文轩科技

单片机那些事儿

中级篇——按键

残弈悟恩

2014

官方淘宝店铺：HTTP://SHOP109195762.TAOBAO.COM

郑重声明

本资料以残弈悟恩开发的 MGMC-V1.0 实验板为硬件平台，以残弈悟恩编写的《深入浅出玩转 51 单片机》为辅助教程，以残弈悟恩录制的《31 天环游单片机》为基础视频。

本资料以个人学习和工作的经验为素材，以单片机初学者、单片机项目开发者为对象。教大家如何走进单片机的世界，如何开发工程项目。限于时间和水平关系，资料中难免有过失之处，望各位高手批评指教，多多拍砖，拍累了，你们休息，我继续上路。

现已连载的方式免费共享于各大电子网站，供单片机新手们参考学习，可以自由下载传阅，但未经残弈悟恩允许，不得用于任何商业目的，若经发现，残弈悟恩将以愚公移山的精神追究到底。

版本：20140709（V1.0）　　制造者：残弈悟恩

让爱充满大地——花 1 秒时间，拯救 1 个人，传递 1 份爱

声明：只是残弈悟恩爱心的喷发，我得不到一分钱，各位不要多想，谢谢！

你知道吗？在非洲北边的某个地区，每一秒都有许许多多的人正在挨饿，每一天至少有一位儿童死于营养不足。你的一次点击就能让某位穷人得到 1.1 杯食物。当然你可以不相信有这样的链接或者是骗点击什么的。事实上，网站确实是帮穷人得 1.1 杯食物的，只要你点进去单机一下中间的黄色按钮，就会出来一系列介绍各种商品的网页（绝对免费的并且不会下载任何软件，也不会有电脑病毒），同时也会有人因为您的一次点击而得到 1.1 杯食物，食物是由商家提拱的，但爱心却是您献出的。如果你觉得残弈悟恩在忽悠大家，你不妨可以在网上查一下是真与假。

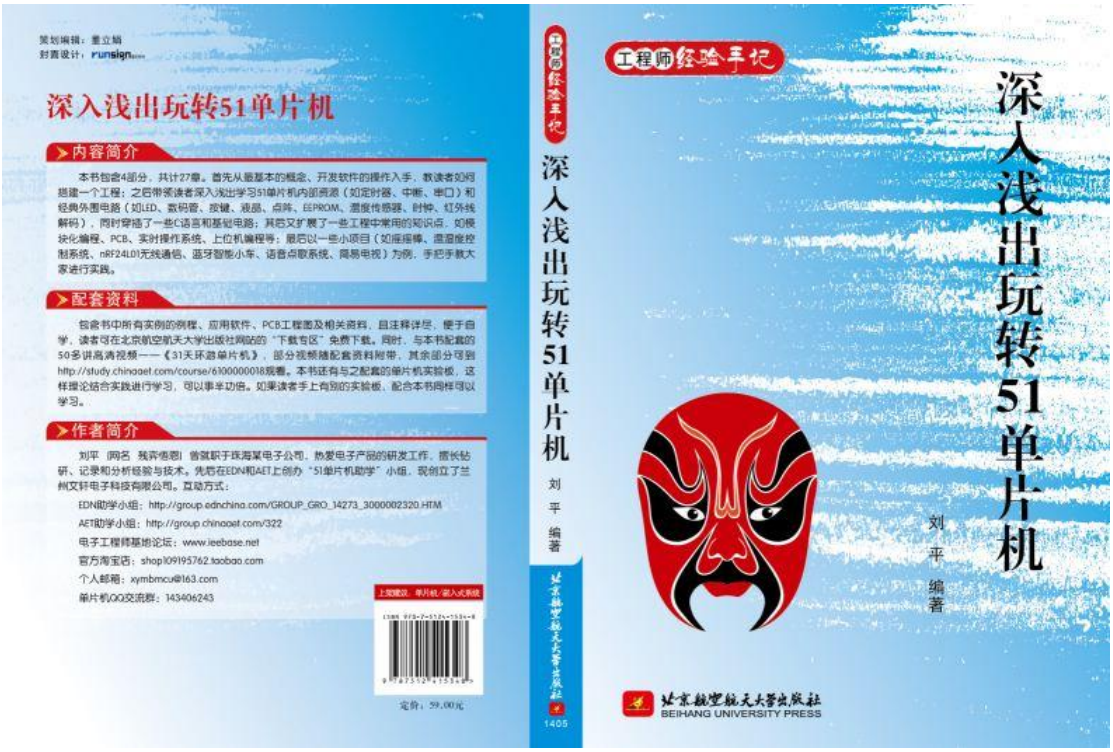
看到这本资料的朋友多数都是电子爱好者、单片机初学者，或者干电子这一行的，管你穷学生还是穷工人，只要能上网，只要愿花一秒种就可以了。人生在世，有两件事不能等：一、孝顺；二、行善。无论你是 LED 小灯、普通灯泡也好，还是荧光灯也吧，最重要就是要懂得用自身的光去照耀别人，光的强度并不重要。

点击链接：

http://www.thehungersite.com/clickToGive/home.faces?siteId=1&link=ctg_ths_home_from_ths_thankyou_sitenav

特别说明

该资料是《深入浅出玩转 51 单片机》一书的“裁剪版、凌乱版”，完整版、整齐版请见由北京航天航空大学出版社于 2014 年 5 月权威出版的书籍——《深入浅出玩转 51 单片机》，这里便于大家直观了解，先贴封面一张，购买地址就不贴了，免得大家说我有什么目的，到当当、卓越、京东、淘宝等网站随便一搜，就能找到此书的购买链接。



第七章　按键

残弈悟恩在给大家讲述该章之前，先告诉大家，这章是用来“玩”的，用来为“小儿科”做的一个铺垫，真正工程、项目中没有这样“弱智”的消抖（用 Delay()函数），那我为何也用 Delay，我说了，是玩的，不是学的，既然这样，大家就先玩玩呗。

