ADT-MC4140

四轴运动控制器

说
明
书

深圳众为兴数控技术有限公司
地址：深圳市南山区马家垅工业区 36 栋 5 楼　邮编：518052
电话：0755-26722719（20 线）　　传真：0755-26722718

本用户手册的所有部分，其著作财产权归属众为兴数控技术有限公司（以下简称众为兴）所有，未经众为兴许可，任何人不可任意地仿制，拷贝、誊抄或转译。本用户手册没有任何形式的担保，立场表达或其他暗示。若有任何因本用户手册或其所提到之产品的所有信息，所引起的直接或间接的资料流出，利益损失或事业终止，众为兴及其所属员工恕不担负任何责任。除此之外，本用户手册提到的产品规格及资料仅供参考，内容有可能会更新，恕不另行通知。

商标声明

用户手册中所涉及到的产品名称仅作识别之用，而这些名称可能是属于其它不同的商标或版权，在此声明如下：

※ INTEL，PENTIUM 是 INTEL 公司的商标。

※ WINDOWS，MS—DOS 是 MICROSOFT 公司产品标识。

※ ADT-MC4140 是众为兴公司的商标。

※ 其它未提到的标识，均属各注册公司所拥有。

众为兴数控技术有限公司

目　　录

第一章　简介

一、功能简述

ADT-MC4140 是一台高性能的现场四轴步进/伺服运动控制器。它内含 32 路光耦输入，4 轴编码器 AB 相脉冲输入，16 路光耦输出，4 路脉冲/方向信号输出，RS232 通讯模块，U 盘功能，可现场编程。采用单色图形液晶显示屏，128×64 点阵。

二、功能特点

1. 采用 SANSUNG 系列的 S3C44B0X 单片机（ARM7）主频 64MHz
2. 内含 2M Nor FLASH ROM（程序空间）
3. 内含 8M SDRAM
4. 内含 32M Nand FLASH ROM（数据存储，可模拟为 U 盘）
5. 支持 USB1.1 设备接口。
6. 4 路步进/伺服电机脉冲光耦隔离输出，最高频率 2MHz。
7. 脉冲输出的频率误差小于 0.1%
8. 脉冲输出可用单脉冲（脉冲+方向）或双脉冲（脉冲+脉冲）方式
9. 任意 2-4 轴直线插补
10. 直线加/减速
11. 32 位计数（逻辑位置和实际位置）
12. 运动中可以实时读出逻辑位置、实际位置、驱动速度
13. 32 路光耦隔离输入（含 4 路 AB 相输入）。
14. 4 轴编码器 AB 相脉冲光耦隔离输入(可做 8 路开关量输入用)。
15. 16 路集电极开路光耦隔离输出，1~8 点共用一公共端，另 8 点各有独立的公共端。
16. RS232（±15KV 静电保护)。
17. 适应性极强的开关电源 85~265VAC 45~65Hz。
18. 可在线编程，仅需一根串口线和一根 USB 线，无须编程器。
19. 开发工具采用 UC/OS-II 操作系统。
20. 所有硬件部分已编成函数库，无需任何硬件知识即可开发

三、电气规格

开关量输入：

通道：32，全部光耦隔离。
输入电压：5-24V
高电平>4.5V
低电平<1.0V
隔离电压：2500V DC

计数输入：

通道：4AB 相编码输入　　，全部光耦隔离。
（与 8 开关量输入复用）
最高计数频率：2MHz
输入电压：5-24V
高电平>4.5V
低电平<1.0V
隔离电压：2500V DC

脉冲输出：

通道：4 脉冲，4 方向，全部光耦隔离。
最高脉冲频率：2MHz
输出类型：5V 差动输出
输出方式：脉冲+方向 或 脉冲+脉冲

开关量输出：

输出通道：16 全部光耦隔离。其中 8 路输出共一公共端，另 8 输出各自独立。
输出类型：NPN 集电极开路 5-24VDC，最大电流 100mA

RS-232 波特率(bps)：

1200、2400、4800、9600、19200、38400、57600、115200

四、应用环境

电源要求：　85-265V AC 45-65Hz
功　　耗：　＜4W
工作温度：　0℃—60℃

储存温度： -20℃—80℃
工作湿度： 20%—95%
储存湿度： 0%—95%

五、适用范围

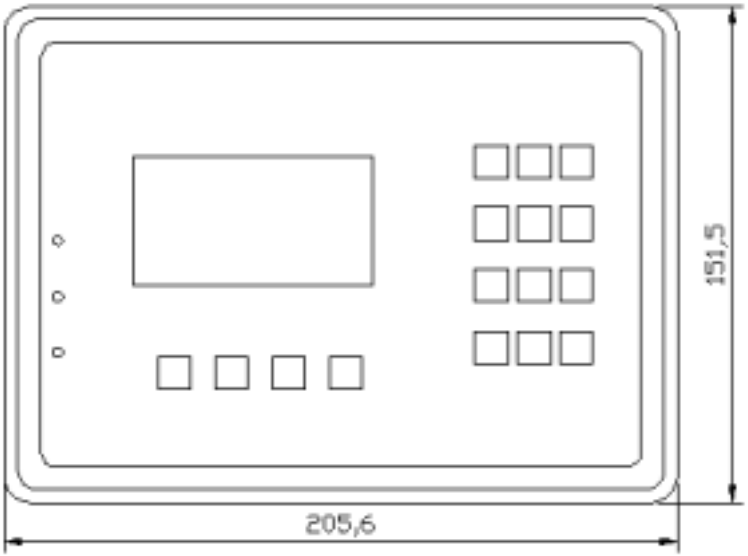
- 可编程 PLC
- 封切机控制器
- 液晶显示
- 1~4 步进/伺服电机控制

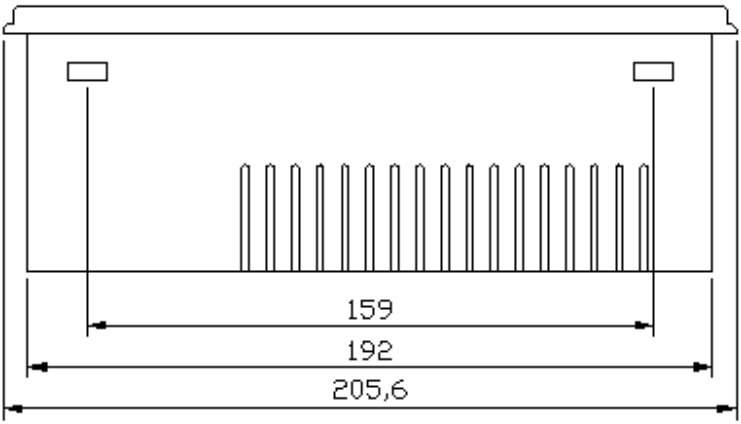
第二章　型号定义

ADT-MC4140 型号定义如下：

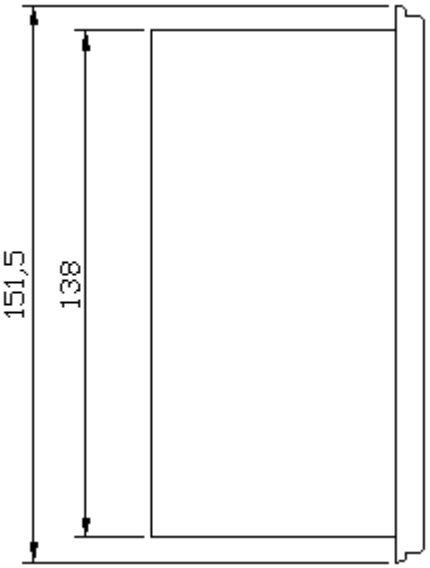
型号	描述
ADT-MC4140AX	16DI 输入
ADT-MC4140AXY	32DI 输入，4 路编码器输入为 5V
ADT-MC4140AXYU	32DI 输入，4 路编码器输入为 12-24V

第三章　外形尺寸

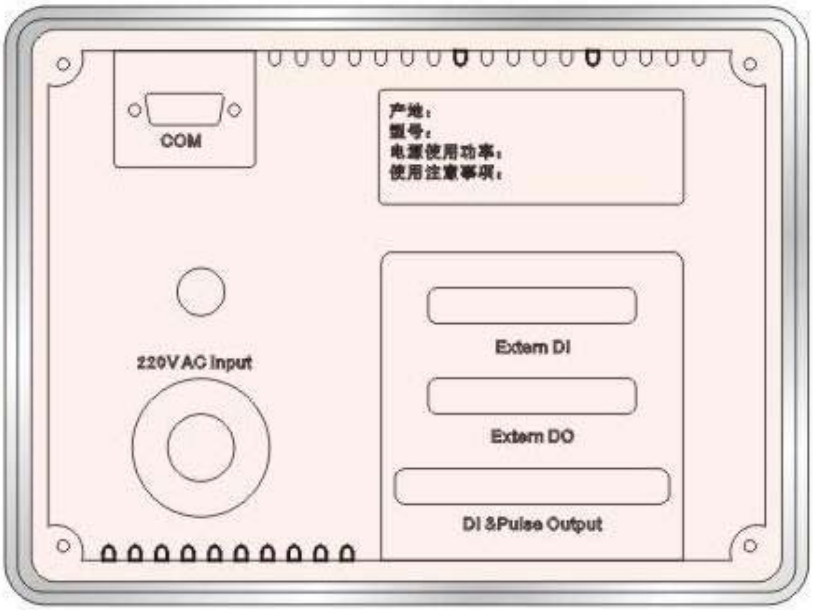


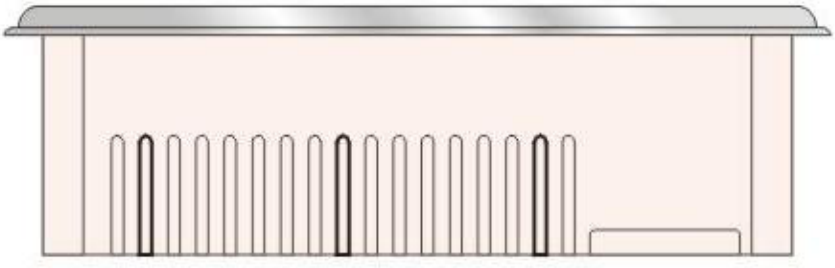



11
5
74,5
18,5
8


151,5
138


USR
Run
Stop
1
2
3
4
5
6
7
8
9
0
ESC
ADT-MC4140


COM
产地：
型号：
电源使用功率：
使用注意事项：
220VAC Input
Extern DI
Extern DO
DI &Pulse Output



说明：

① 正面图上左边是三个发光二极管指示灯。

② 背面图上右下三个长插座从上向下依次是 25 针插座(2 轴 AB 相解码输入、14 路扩展输入)、25 针插座(16 路输出插座)、37 针插座(2 路脉冲/方向输出、16 路基本输入、4 路输出)。

