；对于不同的存储空间，可以使用不同的前缀。例如 “c:0x500” 表示代码空间；“d:0x50” 表示内部 RAM；“x:0x50” 表示外部存储器。双击相应的数据位置可以对数据进行修改

软件使用过程中的一些提示：

1. Keil 软件的中的菜单命令很多都有快捷键和快捷按钮，在使用中要灵活掌握，提高实验的效率
2. 当前的调试模式在状态栏会有提示，使用模拟器方式比较快捷，但在涉及外设的实验中无法得到演示效果
3. 编写程序要具有良好的缩进，以提高可读性

1.3.2 两个十进制数相加

实现两个四位十进制数（8421BCD 码表示）相加的程序。在该程序中，被加数和加数按高字节在先（地址值较小），低字节在后（地址值较大）的顺序（即大端模式），分别存放在程序存储器和以 0000H 单元为首地址的外部数据存储器（实际是使用的 C8051F020 的 XRAM，即片上扩展 RAM）中，相加的和数存放在内部（指单片机内部）数据存储器地址为 20H（低字节）、21H（高字节）、22H（可能进位位）的三个单元中。请注意区分上述三种不同的存储器及其寻址方式，参考程序如下：

```
    WDTCN DATA 0FFH       ;定义地址常量
    ORG 0000H             ;程序定位
    AJMP START            ;跳转到标号
    ORG 0100H             ;程序定位
START:
    MOV WDTCN, #0DEH
    MOV WDTCN, #0ADH      ;禁止看门狗电路复位
    MOV R0, #20H          ;设置内部数据存储器间接寻址指针
    MOV DPTR, #CONST      ;设置程序存储器数据指针
    CLR A
    MOVC A, @A+DPTR       ;读程序存储器数据
    MOV R1, A             ;保存加数低位
    CLR A
    INC DPTR              ;指针地址加1
    MOVC A, @A+DPTR
    MOV R2, A             ;保存加数高位
    MOV DPTR, #0000H      ;设置外部数据存储器指针
```

```
    MOVX A, @DPTR
    ADD A, R1              ;先加低位
    DA A                   ;十进制数加法调整
    MOV @R0, A             ;写内部数据存储器（间接寻址只能@R0或@R1）
    INC DPTR
    INC R0                 ;内部数据指针加1
    MOVX A, @DPTR
    ADDC A, R2             ;带进位加高位
    DA A                   ;十进制数加法调整
    MOV @R0, A             ;保存高位
    INC R0
    CLR A
    MOV ACC.0, C           ;将进位位赋值给寄存器A的最低位
    MOV @R0, A             ;保存最高进位位
    SJMP $                 ;原地循环
CONST:
    DW  1234H              ;加数
    END
```

程序中前两个语句用来关闭 C8051F020 的看门狗复位电路，避免程序自动复位。看门狗电路是对程序意外“跑飞”或死机（可能由于不可预知的输入信号甚至是外部的电磁干扰）的补救措施，使程序可以重新运行以恢复功能。为了表示程序仍然在正确运行，设计代码的时候需要周期性的给看门狗电路发送消息（写寄存器），而一旦经过一段设定的时间看门狗电路没有收到消息，控制电路就会把单片机复位。我们在实验中不使用看门狗的功能，因此在最开始的代码先把它关掉，就是连续写入两个字节 0xDE 和 0xAD。两个字节的设计也是为了防止程序跑飞的时候随机执行的机器代码恰好有关闭看门狗的指令而准备的，从很大程度上提高了看门狗的可靠性。

调试、运行参考程序，学习开发系统及其调试软件的基本操作方法，特别注意如下几个方面：

1. 利用单步调试的功能检查每条汇编语句的执行效果，寄存器值的变化可以参考软件的寄存器窗口（Registers）
2. 注意反汇编窗口（Disassembly）中的汇编语句和其在程序中的保存地址
3. 特别留心程序代码 “DW 1234H” 在反汇编窗口中的表现形式，它的高位和低位是按照什么顺序保存的。

1.3.3 多个十进制数相加

编写一个程序，实现 N（$2 < N \leq 10$）个四位十进制数相加的算术运算。N 个四位十进制数存放在外部数据存储器（注意 XRAM 只有 4K 字节，不要将数据地址越界），相加数的个数存放在程序存储器，和数存放在内部存储器。参考程序流程如图 1.2：

其中循环语句可以使用 DJNZ 指令，指令含义可以参考附录中的手册，具体语法格式如下：

```
DJNZ  Rn, rel
```

假如循环开始的位置标号为 “LOOP”，程序使用的循环变量为 R7，则最终的代码应该为：

```
DJNZ  R7, LOOP
```

程序运行流程和调试的过程如下：

1. 使用内存观察窗口修改加数和被加数对应地址中的内容，将原始数据写入内存
2. 全速运行代码，之后暂停程序，检查和数的值是否正确
3. 保证输入的数据可以对代码的全部功能进行测试
4. 修改了加数之后可以复位程序并重新运行来进行测试
5. 如果出现错误，利用软件提供的各种调试手段进行调试
6. 在内存观察窗口中查看代码空间，找到“AJMP START”和其它指令对应的汇编代码位置

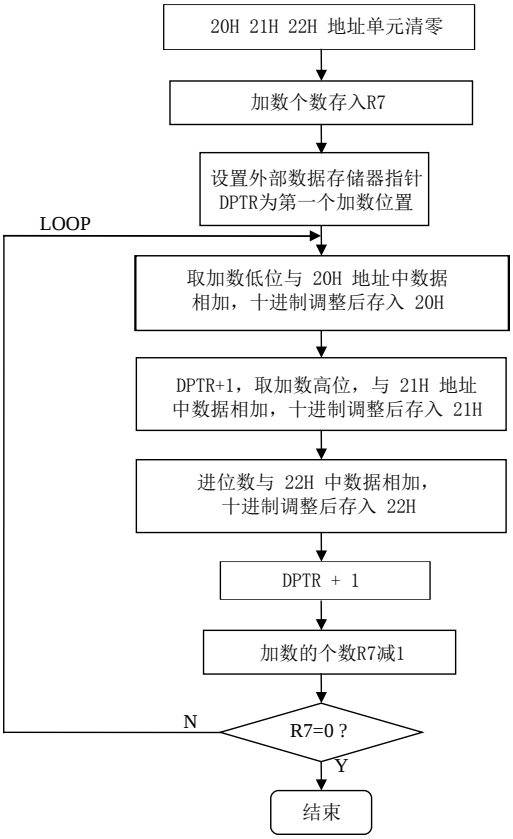


图 1.2: N 个加数加法流程

1.4 思考题

1. 两个十进制数相加的参考程序中“SJMP $”是必需的吗？如果没有“SJMP $”，程序运行后能否得到正确的结果？为什么？
2. 在调试 N 个加数加法算法时，如何保证代码的测试覆盖率。
3. 汇编语言代码如果没有对看门狗的设置，这会对实验结果有什么影响？

实验二　定时器和计数器

2.1　实验目的

1. 掌握使用 C 语言进行单片机程序开发的方法，熟悉开发环境
2. 熟悉单片机 C 语言开发中的特殊语法
3. 了解单片机中的定时器和计数器
4. 了解单片机的中断系统原理和编程方法

2.2　实验原理

2.2.1　C51 简介

为了提高单片机开发的效率，C 语言也是一种非常常用的编程语言，对于 MCS-51 系列的单片机来说，支持它的 C 语言就叫做 C51。C51 和标准 C 有一定的区别，主要体现在数据类型和数据存储结构上的差别，例如下面几种特别的存储类型[1]：

1. 位类型：bit

   布尔处理器是 8051 单片机的特色，使用它可以方便进行逻辑操作。位类型（bit）可以定义一个位变量，由 C51 编译器在 8051 内部 RAM 区 20H∼2FH 的 128 个位地址中分配一个位地址。需要注意的是，位类型不能定义指针和数组。
2. 特殊功能寄存器：sfr

   8051 及其兼容产品的特殊功能寄存器必须采用直接寻址的方式来访问，8051 的特殊功能寄存器离散地分布在 80H∼FFH 的地址空间里。sfr 可以对 8051 的特殊功能寄存器进行定义，sfr 型数据占用一个字节，取值范围 0∼255。
3. 16 位特殊功能寄存器：sfr16

   8051 及其兼容产品的 16 位特殊功能寄存器（如 DPTR），就可以用 sfr16 来定义，sfr16 型数据占用两个字节，取值范围 0∼65535。
4. 可寻址位类型：sbit

   利用 sbit 可以对 8051 内部 RAM 的位寻址空间及特殊功能寄存器的可寻址位进行定义。

   例如：sbit flag = P10；表示将 P1.0 这条 I/O 口线定义为 flag 变量。

在使用中需注意，只有 unsigned char 和 bit 类型是 8051 CPU 可以直接用汇编语言支持的数据类型，它们的操作效率最高。其他的数据类型都有多条汇编指令组合操作，需占用大量的程序存储空间和数据存储器资源。C51 还支持结构类型和联合类型等复杂类型数据，与标准 C 相同，不另介绍。

如果要指定变量的存储区域，还需要一些特殊的 C 语言扩展，下面是相关的几个关键字：

data 直接寻址片内 RAM，00∼7FH 空间，速度最快
bdata 可位寻址的片内 RAM 区 20H∼2FH 空间，容许位和字节混合访问

[1] 这里介绍的语法格式是 Keil C 所支持的，其它不同的 C51 实现可能会不同

idada 间接访问片内 00~FFH 全部 256 个地址空间
pdata 使用 MOVX @Ri 指令访问外部 RAM 分页的 00~FFH 空间
xdata 使用 MOVX @DPTR 访问外部 RAM 的 0000~FFFFH 全部空间
code 使用 MOVC @A+DPTR 访问程序存储器 0000~FFFFH 全部空间

另外还可以通过 _at_ 关键字指定具体的存储位置。例如下面的这个数据定义就指定了变量 a 保存在外部数据空间的 0x500 地址：

```
unsigned int xdata a  _at_  (0x500);
```

为了处理单片机的中断，C51 还提供了 interrupt 关键字，支持直接编写中断服务程序函数。其函数定义的形式为：

```
函数类型  函数名（）  [ interrupt  n ]  [ using  n ]
```

函数类型一般定义为 void，interrupt 后的 n 是中断号，指示相应的中断源。C51 编译器从 code 区的绝对地址 8n+3 处产生中断向量。n 必须是常数，不允许使用表达式。

using n 是可选的，n 为 0~3 的常数，指示选择 8051 的 4 个寄存器组，即由程序状态字 PSW 中 RS0/RS1 所指定的工作寄存器可以保存的 4 个位置。如果不使用 using n，中断函数所有使用的公共寄存器都入栈（保护现场）。如果使用 using n，切换的寄存器就不再入栈（因为使用了不同的寄存器组，因此不需要保护 R0~R7）。

编写 C51 的中断函数时，需要注意的几个问题：

- 中断函数没有返回值，因此它必须是一个 void 类型的函数；
- 中断函数不允许进行参数传递；
- 不允许直接调用中断函数；
- 中断函数对压栈和出栈的处理由编译器完成，无需人工管理；
- 需要严格注意 using n 的使用，必须确保寄存器组的正确切换。

2.2.2 并行端口

C8051F020 一共支持 64 个数字输入输出接口，共组成 8 组 8 位的并行端口。其中 P0、P1、P2 和 P3 被称为低位端口（标准的 MCS-51 只支持 4 个端口），可以通过寄存器按字节或者按位访问；对应的 P4、P5、P6、P7 为高位端口，仅可以按字节访问。所有端口都可以配置成漏级开路输出或推挽输出，图 2.1 是每个输入输出接口的结构图。

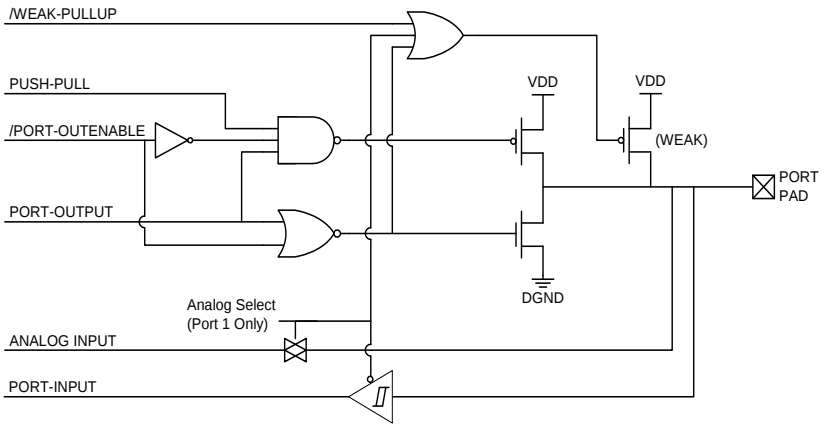


图 2.1: C8051F020 的端口结构

对 P0 ~ P7 寄存器的读写可以实现对相应端口的输入和输出，另外部分端口还支持对其工作状态的设置。例如 P0MDOUT 寄存器可以用来设置 P0 端口 8 个接口的输出模式，其中设置为 1 的位表示对应

的接口为推挽方式输出，设置为 0 的位表示对应的接口采用漏级开路的方式输出。低位端口都支持这种设置方法，但对于高位端口，只有一个 P74OUT 寄存器用来设置端口的输出状态，每个寄存器位和接口的对应关系如表 2.1所示。字母 L 表示低四位，字母 H 表示高四位，也就是说对于高位端口只能按照四位每组的方式进行输出设置。

表 2.1: P74OUT 寄存器结构

D7	D6	D5	D4	D3	D2	D1	D0
P7H	P7L	P6H	P6L	P5H	P5L	P4H	P4L

2.2.3 MCS-51 单片机的中断系统

MCS-51 单片机有 5 个中断源：2 个外部中断源 INT0 和 INT1；2 个为片内定时器/计数器溢出中断 TF0 和 TF1；1 个为片内串行口的收/发中断 RI。C8051F020 具有多达 22 个中断源，其具体描述会再以后的实验中逐步介绍。

与中断有关的专用寄存器有中断允许寄存器 IE、中断优先级控制寄存器 IP、控制寄存器 TCON 等，它们的定义分别列于表 2.2、2.3和 2.4。其中有关定时器/计数器 2 的相关控制位是 C8051F020 扩展的。

表 2.2: 中断允许寄存器 IE

D7	D6	D5	D4	D3	D2	D1	D0
EA		ET2	ES	ET1	EX1	ET0	EX0
总中断允许位		定时器/计数器 2 中断允许位	串行通信口中断允许位	定时器/计数器 1 中断允许位	外部中断 INT1 中断允许位	定时器/计数器 0 中断允许	外部中断 INT0 中断允许位

其中，EA 是总中断允许位。EA=0 时，禁止一切中断；EA=1 时，每个中断源是否被允许取决于各自允许位设置：置 1 表示允许该中断源中断；置 0 表示屏蔽该中断。复位后所有中断允许位均为 0。

表 2.3: 中断优先级控制寄存器 IP

D7	D6	D5	D4	D3	D2	D1	D0
		PT2	PS	PT1	PX1	PT0	PX0
		定时器/计数器 2	串行口	定时器/计数器 1	外部中断 1	定时器/计数器 0	外部中断 0

表 2.3中标出与各中断源对应的优先级控制位，置 1，表示该中断源是高优先级；置 0，为低优先级。复位后各位均为 0，所有中断的优先级是相同的。

MCS-51 的硬件结构仅支持一级中断嵌套，靠 IP 寄存器可以将中断优先级分为高、低两级，它们遵从下面两条基本原则：

1. 低优先级中断可被高优先级中断所中断，反之不能。
2. 一种中断（不论哪个优先级）一旦得到响应，与它同级的中断不能再中断它。

当同时收到几个同一优先级的中断请求时，哪一个请求得到服务，取决于内部的查询顺序，相当于在每个优先级内，还存在另一辅助优先级结构：中断序号小的中断具有较高的优先级。

TF1：当定时器 / 计数器 1 溢出时，由硬件置位，申请中断。进入中断服务程序后被硬件自动清除。TF0 类似于 TF1。

TR1：靠软件置位或清除，置位时，启动定时器 / 计数器 1 工作，清除时，停止工作。TR0 类似于 TR1。

IE1：检测到在 INT1 引脚上出现的外部中断信号的下降沿时，由硬件置位，请求中断。进入中断服务程序后，被硬件自动清除。IE0 类似于 IE1。

表 2.4: 控制寄存器 TCON

D7	D6	D5	D4	D3	D2	D1	D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
定时器/计数器 1 溢出标志	定时器/计数器 1 运行控制	定时器/计数器 0 溢出标志	定时器/计数器 0 运行控制	外部沿触发中断 1 请求标志	外部中断 1 触发类型控制	外部沿触发中断 0 请求标志	外部中断 0 触发类型控制

IT1：靠软件来设置或清除，以控制外部中断 1 的触发类型。置 1 时，下降沿触发；置 0 时，低电平触发。IT0 类似于 IT1。

当单片机检测到中断时间发生（例如定时器发生溢出）并且相应的中断已经在控制寄存器中被允许，则 PC 指针跳转到中断所对应的入口地址执行那里的代码。中断入口一般是一个跳转指令，跳转到相应的中断处理程序（ISR，Interrupt Service Routine）运行。在中断程序最后，会执行 RETI 指定返回到中断前的地址继续运行。在程序存储器中，各中断源的入口地址见表 2.5。

表 2.5: 中断入口地址

中断源	入口地址
外部中断 INT0	0003H
定时器/计数器 T0	000BH
外部中断 INT1	0013H
定时器/计数器 T1	001BH
串行口	0023H

2.2.4 时钟和振荡器

C8051F020 内部包含一个振荡器，当系统复位时，单片机首先使用内部的振荡器作为系统时钟（缺省工作频率为 2MHz，用 SYSCLK 表示），而如果要使用其它的工作频率或者使用外部的晶体振荡器，都必须通过寄存器进行设置。OSCICN 寄存器的定义如表 2.6 所示，OSCXCN 寄存器的定义如表 2.7所示。

表 2.6: OSCICN 寄存器结构

D7	D6	D5	D4	D3	D2	D1	D0
MSCLKE			IFRDY	CLKSL	IOSCEN	IFCN1	IFCN0

MSCLKE 用来配置是否对时钟失效进行检测，如果设置为 1，当 100 微秒没有检测到时钟信号时，系统将复位。

IFRDY 用来检测内部振荡器是否工作在设置的频率下。

CLKSL 用来选择系统使用的振荡器。设置为 1 表示使用外部振荡器，0 表示使用内部振荡器。

IOSCEN 用来设置内部振荡器的工作状态。设置为 1 表示使用内部振荡器，0 表示禁止内部振荡器

IFCN1-0 用来设置内部振荡器的工作频率。00、01、10、11 分别代表 2MHz、4MHz、8MHz 和 16MHz。

XTLVLD 外部晶体振荡器状态。1 表示外部晶体振荡器已经稳定工作，0 表示外部晶体振荡器还没有稳定工作。

XOSCMD2-0 用来设置外部振荡器的模式，00x 表示 XTAL1 内部接地（外部振荡器不工作）；01x 表示使用外部时钟源；10x 表示使用 RC 振荡器，并且进行二分频；11x 表示使用外部晶体振荡器。XOSCMD0 位用来表示是否要进行二分频（对 RC 振荡器模式无效）。

XFCN2-0 用来设置外部振荡器的频率范围，其中设置为 111 表示频率大于 6.7 MHz，其它设置请参考数据手册。

表 2.7: OSCXCN 寄存器结构

D7	D6	D5	D4	D3	D2	D1	D0
XTLVLD	XOSCMD2	XOSCMD1	XOSCMD0		XFCN2	XFCN1	XFCN0

使用外部振荡器可以获得更好的时钟稳定性和更高的时钟频率，具体的设置过程可以参考下面提供的参考代码。

```
void SYSCLK_Init (void)
{
   int i;
   OSCXCN = 0x67;                    // 启动外部晶振
   for (i=0; i < 256; i++);          // 延时一段时间
   while (!(OSCXCN & 0x80));         // 等待振荡稳定
   OSCICN = 0x88;                    // 使用外部振荡器
}
```

2.2.5 定时器和计数器

MCS-51 系列单片机有两个定时器/计数器，即：定时器/计数器 0 和 1（C8051F020 具有更多）。在模式控制寄存器 TMOD 中各有一个控制位（C/T），分别用于控制定时器/计数器 0 和 1 是工作在定时器方式还是计数器方式，当计数值发生溢出的时候会触发中断。

- 选择定时器工作方式时，计数输入是内部系统时钟，每个机器周期（传统 8051 的机器周期是时钟周期除以 12，C8051F020 也支持这种方式，参考后面对表 2.9的说明）使寄存器的值加 1。
- 选择计数器工作方式时，计数脉冲来自相应的外部输入引脚 T0 或 T1，当输入信号由 1 跳变至 0 时，计数寄存器的值加 1。

