USB CY7C68013
固件程序说明
黑金开发板

ALINX

文档版本控制

文档版本	修改内容记录
REV1.0	创建文档

版权申明：

感谢您购买黑金开发板，在使用产品之前，请仔细地阅读该手册并且确保知道如何正确使用该产品，不合理的操作可能会损坏开发板，使用过程中随时参考该手册以确保正确使用。

此手册不断更新中，建议您使用时下载最新版本。

软件版本：

黑金官网：

Http://www.alinx.com.cn

黑金动力社区：

http://www.heijin.org

黑金官方淘宝店：

http://oshcn.taobao.com

联系方式：

021-67676997

黑金微信公众号：

ALINX-HEIJIN

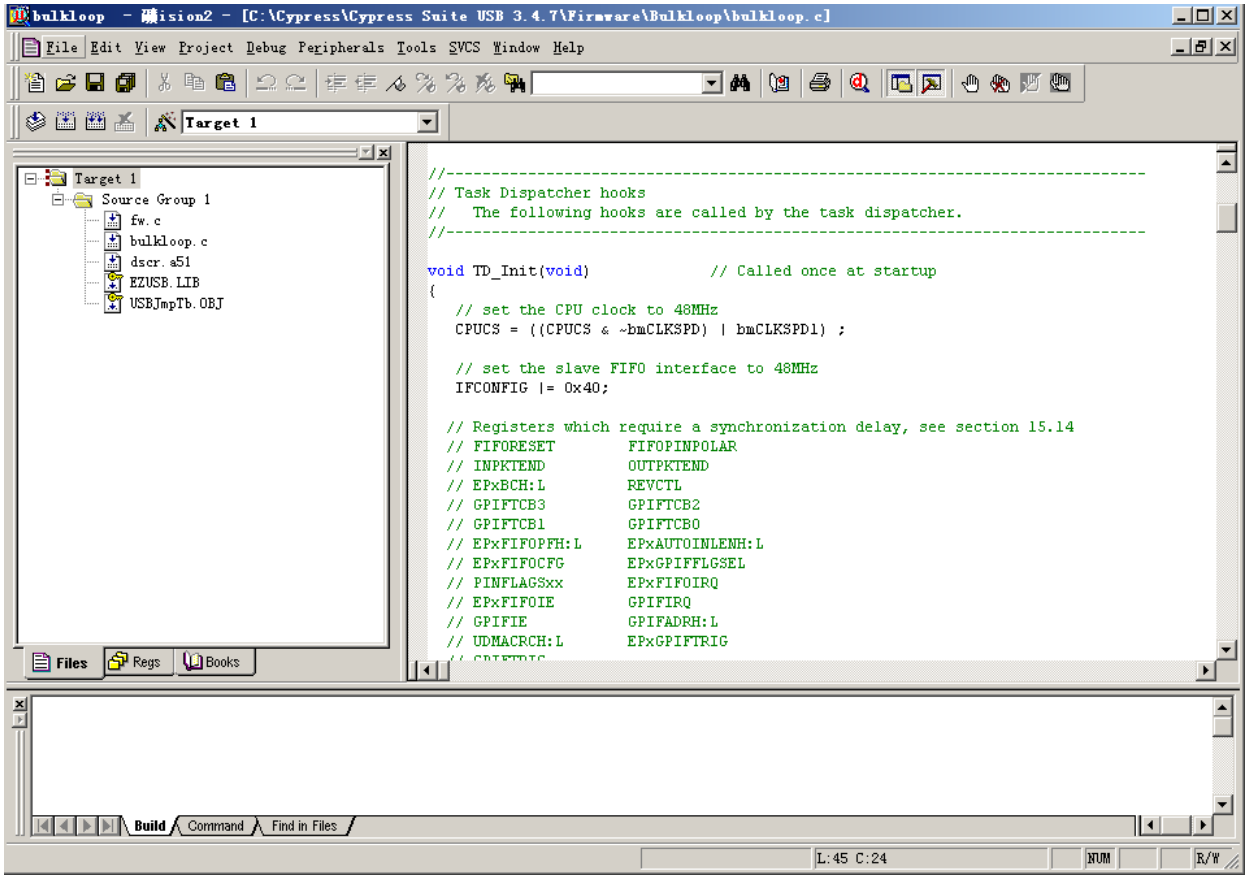
目 录

一、 软件安装

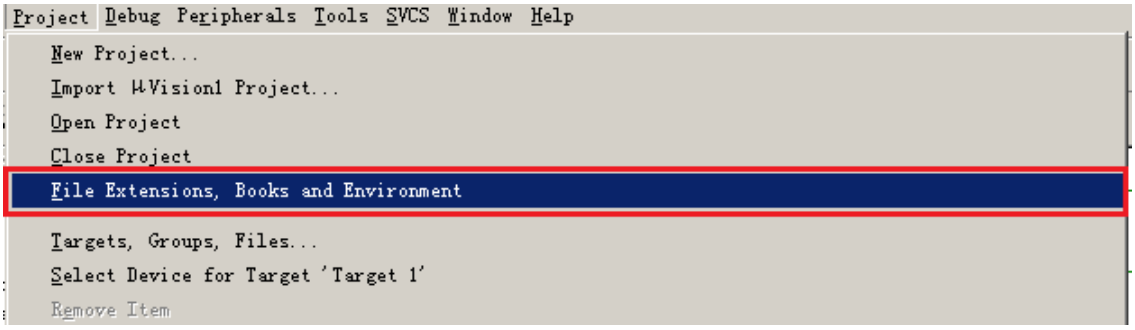
