

# 手持式智能卡读写设备

**ARM7 平台**

**软件开发手册**

**[V2.0]**

**[2008-12-5]**

## 目录

<b>1</b>	<b>概述 .....</b>	<b>5</b>
1.1	ARM7 手持式机概述 .....	5
1.2	软件结构 .....	5
1.3	软件开发环境.....	6
1.4	程序运行过程.....	6
1.5	中断和异常 .....	6
1.6	回调 .....	6
1.7	子系统 .....	7
1.7.1	显示 .....	7
1.7.2	输入 .....	7
1.7.3	声音和背光.....	7
1.7.4	智能卡 .....	8
1.7.5	串行通讯.....	8
1.7.6	IIC 接口.....	8
1.7.7	SIO 接口.....	8
1.7.8	USB 接口.....	8
1.7.9	存储器和数据存储.....	9
1.7.10	电源管理.....	9
1.7.11	实时时钟和定时器.....	9
1.8	参考资料 .....	9
<b>2</b>	<b>软件开发流程 .....</b>	<b>11</b>
2.1	新建一个ADS1.2 项目或打开一个已有项目 .....	11
2.1.1	项目文件路径和命名.....	11
2.1.2	包含必要的头文件库代码.....	11
2.1.3	配置项目, 设置“RO Base”为 0x0C060000.....	12
2.1.4	配置项目, 设置“Image entry point”为AppBegin .....	12
2.1.5	配置项目, 设置“Object/Symbol”为crt0.o, “Section”为Init.....	13
2.1.6	配置项目, 设置输出文件格式为“plain binary” .....	13
2.2	编写和编译代码.....	14
2.3	代码下载 .....	14
2.3.1	复位手持机.....	14
2.3.2	选择下载方法.....	14
2.3.3	连接手持机和PC 机.....	15
2.3.4	选择程序文件, 启动下载.....	15
<b>3</b>	<b>USB驱动安装 .....</b>	<b>16</b>
3.1	步骤 1 .....	16
3.2	步骤 2 .....	16
3.3	步骤 3 .....	17
3.4	步骤 4 .....	17

# 手持式 POS 机

3.5	步骤 5 .....	18
3.6	步骤 6 .....	18
3.7	步骤 7 .....	18
4	典型的应用程序流程图.....	19
5	子系统接口 .....	20
5.1	显示设备 .....	20
5.1.1	设备的控制.....	20
5.1.2	清屏.....	20
5.1.3	文本 .....	20
5.1.4	光标 .....	21
5.1.5	字体和属性.....	21
5.1.6	写字符串.....	21
5.1.7	绘图 .....	22
5.1.7.1	显示数据更新模式(输出模式) .....	22
5.1.7.2	像素.....	22
5.1.7.3	位图.....	22
5.1.8	图标 .....	23
5.2	键盘 .....	23
5.2.1	初始化.....	24
5.2.2	按键模式.....	24
5.2.3	按键状态.....	24
5.2.4	按键伴音和自动的背光照明.....	25
5.3	UART1(RS-232, 红外), UART2.....	25
5.3.1	UART 参数.....	25
5.3.2	数据访问.....	25
5.3.3	状态查询.....	26
5.3.4	外围设备.....	26
5.4	实时时钟(RTC) .....	26
5.5	定时器 .....	27
5.5.1	系统定时器.....	27
5.5.2	回调 .....	27
5.5.3	用户定时器.....	28
5.6	其他功能设置.....	28
5.6.1	蜂鸣器.....	28
5.6.2	EL 背光灯.....	28
5.6.3	LED 指示灯 (0818 only) .....	29
5.7	存储管理 .....	29
5.7.1	NorFlash 空间分配图.....	29
5.7.2	SDRAM 空间分配图.....	30
5.7.2.1	核心库程序.....	30
5.7.2.2	代码(Code) .....	30
5.7.2.3	数据(Data) .....	30
5.7.2.4	未初始化的全局变量(ZI) .....	30

# 手持式 POS 机

---

5.7.2.5	堆 .....	31
5.7.2.6	堆栈.....	31
5.8	系统消息 .....	31
5.8.1	消息轮询.....	31
5.8.2	消息 .....	32
5.9	NORFLASH .....	32
5.9.1	Flash操作API .....	32
5.9.2	写Flash驱动的注意事项.....	33
5.10	EEPROM读写函数 .....	33
5.11	参数读写函数.....	33
5.12	NANDFLASH.....	33
5.13	SIO端口 .....	34
5.14	数据库接口 .....	34
5.15	扩展API.....	34
5.16	接触IC接口 .....	34
5.16.1	卡片接口的电源开/关和初始化.....	34
5.16.2	卡片插槽的选择/关闭.....	35
5.16.3	卡片的ATR.....	35
5.16.4	CPU卡(异步卡)的接口.....	35
5.17	非接触卡操作.....	35
5.18	条码模块操作.....	35
5.19	GPRS模块操作.....	36
5.20	433MHz无线模块操作 .....	36
5.21	打印机模块操作.....	36
5.22	网络模块操作.....	36
5.23	语音模块操作.....	36
5.24	磁卡操作 .....	37

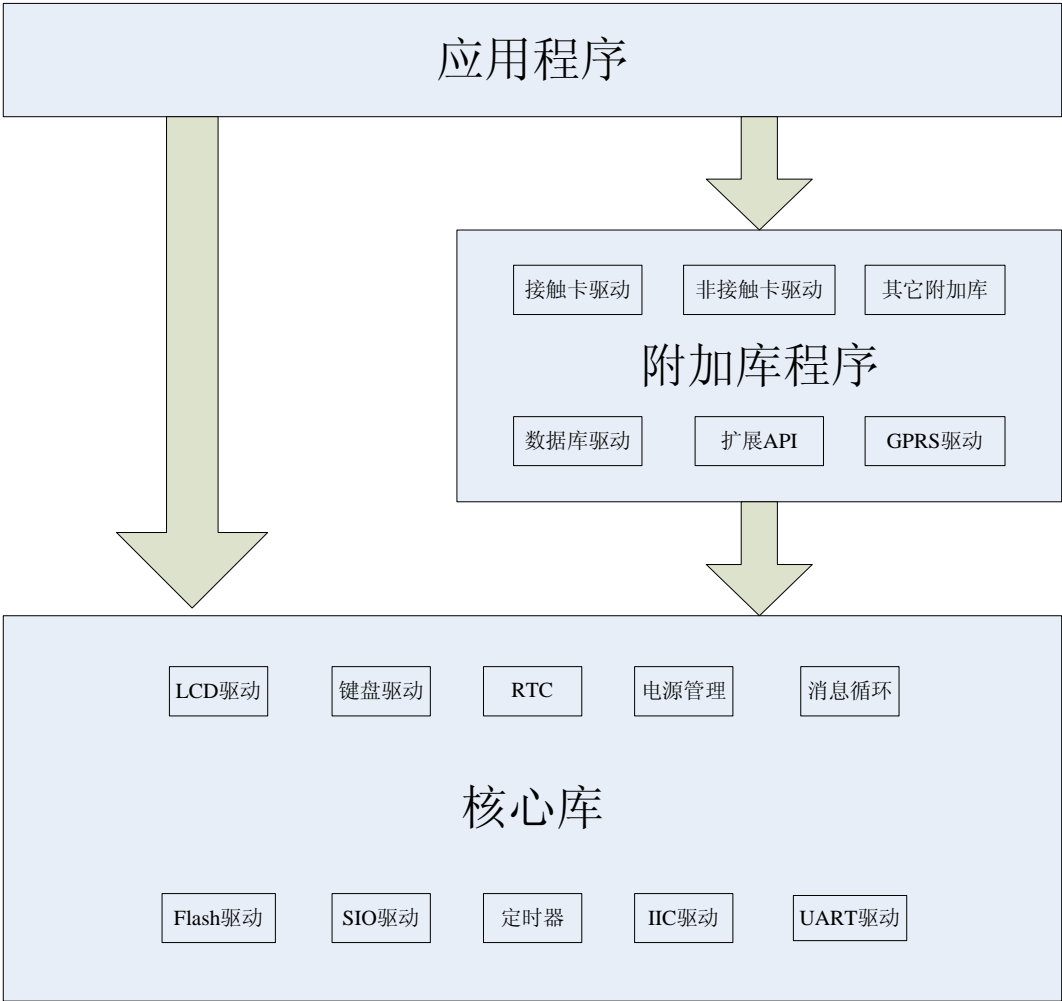
# 1 概述

## 1.1 ARM7 手持式机概述

ARM7 手持机是高级的由电池供电的智能卡读/写设备 (RWD)。它使用 Samsung S3C44B0X ARM7 MCU，具有 2~8M 的 Flash 和 8M 或 32M 字节的 SDRAM；标准输入设备是键盘；标准输出设备是一个 LCD 显示屏，0318/0518 还有一个图标显示行。它的通讯端口包括一个 RS-232 串行端口和一个 IRDA 红外端口，0518/0818 还有 USB 接口。两个 (0818 有四个) PSAM 卡片插槽。为了扩展功能，还可以在手持机中加装一些扩展功能模块 (如条码模块、GSM/GPRS 模块、非接触卡接口模块、433MHz 无线模块等)。