等到读者将这章玩好了，下章（定时器）掌握了，那时才是我们学习按键的黄金时间。不见不散，不要错过哦。

读者在学习这章时，需要思考两个问题？

- 为了消抖，按键按下之后加了 10ms 的延时，那要是外界在按下之后的 5ms 会给单片机 1000 万+N 个美女，持续时间只有 1ms，那这不是白白将打好的计划浪费掉了，多可惜，怎么办了？
- 为了做松手检测，加了一句 while(!KEY1)，那要是遇到一个老太婆，一把按下，再没松手，那单片机所运行的程序岂不是也“死”在这条 while 语句里面了，这又怎么是好？

如果读者能给出答案，那最好不过；如果给不出，后面章节会给你们答案，且学且不急。

7.1 数字电路和 C 语言中的逻辑运算

二进制的逻辑运算，又称其为布尔运算。无论 C 语言中，还是数字电路中，逻辑运算不可缺。在逻辑范畴中，只有“真”和“假”。先来目睹一下 C 语言中的逻辑运算，“0”为“假”，“非 0”为真，不要理解为只有 1 是“真”，2、-43、100 同样也是真。

（1）逻辑运算（是按整体运算），通常叫做逻辑运算符。

&&（and）：逻辑与，只有同为真时结果才为真，近似于乘法。

||（or）：逻辑或，只有同为假时结果才为假，近似于加法。

!（not）：逻辑非，条件为真，结果为假，近似于相反数。

（2）逻辑运算（按每个位来运算），通常叫做位运算符。

&：按位与，变量的每一位都参与（下同），例如：A = 0b0101 1010，B = 0b1010 1010，则 A & B = 0b0000 1010。

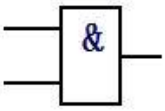
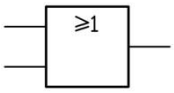
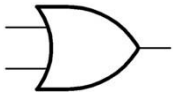
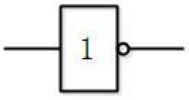
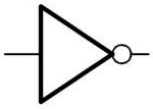
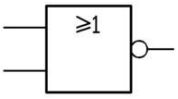
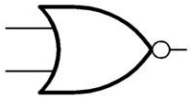
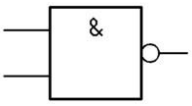
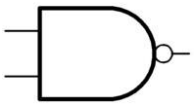
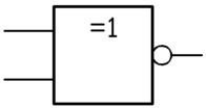
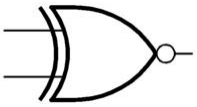
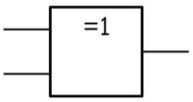
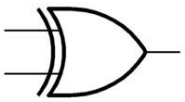
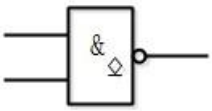
|：按位或。则 A | B = 0b1111 1010。

~：按位取反。则~A = 0b1010 0101。

^:　按位异或，异或的意思是，如果运算双方的值不同（即相异），则结果为真，双方值相同则结果为假。这样 A^B = 0b1111 0000。

数字电路的逻辑运算。读者以后看资料或数据手册时，经常会遇到一些逻辑运算符号，笔者列举到这里，以便读者以后查阅。所有符合如表 4-1 所示。

表 7-1 数字逻辑运算符合

序号	运算名称	国际标准符合	国外流行符号
1	与门		
2	或门		
3	非门		
4	或非门		
5	与非门		
6	同或门		
7	异或门		
8	集电极开漏 OC 门 漏极开漏 OD 门		----

7.2 上拉电阻和下拉电阻

在说上、下拉电阻之前，不得不提两个电流的概念，这两个电流的概念笔者以前也提过，只是没有系统说明，这里就详细说说，希望对读者有所帮助。

7.2.1 拉电流与灌电流

一、拉电流和灌电流的概念

拉电流和灌电流是衡量电路输出驱动能力的参数，这种说法一般用在数字电路中。特别注意：拉、灌都是对输出端而言的，所以是驱动能力。

这里首先要说明，芯片手册中的拉、灌电流是一个参数值，是芯片在实际电路中允许输出端拉、灌电流的上限值（所允许的最大值）。而下面要讲的这个概念是电路中的实际值。

由于数字电路的输出只有高、低（0、1）两种电平值，高电平输出时，一般是输出端对负载提供电流，其提供电流的数值叫“拉电流”；低电平输出时，一般是输出端要吸收负载的电流，其吸收电流的数值叫“灌（入）电流”。

对于输入电流的器件而言：灌入电流和吸收电流都是输入的，灌入电流是被动的，吸收电流是主动的。如果外部电流通过芯片引脚向芯片内‘流入’称为灌电流（被灌入）；反之如果内部电流通过芯片引脚从芯片内‘流出’称为拉电流（被拉出）。

二、为什么能够衡量输出驱动能力

当逻辑门输出端是低电平时，灌入逻辑门的电流称为灌电流，灌电流越大，输出端的低电平就越高。由三极管输出特性曲线也可以看出，灌电流越大，饱和压降越大，低电平越大。然而，逻辑门的低电平是有一定限制的，它有一个最大值 UOLMAX。在逻辑门工作时，不允许超过这个数值,TTL 逻辑门的规范规定 UOLMAX≤0.4～0.5V(STC89C52 的 UOLMAX 为 0.7V)。所以，灌电流有一个上限。

当逻辑门输出端是高电平时，逻辑门输出端的电流是从逻辑门中流出，这个电流称为拉电流。拉电流越大，输出端的高电平就越低。这是因为输出级三极管是有内阻的，内阻上的电压降会使输出电压下降。拉电流越大，输出端的高电平越低。然而，逻辑门的高电平是有一定限制的，它有一个最小值 UOHMIN。在逻辑门工作时，不允许超过这个数值，TTL 逻辑门的规范规定 UOHMIN≥2.4V（STC89C52 的 UOLMAX 为 1.8V)。所以，拉电流也有一个上限。