设计USB2.0 CY7C68013的固件程序之前需要先安装"KEIL uVsion2"软件和"CySuiteUSB_3_4_7_ B204.exe".这两个软件分别可以在我们提供的光盘的12_工具和07_驱动\USB2.0目录下找到。

二、 KEIL C环境设置

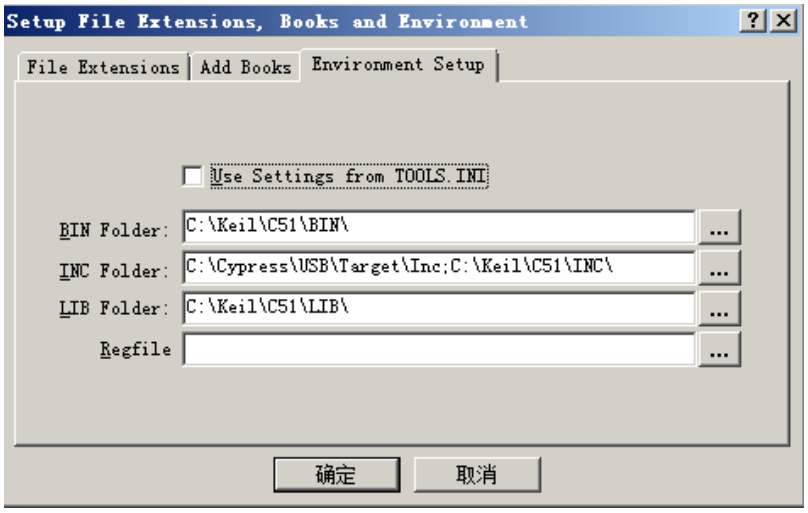
1.打开Keil uVsion2工具，再点击菜单Project->Open Project打开Cypress\USB\Examples\ FX2LP目录下的Bulkloop.uV2工程。



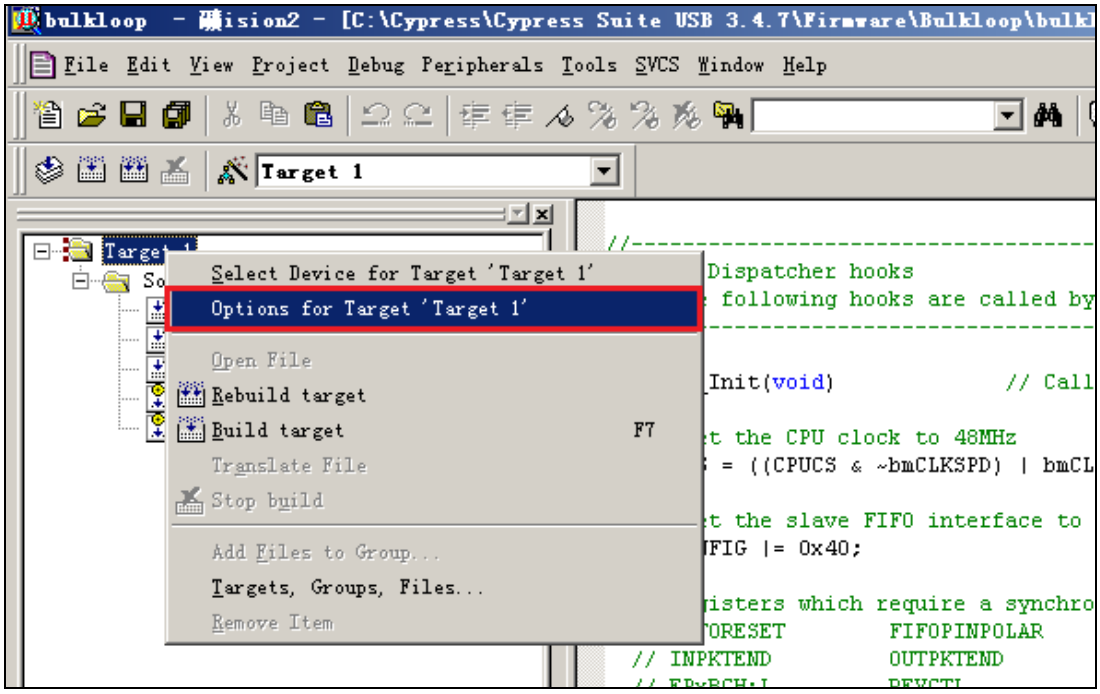
2. 按照下图设置KEIL编译环境，点击Project菜单选择File Extensions, Books and Enviroment。



