FCUSB－CY7C68013-56 开发套件
常见问题

星烁工控科技
网址：www.flickercontrol.com
技术支持邮箱：support@flickercontrol.com
销售邮箱：sales@flickercontrol.com

版本

版本	日期	备注
V1.0	**2004-12-01**	**FCUSB-CY7C68013-56 开发文档补充说明**

1. 固件下载不能正常运行

CYPRESS 开发包安装完成之后，会要求选择开发板类型，需要选择 FX2，如果开始选择错了，见图 1 中 Target 下拉框选择 FX2，关闭，再 Open All 重新打开设置。重新下载固件，则运行正常。

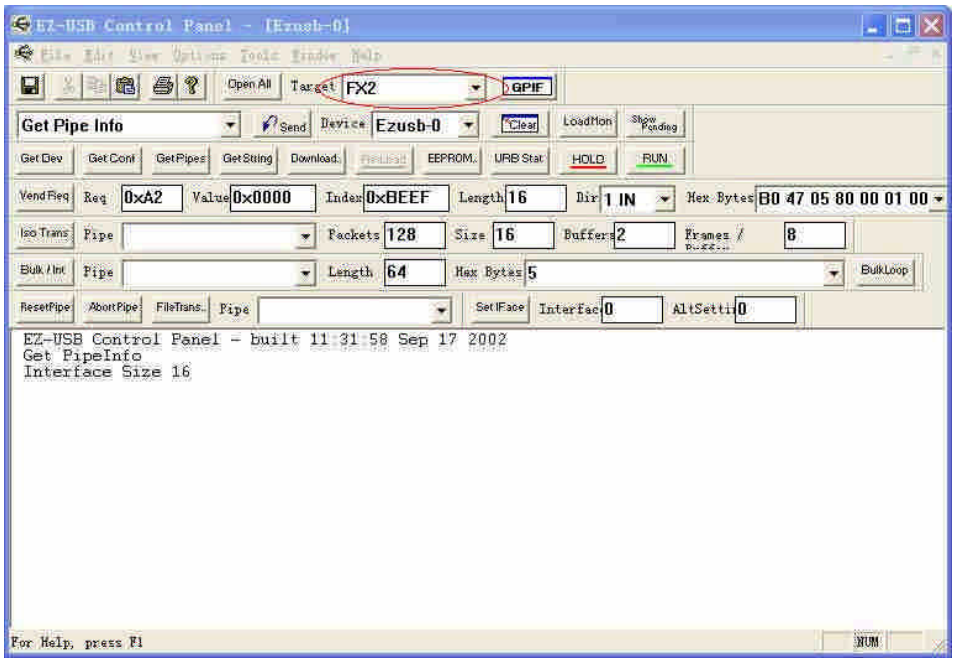


图 1 目标设备选择

2. 本开发板运行 C:\Cypress\USB\Examples\FX2 例子程序不能正常工作

本开发板只有内置的 8.5KB XRAM，而 CYPRESS 例子程序有些地址空间设置在片外，见图 2，需要重新设置，如 Code Range 设为 0x80，Xdata Range 设为 0x1000，具体设置需要根据实际程序 CODE 和 DATA 大小来确定。

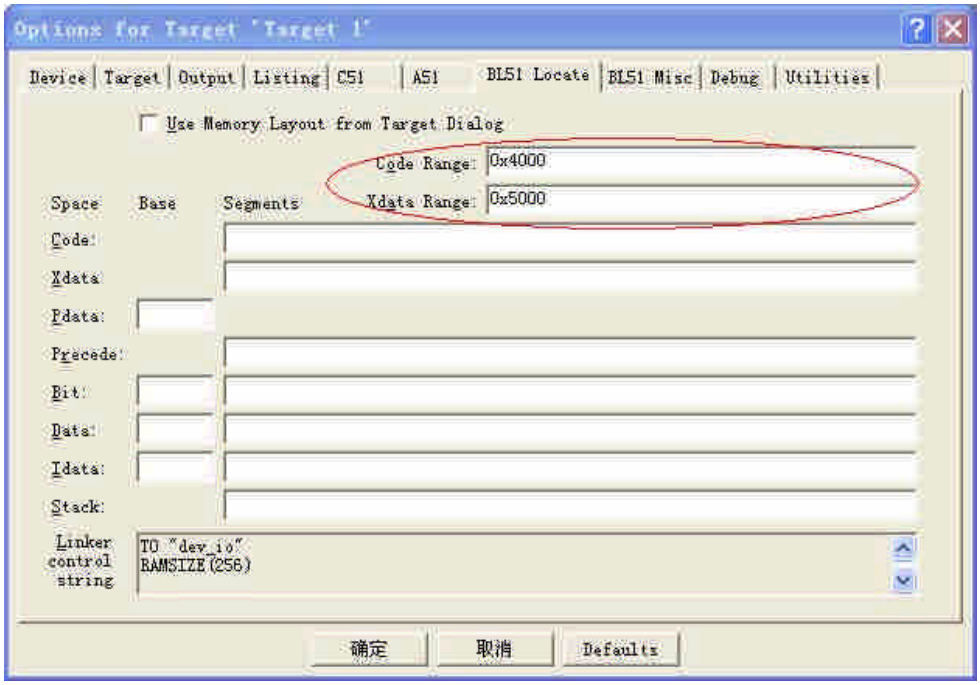


图 2　KEIL 地址空间设置

3. E2PROM 不能正常读写

需要确定 AT24LC02 跳线帽是否装好，A0 引脚是否通过与地短接。

4. 如何改写 E2PROM

如果 E2PROM 中数据为“C0”下载方式，则每次插入开发板提示新的设备，要求安装驱动，则只需要将 AT24LC02 跳线帽取掉，则用缺省方式来响应。等上电完后，再将跳线帽装好，则下载固件 Vend_Ax.hex，在控制面板中修改 E2PROM。

5. 复位按键不能正常复位

因为芯片原因，复位按键按下后芯片无法正常工作，需要重新插拔。通过控制面板每次 Download 固件后，芯片会自动复位和运行，所以无需按下复位键。

FCUSB－CY7C68013-56 开发套件
用户手册
版本 1.0

星烁工控科技
网址：www.flickercontrol.com
技术支持邮箱：support@flickercontrol.com
销售邮箱：sales@flickercontrol.com

版本历史

版本	日期	备注
V1.0	**2004-10-08**	初版

目 录

第一章 USB协议简介

1.1 USB1.1 协议

1.1.1 USB 系统组成

首先了解 USB 设备组成，并带着这些问题去阅读。USB 系统的三个组成：HOST、HUB 和 Device。

HOST：主控器，PC 端的就是 HOST 了，如果我们在 ARM 芯片或者单片机上加上一个 HOST 芯片就可以读写 U 盘和其它的 USB DEVICE 了。

HUB：HOST 只有一个 USB 口，如果要同时使用多个 USB DEVICE，HUB 可以将一个 USB 口扩充多个 USB 口，市场上都有卖。

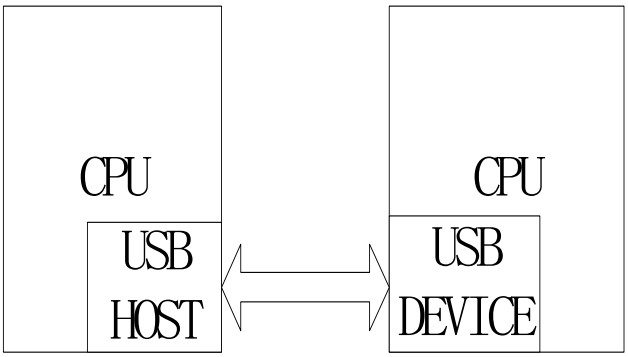


图 1.1 USB 最简单系统组成

DEVICE：似乎大家对这个最为亲切，因为我们接触最多就是 USB DEVICE，例如移动硬盘、打印机、U 盘等。

典型的 USB 系统都是由三者构成的，实际上主要是由 HOST 和 DEVICE 组成，最简洁的 USB 系统构成如图 1.1 所示，

其中 CPU 可以是 PC 机、单片机、ARM、MIPS、COLDFIRE、POWERPC ••••••••••

1.1.2 USB 设备组成

一般，如图 1.2 所示，每个 USB 设备由一个或多个配置（configuration）来控制其行为，使用多配置原因是对操作系统的支持；一个配置中是由接口（Interface）组成；接口则是由管道（Pipe）组成；管道是和 USB 设备的端点（Endpoint）对应，端点都是输入输出成对的。在固件编程中，USB 设备、配置、接口和管道都有描述符来报告其属性。

虽然图中配置、接口、端点很多，但是一般使用的时候配置和接口我们都只设置一个，根据数据传输的实际情况，来选择用哪个端点，每个芯片的端点数都是一定的，例如 AN2131 有 32 个，68013 有 7 个端点，而实际工程中可能采用到其中的几个。

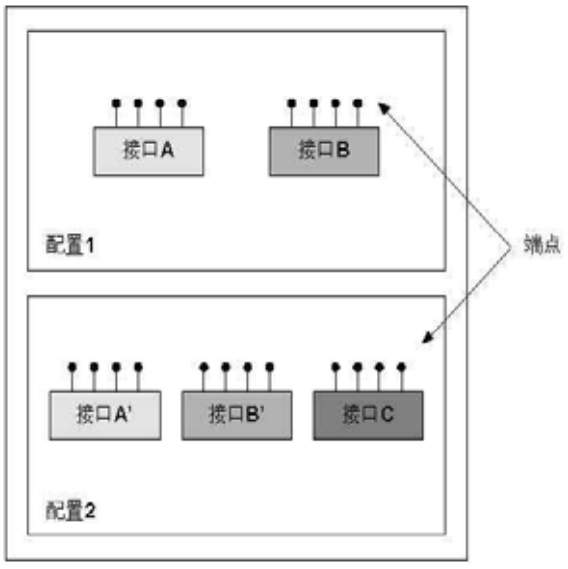


图 1.2 USB 设备组成

端点有了，就可以建立数据传输的管道，一个端点建立一个通道。一般管道的端点总是成对出现，一个 IN（DEVICE->PC），一个 OUT(DEVICE<-PC)。

如图 1.3，端点 0 默认配置为控制管道，用来完成所规定的设备请求，设备请求详见 USB 协议第九章。其它端点可配置为数据管道，对开发而言，主要的大数据传输都是通过数据管道来完成的。

用户需要根据实际数据传输速度来规定数据管道的传输类型。同时，每种数据传输都必须根据数据请求的格式来进行。

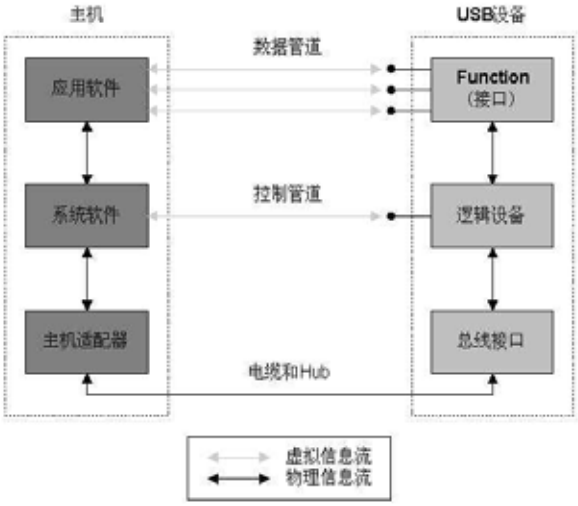


图 1.3　USB 多层次通讯模型

1.1.3 USB 数据传输类型

USB 传输类型包括批量传输、等时传输、中断传输和控制传输，如表 1.1 所示。

表 1.1　各种数据传输的相关特性（仅限 USB1.1 协议）

传输模式	中断传输(Interrupt)	批量传输(Bulk)	等时传输(ISO)	控制传输(Control)
传输速率/Mbps	12（1.5，低速）	12	12	1.5/12
数据的最大长度/字节	1～64（1～8 低速）	8/16/32/64	1～1023	1～64（1～8 低速）
数据周期性	有	没有	有	没有
发生错误重传	可	可	不可	可
应用设备	鼠标、键盘和摇杆	打印机，扫描仪	语音	
可得到的最大带宽/Mbps	6.762（0.051 低速）	9.728	10.240	

端点 0 只能配置为控制传输类型的管道，其它端点传输类型选择则比较灵活，控制传输可靠性是最高的，但速度最慢，等时传输速度快和满足实时性，但可靠性低，传输类型选择的原则是根据工程应用的传输速度和可靠性。例如，语音传输，要求的高速度和实时性，对于丢失帧的情况也可以接收，所以选择等时传输类型；打印机要求的速度较快，而且不能出现数据丢失，综合传输速度和可靠性，所以选择批量传输类型。关于具体的每种传输类型详细说明，可以参阅 USB 协议。

1.1.4 USB 数据请求格式

USB 设备无法枚举成功，往往是标准请求无法正确执行。

在 USB 通信协议中，因为主机取得绝对主动权利，设备只能是“听命令行事”，所以通过一定的命令格式（设备请求）来完成通信。USB 设备请求包括标准请求、厂商请求和设备类请求。设备的枚举是标准请求命令来完成的；厂商请求就是用户定义的请求；设备类请求是特定的 USB 设备类发出的请求，例如海量储存类、打印机类和 HID（人机接口）类。

设备请求必须遵循一定的格式，包括请求类型、设备请求、值、索引和长度，如图 1.4 所示。

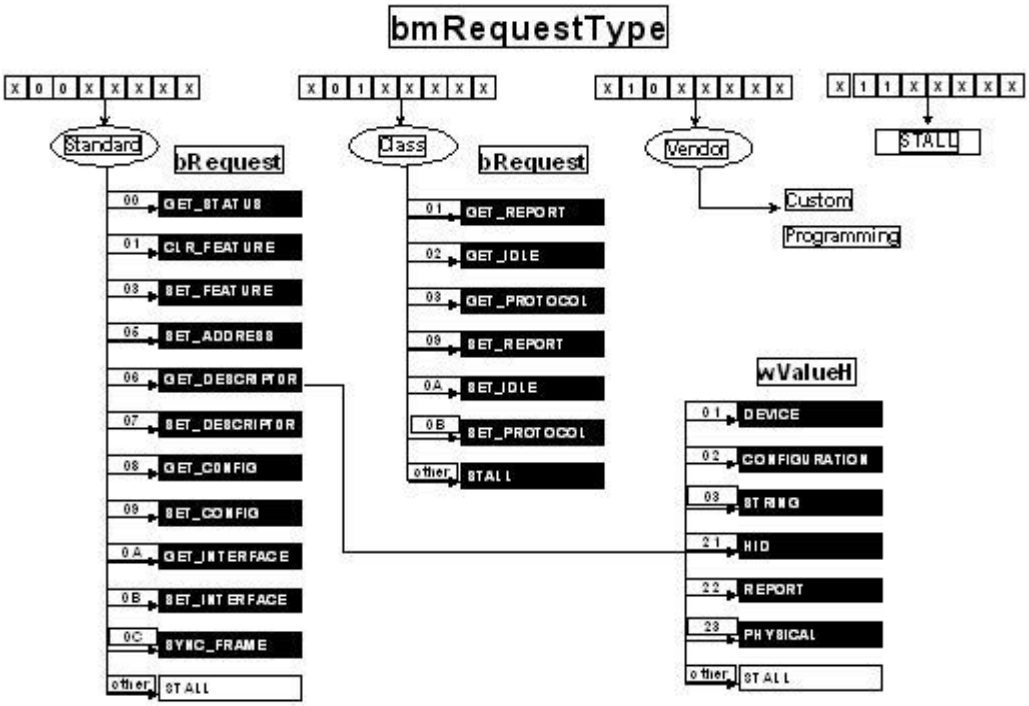


图 1.4　USB 设备请求

下面定义一个设备请求结构体：

```
typedef struct_device_request
{
    unsigned char bmRequestType;        //信息包括：传输方向、类型、接收方信息
    unsigned char bRequest ;            //例如：如果是标准请求，00 则是 Get_STATUS
    unsigned char wValue;               //请求值，根据具体请求而定
    unsigned char wIndex;               //请求索引，根据具体请求而定
    unsigned char wLength;              //请求长度，根据具体请求而定
}          //请求值、请求索引和请求长度是根据 bRequest 来确定的
```

传输方向有 IN 和 OUT；

类型有标准请求（Standard）、设备类请求（Class）、厂商请求（Vendor）；

接收方有设备(Device)、接口(Interface)、端点(Endpoint)、其它(Other)。

USB2.0 协议和 USB1.1 大体相同，下面介绍 USB2.0 相对于 USB1.1 协议增加的部分。

1.2 USB2.0 协议

随着 USB1.X 的普及应用，为了增加 USB 接口的应用范围，USB－IF 推出 USB2.0 规范，该规范在兼容 USB1.X 的基础至上，增加了 480Mbps 的高速数据传输。

总线拓扑上 USB2.0 仍然采用 USB1.X 的树型结构；物理连接上使用 USB1.X 定义好的 A 型和 B 型接口；在传输速度上，USB2.0 支持 1.5Mbps，12Mbps，480Mbps；数据传输上 USB2.0 同 USB1.1 规范，支持四种传输：控制传输，批量传输，中断传输和等时传输；在数据包上，USB2.0 和 USB1.1 有着相同的数据格式，为了支持高速数据，USB2.0 增加了新

的令牌数据包。

1.2.1 USB2.0 数据帧

USB2.0 和 USB1.1 规范最大的不同之处就是数据帧。如图 1.5，在 USB1.1 规范中，USB 数据采用每毫秒一个数据帧的方式进行数据传输，在毫秒数据帧的开始，USB 主机首先产生帧开始（SOF）数据包，并传输当前数据帧号，后面是传输数据。对于 USB2.0 规范，为了支持 480Mbps 高速传输速度，如图 1.6，USB2.0 提出了微帧的概念，每毫秒数据帧又包括 8 个微帧。

图 1.5　USB1.1 数据帧

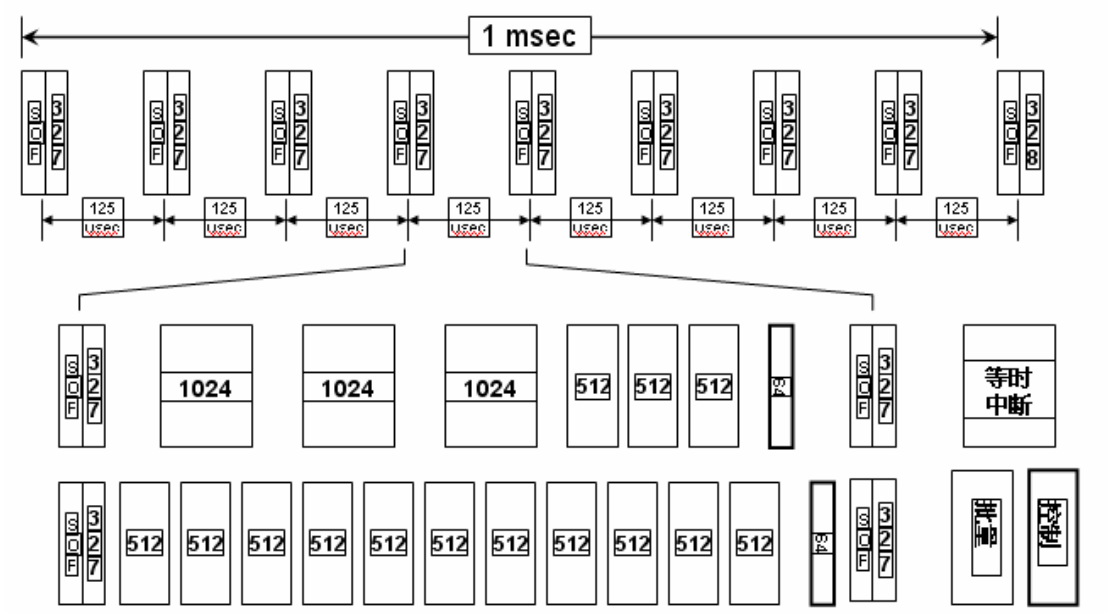


图 1.6　USB2.0 数据帧

从图 1.5 和图 1.6 中可看出，在 USB 每个数据帧中包括的控制、中断、等时和批量传输数据，每个传输类型分配一定的带宽，中断传输和等时传输有时间要求，所以每个数据帧中均要分配一定带宽。

1.2.2 USB2.0 端点缓冲区

相对于 USB1.X，USB2.0 中每种传输类型的端点可以用更大的缓冲区，见表 1.2。

表 1.2 端点缓冲区大小

传输类型	数据包大小	
	USB1.1	USB2.0
控制传输	8,16,32,64	64
批量传输	8,16,32,64	512
中断传输	1~64	1024
等时传输	1025	1024

1.3 USB 插头插座

USB 是良好的屏蔽线，总共由四根线组成，见表 1.3。

表 1.3 USB 接线信号

数字	信号名字	电缆
1	VBUS	红色线
2	D-	白色线
3	D+	绿色线
4	GND	黑色线
Shell	Shield	加蔽线

USB 接插头和座的类型有两种，A 型和 B 型。通过 A 和 B 从联结上来区分 USB HOST 和 USB DEVICE，A 型用在 USB HOST 端，B 型用在 USB DEVICE 端，见表 1.4。

表 1.4 USB 插头和座

A 型	插头	
	座	
B 型	插头	
	座	
MiniB 型	插头	
	座	

其中 miniUSB 型插头和座 USB2.0 协议才有。

1.4 为何选择 CY7C68013

各大公司都推出了 USB2.0 接口芯片，例如 PHILIPS 公司的 ISP1581（USB2.0 接口器件），NETCHIP 公司的 NET2270，ALI 公司的 M5621，CYPRESS 的 CY7C68013，关于更多 USB2.0 器件，可参考本网站“技术园地”栏目。

CYPRESS 公司推出的 CY7C68013 器件自从推出以来就受到广大用户的好评，主要原因有：

- ✧ 出色 USB2.0 单芯片解决方案，68013 包含增强型 8051 内核和智能 USB 接口
- ✧ 优良的性价比，56 引脚的 68013 市场售价不到 50 元
- ✧ 开发简单，CYPRESS 公司提供了完整开发方案，如调试界面和固件框架
- ✧ 真正体现 USB2.0 传输速度，包含通用可编程接口(GPIF)，实现与外设的“胶连接”，增强型 8051 的指令周期只有 4 时钟周期

第二章 CY7C68013芯片简介

2.1 FX 到 FX2 变化

CYPRESS 公司首先推出符合 USB1.1 标准的 EZ-USB（2100 系列）和 FX 系列芯片，随后推出了符合 USB2.0 标准的 FX2 系列芯片，68013 就是典型 FX2 系列芯片。从 FX 到 FX2 系列转变，可分析如何来提供 USB 传输速度的。首先分析影响 USB 传输速度的主要的四大瓶颈：

1. 低速的 MCU，USB1.1 的传输速率可达 12MB/s，而对传统的 8051 单片机而言，一个指令周期就需要 1us。
2. 复杂的 USB 协议，如果 USB 协议都需要通过 MCU 来完成，耗费大量时间。
3. 端点 FIFO 与外界数据交互速度。
4. 端点 FIFO 大小，影响一个 USB 数据帧内数据传输能力。

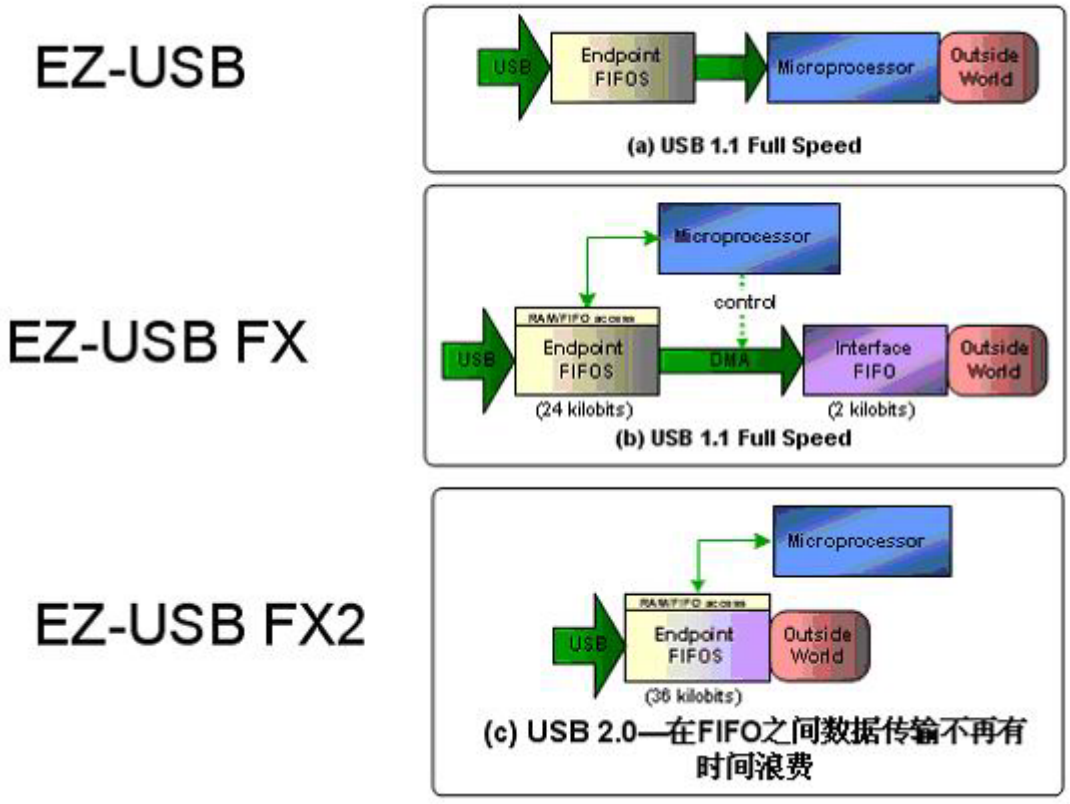


图 2.1 EZ-USB 数据传输结构

从图 2.1 可知，EZ-USB(2100 系列)数据传输都要通过 MCU，USB 传输速度要受限于 MCU 存储外界数据的速度和 MCU 填充端点 FIFO 的速度；FX 系列支持 DMA 传输方式，USB 传输速度仍然要受限于端点 FIFO 和外界 FIFO 数据交换速度；而 FX2 将 USB 端点 FIFO 与外设 FIFO 合二为一，实现与外设的“胶连接”，从而完全克服了 USB 传输四大瓶颈，可以总

