

# **USB CY7C68013A Firmware Program Description**



## Version Record

Version	Date	Release By	Description
Rev 1.0	2022-05-15	Rachel Zhou	First Release

The ALINX official Documents are in Chinese, and the English version was translated by **Shanghai Tianhui** Trading Company. They has **not been officially Review by ALINX** and are for reference only. If there are any errors, please send email feedback to [support@aithtech.com](mailto:support@aithtech.com) for correction.

**Amazon Store:** <https://www.amazon.com/alinx>

**Aliexpress Store:**

<https://alinxfga.aliexpress.com/store/911112202?spm=a2g0o.detail.1000007.1.704e2bedqLBW90>

**Ebay Store:** <https://www.ebay.com/str/alinxfga>

## Customer Service Information

Skype: [rachelhust@163.com](mailto:rachelhust@163.com)

Wechat: [rachelhust](mailto:rachelhust)

Email: [support@aithtech.com](mailto:support@aithtech.com) and [rachehust@163.com](mailto:rachehust@163.com)

## Table of Contents

Version Record .....	2
Part 1: FPGA Development Board Introduction .....	错误！未定义书签。
Part 2: ACU2EG core board .....	错误！未定义书签。
Part 2.1: ACU2EG core board Introduction .....	5
Part 2.2: ZYNQ Chip .....	7
Part 2.3: DDR4 DRAM .....	8
Part 2.4: QSPI Flash .....	9
Part 2.5: eMMC Flash .....	13
Part 2.6: Clock configuration .....	16
Part 2.7: LED .....	错误！未定义书签。
Part 2.8: Power Supply .....	错误！未定义书签。
Part 2.9: ACU2EG Core Board Form Factors .....	错误！未定义书签。
Part 2.10: Board to Board Connectors pin assignment .....	错误！未定义书签。
Part 3: Carrier Board .....	错误！未定义书签。
Part 3.1: Carrier Board Introduction .....	错误！未定义书签。
Part 3.2: M.2 Interface .....	错误！未定义书签。
Part 3.3: DP Interface .....	错误！未定义书签。
Part 3.4: USB3.0 interface .....	错误！未定义书签。
Part 3.5: Gigabit Ethernet Interface .....	错误！未定义书签。
Part 3.6: USB to Serial Port .....	错误！未定义书签。
Part 3.7: SD Card Slot Interface .....	错误！未定义书签。
Part 3.8: Expansion Header .....	错误！未定义书签。
Part 3.9: CAN communication interface .....	错误！未定义书签。
Part 3.10: 485 communication interface .....	错误！未定义书签。
Part 3.11: MIPI camera interface .....	错误！未定义书签。

Part 3.12: JTAG Debug Port .....	错误！未定义书签。
Part 3.13: Real-time clock .....	错误！未定义书签。
Part 3.14: EEPROM and Temperature sensor .....	错误！未定义书签。
Part 3.15: User LEDs .....	错误！未定义书签。
Part 3.16: Keys .....	错误！未定义书签。
Part 3.17: DIP Switch Configuration .....	错误！未定义书签。
Part 3.18: Power Supply .....	错误！未定义书签。
Part 3.19: ALINX Customized Fan .....	错误！未定义书签。
Part 3.20: Carrier Board Form Factors .....	错误！未定义书签。

## Part 1: Software Installation

First install the "KEIL uVSION2" software and "CySuiteUSB\_3\_4\_7\_B204.exe". These two software can be found in the "07\_Software Tools and Drivers\USB2.0 Driver" directory of the documents we provide.



