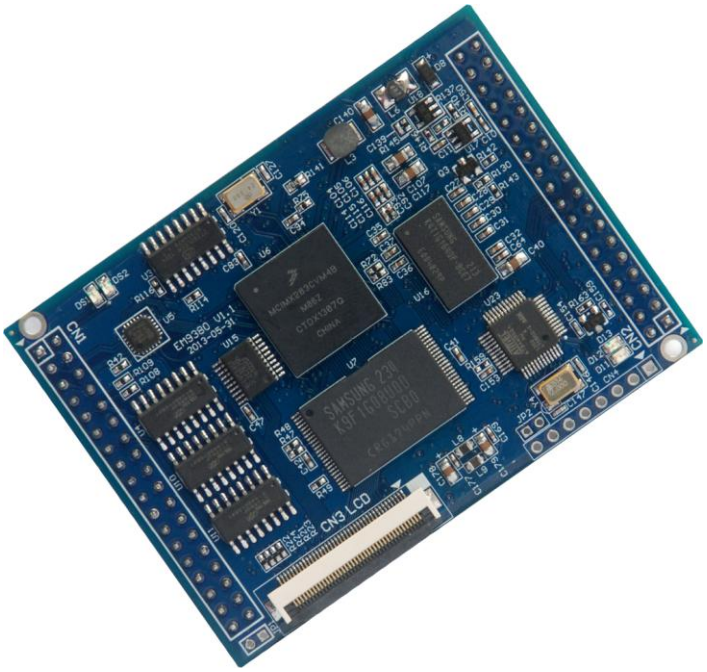
Emtronix



EM9380 工控主板数据手册

感谢您购买英创信息技术有限公司的产品：**EM9380 工控主板。**

传统的嵌入式工控主板通常是由单一CPU与Windows CE或Linux操作系统结合而成，由于操作系统任务调度机制所限，很难满足高速实时控制的应用需求。EM9380 则是针对实时控制的应用特点，在英创工控主板技术基础上，巧妙融入一片高性能 MCU 专门实现实时控制任务，为客户提供了一种高效低成本的实时控制应用的解决方案。EM9380 是一款双 CPU 配置的工控主板产品，其中主 CPU 为 454MHz 主频的 ARM9（FreeScale 的 iMX283），运行 Windows CE6.0（R3），实时控制 CPU 采用 64MHz 主频的 Cortex-M3（Atmel 的 SAM3S2A），CPU 之间通过高速 USB 相连接。从应用角度看，实时控制 CPU 是作为系统的一个专用协处理器，接收主 CPU 的指令，操作所属的硬件资源（数字 IO、PWM、AD 等），完成实时控制算法。在具体的应用程序设计中，实时控制协处理器对应一组特定的驱动程序 API 函数，应用程序通过调用这些 API 就可启动协处理器完成常规的数据采集控制任务。

对需要实现专用实时控制功能的应用，则需要编写在协处理器上的运行程序，其程序开发工具与主程序开发工具是一样的 Visual Studio。为了加快客户的实时控制程序的开发，英创公司还提供配套的范例程序框架。

与英创公司其他工控主板产品一样，EM9380 通过预装完整的 CE 操作系统及接口驱动，为用户构造了可直接使用的通用嵌入式核心平台。针对微软的 Visual Studio 开发平台，提供了完整的接口低层驱动以及丰富的应用程序范例，用户可在此基础上方便、快速地开发出各种工控产品。

EM9380 主要特点：

- **实时控制协处理器：**除了 454MHz 主频的主 CPU（ARM926EJ-S）外，EM9380 还配置了独立运行的协处理器（Cortex-M3），以支持专门的实时控制功能。其 25us 的实时控制周期，可满足大多数实时控制的需求。

- **超快速启动：**EM9380 一半的功能管脚（16 路）是通过硬件协处理器来操作的，而协处理器可在上电 1 秒后即处于工作状态，这意味着所控制的 16 位功能管脚可在 1 秒之内处于可控状态。这些功能管脚可实现的功能包括 GPIO、PWM 输出、AD 输入、脉冲计数等。

- **4 路 USB 接口：**为了适应未来嵌入式外设的发展趋势，EM9380 配备了 4 路 USB 接口，其中 3 路为 USB HOST 接口（主控），1 路为 USB OTG 接口（HOST/Device 自适应）。所有接口均为 USB2.0 高速接口。

- **支持多达 8 路标准串口：**EM9380 带有 8 路标准异步串口（UART）的工控主板产品，8 串口配置可满足目前工控产品绝大部分原因需求，从而加快客户整机产品的开发速度，同时降低其成本。

- **完备的标准接口资源：**除了 USB、串口外，为满足不同应用需求，EM9380 还配置了以下标准接口：（1）1 路 10M/100M 以太网接口；（2）1 路 I2C 接口总线；（3）1 路 SPI 接口；（4）4 路外部中断输入；（5）6 路 PWM 输出；（6）8 路 AD 输入；（7）32 位 GPIO。

- **紧凑的外形尺寸：**EM9380 的外形尺寸继续保持了经典的 74mm×53mm 规格，该规格是业界尺寸最小的 ARM9 工控主板之一，模块采用坚固的 IDC 插针，可非常方便的插入用户的产品底板上，快速搭建各种工控产品。

本手册详细介绍了 EM9380 的硬件配置、管脚定义及相关的技术指标，供用户使用时备查。此外，英创公司针对评估底板的使用编写有《EM9380 开发评估底板手册》。这两个手册都包含在英创为用户提供的产品开发光盘里面，用户也可以登录英创公司的网站下载相关资料的最新版本。用户还可以访问英创公司网站或直接与英创公司联系以获得 EM9380 的其他相关资料。英创信息技术有限公司联系方式如下：

地址：成都市高新区高朋大道 5 号博士创业园 B 座 404# 邮编：610041

联系电话：028-86180660 传真：028-85141028

网址：http://www.emtronix.com 电子邮件：support@emtronix.com

注意：本手册的相关技术内容将会不断的完善，请客户适时从公司网站下载最新版本的数据手册，恕不另行通知。

目　　录

1、主要技术指标

核心单元

- 454MHz 工业级 ARM9 作为主 CPU
- 配备 128MB 系统内存，用户可用空间大于 100MB
- 128MB FLASH 存储器，其中用户文件空间 75MB
- 预装 Windows CE6.0 实时多任务操作系统
- 采用 BinFS 文件系统，系统启动时间缩短至 7 秒水平
- 内置硬件协处理器，支持实时控制应用
- 25us 典型实时控制周期
- 实时时钟 RTC，具有掉电保护功能
- 硬件看门狗（WDT），防止系统死锁

USB 通讯接口

- 3 路 USB 高速主控接口（HOST），USB2.0 规范
- 1 路 USB OTG 接口，支持微软的 ActiveSync 及远程桌面功能
- 支持 WiFi 通讯模块、3G 通讯模块扩展

标准串口配置

- 8 路标准 UART 串口，其中 5 路为高速串口，波特率可达 3Mbps
- 各路串口基本特性如下：

串口名称	串口速度	功能简要说明
COM2	高速串口	支持 RTS/CTS 硬件流控。
COM3	高速串口	3 线制，RS232 电平接口。
COM4	高速串口	3 线制，TTL 电平。
COM5	高速串口	3 线制，TTL 电平。
COM6	高速串口	3 线制 TTL 电平。与 GPIO 复用管脚。
COM7	低速串口	3 线制，波特率不高于 19200bps，可作为 RS485 通讯应用。
COM8	低速串口	

COM9	低速串口	
调试串口	-	RS232 电平，固定 115200-8-N-1

显示单元

- TFT 彩色 LCD 接口（RGB 各 6-bit + 同步时钟信号）
- 支持显示分辨率：480×272 至 1024×768。
- 用户可配置开机画面。
- 支持 4 线制电阻触摸屏。

实时控制单元

- 16 位 GPIO（GPIO0 – GPIO16）支持实时控制。
- 4 路 PWM 脉冲输出，与 GPIO 复用管脚。
- 8 路单端 AD，可配置成 4 路差分输入，与 GPIO 复用管脚。
- AD 分辨率 12-bit，最高采样率 1Msps。
- 3 路内部定时器，支持实时控制、脉冲计数等功能。
- 实时控制功能的超快速启动（小于 1 秒）

常规接口功能

- 1 路以太网接口，10M/100M 自适应
- 支持 telnet、FTP、Web 等常规网络应用
- 1 路 I2C 接口，主控模式，最高波特率 400kbps，与 GPIO 复用管脚
- 1 路 SPI 接口，主控半双工模式，最高波特率 10Mbps，与 GPIO 复用管脚
- 2 路 PWM 输出，与 GPIO 复用管脚。
- 4 路外部中断触发功能，上升沿有效。