③ 背面图上左边三个插座分别是通信口、键盘接口、电源插座。

④ 侧面图上上面的两个长方形孔是控制器的安装孔。

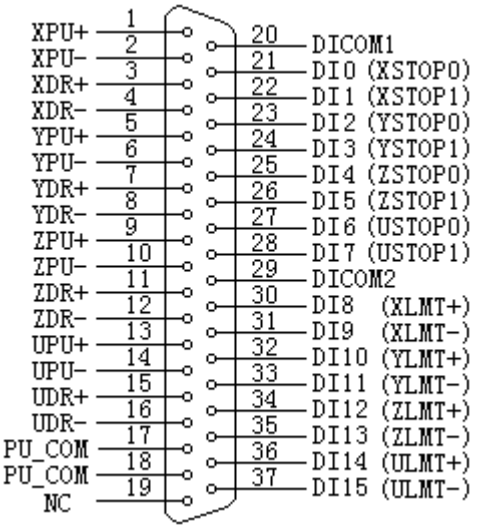
⑤ 侧面有一个 USB 设备接口

特别声明：用户不可自己拆开控制器的机壳，否则该控制器将不在保修范围之内。

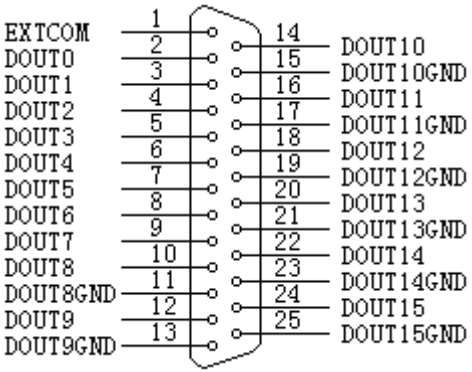
第三章 电气连接

一、接线端子图:

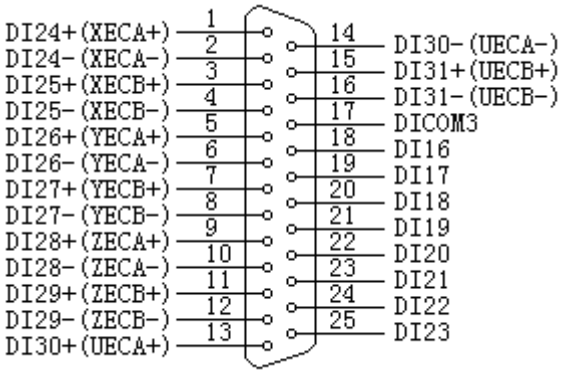
37 芯信号插座---4 路脉冲、4 路方向和 16 路基本数字输入



25 芯信号插座---16 路基本数字输出



25 芯信号插座---16 路扩展数字输入（含 4 轴 AB 相解码输入）



二、端子说明：

37 芯信号插座---4 路脉冲、4 路方向、16 路基本数字输入

本控制器的 16 路数字输入分为两个通道，DI0～DI7 共用第一通道，DI8～DI15 共用第二通道。两个通道均是共阳极接法，第一通道的输入公共端为 DICOM1，第二通道的输入公共端为 DICOM2。这两个公共端须接外部电源正极。

详细使用见编程说明。

PU_COM 是特别为了适应步进或伺服驱动器控制信号不是差动信号的情况而设的，此时驱动器是采用共阳极或共阴极接法。如果驱动器控制信号是采用差动接法，则 PU_COM 就不须接线。**须特别注意，PU_COM 除了为应付驱动器脉冲的非差动接法外，绝对不能做别的用途，否则，将可能造成控制器的损坏。**

25 芯信号插座---16 路基本数字输出

16 路集电极开路输出数字信号中的 8 路共一公共端 EXTCOM，另 8 路各自独立。

25 芯信号插座---16 路扩展数字输入（含 4 轴 AB 相解码输入）

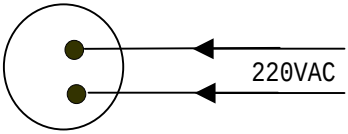
8 路扩展集电极开路数字输入共用一通道，其公共端是 DICOM3，须接外部电源正极。

其余 8 路为独立输入，其中 DI24、DI25 为 X 轴 AB 相解码输入信号，DI26、DI27 为 Y 轴 AB 相解码输入信号，DI28、DI29 为

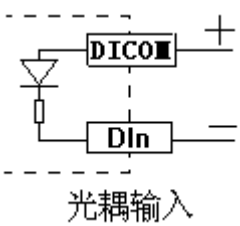
Z 轴 AB 相解码输入信号，DI30、DI31 为 U 轴 AB 相解码输入信号。

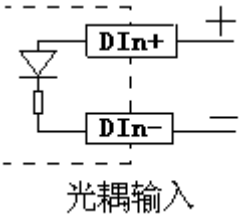
图示 AB 相解码输入信号是差动输入接线方式；如果是共阳极接法，则须将 A 相的正端和 B 相的正端连接在一起；如果是共阴极接法，则须将 A 相的负端和 B 相的负端连接在一起。

三、电源的接线方式



四、输入信号的接线方式





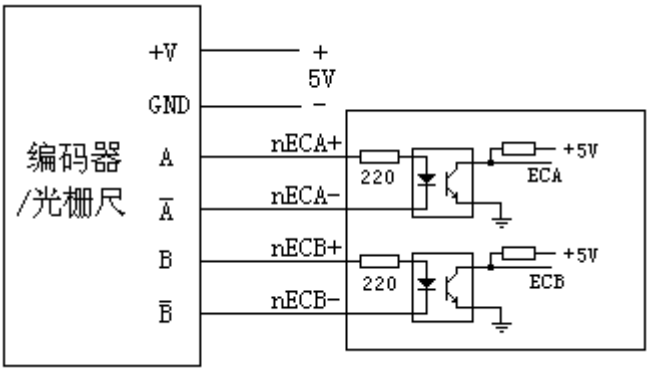
DICOM 端子接外部电源的正端，输入信号接相应端子。
其中 DI0-DI7 的公共端为 DICOM1，DI8-DI15 的公共端为 DICOM2，DI16-DI23 的公共端为 DICOM3。

五、AB 相解码输入信号接线方式

4 路 AB 相解码输入分为差动接法、共阳极接法两种，由编码器的类型决定。

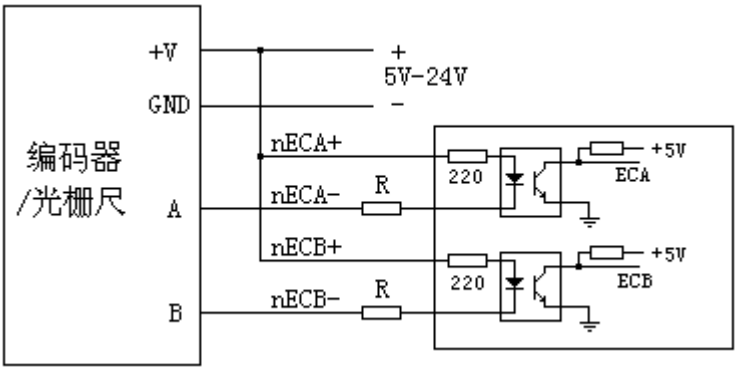
编码器输出一般有集电极开路输出、 互补输出、电压输出和长线驱动器输出。其中集电极开路输出、互补输出、电压输出可采用共阳极接法，长线驱动器输出采用差动接法。

差分接法如下图：



5V 电源由外部提供。

共阳极接法如下图：



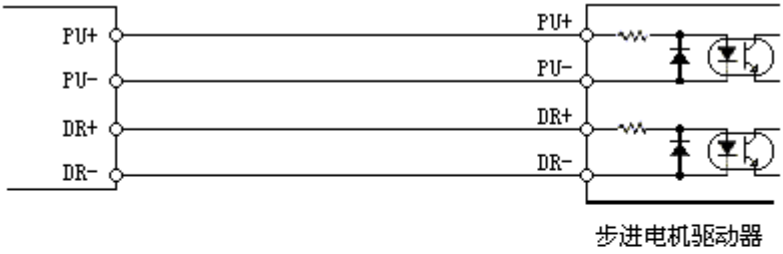
电源电压由编码器决定，使用 5V 电源时，电阻 R 可不用，使用 12V 电源时,R 可用 1K-2K 的电阻,使用 24V 电源时,R 可用 2K-5K 的电阻。

建议采用长线驱动器输出的编码器，因为采用差动方式，在线路较长时抗干扰性要好一些。

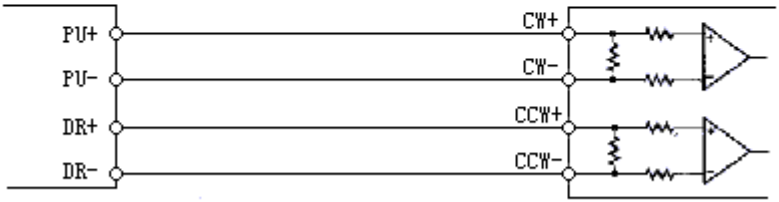
六、脉冲输出信号的接线方式

1． 差动方式：

适用于脉冲和方向独立输入的步进驱动器和大多数伺服驱动器。建议采用此方式，可获得较好的抗干扰性。



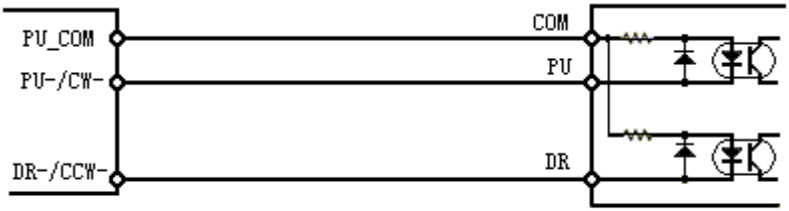
步进电机驱动器



伺服电机驱动器

2． 单端方式：

适用于早期一些脉冲和方向的阳极连在一起的步进驱动器。

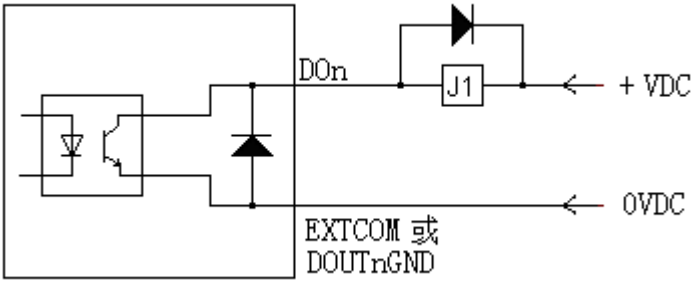


步进电机驱动器

注意：不适用某些脉冲和方向的阴极连在一起的步进驱动器。

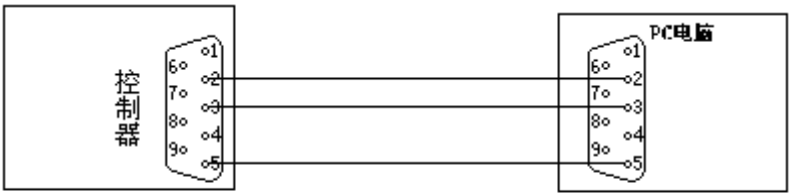
七、输出信号的接线方式

ADT-MC4140 的输出为集电极开路输出，如外接负载为感性负载，如继电器等，必须接续流二极管，如下图所示：



电源电压建议 < 24V，最好不超过 30V，正负极切不可接反，负载也不可短路，否则将损坏模块！

八、通讯信号的接线方式



RS-232 通讯方式

第五章　控制器软件编程

ADT-MC4140 采用 ARM7 系列单片机，采用 C 语言编程，开发工具采用 ARM 常用的 ADS1.2（ARM Developer Suite），有关 ADS 的详细使用，请参见相关书籍。