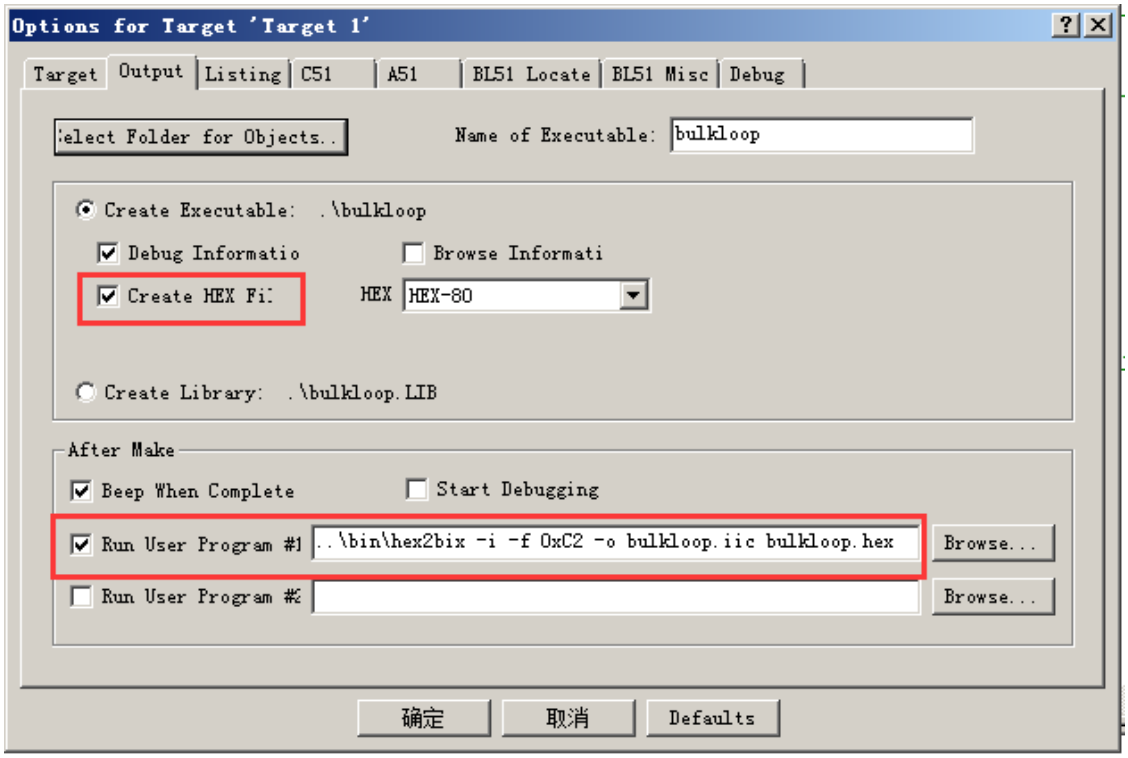
根据软件安装的目录设置正确的路径，我们这里Cypress的软件和KEIL C都安装在C盘的根目录下，所以设置不需要更改。



3. 右键Target 1选择Options for Target 'Target 1'，打开编译器设置界面。

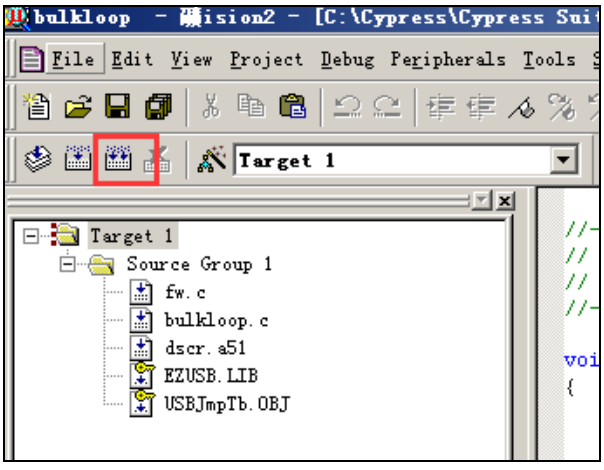


在Output界面，选择生成HEX文件和IIC文件(默认已经选择)。其中HEX文件是下载到CY7C68013内部 RAM的固件程序，IIC文件用于下载到外部EEPROM中的固件程序。