除了可以选择定时器或计数器工作方式外，每个定时器/计数器还有 4 种操作模式，其中前三种模式对两者都是一样的，只有模式 3 对两者不同。

（1）模式 0

在通过 TMOD 寄存器把定时器/计数器 0 或 1 置为模式 0 时，它被设置成 13 位的定时器/计数器。TL1 的高三位未用。在此模式下，允许计数的控制逻辑由表达式 $TRx \times (\overline{GATE} + INTx)$（其中 x=0 或 1）的值决定。其值为真时，允许计数。式中 TRx 是定时器控制寄存器 TCON 的一个控制位，GATE 是模式控制寄存器 TMOD 的一个控制位，INTx 是两个外部中断的输入端之一。图 2.2 是计数器 0 在此模式下的示意图。

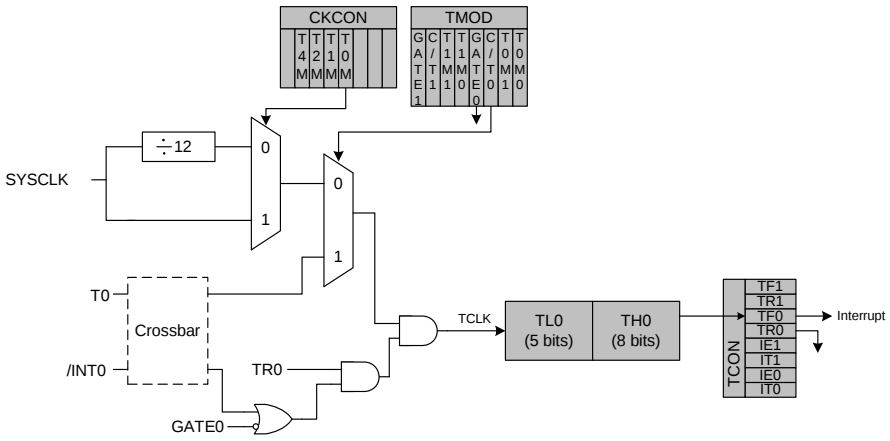


图 2.2: 定时器/计数器 0 的模式 0

（2）模式 1

模式 1 与模式 0 几乎完全相同，唯一的差别是：模式 1 中定时器/计数器是以全 16 位参与操作的。

（3）模式 2

模式 2 把定时器寄存器 TLx 配置成一个可以自动重装载的 8 位计数器。TLx 计数溢出时，不仅使溢出标志置 1，而且还自动把 THx 中的内容重装载到 TLx 中。

（4）模式 3

在此模式下，定时器/计数器 0 和 1 的工作情况是不同的。对于定时器/计数器 1，设置为模式 3 将使它保持原有的计数值，即如同使它停止计数。对于定时器/计数器 0，设置为模式 3 将使 TL0 和 TH0 成为两个互相独立的 8 位计数器，其中 TL0 的计数控制采用 TR0、C/T0、GATE0 和 TF0（参考表 2.8和表 2.4），而 TH0 的计数控制采用定时器/计数器 1 的相关控制位，此时定时器/计数器 1 不用产生中断，但依然可以作为串口的波特率发生器（见后续实验）。

特殊功能寄存器 TMOD 用于控制定时器/计数器 0 或 1 的工作模式，其定义见表 2.8。

表 2.8: TMOD 寄存器

D7	D6	D5	D4	D3	D2	D1	D0
GATE	C/T	M1	M0	GATE	C/T	M1	M0

其中 D4—D7 对应定时器/计数器 1，D0—D3 对应定时器/计数器 0，功能说明如下：GATE 选通门。当 GATE=1 时，只有 INTx 引脚为高电平且 TRx 置 1 时，相应的定时器/计数器才被选通，这时可用于测量 INTx 端出现的正脉冲的宽度。若 GATE=0，则只要 TRx 置 1，定时器/计数器就被选通，而与 INTx 端的电平无关。C/T 计数器方式和定时器方式选择位。C/T=0 时，设置为定时器方式；C/T=1 时，设置为计数器方式。M1 和 M0 用来选择计数器的工作模式 0—3。

C8051F020 还支持另外三个定时器/计数器，编号分别是 2、3 和 4，具体的使用方法在后续实验会有介绍。当工作在定时器模式下时，计数寄存器的值每个机器周期加 1，传统的 MCS-51 的机器周期是系统晶振[2]的 12 倍，C8051F020 也支持这种传统方式，同时还支持每个系统晶振周期加 1 的模式，控制寄存器是 CKCON，具体见表 2.9。其中四个控制位分别控制了定时器/计数器 4、2、1、0 的输入时钟模式，缺省值 0 表示传统的系统晶振频率除以 12，设置为 1 则表示直接采用系统晶振频率。

表 2.9: CKCON 寄存器

D7	D6	D5	D4	D3	D2	D1	D0
	T4M	T2M	T1M	T0M			

2.3 实验内容

2.3.1 控制 LED 闪烁

主板上的 LED 与 CPU 的 P7 端口最高位相连，通过控制 P7 的输出，可以让这个 LED 闪烁，下面是编程的基本流程。

1. 建立工程，使用 C 语言编写代码，源代码扩展名为 c
2. 示例代码如下：

```
#include <C8051F020.h>  // C8051F020 处理器相关的寄存器设置
void PORT_Init (void)
{
    P74OUT = 0x80;       // 推挽输出
}
```

[2]实验板所使用的晶振为 22118400Hz

```
void Delay()
{
    int i, j;
    for (i = 0; i < 500; ++i)
        for (j = 0; j < 100; ++j);
}

void main(void)
{
    WDTCN = 0xde;
    WDTCN = 0xad;       // 禁止看门狗复位
    PORT_Init();
    while (1) {
        P7 = 0x80;      // P7.7 连接 LED
        Delay();        // 循环延时
        P7 = 0;
        Delay();
    }
}
```

3. 调试过程中可以在反汇编窗口看到相应的汇编代码，通过对比可以更加深刻的了解单片机的工作原理
4. 在工程设置中的 Listing 页，勾选 C Compilter Listing 中的 Assembly Code 选项，就可以在编译工程时生成的列表文件（与 C 源文件同名，扩展名为 lst）中包含生成的汇编代码

2.3.2 使用计数器延时

使用计数器 0 控制 LED 的闪烁频率为 1Hz。为了保证闪烁频率的准确，请使用外部晶振。程序代码如下所示，注意中断程序的用法。

```
void main()
{
    WDTCN=0xDE;
    WDTCN=0xAD;         // 禁止看门狗
    SYSCLK_Init();      // 设置外部晶振
    PORT_Init();        // 设置端口输出模式

    EA = 1;             // 允许全局中断
    TMOD |= 0x02;       // 计数器 0 采用 8 位自装载模式
    TH0 = 0x00;
    TL0 = 0x00;         // 设置初值
    TR0 = 1;            // 启动计数
    ET0 = 1;            // 允许计数器中断
    P7 = 0x80;
    while (1);          // 原地循环
}

void TIMER0_ISR (void) interrupt 1  // 计数器 0 溢出中断
{
    static int count;   // 中断次数计数
    count++;
    if (count > 3600){  // 半秒时间
        count = 0;
        P7 = ~P7;       // P7 取反
    }
}
```

2.3.3 控制 LED 亮度

PWM（Pulse Width Modulation）是脉冲宽度调制技术，通过控制输出方波信号的占空比，来达到输出功率的变化。这个技术可以很方便的采用数字输出来进行模拟电路的控制，经常用来控制发光器件的亮度和电机的转速等等。前一个实验内容相当于通过 P7.7 端口输出了一个 1Hz 的方波，这里我们将方波的频率提高到 150Hz 左右，然后通过控制方波的占空比，使 LED 的亮度从暗变亮，再从亮变暗，不断循环。由于占空比大于 50% 之后亮度变化不大，可以让占空比从 0 变到 50%，再变回 0 这样循环。

2.4 思考题

1. 如何估算 C 代码中 Delay 函数的延时
2. 在 PWM 输出的实验中，如果方波频率过低（比如低于 20Hz）会有什么现象发生？

实验三　键盘与显示

3.1　实验目的

1. 了解键盘扫描的原理
2. 了解扫描显示的原理
3. 了解总线扩展外设的原理

3.2　实验原理

3.2.1　交叉开关

交叉开关(Priority Crossbar)是 C8051F020 的一个很有特色的结构,它允许用户将 C8051F020 所包含各种外设的管脚按照一个优先级顺序映射到特定的 IO 管脚上。具体来说就是 C8051F020 的低位端口 P0—P3 既可以作为普通的 IO 端口使用,也可以映射为某种外设的管脚。例如可以把 P0.0 映射为串口的输出,把 P0.1 映射为串口的输入,此时这两个接口就成为了串口外设的输入输出接口,而不能再作为普通 IO 使用了。这种处理方式使得对 IO 端口的使用方法更加灵活,可以满足不同的应用环境需求。

C8051F020 对低位端口的管理通过 XBR0、XBR1、XBR2 寄存器进行设置,图 3.1 展示了端口的对应关系,其中外设的优先级是上端最高,越往下优先级越低;而端口的优先级是左边最高,越往右优先级越低。例如在 XBR 中只使用了 I^2C (SDA、SCL),那么需要设置 XBR0.0,结果是使用 P0.0 作为 SDA,使用 P0.1 作为 SCL。而如果同时串口(TX0,RX0)也被允许(XBR0.2 被设置),由于串口优先级更高,P0.0 将作为 TX0,P0.1 将作为 RX0,而 P0.2 将作为 SDA,P0.3 作为 SCL。如果 P0 端口被用作地址总线的控制线(其中的 ALE、/RD、/WE 功能),那么相应的端口位就不能用作其它功能,在图 3.1 中相应的端口选择就要向右移动。例如串口和 SPI 接口被允许的情况下(XBR0.2 和 XBR0.1 有效),如果 P0 没有作为控制线,那么串口将使用 P0.1 和 P0.2,而 SPI 接口将使用 P0.2 ~ P0.5,而如果 P0 被用作控制线,那么 NSS 管脚就只能使用 P1.0 了。

XBR0 和 XBR1 的定义情况在图 3.1 中大部分已经展示出来了,XBR2 中还有一些位的定义需要参考表 3.1 的说明。

表 3.1: XBR2 寄存器结构

D7	D6	D5	D4	D3	D2	D1	D0
WEAKPUD	XBARE		T4EXE	T4E	UART1E	EMIFLE	CNVSTE

WEAKPUD 用来设置端口的全局上拉电阻。缺省为 0 表示允许上拉(上拉电阻 100k 左右,参考图 2.1),当设置为 1 的时候,上拉电阻被禁止。

XBARE 用来允许交叉开关的功能。设置为 1 表示交叉开关被允许,0 表示交叉开关不工作,P0—P3 被设置为输入模式

EMIFLE 用来设置外部存储器接口的控制位使用情况,当设置为 1 的时候,P0.7 和 P0.6 在配置外设端口的时候会被跳过(因为他们已经作为 WR 和 RD 使用),当外部存储器工作在复用模式的时候 P0.5

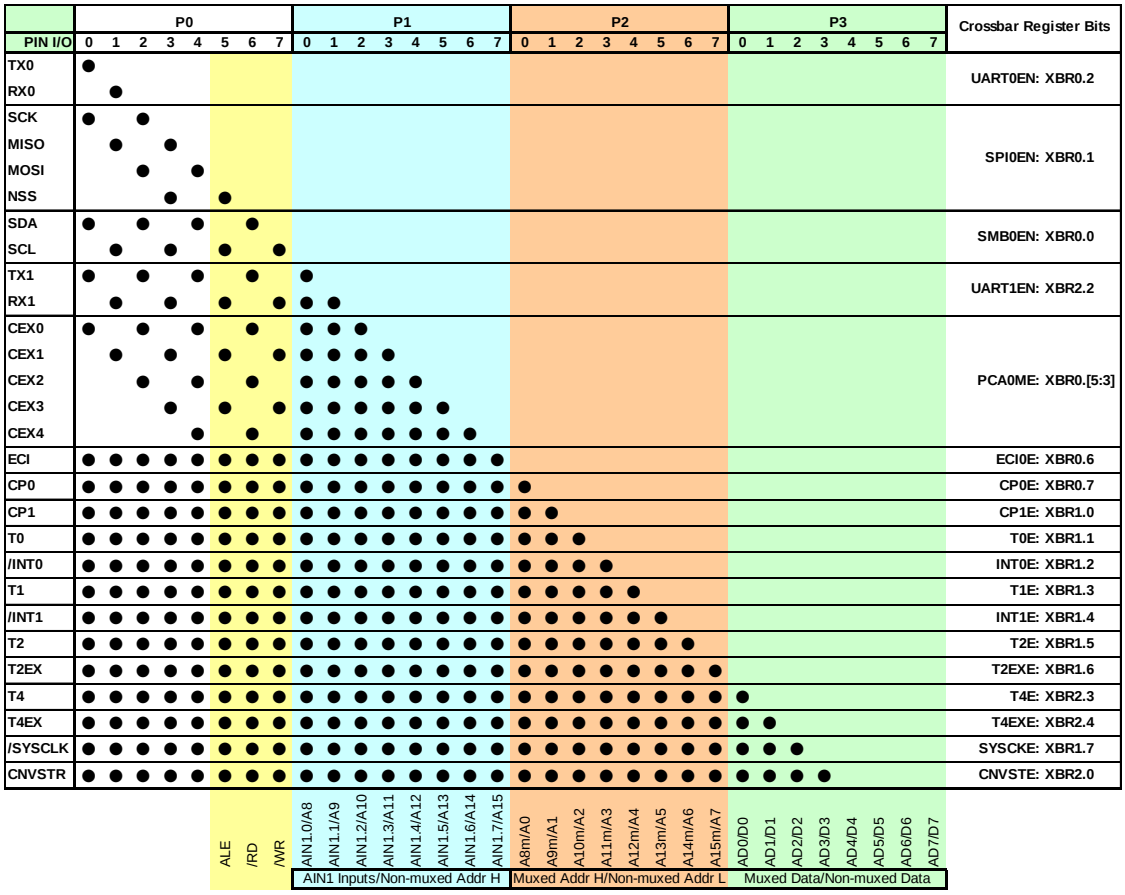


图 3.1: 交叉开关译码表

也会被交叉开关跳过(已经作为 ALE 使用)。当 EMIFLE 设置为 0 的时候,这三个接口的电平状态由端口寄存器或者对应外设决定。

3.2.2 MCS-51 外部总线接口

标准的 MCS-51 单片机仅包含 128 字节的内部数据区,这对于很多应用来说都太少了。C8051F020 内部包含 256 字节的数据区,片上还带有 4K XRAM,如果这些还不够用,就必须通过外部总线扩展的方式提供更多的数据存储器。另外,一些外设单元也是通过外部总线的方式进行连接的,他们对于 CPU 来说表现为一系列的寄存器,CPU 通过特定的地址来访问这些寄存器。MCS-51 单片机的总线接口包含地址和数据总线,可以采用如图 3.2的连接方式和外设连接。这里 $A[15:8]$ 高位地址由 P2 驱动,而 $AD[7:0]$ 由 P0 驱动,是地址和数据复用的总线,必须在 ALE 的上升沿将其锁存,以保证读写周期的完成。

C8051F020 在地址数据复用方式下,驱动 $AD[7:0]$ 的是 P3 口,P2 负责高位地址。C8051F020 不仅支持标准的复用方式连接,还支持非复用的方式,这样就可以省去一个 373 锁存芯片,代价是必须增加一个端口的使用,P2 负责低位地址的驱动,P1 负责高位地址的驱动,而 P3 仅负责数据总线的驱动,如图 3.3所示。

由于 C8051F020 的并行端口非常的多,包括从 P0—P7,通过 EMI0CF 可以设置外部存储器接口使用 P0—P3(P0 提供 RD/WR 读写信号)还是使用 P4—P7,其中 EMI0CF 寄存器的结构如表 3.2 所示。我们在开发板中使用的是 P0—P3 的非复用模式,P0 中用到了 P0.6 和 P0.7 分别作为 /RD 个 /WR。

PRTSEL 用来设置端口的选择,1 表示使用高位端口,0 表示使用低位端口。

EMD2 用来设置复用模式,1 表示采用非复用方式,0 表示使用复用模块。

EMD1-0 用来设置外部存储器的组织方式,00 表示 MOVX 命令仅用来访问内部的 XRAM;11 表示

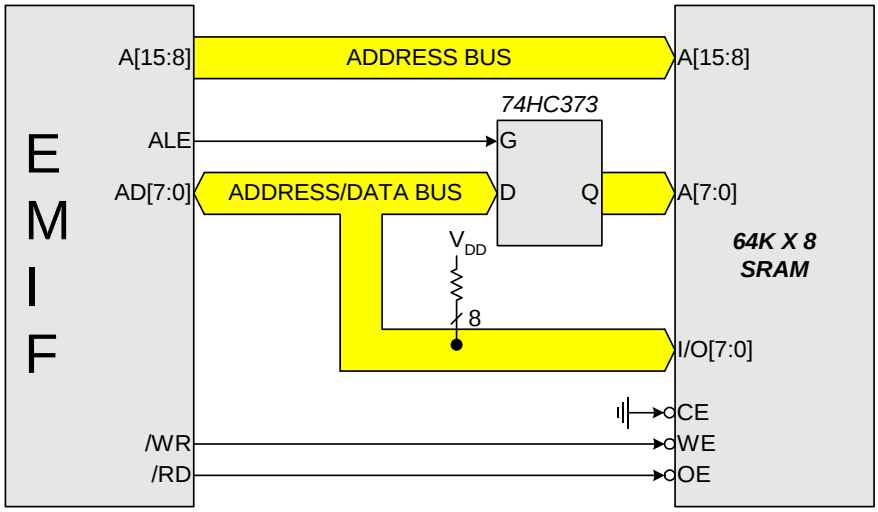


图 3.2: 地址数据复用方式连接

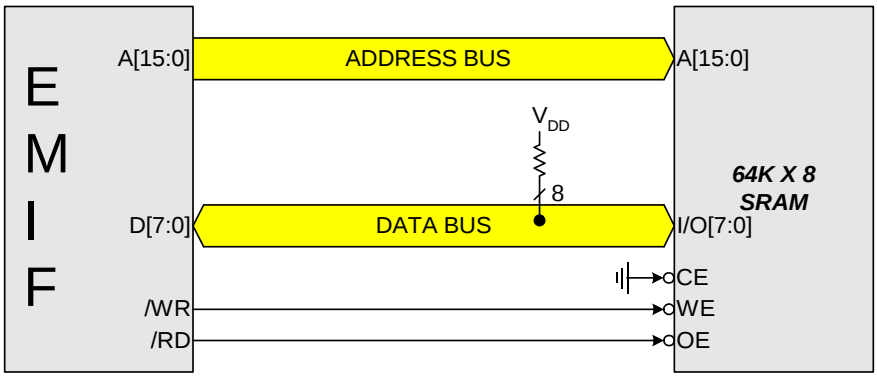


图 3.3: 地址数据非复用方式连接

MOVX 命令仅用来访问外部存储器；而 01 和 10 都表示低 4k 字节使用片内 XRAM，其余地址使用外部存储器，区别是在使用 8 位 MOVX 指令时高位地址的驱动模式是不一样的。

EALE1-0 用来设置 ALE 信号的宽度，00 表示 1 个系统时钟周期，而 11 表示 4 个系统时钟周期。

例如要使用 C8051F020 的低位端口进行复用模式的外部数据接口扩展，必须设置如下几个寄存器，具体代码可以参考后面的示例。

- 设置 EMI0CF 的 PRTSEL 和 EMD2 用来选择端口和复用模式
- 设置 XBR2 允许交叉开关和 P0 端口的总线控制信号
- 设置 P0MDOUT、P1MDOUT、P2MDOUT、P3MDOUT 使相关端口为推挽输出以获得足够的驱动能力

3.2.3 扫描显示

根据附录中的电路图，可以看出开发板通过扩展两个 373 锁存器，驱动了四个共阳极的数码管。其中一个 373 用来驱动数码管的四个阳极端，另外一个 373 用来驱动四个数码管的阴极。数码管的阴极控制线共有 8 根，对应控制寄存器的 8 位，可以用一个 8 位二进制数来表示数码管对应发光管的亮灭，这个二进制数被称为段码。段码所在寄存器的地址是 0x8000。数码管的阳极用来控制整个数码管的亮灭，对应的控制寄存器使用低四位进行控制，相应的编码被称为位码。位码所在寄存器的地址是 0x8001。因此，如果四个数码管同时点亮的话（四个阳极都为高电平），只能显示四个同样的符号，要想让它们在视觉上感觉是显示不同的符号，就必须利用扫描显示的方法。

表 3.2: EMI0CF 寄存器结构

D7	D6	D5	D4	D3	D2	D1	D0
		PRTSEL	EMD2	EMD1	EMD0	EALE1	EALE0

扫描显示是一种常用的发光显示技术，其基本原理是逐个点亮显示器件的每一个字符（只有一个阳极为高电平），利用人的视觉暂留现象，通过快速循环使整个显示部件看起来好像同时显示。这种显示技术不但可以减少器件的连线，还可以减少整个系统消耗的电流，降低系统功耗。