电源及模块机械参数

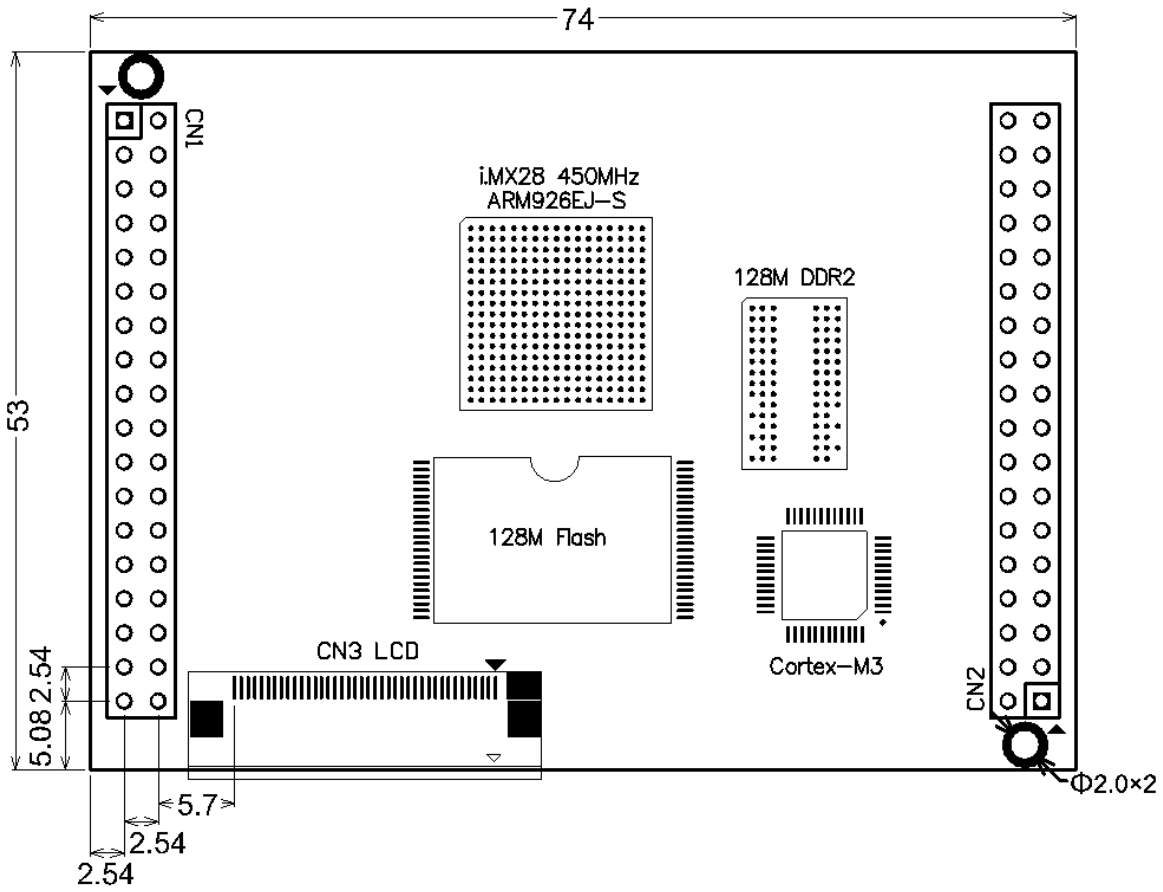
- 供电电压：+5V±5%，平均工作电流 260mA
- 工作温度：-10℃至 60℃；工业级（-40℃至 80℃）可选
- 支持对主板环境温度、供电电压的实时监测。

- 模块外形尺寸：74mm×53mm
- 2 个 36 芯坚固 IDC 双排插针（0.1”）对称分布于模块的两侧
- 独立 LCD 显示接口，ZIF40 插座，英创标准信号定义。

基本软件开发环境

- 提供相应 SDK 开发包，包括各种接口驱动程序 API
- Visual Studio 集成开发环境，同时支持应用程序和实时控制程序
- 支持以太网口（TCP/IP）、USB 口（ActiveSync）应用程序源码调试
- 提供典型应用参考程序源码
- 实时控制框架范例程序（可选），支持用户专用控制功能开发。

2、外形尺寸

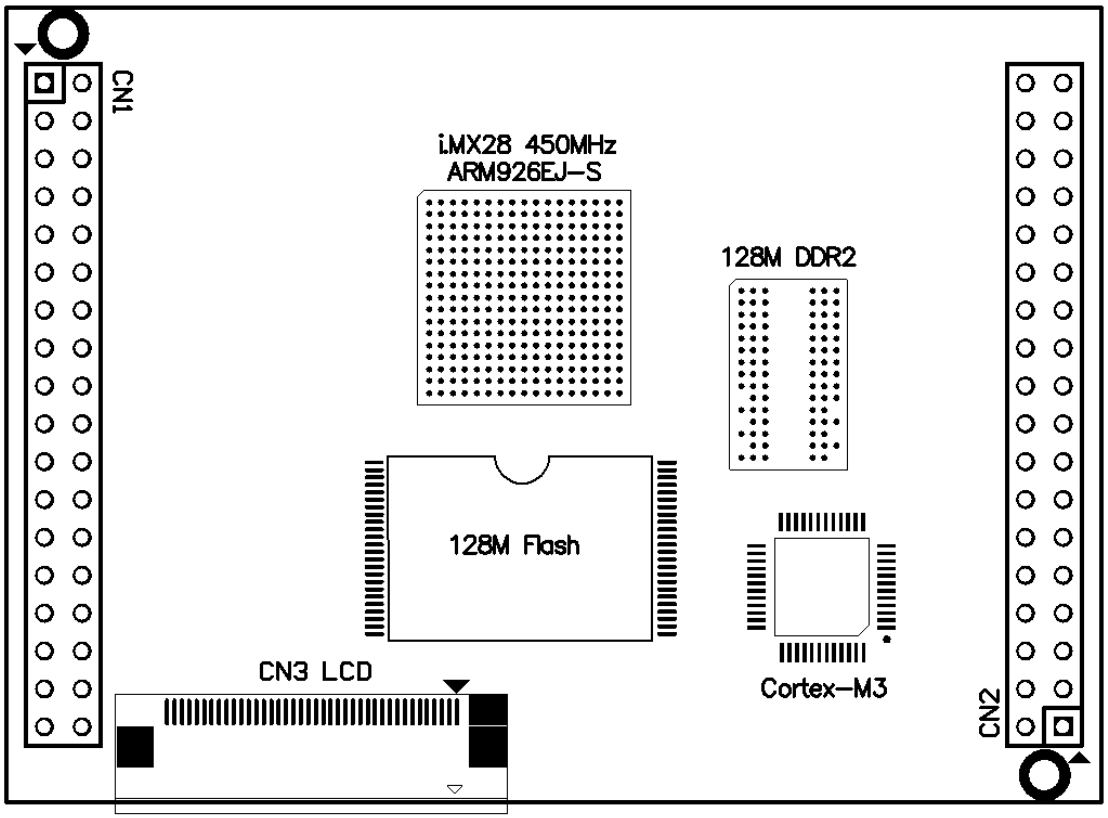


EM9380 外形尺寸示意图（单位 mm）

3、模块信号管脚功能描述

EM9380 的使用是以模块形式，通过板上的相关插针，插在应用主板上，同时实现 EM9380 板卡的固定以及与应用主板的信号连接两个功能。EM9380 共有 3 组信号插针，分别编号为 CN1、CN2 和 CN3，其中的 CN1 和 CN2 分别位于 EM9380 板卡模块的两端，为 2 组标准 0.1 英寸间距 IDC36 针双列直插管脚，EM9380 正是通过 CN1 和 CN2 与应用底板连接在一起的；EM9380 的 CN3 为 40 芯 ZIF 插座，主要引出彩色 LCD 及触摸屏的相关信号，实际应用中通过 40 芯的扁平软带线与 LCD 相连。为了方便起见，在后续对串口的说明中，均采用 CE 的串口名称（即 COM2…COM9）。

EM9380 所有管脚的信号电平，均为 LVTTL（3.3V）电平。除非特殊说明，输入管脚应避免接入 5V 电平信号。对低电平有效的信号，信号名称后均带“#”表示。



EM9380 的 CN1 – CN3 所在位置示意图

EM9380 的 CN1 主要包括以太网接口、异步串口、USB、GPIO 等信号；而 CN2 主要包括数字 IO、USB_OTG 端口、电源输入等信号。CN1 和 CN2 的管脚编号均为奇偶排交错顺序编号，且 1#管脚标志为方形焊盘。以下对 EM9380 所有管脚信号列表逐一说明。

3.1 EM9380 的 CN1 信号定义

CN1 主要包括以太网接口、异步串口、USB、GPIO 等信号如下：

PIN#	信号名称	方向	描述
1	LINK#	O	以太网 Eth0 连接 / 传送指示，低电平有效。
2	SPEED#	O	以太网 Eth0 速度指示，低电平有效。
3	TPTX+	O	以太网差分输出信号
4	TPTX-	O	以太网差分输出信号
5	TPRX+	I	以太网差分输入信号
6	TPRX-	I	以太网差分输入信号
7	VDD_MCT	O	以太网口的网络变压器信号公共端
8	-		系统保留
9	RXD7	I	COM7 数据输入，TTL 电平（3.3V）
10	TXD7	O	COM7 数据输出，TTL 电平（3.3V）
11	RXD8	I	COM8 数据输入，TTL 电平（3.3V）
12	TXD8	O	COM8 数据输出，TTL 电平（3.3V）
13	RXD9	I	COM9 数据输入，TTL 电平（3.3V）
14	TXD9	O	COM9 数据输出，TTL 电平（3.3V）
15	USB1_HD+	I / O	USB1 主控口的差分输入输出。
16	USB1_HD-	I / O	USB1 主控口的差分输入输出。
17	USB2_HD+	I / O	USB2 主控口的差分输入输出。
18	USB2_HD-	I / O	USB2 主控口的差分输入输出。
19	COM2_RXD	I	COM2 数据输入，TTL 电平（3.3V）
20	COM2_TXD	O	COM2 数据输出，TTL 电平（3.3V）
21	COM3_RX	I	COM3 数据输入，RS232 电平（±9V）
22	COM3_TX	O	COM3 数据输出，RS232 电平（±9V）
23	COM4_RXD	I	COM4 数据输入，TTL 电平（3.3V）
24	COM4_TXD	O	COM4 数据输出，TTL 电平（3.3V）
25	COM5_RXD	I	COM5 口数据输入，TTL 电平（3.3V）
26	COM5_TXD	O	COM5 口数据输出，TTL 电平（3.3V）

27	GPIO0	I / O	支持实时控制功能，复用定时器功能。
28	GPIO1	I / O	支持实时控制功能，复用定时器功能。
29	GPIO2	I / O	支持实时控制功能，复用定时器功能。
30	GPIO3	I / O	支持实时控制功能，复用定时器功能。
31	GPIO4	I / O	支持实时控制功能，复用 PWM3
32	GPIO5	I / O	支持实时控制功能，复用 PWM4
33	GPIO6	I / O	支持实时控制功能，复用 PWM5
34	GPIO7	I / O	支持实时控制功能，复用 PWM6
35	GPIO8	I / O	支持实时控制功能，复用 AD0 通道输入
36	GPIO9	I / O	支持实时控制功能，复用 AD1 道输入