结为：增强型高速 MCU，智能 USB 核自动处理 USB 协议，USB 与外设的“胶连接”，大缓冲区并最高可设置为四缓冲。

2.2 CY7C68013 介绍

2.2.1 CY7C68013 特点

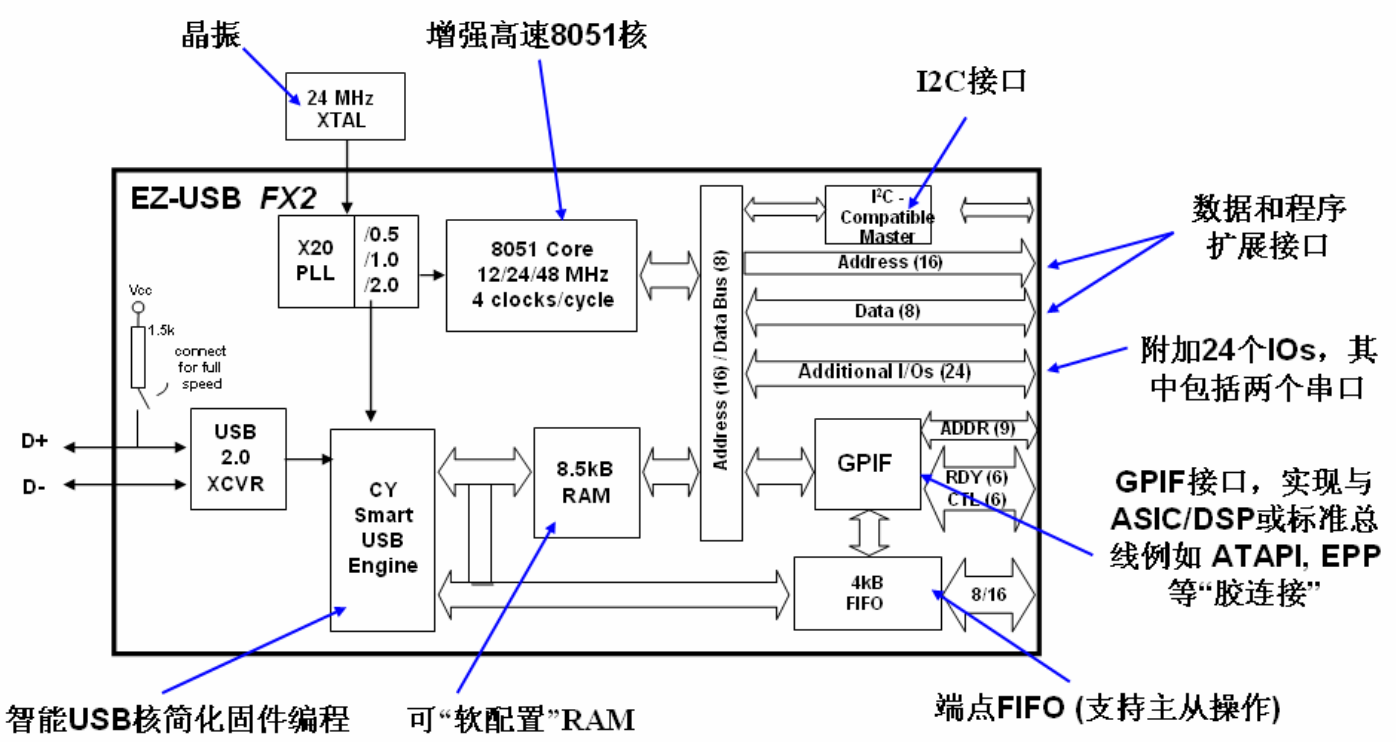


图 2.2　FX2 框架图

CY7C68013 特点：

✧　USB2.0 单芯片解决方案，包括 USB2.0 收发器，串行接口引擎(SIE)和增强型 51 内核

✧　可“软配置”RAM，大小为 8.5K，取代传统 51 的 RAM 和 ROM，程序可通过下面方式下载：

－ 通过 USB 口下载

－ 通过外部 E2PROM 装载

－ 外界存储设备（仅 128 引脚支持）

✧　通用可编程接口 GPIF，GPIF 是 FX2 一个重要技术

－ 可设置为主从模式，主模式下可对外部 FIFO，存储器，ATA 接口设备进行高速读写操作，从模式下外部主控器（如 DSP,MCU）可把 GPIF 端口当作 FIFO 进行高速读写操作。

－ 支持与外设通过并行 8 位或 16 总线传输

－ 支持通过 GPIF 编程工具编程，灵活产生各种波形

－ 支持多 CTL 输出和多 RDY 输入

✧　增强工业级 8051 内核，特点有

－ 支持 48M 时钟

- 4 个时钟指令周期，在时钟为 48M 时，单指令执行时间为 83.3nS
- 两个 UART
- 三个 TIMER
- 多中断系统
- 双数据指针

✧ 3.3V 工作电压
✧ 智能串行接口引擎（SIE）
✧ USB 中断矢量
✧ 100KHz 或 400KHz I2C 接口
✧ 4 个集成 FIFO

- 低成本与外设实现“胶连接”
- 自动实现从 16 位 FIFO 转换
- 支持主从工作模式
- FIFO 支持内外时钟和同步数据触发
- 轻松实现与 ASIC，DSP 连接

✧ 包括 40 多个通用 IO 端口
✧ 4 种可选封装－56 引脚 SSOP 和 QFN，100 引脚 TQFP 和 128 引脚 TQFP。

2.2.2 引脚说明

FX2 的引脚主要可以分为几类：电源引脚，包括数字电源地和模拟电源地；系统引脚，如时钟，USB 挂起外部唤醒，USB 差分数据线，复位引脚，中断，计数器输入，UART 通信；通用 IOs，包括端口 A，B，C，D，E；地址与数据总线，可用来外扩展 RAM；GPIF 主模式引脚；FIFO 从模式 SLAVE FIFO 引脚。

引脚功能见图 2.3，图中 128 引脚封装 68013 包括所有功能引脚，56 引脚封装和 100 引脚封装只有其中的部分引脚。

- XTALIN 和 XTALOUT 时钟输入引脚
- RESET#复位输入引脚，低电平有效
- WAKEUP#外部唤醒引脚输入，低电平有效
- SCL 和 SDA 为 I2C 接口时钟与数据引脚
- IFCLK，GPIF 时钟，可作为输入或输出
- CLKOUT，时钟输出，可设置为 12M，24M 或 48M 时钟输出，作为其他外设时钟
- DPLUS 和 DMINUS，USB 数据线 D+和 D-
- PB0~PB7 端口 B，复用引脚，可设置为 GPIF 数据总线的低八位 FD[0]~FD[7]
- PD0~PD7 端口 D，复用引脚，可设置为 GPIF 数据总线的高八位 FD[8]~FD[15]
- RDY0~RDY5，GPIF 主模式作为状态输入引脚，RDY0 和 RDY1 为复用引脚，FIFO 从模式作为读写信号 SLRD 和 SLWR
- CTL0~CTL5，GPIF 主模式作为控制输出引脚，CTL0～CTL2 可复用为 FIFO 从模式作为状态标志引脚 FLAGA，FLAGB，FLAGC
- PA0~PA7 端口 A，复用引脚，PA0 与中断 INT0 复用，PA1 与中断 INT1 复用，PA3 与 USB 唤醒可选引脚 WU2 复用， PA2 与 FIFO 从模式 FIFO 输出使能引脚复用，PA4 与 FIFO 从模式 FIFO 地址 FIFOADD0 引脚复用，PA5 与 FIFO 从模式 FIFO 地址 FIFOADD1 引脚复用，PA6 与 FIFO 从模式包结束 PKTEND 引脚复用，PA7 与 FIFO 从模式状态标志位 FLAGD，以及和 FIFO 从模式 SLAVE FIFO 使能/触发 SLCS#引脚复用，PA7 引脚

功能由寄存器 IFCONFIG[1:0]来配置

- PC0~PC7 端口 C，复用引脚，与 GPIF 主模式地址 GPIFADR0~7 引脚复用
- PE0~PE7 端口 E，复用引脚，PE7 与 GPIF 主模式地址 GPIFADR8 引脚复用，端口 E 其他引脚复用可见图 2.3

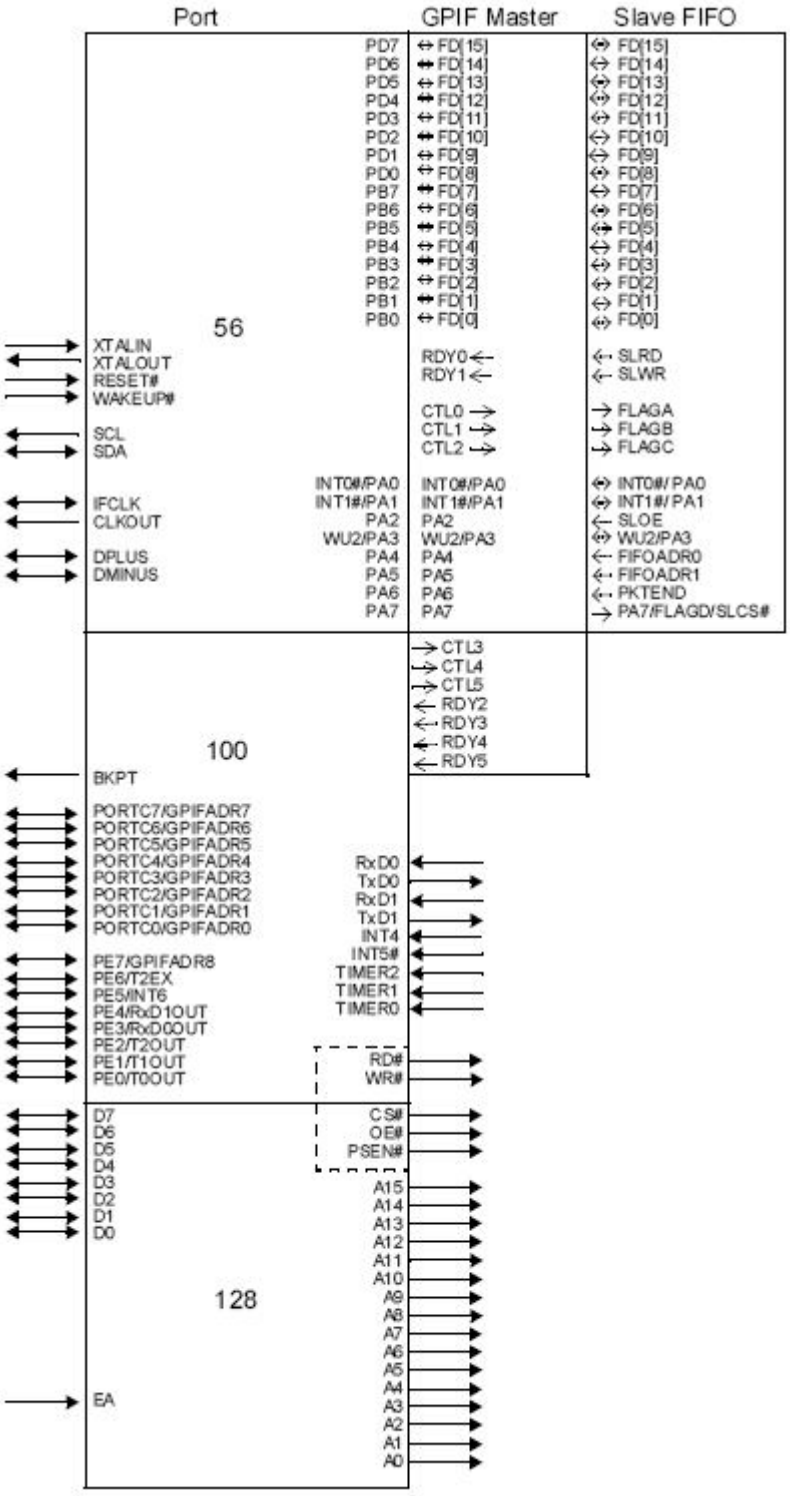


图 2.3 68013 功能引脚

- A0~A15 外扩 RAM 地址总线

- D0~D15 外扩 RAM 数据总线
- RD# ,WR#,CS#,OE#,PSEN#外扩 RAM 控制逻辑
- EA，外部地址使能
- RxD0,TxD0, RxD1,TxD1 串口 1 和 2
- INT4,INT5#，外部中断
- TIMER0，TIMER1,TIMER2 计数器时钟输入引脚

说明：在实际应用中，用户需要根据项目需要来选择合适的 68013 芯片封装，128 引脚功能较全，可以外扩 RAM，两个串口可用来实现与其它 MCU 通信和程序在线仿真功能，另外各种功能引脚也较多，但是设计较为复杂。56 引脚优点是设计简单，成本较低，缺点则是不能外扩 RAM 和在线仿真，功能引脚较少。

2.3 存储器

对于 EZ-USB 存储器都包括 RAM，没有 ROM，所以程序和数据都只能是存储在 RAM 中，RAM 包括内部 RAM 和外部 RAM，内部 RAM 和 8051 的内部 RAM 功能一样，外部 RAM 则是 EZ-USB 将传统 8051 部分外扩 RAM 放到了芯片的内部，用来存放数据和存储，程序在调电后遗失，需要

2.3.1 内部 RAM

内部 RAM 见图 2.4，和传统 8051 内部 RAM 功能相同。

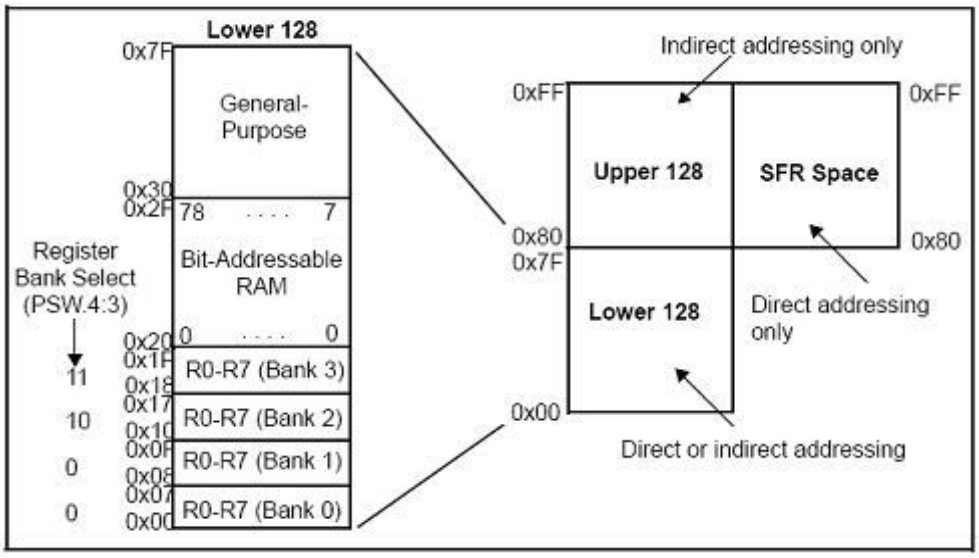


图 2.4 内部 RAM

2.3.2 外部 RAM

图 2.5 是 EA=0 时 FX2 中 RAM 的分布，EA=1 时情况很少使用，所以本文中不做介绍。外部 RAM 包括两部分，片上外部 RAM（Inside FX2）和外扩 RAM（Outside FX2），外扩 RAM 是通过 128PIN 封装中的引脚 D0~D7，A0~A15 和 RD# ,WR#,CS#,OE#,PSEN#来实现的，对于 56PIN 封装，只有片上外部 RAM。

地址 0x0000～0x1FFF 总共 8KB 可以用来存放程序或数据，地址 0xE000～0xE200 总共 0.5KB 只能用来存放数据，地址 0xE200～0xFFFF 是寄存器和端点 FIFO 空间。0xE000～0xFFFF 详细分布见图 2.6。

总共有 8.5K 的程序和地址空间，如果端点缓冲区没有用到，也可以用来存放数据，程序可设置范围为 0x0000～0x1FFF，数据可设置范围为 0x0000～0x1FFF，0xE000～0xE200 和 0xF000~0xFFFF(仅在端点缓冲区没有使用的情况下可以设置)。如果程序使用了中断，则地址空间 0x0000～0x0080 空间作为中断程序入口地址，不能使用。地址空间设置见图 2.7。

对于 128 引脚封装 FX2，如果扩展了外部 RAM，则外部 RAM 中数据可设置范围是 0x2000～0xDFFF，程序可设置范围是 0x2000～0xFFFF。

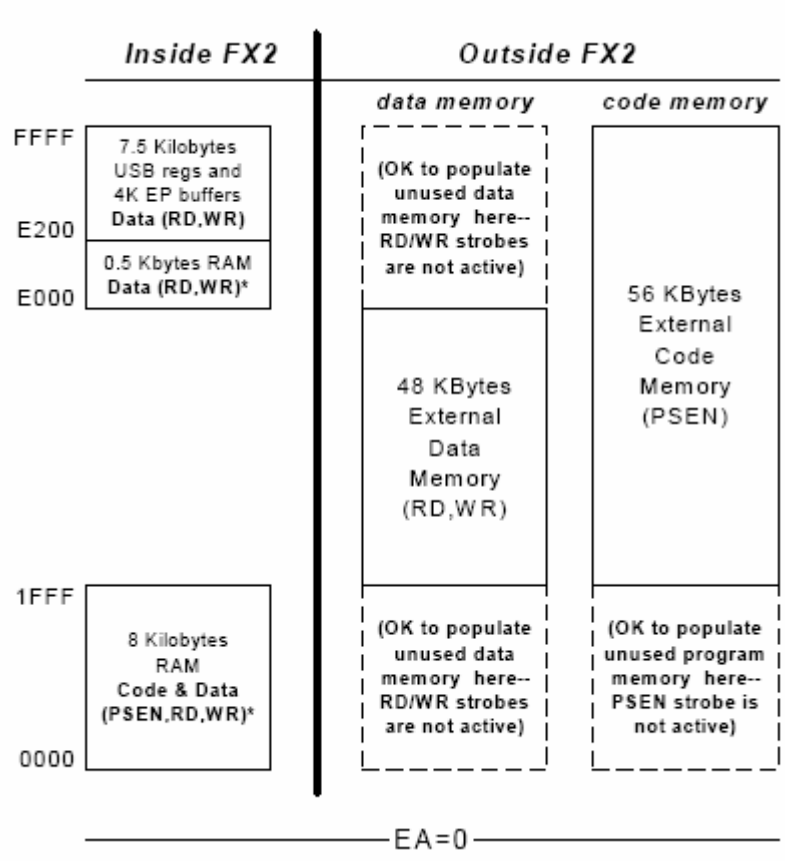


图 2.5 内嵌外部 RAM

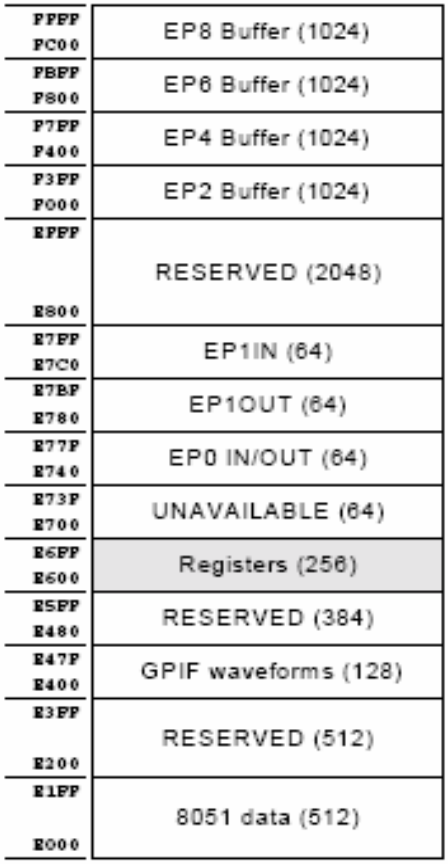


图 2.6 寄存器和断点 FIFO 地址

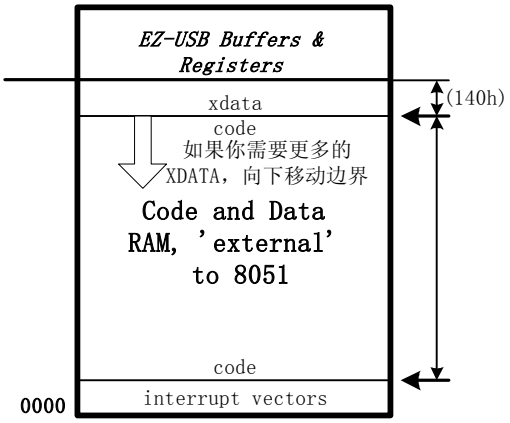


图 2.7 XDATA 和 CODE 空间分配

各地址空间访问方法见图 2.8，包括寄存器、内部 RAM 和外部 RAM。

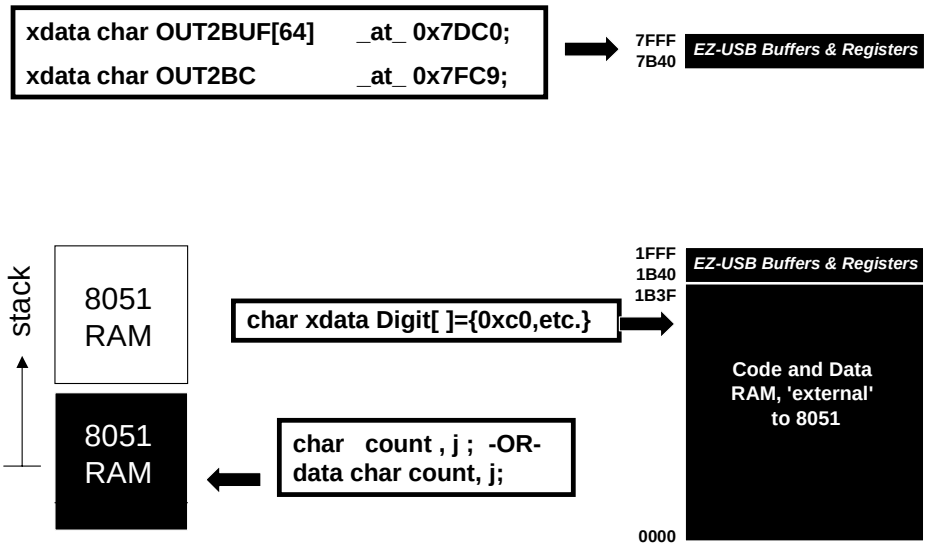


图 2.8 RAM C51 编程访问方法

2.4 中断

FX2 Interrupt	Source	Interrupt Vector	Natural Priority
IE0	INT0 Pin	0x0003	1
TF0	Timer 0 Overflow	0x000B	2
IE1	INT1 Pin	0x0013	3
TF1	Timer 1 Overflow	0x001B	4
RI_0 & TI_0	USART0 Rx & Tx	0x0023	5
TF2	**Timer 2 Overflow**	0x002B	6
Resume	**WAKEUP / WU2 Pin or USB Resume**	0x0033	0
RI_1 & TI_1	**USART1 Rx & Tx**	0x003B	7
USBINT	**USB**	0x0043	8
I²CINT	**I²C-Compatible Bus**	0x004B	9
IE4	**GPIF / FIFOs / INT4 Pin**	0x0053	10
IE5	**INT5 Pin**	0x005B	11
IE6	**INT6 Pin**	0x0063	12

图 2.9 中断向量表

Priority	INT2VEC Value	Source	Notes
1	00	SUDAV	SETUP Data Available
2	04	SOF	Start of Frame (or microframe)
3	08	SUTOK	Setup Token Received
4	0C	SUSPEND	USB Suspend request
5	10	USB RESET	Bus reset
6	14	HISPEED	Entered high speed operation
7	18	EP0ACK	FX2 ACK'd the CONTROL Handshake
8	1C	reserved	
9	20	EP0-IN	EP0-IN ready to be loaded with data
10	24	EP0-OUT	EP0-OUT has USB data
11	28	EP1-IN	EP1-IN ready to be loaded with data
12	2C	EP1-OUT	EP1-OUT has USB data
13	30	EP2	IN: buffer available. OUT: buffer has data
14	34	EP4	IN: buffer available. OUT: buffer has data
15	38	EP6	IN: buffer available. OUT: buffer has data
16	3C	EP8	IN: buffer available. OUT: buffer has data
17	40	IBN	IN-Bulk-NAK (any IN endpoint)
18	44	reserved	
19	48	EP0PING	EP0 OUT was Pinged and it NAK'd
20	4C	EP1PING	EP1 OUT was Pinged and it NAK'd
21	50	EP2PING	EP2 OUT was Pinged and it NAK'd
22	54	EP4PING	EP4 OUT was Pinged and it NAK'd
23	58	EP6PING	EP6 OUT was Pinged and it NAK'd
24	5C	EP8PING	EP8 OUT was Pinged and it NAK'd
25	60	ERRLIMIT	Bus errors exceeded the programmed limit
26	64	reserved	
27	68	reserved	
28	6C	reserved	
29	70	EP2ISOERR	ISO EP2 OUT PID sequence error
30	74	EP4ISOERR	ISO EP4 OUT PID sequence error
31	78	EP6ISOERR	ISO EP6 OUT PID sequence error
32	7C	EP8ISOERR	ISO EP8 OUT PID sequence error

图 2.10 USB 中断矢量

FX2 中断向量见图 2.9，在传统 8051 的 5 个中断中断源基础上，FX2 又增加了 8 个中断，3 个外部中断 INT4,INT5#,INT6， USB 中 USBINT，I2C 中断，串口 1 中断，时钟 2 溢出中断，和 USB 远程唤醒中断。INT4 中断是 GPIF 中断和 FIFOs 中断复用。

USB 中断矢量表见图 2.10，32 个 USB 中断通过或门来产生 USBINT 中断。

USB 中断包括：

- SUDAV，SETUP 阶段完成中断。
  控制传输由三个阶段组成，SETUP 阶段，DATA 阶段和 STATUS 阶段，在 SETUP 阶段就有两个中断，见图 2.11。

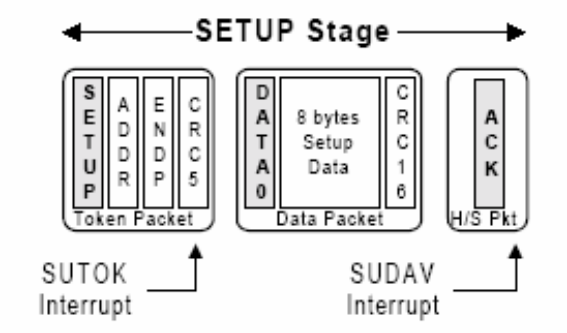


图 2.11 SETUP 阶段中断

- SUTOK，SETUP TOKEN 包中断
- SOF，USB 数据帧起始中断，对于 USB2.0，125 微秒一个 SOF 中断，将更新 USBFRAMEH 和 USBFRAMEL 帧计数寄存器
- SUSPEND，USB 总线挂起请求中断
- USB RESET，USB 总线复位中断
- HISPEED，USB2.0 高速模式枚举成功中断
- EP0ACK，端点 0 控制传输状态阶段结束时产生的中断
- EP0－IN，端点 0 准备好装载数据，并产生中断
- EP0－OUT，端点 0 已经接收到数据，并产生中断
- EP1－IN，端点 1 准备好装载数据，并产生中断
- EP1－OUT，端点 1 已经接收到数据，并产生中断
- EP2，端点 2 中断，作为 IN，表示端点准备好装载数据，作为 OUT，表示接收到数据
- EP4，端点 4 中断，作为 IN，表示端点准备好装载数据，作为 OUT，表示接收到数据
- EP6，端点 6 中断，作为 IN，表示端点准备好装载数据，作为 OUT，表示接收到数据
- EP8，端点 8 中断，作为 IN，表示端点准备好装载数据，作为 OUT，表示接收到数据
- IBN，如果 USB 主控端发出 BULK IN 命令，而 IN 端点却没有向主控端发送数据，则 FX2 自动用 NAKs 来响应，并产生 IN-BULK-NACK 中断
- EP0PING，端点 0 PING 中断，仅在高速模式中有效，为了提高 BULK OUT 方式总线利用率，USB2.0 规范中增加了 PING-NAK 机制，在每次 BULK OUT 传输前，主控器首先发出 PING 命令，如果对应 BULK OUT 缓冲区没有准备好，则用 NAK 来响应，并产生一个中断。
- EP1PING，端点 1 PING 中断
- EP2PING，端点 2 PING 中断
- EP4PING，端点 4 PING 中断
- EP6PING，端点 6 PING 中断