三、 KEIL 工程编译

点击Build All按钮开始编译，编译后会生成Bulkloop.hex和Bulkloop.IIC文件。



编译成功。

```
assembling dscr.a51...
linking...
creating hex file from "bulkloop"...
User command #1: ..\bin\hex2bix -i -f 0xC2 -o bulkloop.iic bulkloop.hex
Intel Hex file to EZ-USB Binary file conversion utility
Copyright (c) 1997-2005, Cypress Semiconductor Inc.
2424 Bytes written.
Total Code Bytes = 2383
Conversion completed successfully.
"bulkloop" - 0 Error(s), 0 Warning(s).
Build  Command  Find in Files
```

四、 bulkloop 工程文件说明

工程下，有以下几个文件：，其中，USBJmp.OBJ, EZUSB.LIB基本上是每个工程都要添加的，是一些中断向量表，EZUSB的函数库等等，不用管它们。

现在重点看前面三个文件：

1. fw.c:

这个文件是整个USB的固件根本（FirmWare的缩写），USB协议方面的通信都是在这里完成的，包括上电枚举，重枚举，唤醒以及调用用户自己的程序和控制命令。基本上，如非必要，尽量不要动这个文件的内容，也不要在里面书写你自己的任何代码。

2. bulkllop.c:

这个就是用户自己的代码书写文件（原始名称：periph.c）。我们所有的代码都在这个文件里书写。Cypress已经给我们搭好了架构。

void TD_Init(void)：这个函数只会在USB启动后调用一次。在这个函数里添加你自己的初始化代码，也就是传输数据前要处理的，例如IO口配置，时钟，端点，FIFO的选择等等。 我们看bulkloop的初始化，它在USB的in，out传输启动前进行了哪些初始化：CPU时钟频率，USB工作模式选择，端点选择，端点传输方向，FIFO大小的配置等等。

void TD_Poll(void)：Poll中文意思调度，这个函数就是用户调度程序，USB会在空闲的时候反复调用该函数，所以我们把自己需要反复执行的代码放在这里。例如在bulkloop里，它就实现了反复从端点2接收上位机数据然后传给端点6，再从端点6传给上位机（4，8端点一样）。

BOOL DR_VendorCmnd(void)：这个函数就是自定义命令代码的书写处。我们的Vendor命令都会写在这里，fw.c固件会自动调用我们的代码。

void ISR_Ep0in(void) interrupt 0～void ISR_Ep8inout(void) interrupt 0：这几个函数是当使用端点中断传输时，中断代码的书写处，很少用。

以上，是经常会用到的几个函数；其他，基本不常用。

3.dscr.51:

这个文件是USB描述符文件，包括了设备描述符，接口描述符，端点描述符，字符串等等。里面的英文都注释得很详细了，我就不多做介绍了，刚开始入门的时候，这个文件也不必改动。

4. 其它包含的头文件：

fx2.h:预定义，宏及函数声明

fx2regs.h:68013的寄存器地址定义。

syncdly.h:同步延时。在其他文件里经常调用的一个函数SYNCDELAY就是这里定义的。

intrins.h:C51一些数据类型及函数定义。

关于Bulkloop的具体功能大家可以看bulkloop文件夹下的readme.txt文件，它告诉我们这个固件主要实现的功能。

五、 工程修改

针对黑金FPGA开发板的CY7C68013A的固件程序，我们只需要修改bulkloop项目中的bulkllop.c文件里的TD_Init函数和TD_Poll函数。