## 1.2 软件结构

手持机的软件构成包含三个层次：核心库，附加库和应用程序，如下图所示。



ARM 手持机软件层次图

# 手持式 POS 机

---

在发行时，核心库存在于手持机的 Flash 中的；附加库文件则需要和应用程序一起编译链接。

## 1.3 软件开发环境

软件开发的编程语言是 C/C++，目前使用的开发工具是 ARM ADS1.2。GNU 开发工具以及相关的 IDE 很快就会提供。

## 1.4 程序运行过程

初始化之后，核心库中的初始代码将检查用户的应用程序是否正确的，如果无误，系统就开始执行用户的应用程序。因为手持机是电池供电的设备，所以要尽可能让系统在绝大部分时间里处于节电模式来降低功耗。如果在使用一些外部设备时不需要提供系统时钟的话(例如不需要 UART，不需要激活与卡之间的会话)，休眠模式是最好的选择，此时电流将会降至 1mA；当需要系统时钟时，可以使用空闲模式，此时对于外围的 RS-232，电流大约为 10mA，对于红外设备，大约为 3mA。当系统在全速运行模式下，其电流大约是 30mA。节电模式(空闲模式/休眠模式)不会自动进入，唯一的进入方法是调用系统的消息定时查询程序(参见系统消息一节)，该程序将会使系统进入节电模式，并在有效的事件发生时返回正常模式。

## 1.5 中断和异常

内核将会处理所有的中断和异常。如果 CPU 处于休眠状态，任何中断和异常会将其唤醒。而且任何致命的异常如地址错误或者总线故障都将立即终止应用程序。

用户不可能直接控制这些中断和异常。然而，如果屏蔽某些消息或者关掉相应的设备，就可以屏蔽其中的一些中断。

## 1.6 回调

通过回调函数可以实现运行用户定义的中断服务程序，系统定时器中断。

另外，0318/0518/0818 的机型还可以接一条额外的中断引脚，它也可以处理用户提供的针对这条中断引脚设置的中断服务程序。

回调函数仅在对时间要求严格的场合使用。

## 1.7 子系统

手持机包含几个子系统：显示、输出、声音、背光、卡操作、UART、IIC、USB、Flash 等。

### 1.7.1 显示

0518 手持机有一个 128\*128 点阵 LCD 显示屏，外加一些图标显示。

0528 手持机有一个 128\*160 点阵的 CSTN 彩屏。

0818 手持机有一个 320\*240 点阵的 TFT 彩屏。

0318 手持机有一个 128\*64 点阵的 LCD 显示屏。

LCD 显示屏拥有两套坐标系统。图形程序(比如像素、位图等)使用图形坐标系，而文本程序使用文本坐标系。

在图形坐标系统中，基本单位是由行列构成的像素，因此左上角是(0, 0)，右下角坐标取决于具体的机器型号，对 0528 是(127, 159)，对 0818 是(319, 239)。

在文本坐标系统中，横向的基本单位是行，一行高 8 像素，一个汉字占两行，最顶端的是第 0 行，最底端取决与屏的大小，例如 0528 为第 19 行，0818 为第 29 行。列的基本单位是像素，因此左上角是(0, 0)，右下角坐标取决于具体的机器型号，对 0528 是(127, 159)，对 0818 是(319, 239)。

### 1.7.2 输入

手持机的输入设备是一个键盘。当将它们按下的时候，可以编程控制它们是否发出嘟嘟声和是否打开背光。

每一个键都有一个扫描码，可以通过 API 函数读取按键扫描码。

关于按键的命名，请参考随产品附带的使用说明书。

### 1.7.3 声音和背光

手持机有一个蜂鸣器，它通过 CPU 中的定时器输出来驱动。声音的频率和持续时间可以用 API 函数控制。

背光系统包括键盘的背光(由发光二极管来照明)和 LCD 背光(由 EL 板提供照明)。由于这两个系统是连接在一起的，因此不能单独控制。

# 手持式 POS 机

## 1.7.4 智能卡

0818 带一个大卡座和四个小卡座，0318 可以接一个大卡座和一个小卡座，0518 手持机可以接两个小卡座，0528 手持机只能接一个小卡座。这些小卡座一般用于需要 PSAM 卡认证的场合，也可以用于操作其他符合 ISO7816/EMV/PBOC 协议的 CPU 卡或存储卡。

## 1.7.5 串行通讯

ARM 手持机包含两个 UART 接口，对于不同的机型，它们连接的设备不同：

机型	UART1	UART2
0518	连接 RS232、红外和内部串口 1	内部串口 2
0528	连接 RS232、红外	连接内部串口
0818	连接 RS232、红外和 modem	连接四个内部串口
0318	连接 RS232、红外和内部串口 1	连接四个内部串口

UART1、UART2 连接的多个接口不能同时使用，但支持各种波特率(最高可达 115200)。

RS-232 接口可以用来连接 POS 机和其它带 RS-232 接口的设备，如 PC、另一台 POS 机、调制解调器、打印机等。

红外收发器可以用来连接其它的红外设备。

## 1.7.6 IIC接口

在 0318/0518/0818 中，包含了一个 IIC 接口的 EEPROM。用户可以用它来存储一些系统参数。根据用户的需要，也可以连接其它 IIC 接口的设备，以扩展系统的功能。速率最高可以达到 400KHz。

## 1.7.7 SIO接口

SIO 端口为任何 SPI 类型的扩展模块提供了扩展接口。速率范围：128Kbps~33Mbps。

## 1.7.8 USB接口

在 0318/0518 机型中，用户可以选配 USB 接口取代 RS-232 接口，0818 有一个独立的 USB 接口。该接口兼容 USB2.0 规范，最高数据速率：2MB/s。



# 手持式 POS 机

---

## 1.7.9 存储器和数据存储

手持机系统的存储空间包括 8M 的 SDRAM (可以扩充至 32M) 和 2M 的 NorFlash (可以扩充至 8M)。

NorFlash 包含了程序代码 (包括核心库和用户的应用程序)。NorFlash 能和 SDRAM 一样读取数据, 但写入的方式却有很大区别, 并且修改起来会更为复杂: 在最差的情况下, NorFlash 会在任何一个单字节字符的写入前, 擦除一个块 (等于 64k 字节)。因为 NorFlash 空间很大, 所以它用于非易失类型的数据存储。

对于 0518, 还可以选配大容量的 NandFlash 存储器, 容量最大可到 8G 字节。

## 1.7.10 电源管理

对于使用电池进行工作的手持机来说, 电源管理显的尤为重要。任何高耗电的外围设备都可以单独关闭以减少电源消耗: RS-232, IR, 卡片接口, 内部设备等。

下列是电源管理的一些基本原则:

- ✧ 当不使用外围设备时, 要将其关闭。
- ✧ 尽量让外围设备处于节电模式。
- ✧ 不要总是让手持机停留在高电源消耗状态, 最好在处理完后进入睡眠模式。

## 1.7.11 实时时钟和定时器

手持机中含有一个实时时钟 (RTC) 系统。它就象普通的钟表, 有日期、时间和定时报警功能, 只要电池有电源的话, 它就会一直保持在运行状态, 它还能通过 API 进行读/写。当报警时, 它就会送出一个消息。

系统包含 2 个系统定时器和 127 个用户定时器。

2 个系统定时器定时范围到达时, 会送出单独的消息, 并且可以设置回调函数。127 个用户定时器共用一个消息, 而且要使用特殊的消息等待函数。

定时器的最小单位为 1/64 秒。

## 1.8 参考资料

为了开发高效无错的手持机程序, 用户还需要参考开发包里面的以下文档:

- 《手持 POS 机概述》
- 《基本 API 参考文档》
- 《扩展 API 参考文档》

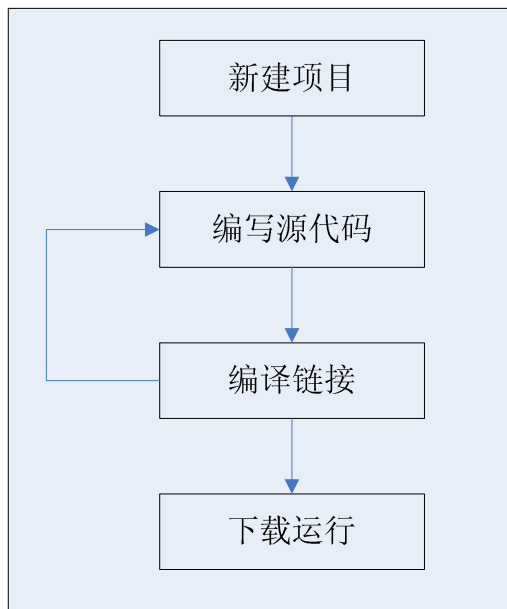
## 手持式 POS 机

---

《接触式 IC 卡 API 参考文档》  
《ISO14443 协议 API 参考文档》  
《ISO15693&ICODE1 协议 API 参考文档》  
《数据库 API 参考文档》  
《GPRS API 参考文档》  
《RS232 传输程序 API 参考文档》

## 2 软件开发流程

编写一个手持 POS 机程序的基本流程如下：



### 2.1 新建一个ADS1.2 项目或打开一个已有项目

在新的 GNU 编译环境开发出来之前，用户可以使用 ADS1.2 进行项目开发。关于 ADS1.2 的安装使用，请用户参考 ARM 公司的相关文档，这里不做详细说明。

使用 ADS 开发时，需要进行一些必要的系统设置，并且需要包含必要的头文件和库文件。为了避免出错，我们强烈建议用户从我们提供的例子程序<hello>出发进行修改。如果是新建项目，需要注意以下几点：

