圆梦小车 Step by Step 之一

—— 第一个程序“ Hello World ”

“ **圆梦小车 DIY 套件** ”推出后，受到了许多爱好者的青睐，深感欣慰！

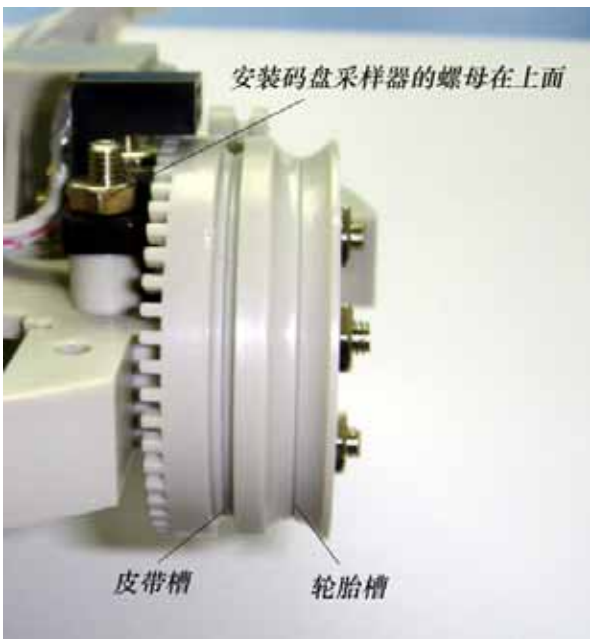
但是，也感到有一些初学者拿到后不知该如何入手？咨询了许多问题。为了减少困惑，我决定逐步基于小车平台写一些示例程序供大家参考，更主要的是期望起到“ 抛砖引玉 ”的作用，营造出一个良好的氛围，让“ 高手 ”有展示的机会，让初学者有尝试的可能。

因为是第一篇，所以顺便将反馈的问题予以回复。

一、 装配注意事项

小车装配时，结构部分要注意：

A. 轮毂由三个部分叠装而成，形成两个槽 —— 皮带槽和轮胎槽，注意两个槽的宽度不同，不要配错了 !!! 主要是中间那一层的方向。



B. 电机的小皮带轮装配要按照“ 装配说明 ”所示，不要装错方向。

C．电机安装时，注意电机皮带轮和车轮的皮带槽位置，要在一条直线上。

D．电机安装时，建议将螺母放在上面，使用 M3*8 螺丝。

E．码盘采样器先装，车轮后装，会方便一些；安装码盘采样器用 M3*16 螺丝，螺母放在上面，这样便于调整。（见上图）

F．车轮安装后，要调整码盘采样器，避免与车轮碰擦。

G．车轮轴安装使用 M3*16 带飞边的螺丝，注意，半圆头处的差别！

H．如果使用配套的控制板，码盘采样器引出线可剪去 4cm；电机引出线可剪去 2cm。

控制部分焊接、装配注意：

a) 焊接时可使用一个开口盒子将 PCB 垫起，便于插件，电阻、电容插件后可先在正面焊一个引脚，固定住器件，之后再在反面焊接。

b) 焊接插座时要对照安装说明，注意插座方向，并且对照原理图，以及连接的对象，检查是否正确，以原理图为准。一定要做这一步，因为这样可以帮助你将实物和原理图对应，理解设计，便于调试。

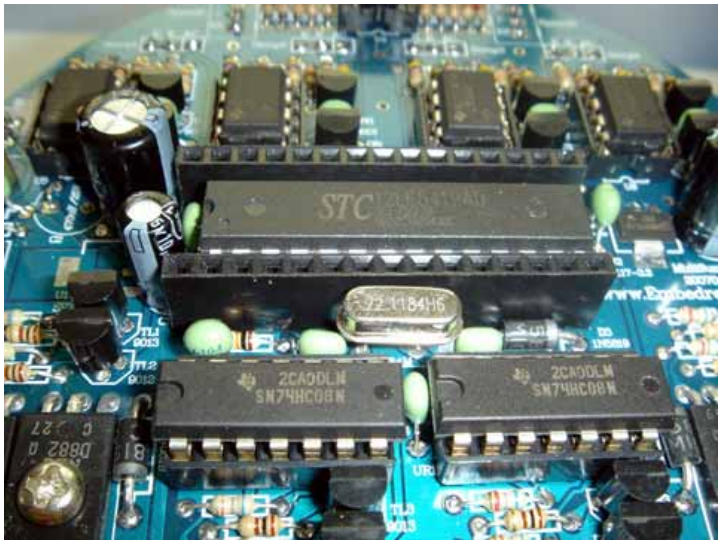
c) 焊接后不要急于插上芯片，装上电池后，打开开关，检查电源指示灯是否正常，正常后再关闭电源，插上芯片。