下面的列表是开发板的参考代码，可以用来显示一个不断增加的十进制计数值。

```
#include <C8051F020.h>
void PORT_Init (void)
{
    SYSCLK_Init();      // 使用外部晶振
    EMI0CF = 0x1F;      // 非复用总线, 不使用内部 XRAM
    XBR2 = 0x42;        // 使用 P0-P3 作为总线, 允许XBR
    P0MDOUT = 0xC0;     // 设置总线相关端口为推挽输出 P0.6 和 P0.7
    P1MDOUT = 0xFF;     // 高位地址
    P2MDOUT = 0xFF;     // 低位地址
    P3MDOUT = 0xFF;     // 数据总线
}

void Delay(int k)               // 延时子程序
{
    int i;
    for (i = 0; i < k; ++i);
}

unsigned char xdata seg _at_ (0x8000); // 数码管段码（阴极）地址
unsigned char xdata cs _at_ (0x8001);  // 数码管位码（阳极）地址

const unsigned char code segs[] =  // 段码表
   {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8,
    0x80, 0x90, 0x88, 0x83, 0xC6, 0xA1, 0x86, 0x8E};
const unsigned char code css[] = {0x7, 0xB, 0xD, 0xE}; //位码表

void main(void)
{
    unsigned int k;             // 循环计数
    WDTCN = 0xde;
    WDTCN = 0xad;               // 禁止看门狗
    PORT_Init();
    k = 0;
    while (1) {                 // 永远循环
        unsigned char i;
        int j;                  // 循环计数 / 64
        j = k / 64;
        for (i = 0; i < 4; ++i) { // 循环显示四位数码
            seg = segs[j % 10]; // 第 i 个段码
            cs = css[i];        // 位码
            j /= 10;            // 十进制右移
            Delay(1000);        // 等待
        }
        k++;
    }
}
```

3.2.4 矩阵键盘

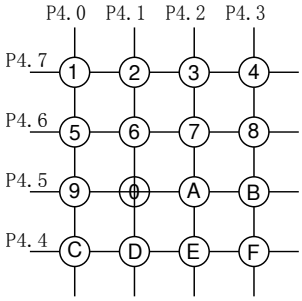


图 3.4: 矩阵键盘原理

键盘是单片机系统中重要的输入设备，行列扫描技术是常用的键盘实现方式。图 3.4为一个 4×4 的键盘阵列，可以把它看成是四个行导线与四个列导线组成的矩阵，当没有按下按钮的时候，行列是绝缘的；而当有键按下时，这个键对应的交叉点就被导通了。因此在这样的结构下，只需要 8 个 IO 就可以实现 16 个按键的输入。常用的判断按键位置的方法有扫描法和线反转法：

1. 扫描法是先使列（行）线全输出低电平，然后判断行（列）线状态，若行线全为高电平，表示无键被按下；若行线不全为高电平表示有键被按下，然后依次使单个列线为低电平，再判断行线状态，当行线全为高电平时，表示被按下的键不在本列；当行线不全为高电平时，表示被按下的键在本列，把此时的行线状态与列线状态和在一起即为被按下的键的位置。
2. 线反转法的第一步也是把列（行）线置低电平，行（列）置高电平然后读行状态；第二步与第一步相反然后读列线状态，若有键按下，则通过两次所读状态的结果即可获得键所在的位置。这样通过两次输出和两次读入就可完成键的识别，比扫描法要简单，并且不论键盘有多少行和多少列只需经过两步即可获得键的位置。

实验板中行线和列线都分别连接到单片机的并行端口上，分别占用 P4 端口高四位和低四位，并都具有弱的上拉，同时单片机的端口输出方式是漏级开路的。此时假如第 3 行、第 2 列的按键被按下，单片机通过线反转法读取键值的流程如下：

1. P4 端口输出 0x0F，即行导线输出低电平，列导线输出高电平
2. 此时由于“0”键被按下，因此 P4.1 会被拉低，因此 P4 端口读入的值为 0x0D，可以判断出第二列有键按下
3. 端口输出 0xF0，即行导线输出高电平，列导线输出低电平
4. P4 端口的读入值为 0xD0，可以判断出第三行有键按下
5. 综合可以判读出按下的键为“0”

机械按键在接合和断开的瞬间都会有个不稳定的状态，这个过程被称为抖动。如果不做处理单次操作在软件上就会被读入多次按键，键盘的去抖可以用硬件或者用软件完成。软件去抖可以在程序上增加延时，当发现有按键动作的时候通过一定延时（一般是 10 到 20 毫秒）做二次判断来解决。下面是读取按键的参考程序，这个程序通过 dec 数组，将端口读出的值转换为相应的行编号或者列编号。并利用 行 $\times 4 +$ 列获得一个小于 16 的键盘编码，然后通过 trans 数组转换为对应的实际键值。

```
#define NOKEY 255
#define uchar unsigned char
uchar getkey()
{
    uchar i;
    uchar key;
    const uchar code dec[] = {0, 0, 1, 0, 2, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0};
```

```
    const uchar code trans[] = {0xC, 9, 5, 1, 0xD, 0, 6, 2, 0xE, 0xA, 7, 3, 0xF, 0xB, 8, 4};
    P4 = 0x0F;              // 低四位高电平输出
    Delay(100);             // 延时等待电平稳定
    i = ~P4 & 0x0F;         // 获取低四位
    if (i == 0) return NOKEY; // 没有按键
    key = dec[i] * 4;       // 转换为列值然后x4
    Delay(1000);            // 延时去抖
    P4 = 0xF0;              // 高四位高电平输出
    Delay(100);             // 延时
    i = ~P4;
    i >>= 4;                // 获取输入值
    if (i == 0) return NOKEY; // 没有按键
    key = key + dec[i];     // 计算出键码
    key = trans[key];       // 转换为真实键值
    return key;
}
```

3.3 实验内容

3.3.1 键盘显示联调

首先验证数码管显示程序的正确，然后编写程序，将按键的键值显示到数码管上面。要求如下：

- 初始状态显示全暗（根据情况可增加段码表的长度）
- 输入按键的内容（0~F 字符）显示在最右边的数码管上
- 输入新内容的时候，数码管上原有的内容全部向左移动一位，最左边的内容消失，新内容依然显示在最右边的数码管上
- 注意判断单次按键的逻辑，长按只记一次

3.3.2 设计数码电子表

利用单片机的定时器，实现一个电子表功能，基本要求如下：

- 显示分秒计数，均为 60 进位。
- 可以设置电子表的显示值并继续计数

附加要求（根据实验时间和完成情况选择实现）：

- 可以按键切换显示时分/分秒两种模式
- 可以有秒表功能：清零、暂停、继续、恢复时钟
- 其它可扩展的功能

完成实验要循序渐进，不要一开始就构想的非常复杂，造成基本功能无法实现。

3.4 思考题

1. 程序中的段码表和位码表是如何与硬件对应的？
2. 示例程序是否可以判断多个键同时按下？如果要做这种判断应该如何编程？

实验四　模数和数模转换

4.1　实验目的

1. 了解数模和模数转换电路的原理和使用方法；
2. 掌握 MCS-51 系列单片机中定时器和计数器的使用方法；
3. 掌握用示波器、信号源对单片机系统进行调试的方法。

4.2　实验原理

4.2.1　数模转换器

在单片机的测控系统中，单片机的处理结果常常需要转换为模拟信号去驱动相应的执行单元，实现对被控对象的控制。这种把数字量转换为模拟量的设备称为数模（D/A）转换器。

C8051F020/1/2/3 系列 MCU 内部有两个 12 位 DAC 和两个比较器。MCU 与每个比较器和 DAC 之间的数据和控制接口通过特殊功能寄存器实现（DAC0 的控制寄存器 DAC0CN 的定义见表 4.1）。MCU 可以将任何一个 DAC 或比较器置于低功耗关断方式。DAC 为电压输出方式，有灵活的输出更新机制，允许用软件写和定时器 2、定时器 3 及定时器 4 的溢出信号更新 DAC 输出。DAC 的结构如图 4.1

表 4.1: DAC0CN 寄存器结构

D7	D6	D5	D4	D3	D2	D1	D0
DAC0EN			DAC0MD1	DAC0MD0	DAC0DF2	DAC0DF1	DAC0DF0

DAC0EN：DAC0 的允许位，0 代表禁止，1 代表允许

DAC0MD1-0：用来设置 DAC0 的工作模式，00 表示 DAC 转换以写入 DAC0H（DAC0 设置值的高位）寄存器触发；01、10、11 分别代表 DAC 的转换开始时间由 Timer3、Timer4、Timer2 溢出触发

DAC0DF2-0：用来设置 DAC 数据保存的格式。由于 DAC 的设置值是个 12 位的数，保存在寄存器（两个 8 位寄存器，DAC0H 和 DAC0L）的时候必须指定如何对应。这个寄存器设置为 000 表示 DAC0H 的高四位不使用，即 DAC0H 仅保存 12 位数的高四位，DAC0L 保存余下的低 8 位，这种对应方式被称为右对齐；设置为 001 表示 DAC0H 的高三位不使用，DAC0L 的最低位不使用，对应方式相当于右对齐左移一位；类似的，设置为 010 表示 DAC0H 的高两位不用，设置为 011 表示 DAC0H 的最高位不用，设置为 1xx（后两位可以是任意值）表示仅 DAC0L 的低四位不使用，这种情况被称为左对齐。

对于 DAC 的转换范围来说，最重要的是它的参考电压 Vref，DAC 所输出的模拟电压一般就是 0 ～ Vref。例如对于 C8051F020 所包含的 12 位 DAC，输入为 0 的时候，输出就是 0V；输入 0x800 的时候，输出就是 $Vref/2$；输入 0xFFF 的时候，输出就是 $Vref * (1 - 2^{-12})$。

C8051F020 具有内部的参考电压，可以使用 REF0CN 寄存器进行控制，这个寄存器的定义参考表 4.2。当使用内部参考电压的时候，注意要把 VREF 信号和 VREFD 信号连接，而 VREF 的电压在内部的 2 倍增益下是 2.4V 左右。

AD0VRS：ADC0 的参考电压选择，0 代表 VREF0 管脚、1 代表 DAC0 输出作为参考电压

AD1VRS：ADC1 的参考电压选择，0 代表 VREF1 管脚、1 代表 AV+ 管脚作为参考电压

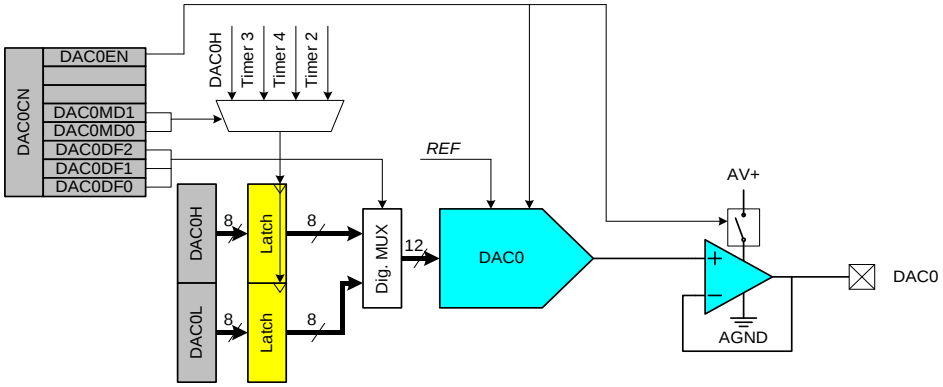


图 4.1: C8051F020 的 DAC

表 4.2: REF0CN 寄存器结构

D7	D6	D5	D4	D3	D2	D1	D0
			AD0VRS	AD1VRS	TEMPE	BIASE	REFBE

TEMPE: 内部的温度传感器允许位，0 代表禁止，1 代表允许

BIASE: ADC/DAC 的偏置电压允许位，在使用 ADC/DAC 的时候必须要设置为 1

REFBE: 内部的参考电压允许位，0 代表禁止，1 代表允许

4.2.2 模数转换器

与数模转换的过程相反，在单片机的测控系统中，经常要把检测到的连续变化的模拟信号，如温度、压力、速度等转换为离散的数字量，才能输入单片机进行处理。实现模拟量到数字量转换的设备就是模数转换器（ADC），简称 A/D。

C8051F020/1 有一个片内 12 位 SAR ADC(ADC0)，一个 9 通道输入多路选择开关和可编程增益放大器。该 ADC 工作在 100ksps 的最大采样速率时可提供真正的 12 位精度。其结构如图 4.2。

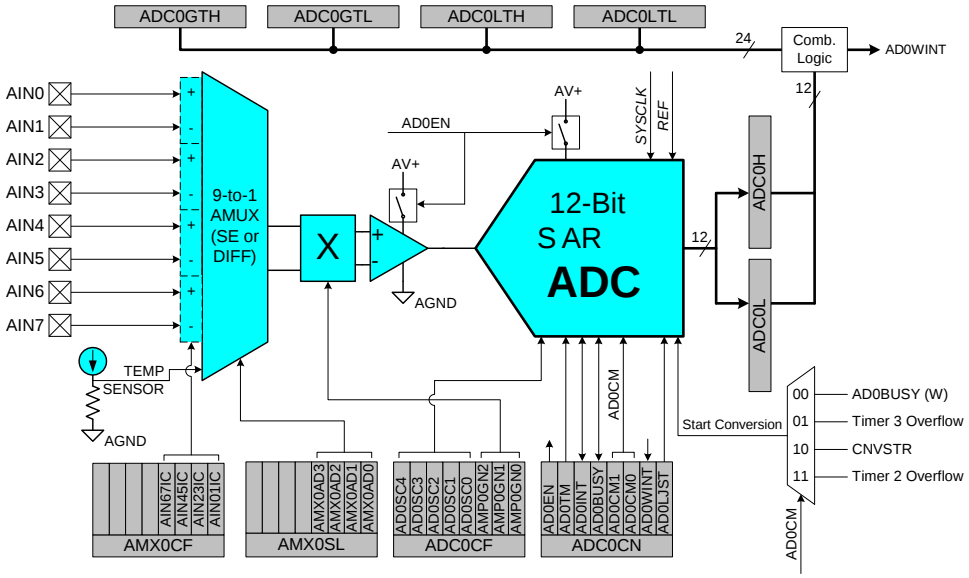


图 4.2: C8051F020 的 ADC

ADC0 共有 9 路输入可以通过模拟开关选择，其中前八个为普通的模拟输入信号，可以通过 AMX0CF 设置成单端输入还是差分输入，其具体定义见表 4.3。具体使用那个输入作为 ADC0 的输入通过寄存器

AMX0SL 寄存器选择，这个寄存器只有低四位有效，在所有输入都为单端输入的时候，设置为 0~7 分别可以用来选择 AIN0~AIN7，而如果设置为 8~15，则用来选择第 9 路输入：内部的温度传感器。

表 4.3: AMX0CF 寄存器结构

D7	D6	D5	D4	D3	D2	D1	D0
				AIN67IC	AIN45IC	AIN23IC	AIN01IC

AIN67IC：用来设置 AIN6 和 AIN7 的输入模式，0 代表单端输入，1 代表差分输入。其余位的设置也类似，分别用来设置另外 6 路输入的模式。

SAR 是 Successive Approximation Register 的缩写，表示通过寄存器逐渐接近模拟电平的数字量。其基本原理如图 4.3所示。

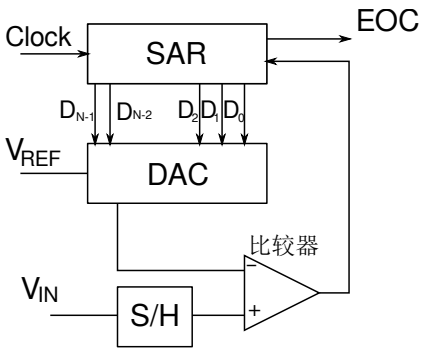


图 4.3: SAR 的基本原理

连续渐进模数转换器主要由四个部分组成：

1. 采样和保持电路取得输入的模拟电压（Vin）
2. 一个内部参考 DAC，用来根据 SAR 的值输出相应的模拟电压
3. 将 Vin 和 DAC 的输出进行比较的模拟比较器
4. 一个可以为 DAC 提供数字输入的寄存器（SAR）

ADC 的 SAR 被初始化为 0，然后先把最高位（MSB，Most Significatn Bit）赋值为 1，此时其 DAC 的输出为 Vref/2，如果此时比较器判断 $Vin > Vref/2$ 则 SAR 的最高位就被置 0，否则最高位置 1。接下来 SAR 的第二位被置 1，同样的测试继续执行就可以判断出这一位应该被设置的值。当最后一位被测试完成的时候，ADC 的过程就结束了（EOC，End of Conversion）。C8051F020 的 ADC 转换时间是 16 个转换时钟周期。

C8051F020 的 ADC0 用 ADC0CF 寄存器设置 SAR 的时钟，ADC0CF 的定义如表 4.4。

表 4.4: ADC0CF 寄存器结构

D7	D6	D5	D4	D3	D2	D1	D0
ADC0SC4	ADC0SC3	ADC0SC2	ADC0SC1	ADC0SC0	AMP0GN2	AMP0GN1	AMP0GN0

其中 ADC0SC4-0 用来设置 SAR 时钟，如果假设这个时钟频率为 CLK_{SAR0}，那么 ADC0SC 的设置为 $ADC0SC = \frac{SYSCLK}{CLK_{SAR0}} - 1$。

AMP0GN2-0 用来设置 ADC0 内部的电压增益，即对 ADC 输入的放大倍数。设置为 000、001、010、011 分别代表增益为 1、2、4、8；设置为 10x 代表增益为 16；设置为 11x 表示增益为 0.5。

ADC0 的控制寄存器为 ADC0CN，其定义如表 4.5

ADC0EN：ADC0 的允许位，0 表示低功耗关闭状态，1 表示可以工作

表 4.5: ADC0CN 寄存器结构

D7	D6	D5	D4	D3	D2	D1	D0
AD0EN	AD0TM	AD0INT	AD0BUSY	AD0CM1	AD0CM0	AD0WINT	AD0LJST

AD0TM：跟踪模式设置，0 表示 ADC0 是连续跟踪的，1 表示仅在 ADC0 启动转换才进行 3 个时钟周期的跟踪以保证转换比较准确

AD0INT：ADC0 的模数转换完成终端标志，1 表示转换完成

AD0BUSY：忙标志，读出 1 表示转换正在进行中

AD0CM1-0：启动 ADC0 转换的方式：00 表示在 AD0BUSY 位写入 “1” 启动转换；01 表示在 Timer3 计数溢出时启动转换；10 表示在 CNVSTR 信号的上升沿启动转换；11 表示在 Timer2 计数溢出时启动转换

AD0WINT：ADC0 的窗口比较中断标志，1 表示中断发生。窗口比较是 ADC0 的特殊功能，可以设置 ADC0GT 寄存器表示 ADC0 转换的上界，设置 ADC0LT 表示 ADC0 转换的下界，当模数转换的时候结果在这个窗口之外，就会发生中断。缺省情况 ADC0GT 为 0xFF，而 ADC0LT 为 0。

AD0LJST：设置为 0 的时候表示数据为右对齐，设置为 1 的时候表示数据为左对齐。

1. 在 ADC0CN 寄存器的 AD0BUSY 位写入 “1”
2. 设置为在 Timer3/Timer2 计数溢出时启动
3. 在 CNVSTR 信号的上升沿

ADC0 在转换完毕之后会产生中断信号，中断编号为 15，入口地址是 0x7B。

4.2.3 定时器/计数器 2

定时器/计数器 2 共有三种工作模式：带捕获的 16 位的定时器/计数器模式，带自装载功能的 16 位定时器/计数器模式，和串口 0 的波特率发生器。其控制寄存器如表 4.6所示。

表 4.6: T2CON 寄存器结构

D7	D6	D5	D4	D3	D2	D1	D0
TF2	EXF2	RCLK0	TCLK0	EXEN2	TR2	C/T2	CP/RL2

TF2 为计数溢出标志，它必须通过软件清除。

EXF2 捕获/自装载事件发生标志，它必须通过软件清除。

RCLK0 设置此位使定时器/计数器 2 作为串口 0 的接收时钟。

TCLK0 设置此位使定时器/计数器 2 作为串口 0 的发送时钟。

EXEN2 设置此位允许 T2EX 管脚作为捕获/自装载事件触发。

TR2 运行控制，类似与 TR0/TR1，设置为 1 允许计数

C/T2 定时器和计数器模式选择，0 表示定时器模式

CP/RL2 捕获/自装载方式选择，0 表示为自装载模式：在 T2EX 管脚发生从高到低跳变或者计数溢出的时候进行自装载。1 表示在 T2EX 管脚发生从高到低跳变时进行捕获。