可见，输出端的拉电流和灌电流都有一个上限，否则高电平输出时，拉电流会使输出电平低于 UOHMIN；低电平输出时，灌电流会使输出电平高于 UOLMAX。所以，拉电流与灌电流反映了输出驱动能力（芯片的拉、灌电流参数值越大，意味着该芯片可以接更多的负载，因为灌电流是负载给的，负载越多，被灌入的电流越大)。

由于高电平输入电流很小，在微安级，一般可以不必考虑，低电平电流较大，在毫安级，所以，往往低电平的灌电流不超标就不会有问题。用扇出系数来说明逻辑门驱动同类门的能力，扇出系数 No 是低电平最大输出电流和低电平最大输入电流的比值。

在集成电路中，吸电流、拉电流输出和灌电流输出是一个很重要的概念。拉即泄，主动输出电流，是从输出口输出电流；灌即充，被动输入电流，是从输出端口流入；吸则是主动吸入电流，是从输入端口流入。

吸电流和灌电流就是从芯片外电路通过引脚流入芯片内的电流,区别在于吸收电流是主动的，从芯片输入端流入的叫吸收电流。灌入电流是被动的,从输出端流入的叫灌入电流。拉电流是数字电路输出高电平给负载提供的输出电流,灌电流时输出低电平是外部给数字电路的输入电流，它们实际就是输入、输出电流的能力。

吸收电流是对输入端（输入端吸入）而言的；而拉电流（输出端流出）和灌电流（输出端被灌入）是相对输出端而言的。

此时读者再回过头去分析一下 3.3.2 节的硬件设计部分，是不是原理很好理解了。

7.2.2 上拉电阻和下拉电阻

笔者在讲述三极管时提到过上拉电阻和下拉电阻,哪里并没有详细讲解上下拉电阻究竟有何用，或者说电阻究竟取多大为宜，接下来围绕这几个问题来讲述一下上下拉电阻。

一、上、下拉电阻的概念

上拉电阻就是把不确定的信号通过一个电阻嵌位在高电平，此电阻还起到限流的作用。同理，下拉电阻是把不确定的信号嵌位在低电平。上拉电阻是针对器件的输入电流（也即灌电流)，而下拉电阻针对的是输出电流（也即拉电流)。

二、上、下拉电阻的作用

（1）上拉就是将不确定的信号通过一个电阻嵌位在高电平，以此来给芯片引脚一个确定的电平，以免使芯片引脚悬空发生逻辑错乱。

（2）为加大输出引脚的驱动能力。下拉同理。

三、上、下拉电阻的应用总结

（1）当 TTL 电路驱动 CMOS 电路时，如果 TTL 电路输出的高电平低于 CMOS 电路的最低高电平（一般为 3.5V），这时就需要在 TTL 的输出端接上拉电阻，以提高输出高电平的值。

（2）OC 门电路必须加上拉电阻，以提高输出的高电平值。

（3）为加大输出引脚的驱动能力，有的单片机管脚上也常使用上拉电阻。

（4）在 CMOS 芯片上，为了防止静电造成损坏，不用的管脚不能悬空，一般接上拉电阻以降低输入阻抗，提供泄荷通路。

（5）芯片的管脚加上拉电阻来提高输出电平，从而提高芯片输入信号的噪声容限，以提高增强干扰能力。

（6）提高总线的抗电磁干扰能力。管脚悬空就比较容易接受外界的电磁干扰。

（7）长线传输中电阻不匹配容易引起反射波干扰，加上下拉电阻是为了电阻匹配，从而有效抑制反射波干扰。

四、上、下拉电阻的选取原则

（1）从节约功耗及芯片的灌电流能力考虑应当足够大；电阻大，电流小。

（2）从确保足够的驱动电流考虑应当足够小；电阻小，电流大。

（3）对于高速电路，过大的上拉电阻可能会使边沿变平缓。

综合考虑以上三点，通常在 1K~10K 之间选取，笔者一般选用 4.7K 或 10K。下拉电阻也有类似的道理。

7.3 工程图示按键

按键在电子设备中应用是很广泛的，主要作用就是人机交换，即通过按键来控制电子设备。按键在实际生活中是无处不在，从手机到电脑，再如上图的机顶盒和遥控器。下班之后闲着没事干，一按开关，机顶盒打开，此时可能播放着浙江卫视，读者并不像看“中国好声音”，而想看科比的一次背身单打、梅西的一次秒射，拿起遥控器，按一下频道加减（CH+、CH-），若声音大小不对，听不到解说，拿起遥控器继续按声音控制键（VOL-、VOL+），总有一款适合自己，呵呵，从小到大，不知读者按过多少次键，可读者知道按键是如何检测的吗？那就借着单片机踏上按键之路吧。