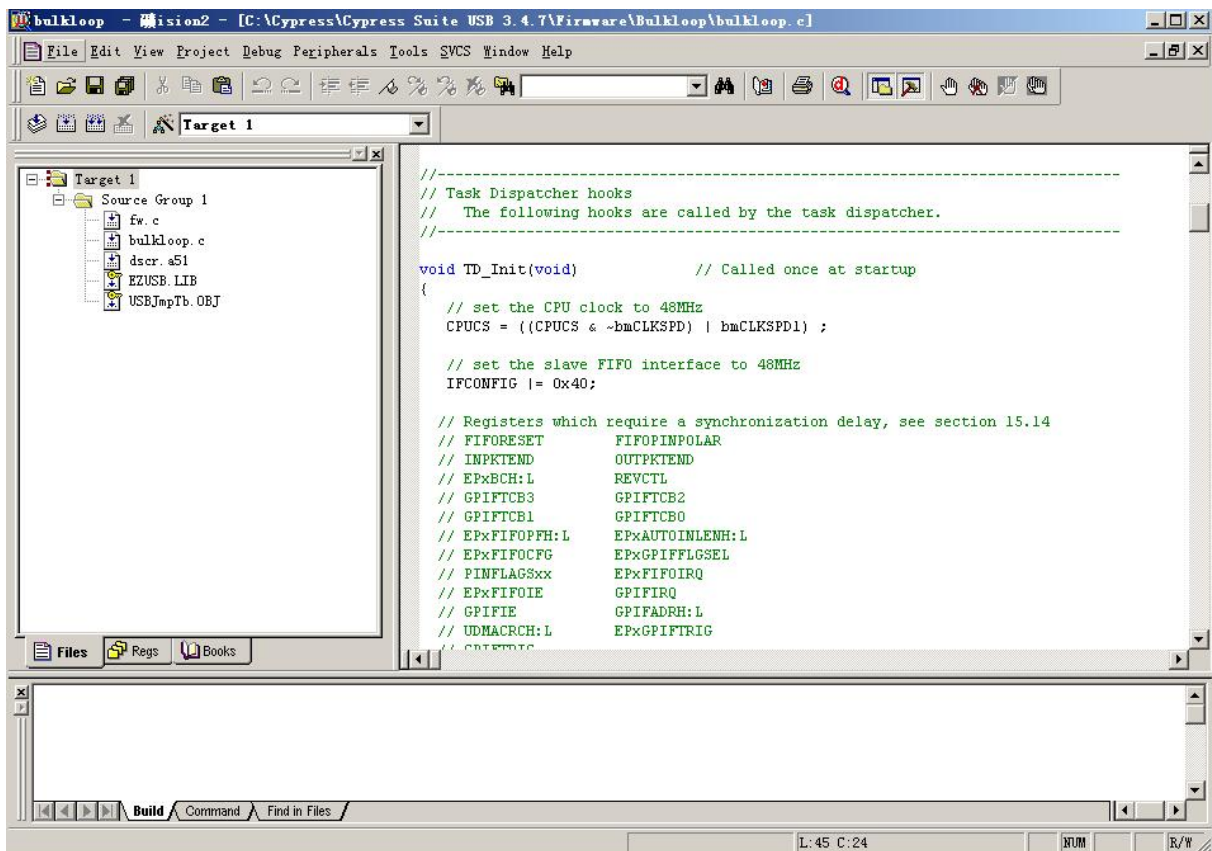
Keiluvision2



CySuiteUSB\_3\_4\_7\_B204.exe

## Part 2: KEIL C environment settings

- 1) Open the **Keil uVSION2** tool and click on the menu **Project->Open Project** to open the **Bulkloop.uV2** project in the **Cypress\USB\Examples\ FX2LP** directory.



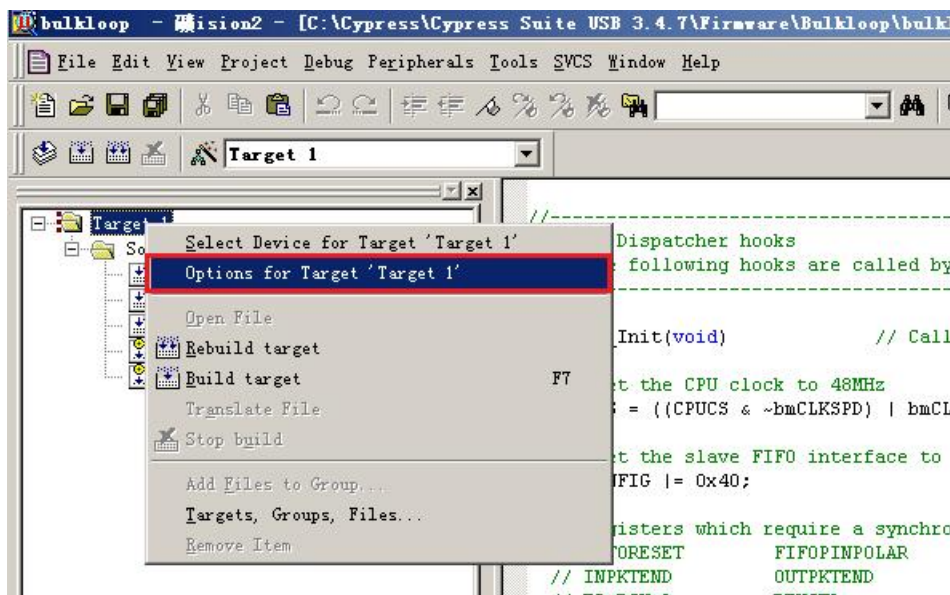
- 2) Set up the **KEIL** compilation environment as shown below, and click on the **Project** menu to select **File Extensions, Books and Environment**.