1. TD_Init函数修改如下：

```
void TD_Init(void)             // Called once at startup
{
   // set the CPU clock to 48MHz
   CPUCS = ((CPUCS & ~bmCLKSPD) | bmCLKSPD1) ;

   // Set Slave FIFO mode
   IFCONFIG |=  0x4B;

  // Registers which require a synchronization delay, see section 15.14
  // FIFORESET         FIFOPINPOLAR
  // INPKTEND          OUTPKTEND
  // EPxBCH:L          REVCTL
  // GPIFTCB3          GPIFTCB2
  // GPIFTCB1          GPIFTCB0
  // EPxFIFOPFH:L      EPxAUTOINLENH:L
  // EPxFIFOCFG        EPxGPIFFLGSEL
  // PINFLAGSxx        EPxFIFOIRQ
  // EPxFIFOIE         GPIFIRQ
  // GPIFIE            GPIFADRH:L
  // UDMACRCH:L        EPxGPIFTRIG
  // GPIFTRIG

  // Note: The pre-REVE EPxGPIFTCH/L register are affected, as well...
  //       ...these have been replaced by GPIFTC[B3:B0] registers

  // default: all endpoints have their VALID bit set
  // default: TYPE1 = 1 and TYPE0 = 0 --> BULK
  // default: EP2 and EP4 DIR bits are 0 (OUT direction)
  // default: EP6 and EP8 DIR bits are 1 (IN direction)
  // default: EP2, EP4, EP6, and EP8 are double buffered

  SYNCDELAY;
  PINFLAGSAB = 0xC8;            // FLAGB - fixed EP2FF  FLAGA - EP2EF
  SYNCDELAY;
  PINFLAGSCD = 0xDE;            // FLAGD - unvalid      FLAGC - fixed EP6FF

  SYNCDELAY;
  PORTACFG = 0x40;              // func. of PA7 pin is SLCS#
```

```
SYNCDELAY;
FIFOPINPOLAR = 0x00;            // all signals active low
SYNCDELAY;

// we are just using the default values, yes this is not necessary...
EP1OUTCFG = 0xA0;
EP1INCFG = 0xA0;
SYNCDELAY;                      // see TRM section 15.14
EP2CFG = 0xA2;                  //EP2 OUT, 512BYTE BULK, DUBBLE BUFFER
SYNCDELAY;
EP4CFG = 0xA0;                  //EP4 OUT, BULK
SYNCDELAY;
EP6CFG = 0xE2;                  //EP6 IN, 512BYTE BULK,  DUBBLE BUFFER
SYNCDELAY;
EP8CFG = 0xE0;                  //EP8 IN, BULK

// handle the case where we were already in AUTO mode...
EP2FIFOCFG = 0x00;              // AUTOOUT=0, WORDWIDE=0
EP4FIFOCFG = 0x00;              // AUTOOUT=0, WORDWIDE=0
SYNCDELAY;
EP2FIFOCFG = 0x11;              // AUTOOUT=1, WORDWIDE=1
EP4FIFOCFG = 0x11;              // AUTOOUT=1, WORDWIDE=1
SYNCDELAY;

EP6FIFOCFG = 0x00;              // AUTOIN=0, WORDWIDE=0
EP8FIFOCFG = 0x00;              // AUTOIN=0, WORDWIDE=0
SYNCDELAY;
EP6FIFOCFG = 0x09;              // AUTOIN=1, WORDWIDE=1
EP8FIFOCFG = 0x09;              // AUTOIN=1, WORDWIDE=1
SYNCDELAY;
// out endpoints do not come up armed

// since the defaults are double buffered we must write dummy byte counts twice
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;
```

```
  // enable dual autopointer feature
  AUTOPTRSETUP |= 0x01;
}
```

初始化函数TD_Init里配置CY7C68013的双功能的IO为Slave FIFO模式，引脚FLAGA用于指示EP2端口的空标志，FLAGB用于指示EP2端口的满标志， FLAGC用于指示EP6端口的满标志。另外端口2和端口4配置成Bulk输出，端口6和端口8配置成Bulk输入。关于寄存器的说明请参考第7章。