7.4 按键的点点滴滴

7.4.1 原理说明

一、键盘的分类

键盘分为编码键盘和非编码键盘。键盘上闭合键的识别由专用的硬件编码器实现，并产生键编码号或键值的称为编码键盘，如计算机键盘。而靠软件编程来识别的称为非编码键盘；

在单片机组成的各种系统中，用的最多的是非编码键盘，也有用到编码键盘的。

非编码键盘又分为：独立键盘和行列式（又称为矩阵式）键盘。

1.独立按键

（1）独立按键，每个按键单独占用一个 I/O 口，I/O 口的高低电平反映了即对应按键的

状态。

（2）独立按键的状态：未按下，对应端口为高电平；按下键，对应端口为低电平。

（3）独立按键的识别流程。

1）查询是否有按键按下？

2）查询是否哪个按键按下？

3）执行按下键相应的键处理

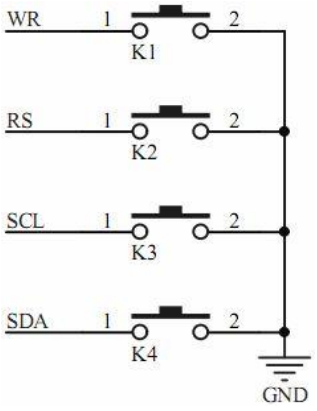


图 7-1 独立按键接口图

现以 MGMC-V1.0 实验板上的独立按键为例，如图 7-1 所示，简述四个按键的检测流程。四个按键分别连接在单片机的 P3.4（WR）、P3.5（RS）、P3.6（SCL）、P3.7（SDA）口上，按流程，检测是否有按键按下，就是读取该四个端口的状态值，若四个口都为高电平（前面说这，单片机默认电平为高电平），说明没有按键按下；若其中某个端口的状态值变为低电平（0V），说明此端口对应的按键被按下，之后就是处理该按键按下的具体操作。

2.矩阵按键

在键盘中按键数量较多时，为了减少 I/O 口的占用（如图 7-4，16 个按键按矩阵的接法比独立的接法少用 8 个 I/O 口），通常将按键排列成矩阵形式，也即每条水平线和垂直线在交叉处不直接连通，而是通过一个按键加以连接。这样的设计方法在硬件上节省 I/O 口，可是在软件上会变得比较复杂，但也不是太难，读者们不要紧张哈，无非就是检测高低电平，具体方法软件分析部分会做详细讲解。

二、键盘消抖的基本原理

通常的按键所用开关为机械弹性开关，当机械触点断开、闭合时，由于机械触点的弹性作用，一个按键在闭合时不会马上稳定的接通，在断开时也不会立即断开。按键按下时会有抖动，也就是说只按一次按键，可实际产生的“按下次数”却是多次的，因而在闭合和断开的瞬间均伴有一连串的抖动，如图 7-2 所示，为了避免这种现象而做的措施就是按键消抖。

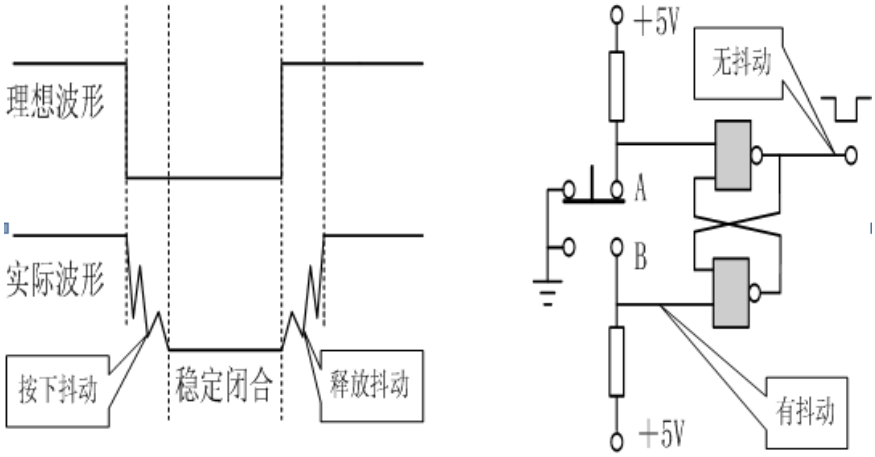


图 7-2 按键抖动和硬件消抖示意图

消抖方法分为：硬件消抖和软件消抖。

1、硬件消抖：在键数较少时可采用硬件方法消抖，如 7-2 右图所示，用 RS 触发器来消抖。图中两个“与非”门构成一个 RS 触发器，当按键未按下时，输出 1；当按键按下时，输出为 0。此时即使用按键的机械性能，使按键因弹性抖动而产生瞬间断开（抖动跳开 B），要按键不返回原始状态 A，双稳态电路的状态不改变，输出保持为 0，不会产生抖动的波形。

也就是说，即使 B 点的电压波形是抖动波形，但经双稳态电路之后，其输出为正规的矩形波。

补充：有时还会采用电容来进行消抖，这里不赘，读者自行研究。

2、软件消抖：如果按键较多，常用软件方法来去抖，即检测到有按键按下时，执行一段延时程序，具体延时时间依机械性能而定，常用的延时是：5MS ~ 20MS，即按键抖动的这段时间不进行检测，等到按键稳定时再读取状态，若仍然为闭合状态电平，则认为真正有按键按下。

是不是被这些糊里糊涂的概念搞得雾里来雾里去的，没事，若不懂就当听了会天书。之后笔者谈谈关于按键的一点工作经历，或许对读者有帮助，若没就当听故事吧。笔者当初所在公司是做机顶盒的，其中参与了 METER 前控板的开发，该系统用 12 个按键和一个编码开关来控制，关于编码开关的知识，读者可以上网搜搜，或者关注笔者的博客，限于篇幅和时间关系，在此就不做介绍了，这里主要简述 12 个按键的处理过程和控制处理器的选型。

图 7-3 Meter 界面图

说按键之前，先简单介绍一下 METER，所谓 METER，就是寻星仪（手持电视和机顶盒的合体——笔者个人的定义），其实物如图 7-3 所示，主要用于电视接收锅的安装等。该 METER 的设计方案是：一款索尼的 32 位处理器+3 个 TUNER（S/T/C）+蓝牙+传感器+WIFI+9 寸的 TFT+前控板。3 个 TUNER（高频头）用 FPGA 做数据选择，前控板用单片机控制，没有用 51，而是 STM8，肯定还有辅助电路，听着听着，是不是感觉越糊涂了，没事，等到读者学了 FPGA、ARM 之后，这些东西就变的简单了。