d) 焊接驱动三极管 D772、D882 时，应先用 M3*8 螺丝固定后再焊接，固定要松紧适

度，不要将三极管压碎了。

e) 发光二极管焊接时要注意正、负极，如果没有把握不要贴着 PCB 焊接。

f) 控制板的晶振（22.1184MHz）在焊好 22p 电容及单排插座后再焊，不要落底。



g) 扩展板上的晶振也不要落底焊接，避免引起其它部分短路。

h) 5V 电源注意原理图上的说明和装配说明的提示！

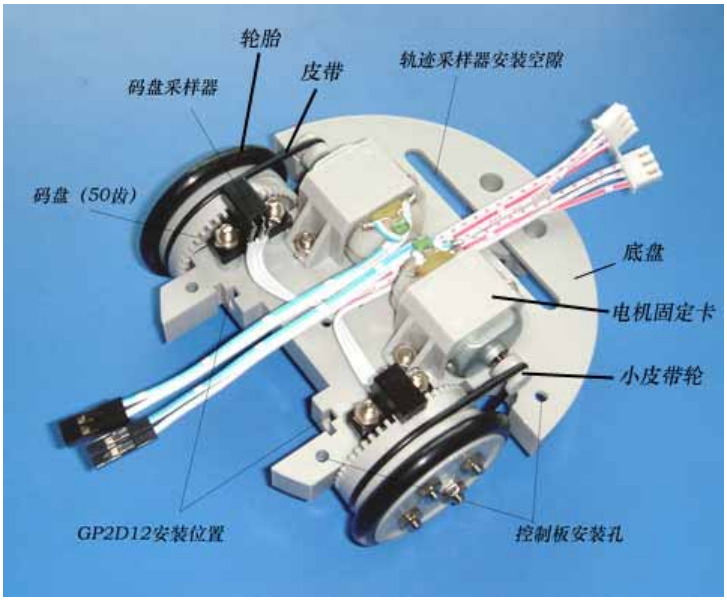
i) 焊接时，最好在原理图中寻找对应器件时，也思考一下该器件的功能、所起的作用，这样调试遇到问题时就能准确找到症结，如果只是简单“ 依样画葫芦 ”，那 DIY 就没有意义了。搞过 PC 机 DIY 的应该有所体验，那个过程对你理解 PC 机原理有多大帮助！

如果能认真看原理图以及装配说明，DIY 应该没有障碍。

二、 小车功能介绍

为了便于后面文章叙述，不产生歧义，这里在介绍功能的同时，将一些常需要涉及的部分约定一个名称，算是“ 术语定义 ”吧！这在工程上是基础，看过技术标准的一定注意到，在技术标准中一定有“ 术语定义 ”部分，以确保标准的描述准确无误。