#### 2.1.1 项目文件路径和命名

由于 ADS 对特殊字符的支持不够好，所以建议项目文件命名和所在路径不要包含除英文字母以外的其它字符。

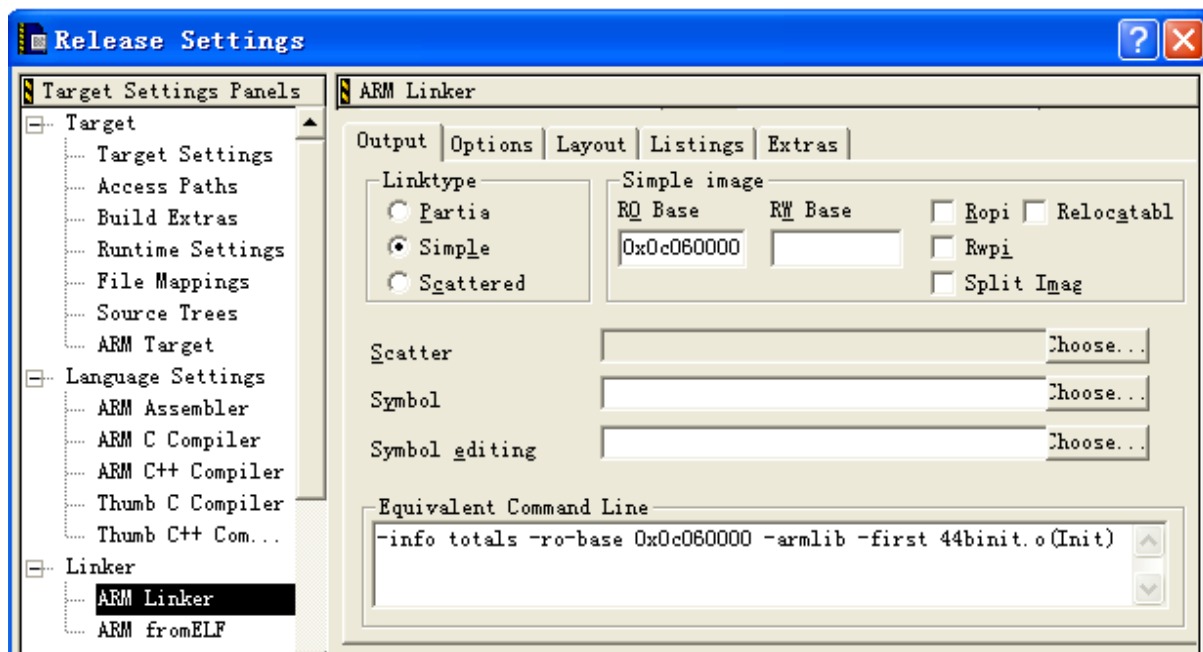
#### 2.1.2 包含必要的头文件库代码

- 将 <inc> 、<libs>、“crt0.s”放入项目相应的文件夹。<inc>和<libs>目录随开发光盘提供。

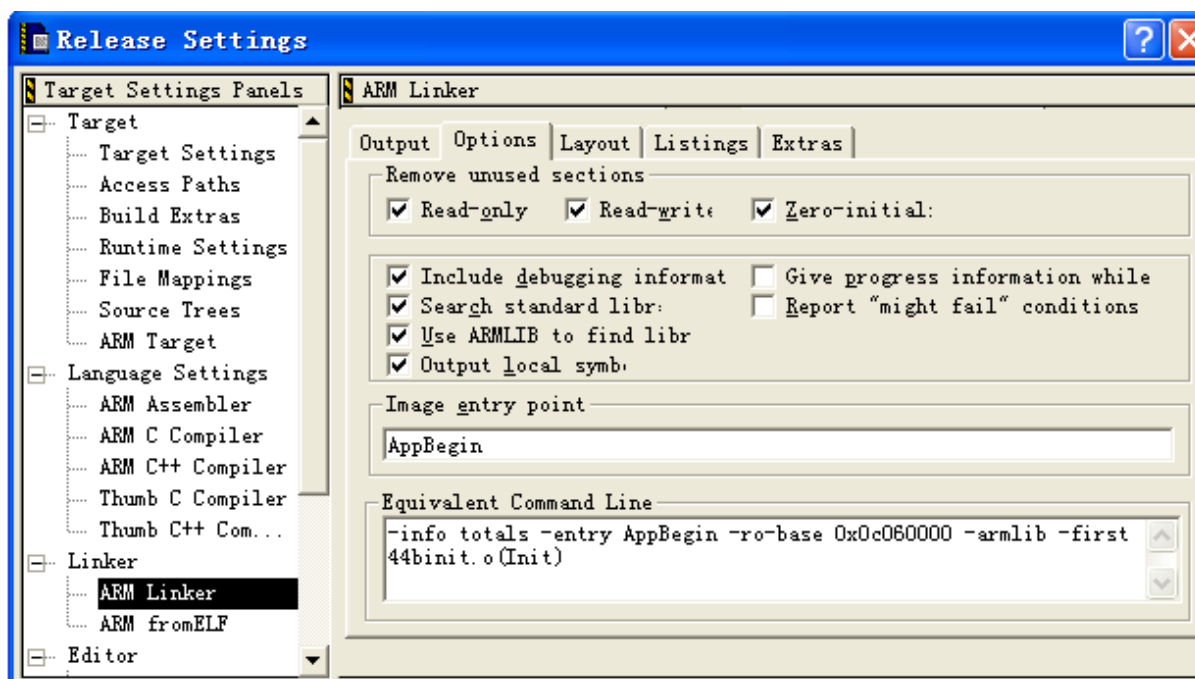
## 手持式 POS 机

- 在项目中包含 `crt0.s` 和 `libARMPos.a`
- 项目源文件需要包含 `/inc/api.h`。
- 根据需要包含其它的库文件，如 `exlib.a` 等。

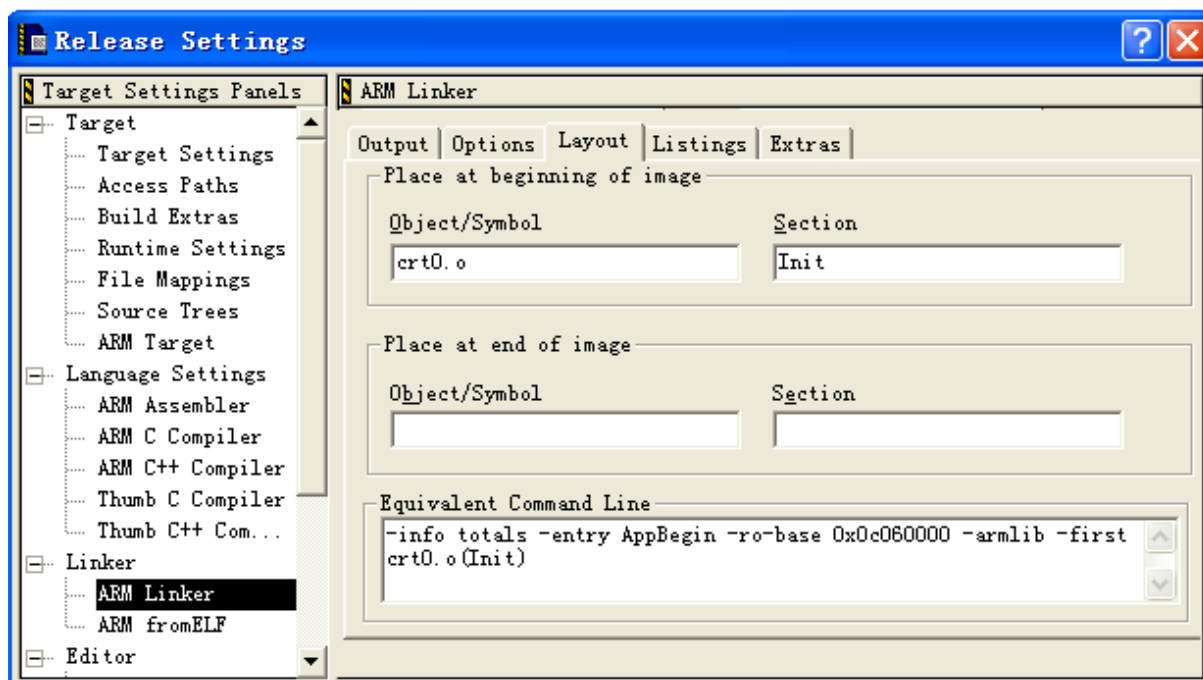
### 2.1.3 配置项目，设置“RO Base”为 `0x0C060000`