当 CP/RL2 设置为 1、EXEN2 设置为 1、TR2 有效时，定时器/计数器 2 工作在带捕获的 16 位的定时器/计数器模式，此时 T2EX 管脚上的从高到低跳变会将 16 位计数值（TH2，TL2）保存在捕获装载寄存器（RCAP2H，RCAP2L）中，同时 EXF2 设置为 1，如果中断被允许还会触发中断。

当 CP/RL2 设置为 0、TR2 有效时，定时器/计数器 2 工作在带自装载功能的 16 位的定时器/计数器模式。当计数器发生溢出的时候，保存在 RCAP2H 和 RCAP2L 中的数值会自动装载到 T2H 和 T2L 中。如果 EXEN2 设置有效，T2EX 管脚上的从高到低跳变也会引发自装载行为。

当 RCLK0 或者 TCLK0 设置为 1 的时候，定时器/计数器 2 将作为串口 0 的波特率发生器，工作方式类似于自装载模式，溢出信号作为串口 0 的时钟，TF2 不会被置位。

4.2.4 定时器/计数器 3

定时器/计数器 3 仅可以工作在自装载模式下，其自装载寄存器为 TMR3RLL 和 TMR3RLH，控制寄存器如表 4.7 所示。

表 4.7: TMR3CN 寄存器结构

D7	D6	D5	D4	D3	D2	D1	D0
TF3	-	-	-	-	TR3	T3M	T3XCLK

TF3 为计数溢出标志，它必须通过软件清除。

TR3 运行控制，类似与 TR0/TR1，设置为 1 允许计数。

T3M 时钟源选择，设置 1 使用系统时钟，否则使用系统时钟的十二分支一。

T3XCLK 当设置为 1 时，计数器的时钟直接采用系统晶振输入的八分频，这种方式运行定时器/计数器 3 作为实时钟使用（此时系统时钟可以采用内部时钟源）。

4.2.5 定时器/计数器 4

定时器/计数器 4 的工作模式和定时器/计数器 2 是一致的，这里仅把其控制寄存器的定义写在下面，当其作为波特率发生器时，仅供串口 1 使用，其捕获装载寄存器为 RCAP4H 和 RCAP4L。

表 4.8: T4CON 寄存器结构

D7	D6	D5	D4	D3	D2	D1	D0
TF4	EXF4	RCLK1	TCLK1	EXEN4	TR4	C/T4	CP/RL4

4.3 实验内容

4.3.1 数模转换

使用 DAC 输出三角波，通过示波器验证输入信号的频率与幅度（探头接 TP_DAC，地线接 TP_AGND，）。其中 DAC 使用内部参考电压，参考程序如下。其中为了寄存器的使用方便，定义了三个 16 位的特殊寄存器，在 C51 中可以直接作为 16 位数使用，其地址根据数据手册得出。

```
#include <c8051f020.h>

sfr16 RCAP4    = 0xe4;              // Timer4 捕获/重载
sfr16 T4       = 0xf4;              // Timer4
sfr16 DAC0     = 0xd2;              // DAC0 数据

#define SYSCLK 22118400             // SYSCLK, 系统时钟频率
#define SAMPLERATE 10000          // DAC 采样频率

void main (void);
void SYSCLK_Init (void);
void PORT_Init (void);
void Timer4_Init (int counts);
void Timer4_ISR (void);

void main (void) {
   WDTCN = 0xde;                    // 禁止看门狗复位
```

```
   WDTCN = 0xad;
   SYSCLK_Init ();
   REF0CN = 0x03;                    // 允许内部参考电压并输出
   DAC0CN = 0x97;                    // 允许 DAC0，数据左对齐
                                     // 使用 Timer4 作为启动采样触发
   Timer4_Init(SYSCLK/SAMPLERATE); // 初始化 Timer4
   EA = 1;                           // 允许全局中断
   while (1);                        // 程序结束
}

void Timer4_Init (int counts)
{
   T4CON = 0;                        // 停止计数，设置为自动重载模式
   CKCON |= 0x40;                    // 使用 SYSCLK 作为时钟源
   RCAP4 = -counts;                  // 重载值设置
   T4 = RCAP4;
   EIE2 |= 0x04;                     // 允许 Timer4 中断
   T4CON |= 0x04;                    // 启动 Timer4
}

void Timer4_ISR (void) interrupt 16
{
   static unsigned phase = 0;        // 相位计数
   DAC0 = phase;                     // 输出电压设置
   phase += 0x10;                    // 更新相位计数
   T4CON &= ~0x80;                   // 清除计数溢出标志
}
```

4.3.2 模数转换

ADC0 的输入由变阻器提供，在每次模数转换结束以后，将转换结果输出到数码管显示。部分参考程序如下：

```
#include <c8051f020.h>

sfr16 TMR3RL   = 0x92;            // Timer3 重载值
sfr16 TMR3     = 0x94;            // Timer3 计数值
sfr16 ADC0     = 0xbe;            // ADC0 数据寄存器

#define SYSCLK      22118400      // 系统时钟
#define SAMPLERATE  10000         // ADC0 采样率

void SYSCLK_Init (void);
void ADC0_Init (void);
void Timer3_Init (int counts);
void ADC0_ISR (void);
void Delay(int k);

unsigned sample;                 // ADC0 转换结果

void main (void) {
   WDTCN = 0xde;                 // 禁止看门狗复位
   WDTCN = 0xad;
   SYSCLK_Init ();               // 初始化系统时钟
   Timer3_Init (SYSCLK/SAMPLERATE);   // 初始化 Timer3
   ADC0_Init ();                 // 初始化 ADC0
   PORT_Init();                  // 端口初始化，驱动外存扩展
   EA = 1;                       // 允许全局中断
```

```
    EIE2 |= 0x02;                  // 允许 ADC0 中断
    while (1) {                    // 循环显示结果
        unsigned char i;
        unsigned j;
        j = sample;                // 记录采样结果
        for (i = 0; i < 4; ++i) { // 分别显示四位数据
            seg = segs[j & 0xF];  // 当前位输出到段码
            Delay(1);              // 延时
            cs = css[i];           // 输出位码
            j = j >> 4;;           // 移出最低四位
            Delay(1000);           // 延时
        }
    }
}

void ADC0_Init (void)
{
    ADC0CN = 0x05;                 // 禁止 ADC0, 启动由 Timer3 溢出触发
                                   // ADC0 数据左对齐
    REF0CN = 0x03;                 // 片上参考电压使能, 并输出
    AMX0SL = 0x00;                 // 选择 AIN0 作为输入
    ADC0CF = (SYSCLK/2500000) << 3;     // ADC 时钟为 2.5MHz
    ADC0CF &= ~0x07;               // PGA 增益设置为 1
    EIE2 &= ~0x02;                 // 禁止 ADC0 中断
    AD0EN = 1;                     // 允许 ADC0
}

void Timer3_Init (int counts)
{
    TMR3CN = 0x02;                 // 停止 Timer3; TF3 清零;
                                   // 使用 SYSCLK 作为时钟源
    TMR3RL  = -counts;             // 初始化重载值
    TMR3    = 0xffff;              // 初始化计数值: 下个周期重载
    EIE2   &= ~0x01;               // 禁止 Timer3 中断
    TMR3CN |= 0x04;                // 启动 Timer3
}

void ADC0_ISR (void) interrupt 15
{
    static unsigned int count = 0;
    count++;
    AD0INT = 0;                     // 清除 ADC 转换结束标志
    if (count >= SAMPLERATE / 2) { // 降低 sample 改变的频率
        sample = ADC0;              // 保存转换结果
        count = 0;
    }
}
```

由于每次 AD 的结果会有一定误差,即使不改变电阻器的值读出的转换结果也可能不同,因此如果按照采样频率更新显示,数码管会闪烁不利于读数。示例程序中使用 count 进行计数,每秒仅有两次把 ADC 的结果输出到 sample 变量,使得显示数据更容易读出。

改变变阻器的电阻,用万用表测量几组典型的(最高、最低、半量程等)ADC 的输入电压,并记录相应的数码管显示数值。

4.3.3 模数和数模联调

1. 编写模数和数模的联调程序，模数转换从信号源输入，将转换的结果通过数模转换再输出，输出结果用示波器查看。信号源输入时，需要将 J6 断开，输入接左边的端子，同时要保证信号的输出范围在 0V 和 2.4V 之间，可以采用峰峰值 2V，偏移 1V 的设置。

2. AD/DA 模块上面已经包含了一个简单的录音回放电路（参考附录中的电路图），其中音频输入已经接到了 ADC0 的 AIN1 上（AMX0SL 需要设置为 01），而音频输出将由 DAC1 驱动。请编写一个数模和模数的联合调试程序，首先将录音采样的数据保存在外部存储器，然后再将录音的内容通过 DA 输出播放出来。录音的采样率为 8K（录音回放电路的防混叠滤波器和输出滤波器都是按照 8K 采用率设计的），这样可以保存的录音时间大致为 2 秒。DAC1 的寄存器地址在 0xD5（低位为 0xD5，高位为 0xD6），控制寄存器定义与 DAC0 类似。

外部数据存储器的大小是 32K，因此定义一个可以保存 32K 容量数据的数组如下所示：

```
#define BUF_LEN 16384
unsigned int xdata buf[BUF_LEN] _at_ (0)
```

表 4.9: DAC1CN 寄存器结构

D7	D6	D5	D4	D3	D2	D1	D0
DAC1EN			DAC1MD1	DAC1MD0	DAC1DF2	DAC1DF1	DAC1DF0

4.4 思考题

1. 单片机实现中断的原理是什么？进而说明如何通过汇编语言实现中断程序的设计（给出参考代码）。
2. ADC 和 DAC 为什么要设置多种触发方式？使用计时器溢出触发的好处是什么？
3. MCS-51 为 8 位单片机，C51 中 sfr16 是如何实现的？给 sfr16 赋值的时候是先写高位还是先写低位？

实验五　基于单片机的串行通信

5.1　实验目的

1. 了解串行通信的基本知识；
2. 掌握用单片机串行口实现串行通信的方法。

5.2　实验原理

5.2.1　串行通信的异步和同步传送方式

CPU 与外设的基本通信方式可分为并行通信和串行通信两类。并行通信是指要传输的数据字各二进制位同时并行传送的通信方式，而串行通信是指数据逐位顺序串行传送的通信方式。

在并行通信中，由于有与字宽对应的多根传输线并行传送数据，因此收发电路简单，与传统的总线类似。但当数据远程距离增加时，传输线路的开销和数据位的同步就成为突出问题。而串行通信只需一对传输线，并且可以利用电话线等现有通信信道作为传输介质，因而可以大大降低传输线路的成本，在实际应用中更为常见。目前由于串行通信的时钟不断提高，串行通信的传送速度明显高于并行通信。

串行通信分为异步传送和同步传送两类。异步通信在发送字符时，发送端可以在任意时刻开始发送字符，因此必须在每一个字符的开始和结束的地方加上标志，即加上开始位和停止位，以便使接收端能够正确地将每一个字符接收下来。单片机串行通信就是一直异步通信方式。

异步传送的特点是：

1. 数据以字符方式随机且断续地在线路上传送（但在同一字符的内部的传送是连续的）。各字符的传送依发送方的需要可连续，也可间断。
2. 通信双方用各自的时钟源来控制发送和接收。
3. 通信双方按异步通信协议传输字符。

异步通信格式如图 5.1所示，每个字符由起始位、数据位、奇偶校验位和停止位四个部分顺序组成。这四个部分组成异步传输中的一个传输单元，即字符帧。

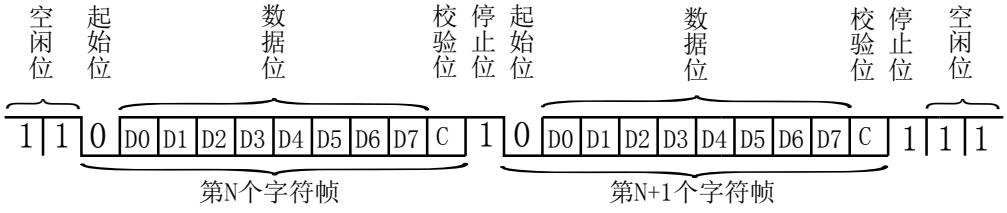


图 5.1: 异步通信字符帧格式

起始位为“0”信号，占 1 位。起始位的作用有两个：

1. 表示一个新字符帧的开始。即线路上不传送字符时，应保持为“1”。接收端检测线路状态连续为“1”后或在停止位后有一个“0”，就知道将发来一个新的字符帧。
2. 用以同步接收端的时钟，以保证后续的接收能正确进行。

数据位紧接于起始位后面，它可以占 5、6、7 或 8 位不等，数据的位数依最佳传送速率来确定。如所传数据为 ASCII 码字符，则常取 7 位。数据位传输的顺序，总是最低位（LSB）D0 在先。

奇偶校验位在数据位之后占 1 位，用来检验信息传送否有错。它的状态常由发送端的奇偶校验电路确定。奇偶位的值取决于校验类型，若为偶校验，则数据位和校验位中逻辑“1”的个数必须是偶数；若为奇校验，则数据位和校验位中逻辑“1”的个数必须是奇数。也可以规定不用奇偶校验位，或用其它的校验方法来检验信息传送过程是否有错。

停止位用“1”来表征一个字符帧的结束。停止位可以占 1 位、1.5 位或 2 位不等。接收端收到停止位时，表明这一字符已接收完毕，也表明下一个字符帧可能到来。若停止位以后不是紧接着传送下一个字符帧，则让线路上保持为“1”，即空闲等待状态。图 5.1既表示一个字符紧接一个字符传送的情况，又表示两个字符间有空闲位的情况。

串行通信的一个重要指标是波特率。它定义为每秒钟传送二进制数码的位数，以“位 / 秒”（bps）为单位。在异步通信中，

波待率 =（每个字符帧的位数）×（每秒传送的字符数）

常用的波特率有 600、1200、2400、4800、9600、19200（bps）等。

由于异步通信双方各用自己的时钟源，若时钟频率等于波特率，则频率稍有偏差就会产生接收错误。时钟频率应比波特率高，时钟频率与波特率的比一般选 16:1 或者 64:1。采用较高频率的时钟，在一位数据内就有 16 或 64 个时钟，就可以保证捕捉正确的信号。

因此，在异步通信中，收发双方必须事先约定两件事：一是规定字符帧格式，即规定字符各部分所占的位数，是否采用校验，以及校验的方式等；二是规定所采用的波特率以及时钟频率和波特率间的比例关系。异步传送由于不传送同步时钟脉冲，所以设备比较简单，实现起来方便，它还可根据需要连续地或有间隙地传送数据，对各字符间的间隙长度没有限制。缺点是在数据字符串中要加上起同步作用的起始位和停止位，降低了有效数据位的传送速率，仅适合于低速通信的场合。

5.2.2 MCS-51 系列单片机的串行通信接口

MCS 一 51 系列单片机内部有一个可编程的全双工串行通信口，可作为通用异步接收和发送器，也可作为同步移位寄存器用。该串行口有 4 种工作模式。片内的定时器 / 计数器可用作波特率发生器。接收、发送均可工作在查询方式或中断方式。

MCS 一 51 系列单片机内部的串行通信口，有二个物理上相互独立的接收、发送缓冲器 SBUF，对外也有两条独立的收、发信号线 RxD 和 TxD。可以同时发送、接收数据，实现全双工传送。发送缓冲器和接收缓冲器不能互换，发送缓冲器只能写入不能读出，接收缓冲器只能读出不能写入。两个缓冲器占用同一个端口地址（99H）。具体对哪一个缓冲器进行操作，取决于所用的指令是发送还是接收。

接收是双缓存的，以避免在接收下一帧数据之前，CPU 未能及时响应接收中断，未把上一帧数据取走而产生两帧数据重叠的问题。而对于发送器，因为发送时 CPU 是主动的，不会产生写重叠的问题，所以不需要双缓存。

与串行通信口有关的寄存器有多个，除 SBUF 之外，还有 SCON、PCON、IE 和定时器/计数器，用校验方式进行通信，有时也会用到程序状态字寄存器 PSW。

SCON 用于控制和监视串行口的工作状态，定义如表 5.1：

表 5.1: 串口控制寄存器 SCON

D7	D6	D5	D4	D3	D2	D1	D0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0 和 SM1：串行口工作模式选择位，对应四种模式，见表 5.2。

SM2：在模式 0 时，SM2 不用，应设置为 0。在模式 1 时，SM2 一般也应设置为 0。若 SM2 = 1 ，则只有收到有效停止位才会激活 RI，并自动发出串行口中断请求（设中断是开放的），若没有接收到有效停止

表 5.2: 串口工作模式选择

SM0	SM1	模式	功能	波特率
0	0	0	同步移位寄存器	$f_{osc}/12$
0	1	1	8 位 UART	可变
1	0	2	9 位 UART	$f_{osc}/64$ 或 $f_{osc}/32$
1	1	3	9 位 UART	可变

位，则 RI 清零。

在模式 2 或模式 3 下，SM2 的设置与字符帧第九位的作用有关。

1. 在第九位用作奇偶位的情形，应置 SM2=0。
2. 在第九位用于表示是地址帧还是数据帧的多机通信情形，若 SM2 = 1 和 RB8 = 1 时，RI 不仅被激活，而且可以向 CPU 请求中断；若 SM2 = 0，串行口以单机发送或接收方式工作，TI 和 RI 以正常方式被激活。

REN：允许接收控制位，由软件置位或清除。REN=1 则允许接收, REN=0, 禁止接收。

TB8：该位是模式 2 和 3 中要发送的第九位数据。在许多通信协议中, 该位是奇偶位，可以按需要由软件置位或清除。在多机通信中, 该位用于表示是地址帧还是数据帧。

RB8：该位是模式 2 和 3 中已接收的第九位数据 (可能是奇偶位, 或是地址帧/数据帧标识位)。在模式 1 中, 若 SM2=0, RB8 是已接收的停止位。在模式 0 中, RB8 未用。

TI：发送中断标志。在模式 0 中, 在发送完第 8 位数据时, 由硬件置位；在其他模式中, 在发送停止位之初, 由硬件置位，申请中断, CPU 响应中断后, 发送下一帧数据。在任何模式中, 都必须由软件清除 TI.

RI：接收中断标志。在模式 0 中, 接收第 8 位数据结束时, 由硬件置位；在其他模式中, 在接收停止位的半中间, 由硬件置位，申请中断, 要求 CPU 取走数据。但在模式 1 中, SM2=1 时, 若未接收到有效的停止位, 则不会对 RI 置位。在任何模式中，都必须由软件清除 RI.