结构部分：



底盘 —— 安装所有部件的基础；

电机固定卡 —— 固定电机；

小皮带轮 —— 电机转动输出；

码盘采样器 —— 一种遮断式光电采样器，间隙 4mm；

码盘 —— 由 50 个约 1mm 的齿组成，和码盘采样器配合，实现对车轮转动的检测；

轮胎 —— 增加摩擦力，使用 5mm 粗的 O 型圈；

皮带 —— 传动用，将电机转动传递到车轮，使用 1.8mm 粗的 O 型圈；

轨迹采样器安装空隙 —— 为轨迹采样器安装提供方便；

GP2D12 安装位置 —— 因为 GP2D12 是最常用的红外测距传感器，所以预留了一个安装位置，便于用户使用；

控制板安装孔 —— 用于固定 4 个六角铜柱，提供控制板的安装位置。

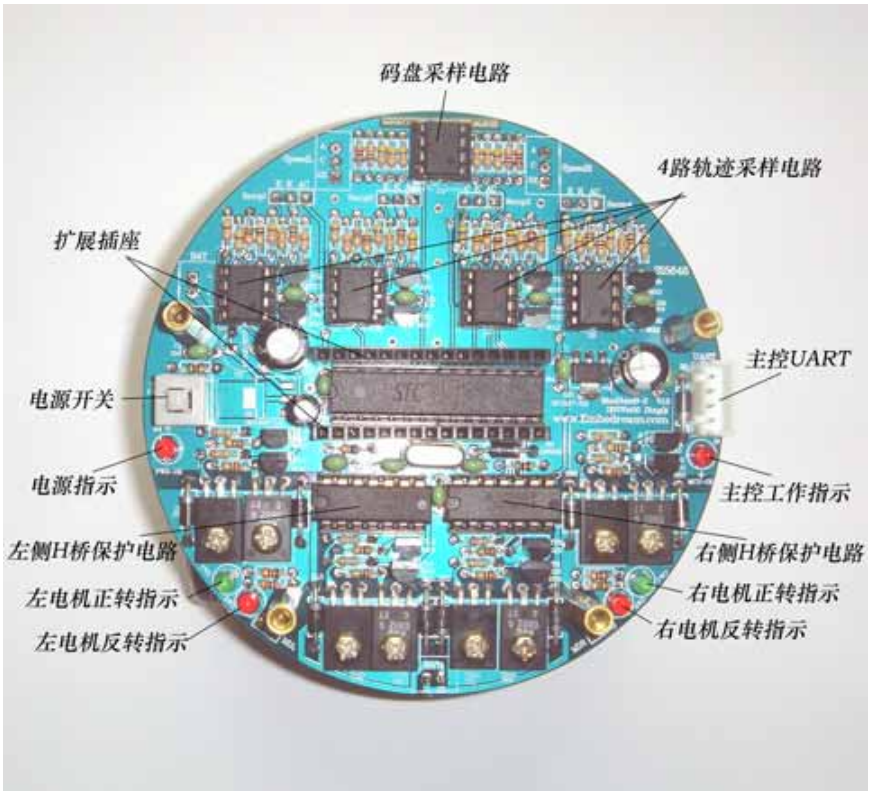
球形万向轮 —— 小车的第三点支撑；

轮毂 —— 由三部分组成，构成了皮带槽、轮胎槽以及码盘，中间还夹有滚珠轴承，以改善转动性能；

轮轴 —— 由 M3*16 带飞边半圆头螺丝构成；

电机皮带张紧槽 —— 为 4 个固定电机的螺丝提供的腰圆孔，便于调节皮带的松紧。

控制部分 —— 主控板：



电源开关 —— 只控制控制部分的电源，不控制 H 桥的电源；

电源指示 —— 监测 3V 电源（MCU 工作电源），正常则亮；

左、右侧 H 桥保护电路 —— 使用组合逻辑电路，确保 H 桥的同侧驱动三极管不同时导通，造成短路。

左、右侧电机正、反转指示 —— 调试时可以不接电机，降低耗电，同时也方便在小车运动时监测电机输出状态；

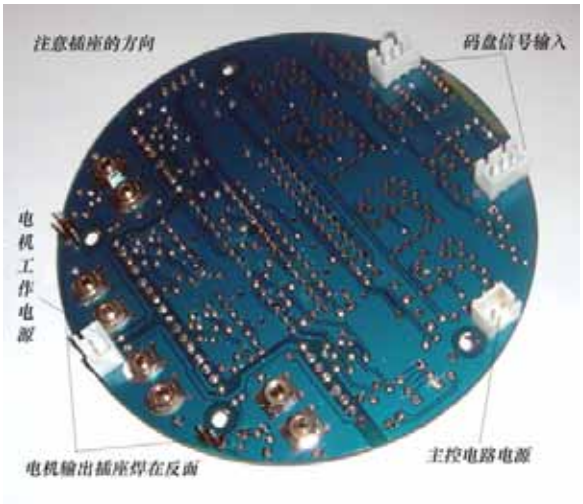
主控工作指示 —— 这个 LED 是控制部分唯一的“ 人机界面 ”，可作为程序正常工作指示，以及调试时监测所期望知道的程序运行状态，是 Debug 的主要手段之一。