- EP8PING，端点 8 PING 中断
- ERRLIMIT，USB 总线错误超过所设置的错误极限，并产生中断
- EP2ISOERR，端点 2 等时传输中断
- EP4ISOERR，端点 4 等时传输中断
- EP6ISOERR，端点 6 等时传输中断
- EP8ISOERR，端点 8 等时传输中断

中断 4 有多个中断矢量组成，如图 2.12，图 2.13 是 FX2 对多中断矢量的调用方法。

Priority	INT4VEC Value	Source	Notes
1	80	EP2PF	Endpoint 2 Programmable Flag
2	84	EP4PF	Endpoint 4 Programmable Flag
3	88	EP6PF	Endpoint 6 Programmable Flag
4	8C	EP8PF	Endpoint 8 Programmable Flag
5	90	EP2EF	Endpoint 2 Empty Flag
6	94	EP4EF	Endpoint 4 Empty Flag
7	98	EP6EF	Endpoint 6 Empty Flag
8	9C	EP8EF	Endpoint 8 Empty Flag
9	A0	EP2FF	Endpoint 2 Full Flag
10	A4	EP4FF	Endpoint 4 Full Flag
11	A8	EP6FF	Endpoint 6 Full Flag
12	AC	EP8FF	Endpoint 8 Full Flag
13	B0	GPIFDONE	GPIF Operation Complete (See *Chapter 10, "General Programmable Interface (GPIF)"*)
14	B4	GPIFWF	GPIF Waveform (See *Chapter 10, "General Programmable Interface (GPIF)"*)

图 2.12　INT4 中断矢量

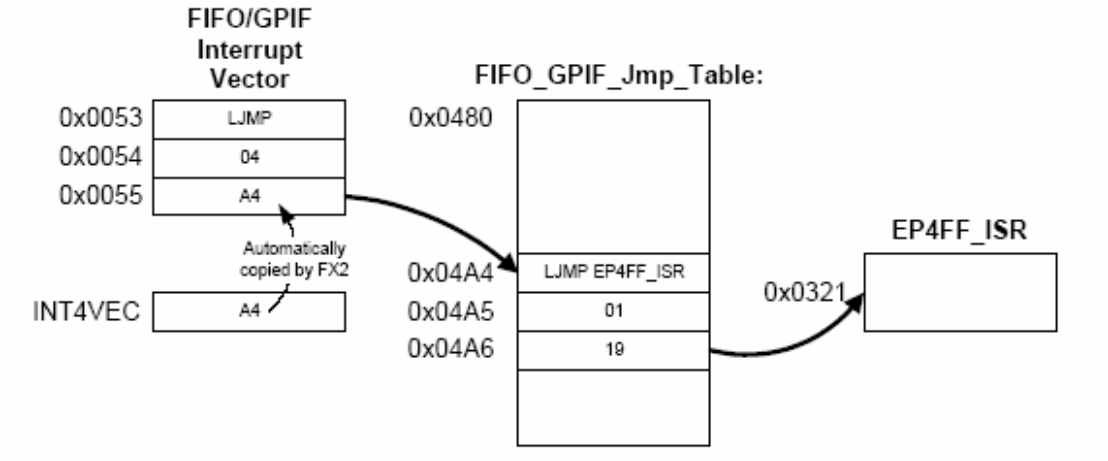


图 2.13　INT4 中断跳转

2.5 重枚举

FX2 的启动过程，上电复位后

1. 如果 E2PROM 不存在，则用默认 PID/VID 启动 USB 设备，缺省的 USB 设备自动处理所有 USB 请求，此时的固件需要通过控制面板来下载，**调试时常用**。
2. 如果 E2PROM 存在，并且 E2PROM 中第一字节是 0xC0，后面字节包括用户定制的 PID/VID，则用定制好的 PID/VID 来从主机下载固件和加载驱动，此方式称为“C0 加载”模式，**产品发布时常用**。
3. 如果 E2PROM 存在，并且 E2PROM 中第一字节是 0xC2，后续内容包括 PID/VID，以及程序代码，则 FX2 从 E2PROM 加载固件，此方式称为“C2 加载”模式。
4. 如果 FLASH 或 EPROM 通过地址总线和数据总线扩展（仅 128 引脚封装有效），并且 EA=1，则程序从外部空间的 0x0000 开始运行。

下面重点介绍 FX2 有三种启动模式：无 E2PROM，“C0 加载”E2PROM，“C2 加载”E2PROM。

1. 无 E2PROM 或 E2PROM 没有有效启动数据

FX2 用默认的 PID/VID 来响应请求，见表 2.1。

表 2.1 FX2 默认 ID

Vendor ID 厂商 ID	0x04B4 (Cypress Semiconductor/)
Product ID 产品 ID	0x8613 (EZ-USB FX2)
Device Release 发布版本号	0xXXYY (depends on revision)

2. E2PROM 存在，并且第一字节是 0xC0

表 2.2 “C0 加载”数据格式

E2PROM地址	数据
0	0xC0
1	VID 低字节
2	VID 高字节
3	PID 低字节
4	PID 高字节
5	DID 低字节
6	DID 高字节
7	配置字节

E2PROM 头 8 个字节见表 2.2，例如 C0 B4 04 82 00 01 00 00，则表示 VID 是 0x04B4，PID 是 0x0082，DID 是 0100，配置字节是 0x00，配置字节内容将改写 FX2 的配置字节寄存器，配置字节的第六位必须设置为零。

3. E2PROM存在，并且第一字节是0xC2

表 2.3 “C2 加载” 数据格式

E2PROM地址	数据
0	0xC2
1	VID 低字节
2	VID 高字节
3	PID 低字节
4	PID 高字节
5	DID 低字节
6	DID 高字节
7	配置字节
8	长度 高字节
9	长度 低字节
10	起始地址 高字节
11	起始地址 低字节
…	数据块
…	长度 高字节
…	长度 低字节
…	起始地址 高字节
…	起始地址 低字节
…	数据块
…	0x80
…	0x01
…	0xE6
…	0x00
最后	00000000

表 2.4 中头八个字节和“C0 加载”数据格式除第一字节外其它字节相同，从第九个开始为的程序块，每个程序块包括起始地址、数据长度和数据块组成，最后一个字节 0x00 将写入到 CPUCS 寄存器（地址是 0xE600）中。

CYPRESS 提供了 hex2bix 来生成“0xC2 加载”格式的数据。

2.6 端点

FX2 端点包括：

- EP0，端点 0，双向数据传输，只能配置为控制传输，缓冲区大小 64 字节
- EP1-IN，EP1-OUT，端点 1，64B 缓冲区，可以设置为中断、批量传输类型
- EP2,EP4,EP6,EP8，端点 2、4、6、8，共用 8 个 512 字节缓冲区，可以设置为中断，批量和等时传输类型，端点 2 和 6 可以设置为双缓冲，三缓冲和四缓冲，缓冲区大小可设置为 512 字节或 1024 字节，设置见图 2.14。

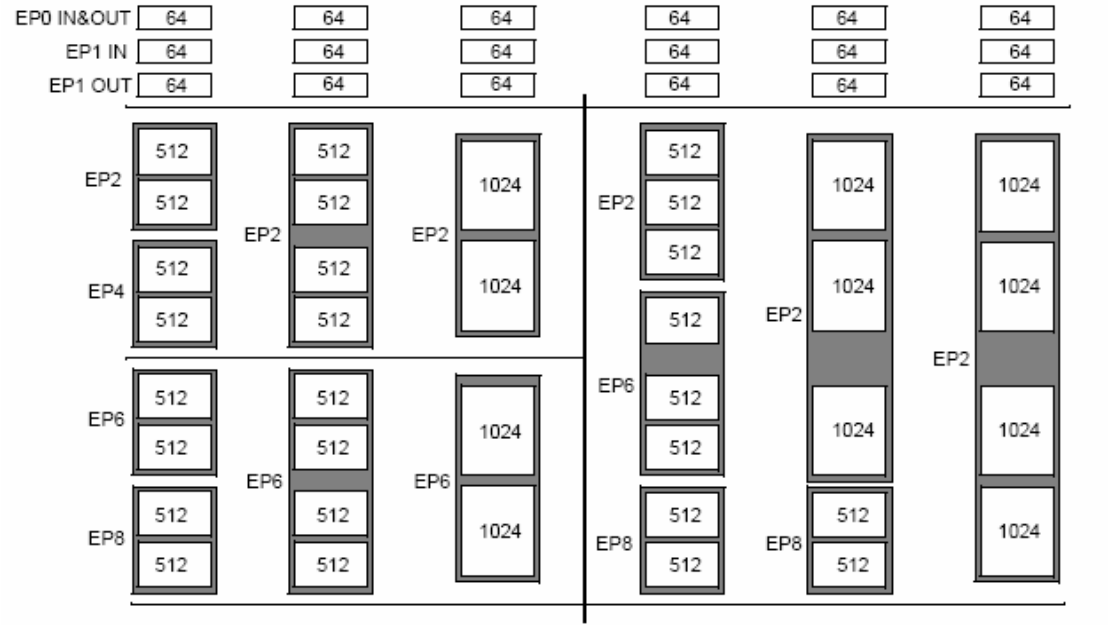


图 2.14　FX2 端点缓冲区设置

2.7 GPIF 简介

为了提供FX2与外设提供高速传输，FX2提供了GPIF接口，见图2.15。把外设分为两种情况，一是不带MCU的，例如FIFO、AD、IDE等，二是带MCU的，例如DSP、ARM等。

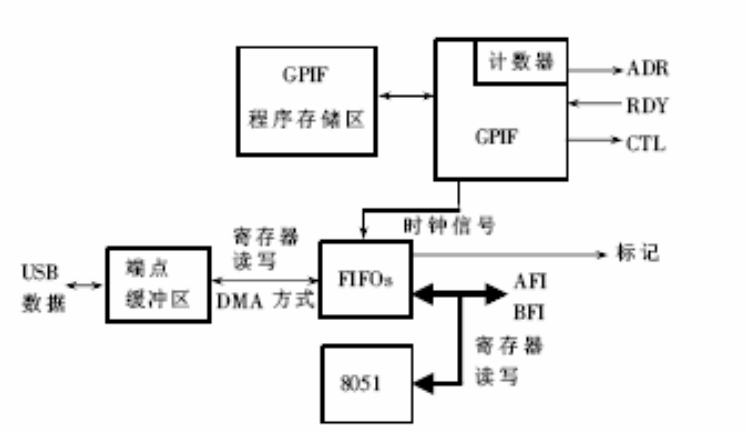


图 2.15　GPIF 与 8051 内核关系

GPIF，即通用可编程接口，是CYPRESS在其EZ-USB、FX和FX2系列单片机里设计的一个可由用户编程的接口， 具有快速、灵活等特点，可使用多种协议完成与外围器件的无缝连接，如EIDE/ATAPI、IEEE1428、Utopia等。对其可以根据需要进行编程，且运行中不需要CPU的干预，仅通过一些寄存器和中断与增强型8051内核通讯。

GPIF主要组成部分：

（1）ADR5:0：地址线，可作为扩展存储器的低位地址，在连续执行GPIF动作时具有自动增一功能。可对其对应的寄存器进行读写（FX2系列中扩充为9根地址线）。

（2）RDY5:0：输入Ready信号，可对指定的信号进行连续采样，以确定GPIF动作继续、等待或是反复不断采样，直到信号的指定状态出现。通常用来等待指定信号的某个状态出现，然后继续剩余动作。

（3）CTL5:0：输出控制信号，根据编程指令输出高低电平或集电极开路。通常用作选通信号、非总线输出信号，以及产生简单的脉冲信号。

（4）FD15:0： 双向 FIFO 数据线， 一般又称 AFI、BFI。

（5）IFCLK：时钟接口，决定使用外接时钟还是使用内部 48MHz 或 30MHz 时钟周期。

（6）GSTATE5:0：当前 GPIF 状态数，可用来判断当前 GPIF 工作状态，一般调试用。

（7）GPIF PROGRAM：GPIF 程序存储区间 0x7900~0x797F 指令，可存储 4 组波形的程序代码（FX2 系列存储区间为 0xE400~0xE47F）。

每个 GPIF 动作都由七段组成：Interval0~Interval6，简称 I0~I6。执行完 I0~I6 的动作后， 最后都进入 I7，即空闲状态，以准备启动下一次 GPIF 动作。每个 Interval 可以定义为 Non-Decision Interval，简称 NDP；或是 Decision point interval，简称 DP。

当某个 Interval 定义为 NDP，在执行这个 Interval 动作时，只是简单地延时，用来确定产生指定电平的延续时间；而当 Interval 定义为 DP 时，它将根据 RDY5:0 引脚上的输入信号状态， 以及内部 FIFO 的可编程标志和一个内部自定义的 Interval RDY 标志， 将这些信号进行逻辑与、逻辑或，或是进行逻辑异或，根据得到的逻辑结果在 I0~I6 中选择下一个即将执行的 Interval。在每个 Interval 执行时，都可指定 CTL5:0 输出用户指定的状态。通过 RDY 和 CTL 以及内部一些标志位的组合，能完成各种复杂的时序电路的控制。

见图2.16和图2.17通过GPIF实现USB主控器和外设之间的高速IN和OUT高速传输。

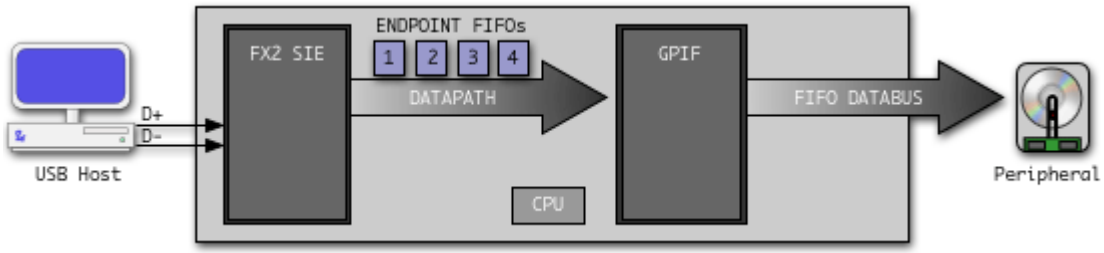


图2.16 利用GPIF实现自动OUT数据流传输

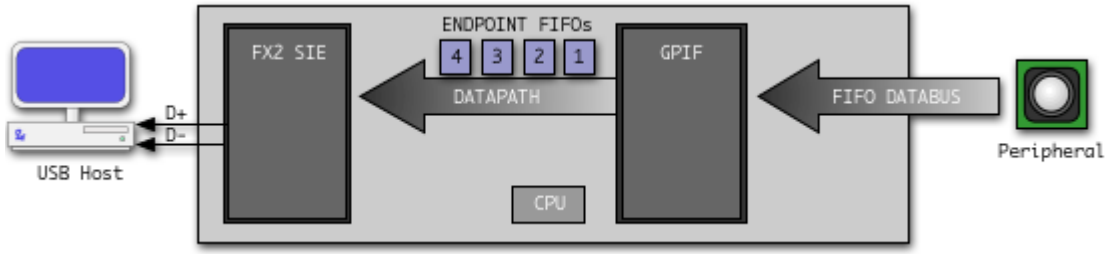


图2.17 利用GPIF实现自动IN数据流传输

为了方便GPIF编程，CYPRESS提供了GPIF波形编辑工具“Cypress GPIF Designer”，通过工具可以很方便编辑访问外设，工作界面如图2.18，本网站将推出利用GPIF访问高速AD的开发板，详细信息见本网站。

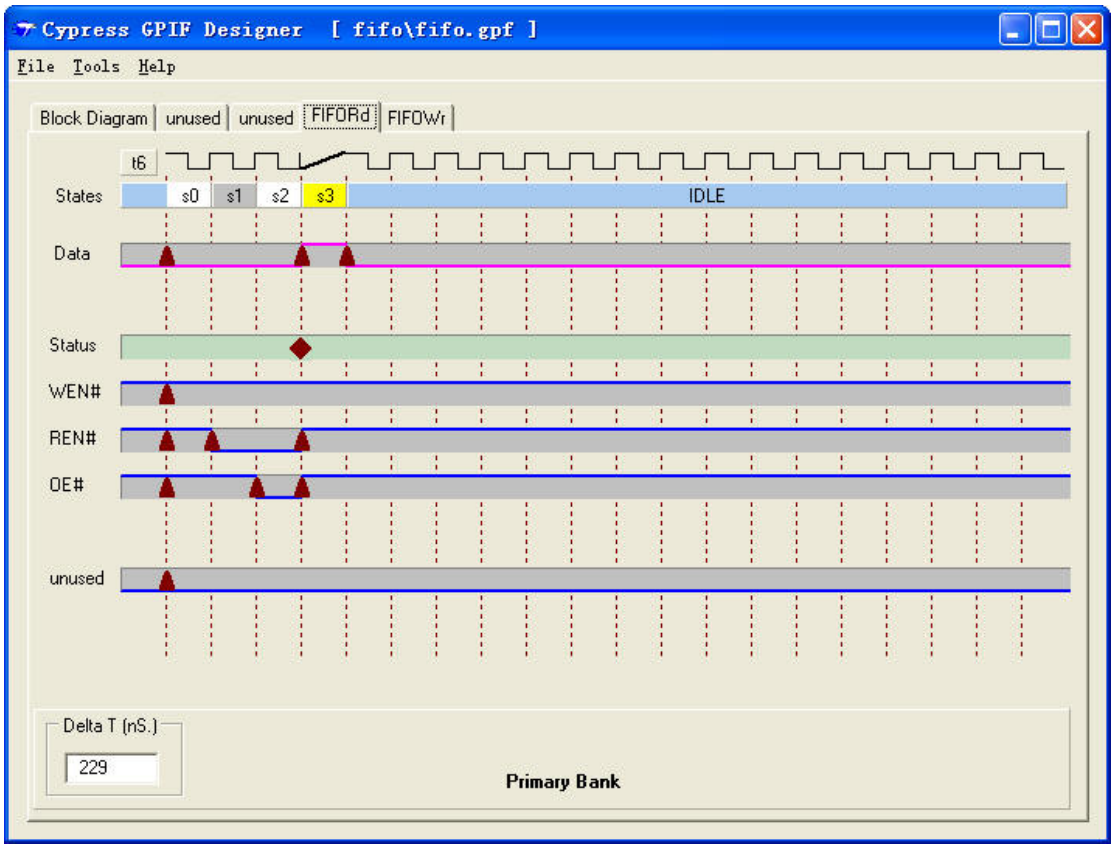


图2.18 “Cypress GPIF Designer”界面

2.8 程序下载和仿真

FX2 提供了在线下载和在线仿真功能，无需编程器和仿真器，大大降低开发成本。

1. FX2 在线下载

FX2 提供了“智能 USB 核”，所以硬件连接图和外部 RAM 配置正确就能识别为 USB 设备，通过 FX2 提供的控制面板，就能下载固件程序，该方法在调试阶段经常使用；产品发布阶段，用户通过 E2PROM 定制的 PID/VID 来加载驱动程序，第一个驱动程序的主要工作就是下载固件。控制面板使用和产品发布将在后面详细介绍。

2. FX2 在线仿真

FX2 和 EZUSB 系统都提供了在线仿真功能，用户通过 USB 先下载一个在线仿真监视程序，然后 KEIL 通过 FX2 串口下载仿真程序实现仿真，用户可以设置断点和观察变量值。本网站推出的 FCUSB-CY7C6813-128 将提供在线仿真功能。FCUSB-AN2131 也提供了仿真功能。

如果用户对 EZUSB 或 FX2 熟悉，56 引脚没有串口通信引脚，所以不能在线仿真，但也能够实现简单调试，主要是通过厂商请求来得到寄存器和变量数值，这种方法虽然简单，往往也很有效，将在后面介绍。

第三章 控制面板

3.1 控制面板使用

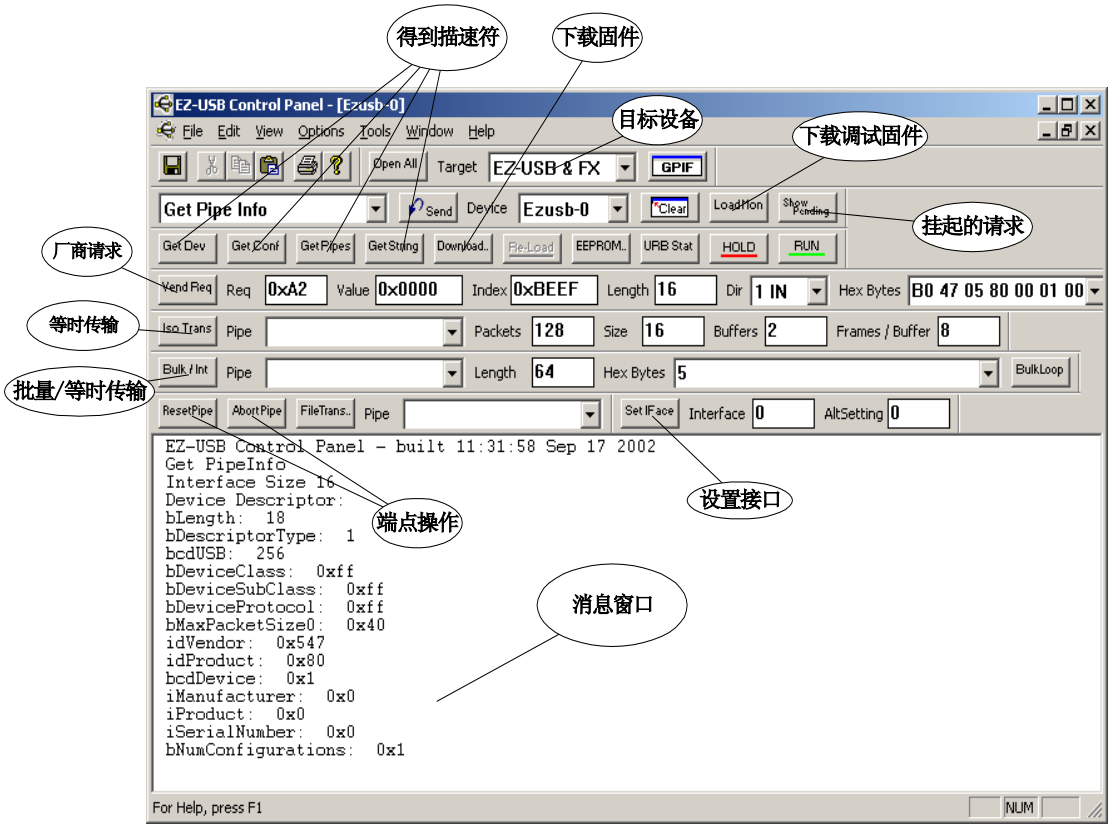


图 3.1 控制面板

控制面板界面见图 3. 1。

控制面板组成：

（1）USB 标准请求，得到各种描速符，包括 GetDev(设备)、GetConf（配置）、GetPipes（管道）、 GetString（字符串），Set Iface 设置接口。插入本开发板，用户可点击控制面板各操作，然后从消息窗口观察操作结果信息，点击 GetDev 在消息串口中得到 FX2 设备信息，设置接口操作见图 3.2，用户改变可选设置的值，然后点击 Get Pipes 观察各种配置下端点的设置情况，并和“FX2 TechRefManual”文档中默认接口配置比较。

图 3.2 Set IFace 操作

说明：开发板插入 USB2.0 主控器后，提示“高速 USB 设备插入非高速集线器”，默认响应为 USB1.1 设备，所以 Get Pipes 得到也是 USB1.1 的端点描速符，需要在固件设置正确后才显示为 USB2.0 设备，详见调试实例一章。

（2）Download 固件下载。因为 EZUSB 没有 ROM，所以每次上电都必须要下载固件，控制面板 Download 是在线下载方式的一种，按钮 Re-load 是再次下载 Download 指定的固件程序。

（3）LoadMon 在线仿真监视程序下载。仿真监视程序位于 C:\Cypress\USB\Target\Monitor 目录下，共有五个监视程序，见表 3.1。

表 3.1 监视程序

监视程序	FX2 串口	地址范围	适用器件
mon-ext-sio1-e0.hex	1	0xe000-0xef75	EZUSB,FX
mon-ext-sio1-c0.hex	1	0xc000-0xcf75	FX2
mon-int-sio1.hex	1	0x0000-0x1075	EZUSB,FX,FX2
mon-ext-sio0-c0.hex	0	0xc000-0xcf75	EZUSB,FX,FX2
mon-int-sio0.hex	0	0x0000-0x1075	EZUSB,FX,FX2

安装时，默认使用的是串口 0 实现在线仿真，如果要使用串口 1，请参考 Monitor 目录下 readme 文档。监视程序选择先点击菜单中的 Option->Properties，然后在弹出的窗体中 Property Sheet 中的 Paths 属性页下选择监视程序，见图 3.3。

图 3.3 监视程序选择

（4）VendReq 厂商请求，见图 3.4，CYPRESS 定义了两个有用的厂商请求，读写 EEPROM 和 RAM，请求格式包括请求 Req（0xA2 是读写 EEPROM，0xA3 是读写 SRAM），请求值 Value（表示读写的初始地址），索引 Index(这里没有用到)，Length（数据长度），Dir（IN 或 OUT），Hex Bytes（只有 DIR 是 OUT 时有意义，表示要写入的数据），在执行请求先要下载固件 Vend_ax。

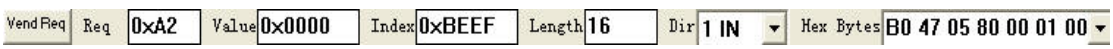


图 3.4 厂商请求

（5）等时传输，包括 Pipe（端点），Packets（帧数目），Size（每帧的大小），Buffers（缓冲

图 3.5 等时传输

区数目），Frames/Buffer（每个缓冲区包括的帧数）。其中 Packets 必须能够整除 (Buffers*Frames/Buffer)。见图 3.5。

(6) 批量/中断传输，见图 3.6，包括 Pipe（端点），Length（传输长度），Hex Bytes（如果是

图 3.6 批量/中断传输

OUT 端点，表示要写入 EZUSB 中的数据），Bulk Loop（循环 Bulk 传输）。

(7) 端点操作，见图 3.7，包括 ResetPipe（当端点传输发生错误并暂时停止需要复位端点，然后可进行新的数据传输），AbortPipe（当端点数据发生严重错误，则放弃该端点）。File