所有的库函数已包含在 MC4140LIB.A 文件中，可分为显示类，运动控制类，文件系统类，操作系统类，其他类。

一、显示编程函数说明

屏幕显示采用单色液晶屏，分辨率为 128X64，目前仅提供一些简单的画点画线函数，以及 6X8、8X8 的 ASCII 字符和 12X12、16X16 点阵的汉字显示。

显示函数使用的头文件分别为 iodisp.h。

1．GUI_Init()

描述：初始化显示屏

原型：void GUI_Init(void);

参数：无

返回值：无

2．GUI_Redraw()

描述：显示屏刷新

原型：void GUI_Redraw(void);

参数：无

返回值：无

注意：下面所有函数执行后，显示不会改变，只有在执行本函数后，显示才会改变。

3．DrawImage()

描述：显示图片

原型：void DrawImage(INT16U x, INT16U y,
INT16U width, INT16U height,
const INT8U* pData);

参数：

x,y：　　图片起始位置

width： 图片宽度
height： 图片高度
pData： 图片数据指针

返回值：无
图片数据可用 zimo 软件生成。

4．ClearArea()
描述：在屏幕上清除区域
原型：void ClearArea(INT16U x, INT16U y,
INT16U width, INT16U height,INT8U color);
参数：
x,y：起始位置
width：区域宽度
height：区域高度
color：填充颜色
返回值：无

5．FillArea()
描述：在区域中填色
原型：void FillArea(INT16U x, INT16U y,
INT16U width, INT16U height, INT8U color);
参数：
x,y：起始位置
width：区域宽度
height：区域高度
color：填充颜色
返回值：无

6．InvertArea()
描述：在屏幕上反转区域图象
原型：void InvertArea(INT16U x, INT16U y,
INT16U width, INT16U height);
参数：
x,y： 起始位置
width： 区域宽度
height： 区域高度
color： 填充颜色

返回值：无

7．DrawPoint()

描述：在屏幕上画点

原型：void DrawPoint(INT16U x, INT16U y,
INT8U color, INT8U linewidth);

参数：

x,y： 坐标
color： 点颜色
linewidth： 点宽度

返回值：无

8．**DrawLine()**

描述：在屏幕上画线段

原型：void DrawLine(INT16U x, INT16U y,
INT16U x1, INT16U y1,
INT8U color, INT8U linewidth);

参数：

x,y： 第一点坐标
x1,y1： 第二点坐标
color： 线段颜色
linewidth： 线段宽度

返回值：无

9．DrawRect()

描述：在屏幕上画矩形

原型：void DrawRect(INT16U x,INT16U y,
INT16U width,INT16U high,
INT8U color);

参数：

x,y： 坐标
width： 矩形宽度
height： 矩形高度
color： 线颜色

返回值：无

10．DrawString12X12()

描述：显示 12X12 点阵的汉字及字符

原型：void DrawString12X12(INT16U x,INT16U y,INT8U *str);
参数：
x,y： 坐标
str： 要显示的字符串
返回值：无

11．DrawString16X16()
描述：显示 16X16 点阵的汉字及字符
原型：void DrawString16X16(INT16U x,INT16U y,INT8U *str);
参数：
x,y： 坐标
str： 要显示的字符串
返回值：无

12．DrawString6X8()
描述：显示 6X8 点阵的字符
原型：void DrawString6X8(INT16U x,INT16U y,INT8U *str);
参数：
x,y： 坐标
str： 要显示的字符串
返回值：无
注意：不能显示汉字

13．DrawString8X8()
描述：显示 8X8 点阵的字符
原型：void DrawString8X8(INT16U x,INT16U y,INT8U *str);
参数：
x,y： 坐标
str： 要显示的字符串
返回值：无
注意：不能显示汉字

二、运动控制编程

运动控制部分包含模式设置，位置管理，单轴驱动，2-4 轴直线插补以及数字输入输出。

运动函数使用的头文件为 motion.h。

1．基本参数设置类

1.1 motion_init()

描述：初始化运动控制
原型：void motion_init(void);
参数：无
返回值：无

1.2 set_stop0_mode()

描述：设置stop0信号的有效/无效和逻辑电平
原型：int set_stop0_mode(int axis, int value,int logic);
参数：

```
axis        轴号（1-4）
value   0：无效        1：有效
logic   0：低电平停止   1：高电平停止
```

返回值： 0：正确 1：错误
初始化时状态为：信号无效，低电平停止

注意：停止方式是立即停止。STOP1 信号也是一样。

1.3 set_stop1_mode()

描述：设置stop1信号的有效/无效和逻辑电平
原型：int set_stop1_mode(int axis, int value,int logic);
参数：

```
axis        轴号（1-4）
value   0：无效        1：有效
logic   0：低电平停止   1：高电平停止
```

返回值： 0：正确 1：错误
初始化时状态为：信号无效，低电平停止

1.4 set_pulse_mode()

描述：设置输出脉冲的工作方式
原型：int set_pulse_mode(int axis, int value,int logic,int dir_logic);

参数：

```
axis        轴号（1-4）
```

value　0：脉冲+脉冲方式
　　1：脉冲+方向方式

logic　　0： 正逻辑脉冲　　1： 负逻辑脉冲

dir-logic　0：方向输出信号正逻辑
　　1：方向输出信号负逻辑

dir_logic	正方向脉冲输出时	负方向脉冲输出时
0	Low	Hi
1	Hi	Low

返回值：　0：正确　　1：错误

初始化时状态为：脉冲+方向方式，正逻辑脉冲，方向输出信号正逻辑

1.5 set_limit_mode()

描述：设定正/负方向限位输入nLMT信号的模式设定

原型：int set_limit_mode(int axis,　int value1,int value2,int logic);

参数：

axis　　轴号（1-4）
value1　0：正限位有效
　　1：正限位无效
value2　0：负限位有效
　　1：负限位无效
logic　　0：低电平有效
　　1：高电平有效

返回值：　0：正确　　1：错误

初始化时状态为：正限位有效，负限位有效，低电平有效

2．驱动状态检查类

2.1 get_status()

描述：获取各轴的驱动状态
原型：int get_status(int axis,int *value)
参数：
axis 轴号（1-4）
value 驱动状态的指针
0：驱动结束
非 0：正在驱动
返回值： 0：正确 1：错误

2.2 get_inp_status()
描述：获取插补的驱动状态
原型：int get_inp_status(int *value)
参数：
value 插补状态的指针
0：插补结束 1：正在插补
返回值： 0：正确 1：错误

3．运动参数设定类
以下参数在初始化后值不确定，使用前必须设定

3.1 set_range()
描述：范围设定
原型：int set_range(int axis,long value);
参数：
axis 轴号
value R值 范围（500-2）
返回值： 0：正确 1：错误

注意：范围是决定速度、加/减速度、加/减速度的变化率的倍率参数，假设把范围数值作为R 倍率是下述的算式

倍率＝500／R

因为驱动速度、初始速度、加/减速度等参数的设定范围在1~8000。若需要设定为8000以上的数值的话，必须提高倍率。提高倍率后可以高速驱动，但是速度分辨率变粗。请在使用的速度范围内设定最小的数值。比如如果需要40KPPS 的速度，在速度范围内1~8000 倍率中最好是5， 即设定R 为100。

R值范围是500-2，相应的倍率为1-250。

在驱动中请不要变更范围（R） 否则速度会变乱。

如：

set_range(0,1,50)；

为设定第一块卡的X轴的倍率为500/50=10倍

set_range(0,3,2)；

为设定第一块卡的Z轴的倍率为500/2=250倍

3.2 set_acc()

描述：加速度设定

原型：int set_acc(int axis,long value);

参数：

axis 轴号

value A值（1-8000）

返回值： 0：正确 1：错误

它是作为直线加减速驱动中直线加速及减速的参数，在S曲线加/减速驱动中，加速度及减速度线性从0 增加至加速度的设定值。

加速度设定值为A 加速度是下述算式

加速度（PPS/SEC）=A*125*倍率

即

加速度（PPS/SEC）=A*125*（500/R）

加速度设定值A 的设定范围是1~8,000。

如：

set_range(0,1,5);

set_acc(0,1,100);

则加速度为：

100*125*（500/5）=1250000 PPS/SEC

3.3 set_startv()

描述：初始速度设定

原型：int set_startv(int axis,long value);

参数：

axis 轴号

value SV值（1-8000）

返回值： 0：正确 1：错误

它是加/ 减速驱动的加速开始时的速度和减速结束时的速度，初始速度设定数值为SV 的话，初始速度是下述算式

初始速度（PPS）=SV*倍率
即 初始速度（PPS）=SV*（500/R）

3.4 set_speed()

描述：驱动速度设定
原型：int set_speed(int axis,long value);
参数：

	axis	轴号
	value	V值（1-8000）

返回值： 0：正确 1：错误

它是加/减速驱动中达到定速区域的速度。定速驱动从该速度开始运行。把这个驱动速度设定在初始速度以下的话，不运行加/减速驱动，开始就运行定速驱动

驱动速度（PPS）=V*倍率
即 驱动速度（PPS）=V*（500/R）

3.5 set_command_pos()

描述：设定逻辑位置计数器的数值
原型：int set_command_pos(int axis,long value);
参数：

	axis	轴号
	value	范围值（-2147483648~+2147483647）

返回值： 0：正确 1：错误
逻辑位置计数器任何时候都能写、任何时候都能读

3.8 set_actual_pos()

描述：设定实际位置计数器的数值
原型：int set_actual_pos(int axis,long value);
参数：

	axis	轴号
	value	范围值（-2147483648~+2147483647）

返回值： 0：正确 1：错误
实际位置计数器任何时候都能写、任何时候都能读

4．运动参数检查类

以下函数在任何时候均可调用

4.1 get_command_pos()

描述：获取各轴的逻辑位置

原型：int get_command_pos(int axis,long *pos)

参数：

axis	轴号
pos	逻辑位置值的指针

返回值： 0：正确 1：错误

此函数可随时得到轴的逻辑位置，在电机未失步的情况下可代表轴的当前位置。

4.2 get_actual_pos()

描述：获取各轴的实际位置（即编码器反馈输入值）

原型：int get_actual_pos(int axis,long *pos)

参数：

axis	轴号
pos	实际位置值的指针

返回值： 0：正确 1：错误

此函数可随时得到轴的实际位置，在电机有失步的情况下也能知道轴的当前位置。

4.3 get_speed()