关于 CN1 中相关信号的进一步说明：

为了提高管脚的利用率，以太网口的状态指示只提供单路低电平有效输出，需要外部提供 3.3V 偏置，才能点亮相应的 LED。为了提高整机的电磁兼容性能，通常情况下网络变压器应布局在客户应用底板上，且尽可能靠近网络的 RJ45 插座。

SD 卡接口：EM9380 标准版为 8 个通用串口，通过硬件配置，EM9380 可以引出一路 SD 卡接口，占用串口 7、8、9 对应管脚，即 EM9380 SD 卡版的配置为一路 SD 卡接口，5 路高速串口。EM9380 最大支持 32G 的 SD 卡，SD 卡信号占用 CN1 的管脚如下：

CN1 管脚	EM9380 标准版	EM9380 SD 卡版
8	系统保留	SD_DET*
9	RXD7	SD_CMD
10	TXD7	SD_SCK
11	RXD8	SD_D0
12	TXD8	SD_D1
13	RXD9	SD_D2
14	TXD9	SD_D3

* SD_DET 是 SD 卡插入检测信号，低电平表示 SD 卡插入。

EM9380 的异步串口，在 CE 系统中的编号从 COM2 开始，8 路串口分别为 COM2 – COM9，其中 COM6 配置在 CN2，且与 GPIO10 - GPIO11 复用管脚。其他的串口均在 CN1，且采用专用管脚引出，以突出 EM9380 串口的可用性，同时也提高了 EM9380 的 GPIO 的利用率。在 CE 操作系统中各串口名称既是对应串口驱动的设备文件名如下表所示：

串口名称	串口速度	功能简要说明
“COM2:”	高速串口	支持 RTS/CTS 硬件流控。
“COM3:”	高速串口	3 线制，RS232 电平接口。
“COM4:”	高速串口	3 线制，TTL 电平。
“COM5:”	高速串口	3 线制，TTL 电平。
“COM6:”	高速串口	3 线制 TTL 电平。与 GPIO 复用管脚。
“COM7:”	低速串口	3 线制，波特率不高于 19200bps，仅支持 8-bit 数据位，可作为 RS485 通讯应用。
“COM8:”	低速串口	
“COM9:”	低速串口	

EM9380 的串口分成两类，其中 COM2 – COM6 为高速串口，其波特率可达 3Mbps；COM7 – COM9 为低速串口，波特率为 1200bps – 19200bps，数据位固定为 8-bit，支持奇偶校验、MARK / SPACE 设置。所以 COM7 – COM9 更适合作为 RS485 使用。在工业现场中的 RS485 接口，当传输距离较长时，往往采用硬件方向控制，来提高通讯的抗干扰能力。EM9380 支持选择指定 GPIO 作为硬件方向控制功能。具体操作方法为：

1. 通过 DeviceIoControl 来指定具体作为 RTS 的 GPIO 管脚。以 GPIO24 为例

```
#include “bsp_drivers.h”

DWORD dwRTSPin = GPIO24;
bRet = DeviceIoControl(m_hSer,          // file handle to the driver
    IOCTL_SET_UART_RTS_PIN,             // I/O control code
    &dwRTSPin,                          // in buffer
    sizeof(DWORD),                      // in buffer size
    NULL,                               // out buffer
    0,                                  // out buffer size
    NULL,                               // pointer to number of bytes returned
    NULL);                              // ignored (=NULL)
```

2. 设置控制参数：

dcb.fRtsControl = RTS_CONTROL_TOGGLE

可用作硬件 RTS 方向控制的 GPIO 管脚有：GPIO16 – GPIO17；GPIO20 – GPIO31。若应用程序选择其他 GPIO 作为 RTS，设置函数将返回 FALSE。

EM9380 的运行状态设置：

EM9380 的 BGSL#设置信号与 COM5_TX 信号复用，在系统启动的瞬间，EM9380 会读入该信号，以配置系统的运行状态。

启动时，DBGSL#通过 5.1K 电阻接到地（状态为低），这时 EM9380 将进入调试状态：系统启动后，会自动复制 USB 盘中的 userinfo.txt 配置文件到 EM9380 的 NANDFLSH 下；DBGSL#悬空（状态为高），这时 EM9380 将进入运行状态，若此时文件 userinfo.txt 包含有效信息，客户的应用的应用程序将被系统启动。需要注意，DBGSL#不能直接对地短接，必须经过 5.1K 电阻后，才能与地短接，否则 COM5 功能不正常，甚至造成硬件损坏。

CN1 中的 GPIO 均支持实时控制：在 EM9380 CN1 中的 GPIO，均来自硬件协处理器，它们均支持实时控制功能，同时还复用其他功能。关于实时控制管脚的功能说明在本文第 5 章有更详细的说明。

下表是 CN1 更直观的简要说明

信号名称及简要描述	CN1		信号名称及简要描述
	PIN	PIN	
LINK#，Eth0 连接/传送指示	1	2	SPEED#，Eth0 速度指示
TPTX+，以太网差分输出	3	4	TPTX-，以太网差分输出
TPRX+，以太网差分输入	5	6	TPRX-，以太网差分输入
VDD_CMT1，网络变压器公共端	7	8	系统保留
ttyS6_RXD（COM7）	9	10	ttyS6_TXD（COM7）
ttyS7_RXD（COM8）	11	12	ttyS7_TXD（COM8）
ttyS8_RXD（COM9）	13	14	ttyS8_TXD（COM9）
USB1_HD+，USB1 Host 信号	15	16	USB1_HD-，USB1 Host 信号
USB2_HD+，USB2 Host 信号	17	18	USB2_HD-，USB2 Host 信号
COM2_RXD	19	20	COM2_TXD
COM3_RXD，232 电平	21	22	COM3_TXD，232 电平
COM4_RXD	23	24	COM4_TXD
COM5_RXD	25	26	COM5_TXD（DBGSL#）
GPIO0 / MCU_T0_IOA	27	28	GPIO1 / MCU_T0_IOB
GPIO2 / MCU_T0_CLK	29	30	GPIO3 / MCU_T1_IOA
GPIO4 / MCU_PWM3	31	32	GPIO5 / MCU_PWM4
GPIO6 / MCU_PWM5	33	34	GPIO7 / MCU_PWM6
GPIO8 / MCU_AD0	35	36	GPIO9 / MCU_AD1

- COM5_TXD 通过 5.1K 电阻接跳线器（跳线器另一端接地），构成 DBGSL#。跳线短接，系统按调试模式启动；跳线器断开，系统按运行模式启动。

3.2 EM9380 的 CN2 信号定义

EM9380 的 CN2 管脚，以数字 IO 作为其基本的功能，同时包括供电、USB 等功能如下表所示：

PIN#	信号名称	方向	描述
1	+5V	P	+5V 电源输入，输入范围+5V±5%
2	+5V	P	+5V 电源输入，输入范围+5V±5%
3	USB_OTG_VBUS	P	USB OTG 电源输入输出（+5V）
4	RSTIN#	I	外部复位输入，低电平有效
5	GND	P	公共地
6	GND	P	公共地
7	USB_OTG_D+	I/O	USB OTG 差分输入输出
8	USB_OTG_D-	I/O	USB OTG 差分输入输出
9	USB_OTG_ID	I	输入类型 ID，为低将使 OTG 端口作为 HOST
10	BATT3V	P	+3V 电池输入，用作 RTC 的后备电源
11	DBG_COM_RX	I	调试串口输入，RS232 电平（±9V）
12	DBG_COM_TX	O	调试串口输出，RS232 电平（±9V）
13	USB3_HD+	I / O	USB3 主控口的差分输入输出。
14	USB3_HD-	I / O	USB3 主控口的差分输入输出。
15	GPIO10	I / O	支持实时控制功能，复用 AD2 通道输入
16	GPIO11	I / O	支持实时控制功能，复用 AD3 通道输入
17	GPIO12	I / O	支持实时控制功能，复用 AD4 通道输入
18	GPIO13	I / O	支持实时控制功能，复用 AD5 通道输入
19	GPIO14	I / O	支持实时控制功能，复用 AD6 通道输入
20	GPIO15	I / O	支持实时控制功能，复用 AD7 通道输入
21	GPIO16	I / O	常规 GPIO，与 COM2 口的 CTS#复用管脚。
22	GPIO17	I / O	常规 GPIO，与 COM2 口的 RTS#复用管脚。
23	GPIO18	I / O	常规 GPIO，与 COM6 口的 RXD。
24	GPIO19	I / O	常规 GPIO，与 COM6 口的 TXD。

25	GPIO20	I / O	常规 GPIO，与“PWM1:”输出复用。
26	GPIO21	I / O	常规 GPIO，与“PWM2:”输出复用。
27	GPIO22	I / O	常规 GPIO，与 I2C 总线数据地址信号 SDA 复用。
28	GPIO23	I / O	常规 GPIO，与 I2C 时钟信号 SCL 复用。
29	GPIO24	I / O	常规 GPIO，与外部中断 IRQ1 复用，上升沿有效。
30	GPIO25	I / O	常规 GPIO，与外部中断 IRQ2 复用，上升沿有效。
31	GPIO26	I / O	常规 GPIO，与外部中断 IRQ3 复用，上升沿有效。
32	GPIO27	I / O	常规 GPIO，与外部中断 IRQ4 复用，上升沿有效。
33	GPIO28	I / O	常规 GPIO，与 SPI 接口 SPI_MISO 复用。
34	GPIO29	I / O	常规 GPIO，与 SPI 接口 SPI_MOSI 复用。
35	GPIO30	I / O	常规 GPIO，与 SPI 接口 SPI_SCLK 复用。
36	GPIO31	I / O	常规 GPIO，与 SPI 接口 SPI_CS0N 复用。