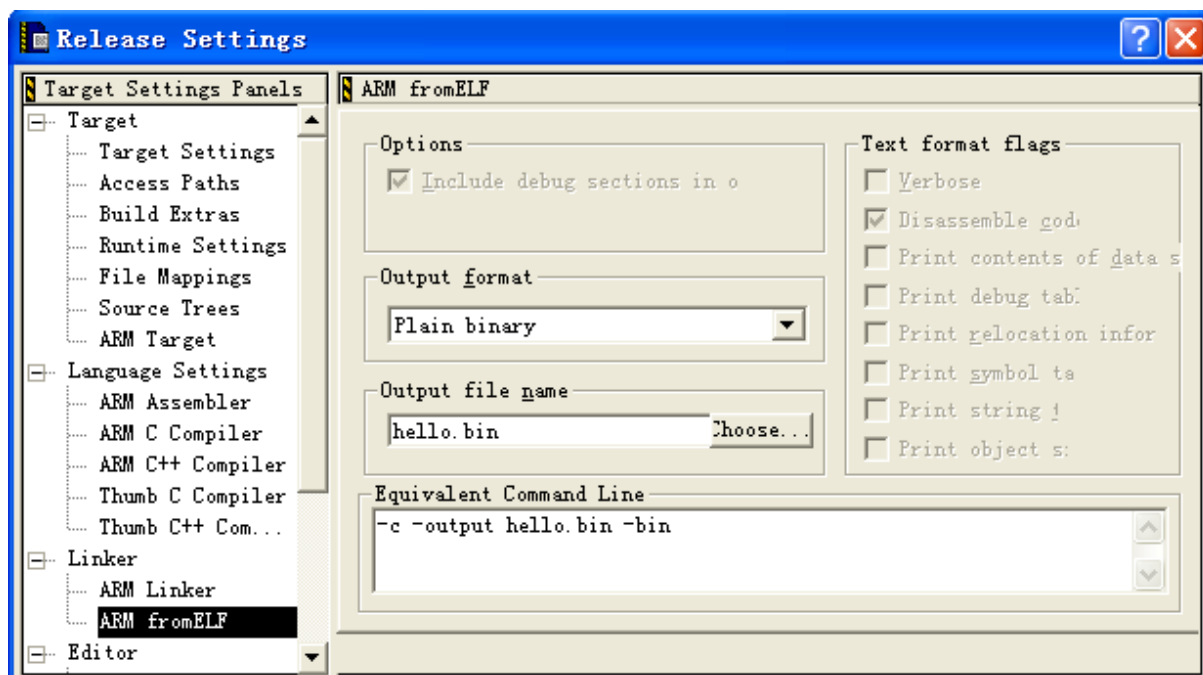
### 2.1.4 配置项目，设置“Image entry point”为 `AppBegin`



## 2.1.5 配置项目，设置 “Object/Symbol” 为 **crt0.o**，“Section” 为 **Init**



## 2.1.6 配置项目，设置输出文件格式为 “**plain binary**”



## 2.2 编写和编译代码

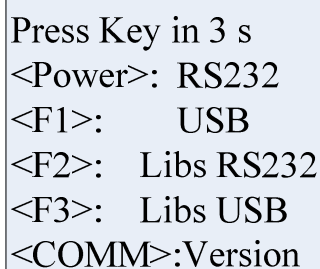
在配置好项目后,可以开始代码的编写和编译工作。代码的编写可以使用 ADS 自带的编辑工具,也可以使用其它编辑工具,如 UltraEdit 等。

## 2.3 代码下载

如果程序编译链接没有错误的话,可将生成的 bin 文件下载进手持机。下面是下载的具体步骤:

### 2.3.1 复位手持机

必须复位手持机才能进入以下下载界面(示意图):




```
Press Key in 3 s
<Power>: RS232
<F1>:    USB
<F2>:   Libs RS232
<F3>:   Libs USB
<COMM>:Version
```

### 2.3.2 选择下载方法

用户根据提示,按键选择相应的下载方法(注意:USB 转 232 方式,手持机端按 232 接口操作)。<F2>和<F3>用于更新核心库文件,<Power>键和<F1>键用于下载应用程序。因为下载核心库文件和下载应用程序的方法是一样的,使用下面仅举例说明应用程序的下载方法。

选择下载方法后将进入以下界面之一:



```
Ready to update
application !
By RS232
```



```
Ready to update
application !
By USB
```

# 手持式 POS 机

## 2.3.3 连接手持机和PC机

RS232 方式：用附带的串口线连接手持机串口和 PC 串口。

USB 方式：用附带的 USB 线连接手持机 USB 口和 PC 的 USB 口。

USB 转 RS232 方式：用附带的 USB 线连接手持机 USB 口和 PC 的 USB 口。

## 2.3.4 选择程序文件，启动下载

打开光盘附带的 ARM 下载程序：



如果使用 RS232 方式，需要选择正确的串口，波特率目前固定为 115200。

如果使用 USB 方式，PC 系统需要先安装相应的 USB 驱动。关于 USB 驱动的安装方法请参考第三章说明。

点击“浏览”按钮选择要下载的文件，然后点击“串口下载”或“USB 下载”，开始下载过程。

## 3 USB驱动安装

以下安装步骤适用于 Windows 2000/XP/VISTA 系统，对于其它系统，安装操作类似。下面用 %windir% 代表 windows 系统文件的安装路径。

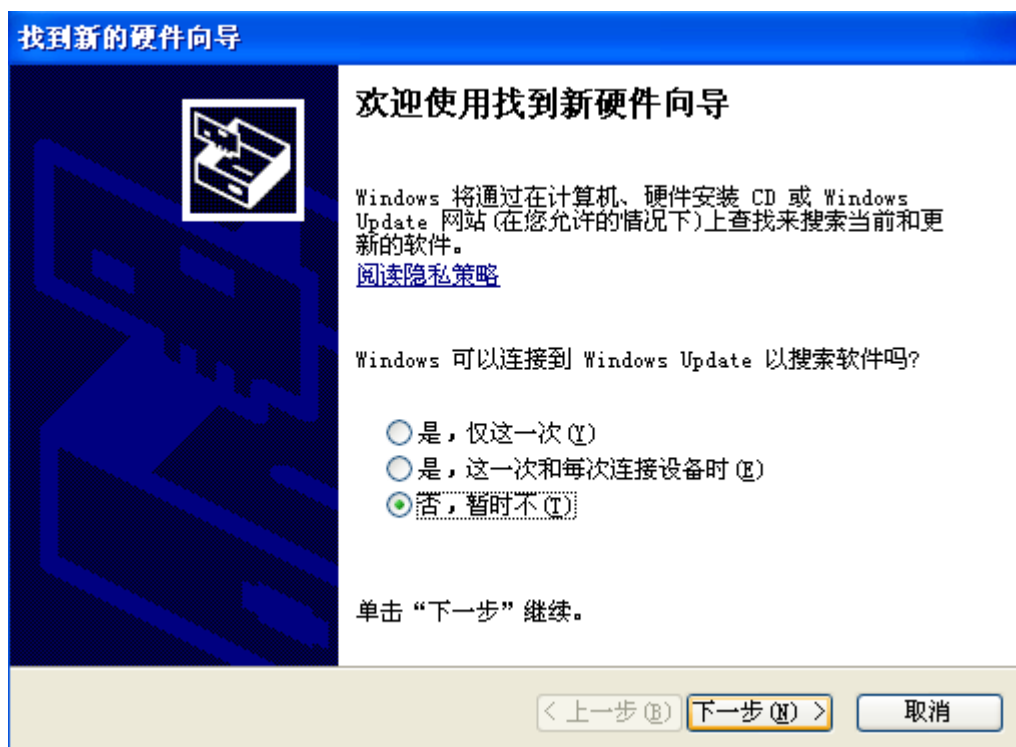
### 3.1 步骤 1

用 USB 线连接手持机 USB 接口和 PC 机 USB 接口，重启手持机，选择 USB 方式下载程序文件。手持机端出现以下界面表示手持机端的 USB 启动成功：