前控板为何交给笔者，因为主板的软件是软件部门（公司一般软硬是分开的）做的，那前控板呢，虽然看着简单，但是要做按键检测、编码开关的解码、指示灯的控制，最重要的一点，与 ARM 通信，这样必须得有一个“软硬通吃”的人来搞，所以读者懂得。具体电路就不做详细介绍了，若想了解可以联系笔者，笔者无偿提供。当初预算成本是单片机的价格不能超过 2 元，就这些功能， I/O 口要占掉 10 多个，若选常用的 51 单片机、AVR 单片机等价格都高于 2 元，若选松翰等单片机，虽然价格低，可以后采购、供货等都是问题，所以必须的在 I/O 口上下工夫，怎么减少 I/O 的用量，或者选择那些常用又便宜的单片机，是工作的重中之重，综合考虑下了，只有这 12 个按键有手脚可做，别的电路对于 I/O 口的需求基本是固定的，笔者首先想到的是矩阵按键，可上网查询按键相关知识时，无意间留意到了一种按键的扫描方法——堪称一绝的按键扫描方法，笔者觉得很好玩，拿来研究了一阵，发现确实很不错，笔者强烈推荐读者学习一下。具体见下面链接。

/* 引用声明 http://wenku.baidu.com/view/3b874d6baf1ffc4ffe47acc5.html */

笔者当初用的并不是——堪称一绝的按键扫描方法，而是 A/D 采样法，先看原理图（如图 7-4 所示），再做讲解。

接下来笔者对此图最简单的介绍，图中有 12 个按键，这里限于篇幅，笔者只贴了三个按键的电路图，其实后面的电路原理都是一样的，就是先串联一个 10K 的电阻，在对地并联一个按键。其中“KEY_VALUE”端子接单片机的 I/O 口（具有 A/D 功能），这样没有按键按下时，I/O 口电压为 VDD（+3.3V），第一个按键按下时 I/O 口的电压为 0V，第二个按键按下时电压为：1/2VDD，这样电压依次为：2/3VDD、3/4VDD...11/12VDD，之后单片机通过采样 I/O 的电压值，就可以判断是哪个按键被按下，只是这种接法具有优先级（左高右低）。该电路的优点是只需占用一个 I/O 口，缺点是所接的控制器必须得有 A/D（关于 A/D 的知识见笔记 12）转换功能，或者要接一块 A/D 转换芯片，这样势必也会增加成本，笔者当初选择的是 STM8S003F3p6，该单片机的价格不到 1.5 元，可以说真是一款性价比很高的 MCU，建议读者学完 51 之后，也可以学学此系列的单片机，对以后做工程非常有利。

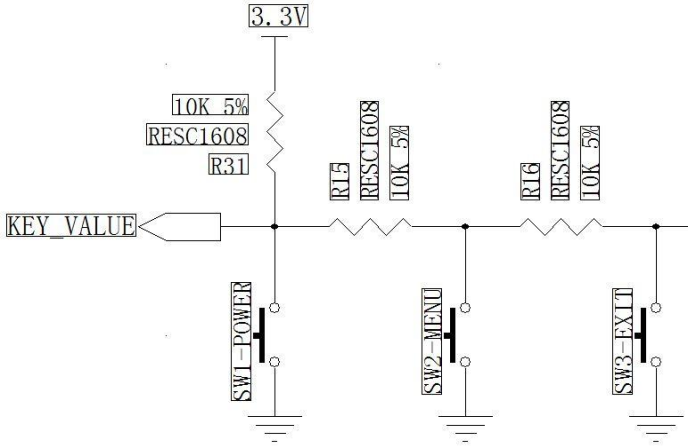


图 7-4 按键原理图

或许说到这里，读者可能会说，为何要学这么多种单片机，或者说那种高级，我直接学习哪一种不就得了，就作者的经验，单片机只需学会 51（51 资料齐全、容易上手）就可以了，别的都是类似的。残弈悟恩只认真学过 51，可做“语音智能小车”时发现，只需简单学习一下，凌阳的 61 单片机也不在话下；做“校园路灯控制系统”时发现，只需看看资料，Silicon Lab 公司的 C8051F 系列的单片机也能熟练运用；当然上面提到的 STM8，笔者以前也没学过，用的时候看看数据手册，也能搞定。所以，单片机不是读者学了多少种，而是看一种是否学精了。进一步说只要会了单片机，像 STM32、FPGA、ARM、DSP 这些高端技术都是比较容易入门的，倘若读者连单片机都没玩好，这些技术恐怕真就难于上青天了。

7.4.2 硬件分析

按键种类繁多，这里就不一一列举了，MGMC-V1.0 实验板上用的是 2 脚的轻触按键，原理就是按下导通，否则断开。这样就可以设计出如图 7-5 所示的矩阵按键原理图，这些最后连接在了单片机的 P3 口，如图 7-6 所示。接着看看 MGMC-V1.0 实验板上的矩阵、独立按键的实物图，这样会有一个感观的认识，具体如图 7-7 所示