关于 CN2 中相关信号的进一步说明：

常规 GPIO，是指从主 CPU 引出的 GPIO，通常它们与一定接口功能复用管脚。当主板上电后，这些管脚均为 GPIO 输入模式，内部上拉电阻使其呈现高电平状态。当对应的接口的设备驱动文件被应用程序打开后，这些管脚将自由转换成相应的接口功能信号。比如打开设备文件“I2C1:”，则 GPIO22 自动转为 I2C 总线的数据地址线 I2C_SDA，而 GPIO23 自动转为 I2C 总线的时钟控制信号 I2C_SCL。EM9380 的 I2C 接口只作为主控口。

GPIO16	与 COM2_CTS#复用管脚，**dcb.fOutxCtsFlow** = TRUE 使能
GPIO17	与 COM2_RTS#复用管脚，**dcb.fOutxCtsFlow** = TRUE 使能
GPIO18	与 COM6 口的 RXD 复用管脚。COM6 设备文件名 = “COM6:”
GPIO19	与 COM6 口的 TXD 复用管脚。
GPIO20	与 PWM1 复用管脚。PWM1 设备文件名 = “PWM1:”
GPIO21	与 PWM2 复用管脚。PWM2 设备文件名 = “PWM2:”
GPIO22	与 I2C 总线的 SDA 复用管脚。I2C 设备文件名 = “I2C1:”
GPIO23	与 I2C 总线的 SCL 复用管脚。
GPIO24	与 IRQ1 复用管脚。IRQ1 设备文件名 = “IRQ1:”
GPIO25	与 IRQ2 复用管脚。IRQ2 设备文件名 = “IRQ2:”

GPIO26	与 IRQ3 复用管脚。IRQ3 设备文件名 = “IRQ3:”
GPIO27	与 IRQ4 复用管脚。IRQ4 设备文件名 = “IRQ4:”
GPIO28	与 SPI_MISO 复用管脚。SPI 设备文件名 = “SPI1:”
GPIO29	与 SPI_MOSI 复用管脚。
GPIO30	与 SPI_SCLK 复用管脚。
GPIO31	与 SPI_CS0N 复用管脚，片选控制，低电平有效。

支持实时控制的 GPIO，是指从硬件协处理器引出的 GPIO，这些管脚可通过驱动程序“MCU2:”实现常规的 GPIO、PWM、AD 采集、以及预置的若干实时控制功能。若用户的专用程序写入协处理器芯片，则这些管脚就可按照用户专用的实时算法工作。根据 AD 采集的模拟信号控制 PWM 脉冲宽度、根据主控下传的数据流控制步进电机、分析 AD 波形数据，存储上传故障波形等等都是典型的实时控制应用。

BATT3V 是板载 RTC 的后备时钟，其容量一般在 50mAH 至 150mAH 为宜。

RSTIN#为对板卡的复位输入，不用时，可悬空。

USB OTG 端口，EM9380 包含一个标准 USB OTG 接口，共 5 条引线：

USB OTG 接口定义	简要说明
USB_OTG_D+	USB OTG 双向差分数据线
USB_OTG_D-	USB OTG 双向差分数据线
USB_OTG_VBUS	双向电源
GND	公共地
USB_OTG_ID	连接类型标志，带上拉电阻。

上述 5 条引线可直接接到底板的微型 AB 插座(mini-AB)。在通常情况下，若连接带线使 USB_OTG_ID 变低（即微型 A 插头），则 EM9380 将作为主控端；若连接带线使 USB_OTG_ID 保持高电平（即微型 B 插头），则 EM9380 将作为设备端。在实际使用中，USB OTG 将通过主机通信协议（HNP）根据实际连接的设备类型，动态切换主机和设备角

色。因此即使 USB_OTG_ID 的电平与设备类型不符，同样可以实现正常连接。

当 EM9380 作为主控端时，将通过 USB_OTG_VBUS 向连接的 USB 设备提供+5V 电源，电流不超过 500mA。当 EM9380 作为设备端时，外部 USB 主控将通过 USB_OTG_VBUS 输入 5V 电源，但 EM9380 并不适用这个电源。

调试串口 DBG_COM，主要是用于输出系统的相关信息，在正常使用中不需要引出调试串口。但在开发阶段，调试串口的输出的信息是有帮助的。调试串口的电平为标准的 RS232 电平（±9V），波特率为 115200bps，数据帧格式为 8-N-1。

下表是 CN2 更直观的简要说明

信号名称及简要描述	CN2		信号名称及简要描述
	PIN	PIN	
+5V 电源输入	1	2	+5V 电源输入
USB_OTG_VBUS	3	4	RSTIN#，外部复位输入
电源地（GND）	5	6	电源地（GND）
USB_OTG_D+	7	8	USB_OTG_D-
USB_OTG_UID	9	10	BATT3V，3.3V 电池输入
DBG_COM_RX，232 电平	11	12	DBG_COM_TX，232 电平
USB3_HD+，USB3 Host 信号	13	14	USB3_HD-，USB3 Host 信号
GPIO10 / MCU_AD2	15	16	GPIO11 / MCU_AD3
GPIO12 / MCU_AD4	17	18	GPIO13 / MCU_AD5
GPIO14 / MCU_AD6	19	20	GPIO15 / MCU_AD7
GPIO16 / COM2_CTS#	21	22	GPIO17 / COM2_RTS#
GPIO18 / COM6_RXD	23	24	GPIO19 / COM6_TXD
GPIO20 / PWM1	25	26	GPIO21 / PWM2
GPIO22 / I2C_SDA	27	28	GPIO23 / I2C_SCL
GPIO24 / IRQ1	29	30	GPIO25 / IRQ2
GPIO26 / IRQ3	31	32	GPIO27 / IRQ4
GPIO28 / SPI_MISO	33	34	GPIO29 / SPI_MOSI
GPIO30 / SPI_SCLK	35	36	GPIO31 / SPI_CS0N

3.3 EM9380 的 CN3 信号定义

EM9380 的缺省显示模式为彩色 LCD 显示接口，CN3 插座主要是引出 LCD 显示输出信号以及引入触摸屏的模拟输入信号。具体信号定义如下：

PIN#	信号名称	方向	信号简要描述
1	GND	P	公共地
2	DCLK	O	串行像素时钟输出（Stream Pixel Clock）
3	HSYNC#	O	行同步脉冲，低有效
4	VSYNC#	O	场同步脉冲（或帧同步脉冲），低有效
5	GND	P	公共地
6-11	**R0 – R5**	**O**	**6-bit 红色分量输出信号，R0 为 LSB，R5 为 MSB。**
12	GND	P	公共地
13-18	**G0 – G5**	**O**	**6-bit 绿色分量输出信号，G0 为 LSB，G5 为 MSB**
19	GND	P	公共地
20-25	**B0 – B5**	**O**	**6-bit 蓝色分量输出信号，B0 为 LSB，B5 为 MSB**
26	GND	P	公共地
27	DE	O	显示使能控制信号
28-29	+3.3V	P	3.3V 电源输出，最大输出电流<200mA
30	BLIGHT#	O	背光控制信号，低电平有效；LCD 显示时有效。
31	-	O	输出固定高电平，系统保留。
32	GND	P	公共地
33-34	+5.0V	P	5V 电源输出，最大输出电流<200mA
35	GND	P	公共地
36	Xm	I	触摸屏 X 方向差分输入-
37	Xp	I	触摸屏 X 方向差分输入+
38	Ym	I	触摸屏 Y 方向差分输入-
39	Yp	I	触摸屏 Y 方向差分输入+
40	GND	P	公共地

关于 CN3 中相关信号的进一步说明：

- DCLK 下降沿更新 RGB 数据，上升沿用于显示设备锁存数据。
- LCD_PWR 信号也可用于 LCD 的背光电源控制。
- EM9380 支持的典型 LCD 显示格式包括：

  480×272，LCD 尺寸为 4.3”，具有很高的性价比；

  640×480，LCD 尺寸一般为 5.6” – 6.4”；

  800×480，LCD 尺寸为 7” – 8”；EM9380 缺省设置

  800×600，LCD 尺寸为 8.4” – 10.4”；一般需转为 LVDS 接口

  1024×768，LCD 尺寸为 10.4” – 12.1”；一般需转为 LVDS 接口
- 触摸屏的输入电阻一般在 200Ω 至 600Ω 这一范围。

4、基本电气特性与注意事项

在客户的应用设计中，EM9380 是作为整个系统的部件之一，与客户的应用底板、电源等其他部件协同工作的。因此在设计中，需详细了解 EM9380 各个管脚的电气特性，以做到系统各个部件间的各项指标的合理配合。

4.1 EM9380 的额定参数

参数名称	最小值	最大值	简要说明
+5V 直流瞬态输入	-0.3V	+7.0V	持续时间小于 30ms。
+5V 直流稳态输入	-0.3V	+7.0V	
GPIO 管脚输入电压	-0.3V	+3.63V	不兼容 5VTTL 电平输入。
GPIO/LCD 人体静电阈值	-	2kV	实际人体静电很容易超阈值。
CPU 基片工作温度	-40℃	85℃	应用程序可监测
CN3 插座电源输出功率	-	200mA	+5V 和+3.3V 二组电源输出
GPIO 信号总的驱动能力	-	±90mA	包括输入输出方式

4.2 RS232 输入输出特性

EM9380 的串口 COM3 缺省配置为 RS232 电平，其输入输出（RX / TX）特性如下表所示：

	Min（最小值）	**Max**（最大值）	简要说明
输入范围	-25V	25V	
输入负载	3kΩ	7kΩ	
输出电压	±5V	±9V	负载条件：3kΩ - 7kΩ

4.3 低速串口输入输出特性

低速串口 COM7 – COM9 的接口电平为 3.3VTTL，其 DC 电气参数如下表所示：

	Min（最小值）	**Max**（最大值）	简要说明
V_{IL}	0	1.0V	输入低电平

V_{IH}	2.3V	3.3V	输入高电平
V_{OL}	-	0.33V	输出低电平
V_{OH}	2.97V	-	输出高电平
I_{OH}	-4mA	-5mA	输出高电平时源电流
I_{OL}	4mA	10mA	输出低电平时吸电流