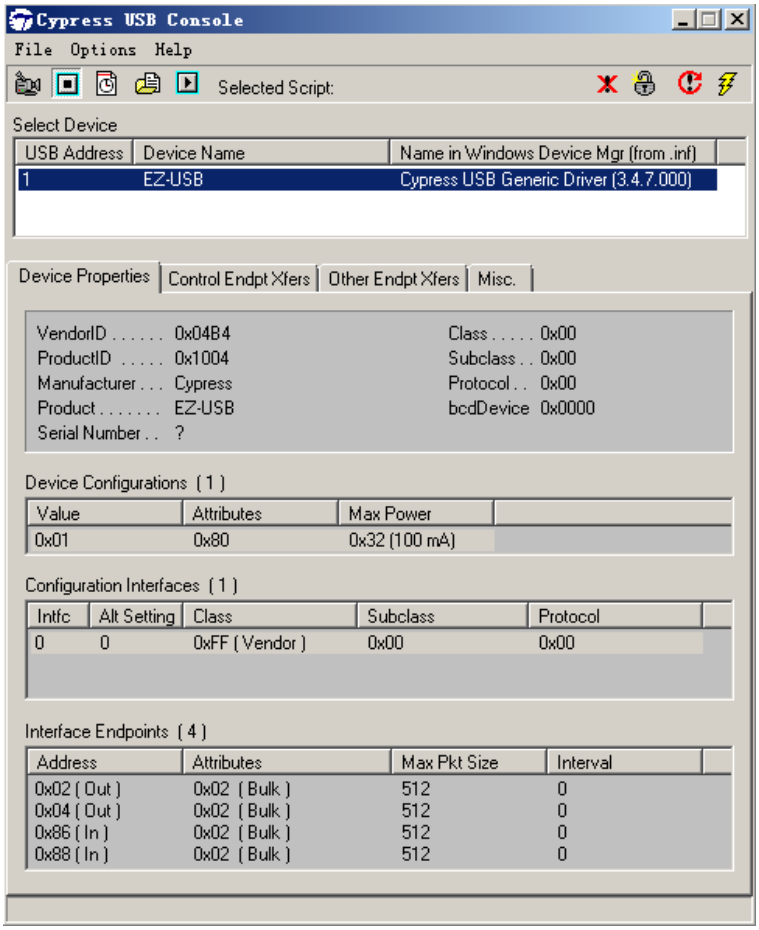
2. 屏蔽掉TD_Poll函数里的全部内容，不然USB固件会自动把EP2的FIFO里的数据读到EP6的FIFO中，把EP4的FIFO里的数据读到EP8的FIFO中。因为Loop这个功能我们会在FPGA的usb_test.v程序中实现了。

```
void TD_Poll(void)              // Called repeatedly while the device is idle
{
 /* WORD i;
  WORD count;

  if(!(EP2468STAT & bmEP2EMPTY))
  { // check EP2 EMPTY(busy) bit in EP2468STAT (SFR), core set's this bit when FIFO is empty
     if(!(EP2468STAT & bmEP6FULL))
     {  // check EP6 FULL(busy) bit in EP2468STAT (SFR), core set's this bit when FIFO is full
        APTR1H = MSB( &EP2FIFOBUF );
        APTR1L = LSB( &EP2FIFOBUF );

        AUTOPTRH2 = MSB( &EP6FIFOBUF );
        AUTOPTRL2 = LSB( &EP6FIFOBUF );

        count = (EP2BCH << 8) + EP2BCL;

        // loop EP20UT buffer data to EP6IN
        for( i = 0x0000; i < count; i++ )
        {
           // setup to transfer EP20UT buffer to EP6IN buffer using AUTOPOINTER(s)
           EXTAUTODAT2 = EXTAUTODAT1;
        }
        EP6BCH = EP2BCH;
        SYNCDELAY;
        EP6BCL = EP2BCL;        // arm EP6IN
        SYNCDELAY;
        EP2BCL = 0x80;          // re(arm) EP20UT
     }
  }

  if(!(EP2468STAT & bmEP4EMPTY))
  { // check EP4 EMPTY(busy) bit in EP2468STAT (SFR), core set's this bit when FIFO is empty
     if(!(EP2468STAT & bmEP8FULL))
     {  // check EP8 FULL(busy) bit in EP2468STAT (SFR), core set's this bit when FIFO is full
        APTR1H = MSB( &EP4FIFOBUF );
        APTR1L = LSB( &EP4FIFOBUF );

        AUTOPTRH2 = MSB( &EP8FIFOBUF );
        AUTOPTRL2 = LSB( &EP8FIFOBUF );

        count = (EP4BCH << 8) + EP4BCL;

        // loop EP40UT buffer data to EP8IN
        for( i = 0x0000; i < count; i++ )
        {
           // setup to transfer EP40UT buffer to EP8IN buffer using AUTOPOINTER(s)
           EXTAUTODAT2 = EXTAUTODAT1;
        }
        EP8BCH = EP4BCH;
        SYNCDELAY;
        EP8BCL = EP4BCL;        // arm EP8IN
        SYNCDELAY;
        EP4BCL = 0x80;          // re(arm) EP40UT
     }
  } */
}
```