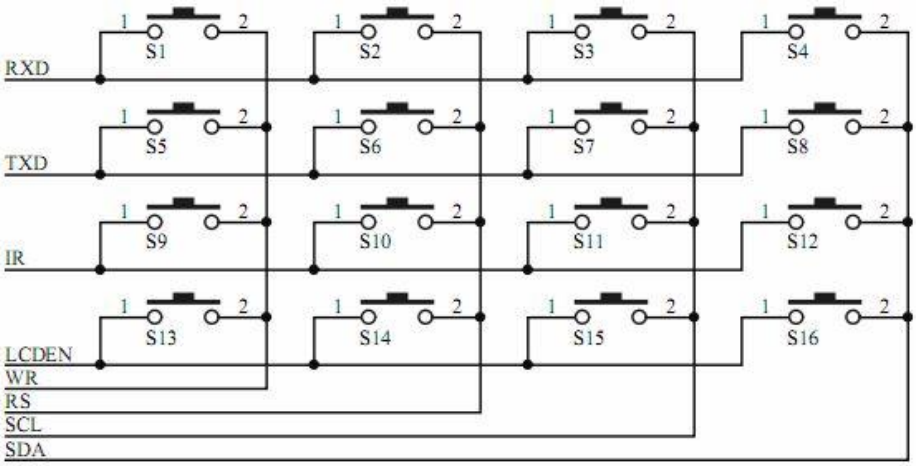


图 7-5 矩阵按键原理图

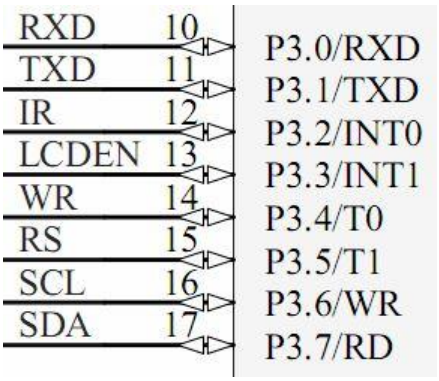


图 7-6 独立、矩阵按键与单片机接口图

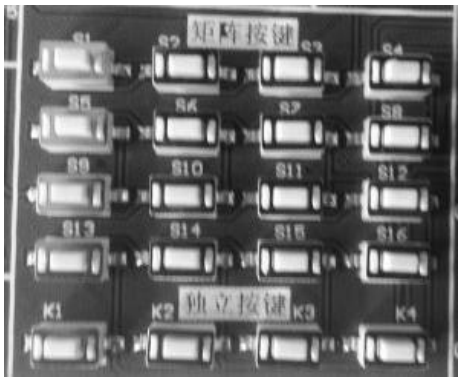


图 7-7 MGMC-V1.0 实验板的独立和矩阵按键

7.4.3 解读软件

经过以上的艰苦奋斗，原理懂了，硬件搭建好了，那万事俱备，只欠读者写程序了。那就看看这些软件如何实现。由于独立按键比较简单，就不专门讲解了，在后面实例中附带就 OK 了，这里主要说明矩阵按键检测方法。

矩阵按键一般有两种检测法：行扫描法和高低电平翻转法。

在说解释两种方法之前，先说说一种关系。因为笔者在培训时发现，学生在这儿就是不能理解，所以有了以下的假设，假如做这样一个“傻”电路，将 P3.0、P3.1、P3.2、P3.3 分别与 P3.4、P3.5、P3.6、P3.7 用导线相连，此时如果给 P3 口赋值 0xfe，那么读到的值就为：0xee。这是一种线“与”的关系，也即 P3.0 的“0”与 P3.4 的“1”进行“与”运算，结果为“0”，因此 P3.4 也会变成“0”。读者们，铭记几句话：NBA 中有一种飞翔叫“乔丹”，有一种背身单打叫“科比”，有一种干拔叫“麦蒂”；单片机中有一种关系叫“线与”。

（1）行扫描法

所谓行扫描法就是先给 4 行中的某一行低电平，别的全给高电平，之后检测列所对应的端口，若都为高，则没有按键按下，相反有按键按下，也可以给 4 列中的某一列低电平，别的全给高电平，之后检测行所对应的端口，若都为高，则表明没有按键按下，相反有按键按下，具体如何检测，笔者来举例说明。

首先给 P3 口赋值 0xfe（0b1111 1110），这样只有第一行（P3.0）为低，别的全为高，之后读取 P3 的状态，若 P3 口电平还是 0xfe，则没有按键按下，若值不是 0xfe 说明有按键按下，具体是哪个，由此时读到值决定，值为 0xee 则表明按下的是 S1，若是 0xde 则是 S2（同理 0xbe→S3、0x7e→S4）；之后给 P3 口赋值 0xfd（0b1111 1101），这样第二行（P3.1）

为低，同理读取 P3 口的状态值，若为 0xfd 表明没有按键按下，若为 0xed 则 S5 按下（同理 0xdd→S6、0xbd→S7、0x7d→S8）；这样依次赋值 0xfb（检测第三行）、0xf7（检测第四行），从而就可以检测出 S9~S16。

（2）高低平翻转法

首先让 P3 口高四位为 1，低四位为 0。若有按键按下，则高四位中会有一个 1 翻转为 0，低四位不会变，此时即可确定被按下的键的列位置。

然后让 P3 口高四位为 0，低四位为 1。若有按键按下，则低四位中会有一个 1 翻转为 0，高四位不会变，此时即可确定被按下的键的行位置。

最后将两次读到的数值进行或运算，从而确定是那个键被按下了。同样举例说明。

首先给 P3 口赋值 0xf0，接着读取 P3 口的状态值，若读到的值为 0xe0，表明第一列有按键按下；接着给 P3 口赋值 0x0f 并读取 P3 口的状态值，若值为 0x0e，则表明第一行有按键按下，最后把 0xe0 和 0x0e 进行按位或运算，结果为 0xee，0xee 怎么这么眼熟，因为方法一中 S1 按下对应的值也是 0xee。这样，一个被按下键即在第一列，又在第一行，不是 S1 是那个键呢？由此可见，两种检测的过程有所不同，但结果还是一一统一的，至于读者选用哪一种，那就仁者见仁，智者见智了，^_^。