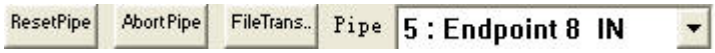


图 3.7 端点操作

Trans •••• 文件传输，允许用户通过 Bulk，ISO 来传送文件。

(8) 属性设置窗体，见图 3.8。

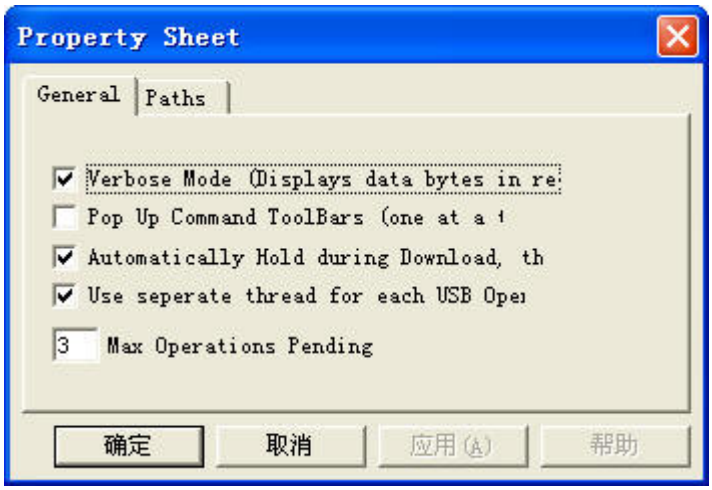


图 3.8 属性设置窗体

属性窗体都使用默认设置，用户一般不需要修改。默认设置为每个 USB 操作都打开一个线程，最大挂起数目为 3。

（9）其它按钮

Open All：见图 3.9，在打开 USB 设备之前，必须先选择目的设备类型，例如 EZUSB，FX2 和 FX，如设备选择不正确，将导致其它操作错误。

图 3.9 打开 USB 设备和 GPIF 编程帮助

GPIF：GPIF 编程，一般利用 GPIF 编程工具编程，所以一般不用它。

Send：先在图 3.10 中左边第一个下拉列表选择操作，然后点击 Send 执行，SEND 很多操作在工具栏中都有。

Device：如果多个插入，通过设备名来选择设备。

Clear：清空消息窗口。

LoadMon：下载调试监视程序。

图 3.10 发送操作等

Show Pending：显示挂起操作。

3.2 控制面板源程序

控制面板源程序位于安装目录 C:\Cypress\USB\Util\EzMr。

开发工具为 VC，采用 MFC 基于多文本框开发，源代码较复杂，其中大部分是针对界面显示和程序数据交互的，本节主要介绍程序主体部分，编程中使用了多线程，一般 USB 操作都是多线程来完成。见图 3.11 程序流程图。

程序的主要处理函数都在源程序 EzMrView.cpp 中，图 中 USB 操作都是通过回调函数 bMainDlgProc 来完成的，源程序和头文件可见 EzMain.cpp 和 ezusbsys.h。

为了使用户能够灵活创建自己的应用程序，下面将详细介绍多线程编程和 IOCTL 函数调用。

在创建一个线程前，利用 CreateEvent 来创建一个新事件，用来监视线程的执行时间，并且设置各个操作的最长等待线程函数等待时间；然后利用函数 AfxBeginThread 创建线程，并通过线程参数、对话框句柄和消息句柄来实现响应操作，并将操作的结果显示出来；最后利用函数 WaitForSingleObject 检查线程的执行结果。

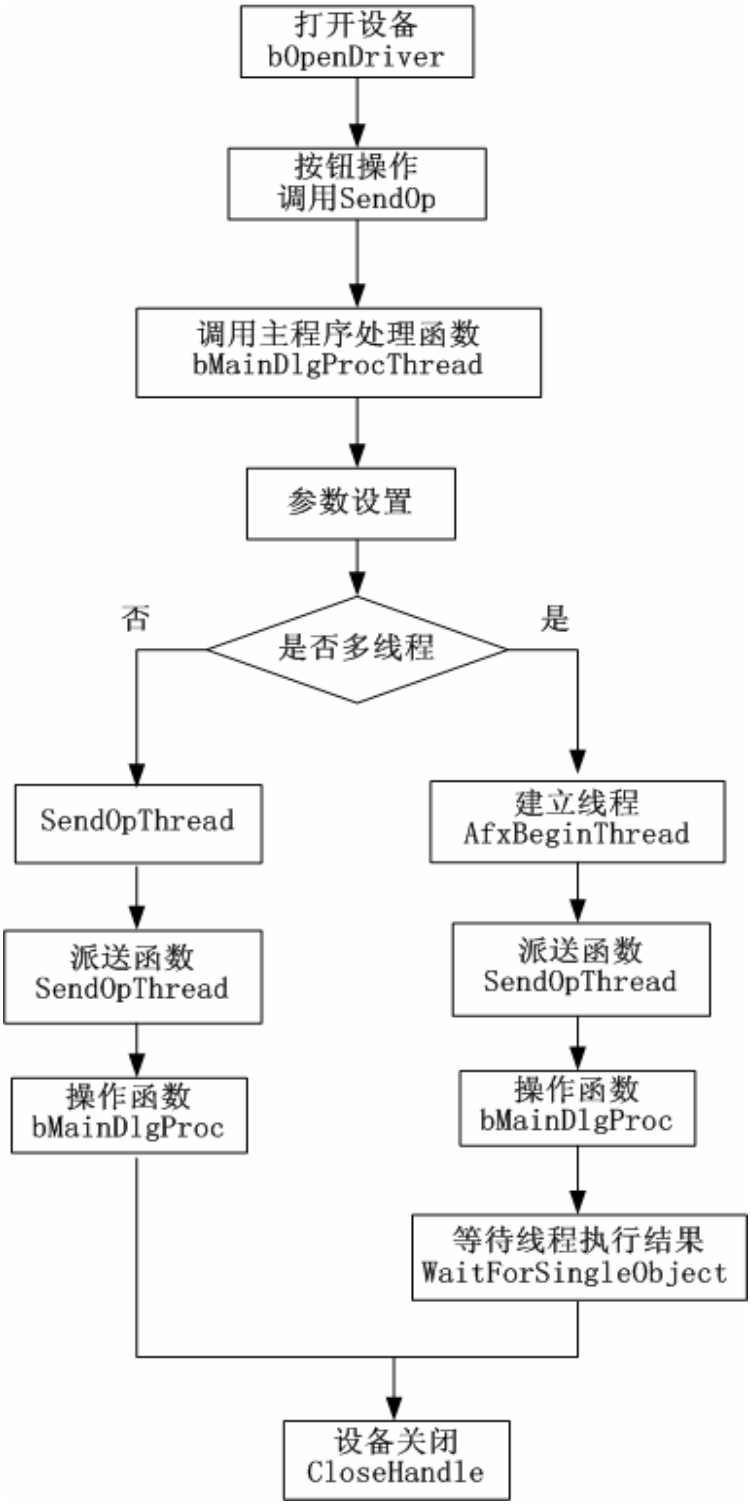


图 3.11 程序流程图

多线程代码：

```
//线程超时设置
int nWaitTime = 3000;//线程等待超时设置
if((pTh->wParam == IDC_ANCHOR_DOWNLOAD))
nWaitTime = 6000;//固件下载操作超时设置
if((pTh->wParam == IDC_VEND_REQUEST) && (pTh->request == 0xA2))
nWaitTime = 48000; // 如果是写数据到 E2PROM，延长超时等待为 48 秒
pTh->m_hEventOpDone = CreateEvent(NULL, TRUE, FALSE, NULL);
// 创建新事件，手动复位，初始化为复位状态
m_nOpsPending++;//挂起操作加 1
((CEzMrFrame*)(pTh->pParentFrame))->OpStart(pTh);
//线程创建
CWinThread* myWinThread =
AfxBeginThread(SendOpThread,pTh,THREAD_PRIORITY_HIGHEST);
//创建线程，SendOpThtread 为回调函数，pTh 为线程参数，THREAD_PRIORITY_HIGHEST 线
//程优先级
// Pause here and wait for a timely return from thread
TRACE("TPM:CEzMrView::bMainDlgProcThread(): Waiting after starting thread\n");
DWORD ThreadStat = WaitForSingleObject(pTh->m_hEventOpDone, nWaitTime);
//等待线程
CloseHandle(pTh->m_hEventOpDone);//关闭事件
switch(ThreadStat)//判断线程状态
{
case WAIT_OBJECT_0: //线程正常执行完成并快速退出
    TRACE("TPM:CEzMrView::bMainDlgProcThread(): Thread WAIT_OBJECT_0\n");
    bResult = 1;
    switch(wParam)
    { // Post Op processing
      case IDC_GET_PIPE_INFO:
        ((CEzMrFrame*)GetParentFrame())->UpdatePipes();
        break;
      case IDC_TRAN_BULK_DATA: // set length field as returned
        break;
      default:
        break;
    }
    ((CEzMrFrame*)(pTh->pParentFrame))->OpDone(pTh);
    delete (CThreadInfo*)pTh; // 清除线程参数，防止内存溢出
    break;
case WAIT_TIMEOUT: //线程等待超时,挂起操作，并保存线程参数，操作请求虽然已经挂起，
//但已发送到底层驱动程序，等待操作完成
    TRACE("TPM:CEzMrView::bMainDlgProcThread(): Thread WAIT_TIMEOUT\n");
    pTh->m_hOpPended = 1; // delayed operation - thread will clean up
    m_pTh[m_nOpsPending-1] = pTh; // save pointer in case we have to exit
```

```
    break;
case WAIT_ABANDONED: // 线程没有执行
    TRACE("TPM:CEzMrView::bMainDlgProcThread(): Thread WAIT_ABANDONED\n");
    break;
default: TRACE("TPM:CEzMrView::bMainDlgProcThread(): Thread STAT?\n");
    break;
```

线程回调函数代码：

```
UINT SendOpThread(LPVOID pParam)//线程函数，指针 pParam 为线程参数
{
int bResult;
CThreadInfo* pTh = (CThreadInfo*)pParam;
TRACE("TPM:CEzMrView::SendOpThread(): Thread started: Idx: %d\n", pTh->OpIndex);
//操作函数，执行 USB 读写请求
bResult = bMainDlgProc(pTh, pTh->hDlg, pTh->message, pTh->wParam, pTh->lParam );

if(pTh->m_hEventOpDone)//事件置位，使线程等待结束
  SetEvent(pTh->m_hEventOpDone); // Alert the main thread ASAP - it may still be waiting
if(pTh->pView)
  ((CEzMrView*)(pTh->pView))->m_nOpsPending--;//挂起计数减 1
if(pTh->m_hOpPended == 1)
{ // Op was pended - main task is no longer waiting - clean up here
  ((CEzMrFrame*)(pTh->pParentFrame))->OpDone((CThreadInfo*)pParam);
  delete (CThreadInfo*)pParam; // this makes the thread evaporate
}
return(bResult);//返回线程线程执行结果
}
```

3.2.2 驱动和 IOCTL 函数

在介绍控制（IOCTL）函数前，先介绍 FX2 通用驱动程序 ezusb.sys。驱动开发是整个 USB 开发中最难的部分，CYPRESS 公司提供了通用驱动程序（ezusb.sys）。除非特殊需要，不建议用户自己编写驱动程序。

驱动动开发工具有 DDK 和还有第三方开发工具，其中 DDK 开发难度最大，第三方开发工具有 DriverStudio 和 Windriver 等，DriverStudio 难度适中，而 Windriver 则属于应用层驱动开发，难度小，但效率低，并存在发布问题。下面主要介绍用 DDK 来开发 USB 驱动程序。

DDK 驱动程序开发工作包括：开发环境设置（VC 编译环境）、驱动程序设计、安装文件（INF 文件）设计。

DDK 的 VC 开发环境设置可以参考 Christ cant 书籍的的第四章的 4.6 和 4.7。

Win2000 和 WinXP 驱动程序设计采用 WDM（Windows Drive Mode），WDM 设备驱动程序提供了一个参考框架，大大降低了由 DDK 书写驱动程序带来的难度，如图 3.12 WDM 驱动模型。

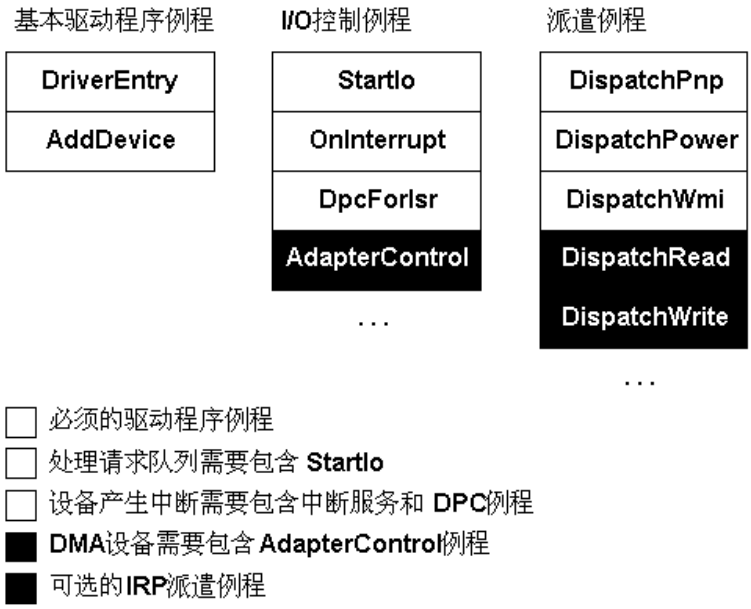


图 3.12　WDM 驱动模型

驱动工作包括：

• 初始化自己；

• 创建和删除设备；

• 处理打开和关闭文件句柄请求；

• 处理 WIN32 输入/输出（I/O）请求；

• 初始化对设备的请求；

• 访问硬件；

• 调用其它驱动程序；

• 取消 I/O 请求；

• 超时 I/O 请求；

• 处理可热插拔设备加入或删除情况；

• 处理电源请求；

• WMI（Windows 管理诊断）。

以下 EZUSB 的驱动程序入口函数：

```
NTSTATUS
DriverEntry(
      IN PDRIVER_OBJECT DriverObject,
      IN PUNICODE_STRING RegistryPath
      )
{
    NTSTATUS ntStatus = STATUS_SUCCESS;
    PDEVICE_OBJECT deviceObject = NULL;
    //
    // Create dispatch points for the various events handled by this
    // driver.    For example, device I/O control calls (e.g., when a Win32
    // application calls the DeviceIoControl function) will be dispatched to
```

```
    // routine specified below in the IRP_MJ_DEVICE_CONTROL case.
    //
    DriverObject->MajorFunction[IRP_MJ_CREATE] = Ezusb_Create;
    DriverObject->MajorFunction[IRP_MJ_CLOSE] = Ezusb_Close;
    DriverObject->DriverUnload = Ezusb_Unload;
    DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = Ezusb_ProcessIOCTL;
    DriverObject->MajorFunction[IRP_MJ_PNP] = Ezusb_DispatchPnp;
    DriverObject->MajorFunction[IRP_MJ_POWER] = Ezusb_DispatchPower;
    DriverObject->DriverExtension->AddDevice = Ezusb_PnPAddDevice;
    return ntStatus;
}
```

DriverEntry 入口函数，和 C 语言主程序中的 main()类似。驱动程序与应用程序和硬件之间通讯都是 IRP（I/O 请求包）来完成的，IRP_MJ_PNP 主要是实现 USB 即插即用，例如设备的添加、删除和资源的分配；IRP_MJ_POWER 来实现电源管理，例如设备的挂起和唤醒；IRP_MJ_CREATE(“创建”)、IRP_MJ_CLOSE(“关闭”)、IRP_MJ_DEVICE_CONTROL（“设备控制”），实现管道的创建、关闭和数据读写，其中“设备控制”具有输入输出缓冲区，可实现读和写功能；AddDevice 和 DriverUnload 来实现设备管理，在设备添加和卸载时，创建和删除设备，以及管理资源分配。

驱动调试，这是 USB 接口开发最困难的部分，调试工具可用 DriverStudio 中 Softice 工具和 Christ cant 中 DebugPrint 跟踪工具，监视工具 Bus Hound 可监视实际 USB 数据传输情况。

对于用户而言主要是需要了解应用程序中如何调用驱动提供的接口函数。

表 3.2 读写函数

Win32 API	DRIVER_FUNCTION_xxx IRP_MJ_xxx
CreateFile	CREATE
ReadFile	READ
WriteFile	WRITE
DeviceIoControl	DEVICE_CONTROL
CloseHandle	CLOSE　CLEANUP

见表 3.2，Win32 API 与驱动 IRP 的对应关系，总共有五个函数，所以调用起来不是很困难。

CreateFile 是通过设备名字打开设备，这个名字在驱动程序中已经注册了，并且获得设备句柄，有了设备句柄就可以读写设备了。

ReadFile 读设备，EZUSB 驱动没有定义 **IRP_MJ_*READ***，所以不能调用该函数来读设备。

WriteFile 写设备，EZUSB 驱动没有定义

CloseHandle 通过设备句柄关闭设备，请求完成之后**不要忘记关闭设备**。

DeviceIoControl 读写设备控制，所以通过该函数来完成所以的读写和其它请求。

下面是 DeviceIoControl 函数原型。

```
BOOL DeviceIoControl(
  HANDLE hDevice,                 // handle to device of interest
  DWORD dwIoControlCode,          // control code of operation to perform
  LPVOID lpInBuffer,              // pointer to buffer to supply input data
  DWORD nInBufferSize,            // size, in bytes, of input buffer
  LPVOID lpOutBuffer,             // pointer to buffer to receive output data
  DWORD nOutBufferSize,           // size, in bytes, of output buffer
```

LPDWORD *lpBytesReturned,* // pointer to variable to receive byte count
LPOVERLAPPED *lpOverlapped* // pointer to structure for asynchronous operation
);

hDevice 设备句柄，通过 CreateFile 函数得到；
DwIoControlCode 请求代码，驱动程序定义各种设备控制请求；
LpInBuffer 输入缓冲区指针；
NinBufferSize 输入缓冲区大小；
LpOutBuffer 输出缓冲区指针；
NoutBufferSize 输出缓冲区大小；
LpBytesReturned 返回字节数；
LpOverlapped 同步操作指针。
对于每个请求代码，重要的是输入正确 LpInBuffer，NinBufferSize， LpOutBuffer，NoutBufferSize 四个参数。
EZUSB 通用驱动程序所有的设备控制请求代码（*DwIoControlCode*）如下：

```
#define Ezusb_IOCTL_INDEX    0x0800 //IOCTL 请求代码索引
//得到端点信息
#define IOCTL_Ezusb_GET_PIPE_INFO        CTL_CODE(FILE_DEVICE_UNKNOWN,   \
                                                  Ezusb_IOCTL_INDEX+0,\
                                                  METHOD_BUFFERED,   \
                                                  FILE_ANY_ACCESS)
//得到设备描速符
#define                                    IOCTL_Ezusb_GET_DEVICE_DESCRIPTOR
CTL_CODE(FILE_DEVICE_UNKNOWN,   \
                                                  Ezusb_IOCTL_INDEX+1,\
                                                  METHOD_BUFFERED,   \
                                                  FILE_ANY_ACCESS)
//得到配置描速符符
#define                                IOCTL_Ezusb_GET_CONFIGURATION_DESCRIPTOR
CTL_CODE(FILE_DEVICE_UNKNOWN,   \
                                                  Ezusb_IOCTL_INDEX+2,\
                                                  METHOD_BUFFERED,   \
                                                  FILE_ANY_ACCESS)
//批量/中断写请求
#define                                  IOCTL_Ezusb_BULK_OR_INTERRUPT_WRITE
CTL_CODE(FILE_DEVICE_UNKNOWN,   \
                                                  Ezusb_IOCTL_INDEX+3,\
                                                  METHOD_BUFFERED,   \
                                                  FILE_ANY_ACCESS)
//批量/中断读请求
#define                                   IOCTL_Ezusb_BULK_OR_INTERRUPT_READ
CTL_CODE(FILE_DEVICE_UNKNOWN,   \
                                                  Ezusb_IOCTL_INDEX+4,\
                                                  METHOD_BUFFERED,   \
                                                  FILE_ANY_ACCESS)
```

```
//厂商请求
#define                                         IOCTL_Ezusb_VENDOR_REQUEST
CTL_CODE(FILE_DEVICE_UNKNOWN,   \
                                                Ezusb_IOCTL_INDEX+5,\
                                                METHOD_BUFFERED,   \
                                                FILE_ANY_ACCESS)
//得到当前配置描速符
#define                                       IOCTL_Ezusb_GET_CURRENT_CONFIG
CTL_CODE(FILE_DEVICE_UNKNOWN,   \
                                                Ezusb_IOCTL_INDEX+6,\
                                                METHOD_BUFFERED,   \
                                                FILE_ANY_ACCESS)
//固件下载请求
#define                                         IOCTL_Ezusb_ANCHOR_DOWNLOAD
CTL_CODE(FILE_DEVICE_UNKNOWN,   \
                                                Ezusb_IOCTL_INDEX+7,\
                                                METHOD_BUFFERED,   \
                                                FILE_ANY_ACCESS)
//复位 USB 设备
#define IOCTL_Ezusb_RESET    CTL_CODE(FILE_DEVICE_UNKNOWN,   \
                                                Ezusb_IOCTL_INDEX+12,\
                                                METHOD_IN_DIRECT,   \
                                                FILE_ANY_ACCESS)
//复位端点
#define IOCTL_Ezusb_RESETPIPE    CTL_CODE(FILE_DEVICE_UNKNOWN,   \
                                                Ezusb_IOCTL_INDEX+13,\
                                                METHOD_IN_DIRECT,   \
                                                FILE_ANY_ACCESS)
//放弃端点
#define IOCTL_Ezusb_ABORTPIPE    CTL_CODE(FILE_DEVICE_UNKNOWN,   \
                                                Ezusb_IOCTL_INDEX+15,\
                                                METHOD_IN_DIRECT,   \
                                                FILE_ANY_ACCESS)
//设置接口
#define IOCTL_Ezusb_SETINTERFACE    CTL_CODE(FILE_DEVICE_UNKNOWN,   \
                                                Ezusb_IOCTL_INDEX+16,\
                                                METHOD_BUFFERED,   \
                                                FILE_ANY_ACCESS)
//得到字符串描速符
#define                                       IOCTL_Ezusb_GET_STRING_DESCRIPTOR
CTL_CODE(FILE_DEVICE_UNKNOWN,   \
                                                Ezusb_IOCTL_INDEX+17,\
                                                METHOD_BUFFERED,   \
                                                FILE_ANY_ACCESS)
```

```
//批量传输读
// lpInBuffer: BULK_TRANSFER_CONTROL stucture specifying the pipe number to read from
// nInBufferSize: sizeof(BULK_TRANSFER_CONTROL)
// lpOutBuffer: Buffer to hold data read from the device.
// nOutputBufferSize: size of lpOutBuffer.    This parameter determines
// the size of the USB transfer.
// lpBytesReturned: actual number of bytes read
#define                                                 IOCTL_EZUSB_BULK_READ
CTL_CODE(FILE_DEVICE_UNKNOWN,   \
                                                        Ezusb_IOCTL_INDEX+19,\
                                                        METHOD_OUT_DIRECT,   \
                                                        FILE_ANY_ACCESS)
//批量传输写
// lpInBuffer: BULK_TRANSFER_CONTROL stucture specifying the pipe number to write to
// nInBufferSize: sizeof(BULK_TRANSFER_CONTROL)
// lpOutBuffer: Buffer of data to write to the device
// nOutputBufferSize: size of lpOutBuffer.    This parameter determines
//       the size of the USB transfer.
// lpBytesReturned: actual number of bytes written
//
#define                                                 IOCTL_EZUSB_BULK_WRITE
CTL_CODE(FILE_DEVICE_UNKNOWN,   \
                                                        Ezusb_IOCTL_INDEX+20,\
                                                        METHOD_IN_DIRECT,   \
                                                        FILE_ANY_ACCESS)
//得到当前帧号
// lpInBuffer: NULL
// nInBufferSize: 0
// lpOutBuffer: PULONG to hold current frame number
// nOutputBufferSize: sizeof(PULONG)
//
#define                                 IOCTL_EZUSB_GET_CURRENT_FRAME_NUMBER
CTL_CODE(FILE_DEVICE_UNKNOWN,   \
                                                        Ezusb_IOCTL_INDEX+21,\
                                                        METHOD_BUFFERED,   \
                                                        FILE_ANY_ACCESS)
//通过端点 0 执行厂商或者类请求
// lpInBuffer: PVENDOR_OR_CLASS_REQUEST_CONTROL
// nInBufferSize: sizeof(VENDOR_OR_CLASS_REQUEST_CONTROL)
// lpOutBuffer: pointer to a buffer if the request involves a data transfer
// nOutputBufferSize: size of the transfer buffer (corresponds to the wLength
// field of the USB setup packet)
#define                                 IOCTL_EZUSB_VENDOR_OR_CLASS_REQUEST
CTL_CODE(FILE_DEVICE_UNKNOWN,   \
```

```
                                        Ezusb_IOCTL_INDEX+22,\
                                        METHOD_IN_DIRECT,   \
                                        FILE_ANY_ACCESS)
//最后一次 URB 错误传输状态
// lpInBuffer: NULL
// nInBufferSize: 0
// lpOutBuffer: PULONG to hold the URB status
// nOutputBufferSize: sizeof(ULONG)
#define IOCTL_EZUSB_GET_LAST_ERROR      CTL_CODE(FILE_DEVICE_UNKNOWN, \
                                        Ezusb_IOCTL_INDEX+23,\
                                        METHOD_BUFFERED,   \
                                        FILE_ANY_ACCESS)

//等时传输读
// Reads from the specified ISO endpoint. (USB IN Transfer)
// lpInBuffer: ISO_TRANSFER_CONTROL
// nInBufferSize: sizeof(ISO_TRANSFER_CONTROL)
// lpOutBuffer: buffer to hold data read from the device
// nOutputBufferSize: size of the read buffer.
#define IOCTL_EZUSB_ISO_READ            CTL_CODE(FILE_DEVICE_UNKNOWN,   \
                                        Ezusb_IOCTL_INDEX+25,\
                                        METHOD_OUT_DIRECT,   \
                                        FILE_ANY_ACCESS)
//  等时传输写
// lpInBuffer: ISO_TRANSFER_CONTROL
// nInBufferSize: sizeof(ISO_TRANSFER_CONTROL)
// lpOutBuffer: buffer to hold data to write to the device
// nOutputBufferSize: size of the write buffer.
#define IOCTL_EZUSB_ISO_WRITE           CTL_CODE(FILE_DEVICE_UNKNOWN, \
                                        Ezusb_IOCTL_INDEX+26,\
                                        METHOD_IN_DIRECT,   \
                                        FILE_ANY_ACCESS)
//  固件下载
// lpInBuffer: PANCHOR_DOWNLOAD_CONTROL
// nInBufferSize: sizeof(ANCHOR_DOWNLOAD_CONTROL)
// lpOutBuffer: pointer to a buffer of data to download to the device
// nOutputBufferSize: size of the transfer buffer
//
#define IOCTL_EZUSB_ANCHOR_DOWNLOAD CTL_CODE(FILE_DEVICE_UNKNOWN, \
                                        Ezusb_IOCTL_INDEX+27,\
                                        METHOD_IN_DIRECT,   \
                                        FILE_ANY_ACCESS)
//    返回驱动程序的最新版本
// lpInBuffer: NULL
```

```
// nInBufferSize: 0
// lpOutBuffer: PEZUSB_DRIVER_VERSION
// nOutputBufferSize: sizeof(EZUSB_DRIVER_VERSION)
//
#define                                    IOCTL_EZUSB_GET_DRIVER_VERSION
CTL_CODE(FILE_DEVICE_UNKNOWN,\
                                                Ezusb_IOCTL_INDEX+29,\
                                                METHOD_BUFFERED,   \
                                                FILE_ANY_ACCESS)
//启动等时传输
#define IOCTL_EZUSB_START_ISO_STREAM    CTL_CODE(FILE_DEVICE_UNKNOWN, \
                                                Ezusb_IOCTL_INDEX+30,\
                                                METHOD_BUFFERED,   \
                                                FILE_ANY_ACCESS)
//关闭等时传输
#define IOCTL_EZUSB_STOP_ISO_STREAM    CTL_CODE(FILE_DEVICE_UNKNOWN, \
                                                Ezusb_IOCTL_INDEX+31,\
                                                METHOD_BUFFERED,   \
                                                FILE_ANY_ACCESS)
//读取等时传输的缓冲区
#define IOCTL_EZUSB_READ_ISO_BUFFER    CTL_CODE(FILE_DEVICE_UNKNOWN, \
                                                Ezusb_IOCTL_INDEX+32,\
                                                METHOD_OUT_DIRECT,   \
                                                FILE_ANY_ACCESS)
//设置特征（USB 标准请求）
#define IOCTL_EZUSB_SET_FEATURE       CTL_CODE(FILE_DEVICE_UNKNOWN, \
                                                Ezusb_IOCTL_INDEX+33,\
                                                METHOD_BUFFERED,   \
                                                FILE_ANY_ACCESS)
```