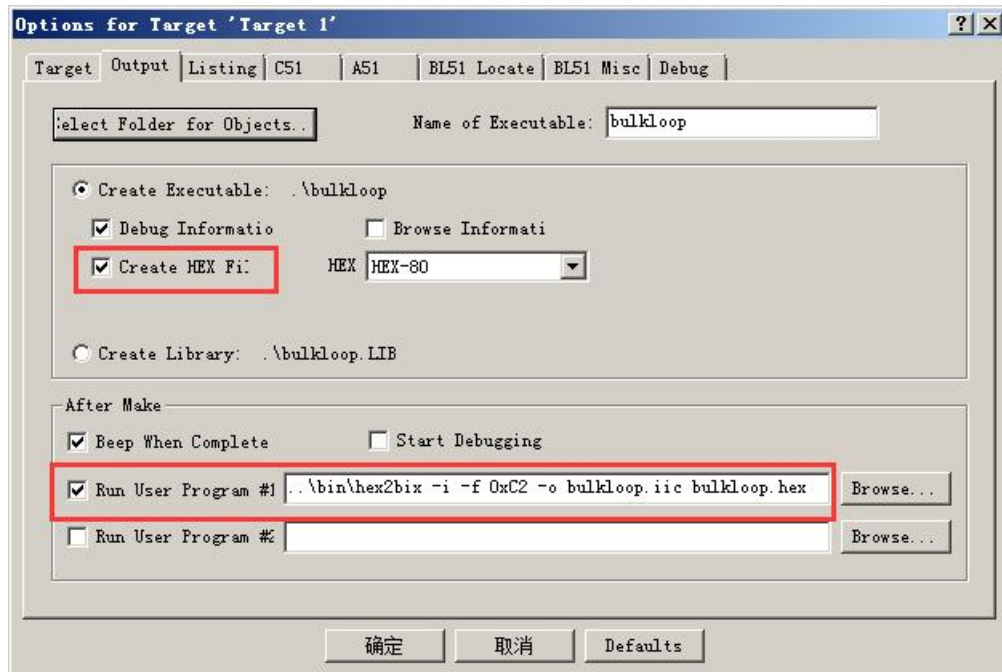
According to the software installation directory set the correct path, here **Cypress** software and **KEIL C** are installed in the root directory of the C drive, so the settings do not need to be changed.



3) Right click **Target 1** and select **Options for Target 'Target 1'** to open the compiler settings interface.

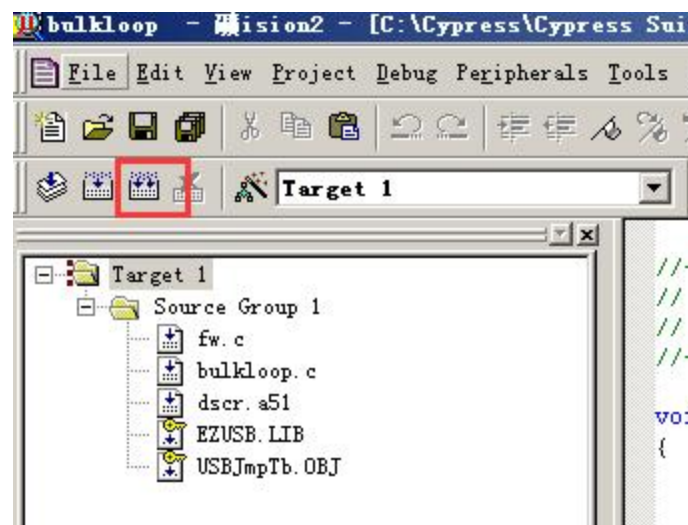


In the Output interface, choose to generate HEX files and IIC files (selected by default). The HEX file is the firmware program downloaded to the internal RAM of CY7C68013, and the IIC file is used to download the firmware program to the external EEPROM.



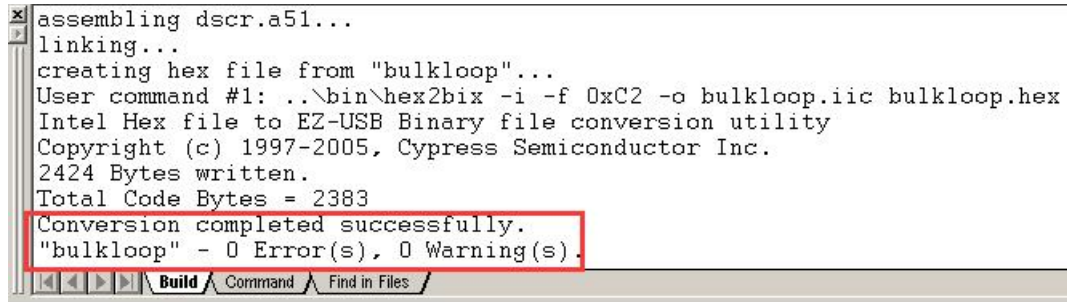
## Part 3: KEIL Project Compilation

Click the **Build All** button to start compiling, and the **Bulkloop.hex** and **Bulkloop.IIC** files will be generated after compilation.