与串行通信相关的寄存器还有 PCON 和 IE，具体如下：

- PCON 中有与串行口通信波特率有关的控制位 SMOD（第 7 位），SMOD = 1 时波特率加倍
- IE 中的 ES 位为串行口中断控制位，ES=1 且总中断允许位 EA=1 时，允许串行口中断

缺省情况下串口 0 使用定时器/计数器 1 作为波特率发生器，在模式 1 和模式 3 下，串行通信的波特率计算公式如下：

$$\text{波特率} = \frac{2^{SMOD}}{32} \times (\text{定时器溢出速率}) = \frac{2^{SMOD}}{32} \times \frac{f_{osc}}{12^{(1-T1M)} \times (256 - TH1)}$$

其中 f_{osc} 为晶振频率, TH1 为定时器/计数器 1 的重装载值，定时器/计数器 1 工作于自动重装载模式, 即模式 2。定时器 1 中断应禁止。

5.3 实验内容

5.3.1 观察 UART 通信波形

设定串行口工作模式 1（10 位异步方式）, 用 1200bps 循环发送一个字节 55H 或 8AH。用示波器观测 TxD 的电平，分析信号的帧结构，标出起始位、数据位和终止位, 用示波器测量码元宽度，给出 1200bps 波特率条件下的 TH1 计算值、码元宽度的计算值和测量值。

参考程序如下：

```
#include <c8051f020.h>
```

```
void PORT_Init()
{
    XBR0     = 0x04;            // UART0 使能
    XBR2     = 0x40;            // 使能交叉开关
    P0MDOUT |= 0x01;            // TX0 设置为推挽输出
}

void UART0_Init (void)
{
   SCON0  = 0x50;                       // SCON0: 串口方式1 使能RX
   TMOD   = 0x20;                       // 定时器 1 采用自装载模式
   TH1    = -(SYSCLK/BAUDRATE/16/12);     // Timer1 载入值
   TR1    = 1;                          // 启动 Timer1
   PCON  |= 0x80;                       // SMOD0 = 1
}

void main()
{
    WDTCN=0XDE;             // 禁止看门狗
    WDTCN=0XAD;
    SYSCLK_Init();          // 系统时钟初始化（代码省略）
    PORT_Init();            // 端口初始化
    UART0_Init();           // 串口初始化
    while(1)
    {
        SBUF0 = 0x8A;       // 通过串口发送数据
        while(!TI0);
        TI0=0;
    }
}
```

开发板的串口连接到了板上的 FT232RL 芯片上，这个芯片是一款 USB 串口设备。因此当使用 USB 线将开发板和 PC 机连接以后，PC 机会识别出一个串口设备，而这个串口设备将和 C8051F020 芯片的串口相连接，可以使用 PC 的串口调试工具和 C8051F020 单片机进行通信。此时通过串口调试工具查看接受到的数据—注意要确认波特率和串口号正确—是否和程序中的一致。

图 5.2 为 C8051F020 与 PC 的连接示意图，可以理解为 PC 增加了一个 USB 串口外设，这个串口的发送和接收和 C8051F020 的串口交叉相连。因此 C8051F020 通过串口 0 发送的数据就可以被 PC 接收，而 PC 通过这个 USB 串口所发送的数据也会被 C8051F020 接收到。在图中两个串口仅连接了 TxD 和 RxD 两个信号，这是因为开发板上已经共地。如果要连接两个不同外设的串口，则至少需要三根信号线，另外增加的就是地线。

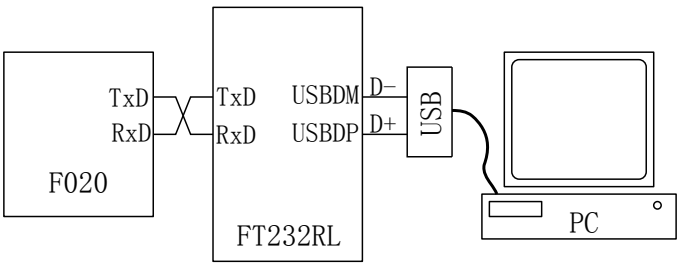


图 5.2: C8051F020 串口与 PC 连接示意图

5.3.2 串口收发实验

单片机串口发送和接收数据的过程都可以通过中断方式完成，一个带缓冲的串口收发示例程序如下：

```
#define RX_LEN 20

char RxBuf[RX_LEN];
char* TxBuf;
unsigned char RxIdx;

void UART0_ISR (void) interrupt 4
{
    char c;
    if (RI0 == 1) {             // 接收中断
        RI0 = 0;                // 清中断标记
        c = SBUF0;              // 读取接收到的数据
        RxBuf[RxIdx] = c;       // 存入缓冲区
        if (RxIdx <= RX_LEN)
            RxIdx++;            // 接收长度增加
    } else if (TI0 == 1) {      // 发送中断
        TI0 = 0;                // 清中断标记
        c = *TxBuf;             // 获取发送字符
        if (c != '\0') {
            TxBuf++;            // 发送指针增量
            SBUF0 = c;          // 写入发送缓冲区
        }
    }
}

void main (void) {
    WDTCN = 0xde;   // 禁止看门狗
    WDTCN = 0xad;
    SYSCLK_Init (); // 初始化时钟
    PORT_Init ();   // 初始化交叉开关和端口
    UART0_Init ();  // 初始化 UART0
    ES0 = 1;        // 允许串口中断
    EA = 1;         // 允许全局中断
    TxBuf = "Hello";
    TI0 = 1;        // 启动发送
    RxIdx = 0;
    while(1);
}
```

编写一个程序使单片机通过 4×4 键盘输入数据，并把这个数据通过串口发送给 PC，同时在 PC 端通过串口调试工具发送数据给单片机，单片机把接收到的数据显示到数码管上。

5.3.3 串口作为标准输入输出

在标准 C 语言中非常常用的标准输入输出函数 printf/scanf 在 Keil C 中也是被支持的，由于单片机没有标准的键盘和显示设备，一般情况下用户需要提供自己的 getchar 和 putchar 函数来实现底层的输入输出操作。但如果用户在程序中初始化了 UART0，则 Keil C 会将这个串口作为标准输入输出设备，用户可以直接使用 printf/scanf 从串口输入和输出数据。例如：

```
#include <c8051f020.h>
#include <stdio.h>

int main(void)
{
    int ac;
    char d;
    int b;
```

```
    WDTCN=0XDE;             // 禁止看门狗
    WDTCN=0XAD;
    SYSCLK_Init();          // 系统时钟初始化（代码省略）
    PORT_Init();            // 端口初始化
    UART0_Init();           // 串口初始化
    TI0 = 1;                // 准备好发送数据
    printf("Enter a signed byte and an int\r\n");
    ac = scanf("%bd %d", &d, &b);
    printf("%bd %d\r\n", d, b);
    while(1);
}
```

编写一个通过串口工具进行操作的计算器，当开发板通过串口和串口工具通信时，用户可以输入两个整数（char 类型），并根据选择对这两个整数进行加减乘除运算，将结果输出到串口。

5.4 思考题

1. 串口收发两端波特率不同就可能带来接收错误，试举例说明错误的原因。
2. 在我们的开发板上，应如何设置串口的波特率为 230400bps？如果要设置成 300bps 呢？

实验六　SPI 总线

6.1　实验目的

1. 了解 SPI 总线的基本时序
2. 了解串行 FLASH 芯片的基本原理
3. 掌握串行 FLASH 芯片的基本用法

6.2　实验原理

6.2.1　SPI 总线简介

SPI（serial peripheral interface）总线是一种同步串行总线标准。SPI 最初由摩托罗拉公司提出，支持在主设备与从设备之间的全双工通信。数据传输由主设备发起，同一个主设备可以连接多个从设备，但每个从设备必须具有独立的片选信号。SPI 总线包含 2 条数据信号线和 2 条控制信号线：

- MOSI（master out slave in）：由主设备传递到从设备的数据线。
- MISO（master in slave out）：由从设备传递到主设备的数据线。同一时间不允许有多于一个从设备进行数据传输。
- SPCK（serial clock）：由主设备驱动的时钟信号，用来控制数据线的传输速度，每个时钟周期传递一个数据位。
- NSS（slave select）：用来选择从设备的控制线。

SPI 总线的优点主要有以下几个方面：

- 全双工通信，较大的数据吞吐率。
- 协议灵活，支持不同的字长和时钟特性。
- 硬件接口简单，从设备不需要精确的时钟，不需要地址译码，不需要总线仲裁。
- 信号单向传输，便于电流隔离。

SPI 总线的缺点主要有如下方面：

- 相对其他串行协议，需要的管脚较多。
- 没有流控信号，没有从设备应答机制。
- 只支持一个主设备。
- 数据传输的距离比较短。

支持 SPI 协议的器件很多，在单片机系统中有很广泛的应用。典型的 SPI 总线连接模式如图 6.1 所示：

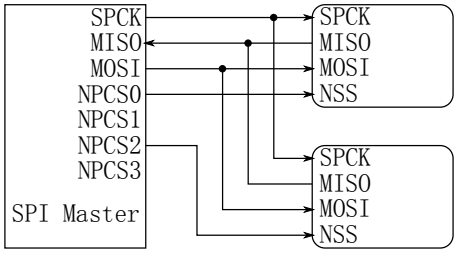


图 6.1: SPI 总线的连接模式

表 6.1: SPI 配置寄存器 SPI0CFG

D7	D6	D5	D4	D3	D2	D1	D0
CKPHA	CKOPL	BC2	BC1	BC0	SPIFRS2	SPIFRS1	SPIFRS0

6.2.2 C8051F020 的 SPI 接口

C8051F020 的 SPI 控制器可以工作在主模式或从模式下，相应的控制和数据信号可以通过 XBR（参考图 3.1）进行设置，其中 NSS 为输入，在从模式下作为片选。SPI 控制器的寄存器主要有如下几个：

CKPHA 用来设置 SPI 时钟的相位。

CKPOL 用来设置 SPI 时钟的极性。CKPHA 和 CKPOL 互相配合可以实现多种时钟形式，其信号时序如图 6.2 所示。

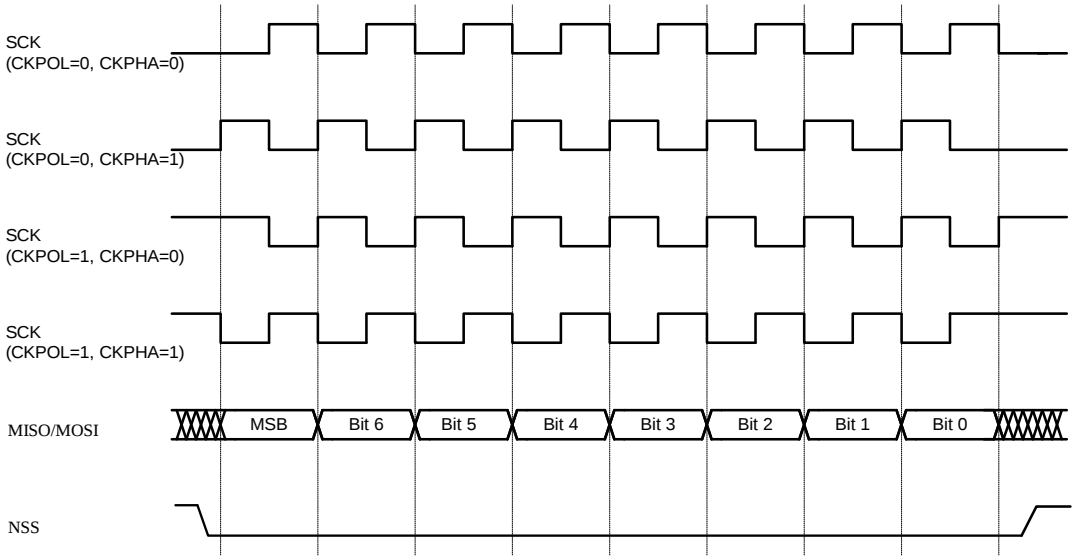


图 6.2: SPI 总线信号

BC2-BC0 用来获得当前帧已经发送的比特数目。

SPIFRS2-SPIFRS0 用来设置帧的大小，其值为 0 ～ 7，表示帧位数为 1 ～ 8。

SPIF SPI 中断标志位，必须由用户清除

WCOL 写入碰撞标志，表示当写入数据寄存器的时候，当前帧还没有发送完成。此标志须由用户清除

MODF 主模式碰撞标志，表示在主模式下 NSS 采样为低。此标志须由用户清除

RXOVRN 接收溢出标志，表示收到新的数据帧时，旧的数据帧仍没有被读取。此标志须由用户清除

TXBSY 发送忙标志，开始发送的时候被硬件置 1，发送完成自动清 0

SLVSEL 从设备选中标志，当 NSS 为低时置 1，NSS 为高时置 0

MSTEN 主模式允许位

SPIEN SPI 允许位

表 6.2: SPI 控制寄存器

D7	D6	D5	D4	D3	D2	D1	D0
SPIF	WCOL	MODF	RXOVRN	TXBSY	SLVSEL	MSTEN	SPIEN

SPI 控制器还有另外两个寄存器，SPI0CKR 用来设置 SPI 的时钟频率，具体频率的计算公式如下：

$$f_{SCK} = \frac{SYSCLK}{2 \times (SPI0CKR + 1)}$$

SPI0DAT 用来保存 SPI 传输的数据内容。写入 SPI0DAT 的数据将在下一次 SPI 传输中传输出去，同时 SPI0DAT 中将保存最近一次接收到的数据。

SPI 控制器的设置过程如下代码所示：

```
#define SPICLK 2000000
void PORT_Init()
{
    XBR0 = 0x06;     // 允许 SPI 和 UART
    XBR2 = 0x40;     // 允许 XBR
    P0MDOUT |= 0x15; // TX, SCK, MOSI 设置为推挽输出
    P74OUT = 0x20;   // p6.7 用作片选信号
}

void SPI0_Init()
{
    SPI0CFG = 0x07;  // 8 位帧大小
    SPI0CN = 0x03;   // 主模式, 允许 SPI 设备
    SPI0CKR = SYSCLK/2/SPICLK;
}
```

通过 SPI 总线发送数据的代码如下所示，由于 SPI 总线总是在发送的同时进行数据接收，所以即使只是从总线读取数据也要在写入一个任意数据的同时获得读入数据。

```
unsigned char SPI_Write(unsigned char v)
{
    SPIF = 0;          // 清除中断标志
    SPI0DAT = v;       // 数据寄存器赋值
    while (SPIF == 0); // 等待发送完成
    return SPI0DAT;    // 同时把接收到的结果返回
}
```

由于硬件上 SPI 引脚使用的是 P0.2 ∼ P0.5，所以在使用 SPI 必须同时允许串口 0。

6.2.3 SPI Flash 的使用

Flash（闪存）是一种非易失性存储器，可以为电子设备提供大量的数据存储空间。目前的 Flash 芯片主要有两种类型，一种是 NAND 型，一种是 NOR 型。两种 Flash 芯片都要求在写入之前先进行擦除操作，但在使用上的主要差别是 NOR 设备可以按照地址进行随机访问读写，但 NAND 设备只能进行按页进行访问。

实验板上包含的 Flash 芯片型号是 W25Q64，是一款 SPI 接口的串行 Flash 芯片。它具有 64M 比特的存储空间，分成 32768 个 256 字节的页，清除操作可以每次操作 16、128、256 或者全部的页，最大的 SPI 时钟频率可以是 80MHz。对 W25Q64 的操作都是通过 SPI 总线进行，它支持的部分 SPI 命令如表 6.3 所示。

W25Q64 在上电之后处于写保护状态，如果要执行写操作（包括写状态字和芯片清除）必须先执行写入允许命令，而当写入完成之后，芯片会自动进入写保护状态。W25Q64 的状态字一共有 16 位，其中只有

表 6.3: W25Q64 芯片的部分命令

功能	命令字	参数 1	参数 2	参数 3	参数 4	参数 5
写入允许	06h	-	-	-	-	-
写入禁止	04h	-	-	-	-	-
读状态字 1	05h	(S7-S0)	-	-	-	-
读状态字 2	35h	(S15-S8)	-	-	-	-
写状态字	01h	(S7-S0)	(S15-S8)	-	-	-
页写入	02h	A23-A16	A15-A8	A7-A0	(D7-D0)	...
4k 清除	20h	A23-A16	A15-A8	A7-A0	-	-
芯片清除	60h	-	-	-	-	-
读 JEDEC ID	9Fh	(MF7-MF0)	(ID15-ID8)	(ID7-ID0)	-	-
读数据	03h	A23-A16	A15-A8	A7-A0	(D7-D0)	...

10 位有定义，我们这里仅介绍其中的两位（其它的状态字和写入保护等其它命令相关，具体功能可以查看手册）。

S0 是 BUSY 位，当 S0 位为 1 时，芯片只能接收读状态字和暂停写入（表中未列出）两种命令，在读状态字的时候，只要片选一直有效，从 SPI 总线上可以一直读到当前的芯片状态，便于程序通过忙等的方式等待芯片可以继续工作。

S1 是 WEL（Write Enable Latch）位，当其设置为 1 时表示芯片可以写入，它可以通过“写入允许”命令进行设置。

页清除命令可以将 Flash 的存储位置置为 0FFh，表 6.3 中仅列出了“4k 清除”和“整片清除”两个命令。其中 4k 清除命令后要接着给芯片输入一个 24 位的地址数据，例如第一个 4k 的地址开始为 000000h，第二个 4k 的地址开始为 001000h。真正的清除操作从芯片接收到全部地址数据并且片选信号无效（从低到高）开始，可以通过状态字的 BUSY 位来查看清除命令是否执行完毕。

页写入命令的参数同样包含一个 24 位的地址和至少一个 8 位数据。只要片选持续有效，写入操作可以继续进行，没写入一个字节，地址自动加 1，但不能跨越页的边界。也就是说每次写入最多可以写入 256 个字节（此时必须从一个页的开始写到一个页的结束），如果发生地址越界，地址会回卷到页的开始。如果在写入操作过程中，写入到一个没有擦除过的位置，写入的结果可能是错误的。

W25Q64 的读操作可以从任意的一个 24 位地址开始读起，然后只要片选保持有效可以一直持续读取后续地址的字节内容，直到最高位地址的数据被读出。也就是说读操作不受页面大小的限制，可以跨越页的边界。

JEDEC 是一个电子工程的标准组织，Flash 芯片中的 JEDEC ID 是对它的唯一标识，其中 MF7-MF0 是厂商标识，ID15-ID8 是存储器类型标识，ID7-ID0 是存储器容量标识。下面的示例代码对 SPI 设备进行了初始化并读取了 Flash 芯片的 JEDEC ID。其中 Timer0_us() 是一个延时 1 微秒的函数，并不要求太精确，仅仅为了保证片选信号能够有效的建立。

```
void main()
{
    unsigned char v;
    WDTCN = 0xde;
    WDTCN = 0xad;
    SYSCLK_Init();      // 系统时钟设置
    PORT_Init();        // IO 管脚设置
    P6 = 0x80;          // 片选无效
    UART0_Init();       // 初始化串口
    SPI0_Init();        // 初始化 SPI
    Timer0_us(1000);    // 延时
    P6 = 0x00;          // 片选有效
    Timer0_us(1);       // 延时
```

```
    SPI_Write(0x9F);    // 写入读 JEDEC ID 命令
    v = SPI_Write(0x00);  // 从 SPI 设备读出厂商标识
    printf("Manufacturer ID: %bx\r\n", v);
    v = SPI_Write(0x00);  // 从 SPI 设备读出存储器类型
    printf("Memory Type ID: %bx\r\n", v);
    v = SPI_Write(0x00);  // 从 SPI 设备读出容量
    printf("Capacity ID: %bx\r\n", v);
    Timer0_us(1);       // 延时
    P6 = 0x80;          // 片选无效
    while(1);
}
```

6.3 实验步骤

6.3.1 示例代码验证

实验箱的录音回放模块包含一个 W25Q64 芯片，通过 SPI 接口与 C8051F020 相连。由于 SPI 中的 MISO 和 SCK 管脚与其它管脚复用，所以在实验前需要检查板上的两个跳线设置为允许 SPI 相关信号（跳线帽联通上面两个管脚）。使用 USB 线连接微机，利用串口调试软件观察串口输出数据，此时运行单片机程序就应该可以在软件中看到打印的消息，如果打印的读出内容都是 0FFh 则表示读出错误。

6.3.2 读写 Flash 设备

利用串口设计一个 Flash 操作界面，主要支持下面几个命令：

1. 显示命令 d：命令格式为“d <address>”，读出对应 Flash 地址开始 16 个字节的内容，用 16 进制的方式显示出来。
2. 写入命令 w：命令格式为“w <address> <data>”，将十六进制的 <data> 内容写入到对应 Flash 地址。
3. 清除命令 c：命令格式为“c <address>”，将地址所在的 4k 块进行清除操作，使得其中内容全部变成 0xFF。

代码框架和部分代码可以参考下面示例，注意片选信号的设置和 scanf 函数的用法。

```
while(1) {
    do { // 过滤掉回车和空格字符, 读取命令字节
        c = getchar();
    } while ((c == ' ') || (c == '\r') || (c == '\n'));
    scanf("%lx", &add);  // 读入地址值
    switch (c) {
        case 'd': // 显示命令
            // 显示代码省略
            break;
        case 'w': // 写入命令
            scanf("%bx", &v1); // 继续读入要写入的字节内容
            P6 = 0x00;          // 片选有效
            Timer0_us(1);
            SPI_Write(0x06);   // 写入允许命令
            Timer0_us(1);
            P6 = 0x80;          // 片选无效
            Timer0_us(1);
            P6 = 0x0;           // 再次设置片选
            Timer0_us(1);
            SPI_Write(0x02);   // 写命令
            SPI_Write((add & 0x00FF0000) >> 16);
            SPI_Write((add & 0x0000FF00) >> 8);
```

```
                SPI_Write(add & 0x00FF); // 24 位地址
                SPI_Write(v1);     // 数据
                Timer0_us(1);
                P6 = 0x80;         // 片选无效
                Timer0_us(1);
                busywait();        // 读取状态等待写入完成
                printf("\r\nW %lx %bx OK\r\n", add, v1);
                break;
            case 'c': // 清除命令
                // 代码省略，需要先写入06号命令再写入20号命令
                break;
            default:
                printf("\r\rWrong command!\r\n");
        }
    }
```

一个参考的操作过程如下所示，其中注释内容是对串口数据的解释：

```
c 0         // 清除命令输入
C 0 OK      // 输出内容，表示清除命令完成
w 0 14      // 写入命令，在地址 0 写入十进制的 14
W 0 14 OK      // 输出写入结果
d 0         // 显示命令，显示地址 0 位置数据，下面是结果
14 FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF
D 0 OK
```

6.4 思考题

1. 对于已经写入 0F4h 内容的 Flash 单元，再次写入 04Fh 会有什么结果？
2. 使用 scanf 函数的时候，如果格式字符串中的数据类型定义与实际变量的定义不同会有什么结果？

实验七　I²C 总线

7.1　实验目的

1. 了解 I²C 总线的基本原理
2. 掌握使用 C8051F020 作为主设备进行 I²C 通信的方法
3. 了解时钟芯片 DS1307 的使用方法

7.2　实验原理

7.2.1　I²C 总线的基本原理

I²C 总线标准是一种具有多主设备、多从设备支持的串行总线。I²C 标准最初由 Philips 公司提出，主要面向一些低速的接口设备，如模数/数模转换芯片和各类传感器芯片等。在 C8051F020 芯片的数据手册中，支持 I²C 的设备被称作 SMB（System Management Bus），它即可以作为总线上的主设备也可以作为从设备，本实验内容仅介绍主设备的使用方式。