描述：获取各轴的当前驱动速度

原型：int get_speed(int axis,long *speed)

参数：

axis	轴号
speed	当前驱动速度的指针

返回值： 0：正确 1：错误

数据的单位和驱动设定数值V一样。

此函数可随时得到轴的驱动速度。

5．驱动类

5.1 pmove()

描述：定量驱动

原型：int pmove(int axis,long pulse)

参数：

axis	轴号
pulse	输出的脉冲数
	>0：正方向移动

<0：负方向移动
范围（-268435455~+268435455）
返回值：　0：正确　　1：错误
写入驱动命令之前一定要正确地设定速度曲线所需的参数

5.3　dec_stop()
描述：驱动减速停止
原型：int dec_stop(int axis)
参数：
axis　　轴号
返回值：　0：正确　　1：错误
在驱动脉冲输出过程中，此命令作出减速停止，驱动速度比初始速度慢的时候也可以用本命令立即停止。

5.4　sudden_stop()
描述：驱动立即停止
原型：int sudden_stop(int axis)
参数：
axis　　轴号
返回值：　0：正确　　1：错误
立即停止正在驱动中的脉冲输出，在加/减速驱动中也立即停止。

5.5　inp_move2()
描述：两轴直线插补
原型：int inp_move2(int axis1,int axis2,long pulse1,long pulse2)
参数：
axis1,axis2 参与插补的轴号
1：X　2：Y　3：Z　4：U
pulse1,pulse2 移动的相对距离
返回值：　0：正确　　1：错误
注意：插补的速度以最小轴速度为基准。

5.8　inp_move3()
描述：三轴直线插补
原型：int inp_move3(int axis1,int axis2,int axis3,
long pulse1,long pulse2,long pulse3)

参数：

axis1,axis2,axis3 参与插补的轴号

1：X 2：Y 3：Z 4：U

pulse1,pulse2,pulse3

各轴移动的相对距离

返回值： 0：正确 1：错误

注意：三轴插补的速度以最小轴速度为基准。

5.8 inp_move4()

描述：三轴直线插补

原型：int inp_move4(long pulse1,long pulse2,long pulse3,long pulse4)

参数：

pulse1,pulse2,pulse3,pulse4

各轴移动的相对距离

返回值： 0：正确 1：错误

注意：四轴插补的速度以X轴速度为基准。

6．开关量输入输出类

6.1 read_bit()

描述：读单个输入点

原型：int read_bit(int number)

参数：

number 输入点（0-31）

返回值： 0：低电平 1：高电平 -1：错误

输入点对应相应的输入号。

6.2 write_bit()

描述：输出单点

原型：int write_bit(int number,int value)

参数：

number 输出点（0-31）

value 0：低 1：高

返回值： 0：正确 1：错误

输出数对应相应的输出号。

二、文件系统编程

文件系统采用标准的 FAT 文件系统，提供了基本的文件和目录的操作，支持 uC/OS-II 系统，并且可通过模拟为一个 U 盘，用电脑进行文件的操作。

文件系统函数使用的头文件是 fs_api.h

以下是函数的详细说明：

1. FS_Exit()

描述：停止文件系统

原型：void FS_Exit(void)；

返回值：void

注意：FS_Exit()移去所有 OS 使用的资源，在调用本函数后，除了 FS_Init()，不能调用其他函数。

2. FS_Init()

描述：启动文件系统

原型：void FS_Init(void)；

返回值：void

注意：FS_Init()初始化文件系统，建立 OS 所需的资源。

示例：

```
#include “fs_api.h”
void main(void)
{
    FS_Init();
   /*
    access file system
    */
   FS_Exit();
}
```

3. FS_FClose()

描述：关闭文件

原型：void FS_Fclose(FS_FILE *fp);

返回值：void

示例：

```
void foo(void)
{
    FS_FILE *fp;
```

```
        fp=FS_FOpen("test.txt","r");
        if(fp !=0)
        {
        /*
              access file
        */
        FS_FClose(fp);
        }
    }
```

4. FS_Fopen()

描述：打开文件

原型：FS_FILE *FS_FOpen(const char *name, const char *mode);

name 全文件名

mode 文件打开方式

返回值：如果文件打开正常，返回 FS_FILE 结构的地址，否则返回 0。

注意：全文件名由路径名和文件名组成

路径名必须是已经存在的目录，FS_FOpen 函数不会建立目录，路径名必须以'\'开头和结尾，如果省略路径名，则操作根目录上的文件，文件名按 FAT 系统的 8.3 格式，不支持长文件名。

mode 参数含义见下表，与 ANSI C 中的 fopen 函数兼容。

模式	功能
R	以读方式打开文本文件
W	以写方式建立文本文件或缩短文件长度到 0
A	以写方式建立或打开文本文件，以添加
Rb	以读方式打开二进制文件
wb	以写方式建立二进制文件或缩短文件长度到 0
Ab	以写方式建立或打开二进制文件，以添加
r+	
W+	
a+	
r+b or rb+	
w+b or wb+	
a+b or ab+	

本函数不区分文本文件和二进制文件，都是按二进制文件处理

示例：

```
FS_FILE *myfile;

void foo1(void){
myfile=FS_FOpen(“test.txt”,”r”);
}

void foo2(void){
myfile=FS_FOpen(“test.txt”,”w”);
}

void foo3(void){
myfile=FS_FOpen(“\\adt\\test.txt”,”r”);
}

void foo4(void){
myfile=FS_FOpen(“\\adt\\test.txt”,”w”);
```

FS_FRead()

描述：读文件

原型：FS_size_t FS_FRead(void *ptr, FS_size_t size, FS_size_t n, FS_FILE *fp);

ptr	数据指针
size	传送的变量长度
n	传送的变量数
fp	FS_FILE 文件结构的指针

返回值：传送的变量数

示例：

```
Char buffer[100];
void foo(void) {
    FS_FILE *myfile;
    myfile = FS_FOpen(“test.txt”,”r”);
    if (myfile!=0) {
        do {
            i = FS_FRead(buffer,1,sizeof(buffer)-1,myfile);
            buffer[i]=0;
            if (i) {
            printf("%s",buffer);
            }
        } while (i);
        FS_FClose(myfile);
```

```
    }
}
```

5. FS_FWrite()

描述：写文件

原型：FS_size_t FS_FWrite(void *ptr, FS_size_t size, FS_size_t n, FS_FILE *fp);

ptr	数据指针
size	传送的变量长度
n	传送的变量数
fp	FS_FILE 文件结构的指针

返回值：传送的变量数

示例：

```
const char welcome[]="hello world\n";
void foo(void) {
    FS_FILE *myfile;
    myfile = FS_FOpen("test.txt","w");
    if (myfile!=0) {
    FS_FWrite(welcome,1,strlen(welcome),myfile);
    FS_FClose(myfile);
}
}
```

6. FS_ FSeek()

描述：设置文件读写位置

原型：Int FS_FSeek(FS_FILE *fp, FS_i32 offset, int whence);

fp	FS_FILE 文件结构的指针
offset	位置指针偏移量
whence	位置指针模式

返回值：正确返回 0　错误返回 –1

注意： whence 可用 FS_SEEK_SET、FS_SEEK_CUR 和 FS_SEEK_END，详细解释可参见标准 fseek()函数。

示例：

```
    const char welcome[]="some text will be overwritten \n";
```

```
void foo(void) {
    FS_FILE *myfile;
    myfile = FS_FOpen(“test.txt”,”w”);
    if (myfile!=0) {
        FS_FWrite(welcome,1,strlen(welcome),myfile);
        FS_FSeek(myfile,- 4,FS_SEEK_CUR);
        FS_FWrite(welcome,1,strlen(welcome),myfile);
        FS_FClose(myfile);
    }
}
```

7. FS_ FTell()

描述：返回文件读写位置

原型：FS_i32 FS_FTell(FS_FILE *fp);

fp FS_FILE 文件结构的指针

返回值：文件读写位置，返回-1 表示错误

示例：

```
const char welcome[]="hello world\n";
void foo(void) {
    FS_FILE *myfile;
    FS_i32 pos;
    myfile = FS_FOpen(“test.txt”,”w”);
    if (myfile !=0) {
        FS_FWrite(welcome,1,strlen(welcome),myfile);
        Pos = FS_FTell(myfile);
        FS_FClose(myfile);
    }
}
```

8. FS_ ClearErr()

描述：清除错误状态

原型：void FS_ClearErr(FS_FILE *fp);

fp FS_FILE 文件结构的指针

返回值：void

示例：

```
void foo(void) {
    FS_FILE *myfile;
```

```
        FS_i16 err;
        myfile = FS_FOpen("test.txt","r");
        if (myfile!=0) {
            err = FS_FError(myfile);
            if (err!=FS_ERR_OK) {
                FS_ClearErr(myfile);
            }
            FS_FClose(myfile);
        }
}
```

9. FS_ FError()

描述：返回错误代码

原型：FS_i16 FS_FError(FS_FILE *fp);

fp FS_FILE 文件结构的指针

返回值：错误代码

错误代码表：

模式	含义
FS_ERR_OK	无错误
FS_ERR_EOF	已到文件尾
FS_ERR_DISKFULL	磁盘满
FS_ERR_INVALIDPAR	参数非法
FS_ERR_WRITEONLY	读文件（以写方式打开的文件）
FS_ERR_READONLY	写文件（以读方式打开的文件）
FS_ERR_READERROR	读文件出错
FS_ERR_WRITEERROR	写文件出错
FS_ERR_DISKCHANGED	磁盘更换，虽然文件一直打开
FS_ERR_CLOSE	关闭文件出错

示例：

```
void foo(void) {
    FS_FILE *myfile;
    FS_i16 err;
    myfile = FS_FOpen("test.txt","r");
    if (myfile!=0) {
            err = FS_FError(myfile);
            FS_FClose(myfile);
        }
```

```
}
```

10. FS_Remove()

描述：删除文件

原型：int FS_Remove(const char * name);

name 全文件名（可参考 FS_FOpen 函数）

返回值：0 表示文件已删除，-1 表示有错误

示例：

```
void foo(void) {
    FS_Remove(“test.txt”);
}
```

11. FS_CloseDir()

描述：关闭一个目录

原型：int FS_CloseDir(FS_DIR *dirp);

dirp 目录结构指针

返回值：0 正常 -1 错误

示例：

```
void foo(void) {
    FS_DIR *dirp;
    struct FS_DIRENT *direntp;
    dirp = FS_OpenDir("");  /* Open root directory of the default device */
    if (dirp) {
        do {
            direntp = FS_ReadDir(dirp);
            if (direntp) {
                sprintf(mybuffer,"%s\n",direntp->d_name);
                _log(mybuffer);
            }
        } while (direntp);
        FS_CloseDir(dirp);
    }
    else {
        _error("Unable to open directory \n");
    }
```

12. FS_MkDir()

描述：建立目录
原型：int FS_MkDir(const char *dirname);
dirname　　全路径名
返回值：0 正确　-1 错误

示例：

```
void foo1(void) {
    /* create mydir in directory test – default driver on default device
*/
    err = FS_MkDir("\\test\\ mydir");
    }
```