最后总结一下矩阵按键的检测过程，无非就是：

赋值（有规律）→读值（高低平翻转法还需运算）→判值（由值确定按键）。

7.5 实例诠释按键

实例 1 孤独的操作手—独立按键

实例说明，就是按下 MGMC-V1.0 实验板上的 K1 键则 LED1 亮，按下 K2 键 LED1 灭。下面就附个小儿科代码，望读者们品尝。

```
1.  #include <reg52.h>
2.  typedef unsigned int uInt16;
3.  sbit LED1 = P2^0;
4.  sbit KEY1 = P3^4;
5.  sbit KEY2 = P3^5;
6.  void DelayMS(uInt16 ValMS)
7.  {    /* 走不出的两个 for 循环 */    }
8.  void main(void)
9.  {
10.     while(1)
11.     {
12.         if(0 == KEY1)              // 检测按键是否按下
13.         {
14.             DelayMS(10);           // 延时去抖
15.             if(0 == KEY1)          // 再次检测
16.             {
17.                 LED1 = 0;          // 点亮 LED 灯
18.                 while(!KEY1);      // 等待按键弹起
19.             }
20.         }
21.         if(!KEY2)                  // 条件判断的另一种写法
```

```
22.             {
23.                 DelayMS(10);            // 延时去抖
24.                 if(!KEY2)
25.                 {
26.                     LED1 = 1;
27.                     while(!KEY2);
28.                 }
29.             }
30.         }
31.     }
```

上述例子，实在太简单了，笔者就不讲究了，这里说个“高级”话题，还是那个 DelayMS() 函数，为何一直在说不好，又在继续用。一，笔者怕在这里讲述“高级”话题会将新手们扼杀在幼儿园阶段。所以先来个“傻瓜式”的按键扫描实例，为读者热热身，为何“傻瓜”，广告之后，自会有“精彩”的武侠剧上演。

实例 2 孕育生命的摇篮—矩阵（按键_行列扫描法）

实例简述，在一次按下矩阵按键 S1~S16 时，8 位数码管都依次显示 0、1…E、F。该程序先用行扫描的方式来写，具体代码如下。

```
1.  #include <reg52.h>
2.  #define uInt16 unsigned int
3.  #define uChar8 unsigned char
4.  #define DATA P0                         //数据口
5.  #define KEYPORT    P3                   //键盘接入端口
6.  sbit SEG_SELECT = P1^7;                 //段选控制端
7.  sbit BIT_SELECT = P1^6;                 //位选控制端
8.  uChar8 code SEG_Tab[] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,
9.  0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71};//段选显示表格
10. uChar8 g_ucKeyNum = 16;                 //赋个初值
11. void DelayMS(uInt16 ValMS)
12. {    /* go on... */   }
13. /* ******************************************************* */
14. //  函数名称：ScanKey(void)
15. //  函数功能：矩阵按键扫描
16. //  入口参数：无
17. //  出口参数：无
18. /* ******************************************************* */
19. void ScanKey(void)
20. {
21.     uChar8 ucTemp;
22.     KEYPORT = 0xfe;                     //检测第一行
23.     ucTemp = KEYPORT;                   //读取键盘端口数值
24.     if(ucTemp != 0xfe)                  //若是不等于 0xFe 表示第一行有按键按下
25.     {
26.         DelayMS(5);                     //去抖
```

```
27.             ucTemp = KEYPORT;                   //读端口值
28.             if(ucTemp != 0xfe)                  //再次判断
29.             {
30.                 ucTemp = KEYPORT;               //取键值
31.                 switch(ucTemp)                  //判断键值对应键码
32.                 {
33.                     case 0xee:g_ucKeyNum = 0;break;     //第一行第一个按下
34.                     case 0xde:g_ucKeyNum = 1;break;     //第一行第二个按下
35.                     case 0xbe:g_ucKeyNum = 2;break;     //第一行第三个按下
36.                     case 0x7e:g_ucKeyNum = 3;break;     //第一行第四个按下
37.                 }
38.                 while(KEYPORT!= 0xfe);          //按键释放检测
39.             }
40.         }
41.         KEYPORT = 0xfd;                         //检测第二行
42.         ucTemp = KEYPORT;
43.         if(ucTemp != 0xfd)
44.         {
45.             DelayMS(5);
46.             ucTemp = KEYPORT;
47.             if(ucTemp != 0xfd)
48.             {
49.                 ucTemp = KEYPORT;
50.                 switch(ucTemp)
51.                 {
52.                     case 0xed:g_ucKeyNum = 4;break;
53.                     case 0xdd:g_ucKeyNum = 5;break;
54.                     case 0xbd:g_ucKeyNum = 6;break;
55.                     case 0x7d:g_ucKeyNum = 7;break;
56.                 }
57.                 while(KEYPORT != 0xfd);
58.             }
59.         }
60.         KEYPORT = 0xfb;                         //检测第三行
61.         ucTemp = KEYPORT;
62.         if(ucTemp != 0xfb)
63.         {
64.             DelayMS(5);
65.             ucTemp = KEYPORT;
66.             if(ucTemp != 0xfb)
67.             {
68.                 ucTemp = KEYPORT;
69.                 switch(ucTemp)
70.                 {
```

```
71.                         case 0xeb:g_ucKeyNum = 8;break;
72.                         case 0xdb:g_ucKeyNum = 9;break;
73.                         case 0xbb:g_ucKeyNum = 10;break;
74.                         case 0x7b:g_ucKeyNum = 11;break;
75.                     }
76.                     while(KEYPORT != 0xfb);
77.                 }
78.             }
79.             KEYPORT = 0xf7;                         //检测第四行
80.             ucTemp = KEYPORT;
81.             if(ucTemp != 0xf7)
82.             {
83.                 DelayMS(5);
84.                 ucTemp = KEYPORT;
85.                 if(ucTemp != 0xf7)
86.                 {
87.                     ucTemp = KEYPORT;
88.                     switch(ucTemp)
89.                     {
90.                         case 0xe7:g_ucKeyNum = 12;break;
91.                         case 0xd7:g_ucKeyNum = 13;break;
92.                         case 0xb7:g_ucKeyNum = 14;break;
93.                         case 0x77:g_ucKeyNum = 15;break;
94.                     }
95.                     while(KEYPORT != 0xf7);
96.                 }
97.             }
98.  }
99.  /* ******************************************************** */
100. //  函数名称：Display (void)
101. //  函数功能：数码管显示函数
102. //  入口参数：要显示的数值（ucVal）
103. //  出口参数：无
104. /* ******************************************************** */
105. void Display(uChar8 ucVal)
106. {
107.     if(ucVal == 16)             //若键值是 16 即没有按键按下，则不显示
108.     {
109.         BIT_SELECT = 1; DATA = 0xff; BIT_SELECT = 0;
110.     }
111.     else                        //若有按下，显示对应键值
112.     {
113.         BIT_SELECT = 1; DATA = 0x00; BIT_SELECT = 0;                //选中 8 位数码管
114.         SEG_SELECT = 1; DATA = SEG_Tab[ucVal]; SEG_SELECT = 0;  //送段选数据
```

```
115.            DelayMS(10);DATA = 0x00;
116.        }
117. }
118. void main(void)
119. {
120.        while(1)
121.        {
122.            ScanKey();
123.            Display(g_ucKeyNum);
124.        }
125. }
```