I²C 的总线规定了两条信号，其中 SDA 为双向的数据信号，SCL 为双向的时钟信号。这两个信号都被设计成漏极开路，因此可以同其他 I²C 设备进行线与。典型的 I²C 总线拓扑结构如图 7.1 所示。

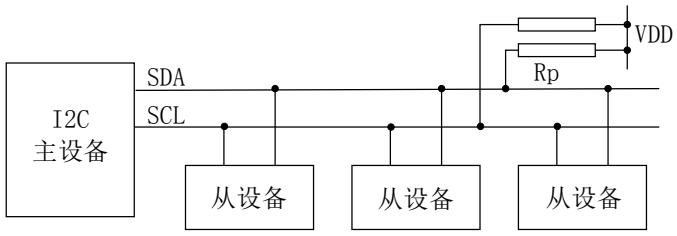


图 7.1: I²C 总线的连接

I²C 设备在进行数据传输的时候，数据信号只在时钟信号为低时变化，在时钟上升沿进行采样。而如果在时钟信号为高的时候数据信号发生变化，则代表两种特殊的控制信号：起始位和停止位。其中数据信号由高变低代表起始位，表示一次数据传输的开始；数据信号由低变高代表停止位，表示一次数据传输的结束。图 7.2 为 I²C 总线的时序图。

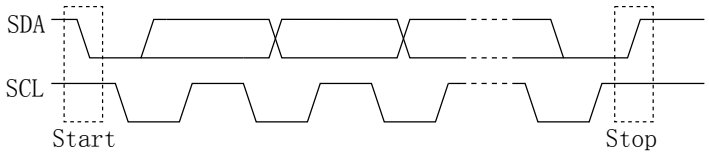


图 7.2: I²C 总线的时序

I²C 数据传输由主设备产生一个起始位开始，然后传递 7 个地址位（对于 7 位地址模式），指定通信的从设备。接下来传递的一位是读写位，0 表示写入，1 表示读出。再接下来的一位由从设备驱动，如果从设备的地址与主设备发出的地址相同，从设备将把 SDA 信号拉低，表示确认这次数据传输（ACK）。当从设

备地址被确认，真正的数据传输就开始了。如果是写入数据，则数据线仍由主设备驱动，从设备在每个字节传输之后也要驱动一位的 ACK 信号。如果是读出数据，则数据线由从设备驱动，主设备仅在应答时输出 ACK 位。最后由主设备产生一个停止位表示数据传输结束。所有的 I^2C 数据与地址，都是先传递最高位，最后传递最低位。

7.2.2 C8051F020 的 I^2C 接口

C8051F020 与 I^2C 相关的寄存器有控制寄存器 SMB0CN（参见表 7.1）、时钟频率寄存器 SMB0CR、数据寄存器 SMB0DAT、地址寄存器 SMB0ADR 和状态寄存器 SMB0STA。

表 7.1: SMB 控制寄存器

D7	D6	D5	D4	D3	D2	D1	D0
BUSY	ENSMB	STA	STO	SI	AA	FTE	TOE

BUSY 是忙状态标志，读取值为 1 的时候表示设备忙

ENSMB 是允许标志，设置为 1 使 SMB 设备被激活

STA 是起始位控制，对于主设备模式，当此位设置为 1 的时候会在总线上发送起始位（前提是总线空闲，否则需要等到接收到一个停止位后再发送起始位）。如果在发送了部分数据之后再设置 STA 位，就会发送一个重复起始状态。为了保证总线工作正常，在设置 STA 之前必须保证 STO 位没有被设置。

STO 是结束位控制，设置为 1 的时候会发送一个停止位，当设备接收到一个停止位后，硬件将设置 STO 为 0。如果 STA 和 STO 同时被设置，一个起始位会在停止位后立即发出。

SI 是中断标志位，当 SMB 设备进入到 27 种不同的工作状态时都会触发中断，用户代码往往会在中断代码中对不同的状态进行处理。这个标志必须由软件进行清除。

AA 是应答标志，在应答过程中，设置此位导致发送 ACK 信号，否则发送 NACK 信号（否认应答）。

FTE 是定时释放总线控制，当设置为 1 的时候，如果 SCL 保持高的时间超过了限定（50 微秒）就会导致超时。

TOE 类似于 FTE，当设置为 1 的时候，如果 SCL 保持低的时间超过了限定（25 毫秒）就会导致超时。

SMB0CR 寄存器中是一个 8 位的无符号数，它将被置于一个专用计数器进行加一运算，当计数溢出的时候 SCL 会进行极性反转，因此 SCL 的频率可以大致表示如下：

$$F_{SMB} = \frac{SYSCLK}{2 * (256 - SMB0CR)}$$

SMB0DAT 寄存器用来保存即将发送的 I^2C 数据，软件只有在 SI 位被设置的时候才可以访问这个寄存器，以免获得不正确的数据。

SMB0ADR 寄存器在从设备模式下保存了设备的地址。其中高 7 位为具体的地址，最低位用来标识该设备是否相应全零地址的请求。

SMB0STA 寄存器保存了 SMB 设备的当前工作状态，表 7.2 是主设备需要处理的几种状态。

图 7.3 是主设备写入数据的时序图，图 7.4 是主设备读入数据的时序图。

7.2.3 DS1307 的软件接口

DS1307 是一款低功耗的实时钟芯片，它可以为软件提供实时时钟的功能，包括年月日时分秒和星期数据的记录和跟踪，并可以正确处理闰年。如果配备有后备的电池，它可以在主电源无效的时候依然保持时钟的更新，这在需要保持准确日历计时的应用中非常方便。

DS1307 包含 64 字节的可由电池保持的寄存器，其中 8 个字节为时钟数据，另外 56 个字节可以作为非易失 RAM 使用。具体的定义如表 7.3 所示。

表 7.2: SMB 状态寄存器

状态码	状态	常见的程序动作
0x08	起始位已发送	SMB0DAT 中写入从设备地址和读写状态，清除 STA
0x10	重复起始位已发送	同上
0x18	从设备地址 + 写已发送，收到 ACK	SMB0DAT 中加载要发送的数据
0x20	从设备地址 + 写已发送，收到 NACK	终止（STO）或重试（STO+STA）
0x28	数据已发送，收到 ACK	SMB0DAT 加载下一个数据或者终止（STO）
0x30	数据已发送，收到 NACK	重传或者终止（STO）
0x38	失去总线（多个主设备竞争）	等待下次传输机会
0x40	从设备地址 + 读已发送，收到 ACK	如果只接收一个字节，清除 AA，等待接收
0x48	从设备地址 + 读已发送，收到 NACK	终止（STO）或重试（STO+STA）
0x50	数据已收到，ACK 已发送	读取 SMB0DAT，等待接收下一个字节，如果下一个是最后字节，清除 AA
0x58	数据已收到，NACK 已发送	设置 STO
0x00	总线错误	设置 STO

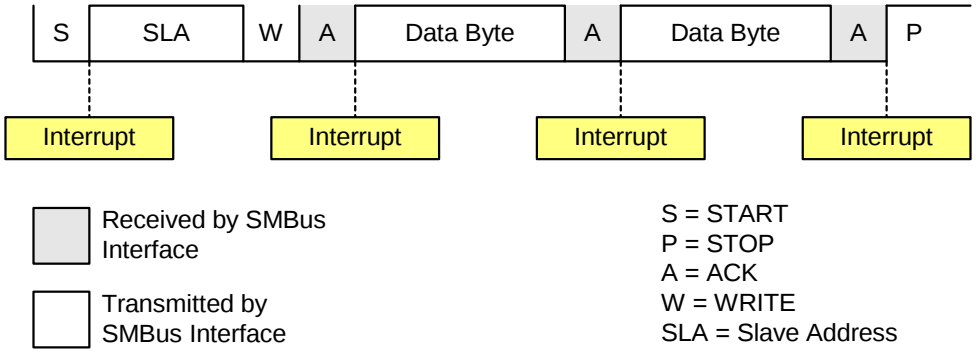


图 7.3: I^2C 总线的写入时序

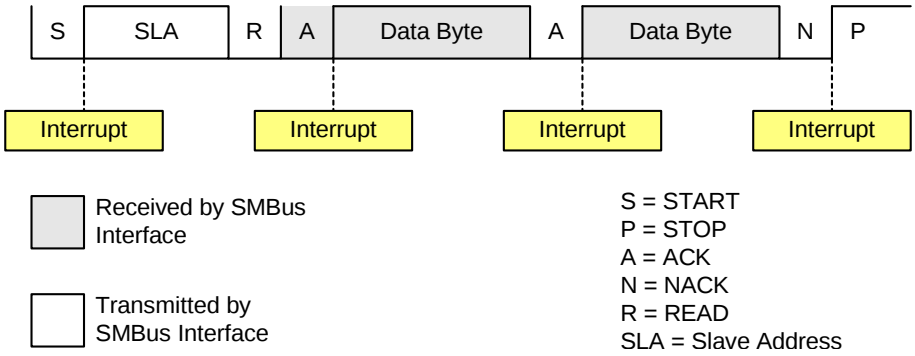


图 7.4: I^2C 总线的读出时序

表中 0 表示永远读出为 0，地址 0 中的 CH（Clock Halt）表示时钟停止，设置为 1 的时候有效，如果要使用计时的功能，需要将其清零。控制寄存器中的 OUT 用来控制 OUT 管脚的状态，0 表示低电平，1 表示高电平。SQWE 如果设置为 1 表示 OUT 管脚的输出是一个方波，其频率由 RS1-RS0 控制：00h 表示 1Hz；01h 表示 4.096kHz；02h 表示 8.192kHz；03h 表示 32.768kHz。

DS1307 的时间数据采用 BCD 格式，当地址 2 中的第 D6 位为高时，表示时钟采用 12 小时模式，此

表 7.3: DS1307 寄存器定义

地址	D7	D6	D5	D4	D3	D2	D1	D0	功能	取值范围
00h	CH	秒十位			秒个位				秒	00-59
01h	0	分十位			分个位				分	00-59
02h	0	12/24	时十位/上下午		时个位				时	1-12/00-23
03h	0	0	0	0	0	星期			星期	01-07
04h	0	0	日十位		日个位				日	01-31
05h	0	0	0	月十位	月个位				月	01-12
06h	年十位				年个位				年	00-99
07h	OUT	0	0	SQWE	0	0	RS1	RS0	控制寄存器	-
08-3Fh									RAM	00h-FFh

时 D5 位 1 表示下午，0 表示上午。而当 D6 为低时，时钟采用 24 小时模式，D5-D4 联合用来表示小时数的十位值。

当采用 I^2C 协议操作 DS1307 时，写入数据的过程为在起始位后先写入设备地址 1101000 加上 0 表示写入，然后 DS1307 会应答 ACK，主设备接下来写入要操作的寄存器地址，最后在写入 0 个或多个数据到 DS1307 中去。每写入一个数据，目标地址会自动加 1，主设备用停止位结束写操作。读入数据的过程类似，只不过在读取数据前可能需要通过写操作来写入一个操作地址，然后读操作才可以从这个地址开始读取数据。

7.3 实验内容

7.3.1 显示时钟内容

SMB 的 SDA 和 SCL 与 SPI 的部分管脚共用，因此要检查跳线是否跳接到了正确的位置（连接偏下方的两个接线柱），之后在开始后续的实验内容。

初始化 SMB 的代码如下所示：

```
void PORT_Init()
{
    XBR0 = 0x05;      // 允许 I2C 和 串口设备
    XBR2 = 0x40;      // 允许 XBR
    P0MDOUT |= 0x01; // TX 管脚推挽输出
}

void I2C_Init()
{
    SMB0CN = 0x07;   // AA=1, 允许超时检测
    SMB0CR = 257 - (SYSCLK / (2 * I2CCLK)); // 设置时钟频率 80K
    SMB0CN |= 0x40;  // 允许 SMB
    STO = 0;
}
```

处理各种 I^2C 状态的中断处理程序示例如下，具体的状态码宏定义可以参考表 7.2。

```
void SMBUS_ISR (void) interrupt 7
{
    bit FAIL = 0;             // 记录异常情况
    static unsigned char i;  // 缓存指针记录

    switch (SMB0STA)          // 状态寄存器
    {
        case SMB_START:       // 主设备起始位已发送
```

```
        case SMB_RP_START:   // 主设备重复起始位已发送
            SMB0DAT = smb_buf[0];  // 地址+读写位
            STA = 0;               // 清除起始位标志
            i = 0;                 // 指针清零
            break;
        case SMB_MTADDACK:         // 主设备发送，从设备确认地址
            SMB0DAT = smb_buf[1];  // 发送后续数据
            break;
        case SMB_MTDBACK:          // 主设备发送，从设备确认数据
            if (i < smb_len)       // 是否有更多数据?
            {
                SMB0DAT = smb_buf[2 + i]; // 传递下一个字节
                i++;               // 指针加 1
            } else {
                result = S_OVER;   // 结束状态
                STO = 1;           // 设置停止位标志
            }
            break;
        case SMB_MRADDACK:     // 主设备接收，从设备确认地址
            if (smb_len == 1)  // 只要接收一个数据
                AA = 0;        // 接下来将回复 NACK
            else
                AA = 1;        // 接下来将回复 ACK
            break;
        case SMB_MRDBACK:      // 主设备接收，已发送 ACK
            if ( i < smb_len)  // 如果还有更多数据
            {
                smb_buf[i + 1] = SMB0DAT;  // 保存已接到数据
                i++;                       // 指针加 1
                AA = 1;                    // 准备发送 ACK
            }
            if (i >= smb_len)  // 当前数据是最后字节
                AA = 0;        // 准备发送 NACK

            break;

        case SMB_MRDBNACK:     // 主设备接受，NACK 已发送
            smb_buf[i + 1] = SMB0DAT;      // 保存已接到数据
            STO = 1;           // 准备发送结束位
            AA = 1;            // AA 状态复位
            result = R_OVER;   // 结束状态
            break;
        case SMB_MTADDNACK:    // 主设备写，从设备未确认地址
        case SMB_MTDBNACK:     // 从设备未确认数据
        case SMB_MTARBLOST:    // 主设备竞争总线失败
            // Master Receiver: Slave address + READ transmitted.  NACK received.
        case SMB_MRADDNACK:    // 主设备读，从设备未确认地址
            FAIL = 1;          // 设置失败标志
            break;
        default:
            FAIL = 1;

            break;
    }
    if (FAIL)                   // 传输失败
    {
        SMB0CN &= ~0x40;        // 复位 SMB 设备
        SMB0CN |= 0x40;
```

```
        STA = 0;
        STO = 0;
        AA = 0;
        result = SMB_FAIL;      // 结束状态
        FAIL = 0;
    }
    SI = 0;                     // 清除中断标志
}
```

从 SMB 总线读取和发送的子函数如下所示：

```
void SMB_Transmit(unsigned char addr, unsigned char len)
{
    result = 0;         // 清除结束状态
    smb_buf[0] = 0xD0;  // 从设备地址 + 写入标志
    smb_buf[1] = addr;  // DS1307 寄存器地址
    smb_len = len;      // 写入数据长度
    STO = 0;            // 准备发送起始位
    STA = 1;
    while (result == 0);// 等待发送结束
    Delay(100);
}
void SMB_Receive(unsigned char len)
{
    result = 0;         // 清除结束状态
    smb_buf[0] = 0xD1;  // 从设备地址 + 读出标志
    smb_len = len;      // 读取的长度
    STO = 0;            // 准备发送起始位
    STA = 1;
    while (result == 0);// 等待发送结束
    Delay(100);
}
```

把时间数据输出到串口的示例程序如下：

```
void main()
{
    int i;
    WDTCN=0xDE;
    WDTCN=0xAD;
    SYSCLK_Init();
    PORT_Init();
    UART0_Init();
    TI0 = 1;
    I2C_Init();
    SI = 0;                 // 清除中断标志
    EIE1 |= 0x02;           // 允许 SMB 的中断
    EA = 1;                 // 允许全局中断

    SMB_Transmit(0, 0);     // 写入 DS1307 操作地址
    SMB_Receive(1);         // 读取第一个字节
    if (smb_buf[1] & 0x80) {    // 如果 CH 为 1
        unsigned char b = smb_buf[1];
        smb_buf[2] = b & 0x7F;  // 设置要写入的数据
        SMB_Transmit(0, 1);     // CH 置 0
    }
    while(1) {
        SMB_Transmit(0, 0);
        SMB_Receive(8);         // 接收时间数据
```

```
        Timer0_ms(500);
        for (i = 0; i < 8; ++i) { // 显示到串口
            printf("%02bx ", smb_buf[i + 1]);
        }
        printf("\r\n");
    }
}
```

请根据示例代码，实现在串口不断输出时间信息的功能，调整显示格式，显示内容为“年-月-日时: 分: 秒”的形式。

7.3.2 设置时钟

设计一个操作逻辑，可以在串口设置时钟的值。

7.4 思考题

1. 根据参考的中断服务程序，绘制 SMB 设备的工作流程图

实验八　字符型背光液晶显示模块

8.1　实验目的

1. 了解字符型液晶显示模块的使用方法
2. 掌握用并行端口模拟接口时序的方法

8.2　实验原理

8.2.1　1602 的硬件接口和时序

液晶显示器以其功耗小，显示内容丰富，体积小巧等诸多优点，在单片机系统中应用很广。我们的开发板包含一个字符型带背光的液晶模块，由于可显示的内容为 2 行 16 个字符，经常被称为 1602 模块。本章将介绍 1602 模块在我们开发板中的使用方法。

1602 模块有 16 个管脚，各个管脚的功能如下：

第 1 脚 VSS 为接地

第 2 脚 VDD 接 5V 电源

第 3 脚 V0 是液晶的对比度调整端，实验板上接了一个 10K 的电位器，可以通过这个电位器来调整显示对比度。如果对比度不合适，可能看不到显示内容或者显示为黑色方框。

第 4 脚 RS 为寄存器选择，高电平时选择数据寄存器，低电平时选择指令寄存器。该引脚连接到了 F020 的 P6.2。

第 5 脚 RW 为读写信号线，高电平时进行读操作，低电平时进行写操作。当 RS 和 RW 共同为低电平时可以写入指令或者显示地址，当 RS 为低电平 RW 为高电平时可以读忙信号，当 RS 为高电平 RW 为低电平时可以写入数据。该引脚连接到了 F020 的 P6.1。

第 6 脚 E 为使能端，当 E 端由高电平跳变成低电平时，液晶模块执行命令。该引脚连接到了 F020 的 P6.0。

第 7-14 脚 D0~D7 为 8 位双向数据线，它们被接到了 F020 的 P5 端口上（参考附录中的电路图）。

第 15,16 脚 空脚

1602 的读写时序如图 8.1和 8.2所示：

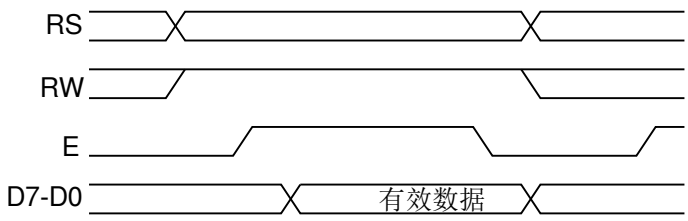


图 8.1: 1602 模块的读数据时序

1602 液晶模块通过总线接口与控制设备通信。它一共支持 12 个指令，通过不同的管脚的电平状态来区别，具体的定义如表 8.1 所示。

表 8.1 中，各个指令的含义详细解释如下：

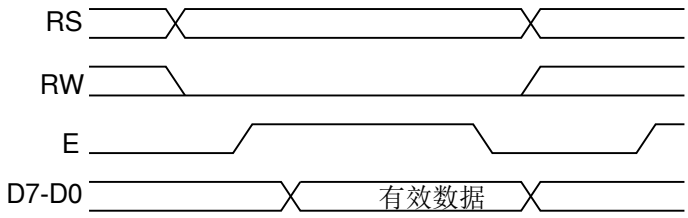


图 8.2: 1602 模块的写数据时序

表 8.1: 1602 液晶模块指令表

编号	指令	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0
1	清显示	0	0	0	0	0	0	0	0	0	1
2	光标返回	0	0	0	0	0	0	0	0	1	*
3	置输入模式	0	0	0	0	0	0	0	1	I/D	S
4	显示开关控制	0	0	0	0	0	0	1	D	C	B
5	光标或字符移位	0	0	0	0	0	1	S/C	R/L	*	*
6	置功能模式	0	0	0	0	1	DL	N	F	*	*
7	置字符发生存储器地址	0	0	0	1	字符发生存储器地址					
8	置数据存储器地址	0	0	1	显示数据存储器地址						
9	读忙标志或地址	0	1	BF	地址计数器						
10	写数据到存储器	1	0	要写的数据							
11	从存储器读数据	1	1	读出的数据							