13. FS_OpenDir()

描述：打开目录
原型：FS_DIR *FS_OpenDir(const char *dirname);
dirname　　全路径名
返回值：返回 FS_DIR 结构指针，如果出错返回 0

示例：

```
FS_DIR *dirp;
    void foo1(void) {
        /* open directory test – default driver on default device */
        dirp = FS_OpenDir(“test”);
    }

    void foo2(void) {
        /* open root directory – default device driver on default
device */
        dirp = FS_OpenDir(“”);
    }
```

14. FS_ReadDir()

描述：读目录
原型：struct FS_DIRENT *FS_ReadDir(FS_DIR *dirp);
　　dirp　　已打开的目录结构 FS_DIR 指针
返回值：返回一个目录项的指针，如果已无目录项或有错误，返回 0

示例：参见 FS_CloseDir()

15. FS_RewindDir()

描述：设置目录项指针到第一个目录项

原型：void FS_RewindDir(FS_DIR *dirp);

dirp　　已打开的目录结构 FS_DIR 指针

返回值：void

示例：

```
void foo(void) {
    FS_DIR *dirp;
    struct FS_DIRENT *direntp;
    dirp = FS_OpenDir("");
    if (dirp) {
        /* display directory */
        do {
            direntp = FS_ReadDir(dirp);
            if (direntp) {
                sprintf(mybuffer,"%s\n",direntp->d_name);
                _log(mybuffer);
            }
        } while (direntp);
        /* rewind to 1st entry */
        FS_RewindDir(dirp);
        /* display directory    again */
        do {
            direntp = FS_ReadDir(dirp);
            if (direntp) {
                sprintf(mybuffer,"%s\n",direntp->d_name);
                _log(mybuffer);
            }
        } while (direntp);
        FS_CloseDir(dirp);
    }
        else  {
            _error("Unable to open directory \n");
        }
```

16. FS_RmDir()

描述：删除目录

原型：int FS_RmDir(const char *dirname);

dirname　　全路径名
返回值：0 正确　-1 错误
注意：不能删除非空目录

示例：

```
void foo1(void) {
    /* remove mydir in directory test – default driver on default device */
    err = FS_RmDir("\\test\\ mydir");
}

void foo2(void) {
    /* remove directory mydir – default driver on default device */
    err = FS_RmDir("ram:\\mydir");
    }
```

三、uC/OS-II 编程

对于 ARM7 系列的单片机，可以不采用任何操作系统，就象普通的 8 位单片机一样。但采用操作系统后，可更加方便的编程，本控制器采用 uC/OS-II 操作系统，可实现实时多任务，有关 uC/OS-II 的详细编程，可参考相关书籍。

下面简单的介绍一下几个常用的函数。

1．建立任务

OSTaskCreate( )

描述：建立一个新任务。任务的建立可以在多任务环境启动之前，也可以在正在运行的任务中建立。中断处理程序中不能建立任务。

原型：

```
INT8U OSTaskCreate(void(*task)(void *pd),
    void *pdata, OS_STK *ptos, INT8U prio);
```

参数：

task 是指向任务代码的指针。

pdata 指向一个数据结构，该结构用来在建立任务时向任务传递参数。

ptos 为指向任务堆栈栈顶的指针。任务堆栈用来保存局部变量，函数参数，返回地址以及任务被中断时的 CPU 寄存器内容。任务堆栈的大小决定于任务的需要及预计的中断嵌套层

数。

prio 为任务的优先级。每个任务必须有一个唯一的优先级作为标识。数字越小，优先级越高。

返回值：

OSTaskCreate（）的返回值为下述之一：

- OS_NO_ERR：函数调用成功。
- OS_PRIO_EXIST：具有该优先级的任务已经存在。
- OS_PRIO_INVALID ： 参 数 指 定 的 优 先 级 大 于 OS_LOWEST_PRIO。
- OS_NO_MORE_TCB：系统中没有 OS_TCB 可以分配给任务了。

注意/警告：

任务堆栈必须声明为 OS_STK 类型。

2．任务延时

OSTimeDly( )

描述：将一个任务延时若干个时钟节拍。

原型：void OSTimeDly (INT16U ticks);

参数：ticks 为要延时的时钟节拍数。

如果延时时间大于 0，系统将立即进行任务调度。延时时间的长度可从 0 到 65535 个时钟节拍。延时时间 0 表示不进行延时，函数将立即返回调用者。延时的具体时间依赖于系统每秒钟有多少时钟节拍(由文件 SO_CFG.H 中的常量 OS_TICKS_PER_SEC 设定)。控制器中 1000 个 ticks 为一秒，也即一个 ticks 为一个毫秒。

返回值：无

有关 uC/OS-II 的其他函数请参见有关书籍。

uC/OS-II 是一种实时多任务操作系统，如果要详细了解，需要花费大量时间，而如果我们只是需要做一个不复杂的小系统，只需要利用多任务来实现编程的方便，就没必要了解 uC/OS-II 的详细工作原理，通常采用上述两个函数已经可以完成大部分工作，实现简单的多任务。

下面介绍一下多任务的建立。

任务的建立可以在多任务环境启动之前，也可以在正在运行的任务中建立。建立任务函数的第一个参数实际就是要建立的任务函数名，

第二个参数是需要带入任务的参数，一般不需要时可设为 0，第三个参数时任务堆栈指针，堆栈的大小决定于任务的需要及预计的中断嵌套层数。因控制器内存大，示范程序使用的堆栈空间也比较大。第四个参数是任务的优先级，同时也作为任务的标识，注意

用户程序中不能使用优先级 0，1，2，3，以及 OS_LOWEST_PRIO-3，OS_LOWEST_PRIO-2，OS_LOWEST_PRIO-1，OS_LOWEST_PRIO。这些优先级 μC/OS 系统保留，其余的 56 个优先级提供给应用程序。

在每个任务中，必须有如下一种调用：延时等待、任务挂起、等待事件发生（等待信号量，消息邮箱、消息队列），以使其他任务得到 CPU。通常采用延时等待（OSTimeDly），

具体操作可见示范程序，增加一个任务，首先要确定任务的标识，即任务的优先级，如下是 LED 闪烁任务的定义

#define Task_LED_PRIO　OS_LOWEST_PRIO – 7

然后建立任务堆栈空间，如下

OS_STK Task_LED_Stack[TASK_STACK_SIZE];

堆栈的大小因内存较大，示范程序中设置为 9000，实际因这个任务很简单，可以设置得很小。

然后在程序中调用如下函数，任务即开始运行：

```
OSTaskCreate(
    Task_LED,                  //任务的函数名
    (void *)0,
    (OS_STK *)&Task_LED_Stack[TASK_STACK_SIZE-1],
    Task_LED_PRIO);
```

以下为任务函数，这里只是简单的让 LED 分别显示。

```
void Task_LED(void *Id)
{
    for(;;)
    {
        Led_Set(~0x01);
        OSTimeDly(500);
        Led_Set(~0x08);
        OSTimeDly(500);
        Led_Set(~0x02);
        OSTimeDly(500);
        Led_Set(~0x04);
```

```
            OSTimeDly(500);
        }
    }
```

关于多个任务之间的通讯，最简单的方法是使用全局变量，不过全局变量虽然简化了任务间的信息交换，但必须保证每个任务在处理共享数据的排他性，以避免竞争和数据的破坏。

一种简单的方法是只在一个任务中读写变量，其他任务只读变量。如果无法避免在多个任务中操作全局变量，可参考 uC/OS-II 相关书籍。

每个任务中必须有释放控制权的函数，如延时等待、任务挂起、等待事件发生，如果没有此类函数，任务将大量占用 CPU 时间，不过优先级比它高的任务还是可以执行，比它低级的任务将无法执行，所以在任务内部中使用 while 循环时请注意释放控制权，以保证其他任务的正确执行。

另外显示库、运动库和文件系统库完全支持 uC/OS-II，调用函数时不需要通过信号量来避免竞争。

uC/OS-II 函数部分是直接使用的源代码，请不要修改任何地方，否则会造成不可预料的错误。

四、键盘、串口及其他编程

1．键盘编程

键盘只需要如下一个函数即可。

相关头文件为“ getkey.h ”

按键状态

GetKey()

描述：获得键盘的状态

原型：unsigned char GetKey(void);

参数：无

返回值：0—无按键

其他—按下的键值

2．串口编程及其他

在控制器初始化函数 ARMTargetInit 中，串口已选择 0，波特率初始化为 115200，相关头文件为“ 44blib.h ”

1）Uart_Init()

描述：设置波特率

原型：void Uart_Init(int ch,int baud);

参数：ch　端口选择： ch=0：串口 0；其他 ：串口 1；

baud　波特率

默认： 0,115200 ；初始化只在程序开始后第一次进行；

返回值：无

2）Uart_Fromat()

描述：设置串口数据格式

原型：void Uart_Format(int ch,char lenght,char Parity,char StopBit);

参数：ch　端口选择： ch=0：串口 0；其他 ：串口 1；默认 0；

lenght 数据位：5/6/7/8 四种选择；

Parity 校验： E：偶；O：奇；M：标记；S：空；N：无；

StopBit 停止位：1：一位；2：两位。

默认：0,8,N,1

返回值：无

例如：Uart_Format(0,8,'E',2); 其中 Parity 校验命令为字符’E’

注意：如 **Uart_Init()**修改波特率后，必须调用此函数重新设置数据格式，否则将回到默认值。

3）Uart_GetKey()

描述：从串口接收一个字符，如果没有收到数据返回 0

原型：char Uart_GetKey(void);

参数：无

返回值：收到的字符

注意：不用等待，如果没有收到数据返回 0

4) Uart_SendByte()

描述：向串口发送一个字节的整型数

原型：void Uart_SendByte(int data);

参数：发送的数据

返回值：无

5）Uart_SendString()

描述：向串口送出一串字符

原型：void Uart_SendString(char *pt);

参数：字符串的指针

返回值：无

6）Uart_Printf()
描述：以标准输出格式向串口输出各种信息
原型：void Uart_Printf(char *fmt,...);
参数：标准的 printf 函数。
返回值：无

3．蜂鸣器、指示灯编程
1）Beep()
描述：蜂鸣器控制
原型：void Beep(unsigned char stat);
参数：stat =0 蜂鸣器鸣叫
stat =1 蜂鸣器不叫
返回值：无

2）Led_Set()
描述：LED 控制
原型：void Led_Set(int LedStatus);
参数：LedStatus 的低 4 位代表四个 LED 的状态
返回值：无

4．实时时钟编程
本控制器内置实时时钟，可用下述函数读写
相关头文件为“ rtc.h ”
1）Get_Time()
描述：获得当前日期、时间
原型：STRU_SYS_TIME Get_Time(void);
参数：无
返回值：当前日期时间。
2）Set_Time()
描述：设置日期、时间
原型：void Set_Time(STRU_SYS_TIME Set_SysTime);
参数：要设置的日期、时间
返回值：无