具体各个控制请求函数的使用方法，可以参考源程序 EzMain.cpp，详细了解每个控制请求的调用和参数设置。

第四章 固件编程框架

4.1 功能介绍

为了简化固件编程，CYPRESS 提供了固件编程框架，在此基础上用户只需修改少量代码就可以完成固件编程。固件编程框架已将 USB 标准请求和 USB 电源管理，并且提供了钩子函数，用户在钩子函数中输入少量代码就可以完成编程。用户也可以直接编程，但是编程难度

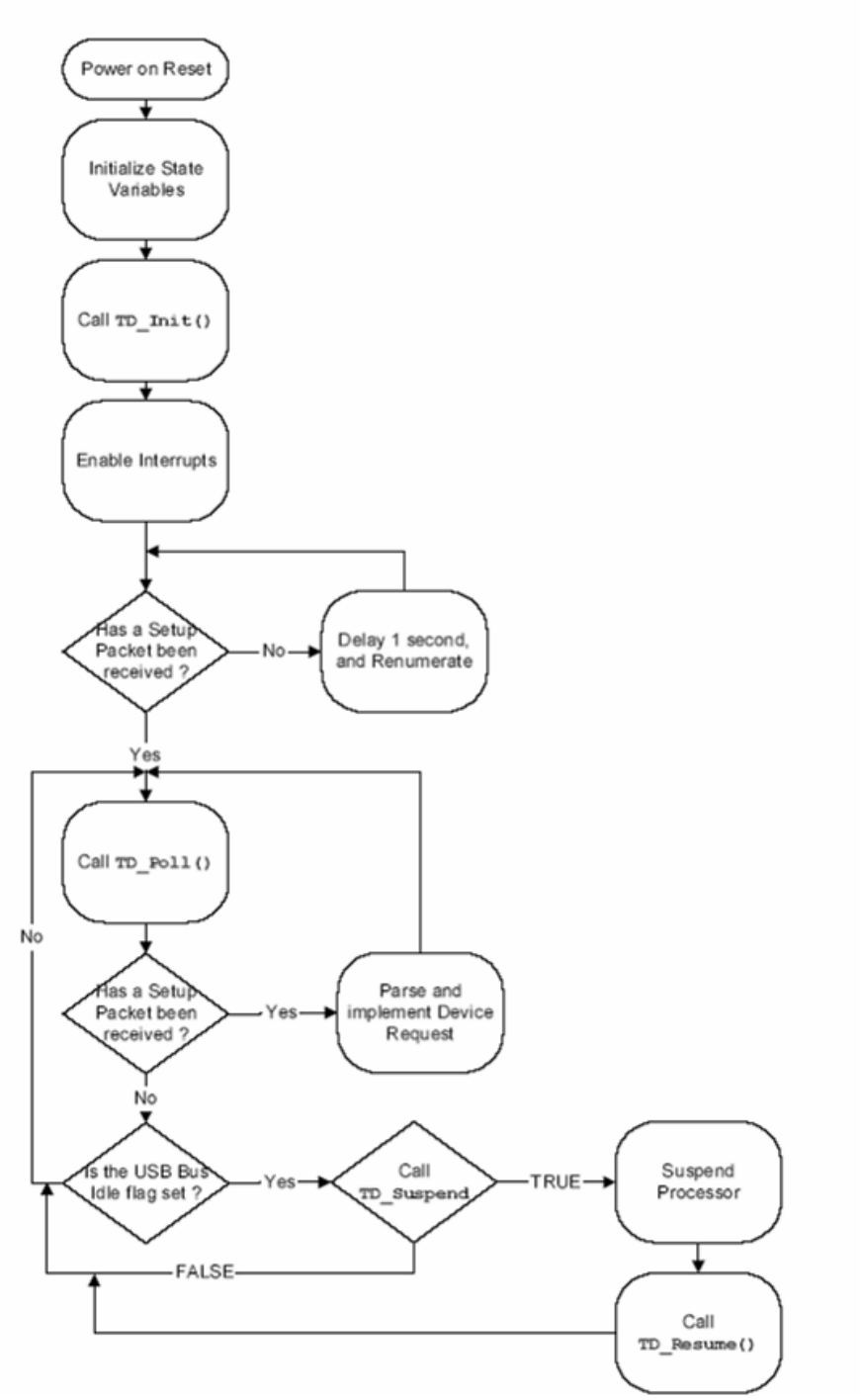


图 4.1　固件程序框架

较大，使用编程框架可降低开发难度，但是造成程序量过大，用户可以根据自己的工程来选择编程方式。

固件程序框架见图 4.1，复位上电时，固件先初始化一些全局变量，接着调用初始化函数 TD_Init（），初始化设备到没有配置的状态和开中断，循环延时 1 秒后重枚举，直到端点 0 接收到 SETUP 包退出循环，进入循环语句 while，执行任务调度，函数包括

（1）TD_POLL（）为用户任务函数

（2）如果发现 USB 请求函数，则执行对应 USB 请求操作。

（3）如果发现 USB 空闲置位，则调用 TD_Suspend（）挂起函数，调用成功则内核挂起，直到出现 USB 远程唤醒信号，调用 TD_Resume（），内核唤醒重新进入 while 循环。

建立框架所需的文件见表 4.1。

表4.1 固件框架源文件和头文件

Reg80320.h	8051 头文件，由KEIL提供
Ezusb.h	库函数申明，以及变量、宏定义，数据类型定义
Fx2regs.h	FX2寄存器头文件
Fw.c	固件框架源文件
Periph.c	用户钩子函数，用户可修改，在不同的应用中文件名不一样
Dscr.a51	USB描述符列表，用户可修改
Ezusb.lib	EZUSB库文件
USBJmpTb.OBJ	中断跳转函数目标文件

实际编程中一般只修改Periph.c 和Dscr.a51两个文件。

4.2 主程序源代码

下面主程序源代码：

```
void main(void)//主程序中实现任务派送。
{
    DWORD     i;
    WORD      offset;
    DWORD     DevDescrLen;
    DWORD     j=0;
    WORD      IntDescrAddr;
    WORD      ExtDescrAddr;

    // 初始化全局变量
    Sleep = FALSE;                    // Disable sleep mode
    Rwuen = FALSE;                    // Disable remote wakeup
    Selfpwr = FALSE;                  // Disable self powered
    GotSUD = FALSE;                   // Clear "Got setup data" flag

    // 初始化用户设备
    TD_Init();
    //描述符地址重映射
    pDeviceDscr = (WORD)&DeviceDscr;
```

```
    pDeviceQualDscr = (WORD)&DeviceQualDscr;
    pHighSpeedConfigDscr = (WORD)&HighSpeedConfigDscr;
    pFullSpeedConfigDscr = (WORD)&FullSpeedConfigDscr;
    pStringDscr = (WORD)&StringDscr;
    if ((WORD)&DeviceDscr & 0xe000)
    {
        IntDescrAddr = INTERNAL_DSCR_ADDR;
        ExtDescrAddr = (WORD)&DeviceDscr;
        DevDescrLen = (WORD)&UserDscr - (WORD)&DeviceDscr + 2;
        for (i = 0; i < DevDescrLen; i++)
            *((BYTE xdata *)IntDescrAddr+i) = 0xCD;
        for (i = 0; i < DevDescrLen; i++)
            *((BYTE xdata *)IntDescrAddr+i) = *((BYTE xdata *)ExtDescrAddr+i);
        pDeviceDscr = IntDescrAddr;
        offset = (WORD)&DeviceDscr - INTERNAL_DSCR_ADDR;
        pDeviceQualDscr -= offset;
        pConfigDscr -= offset;
        pOtherConfigDscr -= offset;
        pHighSpeedConfigDscr -= offset;
        pFullSpeedConfigDscr -= offset;
        pStringDscr -= offset;
    }
//开中断
    EZUSB_IRQ_ENABLE();                 // Enable USB interrupt (INT2)
    EZUSB_ENABLE_RSMIRQ();              // Wake-up interrupt
    INTSETUP |= (bmAV2EN | bmAV4EN);    // Enable INT 2 & 4 autovectoring
    USBIE |= bmSUDAV | bmSUTOK | bmSUSP | bmURES | bmHSGRANT;
// Enable selected interrupts
    EA = 1;                             // Enable 8051 interrupts
//USB 重枚举，如果固件程序是从 E2PROM 加载的该步骤没有必要
#ifndef NO_RENUM
    // Renumerate if necessary.  Do this by checking the renum bit.  If it
    // is already set, there is no need to renumerate.  The renum bit will
    // already be set if this firmware was loaded from an eeprom.
    if(!(USBCS & bmRENUM))
    {
        EZUSB_Discon(TRUE);   // renumerate
    }
#endif
    USBCS &=~bmDISCON;
    CKCON = (CKCON&(~bmSTRETCH)) | FW_STRETCH_VALUE;
    // Set stretch to 0 (after renumeration)
    // clear the Sleep flag.
    Sleep = FALSE;
```

```
    // 任务调度
    while(TRUE)                 // 主循环
    {
       if(GotSUD)               // 端点 0 是否有 SETUP 包
       {
            SetupCommand();             // 执行 USB 请求
            GotSUD = FALSE;             // 清除标志
       }
       // 说明：一旦设备挂起，进入空闲模式，则系统时钟将停止，只有两种方式可以唤
       //醒 USB 设备，一是 WAKEUP 引脚，二是 USB 总线探测到 resume 状态
       if (Sleep)//检查 Sleep 标志
           {
           if(TD_Suspend())//执行 USB 挂起钩子函数
               {
               Sleep = FALSE; //清除标志
               do
                   {
                     EZUSB_Susp();       // 使处理器进入 IDLE 状态，直到 USB 唤醒
                   }
                 while(!Rwuen && EZUSB_EXTWAKEUP());//如果远程唤醒未使能，并且未
                                                    //出现唤醒信号
              EZUSB_Resume();    //如果 WAKEUP#引脚出现低电平，USB 执行唤醒操作
              TD_Resume();     //唤醒钩子函数
               }
           }
       TD_Poll();//用户函数
    }
}
```

实际编程中用户一般只需要修改 TD_Init()和 TD_Poll()两个钩子函数。

4.3 钩子函数

为了方便用户编程，编程框架提供了钩子函数，用户只要根据工程改写钩子函数就可以轻松完成固件编程。

void TD_Init(void)

描述: 在 FX2 重枚举后首先调用，用户可以在进行全局变量和 FX2 寄存器的初始化操作。

void TD_Poll(void)

描述: 该函数在 main 函数 while 程序模块中，除非被高优先级中断所打断，该函数将被重复调用，所以用户可在此添加自己所要实现的功能。

BOOL TD_Suspend(void)

描述:该函数在 USB 设备进入挂起前调用,用户可在此添加进入挂起前需要进行的准备工作，函数返回 TRUE 则设备进入挂起，函数返回 FALSE 则阻止框架进入挂起模式。

void TD_Resume(void)

描速：当有一个外部唤醒事件时，首先调用此函数。

BOOL DR_GetDescriptor(void)

描速：在得到设备描述符的 USB 标准请求之前调用该函数，函数返回 TRUE，则执行请求操

作，返回 FALSE 则不理会该请求。
以下设备请求，基本同上。
BOOL DR_GetInterface(void)
BOOL DR_SetInterface(void)
BOOL DR_GetConfiguration(void)
BOOL DR_SetConfiguration(void)
BOOL DR_GetStatus(void)
BOOL DR_ClearFeature(void)
BOOL DR_SetFeature(void)
BOOL DR_VendorCmnd(void)
描速：厂商请求调用是调用，用户可再次添加自己定义的请求。
void ISR_Sudav(void) interrupt 0
void ISR_Sutok(void) interrupt 0
void ISR_Sof(void) interrupt 0
void ISR_Ures(void) interrupt 0
void ISR_Susp(void) interrupt 0
void ISR_Highspeed(void) interrupt 0
void ISR_Ep0ack(void) interrupt 0
void ISR_Stub(void) interrupt 0
void ISR_Ep0in(void) interrupt 0
void ISR_Ep0out(void) interrupt 0
void ISR_Ep1in(void) interrupt 0
void ISR_Ep1out(void) interrupt 0
void ISR_Ep2inout(void) interrupt 0
void ISR_Ep4inout(void) interrupt 0
void ISR_Ep6inout(void) interrupt 0
void ISR_Ep8inout(void) interrupt 0
void ISR_Ibn(void) interrupt 0
void ISR_Ep0pingnak(void) interrupt 0
void ISR_Ep1pingnak(void) interrupt 0
void ISR_Ep2pingnak(void) interrupt 0
void ISR_Ep4pingnak(void) interrupt 0
void ISR_Ep6pingnak(void) interrupt 0
void ISR_Ep8pingnak(void) interrupt 0
void ISR_Errorlimit(void) interrupt 0
void ISR_Ep2piderror(void) interrupt 0
void ISR_Ep4piderror(void) interrupt 0
void ISR_Ep6piderror(void) interrupt 0
void ISR_Ep8piderror(void) interrupt 0
void ISR_Ep2pflag(void) interrupt 0
void ISR_Ep4pflag(void) interrupt 0
void ISR_Ep6pflag(void) interrupt 0
void ISR_Ep8pflag(void) interrupt 0
void ISR_Ep2eflag(void) interrupt 0

```
void ISR_Ep4eflag(void) interrupt 0
void ISR_Ep6eflag(void) interrupt 0
void ISR_Ep8eflag(void) interrupt 0
void ISR_Ep2fflag(void) interrupt 0
void ISR_Ep4fflag(void) interrupt 0
void ISR_Ep6fflag(void) interrupt 0
void ISR_Ep8fflag(void) interrupt 0
void ISR_GpifComplete(void) interrupt 0
void ISR_GpifWaveform(void) interrupt 0
```

为了能够响应 USB 中断，首先要使能，对于 USB 中断和 GPIF 中断，还要使能中断矢量，中断完成后要清除中断请求标志。

第五章 硬件说明

5.1 电路原理图

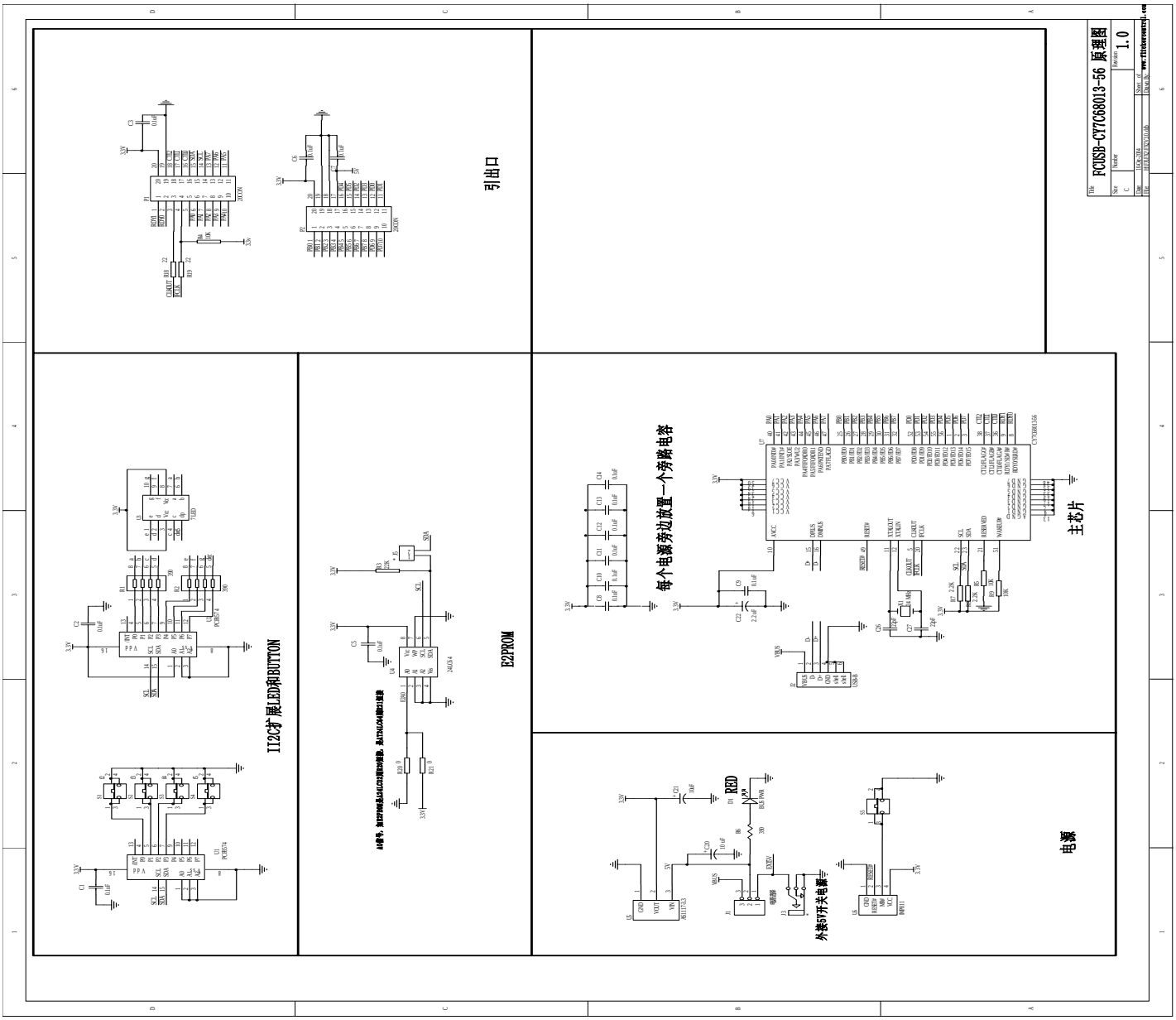


图 5.1 开发板原理图

开发板电路原理图见图 5.1，电路由五部分组成：

1. 电源供应：USB 接口提供的为 5V/500mA 电源， 68013 的工作电压为 3.3V，所以要进行 5V 到 3.3V 电源转换，采用常用芯片 AS1117-3.3V，在实际应用中，用户可根据系统功率大小来选择芯片封装，大功率时大封装散热性优于小封装。J1 跳针可进行供电选择:USB 总线供电或外接供电，外接供电座 J3，可连接到 5V 开关电源。D1 小灯为电源指示灯。

2. 主芯片：本开发板采用 56 引脚封装 CY7C68013，和 128 引脚封装相比，不能外扩 SRAM 和没有串口，所以不能进行在线仿真，但是可通过厂商请求来进行简单调试。56 引脚封装特点是设计简单且低成本。

3.E2PROM：如果是小容量 E2PROM，如 24LC02，可存放 PID/VID，实现用户定制，如果是大容量 E2PROM，如 24LC64（8KB），可存放 PID/VID 程序。采用大小容量 E2PROM，主要区别引脚 A0 接地或接高，本开发板默认接 24LC02，故将 R20 短接。J5 跳针可控制 E2PROM 连接，缺省断口，即无 E2PROM。

4.I2C 扩展 LED 和 BUTTON：电路中采用了两片 PCF8574 I2C 总线驱动芯片，用来驱动 8 段数码管和小灯，编程中经常用来做指示。

5.引出口：引出口将 68013 主要引脚都已经引出，方便用户在此基础做自己应用。

说明:本开发板预留了 AD 采集接口，电路板上没有连接。

5.2 器件清单

表 5.1 器件清单

编号	器件类型	封装
J3	电源插座	POWER_PLUG
J5	E2PROM 跳针，默认断开	SIP2
R20,R21	0	0402
C1,C2,C3,C5,C6,C7 , C8,C9,C10, C11,C12 , C13 ,C14	0.1uF	0805
R7,R8	2.2K	0805
C22	2.2 uF	1206
U3	8 段数码管	7SEGLED
R4, R5 ,R9	10K	805
C20,C21	10 uF	C1210E
P1,P2	引出排针	IDC20
R18,R19	22	0805
R3	22K	0805
C26,C27	22Pf	0805
U4	24LC02	SO-8
X1	24 MHz 晶体	XAL
R6	390	0805
R1, R2	390 阻排	RESPACK
U5	AS1117-3.3	SOT-223
D1	电源指示灯	1206
U7	CY7C68013-56	SSOP56
U6	IMP811 复位芯片	SOT-143
U1,U2	PCF8574T	SOL-16
J2	USB B 型插座	USB-B
S1,S2, S3,S4	按钮	SW
J1	电源选择跳针，默认 1,2 短接	SIP3

5.3 电路设计注意事项

1. EA 引脚通过 10K 电阻接地，否则不能正确找到设备。
2. RESERVED 引脚通过 10K 电阻接地，否则不能正确找到设备。
3. 靠近各电源引脚地方接旁路电容。
4. I2C 引脚 SDA，SCL 必须接 1.5~10K 的的上拉电阻，即使上面没有 I2C 设备。
5. 晶体两旁的电容大小范围为 22~30PF。
6. WAKEUP#引脚没有使用时，通过 10K 电阻上拉。
7. 复位引建议采用专门的复位芯片，或接 10K/1UF RC 电路，使系统正常上电复位。
8. 为了提供 D+ D-抗干扰能力，可接瞬态脉冲抑制芯片，如 TI 公司的 SN75240，同时保证连接正确。
9. 如果使用 3.3V CMOS 有源晶振，输入接 XIN，XOUT 可作为输出。
10. 芯片的 I/O 兼容 5V TTL 电平，另外，可与 5V CMOS 输出连接，但是不能驱动 5V CMOS 电平。
11. 没有用到的引脚不能悬空，对于输入引脚，通过一个电阻上拉或者下拉；输出引脚，可直接接地或者电源。

5.4 第一次开始使用

先必须安装开发板光盘下的 CYPRESS 开发包“EZ-USB_devtools_version_261700.exe”，EZUSB 控制面板和其它文件安装完成。然后插入开发板，对于 WINXP 操作系统，会提示安装驱动程序，选择“自动安装软件”即可，在设备管理中“通用串行总线控制器”会出现“Cypress EZ-USB FX2 (68613) - EEPROM missing”一项，驱动安装完成；对于 WIN2000 设备，设备驱动自动识别，不会驱动安装过程，设备管理器中 USB 设备列表下出现 FX2 设备。此时会提示“高速 USB 设备插入到非高速集线器”，不用理会，下面调试例程一章将介绍。

为了测试开发板工作正常，插入开发板，此时电源指示灯亮，打开 EZUSB 控制面板，点击按钮“DOWNLOAD..”，选择开发板光盘“\src\dev_io”目录下 dev_io.hex，然后下载，按下按钮 S1、S2、S3 和 S4，8 段数码管显示将变化，按下 S1 数码管复位为 0，按下 S4 数码管复位为 F，按下 S2 递增显示，按下 S3 递减显示，则开发板工作正常。

第六章 调试实例

本章将详细介绍 FX2 编程，通过这些编程实例，能够了解 USB 各种类型的数据传输和寄存器操作，寄存器介绍详见第八章介绍。

6.1 控制传输例程

端点 0 固定配置为控制传输类型，通过控制传输，主要进行 USB 设备请求，请求类型包括 USB 标准请求，设备类请求，厂商请求。请求格式见第一章 1.1.4 节，建立阶段 SETUP 包格式见表 6.1。

表6.1 SETUP数据包中的八个字节

字节	域	说明
0	bmRequestType	请求类型，方向，接受者（设备,配置，接口或端点）.
1	bRequest	设备请求6.2(FX2定义设备请求见表6.2).
2	wValueL	16位请求值
3	wValueH	
4	wIndexL	请求索引，根据请求类型确定
5	wIndexH	
6	wLengthL	数据阶段的数据包长度
7	wLengthH	

表6.2 重枚举后(RENUM＝1)由固件处理设备请求，而不是USB SIE

请求	名称	FX2 动作	固件响应
0x00	Get Status	SUDAV中断	提供USB供电模式，挂起和远程唤醒属性
0x01	Clear Feature	SUDAV中断	清除USB供电模式，挂起和远程唤醒能力
0x02	(保留)	无	停止端点0
0x03	Set Feature	SUDAV中断	设置USB供电模式，挂起和远程唤醒属性
0x04	(保留)	无	停止端点0
0x05	Set Address	分配设备一个地址	无
0x06	Get Descriptor	SUDAV中断	提供设备描述符
0x07	Set Descriptor	SUDAV中断	取决于应用
0x08	Get Configuration	SUDAV中断	发送当前配置信息
0x09	Set Configuration	SUDAV中断	改变当前配置
0x0A	Get Interface	SUDAV中断	从RAM提供可选的接口号
0x0B	Set Interface	SUDAV中断	设置可选接口号
0x0C	Sync Frame	SUDAV中断	提供帧计数号
厂商请求			
0xA0 （固件加载）		上传/下载外部RAM	---
0xA1 – 0xAF		SUDAV中断	Cypress公司保留
其它		SUDAV中断	用户可定义

CPRESS保留厂商请求：

```
#define VR_ANCHOR_DLD     0xa0 // 固件加载
#define VR_EEPROM         0xa2 // 上传/下载EEPROM
#define  VR_RAM           0xa3 // 上传/下载外部RAM
#define VR_SETI2CADDR 0xa4 //设置I2C地址
#define VR_GETI2C_TYPE    0xa5 //得到EEPROM类型，8位、16位
#define VR_GET_CHIP_REV 0xa6 // 得到芯片版本号Rev A, B = 0, Rev C = 2
#define VR_TEST_MEM       0xa7 // 测试外部RAM，并返回执行结果
#define VR_RENUM          0xa8 // FX2重枚举
#define VR_DB_FX          0xa9 // 强迫使用双字节地址(仅对FX)
#define VR_I2C_100        0xaa //设置I2C总线为100Khz模式
#define VR_I2C_400        0xab ///设置I2C总线为100Khz模式
#define VR_NOSDPAUTO    0xac // 测试代码，使用SUDPTR时，使能自动计算长度功能，
                                    //改用手动传输设置长度
```

为了区分，用户使用厂商请求时，最好不要占用USB标准请求号和CYPRESS保留的请求号。如果请求号没有定义，固件将停止端点0。

对于USB标准请求，可直接由USB核来处理，例如得到设备描述符(GD_DEVICE),使用建立指针(SUDPTR) ,只需设置SUDPTRH和SUDPTRL，如SUDPTRH = MSB(pDeviceDscr);
SUDPTRL = LSB(pDeviceDscr);返回数据长度将自动设备，固件不用设置。

对于厂商请求，如果需要返回数据，则需设置端点0，方法如下：

```
#define VR_GET_FRAMENO 0xb0//得到帧计数
#define VR_GET_MFRAMENO 0xb1//得到微帧计数
switch(SETUPDAT[1])//判断请求类型
{
    case VR_GET_FRAMENO:
                *EP0BUF = USBFRAMEL;
                *(EP0BUF+1)=USBFRAMEH;
                EP0BCH = 0;
                EP0BCL = 2;
                EP0CS |= bmHSNAK;
        break;
    case VR_GET_MFRAMENO:
                *EP0BUF = MICROFRAME;
                EP0BCH = 0;
                EP0BCL = 1;
                EP0CS |= bmHSNAK;
        break;
    case VR_GETI2C_TYPE:
                *EP0BUF = DB_Addr;//需要回传数据
                EP0BCH = 0;
                EP0BCL = 1; // 设置端点 0 缓冲区计数
                EP0CS |= bmHSNAK; //握手阶段响应主机，
        break;
    case VR_GET_CHIP_REV:
                ChipRev = GET_CHIP_REV();
```

```
                *EP0BUF = ChipRev;
                EP0BCH = 0;
                EP0BCL = 1;
                EP0CS |= bmHSNAK;
            break;
}……
```