1. 用来清空液晶模块的显示内容，同时光标也返回 00H 地址。
2. 光标返回到地址 00H，不改变显示内容。
3. I/D 位设置高电平表示光标的移动方向为右移，置低电平表示光标的移动方向为左移；S 位设置屏幕上的所有文字是否左移或者右移，高电平有效。
4. D 位控制整体显示的开与关，高电平表示开显示，低电平表示关显示；C 位控制光标的开与关，高电平表示有光标，低电平表示无光标；B 位控制光标是否闪烁，高电平闪烁，低电平不闪烁。
5. S/C 位高电平时移动显示的文字，低电平时移动光标；R/L 位控制移动的方向，高电平右移，低电平左移。
6. DL 位高电平时为 8 位总线（D0-D7 都有效），低电平时为 4 位总线（仅 D0-D3 有效）；N 位低电平时为单行显示，高电平时双行显示；F 位低电平时显示 5x7 的点阵字符；高电平时显示 5x10 的点阵字符 (有些模块是 DL 高电平时为 4 位总线，低电平时为 8 位总线)。
7. 置字符发生存储器地址，即当前的写入位置。
8. 置数据存储器地址。
9. 读忙标志或地址，BF 位高电平代表设备忙，模块此时不能接收命令或者数据，只有 BF 为低电平的时候才可以操作模块。
10. 写数据
11. 读数据

1602 液晶模块在使用之前需要进行初始化，初始化的过程就是根据表 8.1 的内容，对 1602 模块的工作方式进行设置。一种可能的设置方式是依次写入命令：0x38、0x08、0x01、0x06、0x0C、0x80、0x02，其含义分别如下：

- 0x38 为 6 号命令，表示 8 位地址总线、双行显示、5x7 点阵
- 0x08 为 4 号命令，表示关闭显示
- 0x01 为 1 号命令，表示清屏
- 0x06 为 3 号命令，表示右移光标，文字不平移

- 0x0C 为 4 号命令，表示打开显示，不显示光标
- 0x80 为 8 号命令，表示设置数据存储器的地址
- 0x02 为 2 号命令，光标位置复位

1602 液晶模块内部的字符发生存储器（CGROM）已经存储了 192 个不同的 5×7 点阵字符图形，每个字符对应一个编码。而 1602 的显示部分对应于其内部的一个存储器，每个显示位置对应一个显示地址。其中第一行字符对应的显示地址是 0x00～0x0F，第二行对应的显示地址是 0x40～0x4F，当把 CGROM 的编码写入到特定的地址，就可以在对应的位置显示出编码所对应的字符。对于常见的英文字母和符号，它们的编码和标准的 ASCII 编码一致。例如要在第二行第三列的位置显示一个字符 “a”，就需要先用 8 号命令写入地址 “0x42”，具体的命令字为 “0xC2”，然后再写入字符 “a”，写入数据时要保证 RS 引脚为高。

8.2.2 参考代码

根据时序图，通过 F020 处理器读取 1602 模块的指令寄存器参考代码如下，这个函数用来读取忙标志，进而判断 1602 模块是否准备好。其中 P6.7 为 SPI 总线的片选，这里置 1 使之无效。

```
char isLcdBusy(void)
{
    P5 = 0xFF;            // 置 P5 为输入模式
    P6 = 0x82;            // RS=0, RW=1, EN=0
    Delay(DELAY_LCD);     // 延时
    P6 = 0x83;            // RS=0, RW=1, EN=1
    Delay(DELAY_LCD);
    return (P5 & 0x80);   // 返回忙状态
}
```

通过 F020 处理器为 1602 模块写入控制命令的参考代码如下：

```
void Lcd1602_Write_Command(unsigned char Command)
{
    while(isLcdBusy());
    P5 = Command;         // 要写入的命令
    P6 = 0x80;            // RS=0, RW=0, EN=0
    Delay(DELAY_LCD);     // 延时
    P6 = 0x81;            // RS=0, RW=0, EN=1
    Delay(DELAY_LCD);
    P6 = 0x80;            // RS=0, RW=0, EN=0
}
```

如果要在模块的指定坐标位置写入指定的字符，可以使用下面的代码：

```
void Lcd1602_Write_Data(Data)
{
    P5 = Data;            // 要写入的数据
    P6 = 0x84;            // RS=1, RW=0, EN=0
    Delay(DELAY_LCD);
    P6 = 0x85;            // RS=1, RW=0, EN=1
    Delay(DELAY_LCD);
    P6 = 0x84;            // RS=1, RW=0, EN=0
}
void Lcd1602_WriteXY_Data(uchar row, uchar column, uchar Data)
{
    while(isLcdBusy());
    if (row == 1)
        column |= 0xC0;   // D7=1, 偏移地址为 0x40
    else
        column |= 0x80;   // D7=1
```

```
    Lcd1602_Write_Command(column); // 设置地址
    Lcd1602_Write_Data(Data);      // 写入数据
}
```

8.3 实验内容

8.3.1 CGRAM 内容展示

1602 显示模块的 CGRAM 一共保存了 192 个字符，但实际的 8 位字节可以表示的字符有 256 个，编程将全部字模分次显示在 LCD 上，查看具体的显示符号都是什么。参考程序如下所示：

```
void main(void)
{
    uchar i, j;
    WDTCN = 0xde;
    WDTCN = 0xad;   // 禁止看门狗电路
    SYSCLK_Init();
    P74OUT = 0x30;  // P6 推挽输出
    Lcd1602_init();
    for (i = 0; i < 8; i++) {
        Lcd1602_Write_Command(0x80);  // 显示第一行
        for (j = 0; j < 16; j++)
            Lcd1602_Write_Data(i * 32 + j); // 数据
        Lcd1602_Write_Command(0xC0);  // 显示第二行
        for (j = 0; j < 16; j++)
            Lcd1602_Write_Data(i * 32 + j + 16);
        for (j = 0; j < 255; j++)
            Delay(10000);             // 延时
    }
    while(1);       // 停机
}
```

运行测试程序可以发现，保存在 CGRAM 中的内容只有 0x20~0x7F 和 0xA0~0xFF 范围内有定义。而在 0x00~0x0F 的范围是用户自定义字符区，我们下面将利用这个功能显示一些特殊的字符。

8.3.2 显示自定义字符

设置 1602 的自定义字符需要用到它的 7 号命令，这个命令允许用户在 CGRAM 中写入新的字模数据。注意到 7 号命令可以使用的地址位数只有 6 位，因此实际可以操作的地址只有 64 个字节，而每个字模都需要 8 个字节来保存（每个字节代表一行显示内容的点阵位图，参考 7 段数码管的段码定义），因此一共可以定义的自定义字符只有 8 个。

下面的代码定义了一个新的字符，并把它显示到 LCD 上：

```
uchar  code bitmap[]={0x1F,0x11,0x11,0x1F,0x11,0x11,0x1F,0x00};  // 汉字"日"

void main(void)
{
    ......
    Lcd1602_Write_Command(0x40);            // 7号命令, 设置地址为0, 第一个字符
    for (i = 0; i < 8; i++) Lcd1602_Write_Data(bitmap[i]);  // 写入位图
    Lcd1602_Write_Command(0x80);            // 显示地址
    Lcd1602_Write_Data（0x00);              // 显示第一个自定义字符
    ......
}
```

利用 7 号命令设计一个新字符，并将其显示在 LCD 上。

8.3.3 简易计算器

利用 4×4 键盘和 1602 显示模块，实现一个简单的电子计算器，可以通过键盘输入算式，并将算式和计算结果同时显示在模块上。

实验九　综合实验

9.1　实验条件

- 基于 C8051F020 实验箱

9.2　要求

- 注意选题的实用性及综合性（相对于此前所做过的所有基本单元实验而言，尽可能使选题内容涵盖或超越单元实验中所含有的知识点）。
- 自由命题、自主发挥、力求新创意。
- 鼓励独立完成，限时三周，并于综合实验第一周上课时报告所选题目。
- 在课题实现过程中，建议采取逐步扩展功能或增加难度的策略，不求一步到位。
- 第三周验收时，可采取同学间相互演示、问答方式进行。
- 写出综合实验报告。

9.3　参考题目

- 数字录音机：可记录并回放一定时长（不少于 12 秒）的声音，声音数据可以保存到 Flash 中。
- 串行通信：通过串口实现 XMODEM 协议，可以把 Flash 中的内容传递给 PC 机，或者将 PC 机传递的内容写入到 Flash 中去。
- 电子琴：通过按键播放相应频率的声音，音长可控，或使电子琴具有较为复杂的音效。
- 电梯控制：用按键模拟电梯每层的按钮，显示电梯的呼叫状态和电梯位置等信息。控制电梯的安全、可靠、高效运行。
- 交通灯控制：显示道路通行状态和交通灯的状态。
- 开发板功能测试：设计测试流程，通过按键选择不同的功能、显示模块显示测试结果，对开发板各个功能进行测试
- 功能比较完善的电子表：具有日历和时钟功能，可以方便的通过按键进行时间设置和功能选择。

综合实验验收测评书

注：此表于综合实验完成时，在提交老师进行验收前由实验完成人填写，所填写的内容作为综合实验成绩评定之参考。凡在自我评估项目中选择 A、B 档的同学在验收时应准备向老师陈述理由，选择 C 档的同学准备老师抽查。

1. 综合实验题目：_____

2. 课题内容提要：_____

_____

3. 验收功能简述（按验收先后顺序逐项书写，验收时教师签字确认）：

_____

_____

_____

4. 自我评估及诚信声明（实验完成人在评估栏目选项中打勾，并签字确认所做声明）

本人郑重声明：以下内容均经过本人认真思考后作出的回答，本人承诺对所有选项的真实有效性负责。

实验完成人_____（签名）学号_____

自我评估项目	A（要向老师说明或举例证明）	B（要向老师说明或举例证明）	C（老师将抽查）	D
知识产权归属涉及内容：整体创意、程序的主体框架、所有主要功能单元模块的形成	100% 属于自己	＞ 80% 属于自己，另 20% 的归属于：	＞ 50% 属于自己，另 50% 归属于：	＜ 50% 属于自己，其余归属于：
调试过程中独立自主完成的程度	100% 属于自己	＞ 80% 属于自己，另 20% 的归属于：	＞ 50% 属于自己，另 50% 归属于：	＜ 50% 属于自己，其余归属于：
项目功能的完善程度	很出色	比较好	一般	功能单一
对难点问题的有效解决	有令自己得意的发挥之处	想办法巧妙绕过了难点问题	对疑难之处放弃了原有功能	对此没感觉
关于同学之间的探讨	提出过有建设性的好主意被采纳	提出过好主意但未被别人采纳	受人启发，并改进了别人的设想	完全借鉴了别人的建议
未完成部分原因				
对不足方面的分析				
收获				
对进一步研发的展望				

附录 A　MCS-51 系列单片机指令系统

A.1　指令列表

- “字节数”为指令代码的长度,“周期数”为指令执行所需的机器周期数。
- “C”、“OV”、“AC”栏内容表示指令执行后标志“C”、“OV”、“AC”的变化状况,“X”表示该标志是置“0”还是置“1”,取决于指令执行结果。
- 间接寻址操作数表示“@Ri”中,i 只能是 0 或 1。

表 A.1: 制程序转移类指令(17 条)

助记符	操作数 1	操作数 2	操作数 3	功能	字节数	周期数	C
ACALL	addr 11			2KB 内绝对调用	2	2	
AJMP	addr 11			2KB 内绝对转移	2	2	
CJNE	@Ri	#data	rel	间接 RAM 与立即数不等转移	3	2	X
CJNE	A	#data	rel	累加器与立即数不等转移	3	2	X
CJNE	A	direct	rel	累加器与直接寻址字节不等转移	3	2	X
CJNE	Rn	#data	rel	寄存器与立即数不等转移	3	2	X
DJNZ	direct	rel		直接寻址字节减 1 不为零转移	3	2	
DJNZ	Rn	rel		直接寻址字节减 1 不为零转移	2	2	
JMP	@A+DPTR			相对 DPTR 的转移	1	2	
JNZ	rel			累加器不为零转移	2	2	
JZ	rel			累加器为零转移	2	2	
LCALL	addr 16			64KB 内长调用	3	2	
LJMP	addr 16			64KB 内长转移	3	2	
NOP				空操作	1	1	
RET				从子程序返回	1	2	
RETI				从中断返回	1	2	
SJMP	rel			短转移	2	2	

表 A.2: 算术运算类指令(24 条)

助记符	操作数 1	操作数 2	功能	字节数	周期数	C	OV	AC
ADD	A	@Ri	间接 RAM 加到累加器	1	1	X	X	X
ADD	A	Rn	寄存器加到累加器	1	1	X	X	X
ADD	A	#data	立即数加到累加器	2	1	X	X	X
ADD	A	direct	直接寻址字节加到累加器	2	1	X	X	X
ADDC	A	@Ri	间接 RAM 带进位加到累加器	1	1	X	X	X
ADDC	A	Rn	寄存器带进位加到累加器	1	1	X	X	X

表 A.2: 算术运算类指令(24 条)续

助记符	操作数 1	操作数 2	功能	字节数	周期数	C	OV	AC
ADDC	A	#data	立即数带进位加到累加器	2	1	X	X	X
ADDC	A	direct	直接寻址字节带进位加到累加器	2	1	X	X	X
DA	A		十进制数调整	1	1	X		
DEC	@Ri		间接 RAM 减 1	1	1			
DEC	A		累加器减 1	1	1			
DEC	direct		直接寻址字节减 1	2	1			
DEC	Rn		寄存器减 1	1	1			
DIV	AB		A 除以 B,商数存 A,余数存 B	1	4	0	X	
INC	@Ri		间接 RAM 加 1	1	1			
INC	A		累加器加 1	1	1			
INC	direct		直接寻址字节加 1	2	1			
INC	DPTR		数据指针加 1	1	2			
INC	Rn		寄存器加 1	1	1			
MUL	AB		A 乘 B,乘积低 8 位存 A,高 8 位存 B	1	4	0	X	
SUBB	A	@Ri	累加器减间接 RAM 和进位	1	1	X	X	X
SUBB	A	Rn	累加器减寄存器和进位	1	1	X	X	X
SUBB	A	#data	累加器减立即数和进位	2	1	X	X	X
SUBB	A	direct	累加器减直接寻址字节和进位	2	1	X	X	X

表 A.3: 逻辑运算和移位类指令(25 条)

助记符	操作数 1	操作数 2	功能	字节数	周期数	C
ANL	A	@Ri	间接 RAM“与”到累加器	1	1	
ANL	A	Rn	寄存器“与”到累加器	1	1	
ANL	A	#data	立即数“与”到累加器	2	1	
ANL	A	direct	直接寻址字节“与”到累加器	2	1	
ANL	direct	A	累加器“与”到直接寻址字节	2	1	
ANL	direct	#data	立即数“与”到直接寻址字节	3	2	
CLR	A		累加器清零	1	1	
CPL	A		累加器取反	1	1	
ORL	A	@Ri	间接 RAM“或”到累加器	1	1	
ORL	A	Rn	寄存器“或”到累加器	1	1	
ORL	A	#data	立即数“或”到累加器	2	1	
ORL	A	direct	直接寻址字节“或”到累加器	2	1	
ORL	direct	A	累加器“或”到直接寻址字节	2	1	

表 A.3: 逻辑运算和移位类指令(25 条)续

助记符	操作数 1	操作数 2	功能	字节数	周期数	C
ORL	direct	#data	立即数“或”到直接寻址字节	3	2	
RL	A		累加器循环左移 1 位,$A_7 \rightarrow A_0$, $A_n \rightarrow A_{n+1}$	1	1	
RLC	A		累加器连进位循环左移 1 位,$C \rightarrow A_0$, $A_7 \rightarrow C$, $A_n \rightarrow A_{n+1}$	1	1	X
RR	A		累加器循环右移 1 位,$A_0 \rightarrow A_7$,$A_n \rightarrow A_{n-1}$	1	1	
RRC	A		累加器连进位循环右移 1 位,$C \rightarrow A_7$,$A_0 \rightarrow C$,$A_n \rightarrow A_{n-1}$	1	1	X
SWAP	A		累加器的高、低 4 位交换	1	1	
XRL	A	@Ri	间接 RAM“异或”到累加器	1	1	
XRL	A	Rn	寄存器“异或”到累加器	1	1	
XRL	A	#data	立即数“异或”到累加器	2	1	
XRL	A	direct	直接寻址字节“异或”到累加器	2	1	
XRL	direct	A	累加器“异或”到直接寻址字节	2	1	
XRL	direct	#data	立即数“异或”到直接寻址字节	3	2	

表 A.4: 数据传送类指令(28 条)

助记符	操作数 1	操作数 2	功能	字节数	周期数	C
MOV	@Ri	A	累加器送到间接 RAM	1	1	
MOV	@Ri	#data	立即数送到间接 RAM	2	1	
MOV	@Ri	direct	直接寻址字节送到间接 RAM	2	2	
MOV	A	@Ri	间接 RAM 送到累加器	1	1	
MOV	A	Rn	寄存器送到累加器	1	1	
MOV	A	#data	立即数送到累加器	2	1	
MOV	A	direct	直接寻址字节送到累加器	2	1	
MOV	direct	@Ri	间接 RAM 送到直接寻址字节	2	2	
MOV	direct	A	累加器送到直接寻址字节	2	1	
MOV	direct	Rn	寄存器送到直接寻址字节	2	2	
MOV	direct	#data	立即数送到直接寻址字节	3	2	
MOV	direct	direct	直接寻址字节送到直接寻址字节	3	2	
MOV	DPTR	#data16	16 位立即数送到 DPTR	3	2	
MOV	Rn	A	累加器送到寄存器	1	1	
MOV	Rn	#data	立即数送到寄存器	2	1	

表 A.4: 数据传送类指令(28 条)续

助记符	操作数 1	操作数 2	功能	字节数	周期数	C
MOV	Rn	direct	直接寻址字节送到寄存器	2	2	
MOVC	A	@A+DPTR	相对 DPTR 的程序代码送到累加器	1	2	
MOVC	A	@A+PC	相对 PC 的程序代码送到累加器	1	2	
MOVX	@DPTR	A	累加器送到外部 RAM	1	2	
MOVX	@Ri	A	累加器送到外部 RAM(256 字节地址空间)	1	2	
MOVX	A	@DPTR	外部 RAM 送到累加器	1	2	
MOVX	A	@Ri	外部 RAM(256 字节地址空间) 送到累加器	1	2	
POP	direct		出栈到直接寻址字节	2	2	
PUSH	direct		直接寻址字节压栈	2	2	
XCH	A	@Ri	累加器与间接 RAM 互换	1	1	
XCH	A	direct	累加器与直接寻址字节互换	1	1	
XCH	A	Rn	累加器与寄存器互换	1	1	
XCHD	A	@Ri	累加器与间接 RAM 低 4 位互换	1	1	

表 A.5: 位操作类指令(17 条)

助记符	操作数 1	操作数 2	功能	字节数	周期数	C
ANL	C	/bit	直接寻址位取反“与”到进位	2	2	X
ANL	C	bit	直接寻址位“与”到进位	2	2	X
CLR	bit		清直接寻址位	2	1	
CLR	C		清进位	1	1	0
CPL	bit		直接寻址位取反	2	1	
CPL	C		进位取反	1	1	X
JB	bit	rel	直接寻址位置位转移	3	2	
JBC	bit	rel	直接寻址位置位转移并清该位	3	2	
JC	rel		进位置位转移	2	2	
JNB	bit	rel	直接寻址位清零转移	3	2	
JNC	rel		进位清零转移	2	2	
MOV	bit	C	进位送到直接寻址位	2	1	
MOV	C	bit	直接寻址位送到进位	2	1	X
ORL	C	/bit	直接寻址位取反“或”到进位	2	2	X
ORL	C	bit	直接寻址位“或”到进位	2	2	X
SETB	bit		置直接寻址位	2	1	
SETB	C		置进位	1	1	1