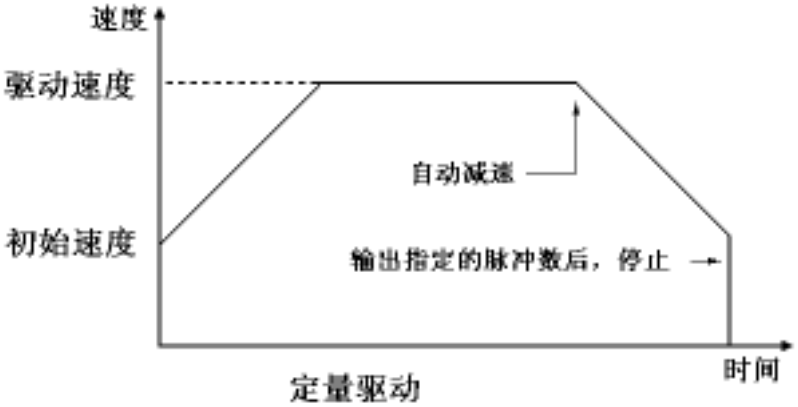
第四章　运动编程

一、定量驱动

定量驱动的意思是以固定速度或加/减速度输出指定数量的脉冲。需要移动到确定的位置或进行确定的动作时，使用此功能。加/减速的定量驱动如下图所示，输出脉冲的剩余数比加速累计的脉冲数少时就开始减速，输出指定的脉冲数后驱动也结束。

进行加/减速的定量驱动需要设定下列参数：

a) 范围 R
b) 加/减速 A/D
c) 初始速度 SV
d) 驱动速度 V
e) 输出脉冲数 P



加/减速定量驱动是一般如上图所示从计算的减速点开始自动减速，此外也可以用手动减速。在下列的情况下，不能正确地计算自动减速点或无法算出此减速点，所以需要手动地计算减速点：

- 直线加/减速定量驱动中需要经常变更速度。
- 用加/减速运行圆弧插补、连续插补。
  需要改为手动减速模式，设定减速点。

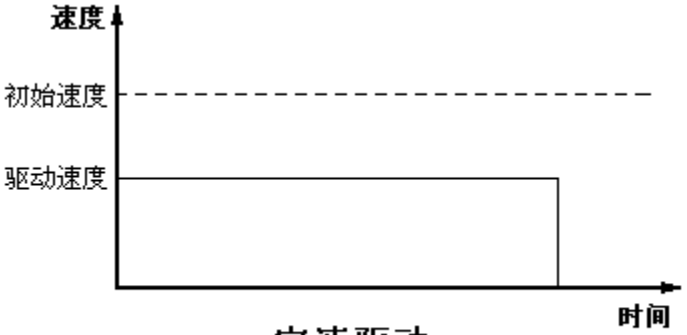
三、速度曲线

3.1 定速驱动

定速驱动就是以一成不变的速度输出驱动脉冲。如果设定驱动速度小于初始速度，就没有加/减速驱动，而是定速驱动。使用搜寻原点、编码器Z相等信号时，找到信号后马上要立即停止的话，不必进行加/减速驱动，而是一开始就运行低速的定速驱动。

为了定速驱动，下列参数应相应预先设定：

- 范围 R
- 初始速度 SV
- 驱动速度 V



定速驱动

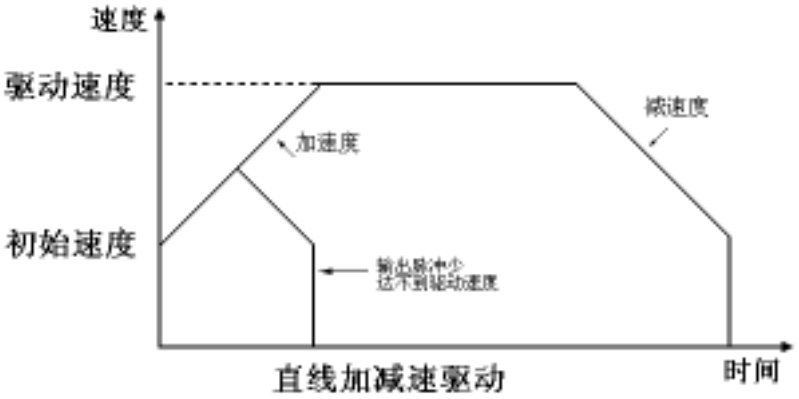
3.2 直线加/速减速驱动

直线加/减速驱动是线性地从驱动开始的初始速度加速到指定的驱动速度。

定量驱动时，加速的计数器记录加速所累计的脉冲数。当剩余输出脉冲数少于加速脉冲后，就开始减速（自动减速）。减速时将用指定的减速度线性地减速至初始速度。

为了直线加/减速驱动，下列参数需预先设定：

- 范围 R
- 加速度 A 加速度和减速度
- 减速度 D 加/减速度个别设定时的减速度(必要时)
- 初始速度 SV
- 驱动速度 V



四、位置管理

逻辑位置计数器和实位计数器

逻辑位置计数器是计数ADT854卡中的正/负方向输出脉冲，输出一个正向脉冲时，向上计1，输出一个负方向脉冲时，向下计1。

实位计数器计数来自外部编码器的输入脉冲，可以在任何时候写入或读出2个计数器的数据，计数范围在
-2,147,483,648~+2,147,483,647之间

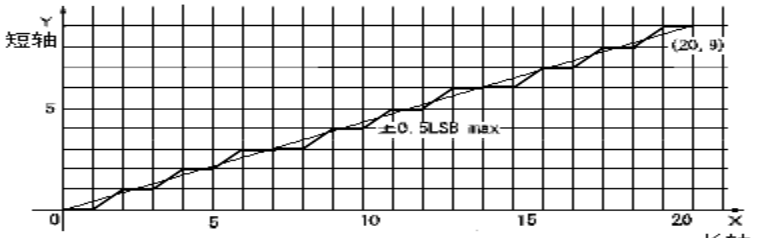
五、插补

MC4140卡可以进行2-4轴的直线插补。

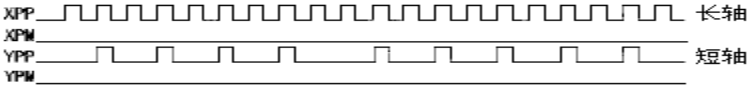
在插补驱动过程中，插补运算是在指定轴的基本脉冲时序下运行的，因此进行插补命令之前，先要设定指定轴的初始速度、驱动速度等参数。

2-4轴直线插补

设定相对于当前位置的终点坐标后就开始进行线性插补。直线插补的坐标范围是带符号的28位字长。



直线插补位置精确度



终点(X: 20, Y: 9)驱动脉冲输出例子

如上图所示，对指定直线的位置精确度，在整个插补范围内有±0.5LSB。上图还有直线插补的驱动脉冲输出例子，在设定的终点数值中绝对值最大的轴是长轴，在插补驱动中此轴一直输出脉冲，其它的轴是短轴，根据直线插补算术的结果，有时输出脉冲，有时不输出脉冲。

六、脉冲输出方式

驱动输出脉冲有下图所示的2种脉冲输出方式，以独立2脉冲方式，正方向驱动时由PU/CW 输出驱动脉冲，负方向驱动时由DR/CCW输出驱动脉冲，采用1脉冲方式则由PU/CW输出驱动脉冲，由DR/CCW输出方向信号。

脉冲输出方式	驱动方向	输出信号波形	
		PU/CW 信号	DR/CCW 信号
独立2脉冲方式	+方向驱动输出		Low 电平
	-方向驱动输出	Low 电平	
1脉冲方式	+方向驱动输出		Low 电平
	-方向驱动输出		HI 电平

七、硬件限制信号

硬件限制信号（LMT+，LMT-）是限定正方向和负方向驱动脉冲的输入信号，当限制信号的逻辑电平和限制信号有效时，将立即停止。

第七章 注意事项

1. 请不要拆开控制器，否则将不能享受保修服务。
2. 请不要使用控制器内部电源给感应开关等使用，这可能引起电源干扰，导致误动作。
3. 不要在充满强酸性气体的环境下工作，这将对从电子元器件到塑胶外壳的全部零部件造成永久性损害。
4. 开关量输出电路最好通过继电器控制外部设备，而不要直接控制外部设备。外接继电器一定要并联续流二极管，否则可能损坏内部电路或者引起误动作。如果外接固态继电器则不需要并联续流二极管。

附录一、MC4140 内存结构

MC4140 控制器内存可分为三部分

1．ROM 区：

采用 2MByte Nor FLASH ROM，位于地址 0x00000000-0x001fffff，包含 BootLoader 程序，16 点阵汉字库字模，12 点阵汉字库字模以及常用的字符字模，应用程序等。

其中 BootLoader 程序占用的地址为 0x00000000-0x0002ffff，字模占用的地址为 0x00030000-0x000affff，应用程序占用的地址为 0x000b0000-0x001fffff ， 因 此 应 用 程 序 的 大 小 不 应 超 过 0x14ffff(1376255)字节，不过一般应用程序都不会超过 1M，应该是足够了。

2．RAM 区

采用 8MByte 的 SDRAM，位于地址 0x0c000000-0x0c7fffff，

0x0c008000 作为程序的固定入口地址，由 BootLoader 程序决定，在编程设置时请勿修改，否则将无法正确启动程序。

程序运行方式类同于 PC 机，采用程序拷贝到 RAM 中运行，可加快程序运行速度。

3．数据存储区

采用 32MByte 的 Nand FLASH ROM，因为是页式操作，仅占用几个地址。

采用 FAT 文件系统的方式进行管理，通过文件和目录的操作进行数据的存储，并且可模拟为 U 盘，与 PC 机进行文件交换。

同时在调试程序时，可将程序文件通过 U 盘方式拷贝到控制器，直接执行，详细说明见 BootLoader 程序说明。

附录二、BootLoader 程序使用指南

BootLoader 程序是控制器首先执行的第一个程序，它完成应用程序的下载，运行和一些其他功能，详细介绍如下：

1.BootLoader 程序的进入

首先需要准备一根串口通讯线和一根 USB 连接线，连接控制器的串口和 PC 机的串口（假定为 COM1），连接控制器的 USB 接口到 PC 机。

打开 WINDOWS 的超级终端（对于超级终端如有疑问，可参考相关资料），其中串口设置如下：

或者可使用其他的串口调试软件，如 SSCOM 等，串口设置为波特率 115200，8 个数据位，1 个停止位，无校验，无流控制。

当控制器上电时，应控制器显示“正在启动”信息，立即按住控制器上的的‘1’键，直到信息消失，松开按键，显示四栏信息，此时

已进入 BootLoader 程序。

注意：如果已进入应用程序，只有重新上电才能进入。

2．BootLoader 程序的功能

按上下光标键选择，按[-]键进入相应功能

以下操作中[-]相当于回车键，[.]相当于取消键

a.设置系统

1、设置时间

控制器内置实时时钟，可通过本功能校正日期和时间，注意输入格式要符合要求，必须是两位数，不足两位要补 0。

只有通过串口软件才能修改时间。

2、启动画面

为防止用户错误进入 BootLoader 程序，可设置两种启动方式，一种是调试方式，此时不需输入密码即可进入 BootLoader 程序，另一种是正常方式，进入 BootLoader 程序时需要输入密码，目前密码固定为 26722719。同时，控制器启动时显示的画面也不相同。

b.设置 BIOS

1．更新 BIOS

此功能用于升级 BootLoader 程序，一般情况不需使用。

2．更新程序

此功能用于将应用程序写入 ROM 区，具体作用见后面的启动方式。

c.U 盘功能

1．U 盘连接

执行此功能后，如果 USB 线已连接好，PC 机将会找到一个 U 盘，如果是 WINDOWS2000 或 WINDOWS XP 系统，无须驱动程序，如果是 WINDOWS98 系统，可用附带的 U 盘万能驱动程序。正确连接后，面板上的 USB 灯应发亮。