源代码见光盘目录“\src\Vend_ax”中源文件VEND_AX.C函数DR_VendorCmnd。
通过控制面板将生成的Vend_ax.hex下载到FX2，将J5短接，能够操作E2PROM。再通过控制面板执行厂商请求，设置如图6.1，然后点击VendReq，在消息窗口可以看到usb数据帧号。

图6.1 得到USB数据帧号厂商请求

其它厂商请求，用户可以根据源代码来自己设置，下面主要介绍E2PROM和外部RAM厂商请求，E2PROM请求号0xA2,RAM请求号0xA3,请求值为操作起始地址,请求索引不计,Dir传输方向:IN上传，OUT下载，如果是OUT，则Hex Bytes为将要下载的十六进制数据。
例如Req=0xA2,Value=0x0000，Length=100，Dir=IN,点击VenReq，在消息窗口将看到E2PROM从地址0开始的100字节数据。

利用厂商请求，可读取感兴趣全局变量和FX2寄存器的值，从而实现简单调试。

6.2 中断传输例程

例程源代码见光盘目录 src\intTest。
首先是要设置配置描述符，设置 USB 接口的一些属性，这些属性可通过控制面板的得到设备描述符、配置描述符、端点描述符以及字符串描述符。蓝色和灰色部分为开发中常修改部分。

```
DeviceDscr:     ;;设备描述符
        db      DSCR_DEVICE_LEN     ;; Descriptor length 描述符长度
        db      DSCR_DEVICE     ;; Decriptor type 描述符长度
        dw      0002H           ;; Specification Version (BCD)  设备版本号
        db      00H             ;; Device class 设备类
        db      00H             ;; Device sub-class 设备子类   通用设备类和子类均设置为零
        db      00H             ;; Device sub-sub-class 设备子子类
        db      64              ;; Maximum packet size 控制端点最大包大小
        dw      4705H           ;; Vendor ID 厂商 ID，
        ;;设备驱动程序加载是通过厂商 ID 和产品 ID 来确定
        dw      0210H           ;; Product ID (Sample Device)产品 ID
        dw      0000H           ;; Product version ID 产品版本号
        db      1               ;; Manufacturer string index 厂商字符串索引 1,为 StringDscr1
        db      2               ;; Product string index 产品字符串描述符 2，为 StringDscr2
        db      0               ;; Serial number string index，为 StringDscr0
        db      1               ;; Number of configurations 配置数目 1
DeviceQualDscr:;;设备限定描述符
        db      DSCR_DEVQUAL_LEN    ;; Descriptor length
        db      DSCR_DEVQUAL    ;; Decriptor type
```

```
        dw      0002H           ;; Specification Version (BCD)
        db      00H             ;; Device class
        db      00H             ;; Device sub-class
        db      00H             ;; Device sub-sub-class
        db      64              ;; Maximum packet size
        db      1               ;; Number of configurations
        db      0               ;; Reserved

HighSpeedConfigDscr:    ;;高速 USB 配置描述符
        db      DSCR_CONFIG_LEN                 ;; Descriptor length
        db      DSCR_CONFIG                     ;; Descriptor type
        db      (HighSpeedConfigDscrEnd-HighSpeedConfigDscr) mod 256
        ;; Total Length (LSB)
        db      (HighSpeedConfigDscrEnd-HighSpeedConfigDscr)  /  256
        ;; Total Length (MSB)
        db      1           ;; Number of interfaces
        db      1           ;; Configuration number
        db      0           ;; Configuration string
        db      10000000b   ;;  配置属性 Attributes
        ;;(b7 – buspwr 总线供电, b6 – selfpwr, b5 – rwu)
        db      50          ;; Power requirement (div 2 ma)
        ;;最大电流 50×2＝100mA，可根据需要更改

        ;; Interface Descriptor 接口描述符
        db      DSCR_INTRFC_LEN         ;; Descriptor length
        db      DSCR_INTRFC             ;; Descriptor type
        db      0                       ;; Zero-based index of this interface
        db      0                       ;; Alternate setting 接口设置 0
        db      2                       ;; Number of end points 端点数目
        db      0ffH                    ;; Interface class
        db      00H                     ;; Interface sub class
        db      00H                     ;; Interface sub sub class
        db      0                       ;; Interface descriptor string index

;; Endpoint Descriptor 端点描述符，端点 2 OUT
        db      DSCR_ENDPNT_LEN         ;; Descriptor length
        db      DSCR_ENDPNT             ;; Descriptor type
        db      02H                     ;; Endpoint number, and direction
        db      ET_INT                  ;; Endpoint type 端点类型
        db      00H                     ;; Maximun packet size (LSB)
        db      02H                     ;; Max packect size (MSB)  包大小为 0x200 字节
        db      10H                     ;; Polling interval 轮询间隔 16ms
;; Endpoint Descriptor 端点描述符  端点 6 IN
        db      DSCR_ENDPNT_LEN         ;; Descriptor length
```

```
        db      DSCR_ENDPNT             ;; Descriptor type
        db      86H                     ;; Endpoint number, and direction
        db      ET_INT                  ;; Endpoint type
        db      00H                     ;; Maximun packet size (LSB)
        db      02H                     ;; Max packect size (MSB)
        db      10H                     ;; Polling interval
HighSpeedConfigDscrEnd:

FullSpeedConfigDscr:    ;;全速 USB 设备描述符
        db      DSCR_CONFIG_LEN                 ;; Descriptor length
        db      DSCR_CONFIG                     ;; Descriptor type
        db      (FullSpeedConfigDscrEnd-FullSpeedConfigDscr) mod 256
        ;; Total Length (LSB)
        db      (FullSpeedConfigDscrEnd-FullSpeedConfigDscr)  /  256
        ;; Total Length (MSB)
        db      1       ;; Number of interfaces
        db      1       ;; Configuration number
        db      0       ;; Configuration string
        db      10000000b       ;; Attributes (b7 - buspwr, b6 - selfpwr, b5 - rwu)
        db      50      ;; Power requirement (div 2 ma)

;; Interface Descriptor
        db      DSCR_INTRFC_LEN         ;; Descriptor length
        db      DSCR_INTRFC             ;; Descriptor type
        db      0                       ;; Zero-based index of this interface
        db      0                       ;; Alternate setting
        db      2                       ;; Number of end points
        db      0ffH                    ;; Interface class
        db      00H                     ;; Interface sub class
        db      00H                     ;; Interface sub sub class
        db      0                       ;; Interface descriptor string index

;; Endpoint Descriptor
        db      DSCR_ENDPNT_LEN         ;; Descriptor length
        db      DSCR_ENDPNT             ;; Descriptor type
        db      02H                     ;; Endpoint number, and direction
        db      ET_INT                  ;; Endpoint type
        db      40H                     ;; Maximun packet size (LSB)
        db      00H                     ;; Max packect size (MSB)  包大小为 0x40
        db      10H                     ;; Polling interval

;; Endpoint Descriptor
        db      DSCR_ENDPNT_LEN         ;; Descriptor length
        db      DSCR_ENDPNT             ;; Descriptor type
```

```
        db      86H                     ;; Endpoint number, and direction
        db      ET_INT                  ;; Endpoint type
        db      40H                     ;; Maximun packet size (LSB)
        db      00H                     ;; Max packect size (MSB)
        db      10H                     ;; Polling interval

FullSpeedConfigDscrEnd:

StringDscr:;;字符串描述符

StringDscr0:    ;;描述符 0,序列号
        db      StringDscr0End-StringDscr0      ;; String descriptor length
        db      DSCR_STRING
        db      09H,04H
StringDscr0End:

StringDscr1:    ;;描述符 1，制造商名称
        db      StringDscr1End-StringDscr1      ;; String descriptor length
        db      DSCR_STRING
        db      'F',00
        db      'L',00
        db      'C',00
        db      'T',00
        db      'E',00
        db      'C',00
        db      'H',00
StringDscr1End:

StringDscr2:    ;;描述符 2，产品名称
        db      StringDscr2End-StringDscr2      ;; Descriptor length
        db      DSCR_STRING
        db      'F',00
        db      'X',00
        db      '2',00
        db      '-',00
        db      'I',00
        db      'N',00
        db      'T',00
StringDscr2End:
```

用户钩子函数源代码 intTest.c

主要修改了三个函数 void TD_Poll(void)，void TD_Init(void)，BOOL DR_VendorCmnd(void)

下面列出 TD_Poll 函数源代码:

```
void TD_Poll(void)                      // Called repeatedly while the device is idle
{
```

```
  WORD i;
  EZUSB_InitI2C();            //初始化 EZ-USB I2 控制器
  //中断 IN 传输
  if(bInEnable)//厂商请求命令 VR_B3 启动中断 IN 传输，VR_B5 结束 IN 传输
  {
     if(!(EP2468STAT & bmEP6FULL))//端点 6 非满，则进行上传数装载
     {
        for( i = 0x0000; i < 0x200; i++ )
        {
           EP6FIFOBUF[i]=(BYTE)(i&0x00ff);
        }
         EP6BCH = 0x02;
         SYNCDELAY;
         EP6BCL = 0x00;
         SYNCDELAY;
     }
      bInEnable=FALSE;
      status+=1;
      EZUSB_WriteI2C(LED_ADDR, 0x01, &(Digit[status&0x0f]));//数码管显示
      EZUSB_WaitForEEPROMWrite(LED_ADDR);
  }
  //中断 OUT 传输
    if(!(EP2468STAT & bmEP2EMPTY))//端点 2 非空则处理
     {
         EZUSB_WriteI2C(LED_ADDR, 0x01, &(Digit[EP2FIFOBUF[0]&0x0f]));
         EZUSB_WaitForEEPROMWrite(LED_ADDR);
//重新装载端点计数器，以进行新传输，因为是双缓冲，所以置两次
         EP2BCL = 0x80;
         SYNCDELAY;
         EP2BCL = 0x80;
         SYNCDELAY;
     }
}
```

下面介绍通过控制面板来调试，见图 6.2，首先点击“Download..”下载固件程序，因为固件中 PID/VID 改变，所以会提示新的 USB 设备，并要求安装驱动，选择“自动安装软件”即可，新的驱动显示设备名为"Cypress EZ-USB Sample Device"，并响应为 **USB2.0 高速模式**。

点击“Getpipes”在端点下拉列表中得到端点号，在控制和中断传输端点下拉列表中选中“1:ENDPOINT 6 IN”，设置“Length”为 512，点击“Bulk/Int”发出中断 IN 传输请求，因为此时固件还没有准备好发送数据，所以点击“Show Pending”在“Pending Operation”中可观察到请求被挂起，此时在 USB 请求中设置“Req”为 0xb3，点击“VendReq”发送定义好的厂商请求，可看到挂起的中断 IN 请求操作完成。

选中“0:ENDPOINT 2 OUT”，设置“Length”为 512，完成一次中断 OUT 请求。

说明：本例程测试为 USB2.0 高速模式，如果主机不支持 USB2.0 则不能正常测试。

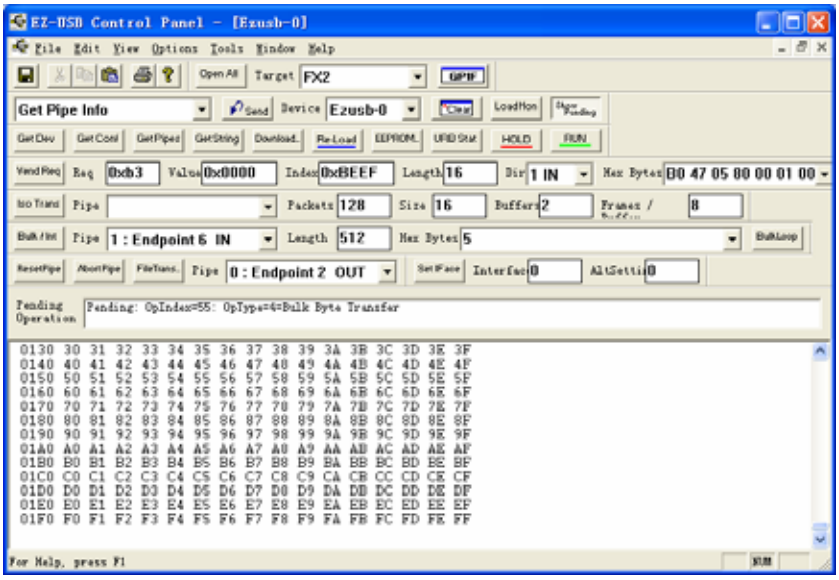


图 6.2 中断传输测试

6.3 批量传输例程

例程源文件见光盘目录 src/bulktest/bulkloop。

描述符源文件 dscr.a51，下面列出高速 USB 配置描述符。

```
HighSpeedConfigDscr:
        db    DSCR_CONFIG_LEN                ;; Descriptor length
        db    DSCR_CONFIG                   ;; Descriptor type
        db    (HighSpeedConfigDscrEnd-HighSpeedConfigDscr) mod 256
              ;; Total Length (LSB)
        db    (HighSpeedConfigDscrEnd-HighSpeedConfigDscr)  /  256
              ;; Total Length (MSB)
        db    1       ;; Number of interfaces 接口数目
        db    1       ;; Configuration number
        db    0       ;; Configuration string
        db    10000000b    ;; Attributes (b7 - buspwr, b6 - selfpwr, b5 - rwu)
        db    50       ;; Power requirement (div 2 ma)
;; Interface Descriptor
        db    DSCR_INTRFC_LEN       ;; Descriptor length
        db    DSCR_INTRFC          ;; Descriptor type
        db    0                ;; Zero-based index of this interface
        db    0                ;; Alternate setting 接口设置 0
        db    4                ;; Number of end points 端点数据
        db    0ffH             ;; Interface class
        db    00H                ;; Interface sub class
        db    00H                ;; Interface sub sub class
        db    0                ;; Interface descriptor string index
;; Endpoint Descriptor 端点 2 OUT 最大包大小 0x200 批量类型
```

```
        db    DSCR_ENDPNT_LEN       ;; Descriptor length
        db    DSCR_ENDPNT           ;; Descriptor type
        db    02H                ;; Endpoint number, and direction
        db    ET_BULK             ;; Endpoint type
        db    00H                ;; Maximun packet size (LSB)
        db    02H                ;; Max packect size (MSB)
        db    00H                ;; Polling interval
;; Endpoint Descriptor 端点 4 OUT 最大包大小 0x200 批量类型
        db    DSCR_ENDPNT_LEN       ;; Descriptor length
        db    DSCR_ENDPNT           ;; Descriptor type
        db    04H                ;; Endpoint number, and direction
        db    ET_BULK             ;; Endpoint type
        db    00H                ;; Maximun packet size (LSB)
        db    02H                ;; Max packect size (MSB)
        db    00H                ;; Polling interval
;; Endpoint Descriptor 端点 6 IN 最大包大小 0x200 批量类型
        db    DSCR_ENDPNT_LEN       ;; Descriptor length
        db    DSCR_ENDPNT           ;; Descriptor type
        db    86H                ;; Endpoint number, and direction
        db    ET_BULK             ;; Endpoint type
        db    00H                ;; Maximun packet size (LSB)
        db    02H                ;; Max packect size (MSB)
        db    00H                ;; Polling interval
;; Endpoint Descriptor 端点 8 IN 最大包大小 0x200 批量类型
        db    DSCR_ENDPNT_LEN       ;; Descriptor length
        db    DSCR_ENDPNT           ;; Descriptor type
        db    88H                ;; Endpoint number, and direction
        db    ET_BULK             ;; Endpoint type
        db    00H                ;; Maximun packet size (LSB)
        db    02H                ;; Max packect size (MSB)
        db    00H                ;; Polling interval

HighSpeedConfigDscrEnd:
```

钩子函数源代码 bulkloop.c

函数 TD_Init 中初始化端点和全局变量，代码如下

```
EP1OUTCFG = 0xA0;//端点配置，详见寄存器一章
EP1INCFG = 0xA0;
SYNCDELAY; //同步延时
EP2CFG = 0xA2;
SYNCDELAY;
EP4CFG = 0xA0;
SYNCDELAY;
EP6CFG = 0xE2;
SYNCDELAY;
```

```
  EP8CFG = 0xE0;
  // OUT 端点计数器需要预先加载来启动传输，因为默认是双缓冲所以加载两次
  SYNCDELAY;
  EP2BCL = 0x80;                    // arm EP2OUT by writing byte count w/skip.
  SYNCDELAY;
  EP2BCL = 0x80;
  SYNCDELAY;
  EP4BCL = 0x80;                    // arm EP4OUT by writing byte count w/skip.
  SYNCDELAY;
  EP4BCL = 0x80;
  //  使能自动指针特征，因为有两个 DPTR，利用它们可以很方便在两个内存间传输数据
  AUTOPTRSETUP |= 0x01;
```

TD_Poll 函数中完成主要功能，端点 2 OUT 端点在接收到数据后，通过端点 6 IN 端点原样发回主机；端点 4 OUT 数据则通过端点 8 IN 发回主机。部分代码源代码如下：

```
if(!(EP2468STAT & bmEP2EMPTY))
  { //如果端点 2 缓冲区有数据
     if(!(EP2468STAT & bmEP6FULL))//并且端点 6 缓冲区非满
     {
        APTR1H = MSB( &EP2FIFOBUF );
        APTR1L = LSB( &EP2FIFOBUF );//DPTR1 指针
        AUTOPTRH2 = MSB( &EP6FIFOBUF );
        AUTOPTRL2 = LSB( &EP6FIFOBUF );//DPTR2 指针
        count = (EP2BCH << 8) + EP2BCL;//字节计数
        // loop EP2OUT buffer data to EP6IN
        for( i = 0x0000; i < count; i++ )
        {
          EXTAUTODAT2 = EXTAUTODAT1; //通过两个 DPTR 在两个缓冲区间传输数据
        }
        EP6BCH = EP2BCH;
        SYNCDELAY;
        EP6BCL = EP2BCL;              // arm EP6IN
        SYNCDELAY;
        EP2BCL = 0x80;                //  重加载 EP2OUT 计数器
     }
  }
```

应用程序源代码见目录 src/bulktest/BtestApp，程序中采用多线程设计。测试界面见图 6.3，测试前先通过控制面板下载固件程序，点击“测试线程启动”，在“通过”会显示测试结果。

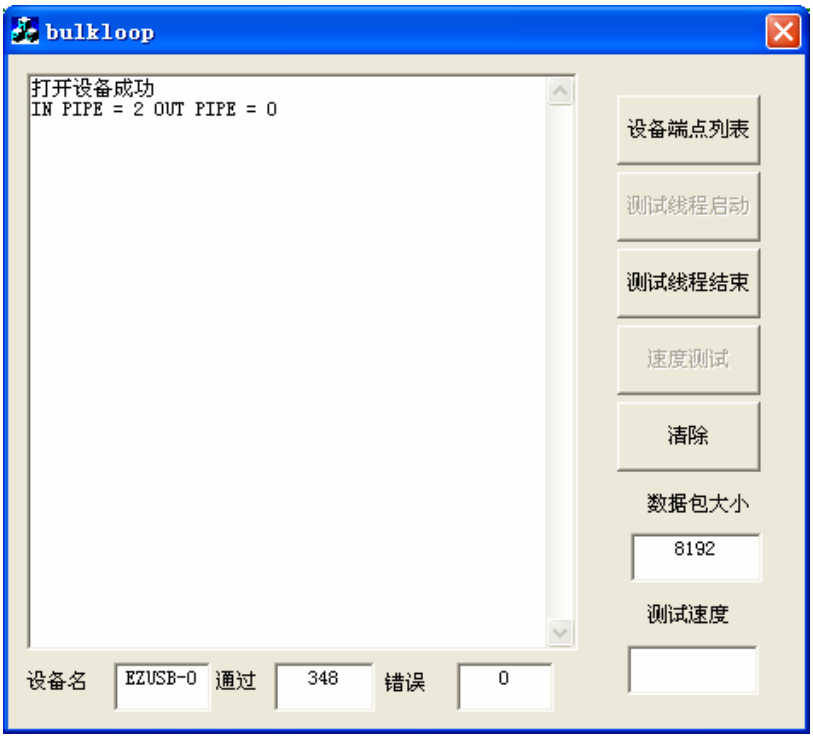


图 6.3　bulkloop 应用程序界面

6.4 USB2.0 速度测试

固件源代码和测试用例在光盘目录 src/speedtest 下。

固件主要源代码：

在 TD_Init 函数中初始化端点设置，只有端点 2 OUT 有效，批量传输类型，并设为多缓冲

```
SYNCDELAY;
EP2CFG = 0xA0;        // EP2OUT, bulk, size 512, 4 缓冲
SYNCDELAY;
EP4CFG = 0x00;        // EP4 not valid
SYNCDELAY;
EP6CFG = 0x00;        // EP6 not valid
SYNCDELAY;
EP8CFG = 0x00;        // EP8 not valid
```

在 TD_Poll 函数功能读取数据和端点缓冲区计数器装载。

```
if(!(EP2468STAT & bmEP2EMPTY))//ENDPOINT2 非空，则开始接收数据
  {
/*************************数据读取************************
        APTR1H = MSB( &EP2FIFOBUF );
        APTR1L = LSB( &EP2FIFOBUF );
        count = (EP2BCH << 8) + EP2BCL;
        //  循环将端点 2 中的数据取出，因为单片机指令慢，会影响测试速度
         for( i = 0x0000; i < count; i++ )
        {
```

```
            //利用 DPTR1 传输数据
            temp = EXTAUTODAT1;

        }
**********************************************************/
        //根据端点 2 缓冲区数目多少来确定装载端点计数器的次数，这里装载 4 次
        SYNCDELAY;
        EP2BCL = 0x80;          // re(arm) EP2OUT
        SYNCDELAY;
        EP2BCL = 0x80;          // re(arm) EP2OUT
        SYNCDELAY;
        EP2BCL = 0x80;          // re(arm) EP2OUT
        SYNCDELAY;
        EP2BCL = 0x80;          // re(arm) EP2OUT

  }
```

应用程序测试程序主要代码，详细代码见目录 speedtest 下 testApp。

```
mstep=GetTickCount(); //起始时间
for( i=0;i<nTestCount;i++)//多次循环测试
{
status =   DeviceIoControl (hOutDevice,
                            IOCTL_EZUSB_BULK_WRITE,
                            (PVOID)&outBulkControl,
                             sizeof(BULK_TRANSFER_CONTROL),
                             outBuffer,//输出缓冲区
                             m_nSize,//发送字节数
                             &BytesReturned,//返回字节数据
                             //这里为了测试速度，没有检验返回字节数
                              NULL);
}
mlength=GetTickCount();//结束时间
```

然后将所传输的字节数除以所用时间可得传输速度。通过面板下载固件程序，然后打开应用程序界面，见图 6.4，测试中速度最高峰可达 29M 字节每秒，测试速度主要是要修改发送字节数，本实验中采用 60000 为较佳值。

分为两种情况下进行测试，一种需要读取缓冲区中数据，一种只是进行缓冲区计数器装载，前者因为装载数据的单片机指令时间较长，降低了传输速度，而后者则传输速度大大提高，实际中批量端点可利用 GPIF 技术进行传输，具体参考 FX2 数据手册。

说明：

1. 在 FX2 数据很少用到等时传输，是因为在 GPIF 批量传输速度方式中可达到很高的传输速度，而等时传输中需要 1ms 发生一次 SOF 中断，而一次中断加载的数据有限，并且考虑到数据装载的单片机指令较长，大大降低了传输速度。
2. 本例程测试为 USB2.0 高速模式，如果主机不支持 USB2.0 则不能正常测试。

6.5 IO 测试例程

通过本试验测试 FX2 芯片 IO 端口的简单使用，下载程序后通过示波器可观察到端口 A、B、

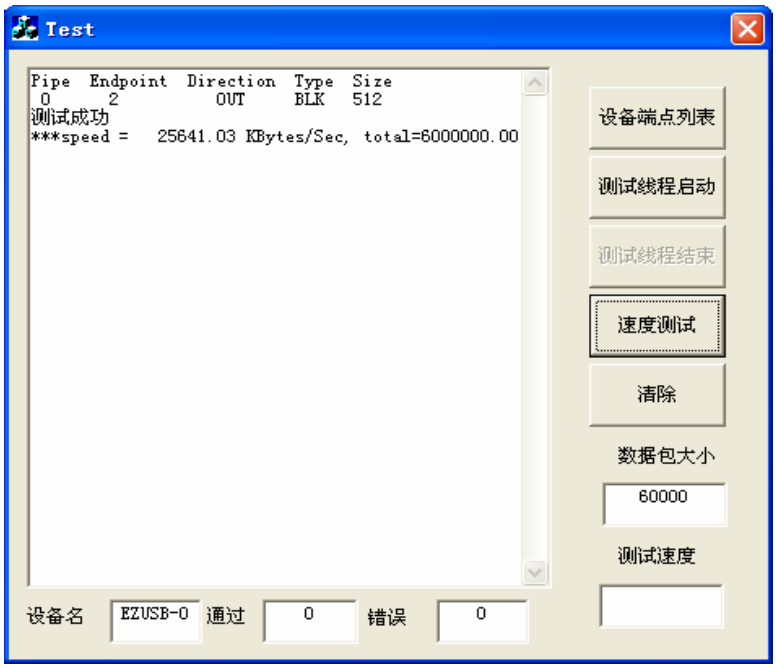


图 6.4 测试速度

D 方波波形，主要程序源代码如下

函数 TD_Init 部分代码：

```
OEA = 0xff;      //A 端口输出使能
IOA = 0x00;      //输出低电平
OEB = 0xff;      //B 端口输出使能
IOB = 0x00;      //输出低电平
OED = 0xff;      //D 端口输出使能
IOD = 0x00;      //输出低电平
```

函数 TD_Poll 部分代码：

```
IOA^=0xff;
IOB^=0xff;
IOD^=0xff;
EZUSB_Delay (1); //在一段时间后输出波形翻转
```

第七章 产品发布

产品发布所有源代码在光盘目录 src\bulktest 下。

7.1 下载固件驱动程

（1）转化 HEX 文件为 C 文件

最终产品包括两个驱动程序，一个驱动是用来下载固件。当固件下载完成后，模拟一次断开重新连接，此时下载的固件来响应 USB 枚举，并加载 USB 设备通用驱动程序。

首先将 HEX 文件转化为 C 文件，在 Windows DOS 模式下执行
Hex2C bulkloop.hex bulkloop.c
Hex2C 工具在 安装路径\Cypress\USB\Bin 下面。
生成的 bulkloop.c 包括一个 Intel 数组，称为 fireware。

（2）驱动程序编译

通过构造好的驱动 VC 编译环境，源代码在 src\bulktest\ezloader。用户只需要将 ezloader 项目文件 fireware.c 中的 fireware[]数组用 bulkloop.c 中的数组代替。

最后通过上图的 DDK 进行编译。如果 VC 中没有出现上图的 DDK 编译工具，则需要安装光盘下的 Compuware SoftICE Driver Suite 2.7。