Ready to update  
application !  
By USB

### 3.2 步骤 2

这时，PC 端将提示“找到新的硬件向导”。如下图，选中“否，暂时不”；然后按“下一步”。



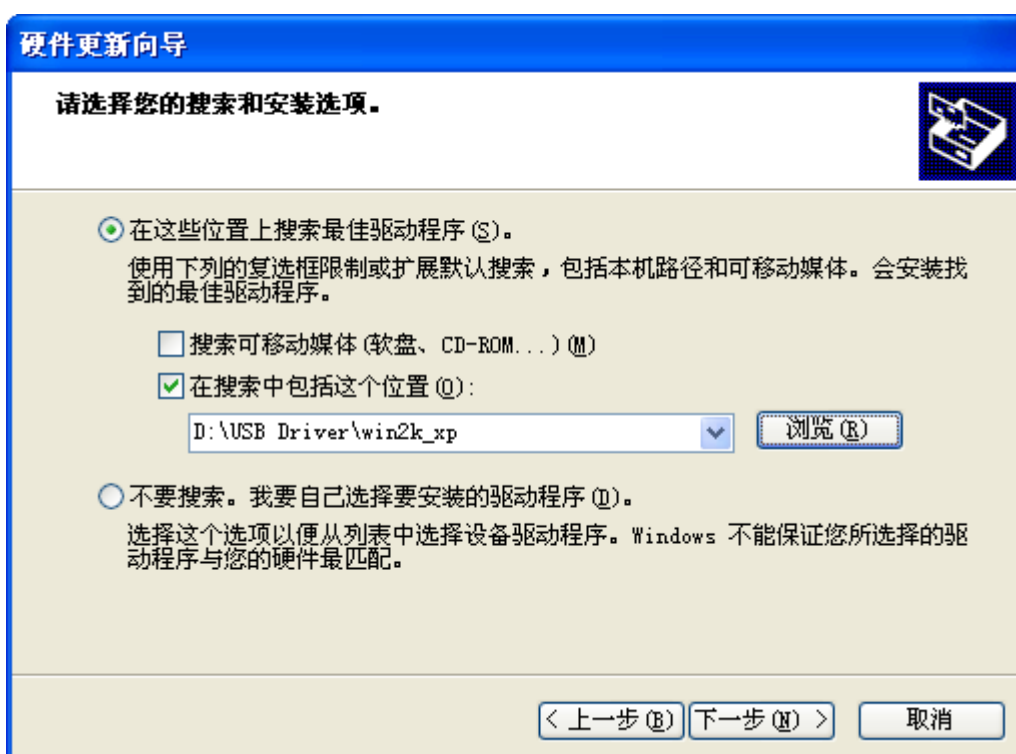


## 3.3 步骤 3

在下面界面中选中“从列表或指定位置安装（高级）”，点击“下一步”按钮继续：



## 3.4 步骤 4



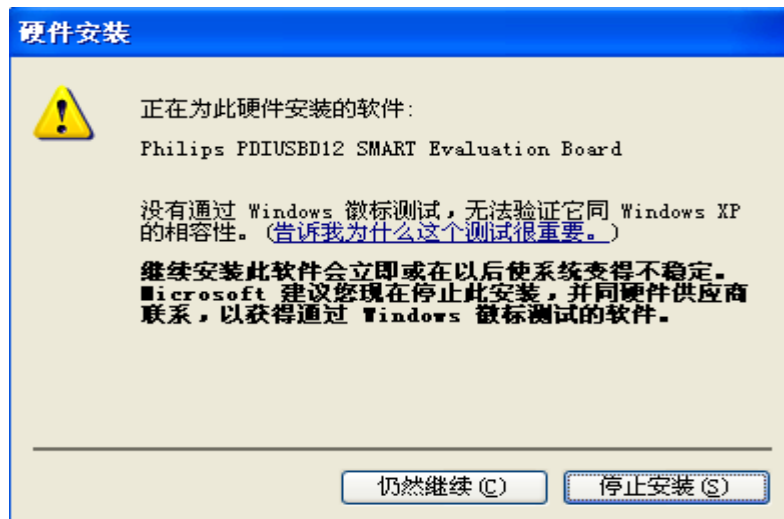
选中“在搜索中包括这个位置”选项，然后点击“浏览”按钮选择开发包的 *USB Driver\win2k\_xp*

# 手持式 POS 机

目录，点击下一步继续。

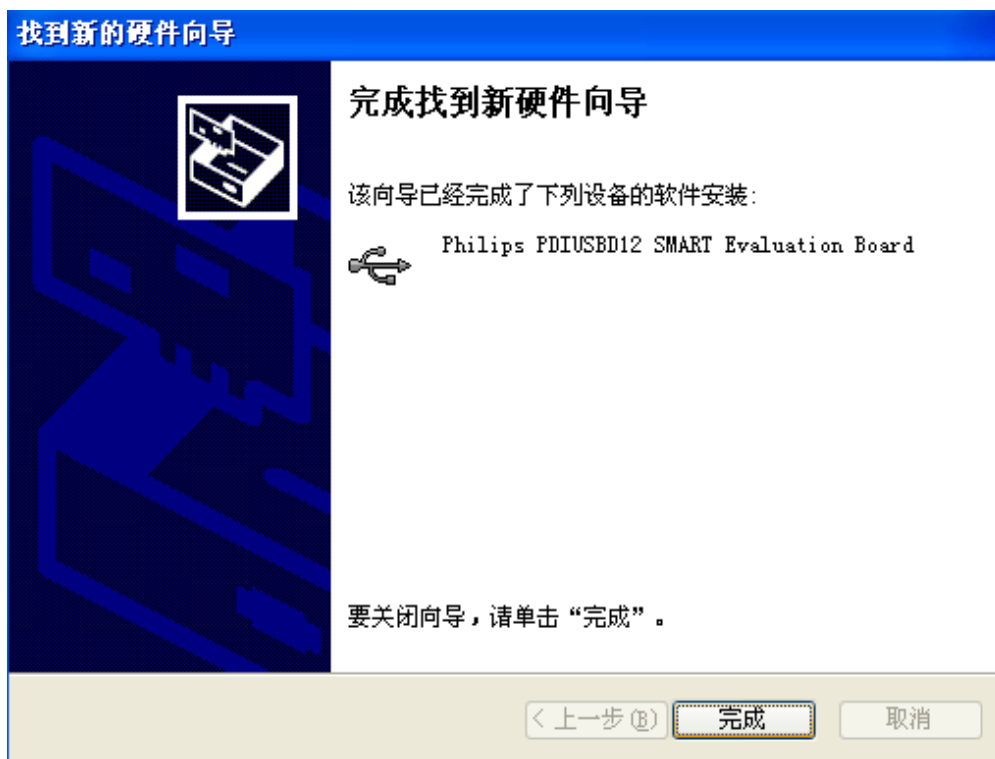
## 3.5 步骤 5

出现警告提示时点击“仍然继续”。



## 3.6 步骤 6

安装完成后将出现下述提示信息，点击“完成”按钮，驱动安装结束。



## 3.7 步骤 7

将开发包 USB Driver 目录里面的 EasyUSB.dll 文件复制到 %windir%\system32 目录下。

## 4 典型的应用程序流程图

在手持机上编写程序和 PC 下非常相似，甚至更简单：一般仅包含有一个没有返回值的应用程序。在初始化系统之后，就由应用程序来控制系统了。应用程序主要是在用 C 语言编写的。

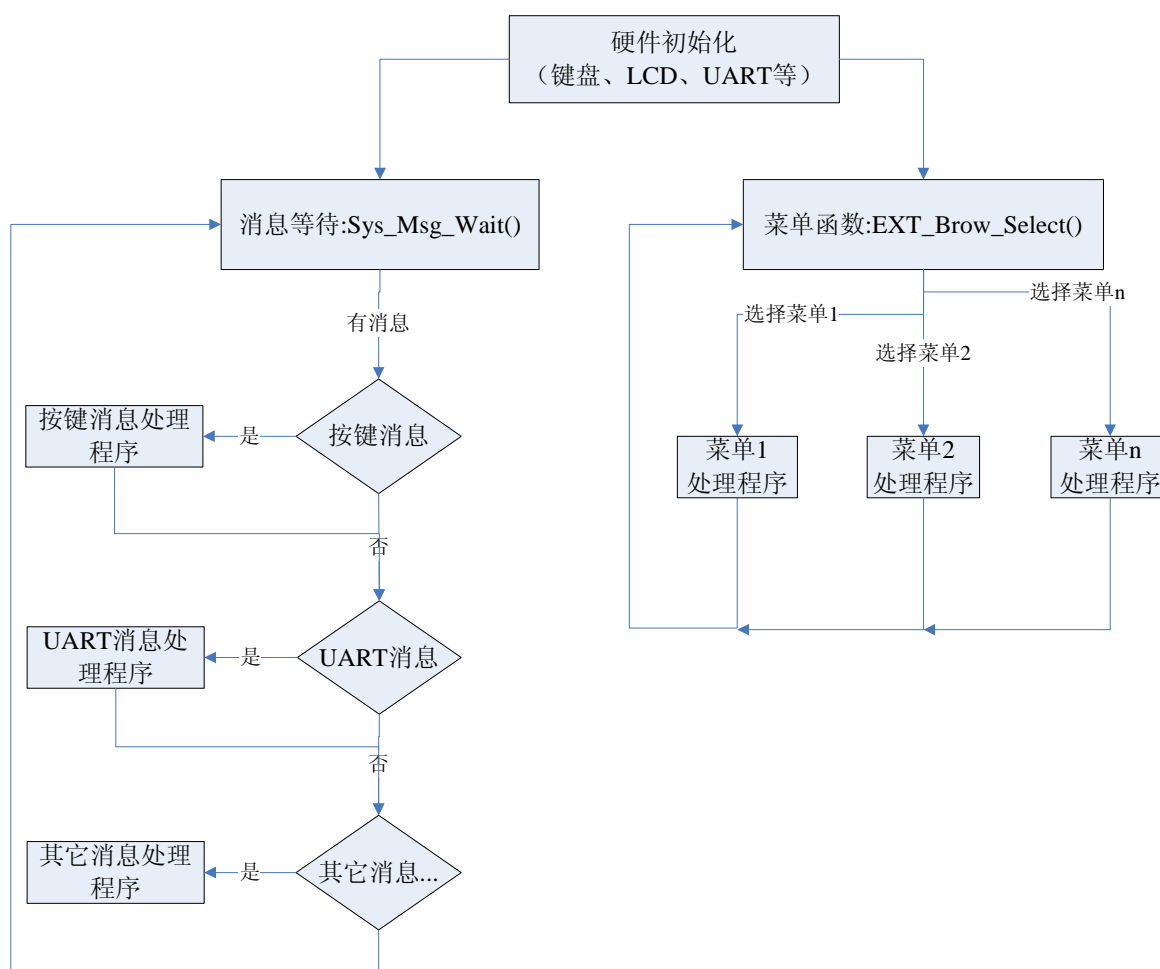
程序从大家熟悉的 `main()` 函数开始执行，请注意 `main()` 是一个不返回的函数。

`main()` 函数运行后的第一步就是初始化需要的硬件接口，如显示设备、键盘、卡片接口、SPI 端口等。

在初始化之后，就是一个死循环了。应用程序不断调用函数 `Sys_Msg_Wait()` 来检查各种事件。当一个事件发生后，应用程序就会试着运行它，结束后返回 `Sys_Msg_Wait()` 继续等待下一个事件。