4.4 以太网口的基本参数

	典型值	简要说明
差分输出电压	2.0V	100BASE-TX 模式
差分输出电流	26mA	100BASE-TX 模式
差分输出电压	2.5V	100BASE-T 模式
VDD_MCT	3.3V	共模偏置电压，100Ω 终端电阻

4.5 3.3V TTL 信号的基本参数

EM9380 共引出 32 位通用数字 IO(也称为 GPIO),均为 3.3V TTL 电平。此外,EM9380 的 COM2、COM4、COM5 的 RXD 和 TXD 也为 3.3VTTL 电平信号，其 DC 电气特性与 EM9380 的 GPIO 是完全一致的。这些信号管脚的具体电气参数如下表所示：

	Min（最小值）	**Max**（最大值）	简要说明
V_{IL}	-	0.8V	输入低电平
V_{IH}	2.0V	3.3V	输入高电平
V_{OL}	-	0.4V	输出低电平
V_{OH}	2.4V	-	输出高电平
I_{OH}	-8mA	-	输出高电平时源电流
I_{OL}	8mA	-	输出低电平时吸电流
I_{IL}	-	10uA	输入低电平时的泄漏电流
I_{IH}	-	10uA	输入高电平时的泄漏电流

EM9380 的复位输出 RSTOUT#，也为 3.3VTTL 电平，其输出特性如下：

	Min（最小值）	**Max**（最大值）	简要说明

V_{OL}	-	0.25V	输出低电平，I_{OL} = 10mA
V_{OH}	2.8V	-	输出高电平，I_{OH} = -10mA

4.6 设计注意事项

1. EM9380 的核心 CPU 芯片 iMX283 内部还包含了一个电源管理单元，正是利用该电源管理单元使 EM9380 获得很高的性能价格比。对接入 EM9380 的+5V 电源有以下要求：电源上电时的电压过冲脉冲时间小于 30ms，同时脉冲的占空比小于 0.05%。例如，过冲脉冲的脉宽为 100us，则脉冲周期需大于 200ms。长时间过电压施加在 EM9380 上，可能造成核心芯片电源单元的损坏。
2. EM9380 上 CN1 – CN3 的大部分 LVTTL 信号均直接来自于系统的核心 CPU 芯片 iMX283，包括 GPIO 信号、LCD 的信号。它们抗人体静电的能力只有 2kV，这不是一个很高的阈值，冬季人体静电达到 4-5kV 是很容易发生的。
3. EM9380 的 GPIO 管脚不是 5V 输入兼容的，尽管在通电状态下接入个别 5V 电平信号不会影响系统工作。但若长时间接入 5V 信号，不能保证信号管脚不被损坏。此外在 EM9380 上电前，若接入 5V 电平信号的管脚较多，还可能会影响系统正常启动。
4. CN3 是 LCD 的专用插座，为了方便 LCD 屏的连接，CN3 上包含了+5V 和+3.3V 的电源输出，可满足大部分 LCD 屏的信号接口电路的需要。在安装扁平带线时，需特别注意管脚的一一对应及可靠的接触。信号管脚错位，可能会导致电源输出被短接，从而引起 EM9380 的损坏。
5. 尽管单个 GPIO 的驱动能力能够达到±8mA，但仍需在设计中应避免 GPIO 总的输入输出电流和超过额定驱动能力的阈值。长时间超阈值可能会导致 GPIO 管脚的损坏。对有可能存在超驱动能力阈值的应用，强烈建议在应用底板上增加驱动芯片（如 74HC245），通过把电流负载转移到驱动芯片上，来保护 EM9380 的 GPIO 管脚。
6. EM9380 的 USB 接口，在拔插过程中，会产生瞬间的浪涌电压，该电压有可能损坏 EM9380 的 USB 数据收发单元，因此强烈推荐客户的应用底板参考 EM9380 开发评估底板的相关电路，在 USB 接口处增加 ESD 保护芯片，并在电源回路中串入磁珠。

5、EM9380 的常规功能说明

5.1 WDT 看门狗定时器

EM9380 直接使用了 iMX283 芯片内部的独立看门狗定时器，系统启动后设置看门狗的超时时间为 10 秒，且由 WinCE 内核的 Watchdog 线程按 5 秒间隔对看门狗进行刷新。此模式可以防止应用程序占用 CPU 的死循环，但对应用程序异常退出或挂起没有作用。

EM9380 为应用程序设计了专门的 WDT 驱动程序，应用程序可通过打开 WDT 设备文件“WDT1:”来接管对看门狗的操作，使之更为全面的监管应用程序行为的有效性。应用程序接管看门狗后，需按 5 秒的间隔对看门狗进行刷新操作。用户应用程序可通过 Read 函数来获取 WDT 加载周期，通过 Write 函数执行看门狗刷新操作，主要代码如下：

打开 WDT 文件

```
HANDLE hWDT;
hWDT = CreateFile(_T("WDT1:"),                  // name of device
    GENERIC_READ|GENERIC_WRITE,                 // desired access
    FILE_SHARE_READ|FILE_SHARE_WRITE,           // sharing mode
    NULL,                                       // security attributes (ignored)
    OPEN_EXISTING,                              // creation disposition
    FILE_FLAG_RANDOM_ACCESS,                    // flags/attributes
    NULL);                                      // template file (ignored)
```

获取 WDT 刷新周期

```
DWORD     dwWDTPeriod;                          //in ms
DWORD     dwNumberOfBytesRead = 0;
BOOL      bRet;

bRet = ReadFile(hWDT, &dwWDTPeriod, sizeof(DWORD),
                  &dwNumberOfBytesRead, NULL);
```

执行 WDT 刷新操作

```
DWORD     dwNumberOfBytesWritten = 0;
BOOL      bRet;

bRet = WriteFile(hWDT, NULL, 0, &dwNumberOfBytesWritten, NULL);
```

5.2 USB 接口

EM9380 可提供 2 个 USB 端口：一个高速主控接口，和一个 USB OTG 接口。EM9380 的 USB 主控接口可直接与标准 U 盘相连，EM9380 会自动把 U 盘中的系统配置文件 userinfo.txt 拷贝到系统中，并按照 userinfo.txt 设置 IP 等参数，最后启动用户的应用程序。USB 主控口也可支持标准的键盘、鼠标等设备。EM9380 的 USB OTG 接口，即可作为 USB 主控接口使用，也可作为 USB 设备接口使用。作为 USB 设备接口的一个典型应用，就是支持 Microsoft 的 ActiveSync 传输协议，用户可利用它方便的实现对 EM9380 文件的管理，也可以利用 ActiveSync 来调试应用程序。另外 ActiveSync 还把 USB 设备口映射成串口，占用串口逻辑号 COM1，所以 EM9380 真正的物理串口对应的逻辑编号从 COM2 开始。主控 USB 的供电电路很简单，布置在 EM9380 的评估底板上，客户在设计自己的应用底板时，可参考该电路。

5.3 UART 异步串口

EM9380 物理上有 8 个串口， 8 个物理串口分别对应的逻辑编号为 COM2 – COM9，其中只有 COM6 口与 GPIO10 – 11 复用管脚，其他的 7 个串口均配置在 CN1 上，且具有独立使用的信号管脚。EM9380 的这种设计主要是充分发挥其多串口的功能，另一方面，由于串口与 GPIO 的复用很少，相应的也提高了 GPIO 的利用率。此外 EM9380 板上还保留了调试串口的引出插针。调试串口的波特率固定为 115200bps，帧格式则为 8-N-1，主要用于系统输出相关信息，以便于系统的维护，用户原则上可以不关心它。

EM9380 的 8 个串口按最高波特率分为高速串口 COM2 – COM6 和低速串口 COM7 – COM9。高速串口的最高波特率可达 3Mbps，而低速串口允许的波特率在 1200bps – 19200bps 之间，数据固定为 8-bit，支持奇偶校验、MARK / SPACE 设置。因此低速串口更适合作为 RS485 端口来应用。EM9380 在 RS485 驱动方面，除了可以采用 TXD 自动控制数据收发方向切换（具体电路请参考 EM9380 开发评估底板电路原理图）外，还可选择一位 GPIO 作为 RTS，实现硬件方向控制。可作为 RTS 硬件方向控制的 GPIO 有：GPIO6、GPIO7、GPIO20 – GPIO31。在应用软件方面，需要主要代码如下：

```
打开串口设备文件

HANDLE hSer;
hSer = CreateFile(_T("COM7:"),                  // name of device
    GENERIC_READ|GENERIC_WRITE,          // desired access
    FILE_SHARE_READ|FILE_SHARE_WRITE,// sharing mode
```

```
        NULL,                                   // security attributes (ignored)
        OPEN_EXISTING,                          // creation disposition
        FILE_FLAG_RANDOM_ACCESS,                // flags/attributes
        NULL);                                  // template file (ignored)
```

设置一位 GPIO 作为 RTS

```
DWORD dwRtsGpioPin = GPIO26;                    //选择 GPIO26 作为 RTS

If（!DeviceIoControl（hSer,
                        IOCTL_SET_UART_RTS_PIN,
                        & dwRtsGpioPin, sizeof(DWORD),
                        NULL, 0,
                        NULL, NULL））
{
    // 出错处理。。。
}
```

设置串口 RTS 控制模式

```
DCB    SerDCB;

SerDCB.DCBlength = sizeof(DCB);
GetCommState(hSer, &SerDCB );                   // 从驱动读取当前DCB
SerDCB.fRtsControl = RTS_CONTROL_TOGGLE;
SetCommState(hSer, &SerDCB );                   // 再设置回驱动
```

高速串口中，只有 COM2 配置有 RTS/CTS 硬件握手功能，而其他都是常规的三线制串口。由于 RTS/CTS 硬件握手功能的应用并不是很多，同时考虑充分利用 GPIO 的功能，在打开“COM2:”时，RTS/CTS 硬件握手功能并没有激活，而对应管脚 GPIO0、GPIO1 继续保持为 GPIO 状态。应用程序需通过设置才能激活 RTS/CTS 硬件握手功能：