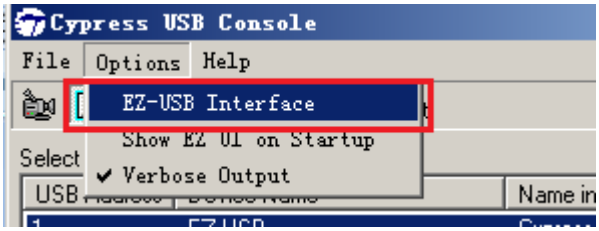
修改完成后保存工程并重新编译生成.hex文件和.iic文件。

六、 固件下载

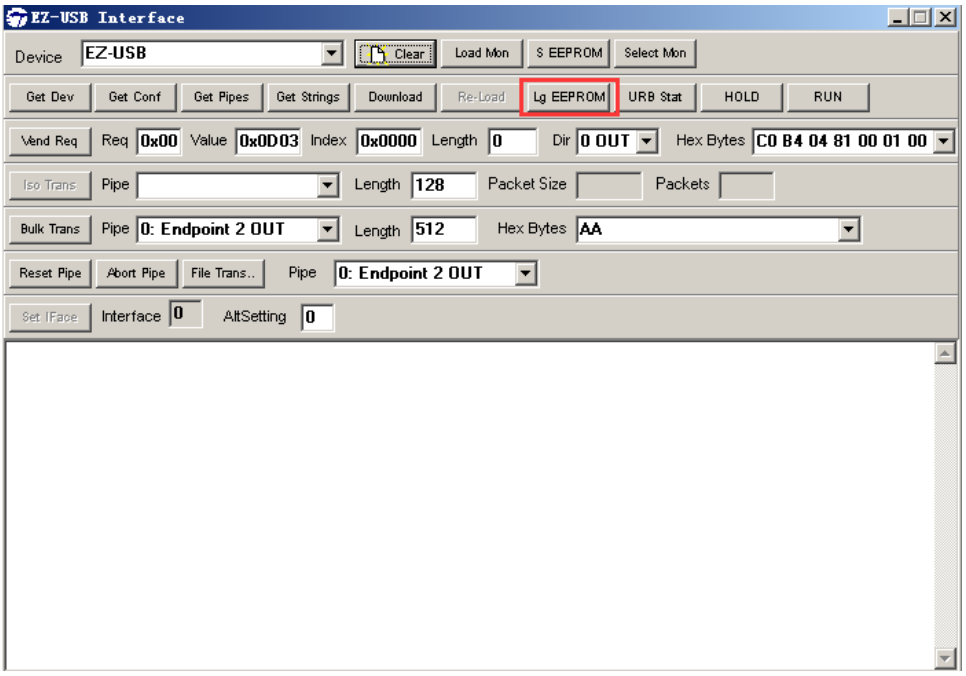
接下去我们要把生成的bulkloop.iic文件下载到开发板上CY7C68013的配置EEPROM里。连接PC的USB口和开发板的USB接口，并打开Cypress USB Console工具，这时我们可以看到PC已经检测到开发板的USB设备。