附录 B　中断向量表

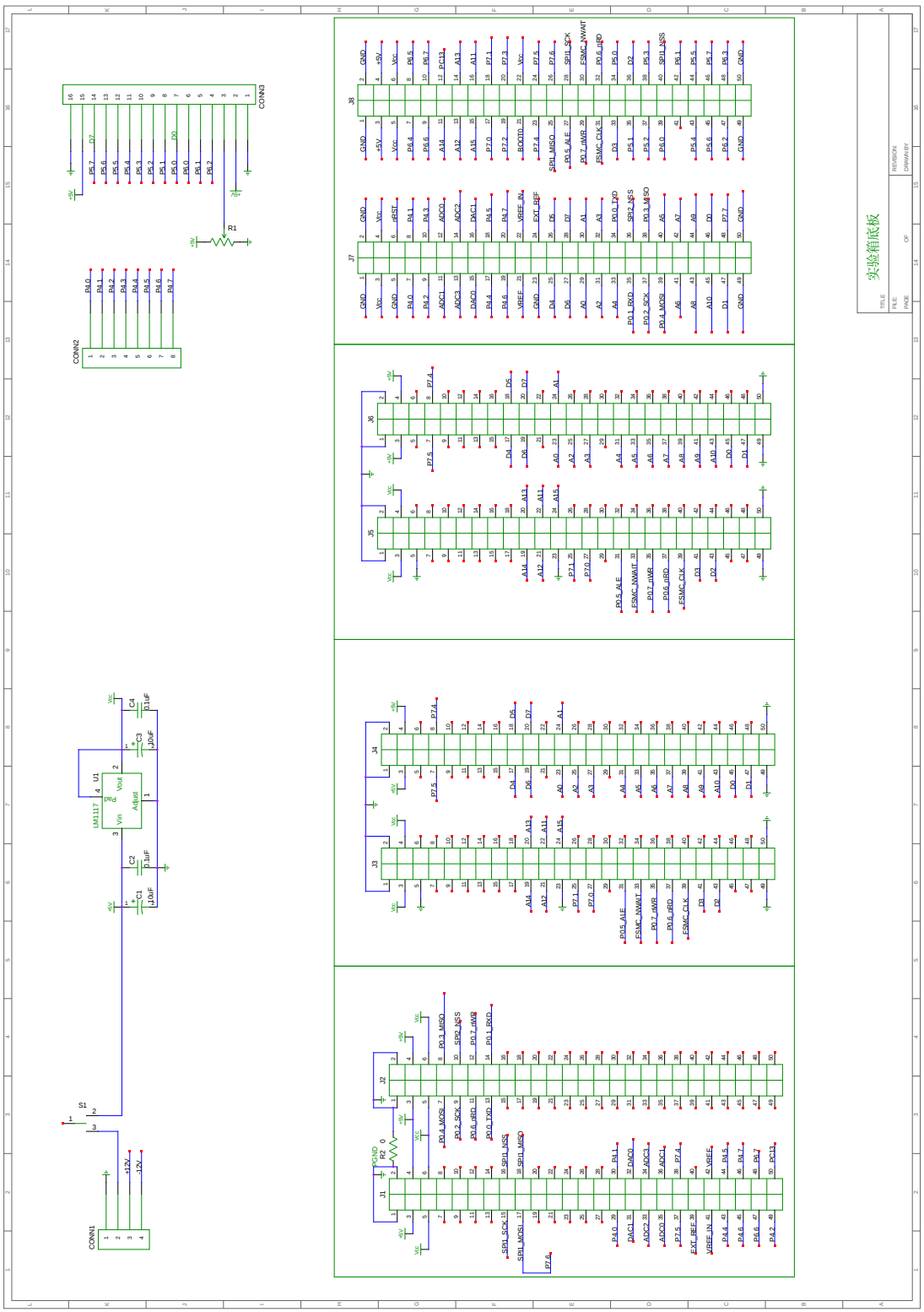
中断源	中断向量	优先级	中断标志	可位寻址	硬件复位	允许标志	优先级控制
Reset	0x0000	Top	None	N/A	N/A	Always Enabled	Always Highest
External Interrupt 0 (/INT0)	0x0003	0	IE0 (TCON.1)	Y	Y	EX0 (IE.0)	PX0 (IP.0)
Timer 0 Overflow	0x000B	1	TF0 (TCON.5)	Y	Y	ET0 (IE.1)	PT0 (IP.1)
External Interrupt 1 (/INT1)	0x0013	2	IE1 (TCON.3)	Y	Y	EX1 (IE.2)	PX1 (IP.2)
Timer 1 Overflow	0x001B	3	TF1 (TCON.7)	Y	Y	ET1 (IE.3)	PT1 (IP.3)
UART0	0x0023	4	RI0 (SCON0.0) TI0 (SCON0.1)	Y		ES0 (IE.4)	PS0 (IP.4)
Timer 2 Overflow (or EXF2)	0x002B	5	TF2 (T2CON.7)	Y		ET2 (IE.5)	PT2 (IP.5)
Serial Peripheral Interface	0x0033	6	SPIF (SPI0CN.7)	Y		ESPI0 (EIE1.0)	PSPI0 (EIP1.0)
SMBus Interface	0x003B	7	SI (SMB0CN.3)	Y		ESMB0 (EIE1.1)	PSMB0 (EIP1.1)
ADC0 Window Comparator	0x0043	8	AD0WINT (ADC0CN.2)	Y		EWADC0 (EIE1.2)	PWADC0 (EIP1.2)
Programmable Counter Array	0x004B	9	CF (PCA0CN.7) CCFn (PCA0CN.n)	Y		EPCA0 (EIE1.3)	PPCA0 (EIP1.3)
Comparator 0 Falling Edge	0x0053	10	CP0FIF (CPT0CN.4)			ECP0F (EIE1.4)	PCP0F (EIP1.4)
Comparator 0 Rising Edge	0x005B	11	CP0RIF (CPT0CN.5)			ECP0R (EIE1.5)	PCP0R (EIP1.5)
Comparator 1 Falling Edge	0x0063	12	CP1FIF (CPT1CN.4)			ECP1F (EIE1.6)	PCP1F (EIP1.6)
Comparator 1 Rising Edge	0x006B	13	CP1RIF (CPT1CN.5)			ECP1R (EIE1.7)	PCP1F (EIP1.7)
Timer 3 Overflow	0x0073	14	TF3 (TMR3CN.7)			ET3 (EIE2.0)	PT3 (EIP2.0)
ADC0 End of Conversion	0x007B	15	AD0INT (ADC0CN.5)	Y		EADC0 (EIE2.1)	PADC0 (EIP2.1)
Timer 4 Overflow	0x0083	16	TF4 (T4CON.7)			ET4 (EIE2.2)	PT4 (EIP2.2)
ADC1 End of Conversion	0x008B	17	AD1INT (ADC1CN.5)			EADC1 (EIE2.3)	PADC1 (EIP2.3)
External Interrupt 6	0x0093	18	IE6 (P3IF.5)			EX6 (EIE2.4)	PX6 (EIP2.4)
External Interrupt 7	0x009B	19	IE7 (P3IF.6)			EX7 (EIE2.5)	PX7 (EIP2.5)
UART1	0x00A3	20	RI1 (SCON1.0) TI1 (SCON1.1)			ES1	PS1
External Crystal OSC Ready	0x00AB	21	XTLVLD (OSCXCN.7)			EXVLD (EIE2.7)	PXVLD (EIP2.7)

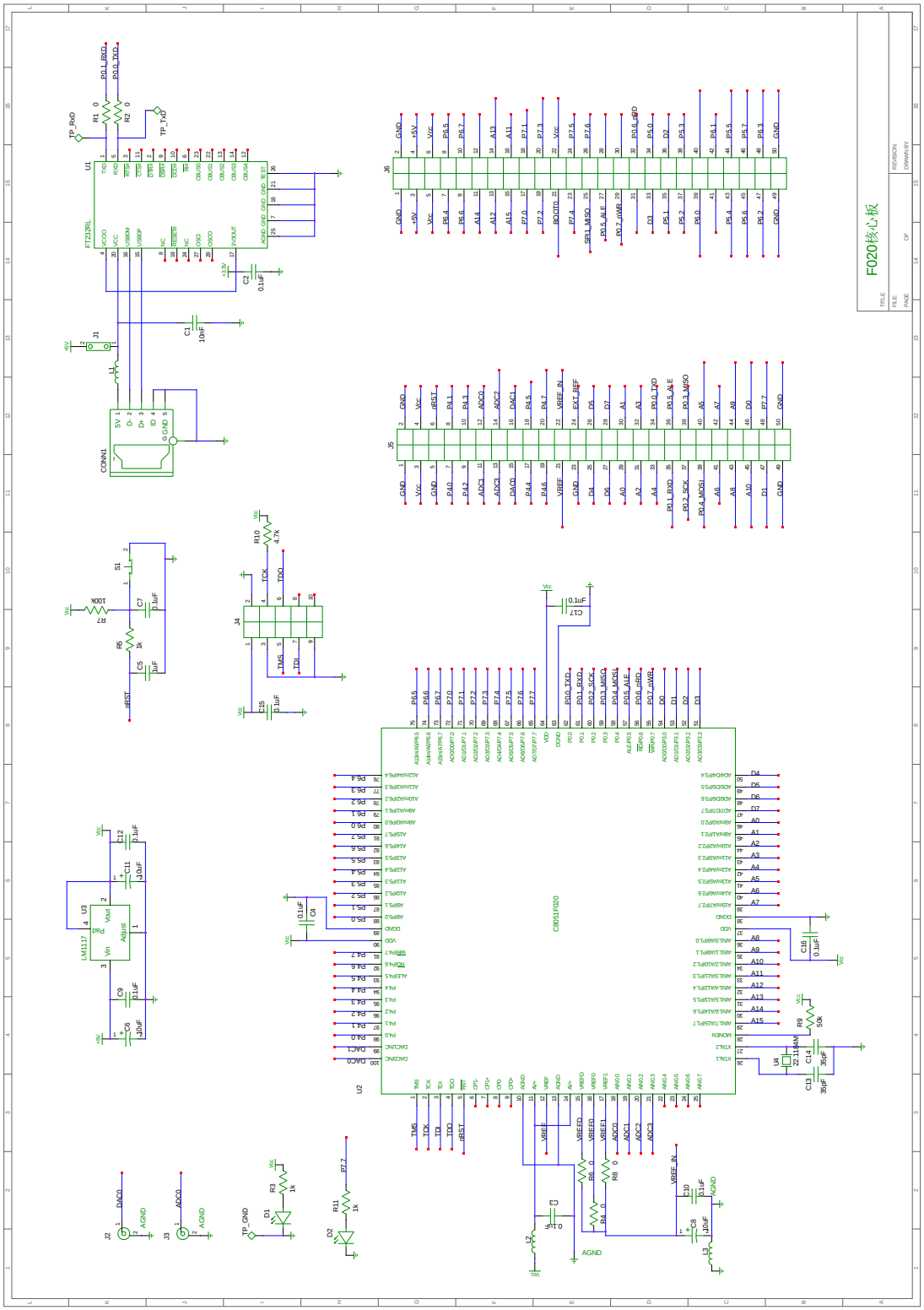
附录 C　参考书目

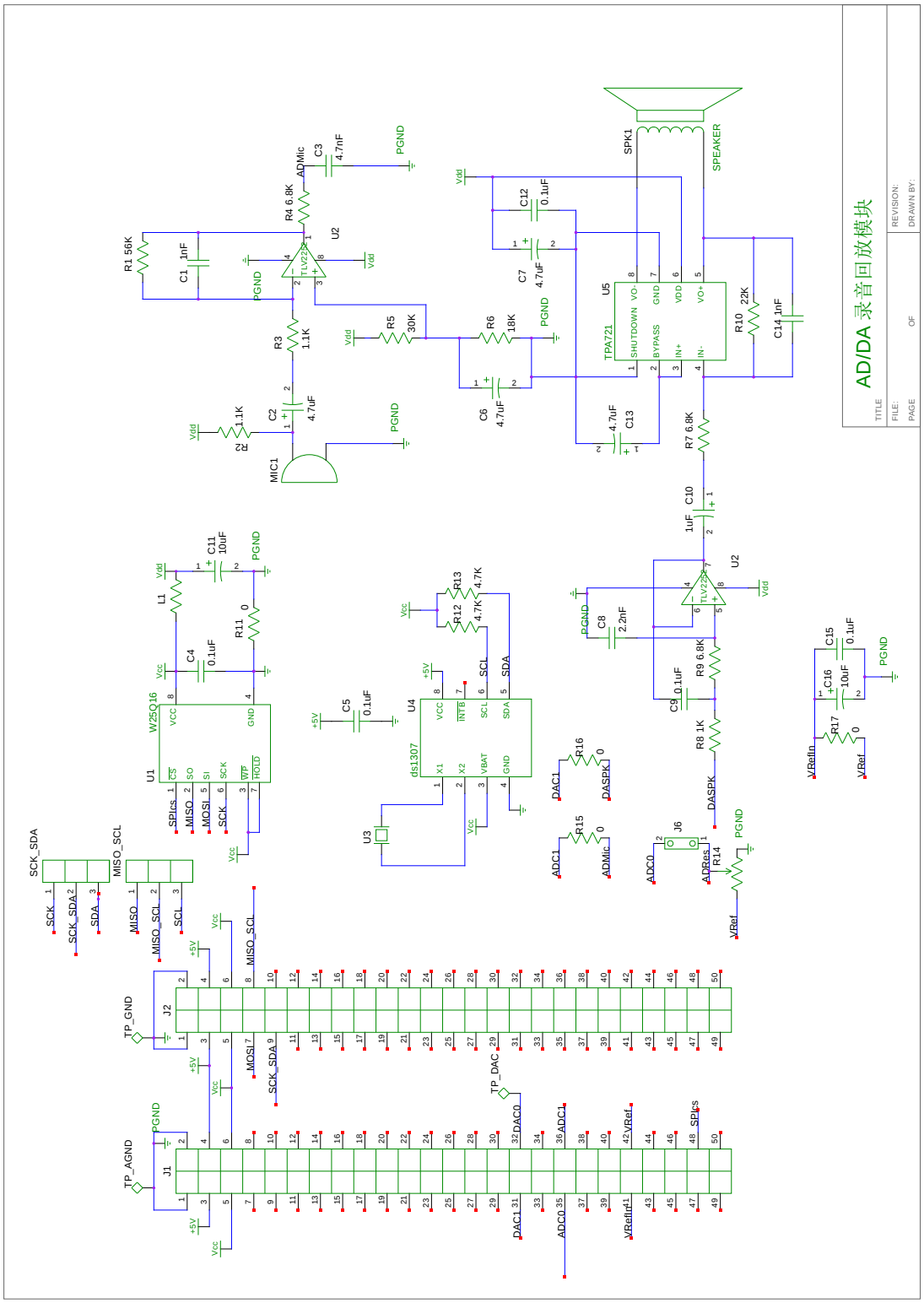
1. 胡汉才. 单片机原理及其接口技术(第 2 版). 清华大学出版社,2005
2. 孙涵芳, 徐爱卿. MCS-51/96 系列单片机原理及应用. 北京航空航天大学出版社,1998
3. 马忠梅等. 单片机的 C 语言应用程序设计(第 4 版). 北京航空航天大学出版社,2007
4. 王晓萍. 微机原理与接口技术. 浙江大学出版社,2014

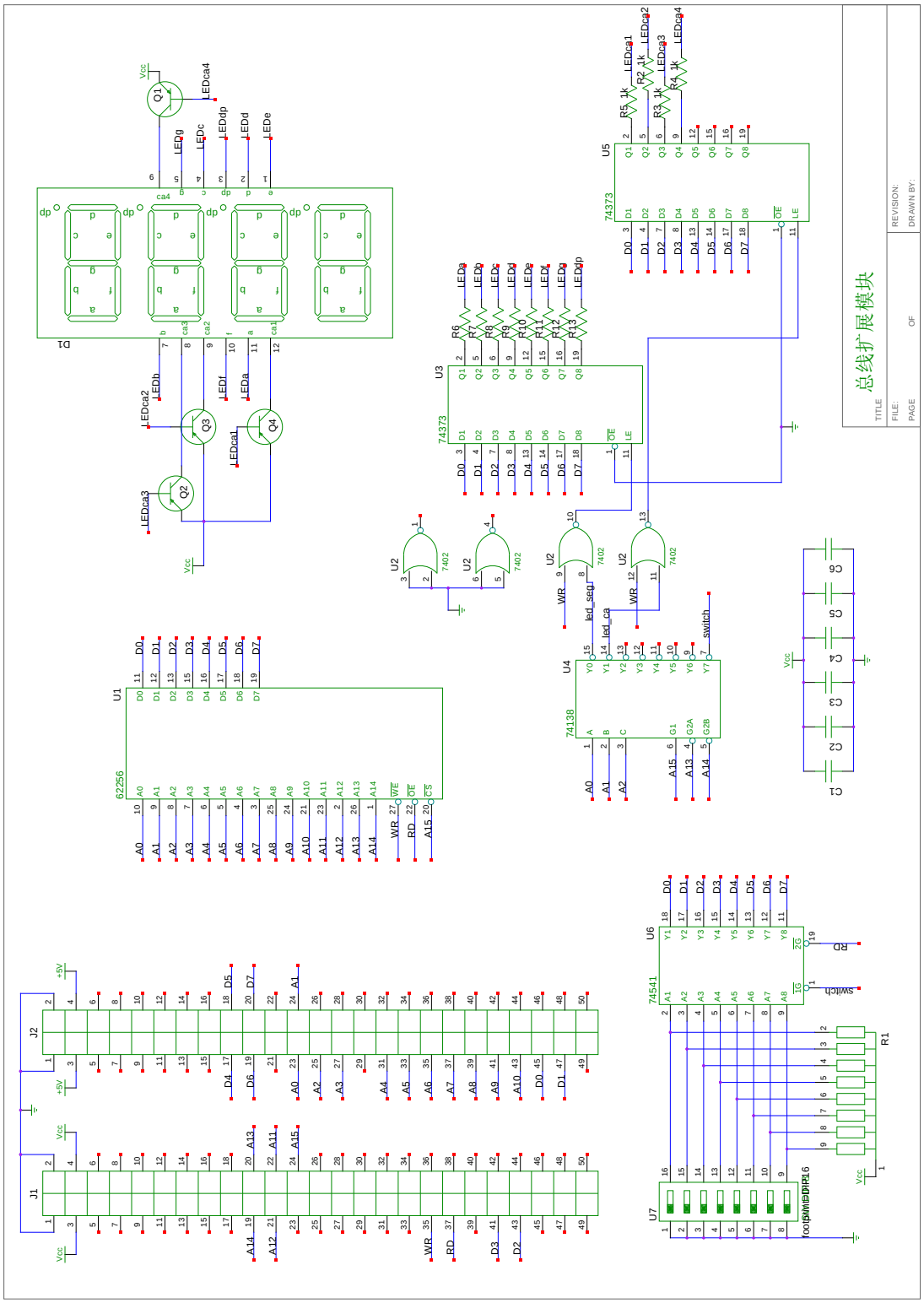
附录 D　电路图

1. 实验箱底板
2. F020 核心板
3. AD/DA 录音回放模块
4. 总线扩展板


实验箱底板
J1
J2
J3
J4
J5
J6
J7
J8
CONN1
CONN2
CONN3
U1
LM1117
R1
R2
C1
C2
C3
C4
S1


F020核心板
TITLE
FILE
PAGE
REVISION
DRAWN BY
CH
U1
FT232RL
U2
C8051F020
U3
LM1117
U4
22.1184M
J1
J2
J3
J4
J5
J6
CONN1
S1
D1
D2
P0.1_RXD
P0.0_TXD
TP_RXD
TP_TXD
P0.2_SCK
P0.3_MISO
P0.4_MOSI
P0.5_ALE
P0.6_nRD
P0.7_nWR
TCK
TDO
TMS
TDI
nRST
VREF_IN
ADC0
ADC1
ADC2
ADC3
DAC0
DAC1
TP_GND
AGND
+5V
+3V
GND
Vcc


AD/DA 录音回放模块
TITLE
FILE:
PAGE
OF
REVISION:
DRAWN BY:
SPK1
SPEAKER
TPA721
U5
SHUTDOWN
BYPASS
VO-
GND
VDD
VO+
IN+
IN-
TLV225
U2
R1 56K
C1 1nF
R2 1.1K
R3 1.1K
R4 6.8K
ADMic
C3 4.7nF
R5 30K
R6 18K
C2 4.7uF
C6 4.7uF
C7 4.7uF
C12 0.1uF
C13 4.7uF
R10 22K
C14 1nF
R7 6.8K
C10 1uF
C8 2.2nF
C9 0.1uF
R9 6.8K
R8 1K
DASPK
MIC1
PGND
Vdd
W25Q16
U1
CS
SO
SI
SCK
WP
HOLD
L1
C11 10uF
C4 0.1uF
R11 0
SPIcs
MISO
MOSI
SCK
Vcc
ds1307
U4
INTB
SCL
SDA
VBAT
X1
X2
C5 0.1uF
R12 4.7K
R13 4.7K
U3
R16 0
R15 0
DAC1
ADC1
ADC0
J6
ADRes
R14
VRef
C15 0.1uF
C16 10uF
R17 0
VRefin
SCK_SDA
MISO_SCL
SDA
SCL
TP_GND
TP_AGND
TP_DAC
J1
J2
+5V
DAC0
VRefin
SPIcs


总线扩展模块
TITLE:
FILE:
PAGE
OF
REVISION:
DRAWN BY:
D1
U1
62256
U2
7402
U3
74373
U4
74138
U5
74373
U6
74541
U7
R1
R2 1K
R3 1K
R4 1K
R5 1K
R6
R7
R8
R9
R10
R11
R12
R13
Q1
Q2
Q3
Q4
J1
J2
C1
C2
C3
C4
C5
C6
LEDca1
LEDca2
LEDca3
LEDca4
LEDa
LEDb
LEDc
LEDd
LEDe
LEDf
LEDg
LEDdp
led_seg
led_ca
switch
WR
RD
CS
OE
WE
Vcc
+5V