激活串口 RTS/CTS 硬件握手功能

```
DCB    SerDCB;

SerDCB.DCBlength = sizeof(DCB);
GetCommState(hSer, &SerDCB );                   // 从驱动读取当前DCB
SerDCB.fRtsControl = RTS_CONTROL_HANDSHAKE;
SetCommState(hSer, &SerDCB );                   // 再设置回驱动
```

5.4 I²C 接口

EM9380 的 I²C 接口为 2 线制标准 I²C 接口，信号电平为 3.3V 的 TTL 电平（LVTTL），最高传输波特率为 400kbps。在使用 I2C 接口时，应对 SCL 和 SDA 两个信号线均加 10K 的上拉电阻，在高波特率的情况下，上拉电阻是必须的。EM9380 板上已固化了面向 I²C 接口的 WinCE 标准驱动程序，应用程序只需打开文件名为“I2C1:”的文件对象，就可通过标准的 ReadFile（…）和 WriteFile（…）函数进行 I²C 数据传输了。

基本的 I²C 数据结构如下：

```
typedef struct
{
    BYTE    uHwAddr;    // 7-bit I2C器件地址 + 1-bit 读写标志（LSB）
    DWORD dwCmd;        // 对I2C器件发送的命令，可选
    PBYTE   pDatBuf;    // 指向存储读写数据的buffer
    DWORD dwDatLen;     // 需要读写数据的字节长度
} I2C_INFO, *PI2C_INFO;
```

在上述结构中，dwCmd = 0xFFFFFFFF，表示无效命令，驱动程序会跳过命令的发送；若 dwCmd 的最高位（D31）= 0，表示为单字节命令，最低字节（D7 – D0）将作为命令被发送；若 dwCmd 的最高位（D31）= 1，表示为双字节命令，驱动程序会首先发送高字节（D15 – D8），然后再发送低字节（D7 – D0）命令。dwCmd 通常为 I²C 器件寄存器的起始地址。I²C 操作的主要代码如下：

打开 I²C 文件

```
HANDLE hI2C;
hI2C = CreateFile(_T("I2C1:"),                  // name of device
    GENERIC_READ|GENERIC_WRITE,                 // desired access
    FILE_SHARE_READ|FILE_SHARE_WRITE,           // sharing mode
    NULL,                                       // security attributes (ignored)
    OPEN_EXISTING,                              // creation disposition
    FILE_FLAG_RANDOM_ACCESS,                    // flags/attributes
    NULL);                                      // template file (ignored)
```

从 I²C 器件读取数据

```
I2C_INFO    I2cInfo;
DWORD       dwNumberOfBytesRead = 0;
BOOL        bRet;
```

```
// I2C 数据结构初始化，注意数据 buffer 指针一定要指向有效 buffer
// … …
bRet = ReadFile(hI2C, & I2cInfo, sizeof(I2cInfo),
                    &dwNumberOfBytesRead, NULL);
```

向 I^2C 器件写入数据

```
I2C_INFO    I2cInfo;
DWORD       dwNumberOfBytesWritten = 0;
BOOL        bRet;

// I2C 数据结构初始化，注意数据 buffer 指针一定要指向有效 buffer
// … …
bRet = WriteFile(hI2C, &I2cInfo, sizeof(I2cInfo),
                    &dwNumberOfBytesWritten, NULL);
```

用户可从 EM9380 的资料光盘中的 I2C 应用范例了解其详细的使用方法。

5.5 SPI 同步串口

EM9380 的 SPI 接口为 4 线制标准 SPI 接口，信号电平为 3.3V 的 TTL 电平（LVTTL），最高传输波特率为 10Mbps。主要应用于设备内部各功能单元之间的短距离高速传输。EM9380 板上已固化了面向 SPI 接口的 WinCE 标准驱动程序，应用程序只需打开文件名为“SPI1:”的文件对象，就可通过 SPI 进行数据传输了。用户可从 EM9380 的资料光盘中的 SPI 应用范例了解其详细的使用方法。

5.6 PWM 脉冲输出

EM9380 共有 6 路 PWM 输出，其中的 PWM1 和 PWM2 分别由独立的驱动程序控制，最高输出频率可达 12MHz，但如果希望保证一定精度的占空比（1%的精度），则输出最高频率只能到 240KHz。EM9380 板上已固化了面向 PWM 接口的 WinCE 标准驱动程序，应用程序只需打开文件名为“PWM1:”或“PWM2:”的文件对象，再通过 WriteFile 设置启动 PWM 脉冲的参数（频率和占空比）即可，应用程序也可通过 WriteFile 随时停止 PWM 的输出。典型的 PWM 应用，包括为红外串口提供调制信号（38.5KHz，50%占空比）、为 ISO7816 提供时钟信号（3.5712MHz，9600bps 波特率）。

基本的 PWM 数据结构如下：

```
typedef struct
{
    DWORD    dwFreq;          // 单位为Hz, = 0: 停止PWM输出
    DWORD    dwDuty;          // 单位为%, 1 - 99
    DWORD    dwResolution;    // = 1: unit; = 10: 0.1 unit; = 100: 0.01 unit
} PWM_INFO, *PPWM_INFO;
```

上述结构中，一般分辨率 dwResolution 选为 1，这时 dwFreq=1000000，就表示输出脉冲频率为 1MHz；若 dwResolution = 10，dwFreq=1000000 则表示 100KHz。PWM 操作的主要代码如下：

打开 PWM 文件

```
HANDLE hPWM;
hPWM = CreateFile(_T("PWM1:"),                  // name of device
    GENERIC_READ|GENERIC_WRITE,                 // desired access
    FILE_SHARE_READ|FILE_SHARE_WRITE,           // sharing mode
    NULL,                                       // security attributes (ignored)
    OPEN_EXISTING,                              // creation disposition
    FILE_FLAG_RANDOM_ACCESS,                    // flags/attributes
    NULL);                                      // template file (ignored)
```

执行 PWM 输出操作

```
PWM_INFO  PwmInfo;
DWORD     dwNumberOfBytesWritten = 0;
BOOL      bRet;

PwmInfo.dwFreq = 3571200;       // ISO7816 时钟：3.5712MHz
PwmInfo.dwDuty = 50;            // 占空比 50%
PwmInfo.dwResolution = 1;
bRet = WriteFile(hPWM, &PwmInfo, sizeof(PwmInfo),
                 &dwNumberOfBytesWritten, NULL);
```

获取 PWM 设置的实际参数

```
PWM_INFO  PwmInfo;
DWORD     dwNumberOfBytesRead = 0;
BOOL      bRet;

bRet = ReadFile(hPWM, & PwmInfo, sizeof(PwmInfo),
                &dwNumberOfBytesRead, NULL);
```

关闭设备文件，将停止 PWM 脉冲输出。

EM9380 的其他 4 路 PWM 的常规操作通过设备驱动程序“MCU2:”来实现，此外这 4 路 PWM 还支持实时控制应用。进一步的介绍请参见 5.9 节。

5.7 IRQ 外部中断

EM9380 共有 4 路外部中断输入 IRQ1 – IRQ4，分别与 GPIO24 – GPIO27 复用管脚。当应用程序打开 IRQ 驱动程序对应的设备文件“IRQ1:”- “IRQ4:”后，外部中断输入上升沿正脉冲，脉冲宽度大于 100us，驱动程序将响应该上升沿中断，并产生事件通知处于等待中的应用线程。典型代码包括：

打开 IRQ 文件

```
HANDLE hIRQ;
hIRQ = CreateFile(_T("IRQ1:"),                  // name of device
    GENERIC_READ|GENERIC_WRITE,          // desired access
    FILE_SHARE_READ|FILE_SHARE_WRITE,// sharing mode
    NULL,                                          // security attributes (ignored)
    OPEN_EXISTING,                          // creation disposition
    FILE_FLAG_RANDOM_ACCESS,          // flags/attributes
    NULL);                                         // template file (ignored)
```

等待 IRQ 时间子程序

```
DWORD WaitIRQEvent（HANDLE hIRQ, DWORD dwTimeoutMS）
{
    DWORD dwReturn = 0;

    If（!DeviceIoControl（hIRQ,
                          IOCTL_WAIT_FOR_IRQ,
                          &dwTimeouMS, sizeof(DWORD),   //超时时间单位为 ms
                          &dwReturn, sizeof(DWORD),
                          NULL, NULL））
    {
        //出错
        dwReturn = WAIT_FAILED;
    }
    Return dwReturn;
}
```

应用线程等待中断事件

```
DWORD dwTimeoutMS = 5000;          //超时时间设置为 5 秒
DWORD dwReturn;

dwReturn = WaitIRQEvent（hIRQ, dwTimeoutMS）;
if（dwReturn == WAIT_OBJECT_0）
{
    //外部中断发生，进行中断处理
    //… …
}
else if（dwReturn == WAIT_TIMEOUT）
{
    //超时处理
    //… …
}
else
{
    //出错处理
    //… …
}
```

5.8 常规 GPIO

EM9380 的 32 位 GPIO 分成两组，一组包括 GPIO16 – GPIO31，属于常规 GPIO，通过驱动程序“PIO1:”来操作。另一组包括 GPIO0 – GPIO15，通过驱动程序“MCU2:”来操作。GPIO0 – GPIO15 这组 GPIO 是从硬件协处理器引出，支持实时控制应用，进一步的介绍在 6.1 节给出。

常规 GPIO 的每位均为可独立设置其方向，若设置为输入模式，可读取管脚状态；若设置为输出模式，则可设置输出的高低电平。所有操作是在打开驱动程序“PIO1:”后，执行 DeviceIoControl 来实现。DeviceIoControl 的命令码如下：

```
GPIO_IOCTL_OUT_ENABLE      //设置GPIO为输出模式
GPIO_IOCTL_OUT_DISABLE     //设置GPIO为输入模式
GPIO_IOCTL_OUT_SET         //设置GPIO输出高电平
GPIO_IOCTL_OUT_CLEAR       //设置GPIO输出低电平
GPIO_IOCTL_PIN_STATE       //读取GPIO管脚状态
```