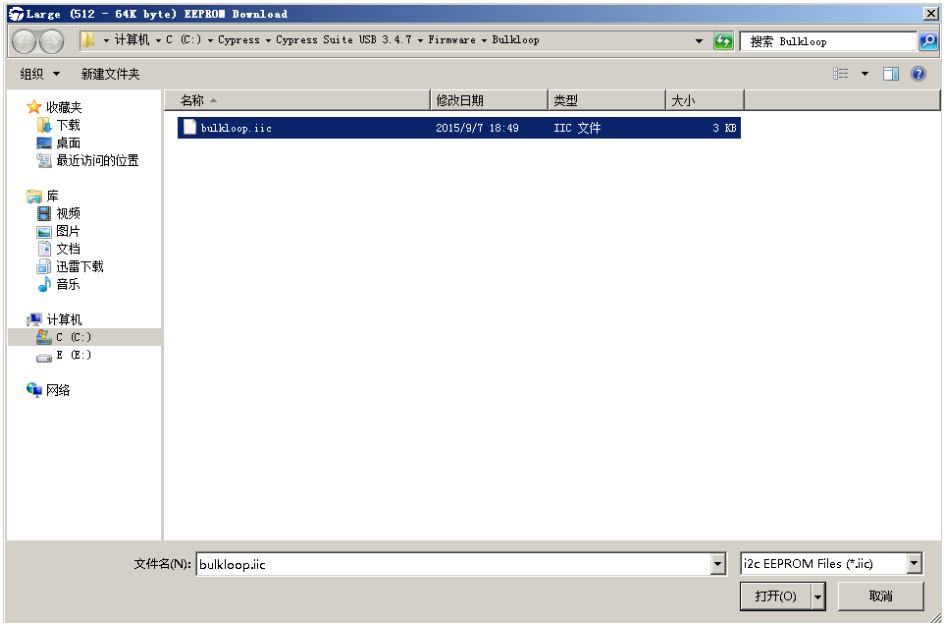
选择Options->EZ-USB Interface



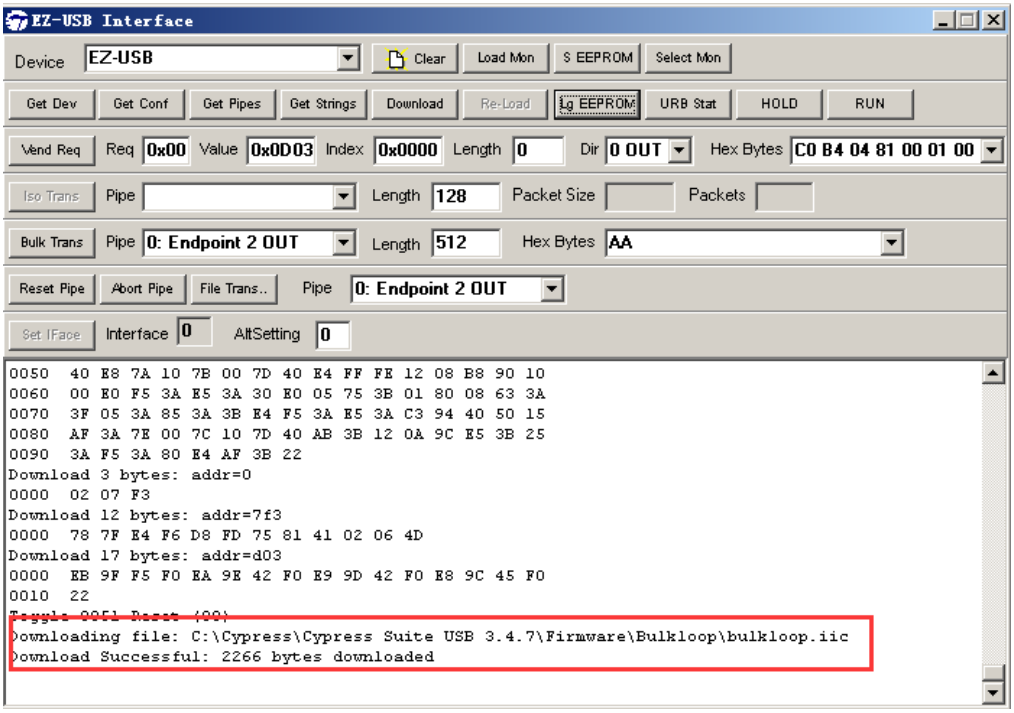
在EZ-USB Interface界面里选择Lg EEPROM按钮



选择刚才生成的bulkloop.iic文件打开。



出现以下的提示说明程序已经下载到EEPROM里了。



开发板重新上电后下载的USB固件就生效了。

七、 CY7C68013 部分寄存器说明

1 系统配置寄存器

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

PORTCSB: 1 读写端口C 时产生RD#和WR#信号，0 不产生读写信号。

CLKSPD1,CLKSPD0: CPU 时钟设置，默认设置为12M，设置见表7.1。

表7.1 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

1.2 接口配置寄存器

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

IFCLOKE: 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,GPIF 调试时可使用。

IFCFG0, IFCFG1: 模式设置，模式决定了多个引脚状态，详见表7.2。

表7.2 模式设置

IFCFG1	IFCFG0	Configuration
0	0	Ports
0	1	Reserved
1	0	GPIF Interface (internal master)
1	1	Slave FIFO Interface (external master)

1.3 Slave FIFO 方式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,FIFOFLAGC 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 方式中，用引脚FLAGA～FLAGD 来定义用端点FIFO 的状态，并可灵活编程来设置FLAGx，见表7.3。

表7.3 FLAGA～FLAGD 引脚功能设置

FLAGx3	FLAGx2	FLAGx1	FLAGx0	引脚功能
0	0	0	0	FLAGA=PF, FLAGB=EF, FLAGC=EF, FLAGD=EP2PF(除 FLAGD 之外，FLAGA FLAGC 对应的端点 FIFO 由引脚 FIFO[0,1]来确定 详见表 3.4)
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 编程标志，EF 表示FIFO 已空，FF 表示FIFO 已满

2 端口配置

2.1 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 则使能端口A 复用引脚功能

2.2 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

2.3 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

3 Slave FIFO 方式信号有效寄存器

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 方式信号有效电平，0 信号低有效，1 信号高有效。

4 断点配置寄存器

4.1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7.5

表7.5 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

4.2 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	**DIR**	**TYPE1**	**TYPE0**	**SIZE**	**0**	**BUF1**	**BUF0**
R/W	R/W	R/W	R/W	R/W	R	R/W	R/W
1	1	1	0	0	0	1	0

EP8CFG，端点8 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	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7.6

表7.6 端点2，4，6，8 类型

TYPE1	TYPE0	类型
0	0	无效
0	1	同步
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7.7

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

4.3 Slave FIFO 方式端点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 使能
AUTOOUT：自动OUT 使能
AUTOIN：自动IN 使能
ZEROLENIN：零长度IN 端点数据包提交使能
WORDWIDE：数据宽度，8 位或16 位

4.4 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