主控 UART —— 控制 MCU 的 UART 接口，用于下载程序，与无线接口配合可以实现无线数据通讯，与 USB 转 UART 接口配合可以与 PC 机通讯；

4 路轨迹采样电路 —— 带有背景光消除的反射式采样电路，其功能不一定局限于轨迹采样，后面再详细描述；

码盘采样电路 —— 左右两路码盘采样预处理，主要是提供了可调整回差的施密特输入处理，以避免采样器停留在码盘齿边沿时造成误信号。

扩展插座 —— 将 MCU 的 28 个引脚及 5V 电源引到扩展板上，为扩充提供方便，同时使替换控制 MCU 成为可能。



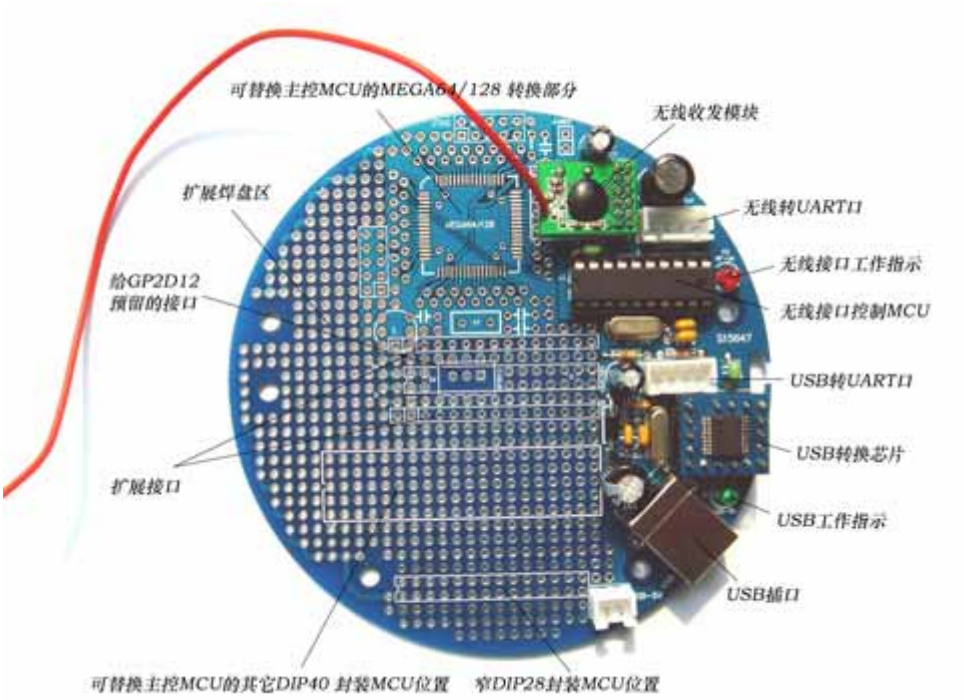
主控电源插座 —— 提供控制部分的工作电源；

电机电源插座 —— 提供 H 桥驱动电源，这样设计一是为了降低干扰，二是为改变电机工作电压提供可能（注意：改变电机工作电压时要相应修改 H 桥驱动三极管基极电阻）；

码盘输入插座 —— 两路码盘采样器输入，注意焊接引线时与电原理图对应；

电机输出插座 —— 使用普通 2 针，主要是为了方便调整电机的转动方向。

控制部分 —— 扩展板：



USB 插口 —— 因为目前的笔记本电脑已基本无串口，所以使用 USB 与 PC 连接；

USB 工作指示 —— 如果焊接正确，且在 PC 上安装了相应的驱动程序，则连接到 PC 机后，此 LED 应亮；

USB 转换芯片 —— 这是一个 USB 转 UART 芯片，和 PC 机上的驱动程序配合，可以利用 USB 口形成一个虚拟的串口，既避免了没有串口的困惑，也回避了编写 USB 控制程序的困难。由于 STC 的 MCU 下载程序对串口时序有要求，不是每款 USB 转换芯片都可以使用。