编译后将在 ezloader\lib\i386 目录下生成 ezloader.sys。

7.2 编写 INF 文件

这里给出了 INF 文件的格式，用户需要的只是改动黑体部分，根据自己 EEPROM 所写入的 PID/VID。

```
[Version]
Signature="$CHICAGO$"
Class=USB
provider=%Cypress%
LayoutFile=layout.inf

[Manufacturer]
%Cypress%=Cypress

[Cypress]
;
; This is the VID/PID for the EZ-USB development board.  This device
```

```
; is bound to a version of the general purpose driver that will
; automatically download the Keil 8051 monitor to external RAM.
; Do not use this VID/PID for your own device or the monitor will
; wipe out your firmware.
;
%USB\VID_04B4&PID_1009.DeviceDesc%=EZUSBDEV.Dev, USB\VID_04B4&PID_1009
;
; This VID/PID is used by several of the EZ-USB development kit
; samples.    This device is bound to the general purpose driver.
;
%USB\VID_04B4&PID_1002.DeviceDesc%=EZUSB.Dev, USB\VID_04B4&PID_1002

[PreCopySection]
HKR,,NoSetupUI,,1

[DestinationDirs]
EZUSB.Files.Ext = 10,System32\Drivers
EZUSB.Files.Inf = 10,INF
EZUSBDEV.Files.Ext = 10,System32\Drivers
EZUSBDEV.Files.Inf = 10,INF

[EZUSB.Dev]
CopyFiles=EZUSB.Files.Ext, EZUSB.Files.Inf
AddReg=EZUSB.AddReg

[EZUSB.Dev.NT]
; copyfiles commented out for Win2K to avoid user intervention during install
; CopyFiles=EZUSB.Files.Ext, EZUSB.Files.Inf
AddReg=EZUSB.AddReg

[EZUSB.Dev.NT.Services]
Addservice = EZUSB, 0x00000002, EZUSB.AddService

[EZUSB.AddService]
DisplayName      = %EZUSB.SvcDesc%
ServiceType      = 1                  ; SERVICE_KERNEL_DRIVER
StartType        = 2                  ; SERVICE_AUTO_START
ErrorControl     = 1                  ; SERVICE_ERROR_NORMAL
ServiceBinary    = %10%\System32\Drivers\ezusb.sys
LoadOrderGroup = Base
[EZUSB.AddReg]
HKR,,DevLoader,,*ntkern
HKR,,NTMPDriver,,ezusb.sys
```

```
[EZUSB.Files.Ext]
ezusb.sys
[EZUSB.Files.Inf]
dj.Inf
[EZUSBDEV.Dev]
CopyFiles=EZUSBDEV.Files.Ext, EZUSBDEV.Files.Inf
AddReg=EZUSBDEV.AddReg
[EZUSBDEV.Dev.NT]
; copyfiles commented out for Win2K to avoid user intervention during install
; CopyFiles=EZUSBDEV.Files.Ext, EZUSBDEV.Files.Inf
AddReg=EZUSBDEV.AddReg
[EZUSBDEV.Dev.NT.Services]
Addservice = EZUSBDEV, 0x00000002, EZUSBDEV.AddService
[EZUSBDEV.AddService]
DisplayName      = %EZUSBDEV.SvcDesc%
ServiceType      = 1                  ; SERVICE_KERNEL_DRIVER
StartType        = 2                  ; SERVICE_AUTO_START
ErrorControl     = 1                  ; SERVICE_ERROR_NORMAL
ServiceBinary    = %10%\System32\Drivers\dj.sys
LoadOrderGroup = Base
[EZUSBDEV.AddReg]
HKR,,DevLoader,,*ntkern
HKR,,NTMPDriver,,dj.sys
[EZUSBDEV.Files.Ext]
dj.sys
[EZUSBDEV.Files.Inf]
dj.Inf
;-----------------------------------------------------------------;
[Strings]
Cypress="Cypress Semiconductor"
USB\VID_04B4&PID_1009.DeviceDesc="固件下载驱动程序"
USB\VID_04B4&PID_1002.DeviceDesc="EZUSB 通用驱动程序"
EZUSB.SvcDesc="Cypress General Purpose USB Driver (ezusb.sys)"
EZUSBDEV.SvcDesc="Cypress General Purpose USB Driver w/ Keil Monitor (dj.sys)"
```

上面 INF 文件中包括两个安装文件，一个是固件下载驱动程序（VID_04B4&PID_1009），一个是通用驱动程序（VID_04B4&PID_1002）。固件下载 PID/VID 用户可自己定制。

7.3 安装

根据上面下载固件驱动的 PID/VID 对开发板上的 EEPROM 编程。

安装文件包括.INF 文件，两个驱动文件 dj.sys 和 ezusb.sys，前面的例子中将固件下载驱动 ezloader.sys 改名为 dj.sys。

注意：如果安装过程中，设备无法正常安装，则用户可以建立一个批处理文件将.sys 和.inf

文件拷贝到相应目录下。

提示：在实际开发中，考虑到建立环境困难，用户可直接在本例程基础上进行修改。

第八章 CY7C68013寄存器

寄存器定义在 FX2 寄存器固件程序头文件 fx2reg.h 中，本章介绍编程中常用寄存器。

8.1 系统配置寄存器

8.1.1 CPU 控制与状态寄存器

✧ CPUCS，CPU 控制与状态寄存器

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**PORTCSTB**	**CLKSPD1**	**CLKSPD0**	**CLKINV**	**CLKOE**	**0**
R	R	R/W	R/W	R/W	R/W	R/W	R
0	0	0	0	0	0	1	0

PORTCSTB －1 表示读写端口 C 时产生 RD# 和 WR#信号，0 表示不产生读写信号

CLKSPD1，CLKSPD0－CPU 时钟设置，设置见表 1。

表 8.1 时钟设置

CLKSPD1	CLKSPD0	CPU Clock
0	0	12 MHz (Default)
0	1	24 MHz
1	0	48 MHz
1	1	Reserved

CLKINV－时钟状态反转

CLKOE－时钟使能

8051RES－复位 CPU

8.1.2 接口配置寄存器（IO，GPIF 和 slave FIFO 设置）

✧ IFCONFIG，配置寄存器

b7	b6	b5	b4	b3	b2	b1	b0
IFCLKSRC	**3048MHZ**	**IFCLKOE**	**IFCLKPOL**	**ASYNC**	**GSTATE**	**IFCFG1**	**IFCFG0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	0	0	0	0	0	0

IFCLKSRC—0外部时钟源，1内部时钟源

3048MHZ－0 IFCLK时钟30M，1 IFCLK时钟48M

IFCLKOE－IFCLK时钟输出使能，0关闭，1打开

IFCLKPOL－IFCLK 信号反转，0 不反转，1 反转

ASYNC－GPIF 同步或异步操作，0 同步，1 异步

GSTATE－GPIF 状态输出使能，0 关闭，1 使能，引脚 PE0 PE1 PE2 和 GPIF 状态 GSTATE0, GSTATE1, GSTATE2。

IFCFG0,IFCFG1－模式设置，模式决定了多个引脚的状态，详见表 8.2。

表 8.2 模式设置

IFCFG1	IFCFG0	Configuration
0	0	普通端口模式
0	1	保留
1	0	GPIF（GPIF Interface）
1	1	Slave FIFO 模式

8.1.2 Slave FIFO 模式 FLAGA/B/C/D 引脚配置寄存器

✧ PINFLAGSAB，FIFO FLAGA 和 FLAGB 配置寄存器

b7	b6	b5	b4	b3	b2	b1	b0
FLAGB3	**FLAGB2**	**FLAGB1**	**FLAGB0**	**FLAGA3**	**FLAGA2**	**FLAGA1**	**FLAGA0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

✧ PINFLAGSCD，FIFO FLAGC 和 FLAGD 配置寄存器

b7	b6	b5	b4	b3	b2	b1	b0
FLAGD3	**FLAGD2**	**FLAGD1**	**FLAGD0**	**FLAGC3**	**FLAGC2**	**FLAGC1**	**FLAGC0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	1	0	0	0	0	0	0

Slave Fifo 模式中，用引脚 FLAGA～FLAGD 来定义用端点 FIFO 的状态，并可灵活编程来实现 FLAGx 设置，见表 8.3。

表 8.3 FLAGA~PLAGD 引脚功能设置

FLAGx3	FLAGx2	FLAGx1	FLAGx0	引脚功能
0	0	0	0	FLAGA=PF, FLAGB=FF, FLAGC=EF,FLAGD=EP2PF (除FLAGD之外，FLAGA ~FLAGC对应的端点FIFO由引脚FIFOADR[0,1]来确定，详见表8.4)
0	0	0	1	保留
0	0	1	0	
0	0	1	1	
0	1	0	0	EP2PF
0	1	0	1	EP4PF
0	1	1	0	EP6PF
0	1	1	1	EP8PF
1	0	0	0	EP2EF
1	0	0	1	EP4EF
1	0	1	0	EP6EF
1	0	1	1	EP8EF

1	1	0	0	EP2FF
1	1	0	1	EP4FF
1	1	1	0	EP6FF
1	1	1	1	EP8FF

说明：PF表示FIFO可编程，EF表示FIFO已空，FF表示FIFO已满

表 8.4　端点选择

FIFOADR1 pin	FIFOADR0 pin	选择端点
0	0	EP2
0	1	EP4
1	0	EP6
1	1	EP8

8.1.3 端点缓冲区复位寄存器

✧ FIFORESET

b7	b6	b5	b4	b3	b2	b1	b0
NAKALL	**0**	**0**	**0**	**EP3**	**EP2**	**EP1**	**EP0**
W	W	W	W	W	W	W	W
x	x	x	x	x	x	x	x

NAKALL－0 关闭 NAK 功能，1 用 NAK 响应主控器请求，例如在复位端点 FIFO 时，为了保证复位正常，防止主控器请求的干扰，先写入 0x80，然后复位端点，最后写入 0x00，使能响应请求。

EP3~EP0，1 则复位对应的端点缓冲区，其中 EP3~EP0 分别对应端点 EP8，EP6，EP4，EP2。

8.1.4 仿真断点寄存器

✧ BREAKPT，断点控制寄存器

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**BREAK**	**BPPULSE**	**BPEN**	**0**
R	R	R	R	R/W	R/W	R/W	R
0	0	0	0	0	0	0	0

BREAK－0 非使能断点功能，1 使能断点功能，当程序计数器和 BPADDR 相同时置位

BPPLUSE－0 如果程序运行端点处，则 BKPT 引脚保持高，直到被固件清除，1 则 BPPT 保持高电平 8 个 CPU 时钟后自动置低

BPEN－断点使能

✧ BPADDRH 断点高地址

b7	b6	b5	b4	b3	b2	b1	b0
A15	**A14**	**A13**	**A12**	**A11**	**A10**	**A9**	**A8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

✧ BPADDRL 断点低地址

b7	b6	b5	b4	b3	b2	b1	b0
A7	**A6**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

8.1.5 串口波特率设置寄存器

✧ UART230

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**0**	**230UART1**	**230UART0**
R	R	R	R	R	R	R/W	R/W
0	0	0	0	0	0	0	0

230UART1/230UART0－1 设置串口 1/串口 0 波特率为 230K，和 CPU 时钟频率无关。

8.1.6 slave FIFO 模式引脚电平设置寄存器

✧ FIFOPINPOLAR

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**PKTEND**	**SLOE**	**SLRD**	**SLWR**	**EF**	**FF**
R	R	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

用来设置 slave FIFO 模式各引脚电平，0 置低，1 置高。

8.1.7 芯片版本号

✧ REVID

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**PKTEND**	**SLOE**	**SLRD**	**SLWR**	**EF**	**FF**
R	R	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

芯片版本为只读

8.1.8 .芯片版本控制寄存器

✧ REVCTL

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**0**	**DYN_OUT**	**ENH_PKT**
R	R	R	R	R	R	R/W	R/W
0	0	0	0	0	0	0	0

8.1.9 GPIF 模式数据保持时间

✧ GPIFHOLDTIME

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**0**	**HOLDTIME[1:0]**	
R	R	R	R	R	R	RW	RW
0	0	0	0	0	0	0	0

HOLDTIME[1,0]－保持时间
00=0 个 IFCLK 周期
01=0.5 个 IFCLK 周期
10=1 个 IFCLK 周期
11=保留

8.2 端点配置寄存器

8.2.1 端点 1IN 和 1OUT 配置

✧ EP1OUTCFG，端点 1OUT 配置

B7	b6	b5	b4	b3	b2	b1	b0
VALID	**0**	**TYPE1**	**TYPE0**	**0**	**0**	**0**	**0**
R/W	R	R/W	R/W	R	R	R	R
1	0	1	0	0	0	0	0

✧ EP1INCFG，端点 1IN 配置

b7	b6	b5	b4	b3	b2	b1	b0
VALID	0	**TYPE1**	**TYPE0**	0	0	0	0

Valid－1 端点有效，0 端点无效
TYPE1,TYPE0—端点类型见表 8.5

表 8.5 端点 1IN 和 1OUT 类型设置

TYPE1	TYPE0	类型
0	0	无效
0	1	无效
1	0	批量（默认）
1	1	中断

8.2.2 端点 2,4,6,8 配置

✧ EP2CFG，端点 2 配置

b7	b6	b5	b4	b3	b2	b1	b0
VALID	**DIR**	**TYPE1**	**TYPE0**	**SIZE**	**0**	**BUF1**	**BUF0**
R/W	R/W	R/W	R/W	R/W	R	R/W	R/W
1	0	1	0	0	0	1	0

✧ EP4CFG，端点 4 配置

b7	b6	b5	b4	b3	b2	b1	b0

VALID	DIR	TYPE1	TYPE0	0	0	0	0
R/W	R/W	R/W	R/W	R	R	R	R
1	0	1	0	0	0	0	0

✧ EP6CFG，端点 6 配置

b7	b6	b5	b4	b3	b2	b1	b0
VALID	**0**	**TYPE1**	**TYPE0**	**0**	**0**	**0**	**0**
R/W	R	R/W	R/W	R	R	R	R
1	0	1	0	0	0	0	0

✧ EP8CFG，端点 8 配置

7	b6	b5	b4	b3	b2	b1	b0
VALID	**DIR**	**TYPE1**	**TYPE0**	**0**	**0**	**0**	**0**
R/W	R/W	R/W	R/W	R	R	R	R
1	1	1	0	0	0	0	0

VALID－0 端点无效，1 端点有效

DIR－端点方向，0=OUT 方向，1=IN 方向，默认端点 2,4 为 IN，端点 6,8 为 OUT

TYPE1，TYPE0－端点类型，见表 8.6

表 8.6 端点 2,4,6,8 类型

TYPE1	TYPE0	类型
0	0	无效
0	1	等时
1	0	批量（默认）
1	1	中断

SIZE－缓冲区大小（仅端点 2 和端点 6），0=512 字节，1=1024 字节

BUF1，BUF0－端点缓冲区个数（仅端点 2 和端点 6），见表 8.7。

表 8.7 端点 2 和 6 缓冲区个数

BUF1	BUF0	类型
0	0	四缓冲
0	1	无效
1	0	双缓冲（默认）
1	1	三缓冲

8.2.3 slave FIFO 模式端点 2,4,6,8 配置

✧ EP2FIFOCFG，EP4FIFOCFG，EP6FIFOCFG，EP8FIFOCFG

b7	b6	b5	b4	b3	b2	b1	b0
0	**INFM1**	**OEP1**	**AUTOOUT**	**AUTOIN**	**ZEROLENIN**	**0**	**WORDWIDE**
R	R/W	R/W	R/W	R/W	R/W	R	R/W
0	0	0	0	0	1	0	1

INFM1—IN端点满减1

OEP1－OUT端点空加1
AUTOOUT－实时连接到OUT端点缓冲区
AUTOIN－IN端点数据自动呈交SIE
ZEROLENIN－使能零长度IN端点数据包
WORDWIDE－数据宽度，8位或16位

8.2.4 端点 2,4,6,8 AUTOIN 长度，仅对 IN 类型端点

✧ EP2AUTOINLENH,EP4AUTOINLENH,EP6AUTOINLENH,EP8AUTOINLENH，包长度高字节

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**PL10**	**PL9**	**PL8**
R	R	R	R	R	R/W	R/W	R/W
0	0	0	0	0	0	1	0

✧ EP2AUTOINLENL,EP4AUTOINLENL,EP6AUTOINLENL,EP8AUTOINLENL，包长度低字节

b7	b6	b5	b4	b3	b2	b1	b0
PL7	**PL6**	**PL5**	**PL4**	**PL3**	**PL2**	**PL1**	**PL0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

PL10 仅端点 2 和 6 有效

8.2.5 slave FIFO 模式 Programmable-Level FLAGx 触发电平

✧ EP2FIFOPFH _AT_ 0xE630; // EP2 Programmable Flag trigger H
✧ EP2FIFOPFL _AT_ 0xE631; // EP2 Programmable Flag trigger L
✧ EP4FIFOPFH _AT_ 0xE632; // EP4 Programmable Flag trigger H
✧ EP4FIFOPFL _AT_ 0xE633; // EP4 Programmable Flag trigger L
✧ EP6FIFOPFH _AT_ 0xE634; // EP6 Programmable Flag trigger H
✧ EP6FIFOPFL _AT_ 0xE635; // EP6 Programmable Flag trigger L
✧ EP8FIFOPFH _AT_ 0xE636; // EP8 Programmable Flag trigger H
✧ EP8FIFOPFL _AT_ 0xE637; // EP8 Programmable Flag trigger L

8.2.6 端点 2,4,6,8 等时 IN 端点传输每数据帧包数目

✧ EP2ISOINPKTS，EP4ISOINPKTS，EP6ISOINPKTS， EP8ISOINPKTS

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**0**	**INPPF1**	**INPPF0**
R	R	R	R	R	R	R/W	R/W
0	0	0	0	0	0	0	1

INPF1,INPF0－见表 8.8

表 8.8　INPF1，INPF0 配置

INPPF1	INPPF0	Packets
0	0	Invalid
0	1	1
1	0	2
1	1	3

8.2.7 强行使 IN 传输结束

✧ INPKTEND

b7	b6	b5	b4	b3	b2	b1	b0
SKIP	**0**	**0**	**0**	**EP3**	**EP2**	**EP1**	**EP0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

SKIP－当 ENH_PKT（REVCTL 寄存器 bit0）为 1 时，0 表示自动“分配”一个 IN 缓冲区，1 表示将跳过一个 IN 缓冲区

EP3,EP2,EP1,EP0－代替 PKTEND 引脚功能，强行结束 IN 类型端点 8,6,4,2 IN 数据传输。

8.2.8 强行使 OUT 传输结束

✧ OUTPKTEND

b7	b6	b5	b4	b3	b2	b1	b0
SKIP	**0**	**0**	**0**	**EP3**	**EP2**	**EP1**	**EP0**
W	W	W	W	W	W	W	W
x	x	x	x	x	x	x	x

SKIP－当 ENH_PKT（REVCTL 寄存器 bit0）为 1 时，0 自动“分配”一个 OUT 缓冲区，1 将跳过一个 OUT 缓冲区

EP3,EP2,EP1,EP0－代替 EPxBLH.7=1 引脚功能，强行结束 OUT 类型端点 8,6,4,2 数据传输。

8.3 中断寄存器

8.3.1 slave FIFO 模式端点 FIFO 中断使能/请求（INT4）

✧ EP2FIFOIE, EP4FIFOIE, EP6FIFOIE, EP8FIFOIE

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**EDGEPF**	**PF**	**EF**	**FF**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

EDGPF－PF 中断触发沿，0 上升沿触发，1 下降沿触发

PF－1 使能端点 FIFO PF 中断，0 非使能

EF－1 使能端点 FIFO EF 中断，0 非使能

FF－1 使能端点 FIFO FF 中断，0 非使能

✧ EP2FIFOIRQ，EP4FIFOIRQ，EP6FIFOIRQ，EP8FIFOIRQ

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**PF**	**EF**	**FF**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

PF－0 无 PF 中断，1 有 PF 中断

PF－0 无 EF 中断，1 有 EF 中断

PF－0 无 FF 中断，1 有 FF 中断

说明：写 1 可清除中断请求位。

8.3.2 IN-BULK-NAK 中断使能/请求（INT2）

✧ IBN 中断使能

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**EP8**	**EP6**	**EP4**	**EP2**	**EP1**	**EP0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

✧ IBNIRQ 中断请求

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**EP8**	**EP6**	**EP4**	**EP2**	**EP1**	**EP0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

主控器向 USB IN 端点发传输请求，而数据还没有准备好，则 USB SIE 将自动产生 NAK 给主控器，同时产生 IBN 中断。

8.3.3 端点 PING-NAK/IBN 中断使能/请求（INT2）

✧ NAKIE 中断使能

b7	b6	b5	b4	b3	b2	b1	b0
EP8	**EP6**	**EP4**	**EP2**	**EP1**	**EP0**	**0**	**IBN**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

✧ NAKIRQ 中断请求

b7	b6	b5	b4	b3	b2	b1	b0
EP8	**EP6**	**EP4**	**EP2**	**EP1**	**EP0**	**0**	**IBN**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

USB2.0 中，对于 BULK 传输为了增加带宽，增加了 PING-NAK 请求，在 BULK OUT 传输前，发 PING 请求，询问 OUT 端点准备好接收数据，如没准备好，发 NAK 响应 PING 请求，并产生 NAK 中断，用户可在该中断中复位 OUT 缓冲区，准备接收数据。

IBN－IBN中断使能/请求，作用与IBNIE,IBNIRQ相同，只不过是针对于所有IN端点。

8.3.4 USB 中断使能/请求（INT2）

✧ USBIE 中断使能

b7	b6	b5	b4	b3	b2	b1	b0
0	**EP0ACK**	**HSGRANT**	**URES**	**SUSP**	**SUTOK**	**SOF**	**SUDAV**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

✧ USBIRQ 中断请求

b7	b6	b5	b4	b3	b2	b1	b0
0	**EP0ACK**	**HSGRANT**	**URES**	**SUSP**	**SUTOK**	**SOF**	**SUDAV**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

EP0ACK－端点0 ACK中断
HSGRANT－设备为USB2.0中断
URES－USB总线复位中断
SUSP－USB总线挂起中断
SUTOK－SUTOK中断
SOF－数据帧中断
SUDAV－SUDAV中断

8.3.5 端点中断使能/请求（INT2）

✧ EPIE

b7	b6	b5	b4	b3	b2	b1	b0
EP8	**EP6**	**EP4**	**EP2**	**EP1OUT**	**EP1IN**	**EP0OUT**	**EP0IN**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

✧ EPIRQ

b7	b6	b5	b4	b3	b2	b1	b0
EP8	**EP6**	**EP4**	**EP2**	**EP1OUT**	**EP1IN**	**EP0OUT**	**EP0IN**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

对于IN端点，如果IN缓冲区装载好数据产生中断请求
对于OUT端点，如果OUT缓冲区接收到主机发来的数据产生中断请求

8.3.6 GPIF 模式中断使能/请求(INT4)

✧ GPIFIE

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**0**	**GPIFWF**	**GPIFDONE**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

0	0	0	0	0	0	0	0

✧ GPIFIRQ

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**0**	**GPIFWF**	**GPIFDONE**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

GPIFWF－当 FIFO 读/写波形产生中断请求

GPIFDONE－GPIF 处于 IDLE 状态中断

8.3.7 USB 错误中断使能/请求（INT2）

✧ USBERRIE

b7	b6	b5	b4	b3	b2	b1	b0
ISOEP8	**ISOEP6**	**ISOEP4**	**ISOEP2**	**0**	**0**	**0**	**ERRLIMIT**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

✧ USBERRIRQ

b7	b6	b5	b4	b3	b2	b1	b0
ISOEP8	**ISOEP6**	**ISOEP4**	**ISOEP2**	**0**	**0**	**0**	**ERRLIMIT**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

ISOEP8,ISOEP6,ISOEP4,ISOEP2－ISO 端点 8,6,4,2 数据传输有发生错误中断

ERRLIMIT－错误数目超过极限中断

8.3.8 USB 错误数极限

✧ ERRCNTLIM，USB 错误计数和错误极限设置

b7	b6	b5	b4	b3	b2	b1	b0
EC3	**EC2**	**EC1**	**EC0**	**LIMIT3**	**LIMIT2**	**LIMIT1**	**LIMIT0**
R	R	R	R	R/W	R/W	R/W	R/W
x	x	x	x	0	1	0	0

EC[3:0]－USB 错误计数，最大为 15

LIMIT[3:0]－USB 错误极限，最大为 15，当错误计数和极限相等，则产生 ERRLIMIT 中断

8.3.9 USB 错误计数清除

✧ CLRERRCNT

b7	b6	b5	b4	b3	b2	b1	b0
x	**x**	**x**	**x**	**x**	**x**	**x**	**x**
W	W	W	W	W	W	W	W
x	x	x	x	x	x	x	x

写入 0xFF，将 ERRCNTLIM 寄存器中 EC[3:0]清零。

8.3.10 INT2 中断矢量

✧ INT2IVEC

b7	b6	b5	b4	b3	b2	b1	b0
0	**I2V4**	**I2V3**	**I2V2**	**I2V1**	**I2V0**	**0**	**0**
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

I2V[4:0]－FX2 采用二级中断源机制，当 USB 中断(INT2)发生时，程序跳转到地址 0x43，再通过中断矢量查找具体中断，并跳转到对应中断函数。

8.3.11 INT4 中断矢量

✧ INT4IVEC，GPIF 主模式和 FIFO 从模式中断

b7	b6	b5	b4	b3	b2	b1	b0
1	**0**	**I4V3**	**I4V2**	**I4V1**	**I4V0**	**0**	**0**
R	R	R	R	R	R	R	R
1	0	0	0	0	0	0	0

I4V[3:0]－INT4 中断发生时，程序跳转到地址 0x53，再根据中断矢量，跳转到具体的中断函数开始执行。

8.3.12 INT2 和 INT4 中断设置

✧ INTSETUP

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**AV2EN**	**0**	**INT4SRC**	**AV4EN**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

AV2EN－使能 INT2 中断矢量
INT4SRC－0 INT4 中断源是引脚 INT4，1 INT4 中断源是 GPIF 主模式和 FIFO 从模式
AV4EN－使能 INT4 中断矢量

8.4 端口配置

8.4.1 端口 A 配置

✧ PORTACFG

b7	b6	b5	b4	b3	b2	b1	b0
FLAGD	**SLCS**	**0**	**0**	**0**	**0**	**INT1**	**INT0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

置 1 则配置为对应的复用引脚功能

8.4.2 端口 C 配置

✧ PORTCCFG

b7	b6	b5	b4	b3	b2	b1	b0
GPIFA7	**GPIFA6**	**GPIFA5**	**GPIFA4**	**GPIFA3**	**GPIFA2**	**GPIFA1**	**GPIFA0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

8.4.3 端口 E 配置

✧ PORTECFG

b7	b6	b5	b4	b3	b2	b1	b0
GPIFA8	**T2EX**	**INT6**	**RXD1OUT**	**RXD0OUT**	**T2OUT**	**T1OUT**	**T0OUT**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

8.4.4 I2C 寄存器

✧ I2CS，控制与状态寄存器

b7	b6	b5	b4	b3	b2	b1	b0
START	**STOP**	**LASTRD**	**ID1**	**ID0**	**BERR**	**ACK**	**DONE**
R/W	R/W	R/W	R	R	R	R	R
0	0	0	x	x	0	0	0

✧ I2DAT，数据寄存器

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

✧ I2CTL，控制寄存器

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**0**	**STOPIE**	**400KHZ**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