文件交换完成后，如果是在 WINDOWS2000 或 WINDOWS XP 系统，请首先弹出 U 盘后，按[.]键退出，如果是在 WINDOWS98 系统，请看控制器上的显示，确认已进入“ 空闲 ”状态一秒以上，按[.]键退出。否则可能破坏文件系统。

2．格式化 U 盘

此功能将 U 盘上所有数据清除，建立两个目录（ADT 和 PRG），因为此功能将丢失所有数据，如果需要保存某些文件，可将之拷贝到 PC 机上，格式化完成后再拷贝回来。

d．启动方式

目前支持 U 盘启动和正常启动，串口启动因速率太慢，未使用。

正常启动是指 BootLoader 程序将 ROM 区 0x000b0000 地址的应用程序拷贝到 RAM 区 0x0c008000 地址中运行。

U 盘启动是指 BootLoader 程序将 U 盘中 ADT 目录下的 ADTROM.BIN 文件拷贝到 RAM 区 0x0c008000 地址中运行。

使用 U 盘启动方式，可方便的调试程序，只要将编译生成的 ADTROM.BIN 文件通过 U 盘拷贝到 ADT 目录下，重新启动控制器，即可执行程序。

对于已调试好的应用程序，建议采用正常启动方式，因为在 U 盘方式下，如果文件系统被破坏，将无法启动应用程序。

前面所说的更新程序即是将 ADT 目录下的 ADTROM.BIN 文件写入 ROM 区，以便采用正常方式启动。

附录三、ADS1.2 软件使用简介

1. ADS1.2 集成开发环境简介

ADS1.2是一个使用方便的集成开发环境，全称是ARM Developer Suite v1.2。它是由ARM 公司提供的专门用于ARM 相关应用开发和调试的综合性软件。在功能和易用性上比较SDT 都有提高，是一款功能强大又易于使用的开发工具。以下就我们对ADS1.2 进行一些简要的介绍。

ADS 囊括了一系列的应用，并有相关的文档和实例的支持。使用者可以用它来编写和调试各种基于ARM 家族RISC 处理器的应用。你可以用ADS 来开发、编译、调试采用包括C、C++和ARM 汇编语言编写的程序。

ADS 主要由以下部件构成：

- 命令行开发工具；
- 图形界面开发工具；
- 各种辅助工具；
- 支持软件。

其中重点介绍一下图形界面开发工具。

- AXD 提供给基于Windows 和UNIX 使用的ARM 调试器。它提供了一个完全的Windows 和UNIX环境来调试你的C, C++, 和汇编语言级的代码。
- **CodeWarrior IDE** 提供基于Windows使用的工程管理工具。它的使用使源码文件的管理和编译工程变得非常方便。但CodeWarrior IDE 在UNIX下不能使用。

2.使用CodeWarrior 建立工程并进行编译

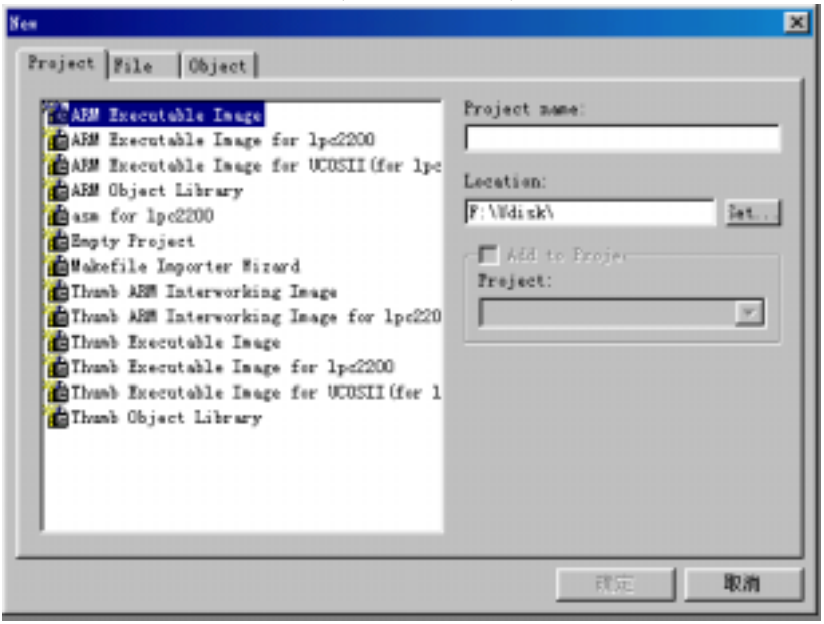
首先我们学习如何使用ADS 中的CodeWarrior——项目管理器来管理源代码。一个嵌入式系统项目通常是由多个文件构成的，这其中包括用不同的语言（例如汇编或C）、不同的类型（源文件，或库文件）的文件。CodeWarrior 通过“ 工程（Project） ”来管理一个项目相关的所有文件。因此，在我们正确编译这个项目代码以前，首先要建立“ 工程 ”，并加入必要的源文件、库文件等。

2.1调入模板或重新建立项目

我们通常采用工程模板来建立新的工程，工程模板已经针对目标

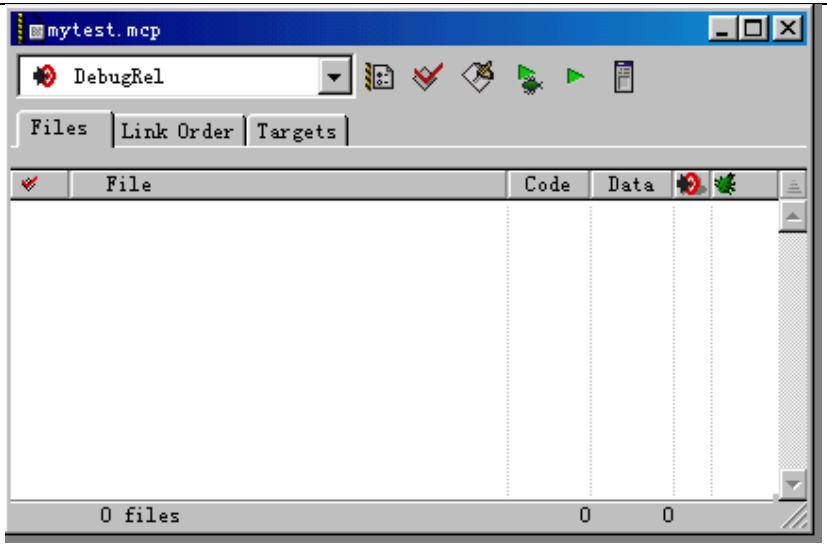
系统对编译选项进行了设置，为避免重复设置，我们可使用示范程序使用的模板——mc4140test.mcp。

如果你不想利用模板，也可以按照以下步骤来新建一个工程：选择File 菜单下的new选项，或直接按下，出现以下对话框：

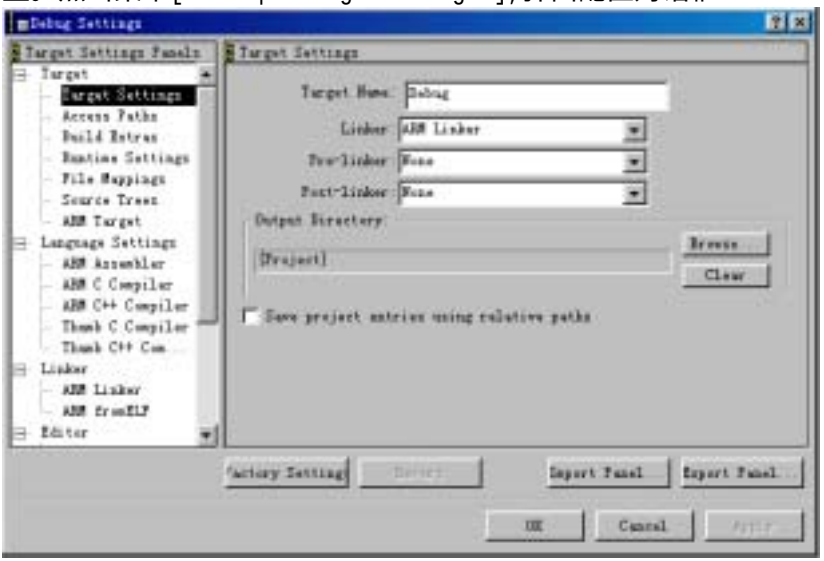


新建工程

2）．选中“ ARM Executable Image ”选项，在右边的编辑框中输入工程名（例如mytest），
在下面的Location 栏中，点击“ Set… ”，选择放置工程的路径。
3）．点击“ 确定 ”则工程被建立：



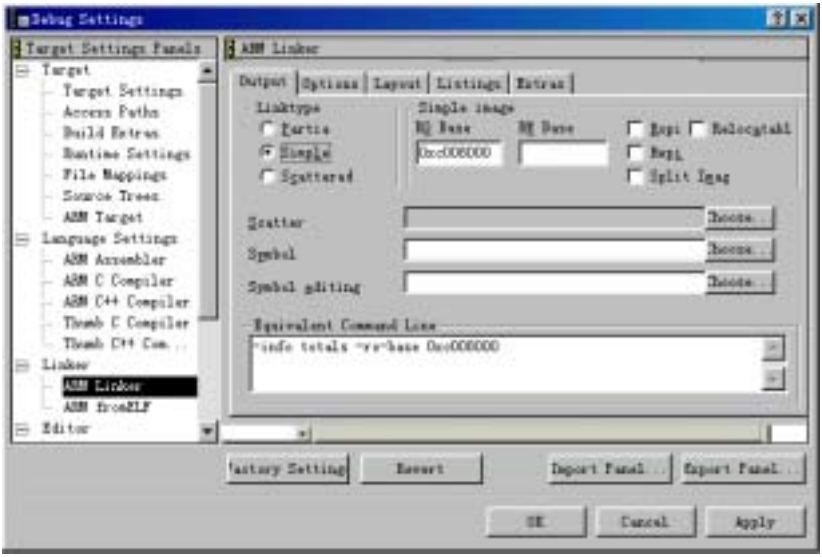
但这样的工程还并不能正确地被编译，还需要对工程的编译选项进行适当配置。为了设置方便，先点选Targets页面，选中DebugRel和Relese变量，按下Del键将它们删除，仅留下供调试使用的Debug变量。点击菜单[Edit | Debug Setting…],弹出配置对话框：