讲述该实例之前，先来张流程图，如图 7-8 所示，之后再来详解。

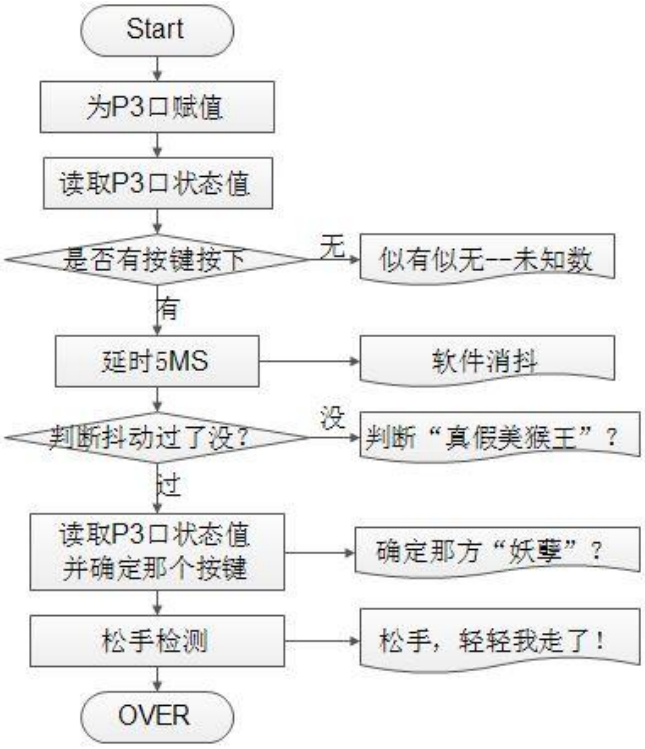


图 7-8 行列扫描法流程图

该程序实现的原理在软件分析部分已经做了详解。检测过程如图 7-8 所示，再结合源代码，相信读者能够理解，因此就再也不烦大家了，望见谅。

实例 3 孕育生命的摇篮—矩阵（按键_高低电平翻转法）

该实例与实例 14 相比，除了按键扫描函数以外，别的都是相同的，鉴于篇幅原因，笔者在这里只贴按键扫描函数，其实现过程可参考软件分析部分。

```
1.  /* ******************************************************* */
2.  // 函数名称：ScanKey(void)
3.  // 函数功能：矩阵按键扫描
4.  // 入口参数：无
5.  // 出口参数：无
6.  /* ******************************************************* */
7.  void ScanKey(void)
```

```
{
    uChar8 RowTemp,ColumnTemp,RowColTemp;
    KEYPORT = 0xf0;                          // 先给高四位高电平
    RowTemp = KEYPORT & 0xf0;                // 读取行值，为确定是那一行用
    if(RowTemp != 0xf0)                      // 判断是否有按键
    {
        DelayMS(5);                          // 去抖动
        if(RowTemp != 0xf0)
        {
            RowTemp = KEYPORT & 0xf0;        // 说明真的有键按下，那么读取行值
            KEYPORT = 0x0f;                  // 接着给低四位高电平
            ColumnTemp = KEYPORT & 0x0f;     // 读取列值，为确定是那一列用
            RowColTemp = RowTemp | ColumnTemp;
            // 行列值进行按位或运算，从而确定行列值
            while((KEYPORT & 0x0f) != 0x0f); // 松手检测
        }
    }
    switch(RowColTemp)                       // 确定按键
    {
        case 0xee:  g_ucKeyNum = 0; break;   // 以下分别是按键：S1、S2、...、
        case 0xde:  g_ucKeyNum = 1; break;   // S16 等按键被按下了。
        case 0xbe:  g_ucKeyNum = 2; break;
        case 0x7e:  g_ucKeyNum = 3; break;
        case 0xed:  g_ucKeyNum = 4; break;
        case 0xdd:  g_ucKeyNum = 5; break;
        case 0xbd:  g_ucKeyNum = 6; break;
        case 0x7d:  g_ucKeyNum = 7; break;
        case 0xeb:  g_ucKeyNum = 8; break;
        case 0xdb:  g_ucKeyNum = 9; break;
        case 0xbb:  g_ucKeyNum = 10; break;
        case 0x7b:  g_ucKeyNum = 11; break;
        case 0xe7:  g_ucKeyNum = 12; break;
        case 0xd7:  g_ucKeyNum = 13; break;
        case 0xb7:  g_ucKeyNum = 14; break;
        case 0x77:  g_ucKeyNum = 15; break;
        default:    g_ucKeyNum = 16; break;
    }
}
```

程序比较简单，并且有详细的注释，因此，笔者即不费口舌了。读者可以思考一下，这两种写法有什么共同点和不同点，或者说那种方法好理解，更紧凑一点？当然读者还可以思考、总结更好的矩阵按键扫描方法。