在实际编程过程中，可以使用扩展库 API 中的菜单浏览选择函数，程序根据用户选择的菜单执行相应的功能，类似于 Windows 程序。但是其底层机理仍然是一个消息处理过程。

下面是大部分应用程序的执行流程图：



## 5 子系统接口

通过调用 API 和附加库程序，可以控制所有的子系统。以下 API 除非特别说明，都在《基本 API 参考文档》中描述。

### 5.1 显示设备

通常来说，我们用显示设备只做两件事：输出文本和图片。

#### 5.1.1 设备的控制

内核初始化设备之后，显示设备就可以工作了。它有三种模式可以选择：工作模式、节电模式、关闭模式。

在工作模式下，屏幕会显示出屏幕缓冲器中的任何数据，由此数据的变化会迅速的显示出来。

在节电模式和关闭模式下，主显示区(包括点阵区域和绝大部分图标区)都将关闭，因此任何数据的变化都无法显示。但在这两种模式下，耗电量会降至最低(只有几毫安)。

对于 0318/0518，除了静态图标，节电模式和关闭模式几乎是相同的。静态图标是屏幕左下角一个像小牛眼睛似的东西。对于 0528/0818，节电模式和关闭模式是一样的。

不论显示设备处于何种状态，所有的信息都会先送到设备本身的屏幕缓冲器中。

**相关的API 函数：**

`Disp_Init()`

#### 5.1.2 清屏

如果你不想看到上一屏显示留下的"尸体"，保险的做法是在显示屏开启或工作模式开启前使用清屏命令。该命令不但可以清楚整个屏幕，还可以清除选定矩形区域内的所有内容。

**相关的API 函数：**

`Disp_Clear()`

`Disp_Clr_Win()`

#### 5.1.3 文本

# 手持式 POS 机

在文本中，ASCII 字符占 8-bit，而一个汉字字符占 16-bit。要显示的字符串应当以空字符结尾。

## 5.1.4 光标

光标用来提示文本的显示位置。光标是不可见的，而且是可写不可读的。在显示了一个字符串之后，光标会自动的跳转到行尾(即下一个字符要显示的位置)。光标的 X 坐标(横坐标)的范围，0318/0518/0528 是 0~127 像素，0818 是 0~319 像素。因此，我们可以将光标设置在行的中间。Y 坐标(纵坐标)的范围，0518 是(0~15)，0528 是(0~19)，0818 是(0~29)，0318 是(0~7)。例如，当一个字符串显示在(90, 7)这点，则该字符串显示的位置是绘图坐标模式下的(90, 56)。除非您非常确定光标的位置，否则请在使用光标前设置它的位置。

**相关的 API 函数:**

`Disp_Goto_XY ()`

## 5.1.5 字体和属性

手持机中包括有一个默认的字体库。它含有下列这些字体:

6\*8 英文字符

8\*16 英文字符

16\*16 GB2312 汉字字符(一个字符占两行)

属性设置还可以定义字符是否反相显示。属性和字符必须在写入文本之前进行选择。

使用扩展库 API 的函数，还可以显示 Big5 和 GBK 字符集的字符。

对于 0528/0818 彩屏机型，还可以设置字体和背景的颜色。

**相关的 API 函数:**

`Disp_Set_Font_Attrib ()`

`Disp_Set_Color()`

## 5.1.6 写字符串

当光标和字体/属性设置好之后，写入一个字符串就非常容易了。也可以单字符写入以提高效率。请注意，如果字符串太长的话，最后的部分会被自动剪除。

**相关 API 程序:**

`Disp_Write_Str()`

`Disp_Write_Char ()`

# 手持式 POS 机

---

`Disp_Write_Str_Col()`

`Disp_Write_Char_Col()`

## 5.1.7 绘图

在绘图模式下，整个显示屏被看作一个矩形点阵，并且绘图是以像素为单位的。

绘图模式的坐标系是用像素做为单位的，比如对于 0518, (0, 0)是左上角, (127, 127)是右下角。

### 5.1.7.1 显示数据更新模式(输出模式)

对于 0318/0518，在显示屏上输出一个像素或一组像素(bitmap)有四种方法：

直接绘制       ：用新的数据(ND)代替已显示数据(DD)，不论原始数据(OD)是什么。

OR               ：DD=OD 同 ND 相或。

XOR             ：DD=OD 同 ND 异或。

AND             ：DD=OD 同 ND 相与。

输出模式是当绘制像素时设置的一个参数。但它要在绘制位图前进行设置。

**相关的API 函数:**

`Disp_Set_Bmp_Mode ()`

### 5.1.7.2 像素

在绘图模式下，最直接的方式是一个像素接着一个像素的绘制。它没什么限制，但效率很低。

如果您知道现在屏幕上的显示是什么的话，其实可以直接将像素读入。0528 机器不支持读取像素的函数。

**相关的API 函数:**

`Disp_Get_Pixel ()`

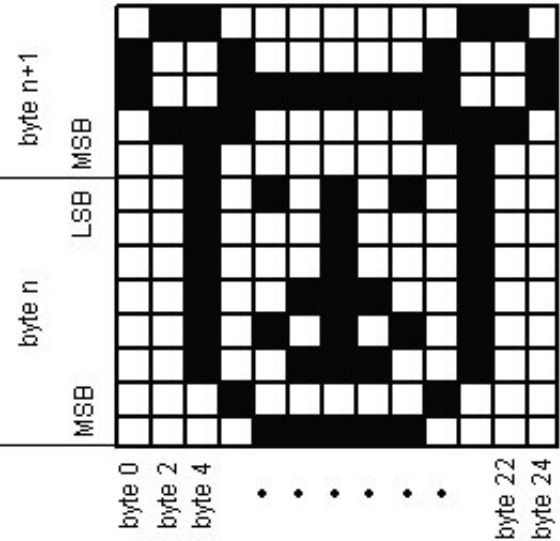
`Disp_Put_Pixel ()`

`Disp_Put_Pixel_Col()`

### 5.1.7.3 位图

位图可被定义为显示在一个矩形区域的一组数据。它能被读取，能被储存在存储器中，如果修改它，它能够覆盖原图或者也可以另外存储。这是一个改变屏幕显示的一个很便捷的方法，也是做动画的最基本解决思路。

在 API 手册中介绍了位图作为一条记录来存储的相关内容(参见位图结构一节)。根据显示设备的数据组织形式，位图数据是垂直存储的。



这个玩具熊脸谱的数据资料如下所示：

0X00	0X30	0X00	0x4B
0X3F	0XCB	0X40	0X70
0X91	0X20	0XAB	0X20
0XBF	0X20	0XAB	0X20
0X91	0X20	0X40	0X70
0X3F	0XCB	0X00	0X4B
0X00	0X30		

**备注：**

当绘图的矩形区域超出屏幕时，照例它会被修剪。但在读入一个位图和写入一个位图时，修剪方式是稍有不同的。当修剪读入的一个位图时，位图会被调整到适合屏幕大小的尺寸，然后再存储。比如你读入一个 10\*10 的位图在(120, 56)处，则位图将被保存为一个 8\*8 的位图。但当写入一个位图时，位图数据不会受影响的，屏幕外的部分仅仅是简单的不可见。因此，在制作动画、使用到连续的位图时一定要谨慎，当目标的移动要经过屏幕边界时，一定要小心数据的丢失。

**相关的API 函数：**

Disp\_Put\_Bmp ()

Disp\_Put\_Bmp\_Raw()

**5.1.8 图标**

在复位之后，0318/0518 的图标由核心程序来控制。用户也可以通过 API 来控制，但不建议这样使用。

对于 0528/0818，彩屏上没有专门的图标显示区，但可以调用 API 启用核心程序的图标显示。

**相关的API 函数：**

Disp\_Icon\_Set ()

Disp\_Icon\_Battery\_Time\_Set()

Disp\_7\_SEG\_Time()

**5.2 键盘**

## 5.2.1 初始化

输入子系统是必须进行初始化的。任何的输入模式的改变都会调用初始化函数。初始化也能设置当按键被按下时，背光灯是否打开，蜂鸣器是否发出声音等。

请注意，初始化将会清空键盘缓冲器和键盘的原有状态。因此如果你定义当按下一个键时来调用初始化函数就会让系统非常的混乱：因为此时，它不但在进行初始化，还会送一个"按键被按下"的消息给系统。

*相关的API 函数:*

`KEY_Init()`

## 5.2.2 按键模式

当一个键被按下系统会送出一个消息，这个键的代码就被压入 **FIFO**(先进先出)中。按键(包括所有的 16 个键)进入缓冲区后，即使您键入的速度很快，这个信息也不会丢失。如果 **FIFO** 是非空的，则按键的代码可以从 **FIFO** 中弹出，从而就读取了这个按键的消息。如果您一直按着一个键，系统也只会送出一个按键消息。而且当一个键被按下时，另外的按键消息就无法送出了，除非将原先按着的按键松开：



因为有一些操作有可能影响到按键缓冲器，所以我们在读取按键消息的时候，最好将所有的消息读取后，再进行下一次操作(尤其当按键被初始化后)。

*相关的API 函数:*

`KEY_Read()`

## 5.2.3 按键状态

矩阵位图可以通过 **API** 函数读出来，如果同时按下几个键，想要看按键的结果，就可以调用这个函数。

*相关的API 函数:*

`KEY_Get_Status()`



## 5.2.4 按键伴音和自动的背光照明

按键伴音和自动的背光照明可以被独立的打开，因此一些特殊的按键如"shift"和"power"等可以设置成无伴音键。

*相关的API 函数:*

`KEY_EL_Mask_Set()`

`KEY_Beep_Mask_Se ()`

`KEY_EL_Mask_Read()`

`KEY_Beep_Mask_Read()`

## 5.3 UART1(RS-232, 红外), UART2

### 5.3.1 UART参数

可以设定一些串行通讯的参数:

- 1) 奇偶校验: 非奇非偶, 奇, 偶。
- 2) 停止位: 1 个或 2 个停止位。
- 3) 数据位: 7 位或 8 位。
- 4) 波特率: 可设置为下列值中的一个:

115200, 57600, 28800, 14400, 38400, 19200,

9600, 4800, 2400, 1200, 600, 300

当可以建立稳定连接时, 推荐使用高的波特率来提高效率, 比如 RS-232 电缆连接或者处于一定范围内的 IR 连接(一般在 20cm 之内)。

因为 UART 的初始化程序会选择三个外围设备中的一个, 所以当改变 UART 的参数时, 要注意函数的其它参数不会切换外围设备。

*相关的API 函数:*

`UART_Init()`

`UART2_Init()`

### 5.3.2 数据访问

串行通讯可以是全双工通信制: 当传送被激活时, 所有的数据能被接受并存入 FIFO。所有的数

## 手持式 POS 机

---

据接受是在后台进行的，所以不会丢失数据。请注意 FIFO 只有 1K 字节的空间，不要压入过多的数据，最好使用比起容量小数据包。如果 FIFO 是非空的，那么系统就会送出消息 MSG\_COMM\_DATA 或 MSG\_COMM2\_DATA。FIFO 能用 API 程序读取。

数据发送采用查询方式进行，会一直等到 UART 的传输缓存器为空再传送。

*相关的 API 函数：*

UART\_Send\_Char()

UART2\_Send\_Char()

UART\_Rev\_Char()

UART2\_Rev\_Char()

UART\_Send\_Str()

UART2\_Send\_Str()

### 5.3.3 状态查询

当 UART 通讯出错时，可以通过查询其状态来判断出差原因。

*相关的 API 函数：*

UART\_Stat ()

UART2\_Stat()

### 5.3.4 外围设备

即便是在没有数据传输时，UART 连接的外围设备也会给设备带来极大的负荷。因此不是绝对的需要它们时，请不要将其打开。

## 5.4 实时时钟(RTC)

关于 RTC 的 API 是很容易理解的。但是系统没有参数范围检查功能，所以用户必须设置正确的时间。

*相关的 API 函数：*

RTC\_Set\_Date()

RTC\_Set\_Time()

RTC\_Set\_DateTime()

RTC\_Get\_Date()

RTC\_Get\_Time()

RTC\_Get\_DateTime()

RTC\_Set\_Alarm()

RTC\_Get\_Alarm()

RTC\_Alarm\_Control()

RTC\_Alarm\_Status()

## 5.5 定时器

### 5.5.1 系统定时器

系统定时器的基本功能是作为一个独立延时装置。只要将时钟设置好，它就会自动的记时，一直到设定的时刻。如果将时刻设为零的话，相当于使其立即失效。

举个例子来说，如果您想让手持机连续 2 秒钟发出声音，您只要控制开启蜂鸣器，并设置周期时钟为 2 秒(128\*1/64)，然后设置当周期时钟到点后，在周期时钟到期处理程序中关闭蜂鸣器即可。

如果我们对精确性要求不高的话，可以在处理程序中设置下一次定时。这样就可以实现一个真正的周期时钟的功能。还是用上面那个例子来说明，时钟还是被设置为 2 秒钟，时间到的时候处理消息，但不再将蜂鸣器关闭了，而是让它在两种状态中切换，即蜂鸣 2 秒，再静音 2 秒，成为一个周期为 4 秒的循环。

目前有两个系统定时器，它们到达设定时刻时分别送出消息 time\_out 和 time\_out2。

**相关的 API 函数:**

Sys\_Timer\_Start()

Sys\_Timer\_Read()

### 5.5.2 回调

如果需要在定时器到来时马上执行某个任务，可以使用回调来实现。

回调函数必须在激活系统定时器之前注册。

定时器的重新设置可以实现为回调函数的一部分，这样就可以实现周期定时，回调函数本身就相当于一个时钟 ISR。在这种情况下，回调函数一定要在间隔时间到达之前返回，函数执行会交错在一起。

将回调函数注册为 NULL 可以使它失效。

**相关的 API 函数:**

Sys\_Timer\_Callback()

## 5.5.3 用户定时器

因为系统定时器数量较少,不适合于要使用很多定时器的场合(例如在程序有多重嵌套循环时)。这时可以使用用户定时器。

用户定时器的使用和系统定时器类似,但是在使用之前要申请,在使用之后要释放!

用户定时器共用一个 MSG\_USER\_TIME\_OUT 消息,而且必须使用 SYS\_Msg\_Wait\_Ext() 才能等待此消息。用户也可以不等待消息,直接读取定时器是否到期。

*相关的 API 函数:*

User\_Timer\_Get()

User\_Timer\_Start()

User\_Timer\_End()

User\_Timer\_Read()

## 5.6 其他功能设置

### 5.6.1 蜂鸣器

蜂鸣器是由一个独立的 API 函数来控制的,可以设置它的频率和持续时间。其频率是根据系统时钟而定的,因此如果系统时钟与默认值不同,频率也会随之有所不同。

蜂鸣器是在一个极窄的频率范围内工作的,因此任何超出频率的设置都会导致其发出的音频变得很低。它的峰值频率可以在 2.7KHz 到 3.1KHz 之间变化。

*相关的 API 函数:*

Beep\_Sound()

### 5.6.2 EL背光灯

背光灯子系统可以自动或手动开启。但无论它通过何种方式开启,它都能够在一段时间后自动关闭以达到节电的目的。这个延迟的时间可以通过 API 子程序来设置。每次打开背光灯(当按键按下后自动或手动的开启),延迟时钟都会重载。

*相关的 API 函数:*

EL\_Set()

EL\_Set\_Timeout()

5.6.3 LED指示灯（0818 only）

0818 POS 有三个 LED 指示灯，三个指示灯可以独立控制。

相关的 API 函数：

```
SYS_LED_Control()
```

5.7 存储管理

5.7.1 NorFlash空间分配图

手持机的存储空间分配图如下所示。三种类型的 Flash 可以使用，它们有相同的引脚分布，其存储空间的大小分别为：2Mbytes，4Mbytes，8Mbytes。

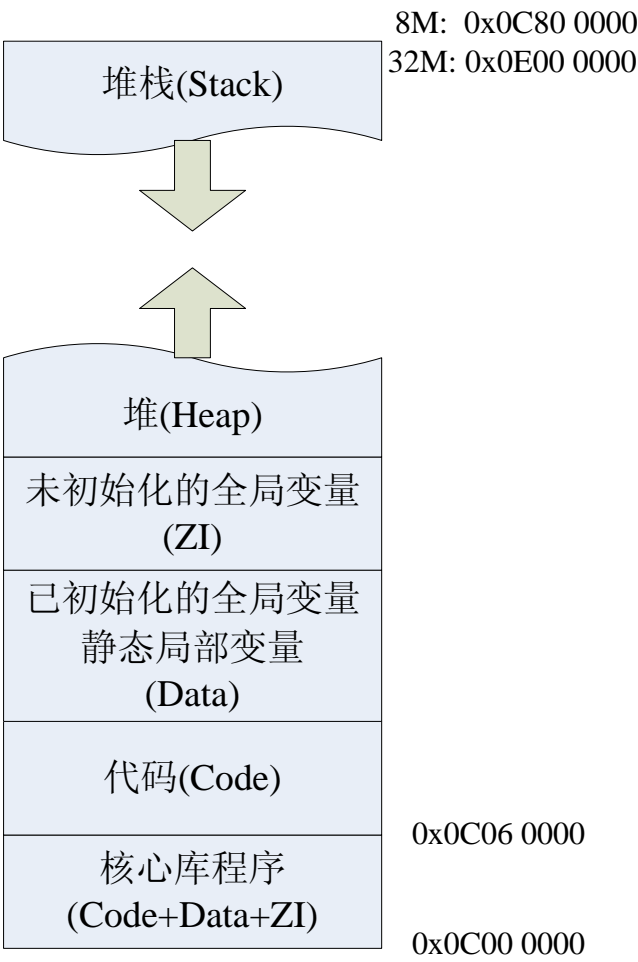


只有应用程序之后的 Flash 空间可以自由使用。我们提供的数据库函数也是操作这部分空间。

对于 0318/0518/0818，如果用户需要更大的空间，还可以选配 NandFlash 模块，最大可以达到 8G 字节。

5.7.2 SDRAM空间分配图

程序中配置有 8M 字节 SDRAM，可满足大多数应用程序的要求。必要时可以扩充至 32M 字节。



5.7.2.1 核心库程序

核心库程序占据了底层的 384K 空间。在其之上的空间是应用程序。

5.7.2.2 代码 (Code)

程序代码存放在 Flash 的代码部分。在运行时复制到 SDRAM 中以便加快执行速度。

5.7.2.3 数据 (Data)

数据 (Data) 部分包含所有已经初始化的全局变量和静态局部变量。

5.7.2.4 未初始化的全局变量 (ZI)