STOPIE－使能 STOP 中断

400KHZ－0 则 I2C 时钟频率大概 100KHZ，1 则 I2C 时钟频率大概 400KHZ

8.4.5 数据指针寄存器

✧ XAUTODAT1

✧ XAUTODAT2

b7	b6	b5	b4	b3	b2	b1	b0

D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

FX2 有两个 DPTR，通过它们，可以很方便实现内存（XDATA）间数据传输。

8.5 USB 控制寄存器

8.5.1 USB 控制与状态寄存器

✧ USBCS

b7	b6	b5	b4	b3	b2	b1	b0
HSM	**0**	**0**	**0**	**DISCON**	**NOSYNSOF**	**RENUM**	**SIGRSUME**
R	R	R	R	R/W	R/W	R/W	R/W
x	0	0	0	0	0	0	0

HSM－如果支持USB2.0，则HSM=1，并且会产生HSGRANT（发生高速USB设备）中断
DISCON－模拟 USB 断开，对于启动 E2PROM 则会置位 DISCON
NOSYNSOF－1 非使能同步丢失数据帧
RENUM－重枚举，0 则由 USB 核处理所有的设备请求，1 则由固件处理出 SET_ADDRESS 以外所有的设备请求。在“C2 加载”E2PROM 完成时，会置位 RENUM
SIGRSUME－1 使 USB 总线产生“K”信号

8.5.2 USB 总线挂起

✧ SUSPEND

b7	b6	b5	B4	b3	b2	b1	b0
x	**x**	**x**	**x**	**x**	**x**	**x**	**x**
W	W	W	W	W	W	W	W
x	x	x	x	x	x	x	x

写入0xff，使USB进入挂起状态。

8.5.3 USB 总线唤醒控制与状态

✧ WAKEUPCS

b7	b6	b5	b4	b3	b2	b1	b0
WU2	**WU**	**WU2POL**	**WUPOL**	**0**	**DPEN**	**WU2EN**	**WUEN**
R/W	R/W	R/W	R/W	R	R/W	R/W	R/W
x	x	0	0	0	1	0	1

WU2－从引脚 WU2 初始化唤醒，写 1 清除
WU－从引脚 WU 初始化唤醒，写 1 清楚
WU2POL－WU2 输入引脚极性，0 低有效，1 高有效
WUPOL－WU 收入引脚极性，0 低有效，1 高有效
DPEN－D+唤醒总线使能，0 非使能，1 使能

WU2EN－WU2 唤醒使能，0 非使能，1 使能
WUEN－WU 唤醒使能，0 非使能，1 使能

8.5.4 USB 数据 toggle 控制

✧ TOGCTL

b7	b6	b5	b4	b3	b2	b1	b0
Q	**S**	**R**	**IO**	**EP3**	**EP2**	**EP1**	**EP0**
R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

8.5.5 USB 数据帧计数

✧ USBFRAMEH，USB 数据帧计数高字节

b7	b6	b5	B4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**FC10**	**FC9**	**FC8**
R	R	R	R	R	R	R	R
0	0	0	0	0	x	x	x

USB 数据帧计数最大值为 1024。

✧ USBFRAMEL，USB 数据帧计数低字节

b7	b6	b5	b4	b3	b2	b1	b0
FC7	**FC6**	**FC5**	**FC4**	**FC3**	**FC2**	**FC1**	**FC0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

✧ MICROFRAME，USB 微帧计数

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**MF2**	**MF1**	**MF0**
R	R	R	R	R	R	R	R
0	0	0	0	0	x	x	x

微帧计数，范围为 0 到 7。

✧ FNADDR，USB 唯一功能地址，只读

b7	b6	b5	b4	b3	b2	b1	b0
0	**FA6**	**FA5**	**FA4**	**FA3**	**FA2**	**FA1**	**FA0**
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

8.6 端点寄存器

8.6.1 端点 0 数据字节计数

✧ EP0BCH，EP0 计数高字节

B7	b6	b5	b4	b3	b2	b1	b0
(BC15)	**(BC14)**	**(BC13)**	**(BC12)**	**(BC11)**	**(BC10)**	**(BC9)**	**(BC8)**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

✧ EP0BCL，EP0 计数低字节

B7	b6	B5	b4	b3	b2	b1	b0
(BC7)	**BC6**	**BC5**	**BC4**	**BC3**	**BC2**	**BC1**	**BC0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

虽然端点 0 缓冲区只有 64 字节，但可通过 SUDPTR 来扩展

8.6.2 端点 1OUT 和 1IN 计数

✧ EP1OUTBC，端点 EP1OUT 计数

✧ EP1INBC，端点 EP1IN 计数

b7	b6	b5	b4	b3	b2	b1	b0
0	**BC6**	**BC5**	**BC4**	**BC3**	**BC2**	**BC1**	**BC0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	x	x	x	x	x	x	x

8.6.3 端点 2 和 6 计数高字节

✧ EP2BCH 和 EP6BCH

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**BC10**	**BC9**	**BC8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	x	x	x

端点2和6缓冲区可以配置为512或1024字节。

8.6.4 端点 4 和 8 计数高字节

✧ EP4BCH 和 EP8BCH

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**0**	**BC9**	**BC8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	x	x

端点 4 和 8 缓冲区只能配置 512 字节，所以计数最高只有 9 比特。

8.6.5 端点 2，4，6 和 8 计数低字节

✧ EP2BCL,EP4BCL,EP6BCL,EP8BCL

b7	b6	b5	b4	b3	b2	b1	b0
BC7	**BC6**	**BC5**	**BC4**	**BC3**	**BC2**	**BC1**	**BC0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

8.6.6 端点 0 控制与状态寄存器

✧ EP0CS

b7	b6	b5	b4	b3	b2	b1	b0
HSNAK	**0**	**0**	**0**	**0**	**0**	**BUSY**	**STALL**
R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
1	0	0	0	0	0	0	0

HSNAK－在控制传输状态阶段，如果缓冲数据为空，而设备可通过ACK，NAK或STALL来响应，置为1则设备用NAK响应。

BUSY－端点 0 忙，只读

STALL－端点 1 停止，如果端点停止，则端点请求将被挂起，写 1 使端点 0 停止，写 0 清除端点停止

8.6.7 端点 1OUT 和 IN 控制与状态寄存器

✧ EP1OUTCS，EP1INCS

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**0**	**BUSY**	**STALL**
R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
0	0	0	0	0	0	0	0

8.6.8 端点 2 控制与状态寄存器

✧ EP2CS

b7	b6	b5	b4	b3	b2	b1	b0
0	**NPAK2**	**NPAK1**	**NPAK0**	**FULL**	**EMPTY**	**0**	**STALL**
R	R	R	R	R	R	R	R/W
0	0	1	0	1	0	0	0

NPAK2~NPAK0－端点2 FIFO中数据包数目，范围0到4

FULL－端点缓冲区满，1缓冲区为满

EMPTY－端点缓冲区空，1缓冲区为空

STALL－端点停止

8.6.9 端点 4 控制与状态寄存器

✧ EP4CS

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**NPAK1**	**NPAK0**	**FULL**	**EMPTY**	**0**	**STALL**
R	R	R	R	R	R	R	R/W
0	0	1	0	1	0	0	0

NPAK1,NPAK0－端点4 FIFO中数据包数目，范围0到2

8.6.10 端点 6 和 8 控制与状态寄存器

✧ EP6CS

b7	b6	b5	b4	b3	b2	b1	b0
0	**NPAK2**	**NPAK1**	**NPAK0**	**FULL**	**EMPTY**	**0**	**STALL**
R	R	R	R	R	R	R	R/W
0	0	0	0	0	1	0	0

✧ EP8CS

b7	b6	b5	b4	b3	b2	b1	b0
0	**NPAK2**	**NPAK1**	**NPAK0**	**FULL**	**EMPTY**	**0**	**STALL**
R	R	R	R	R	R	R	R/W
0	0	0	0	0	1	0	0

8.6.11 slave FIFO 模式端点 2,4,6,8 标志寄存器

✧ EP4FIFOFLGS，EP6FIFOFLGS ，EP4FIFOFLGS，EP6FIFOFLGS

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**PF**	**EF**	**FF**
R	R	R	R	R	R	R	R
0	0	0	0	0	0	1	0

PF－FIFO 编程标志

EF－FIFO 空标志

FF－FIFO 满标志

8.6.12 slave FIFO 模式端点 2 FIFO 字节计数高字节

✧ EP2FIFOBCH

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**BC12**	**BC11**	**BC10**	**BC9**	**BC8**
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

端点2 FIFO最大字节数为4096字节

8.6.13 slave FIFO 模式端点 6 FIFO 字节计数高字节

✧ EP6FIFOBCH

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**BC11**	**BC10**	**BC9**	**BC8**
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

端点6 FIF0最大字节数为2048字节

8.6.14 slave FIFO 模式端点 4 和 8 FIFO 字节计数高字节

✧ EP4FIFOBCH, EP8FIFOBCH

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**BC10**	**BC9**	**BC8**
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

端点4和8 FIFO最大字节数为1024字节

8.6.15 slave FIFO 模式端点 2,4,6,8 FIFO 字节计数低字节

✧ EP2FIFOBCL ， EP4FIFOBCL ， EP6FIFOBCL ， EP8FIFOBCL

b7	b6	b5	b4	b3	b2	b1	b0
BC7	**BC6**	**BC5**	**BC4**	**BC3**	**BC2**	**BC1**	**BC0**
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

8.6.16 SETUP 数据指针寄存器

✧ SUDPTRH，数据指针高字节

b7	b6	b5	b4	b3	b2	b1	b0
A15	**A14**	**A13**	**A12**	**A11**	**A10**	**A9**	**A8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

✧ SUDPTRL，数据指针低字节

b7	b6	b5	b4	b3	b2	b1	b0
A7	**A6**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R
x	x	x	x	x	x	x	0

✧ SUDPTRCTL，SETUP 数据指针自动模式

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**0**	**0**	**SDPAUTO**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	1

为了发送一个数据块到，数据块的起始地址加载到 SUDPTRH:L， 数据块的长度必须先设

置好，设置方式取决于 SUDPTRCTL。

SDPAUTO－0 手动模式：用于一般数据块传输，块长度写到 EP0BCH:L 中

1 自动模式：仅用于发送描述符，块长度自动从描述符长度自动读取，无需设置 EP0BCH:L。

✧ SETUPDAT[8]，SETUP 数据

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

8.7 GPIF 寄存器

8.7.1 GPIF 波形选择

✧ GPIFWFSELECT

b7	b6	b5	b4	b3	b2	b1	b0
SINGLEWR1	**SINGLEWR0**	**SINGLERD1**	**SINGLERD0**	**FIFOWR1**	**FIFOWR0**	**FIFORD1**	**FIFORD0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	0	0	1	0	0

SINGLEWR1:0－简单读方式波形索引

SINGLERD1:0－简单写方式波形索引

FIFOWR1:0－FIFO写方式波形索引

FIFORD1:0－FIFO读方式波形索引

GPIF波形编辑软件可以生成4个独立的波形，可用于不同数据读写方式，可通过索引号来选择。

8.7.2 GPIF 传输完成和空闲控制与状态寄存器

✧ GPIFIDLECS

b7	b6	b5	b4	b3	b2	b1	b0
DONE	**0**	**0**	**0**	**0**	**0**	**0**	**IDLEDRV**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	0	0	0	0	0	0	0

DONE－0传输正在进行，1传输完成

IDLEDRV－空闲状态总线驱动，0 三态，1 驱动

8.7.3 CTL 引脚设置

✧ GPIFIDLECTL，总线非激活时，CTL 引脚状态

b7	b6	b5	b4	b3	b2	b1	B0
0/ CTLOE3	**0/ CTLOE2**	**CTL5/ CTLOE1**	**CTL4/ CTLOE0**	**CTL3**	**CTL2**	**CTL1**	**CTL0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	1	1	1	1	1	1	1

CTLOE3:0－CTL 输出使能

CTL5:0－CLT 输出状态

✧ GPIFCTLCFG，CTL 输出驱动模式

b7	b6	b5	b4	b3	b2	b1	b0
TRICTL	**0**	**CTL5**	**CTL4**	**CTL3**	**CTL2**	**CTL1**	**CTL0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

通过TRICLT来控制CTLx引脚输出方式，见表8.9

表8.9 GPIF模式 引脚CTL输出设置

TRICTL (GPIFCTLCFG.7)	**GPIFCTLCFG[6:0]**	**CTL[3:0]**	**CTL[5:4]**
0	0	CMOS非三态输出	CMOS非三态输出
0	1	开漏输出	开漏输出
1	X	CMOS三态输出	CMOS三态输出

表8.10 GPIF在空闲器件，控制引脚输出状态

TRICTL	**Control Output**	**Output State**	**Output Enable**
0	CTL0	GPIFIDLECTL.0	如TRICTL＝1，输出始终使能
	CTL1	GPIFIDLECTL.1	
	CTL2	GPIFIDLECTL.2	
	CTL3	GPIFIDLECTL.3	
	CTL4	GPIFIDLECTL.4	
	CTL5	GPIFIDLECTL.5	
1	CTL0	GPIFIDLECTL.0	GPIFIDLECTL.4
	CTL1	GPIFIDLECTL.1	GPIFIDLECTL.5
	CTL2	GPIFIDLECTL.2	GPIFIDLECTL.6
	CTL3	GPIFIDLECTL.3	GPIFIDLECTL.7
	CTL4	N/A	
	CTL5	(CTL4 and CTL5 are not available when TRICTL = 1)	

8.7.4 GPIF 地址

✧ GPIFADRH，地址高字节

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**0**	**0**	**GPIFA8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

✧ GPIFADRL，地址低字节

b7	b6	b5	b4	b3	b2	b1	b0
GPIFA7	**GPIFA6**	**GPIFA5**	**GPIFA4**	**GPIFA3**	**GPIFA2**	**GPIFA1**	**GPIFA0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

写入GPIF地址数据可控制引脚ADR[7:0]状态

8.7.5 GPIF 数据传输计数

✧ GPIFTCB3，计数字节 3

b7	b6	b5	b4	b3	b2	b1	b0
TC31	**TC30**	**TC29**	**TC28**	**TC27**	**TC26**	**TC25**	**TC24**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

✧ GPIFTCB2，计数字节 2

b7	b6	b5	b4	b3	b2	b1	b0
TC23	**TC22**	**TC21**	**TC20**	**TC19**	**TC18**	**TC17**	**TC16**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

✧ GPIFTCB1，计数字节 1

b7	b6	b5	b4	b3	b2	b1	b0
TC15	**TC14**	**TC13**	**TC12**	**TC11**	**TC10**	**TC9**	**TC8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

✧ GPIFTCB0 计数字节 1

b7	b6	b5	b4	b3	b2	b1	b0
TC7	**TC6**	**TC5**	**TC4**	**TC3**	**TC2**	**TC1**	**TC0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	1

计数字节范围为4G。

8.7.6 GPIF 模式端点 2，4，6，8 标志选择

✧ EP2GPIFFLGSEL，EP4GPIFFLGSEL，EP6GPIFFLGSEL，EP8GPIFFLGSEL

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**0**	**FS1**	**FS0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

FS1:0－见表8.11

表8.11 端点标志选择

FS1	FS0	Flag
0	0	Programmable
0	1	Empty
1	0	Full
1	1	Reserved

8.7.7 GPIF 模式端点 2,4,6,8Programmable 标志停止数据传输

✧ EP2GPIFPFSTOP，EP4GPIFPFSTOP，EP6GPIFPFSTOP，EP8GPIFPFSTOP

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**0**	**0**	**FIFO[2,4,6,8] FLAG**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

FIFO[2,4,6,8]FLAG－如果寄存器EPxGPIFFLGSEL设置为Programmable标志，1传输完成0传输未完成

8.7.8 slave FIFO 模式端点 2,4,6,8 触发器

✧ EP2GPIFTRIG, EP4GPIFTRIG, EP6GPIFTRIG, EP8GPIFTRIG

b7	b6	b5	b4	b3	b2	b1	b0
x	**x**	**x**	**x**	**x**	**x**	**x**	**x**
W	W	W	W	W	W	W	W
x	x	x	x	x	x	x	x

写入0xFF将启动一次GPIF写操作，读该寄存器将启动一次GPIF读操作。

8.7.9 GPIF 数据高字节（16 位模式）

✧ XGPIFSGLDATH

b7	b6	b5	b4	b3	b2	b1	b0
D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

D15:8－包括使用GPIF波形要写入的数据或从引脚FD15:8要读入的数据

8.7.10 GPIF 数据低字节和触发传输

✧ XGPIFSGLDATLX

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

D7:0－包括使用GPIF波形要写入的数据或从引脚FB15:8要读入的数据，读写低字节将触发一次GPIF数据传输操作

8.7.11 读取 GPIF 低字节数据和非触发传输

✧ XGPIFSGLDATLNOX

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R	R	R	R	R	R	R	R
x	x	x	x	x	x	x	x

8.7.12 GPIF RDY 引脚配置

✧ GPIFREADYCFG

b7	b6	b5	b4	b3	b2	b1	b0
INTRDY	**SAS**	**TCXRDY5**	**0**	**0**	**0**	**0**	**0**
R/W	R/W	R/W	R	R	R	R	R
0	0	0	0	0	0	0	0

INTRDY－内部RDY，作为第六个RDY输入，与其它引脚RDY不同，它由内部固件控制

SAS－与 GPIF IFCLK 同步异步控制，如果使用内部的 IFCLK 时钟，1 为同步方式，RDY 必须满足建立和保持时间要求，0 为异步方式

TCXRDY5－1 表示用 GPIF 计数结束取代 RDY5 引脚输入，0 表示则 RDY5 信号来自引脚输入

8.7.13 GPIF 状态寄存器

✧ GPIFREADYSTAT，RDY 引脚状态，只读

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**RDY5**	**RDY4**	**RDY3**	**RDY2**	**RDY1**	**RDY0**
R	R	R	R	R	R	R	R
0	0	x	x	x	x	x	x

8.7.14 放弃 GPIF 传输

✧ GPIFABORT

b7	b6	b5	b4	b3	b2	b1	b0
x	**x**	**x**	**x**	**x**	**x**	**x**	**x**

W	W	W	W	W	W	W	W
x	x	x	x	x	x	x	x

写入0xff立刻放弃GPIF传输，进入空闲状态

8.8 GPIF Flowstate/UDMA 寄存器

✧ FLOWSTATE，流状态

b7	b6	b5	b4	b3	b2	b1	b0
FSE	**0**	**0**	**0**	**0**	**FS[2:0]**		
0	0	0	0	0	0	0	0
RW	R	R	R	R	RW	RW	RW

GPIF Flowstate传输是通过状态图的方式来工作，最多可定义7个状态

FSE－流状态使能，写入1使能，只有使能FS[2:0]才有意义，写入0非使能

FS[2:0]－流状态，有效值为0到6

✧ FLOWLOGIC ，流逻辑控制

b7	b6	b5	b4	b3	b2	b1	b0
LFUNC[1:0]		**TERMA[2:0]**			**TERMB[2:0]**		
0	0	0	0	0	0	0	0
RW	RW	RW	RW	RW	RW	RW	RW

GPIF Flowstate传输中为了实现状态的跳转，必须定义跳转控制逻辑，由变量A和变量B组成。

LFUNC[1:0]－逻辑定义

00 = A AND B

01 = A OR B

10 = A XOR B

11 = !A AND B

THERMA[2:0]，THERMB[2:0]－逻辑变量A和B定义

0 = RDY[0]

1 = RDY[1]

2 = RDY[2]

3 = RDY[3]

4 = RDY[4]

5 = RDY[5] or TC-Expiration (depending on GPIF_READYCFG.5)

6 = FIFO Flag (PF, EF, or FF depending on GPIF_EPxFLAGSEL)

7 = 8051 RDY (GPIF_READYCFG.7)

✧ FLOWEQ0CTL

b7	b6	b5	b4	b3	b2	b1	b0
CTLOE3	**CTLOE2**	**CTLOE1/ CTL5**	**CTLOE0/ CTL4**	**CTL3**	**CTL2**	**CTL1**	**CTL0**
0	0	0	0	0	0	0	0
RW	RW	RW	RW	RW	RW	RW	RW

✧ FLOWEQ1CTL

b7	b6	b5	b4	b3	b2	b1	b0
CTLOE3	**CTLOE2**	**CTLOE1/ CTL5**	**CTLOE0/ CTL4**	**CTL3**	**CTL2**	**CTL1**	**CTL0**
0	0	0	0	0	0	0	0
RW	RW	RW	RW	RW	RW	RW	RW

FLOWEQ0CTL是流控制逻辑为0时引脚CTL5:0输出状态

FLOWEQ1CTL 是流控制逻辑为零 1 引脚 CTL5:0 输出状态

表8.12 在流状态下CTL引脚输出状态

TRICTL	Control Output	Output State	Drive Type (0 = CMOS, 1 = Open-Drain)	Output Enable
0	CTL0	FLOWEQxCTL.0	GPIFCTLCFG.0	N/A (CTL Outputs are always enabled when TRICTL = 0)
	CTL1	FLOWEQxCTL.1	GPIFCTLCFG.0	
	CTL2	FLOWEQxCTL.2	GPIFCTLCFG.0	
	CTL3	FLOWEQxCTL.3	GPIFCTLCFG.0	
	CTL4	FLOWEQxCTL.4	GPIFCTLCFG.0	
	CTL5	FLOWEQxCTL.5	GPIFCTLCFG.0	
1	CTL0	FLOWEQxCTL.0	N/A (CTL Outputs are always tristatable CMOS when TRICTL = 1)	FLOWEQxCTL.4
	CTL1	FLOWEQxCTL.1		FLOWEQxCTL.5
	CTL2	FLOWEQxCTL.2		FLOWEQxCTL.6
	CTL3	FLOWEQxCTL.3		FLOWEQxCTL.7
	CTL4	N/A (CTL4 and CTL5 are not available when TRICTL = 1)		
	CTL5			

FLOWHOLDOFF，数据保持时间

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**0**	**HOLDTIME[1:0]**	
R	R	R	R	R	R	RW	RW
0	0	0	0	0	0	0	0

HOCTL[2:0]－数据保持时间

00 = 0 IFCLK cycles

01 = ½ IFCLK cycle

10 = 1 IFCLK cycle

11 = Reserved

✧ FLOWSTB

b7	b6	b5	b4	b3	b2	b1	b0
SLAVE	**RDYASYNC**	**CTLTOGL**	**SUSTAIN**	**0**	**MSTB[2:0]**		
0	0	1	0	0	0	0	0
RW	RW	RW	RW	R	RW	RW	RW

SLAVE－写入0则GPIF是传输总线的主，则由控制引脚CTL5:0其中一个作为主触发，
写入1则GPIF是传输总线的从，则由控制引脚的RDY5:0其中一个作为主触发
RDYASYNC－如果SLAVE=0，该位不起作用，写入0则RDY与IFCLK异步，写入1则同步
CTLTOGL－如果SLAVE=1，该位不起作用。写入1如果流逻辑为1则数据写到总线上
SUSTAIN－如果 SLAVE=1，该位不起作用。
MSTB[2:0]－CTL5:0 与 RDY5:0 选择

✧ FLOWSTBEDGE，触发边沿

b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**0**	**0**	**0**	**0**	**FALLING**	**RISING**
R	R	R	R	R	R	R/W	RW
0	0	0	0	0	0	0	1

FALLING－下降沿触发
RISING－上升沿触发

✧ FLOWSTBHPERIOD

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
RW	RW	RW	RW	RW	RW	RW	RW
0	0	0	0	0	0	1	0

如果是CTL引脚，主触发信号半周期包含的IFCLK周期数目，如果是2则主触发信号半周期是为一个完整IFCLK时钟。

8.9 端点缓冲区

✧ EP0BUF[64]，端点 0 缓冲区，共 64 字节

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X

✧ EP1OUTBUF[64] ，EP1-OUT 缓冲区，共 64 字节

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X

✧ EP1INBUF[64] ，EP1-IN 缓冲区，共 64 字节

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X

✧ EP2FIFOBUF[1024] ， EP2 缓冲区

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X

✧ EP4FIFOBUF[512] ， EP4 缓冲区

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X

✧ EP6FIFOBUF[1024] ， EP6 缓冲区

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X

✧ EP8FIFOBUF[512] ， EP8 缓冲区

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
X	X	X	X	X	X	X	X

8.10 GPIF 波形存储器寄存器

✧ GPIF_WAVE_DATA，波形存放起始地址

b7	b6	b5	b4	b3	b2	b1	b0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
x	x	x	x	x	x	x	x

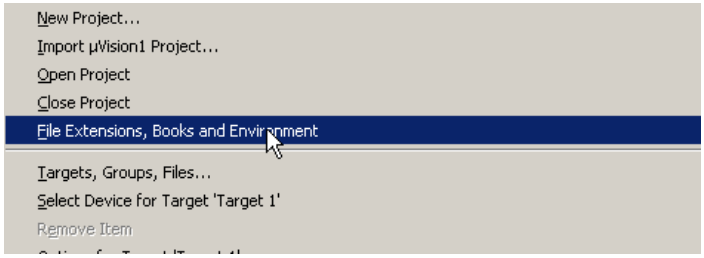
RES_WAVEDATA_END，GPIF 波形存放结束地址

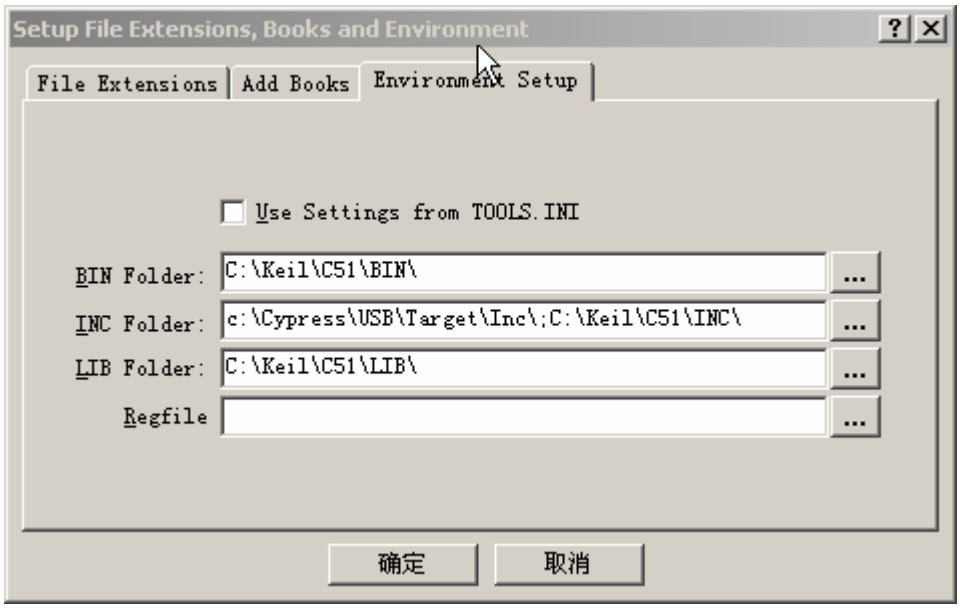
附录

KEIL 环境设置

如果在安装 KEIL7.02 后发现不能编译程序，则首先需要按照下图设置 KEIL 编译环境。

KEIL 编译菜单的 Project 菜单





KEIL 路径设置

参考资料

1. USB 1.1 中文版
2. USB 2.0 英文版
3. USB OTG 英文版
4. 颜容江，EZ-USB 2100 系列单片机机原理、编程及应用，北京航空航天大学出版社
5. Christ Cant. Windows WDM 设备驱动程序开发指南，机械工业出版社
6. Water Only. Programming the Microsoft Windows Driver Mode. 中文版
7. www.cypress.com
8. www.driverdevelop.com
9. www.flickercontrol.com
10. 张剑波，颜钢锋，“基于 GPIF 的 USB-ATA 解决方案”，计算机应用