对 GPIO 操作的典型代码如下：

```
打开 PIO 文件

HANDLE hPIO;
```

```
hPIO = CreateFile(_T("PIO1:"),                  // name of device
      GENERIC_READ|GENERIC_WRITE,               // desired access
      FILE_SHARE_READ|FILE_SHARE_WRITE,         // sharing mode
      NULL,                                     // security attributes (ignored)
      OPEN_EXISTING,                            // creation disposition
      FILE_FLAG_RANDOM_ACCESS,                  // flags/attributes
      NULL);                                    // template file (ignored)
```

设置 GPIO16 和 GPIO17 为高电平

```
DWORD    dwGpios = GPIO16 | GPIO17;

if (!DeviceIoControl(hPIO,              // file handle to the driver
      GPIO_IOCTL_OUT_SET,               // I/O control code
      &dwGpios,                         // in buffer
      sizeof(DWORD),                    // in buffer size
      NULL,                             // out buffer
      0,                                // out buffer size
      NULL,                             // pointer to number of bytes returned
      NULL))                            // ignored (=NULL)
{
      //出错处理;
}
```

读取 GPIO24 管脚状态

```
DWORD    dwGpios = GPIO24;      //指定 GPIO
DWORD    dwState = 0;           //保存读取的状态值

if (!DeviceIoControl(hPIO,              // file handle to the driver
      GPIO_IOCTL_OUT_SET,               // I/O control code
      &dwGpios,                         // in buffer
      sizeof(DWORD),                    // in buffer size
      &dwState,                         // out buffer
      Sizeof(DWORD),                    // out buffer size
      NULL,                             // pointer to number of bytes returned
      NULL))                            // ignored (=NULL)
{
      //出错处理;
}
```

所有常规 GPIO 的上电初始状态均为输入状态带上拉电阻。一旦应用程序打开某个接口

的设备文件，则对应的 GPIO 功能将被禁止。注意即使应用程序关闭了设备文件，对应的 GPIO 功能同样是被禁止的。因为在嵌入式系统中，不可能存在一条管脚动态复用的情况。

5.9 板卡及芯片温度监测

EM9380 支持对输入的+5V 电源电压、EM9380 板卡本身运行的环境温度以及核心 CPU 基片温度的监测。应用程序可打开 AD 驱动程序对应的设备文件“LDC1:”后，可多次调用 ReadFile 来读取各类数据。需要设置的命令及参数如下：

```
#define  EM9380_DAQ_VDD_5V                  2
#define  EM9380_DAQ_CPU_TEMPERATURE         6
#define  EM9380_DAQ_BOARD_TEMPERATURE       7

typedef struct
{
    DWORD     dwCmd;          // 命令码 = 0, 1, 2, ....
    DWORD     dwData;         // 返回的AD数据
    char      UnitName[16];   // 返回的单位字串: "mV", "Kalvin"等
} DAQ_INFO, *PDAQ_INFO;
```

注意返回的温度参数均为开氏温度，转换成摄氏温度，大致减去 273 即可。在此基础上，应用程序的典型代码如下：

打开 AD 文件

```
HANDLE hLRADC;
hLRADC = CreateFile(_T("LDC1:"),                // name of device
    GENERIC_READ|GENERIC_WRITE,                 // desired access
    FILE_SHARE_READ|FILE_SHARE_WRITE,           // sharing mode
    NULL,                                       // security attributes (ignored)
    OPEN_EXISTING,                              // creation disposition
    FILE_FLAG_RANDOM_ACCESS,                    // flags/attributes
    NULL);                                      // template file (ignored)
```

读取板卡环境温度数据

```
DAQ_INFO   DaqInfo;
DWORD      dwNumberOfBytesRead = 0;
BOOL       bRet;

DaqInfo.dwCmd = EM9380_DAQ_BOARD_TEMPERATURE;  //读取板卡温度
dwNumberOfBytesRead = 0;
```

```
bRet = ReadFile(hLRADC, &DaqInfo, sizeof(DAQ_INFO),
                    &dwNumberOfBytesRead, NULL);
if(!bRet)
{
    //出错处理
    //… …
}

printf("T = %d(%s)\n", DaqInfo.dwData, DaqInfo.UnitName);
```

返回的数据 dwData 为开氏温度，大致减去 273 即为摄氏温度。EM9380 是利用二极管的温度特性进行测温的。由于二极管器件的离散性，精度不是很高，一般误差在±3 度。在实际应用中，一般只有当环境温度很高时（比如 45℃或以上），才有温度检测的意义。因此该功能对设备运行于高温环境是有积极意义的。

6、EM9380 的实时控制功能

从应用角度来看，EM9380 的实时控制功能，就是通过设备驱动程序“MCU2:”提供的 API 函数，来实现相关的功能。EM9380 的实时控制功能分成 4 个子类如下：

1. GPIO 子类，包括基本的 GPIO 操作，以及数字输入变化报警、数字输出控制流两项实时应用。
2. ADC 子类：包括基本 AD 数据采集，以及信号实时监测，获取报警单组数据或数据波形两项实时应用。
3. PWM 子类：4 路 PWM 可独立设置输出频率和占空比。
4. PID 子类：综合使用 EM9380 协处理器资源，实现多路经典 PID 实时控制应用。应用程序可设置 PID 的整定参数，以及输入输出通道。

以下对这 4 类应用做进一步说明。

6.1 GPIO 的实时控制

EM9380 的硬件协处理器控制 GPIO0 – GPIO15 这 16 位 GPIO 管脚，与常规 GPIO 一样（即 GPIO16 – GPIO31），每位均可独立操作，实现以下基本功能：

```
MCU_GENERIC_GPIO_OE          // 输出使能
MCU_GENERIC_GPIO_OD          // 输出禁止，作为数字输入
MCU_GENERIC_GPIO_SET         // 输出高电平
MCU_GENERIC_GPIO_CLR         // 输出低电平
MCU_GENERIC_GPIO_PIN         // 读取输入管脚电平状态
MCU_GENERIC_GPIO_OF          // 实时控制输出流
```

应用程序通过专门的数据结构来实现 GPIO 的命令：

```
typedef struct
{
    BYTE    ucSize;          // 本数据结构大小 = 17字节
    BYTE    ucCmd;           // GPIO命令码：MCU_GENERIC_GPIO_XXX
    DWORD   dwPins;          // 操作的管脚位
    DWORD   dwStatus;        // 返回的管脚电平状态
    DWORD   dwPeriod;        // 实时控制周期，单位us；= 0: 常规功能
    bool    bFlashed;        // 保存本配置作为启动缺省功能
    BYTE    ucChkSum;        // 校验和
} MCU_GPIO_INFO, *PMCU_GPIO_INFO;   // struct for GPIO
```

常规操作时，设置参数 dwPeriod 为 0。GPIO 的实时应用是在常规应用的基础上通过参数 dwPeriod 与基本命令相结合来实现的。

数字输入管脚电平实时监测，是监测指定的 GPIO 输入电平，当任何一位 GPIO 发生变化时，将产生数据提交应用程序。具体操作使用命令 MCU_GENERIC_GPIO_PIN，同时设置 dwPeriod 值，其最小值应大于等于 25us，表示硬件协处理器将按 dwPeriod 周期间隔监测 GPIO 输入电平。

数字输出控制流，是指应用程序输出数据，控制指定的 GPIO（最多不超过 8 位）按 dwPeriod 周期更新输出电平。本功能的典型应用是控制步进电机实现某一流程。具体操作是先通过命令 MCU_GENERIC_GPIO_SET 或 MCU_GENERIC_GPIO_CLR 来指定需要实时更新的 GPIO 位，注意实时控制输出的 GPIO 最多不超过 8 位，相应的设置这些 GPIO 的初始电平为高电平或低电平。数据结构中的 dwPeriod 表示输出数据的更新周期，其最小值应大于等于 25us。输出数据则需要通过专门的数据结构传送给协处理器：

```
typedef struct
{
    BYTE    ucSize;          // 本数据结构大小 = 64字节
    BYTE    ucCmd;           // 命令码：MCU_GENERIC_GPIO_OF
    BYTE    ucRawDat[60];    // 输出数据流，每字节对应一次更新
    BYTE    ucChkSum;        // 校验和
} MCU_GPIO_FLOW, *PMCU_GPIO_FLOW;     // struct for GPIO
```

通过命令 MCU_GENERIC_GPIO_OF 把数据按 60 个字节一组整批送给协处理器，这样可有更高的传送效率。ucRawDat 的数据格式，则根据 GPIO 的位编号，最低位对应最小的编号，最高位对应最大的编号。可通过范例程序来进一步了解实时控制的实现细节。

注意：GPIO 的两项实时监控功能，在实际应用中只能二选一，不能同时启动。

6.2 多通道 AD 数据采集

EM9380 的多通道 AD 与 GPIO8 – GPIO15 复用管脚。支持以下 7 种通道采集模式：

```
MCU_GENERIC_ADC_SE1      // 单端输入，1通道，使用GPIO8
MCU_GENERIC_ADC_SE2      // 单端输入，2通道，使用GPIO8 – GPIO9
MCU_GENERIC_ADC_SE4      // 单端输入，4通道，使用GPIO8 – GPIO11
MCU_GENERIC_ADC_SE8      // 单端输入，8通道，使用GPIO8 – GPIO15
MCU_GENERIC_ADC_DI1      // 差分输入，1通道，使用GPIO8 – GPIO9
MCU_GENERIC_ADC_DI2      // 差分输入，2通道，使用GPIO8 – GPIO11
```

```
MCU_GENERIC_ADC_DI4          // 差分输入，4通道，使用GPIO8 – GPIO15
```

单端输入时，使用公共地，每位 GPIO 对应一个 AD 通道：

GPIO#	信号公共端	对应的 AD 通道
GPIO8	GND	AD_CH0
GPIO9		AD_CH1
GPIO10		AD_CH2
GPIO11		AD_CH3
GPIO12		AD_CH4
GPIO13		AD_CH5
GPIO14		AD_CH6
GPIO15		AD_CH7