USB 转 UART 接口 —— TTL 电平串口，可以和主控电路连接，实现 PC 机与小车的通讯，

以及控制 MCU 程序下载。也可以和无线接口连接，构建一个 PC 机的无线收发接口，还可以对无线接口的 MCU 编程；

无线接口控制 MCU —— 为了方便实现无线通讯，通过此 MCU 实现 UART 通讯向无线通讯的转换，使得控制 MCU 收发程序不需要任何改动即可实现无线通讯；

无线接口工作指示 —— 作为无线控制 MCU 的人机界面，方便监测其工作状态；

无线转 UART 口 —— 与控制部分的主控 UART 相连，可以实现小车无线通讯；与 USB 转 UART 相连，可以为 PC 机增加一个无线收发适配器；

无线收发模块 —— 433MHz 的 FSK 无线收发模块，价廉物美，十分适合于这种学习性的应用。

MEGA64/128 转换部分 —— 因为 AVR 已渐流行，低端的 AVR 芯片和所选用的控制 MCU 相差不大，无替换价值；但是 MEGA64/128 功能强许多，但其封装限制了学习者，所以此处提供了一个转换区域，便于学习者选用。（买一块单独得转换 PCB 价格也不菲）

DIP40/窄 DIP28MCU 焊接区 —— 因为 PIC 单片机也广为使用，其多为 DIP40 封装，且 DIP40 封装的其它 MCU 也很多，所以设计了一个装换区，以方便选用。