未初始化的全局变量集中在 ZI 段，在程序启动时自动初始化为 0。

# 手持式 POS 机

---

## 5.7.2.5 堆

堆是自由的存储区域,并未分配任何的静态存储内容。这个区域可以使用 malloc() 函数和 free() 函数来进行自由的分配。在使用时堆会自底向上生长。

考虑嵌入式开发的特点,建议用户尽可能少用动态分配。

## 5.7.2.6 堆栈

系统堆栈从内存顶部开始向下生长。由于内存空间足够大,所以系统有足够可用的堆栈空间。

# 5.8 系统消息

系统消息是内核向应用程序提示何种事件发生的唯一途径。为了丢失任何消息,应用程序必须检查这些消息。

## 5.8.1 消息轮询

系统消息存储于一个 16-bit 的字中。API 有消息轮询函数。消息轮询可以是类似于下面这个程序的循环:

```
while(1)
{
    typ_msg_word mw;
    // 检查消息,没有消息就进入睡眠状态
    mw.s_word = Sys_Msg_Wait(SM_GOTO_SLEEP);
    if (mw.bits.icc_move)
    {
        // 在这里放置卡片操作处理程序
    }
    if (mw.bits.charger_on)
    {
        // 在这里放置充电器插入处理程序
    }
    if (mw.bits.key_available)
    {
        // 在这里放置按键处理程序
    }
    // .....
    if (mw.bits.time_out) {
        // 在这里放置定时处理程序
    }
}
```

```
    }  
} //循环结束
```

*相关的 API 函数:*

`SYS_Msg_Wait()`

`SYS_Msg_Wait_Ext()`

### 5.8.2 消息

共有 13 种类型的系统消息:

<code>icc_move</code>	如有卡片插入, 则其值设为非零
<code>alarm_clock</code>	如时钟报警事件发生, 则其值设为非零
<code>comm_err</code>	如 UART1 或 UART2 通讯出错, 则其值设为非零
<code>comm_data</code>	如 UART1 的 FIFO 中有新的数据, 则其值设为非零
<code>comm2_data</code>	如 UART2 的 FIFO 中有新的数据, 则其值设为非零
<code>time_out</code>	如系统定时器 1 到达预设时刻, 则其值设为非零
<code>time2_out</code>	如系统定时器 2 到达预设时刻, 则其值设为非零
<code>charger_off</code>	如充电器断开电源, 则其值设为非零
<code>charger_on</code>	如充电器插入电源, 则其值设为非零
<code>key_available</code>	如有新的按键信息, 则其值设为非零 (在 FIFO 模式下)
<code>key_up</code>	如一个按键被松开, 则其值设为非零
<code>key_down</code>	如一个按键被按下, 则其值设为非零
<code>user_time_out</code>	如果指定的用户定时器到期, 其值非零

这些消息都可以按位进行屏蔽, 屏蔽后消息就不会送出了。

*相关的 API 函数:*

`SYS_Msg_Mask_Set()`

## 5.9 NorFlash

根据 5.7.1 的 NorFlash 分布图, 应用程序下载后, 一般还会有大量的 Flash 空间, 它们可以用来存储非易失性数据。

### 5.9.1 Flash操作API

API 仅为 Flash 提供了三个基本函数来进行 Flash 的写入和擦除。



*相关的 API 函数:*

`Flash_Get_Size();`

`FLASH_Erase_One_Sector()`

`FLASH_Erase_One_Block()`

`FLASH_Write_Record()`

## 5.9.2 写Flash驱动的注意事项

在手持机中，Flash 一个扇区是 4K 字节，一个块是 64k 字节。在写 Flash 驱动的时候请务必注意：Flash 不是 RAM(随机存取存储器)，只有在擦除之后的第一次写入是有效的。

我们提供了使用 NorFlash 的一个简单的数据库实现，具体请参考《数据库 API 参考文档》和相关的例子程序。

## 5.10 EEPROM读写函数

038/0518/0818 机型可以选配一个 EEPROM 模块。EEPROM 在掉电后不会丢失系统信息，所以可以用来存储一些系统参数，如 IP 地址等。

*相关的 API 函数:*

`EEPROM_Init()`

`EEPROM_Read_Byte()`

`EEPROM_Write_Byte()`

## 5.11 参数读写函数

有些机型没有配置 EEPROM，这时可以使用专门的参数读写函数保存和读取系统参数。虽然配有 EEPROM 的机器也可以使用这里的函数，但是参数读写函数使用的是系统的 Flash 空间，速度比读写 EEPROM 要慢一些。这里使用的 Flash 空间是内核的空间，没有占用系统的剩余可用空间。

*相关的 API 函数:*

`User_Param_Read()`

`User_Param_Write()`

## 5.12 NandFlash

如果使用的是 0318/0518/0818，根据用户需要，还可以添加大容量的 NandFlash 模块，它的容量从 32M 字节到 8G 字节不等。

*相关的 API 函数:*

`Nand_Read_ID()`

`Nand_Is_Bad_Block()`

`Nand_Read_Pages()`

`Nand_Read_Random()`

`Nand_Write_OnePage()`

`Nand_Erase_OneBlock()`

`Nand_Calculate_ECC()`

`Nand_Correct_Data()`

## 5.13 SIO端口

这个接口单元由系统驱动程序，编制应用程序的程序员要避免直接调用。

## 5.14 数据库接口

系统的附加库文件包含了一个简单的数据库系统，具体使用方式请参考《数据库 API 参考文档》。

## 5.15 扩展API

系统的附加库文件定义了一些用户常用的例程，如输入法，菜单选择和浏览等，使用这些例程可以加快用户开发应用程序的过程。具体请参考《扩展 API 参考文档》。

## 5.16 接触IC接口

手持机支持各种符合 ISO7816/EMV/PBOC 协议的存储卡和 CPU 卡，具体使用方式请参考《接触式 IC 卡 API 参考文档》。

### 5.16.1 卡片接口的电源开/关和初始化

通常卡片接口是关闭的，以达到节电的目的。如果将卡片接口打开，则电流值会升高。然而即便当卡片接口打开了，卡片插槽还是处于关闭状态的。

*相关的API 函数:*

`ICC_interface_power()`

## 5.16.2 卡片插槽的选择/关闭

您可以选择一个卡片插槽作为当前插槽(current-socket), 然后就可以调用那些对卡片进行操作的函数。这时上一个”当前”插槽和 MCU 卡的总线已经分离, 但如果卡仍然是开着的话, 它仍然会消耗电源, 所以在卡片切换后原卡的所有安全状态仍然保留。

*相关的 API 函数:*

ICC\_select\_sock ()

ICC\_get\_curr\_sock()

ICC\_deactivate\_current\_sock()

## 5.16.3 卡片的ATR

当系统进入主程序 main()后, 卡片接口就会关闭。ATR 函数可以打开卡片接口, 并且得到插入指定插槽中卡片的 ATR。为了节省时间和电量, 最好只使用指定的卡片类型。

*相关的 API 函数:*

ICC\_cpucard\_ATR()

ICC\_memcard\_ATR()

## 5.16.4 CPU卡(异步卡)的接口

在现有的库程序中, 支持两种类型的协议: T=0 和 T=1。

*相关的 API 函数:*

ICC\_T0\_TPDU()

ICC\_T1\_frame()

ICC\_T1\_APDU\_Exchange()

## 5.17 非接触卡操作

手持机支持各种符合 ISO14443/ISO15693/ICODE1 的非接触卡和标签, 具体使用方式请参考《ISO14443 协议 API 参考文档》和《ISO15693&ICODE1 协议 API 参考文档》。

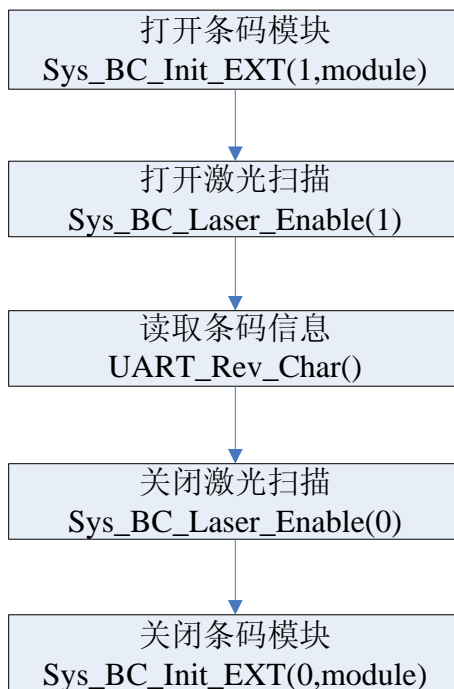
## 5.18 条码模块操作

目前只有 0518 连接有条码模块。条码模块使用了 0518 内部串口一。

## 手持式 POS 机

---

目前用户可以根据需要选用恒山或 Symbol 的条码模块。



### 5.19 GPRS模块操作

关于 GPRS 模块的使用，请参考<GPRS>目录下的相关文档。

### 5.20 433MHz无线模块操作

关于 433MHz 无线模块的使用，请参考<RF433>目录下的相关文档。

### 5.21 打印机模块操作

关于打印模块的使用，请参考《打印机模块 API 参考文档》。

### 5.22 网络模块操作

关于网络模块的使用，请参考<TCPIP>目录下的相关文档。

### 5.23 语音模块操作

关于语音模块的使用，请参考《语音模块 API 参考文档》。

### 5.24 磁卡操作

关于磁卡模块的使用，请参考《磁卡模块 API 参考文档》。