差分输入时，AD 转换每对 GPIO 的差值信号，GPIO 与 AD 通道的对应关系如下：

GPIO#	信号端	对应的 AD 通道
GPIO8	AD_CH0+	AD_CH0
GPIO9	AD_CH0-	
GPIO10	AD_CH2+	AD_CH2
GPIO11	AD_CH2-	
GPIO12	AD_CH4+	AD_CH4
GPIO13	AD_CH4-	
GPIO14	AD_CH6+	AD_CH6
GPIO15	AD_CH6-	

对单端信号每通道的量程范围为 0V - +2.5V，对差分信号，内部设有衰减，量程范围为 -2.5V - +2.5V。当管脚没有作为 AD 输入通道是，仍然可以作为 GPIO 使用。

应用程序需要通过以下数据结构，来实现 AD 数据采集的功能：

```
typedef struct
{
    BYTE    ucSize;          // 本数据结构大小 = 25字节
    BYTE    ucCmd;           // GPIO命令码：MCU_GENERIC_ADC_XXX
```

```
    DWORD dwPeriod;          // 采样周期，单位us；= 0: 常规单次采集
    WORD  wData[8];          // 返回的AD数据
    bool  bFlashed;          // 保存本配置作为启动缺省功能
    BYTE  ucChkSum;          // 校验和
} MCU_ADC_INFO, *PMCU_ADC_INFO;     // struct for ADC
```

参数 dwPeriod 为采样间隔，设置为 0 表示常规的数据采集，不为 0 为实时数据采集应用。最短采样间隔为 25us，即最高采样率 40Ksps。注意若启动多通道数据采集，每通道的采样周期为 dwPeriod×通道数。wData[8]为采集的数据，分别对应 AD 的 CH0 – CH7。wData 的数据格式为：

16-bit AD 转换数据	
D15 – D12（4-bit）	D11 – D0（12-bit）
通道号（0 – 7）	ADC 实际转换输出的数据值（0 – 4095）

单端输入时，数据值与输入电压的关系如下表所示：

AD 转换数据（HEX）	单端输入（SE）
4095（0xFFF）	+2.5V
2048（0x800）	
0（0x000）	0.0V

差分输入时，数据值与输入电压的关系如下表所示：

AD 转换数据（HEX）	CH+	CH-	备注
4095（0xFFF）	2.5V	0V	正向最大差
2048（0x800）	CH+ = CH-		
0（0x000）	0	2.5V	反向最大差

以下简要介绍 AD 数据的实时监测及故障数据捕获。

AD 数据实时报警，本应用按 dwPeriod 为周期采集指定通道数据，并判断 CH0 数据是否超出设定的阈值范围。若超出，该数据将提交应用程序。本实时应用的阈值参数由 wData 传入，其中 wData[1]为阈值窗口的上限值，若无上限值要求，该值设置为 0xFFF；wData[2]为阈值窗口的下限值，若无下限值要求，该值设置为 0。当 CH0 的数据超出阈值窗口，该

数据连同其他通道数据，将通过数据结构 MCU_ADC_INFO 传送给应用程序。

故障波形采集，本应用也是按 dwPeriod 为周期采集指定通道数据，并判断 CH0 数据是否超出设定的阈值范围。当出现超阈值数据，表示出现异常情况，本功能将把异常出现前后一段时间的数据波形提交应用程序。提交的数据长度由 wData[0]来指定。波形数据使用以下数据结构，以提高传送效率：

```
typedef struct
{
    BYTE    ucSize;          // 本数据结构大小 = 52字节
    BYTE    ucCmd;           // 命令码：MCU_GENERIC_ADC_XXX
    WORD    wRawDat[24];     // 波形数据流
    BYTE    ucChkSum;        // 校验和
} MCU_ADC_FLOW, *PMCU_ADC_FLOW;  // struct for ADC
```

注意 wData[0]为样点的总数，应确保每通道的采样点至少要大于 2，否则就不能构成波形了。wData[0]的上限值为 2048。与数据实时报警功能一样，wData[1]和 wData[2]构成窗口的上下限，当 AD 数据超出窗口上限或下限时，就触发波形扑捉。当设置的 wData[1]等于 wData[2]时，就变成普通的波形数据采集。

6.3 PWM 基本应用

一般来说，PWM 作为输出通道，需要结合具体的应用才能构成实时控制功能。在驱动程序“MCU2:”中只是实现了其基本功能。EM9380 的硬件协处理器控制 PWM3 – PWM6 这 4 路 PWM，可通过以下数据结构独立设置各通到的输出频率，占空比等参数：

```
typedef struct
{
    BYTE    ucSize;          // 本数据结构大小 = 17字节
    WORD    wCmd;            // PWM命令码：MCU_GENERIC_PWM_CH#
    DWORD   dwFrequency;     // 输出频率，单位Hz
    DWORD   dwDuty;          // 输出占空比 = 0 – 1000，单位0.1%
    DWORD   dwPolarity;      // 输出初始电平；= 0: 低电平，= 1: 高电平
    bool    bFlashed;        // 保存本配置作为启动缺省功能
    BYTE    ucChkSum;        // 校验和
} MCU_PWM_INFO, *PMCU_PWM_INFO;   // struct for PWM
```

PWM3 – PWM6 输出的最高频率为 50KHz。

6.4 PID 实时控制

PID 控制因其结构简单、稳定性好、工作可靠、调整方便而成为工业控制的主要技术之一，EM9380 的硬件协处理器支持了两路位置式 PID 的时实控制：PID1、PID2，采用了比例带控制、一阶惯性滤波、积分分离等 PID 控制算法。PID 控制是典型的闭环控制逻辑，综合使用了 EM9380 硬件协处理器上的 AD、IO、PWM 等资源。PID1、PID2 占用的硬件资源如下：

GPIO#	PID 通道功能
GPIO8	PID1 模拟量输入
GPIO9	
GPIO0	PID1 下限报警输出
GPIO1	PID1 上限报警输出
GPIO4	PID1 控制输出
GPIO10	PID2 模拟量输入
GPIO11	
GPIO2	PID2 下限报警输出
GPIO3	PID2 上限报警输出
GPIO5	PID2 控制输出

EM9380 的两路 PID 控制参数可独立设置，采样周期为 250ms（4Hz）,PID 的控制参数仍然通过驱动程序”MCU2:”传递，命令字为：MCU_GENERIC_PDI#，其数据结构如下：

```
typedef struct
{
    uint8_t     ucSize;          // 本数据结构大小 = szieof(MCU_PID_SETUP)
    uint8_t     ucCmd;           // PID命令码：MCU_GENERIC_PID#
    float       fSP;             // 设定值
    float       fPVDHi;          // 过程值显示上限
    float       fPVDLo;          // 过程值显示下限
    float       fPVAHi;          // 过程值上限报警
    float       fPVALo;          // 过程值下限报警
    int16_t     wP;              // 比例增益：0~32767(%); P=0：开关控制
    int16_t     wI;              // 积分时间：0~32768(×100ms)； I=0：无积分
    int16_t     wD;              // 微分时间：0~32768(×100ms)；D=0：无微分
    uint8_t     ucControlCycle;  // PID控制周期(0~255s); =0:PWM输出(200Hz）
    uint8_t     ucOutHiLimit;    // PID输出量上限幅(0~99%)；=0：无限制
```

```
    uint8_t     ucOutLoLimit;   // PID输出量下限幅(0~99%)；=0：无限制
    uint8_t     ucDeadBand;     // PID不灵敏死区(0~255)
    uint8_t     ucHysteresis;   // 过程值报警回差滞环(0~255)
    PID_ACT     sPidACT;        // PID动作方向
    uint8_t     ucSersorType;   // 传感器类型（输入信号类型）
    uint8_t     ucChkSum;       // 校验和
}MCU_PID_SETUP, *PMCU_PID_SETUP;
```

MCU_PID_SETUP 结构中的 PID_ACT 是定义 PID 动作方向的位联合，其定义如下：

```
typedef union
{
    uint8_t     U;
    struct
    {                                                       0       1
        unsigned PID_DIRECTION          : 1;    // PID动作方向  正动作   反动作
        unsigned PID_AUTO_REPORT        : 1;    // 自动状态报告  否      是
        unsigned DIFFERENCE_INPUT       : 1;    // 差分输入     否      是
        unsigned INTELLIGENT_ALARM      : 1;    // 智能报警     否      是
        unsigned ALARM_MODE             : 3;    // 报警模式
        unsigned RSRVD                  : 1;    // 保留
    }B;
}PID_ACT;
```

一旦设置了 PID 参数，相应的通道就开始运行。如果使能了自动状态报告,EM9380 协处理器将周期性（与采样同期相同）的将 PID 通道的状态发送给”MCU2:”驱动程序,应用程序也可以通过 MCU_GENERIC_PID#_STATUS 命令主动读取当前 PID 通道状态。PID 状态的数据结构定义如下：

```
typedef struct
{
    uint8_t     ucSize;         // 本数据结构大小= szieof(MCU_PID_STATUS)
    uint8_t     ucCmd;          // PID命令码：MCU_GENERIC_PID#_STATUS
    float       fPV;            // 过程值
    float       fPIDOutPercent; // PID控制量
    PID_STATUS  sStatus;        // PID状态
    uint8_t     ucChkSum;       // 校验和
}MCU_PID_STATUS, *PMCU_PID_STATUS;
```

MCU_PID_STATUS 结构中的 PID_STATUS 是定义 PID 状态的位联合，其定义如下：

```
typedef union
{
    uint8_t     U;
```

```
        struct
        {                                                          0       1
            unsigned PID_IS_RUN         : 1;    // PID是否运行     否      是
            unsigned ALARM_ENBALE       : 1;    // 允许报警        否      是
            unsigned PVLO_ALARM         : 1;    // 下限报警        无报警  报警
            unsigned PVHI_ALARM         : 1;    // 上限报警        无报警  报警
            unsigned RSRVD              : 4;    // 保留
        }B;
    }PID_STATUS;
```