扩展接口 —— 控制板上的 MCU 所有引脚都接至此，可方便的适用和测试；

GP2D12 预留接口 —— 因为底盘上设计了 GP2D12 的安装位置，所以此处也为其预留的插座位置，以方便使用。

扩展焊盘区 —— 余下的空间均布满焊盘，以提供发挥的空间。

以上所有**术语**会在后面的叙述中使用，如遇到且不清楚含义时可查看以确定其位置和功能，帮助理解所描述的内容。

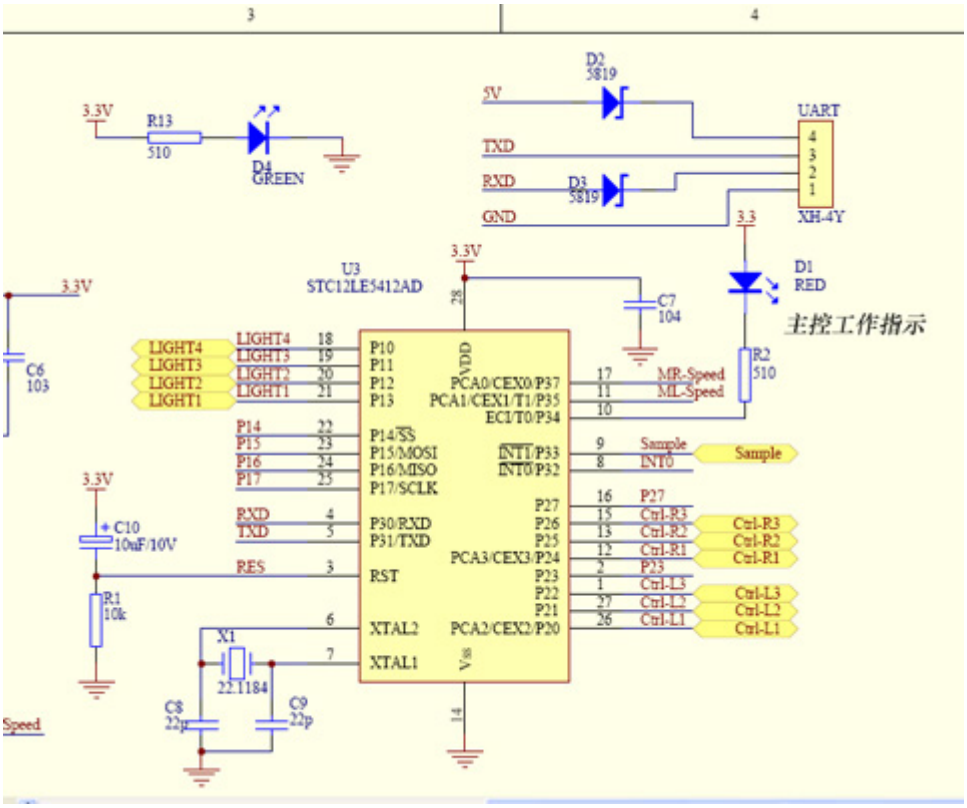
三、 小车的第一个程序

前面介绍小车各部分功能时说过，只有**主控工作指示**是作为“ 人机界面 ”设计的，相当于 PC 机的显示器，而学过 C 语言的都知道，开始的第一个程序通常是显示：

Hello World !

所以小车的第一个程序也想控制这唯一的“ 显示 ”。

要实现控制**主控工作指示**，先看一下硬件是如何连接的：



从图中可以看出，主控工作指示由 MCU 的 P3.4 控制，低电平亮、高电平灭。根据这个首先编一个简单的程序以实现对它的控制。

这里顺便探讨一个问题：是用 C 语言开始学习还是用汇编？

我倾向于使用 C 语言，至少对学过计算机原理的大学生而言如此，因为单片机应用学习主要是掌握这种智能器件的使用特点和过程，而非关注其内部工作原理，使用 C 语言可以方便的转向任何一种单片机，而不会过多的被某种单片机所羁绊。

汇编语言可以适当了解，在集成开发环境（IDE）下，多留意 C 语言编译后的汇编代码，看看是如何实现的即可。

言归正传，首先，做一个最简单的程序，控制**主控工作指示**灯闪烁：

```
/*******************************************************/
/*         圆梦小车 StepbyStep 程序之一          */
/*          —— 控制“主控工作指示”              */
/*              20070609 by Dingqi                */
/*******************************************************/
// 注：以下文档的 TAB 为 2 个字符！
/*-------------------------------------------------------
这段程序用最简单的方式控制"主控工作指示" ，
目的是示意一下如何装载一段有效的 C 语言程序。

因为程序简单，故所有内容合并在一个文件中。
-------------------------------------------------------*/

#include  <STC12C5410AD.h>
/* STC12C5410AD 的头文件,为 MCU 中各个硬件寄存器定个名称，以便 C 语言中使用*/

sbit          Work_Display = P3^4;
// 根据硬件设计用与“主控工作指示”接近的名称取代硬件名称，使程序减少与硬件的相关性

#define       LIGHT         0
// 亮逻辑值，用直观的符号化常数替换与硬件密切相关的逻辑值，增加程序的可读性、可移植性。
#define       DARK      1
// 暗逻辑值

#define       LIGHT_TIME    1000          // 亮时间，使用符号化常数，便于修改，增加程序可读性
#define       DARK_TIME     1000          // 暗时间

#define       P3MODE0        0x10         // 00010000，P3.4 设置为开漏输出，其余均为标准 51 口；
#define       P3MODE1        0x10         // 00010000，

/********** 主程序 ****************/

void DelayNms(unsigned int uiNms);    // 因为子程序在后面，所以必须在此先声明。

void main(void)
{
```

```
    // 初始化硬件
    P3M0 = P3MODE0;                 // 因为只涉及 P3 口，所以此处只初始化 P3
    P3M1 = P3MODE1;

    while(1)
    {
        Work_Display = LIGHT;           // 点亮
        DelayNms(LIGHT_TIME);

        Work_Display = DARK;            // 熄灭
        DelayNms(DARK_TIME);
    }                                   // 不断循环，在所有嵌入式应用的主程序中,都有这样一个无限循环.
}

// 延时子程序，实现用软件循环方式延时 N ms
void DelayNms(unsigned int uiNms)
{
    unsigned int i,uiCnt_ms;
    for(uiCnt_ms=0;uiCnt_ms<uiNms;uiCnt_ms++)
    {
        for(i=0;i<1400;i++)         // 1ms 延时循环，此数值由 MCU 的指令执行时间计算而来， 与时钟和 MCU 有关
        {
        }
    }
}
```