首先选中Target Setting，将其中的Post-linker设置为ARM

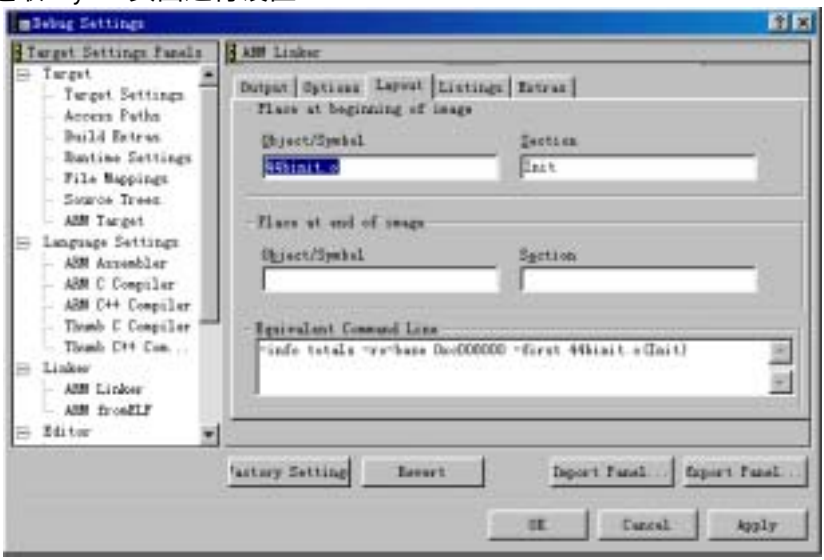
fromELF，使得工程在链接后再通过fromELF 产生二进制代码。

然后选中ARM Linker，对链接器进行设置：



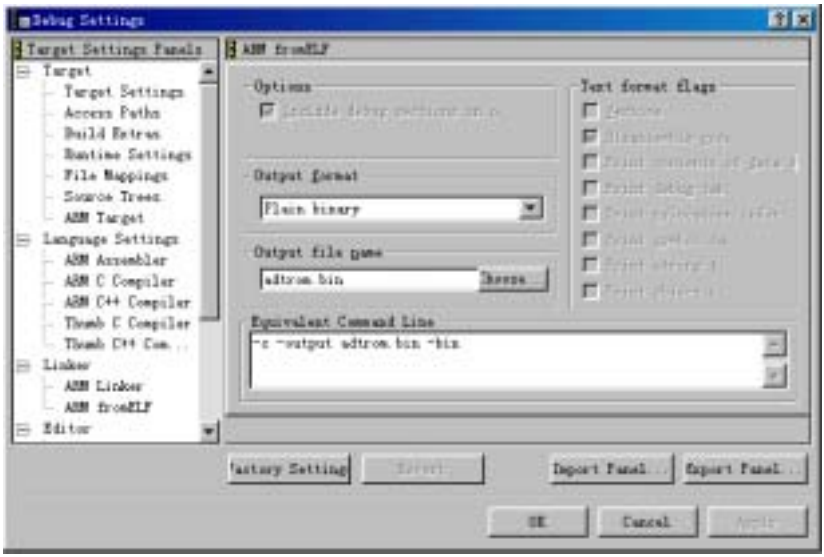
注意， -ro-base的设置必须为0xc008000。

选取Layout页面进行设置：



将44binit.o 放在映象文件的最前面，它的区域名是Init。

最后，编译的最后生成二进制文件，就要设置ARM fromELF：



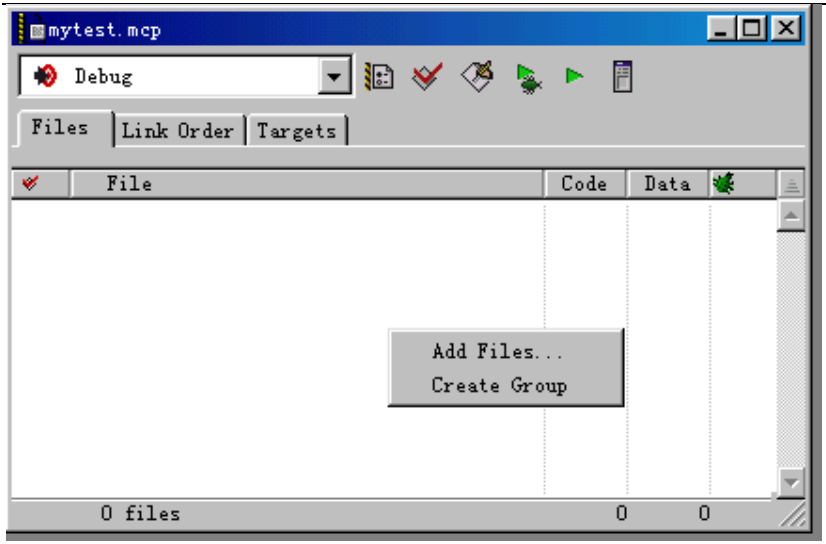
在Output format 栏中选择Plain binary，在Output file name 栏中，点击“ Choose… ”选择你要输出的二进制文件的文件名和路径（因为控制器要求文件名必须为adtrom.bin，建议在此处设置为此名称）。

这样，对于Debug变量的基本设置都完成了。按下“ OK ”键退出。

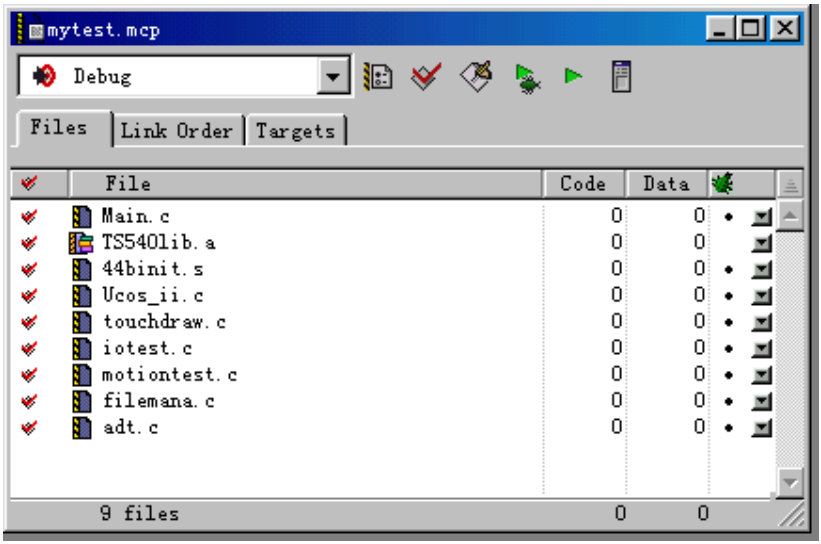
2.2在工程中添加源文件

在前图的新建对话框中，点选File 页面，选中Text File，并设置好文件名和路径，点击确定，CodeWarrior就会为你新建一个源文件，并可以开始编辑该空文件。CodeWarrior与SDT中的APM不同，它具有一个很不错的源代码编辑器，因此，大多数时候，我们可以直接采用它的代码编辑器来编写好程序，然后再添加到工程中。

添加源文件的步骤如下：例如添加main.c文件，点选Files 页面，在空白处按下鼠标右键，点选“ Add Files ”项，从目录中选取main.c文件，点击“ 打开 ”，main.c文件就被加入了工程中。



用同样的方法，将所有的*.C和*.S源文件文件都添加到source 中去，同时还有一个mc4140lib.a文件，这是一个库文件，这个文件也必须添加到工程中。同样的方法，按鼠标右键，点选“Add Files”项，将mc4140lib.a文件添加到工程中。所有必须的文件添加完成后，如下图所示。



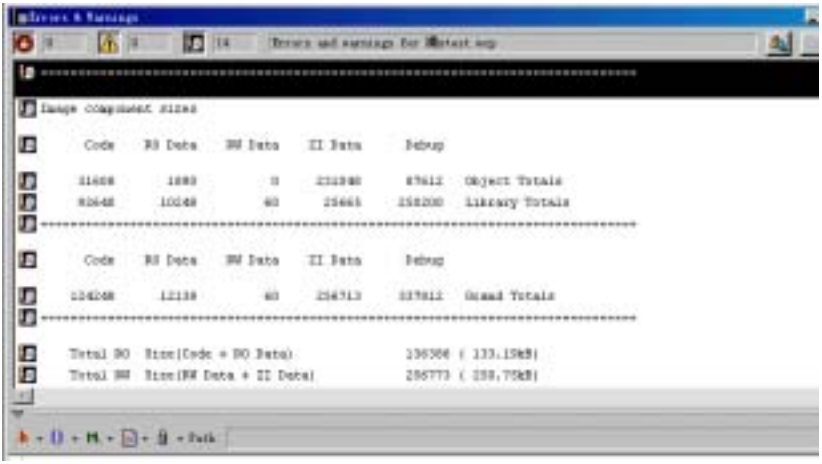
2.3进行编译和链接

注意到在上图中新加入的文件前面有个红色的“ 钩 ”，说明这个文件还没有被编译过。

在进行编译之前，你必须正确设置该工程的工具配置选项。如果前面采用的是直接调入工程模板，有些选项已经在模板中保存了下来，可以不再进行设置。如果是新建工程，则必须按照上节中所述的步骤进行设置。

- 选中所有的文件，点击  图标进行文件数据同步；
- 点击  按钮，对工程进行make，make的行为包括以下过程：
  - 编译和汇编源程序文件，产生*.o对象文件；
  - 链接对象文件和库产生可执行映像文件；
  - 产生二进制代码。

Make之后将弹出“ Errors & Warnings ”对话框，来报告出错和警告情况。编译成功后的显示如下。注意到左上脚标示的错误和警告数目都是0：



在\mytest\mytest_Data\Debug目录下会生成adtrom.bin文件。

如果有错误产生，将不会产生adtrom.bin文件。

如果有警告产生，虽然能产生adtrom.bin文件，但请确认警告信息是正常的，或者修改程序，使警告不再产生。

注意：

1．如果是将程序整个目录拷贝到其他目录或其他电脑上，请首先移去所有目标代码，可用菜单选项“ Project ”下的“ Remove Object Code… ”,选择OK即可。

2．请不要使用汉字做目录名，否则点击.mcp文件时将产生错误。

3.下载程序并运行

经过上述步骤产生的adtrom.bin文件，即是可执行的二进制文件，可通过BootLoader程序进入U盘，将文件拷贝到U盘的adt目录下，具体操作请参考BootLoader程序使用说明，请确认是U盘启动方式，关闭电源，重新打开电源，程序即开始运行。

4．程序调试要点

44binit.s文件是程序的入口，采用汇编语言编写，请勿做任何修改，否则可能导致程序无法启动。

调试程序的方法一般是利用串口，在程序中使用Uart_Printf等串口函数与PC机进行交互。PC机可用超级终端或者SSCOM等软件。