使用你所熟悉的编译器编译上述程序，将 HEX 文件下载到小车中，你将会看到所期望的结果，修改 LIGHT_TIME、DARK_TIME 的数值可随意控制闪烁效果。

注意，必须将 STC12C5410AD.h 文件拷贝到源程序所在的目录下，否则编译器不认识你所选择的 MCU，也就无法将看上去基本与 MCU 无关的 C 程序翻译成 MCU 所能执行的程序。

注意，唯一涉及的硬件 P3 口比标准 51 有所改进，详情请看 STC12C54 的数据手册。

上面的程序并不合理，之所以那样设计，是为了避免开始就为硬件所困扰。从程序中可以看出，所用的计时方式类似于人用“数数”方式计时，费力且不准，最好的方式是有一个“带闹”的计时器，到时候提醒操作。

看下一个程序：

```
/********** 主程序 *****************/
bit             gb_1msFlag;             // 1ms 中断标志

void main(void)
{
    unsigned int  guiDisplayTime;       // 显示时间计数器
    bit           gb_CurDisp;           // 记忆当前显示状态

    // 初始化硬件
    P3M0 = P3MODE0;                     // 因为只涉及 P3 口，所以此处只初始化 P3
    P3M1 = P3MODE1;

    /* 初始化定时器 */
    TMOD = T0MODE1;                     // Timer0 工作在模式 1 ，16 位定时
    AUXR = AUXR&CLR_T0X12_C;            // Timer0 工作在 12 分频

    TCON = 0;                           /* 未使用外部中断，所以不用定义中断的触发方式 */

    TH0 = TIME1msH_C;
    TL0 = TIME1msH_C;
    TR0 = TRUE;

    /* 初始化中断 */
    IE = EnT0_C;                        // 此处只允许 Timer0 中断，
    IPH = NOIP_C;                       // 此处不设优先级，有各功能模块自身设定
    IP = NOIP_C;

    // 初始化起始状态 —— 亮
    Work_Display = LIGHT;               // 点亮
    gb_CurDisp = LIGHT;                 // 记忆当前状态，
    guiDisplayTime = LIGHT_TIME;        // 显示时间

    gb_1msFlag = FALSE;
    EA = TRUE;                          // 启动中断，开始正常工作

    while(1)
    {
        if(gb_1msFlag == TRUE)
        {
            gb_1msFlag = FALSE;

            guiDisplayTime--;               // 计时
            if(guiDisplayTime ==0)
            {
```

```
                if(gb_CurDisp == LIGHT)
                {
                    Work_Display = DARK;        // 熄灭
                    gb_CurDisp = DARK;
                    guiDisplayTime = DARK_TIME;
                }
                else
                {
                    Work_Display = LIGHT;       // 点亮
                    gb_CurDisp = LIGHT;
                    guiDisplayTime = LIGHT_TIME;
                }
            }
        }
    }                           // 不断循环，在所有嵌入式应用的主程序中,都有这样一个无限循环.
}

/*****************************************/
/*       定时器 0 中断服务                */
/* 说明:  1ms 中断一次,                   */
/*****************************************/

void  Timer0_Int(void) interrupt 1 using 1
{
    TH0 = TIME1msH_C;
    TL0 = TIME1msL_C;

    gb_1msFlag = TRUE;
}
```

由于篇幅原因，将程序中的宏定义部分略去，可下载 YM1_Prog-1A.C 看全部内容。

从这个程序可以看出，两个功能完全相同，只是计时任务主要交给了定时器 0 完成，它通过中断方式设置一个标志，相当于“ ALARM ”，主程序中只当有了这个标志时才作相应处理，其余时间处于等待状态，为做其它事提供了可能！

至此，我们已可以控制**主控工作指示**亮、暗了，但似乎单调了些，没有内容。

用一个灯如何表达“ hello World ”？

莫尔斯电码（Morse code）！

莫尔斯电码（Morse code)是美国人莫尔斯于 1844 年发明的， 由点（.)、划（-）两种符号组成。

1) 一点为一基本信号单位，一划的长度 = 3 点的长度。

2) 在一个字母或数字内，各点、划之间的间隔应为两点的长度。

3) 字母（数字）与字母（数字）之间的间隔为 7 点的长度。

电码表见附件一。

我们可以控制主控工作指示灯的亮、暗时间模拟“ 点 ”、“ 划 ”, 实现显示“ hello World ”。

因为程序稍长，不在文中列出，可下载附件中的 “ SbS_Prog-1B.C ” 参阅。程序是在前面程序的基础上修改的，主要构思基于如下变量的设计：

```
unsigned char gucDispBuf[MAX_CHAR_NUM+1];       // 显示缓冲区，存放要显示的字符，为 “ 0 ” 表示结束
unsigned char gucGetCharPtr;                    // 从显示缓冲区取字符指针

unsigned char gucCharDispBuf[MAX_GAP_NUM];
// 一个字符显示时亮、暗显示序列，存放显示基本时间单位的个数，为 “ 0 ” 表示结束
unsigned char gucGetDispNum;                    // 取亮、暗基本时间数指针

unsigned char gucDispNumCnt;                    // 基本时间数计数器
unsigned int  guiBaseTime;                      // 基本时间计数器，1ms 计数

unsigned char code DISP_CONTENT[MAX_CHAR_NUM+1] = {"HELLOWORLD"};
// 要显示的内容,因为莫尔斯电码没有定义空格
```

从上面变量的设置可以大概看出程序的构思：

使用一个 1ms 计数器形成莫尔斯电码的信号单位，构建一个数组存放一个字符的亮、暗控制序列，存放信号单位数。构建一个数组作为显示缓冲区，存放要显示的字符。

通过程序将字符翻译为符合莫尔斯电码的亮、暗控制串。注意：翻译程序中设计了一个常数数组 unsigned char code gucMorseCode[26][9]，读者可以比较这个处理与汇编程

序的差别，是否可读性大大增强了？

读者可以改变 DISP_CONTENT ，以显示自己想要的内容，还可以修改 BASE_TIME ，方便的改变发送的速度。

四、 结语

做上述程序主要目的是为了让初学者能够找到驾驭单片机的感觉，让他觉得这一切并不神秘，只要有一定的基础知识并且会 PC 机操作即可。

在此，我想谈一下对“ 单片机应用 ”学习目标的一旦看法：

“ 单片机应用 ”学习应该是一次实践，它本身没有任何属于自己的知识点，是建筑在“ 计算机原理 ”、“ C 语言编程 ”、“ 电子技术基础 ”等课程基础上的，是这些知识在具体产品（单片机）中的应用。而学习它的目的主要是学会运用上述基础知识看懂某个产品（单片机）的技术资料，并且通过使用之验证自己的理解，培养阅读、设计、实践的能力。

不知大家是否认同？

这是一个开始，之二将着眼于 MCU 与 PC 的通讯，人需要交流，机器也同样！

————————————

Hanker

2007 年 6 月 10 日星期日

资料：

圆梦小车的第一个程序

显示“ SOS ”的视频

附件一：

莫尔斯电码表：

字符	电码符号	字符	电码符号字符
A	●—	N	—●
B	—●●●	O	— — —
C	—●—●	P	●— —●
D	—●●	Q	— —●—
E	●	R	●—●
F	●●—●	S	●●●
G	— —●●	T	—
H	●●●●	U	●●—
I	●●	V	●●●—
J	●— — —	W	●— —
K	—●—	X	—●●—
L	●—●●	Y	—●— —
M	— —	Z	— —●●

数字	电码符号
1	●— — — —
2	●●— — —
3	●●●— —
4	●●●●— —
5	●●●●●
6	—●●●●
7	— —●●●
8	— — —●●
9	— — — —●
0	— — — — —

标点符号	电码符号
？	●●— —●●
/	—●●—●
( )	—●— —●—
—	●●●●—
。	●—●—●—