Compiled successfully





```
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).
```

## Part 4: Bulkloop Project File Description

Under the project, there are the following files: Among them, USBJump.OBJ, EZUSB.LIB is basically added for each project, some interrupt vector table, EZUSB function library, etc., will not be introduced. Now focus on the following three files:

### 1) fw.c:

This file is the root of the entire USB firmware (short for FirmWare). Communication on the USB protocol is done here, including power-up enumeration, re-enumeration, wake-up, and calling the user's own programs and control commands. Basically, if you don't have to, try not to move the contents of this file, and don't write any code of your own.

### 2) bulkloop.c

This is the user's code writing file (original name: periph.c). All our code is written in this file. Cypress has already set up the architecture for us.

Void TD\_Init(void): This function will only be called once after USB is started. Add your own initialization code in this function, which is to be processed before transferring data, such as IO port configuration, clock, endpoint, FIFO selection and so on. Let's look at the initialization of the bulkloop, which initialization is done before the USB in, out transfer starts:

Void TD\_Poll(void): Poll Chinese means scheduling. This function is the user scheduler. USB will call this function repeatedly when it is idle, so we put the code that we need to execute repeatedly. For example, in the bulkloop, it



implements the repeated reception of the host computer data from the endpoint 2 and then to the endpoint 6, and then from the endpoint 6 to the host computer (the same as the 4, 8 endpoints).

**BOOL DR\_VendorCmnd(void):** This function is the writing place of the custom command code. Our Vendor command will be written here, and the fw.c firmware will automatically call our code.

**Void ISR\_Ep0in(void) interrupt 0~void ISR\_Ep8inout(void) interrupt 0:** These functions are used when the interrupt is transmitted using the endpoint interrupt, and are rarely used.

The above are several functions that are often used; others are basically not commonly used

### **3) dscr.51**

This file is a USB descriptor file that includes device descriptors, interface descriptors, endpoint descriptors, strings, and more. The English inside is very detailed, I will not introduce it. This file does not have to be changed when you first get started.

### **4) Other included header files**

**fx2.h:** predefined, macro and function declarations

**fx2regs.h:** 68013: Register address definition for 68013

**syncdly.h:** Synchronization delay. A function SYNCDELAY that is often called in other files is defined here.

**intrins.h:** C51 some data types and function definitions.

For the specific functions of **Bulkloop**, you can read the **readme.txt** file in the **bulkloop** folder, which tells us the main functions of this firmware.

## **Part 5: Project Modification**

For the CY7C68013A firmware program of the ALINXFPGA development board, we only need to modify the TD\_Init function and the TD\_Poll function in

the bulkloop.c file in the bulkloop project.

1) The TD\_Init function is modified as follows:

```
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 - invalid      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;
}
```

The dual function IO of the CY7C68013 is set to the Slave FIFO mode in the initialization function TD\_Init, the FLAGA pin is used to indicate the empty flag of the EP2 port, the FLAGB is used to indicate the full flag of the EP2 port, and the FLAGC is used to indicate the full flag of the EP6 port. In addition, ports 2 and 4 are configured as Bulk outputs, and ports 6 and 8 are configured as Bulk inputs. Please refer to Chapter 7 for a description of the registers.

- 2) Mask out all the contents of the TD\_Poll function, otherwise the USB firmware will automatically read the data in the EP2 FIFO into the EP6 FIFO, and read the data in the EP4 FIFO into the EP8 FIFO. Because the Loop function is implemented in the usb\_test.v program of the FPGA.

```

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 );
        APTRL1 = LSB( &EP2FIFOBUF );

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

        count = (EP2BCH << 8) + EP2BCL;

        // loop EP2OUT buffer data to EP6IN
        for( i = 0x0000; i < count; i++ )
        {
            // setup to transfer EP2OUT buffer to EP6IN buffer using AUTOPOINTER(s)
            EXTAUTODAT2 = EXTAUTODAT1;
        }
        EP6BCH = EP2BCH;
        SYNCDELAY;
        EP6BCL = EP2BCL;          // arm EP6IN
        SYNCDELAY;
        EP2BCL = 0x80;           // re(arm) EP2OUT
      }
    }

    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 );
        APTRL1 = LSB( &EP4FIFOBUF );

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

        count = (EP4BCH << 8) + EP4BCL;

        // loop EP4OUT buffer data to EP8IN
        for( i = 0x0000; i < count; i++ )
        {
            // setup to transfer EP4OUT buffer to EP8IN buffer using AUTOPOINTER(s)
            EXTAUTODAT2 = EXTAUTODAT1;
        }
        EP8BCH = EP4BCH;
        SYNCDELAY;
        EP8BCL = EP4BCL;          // arm EP8IN
        SYNCDELAY;
        EP4BCL = 0x80;           // re(arm) EP4OUT
      }
    }
  }
}

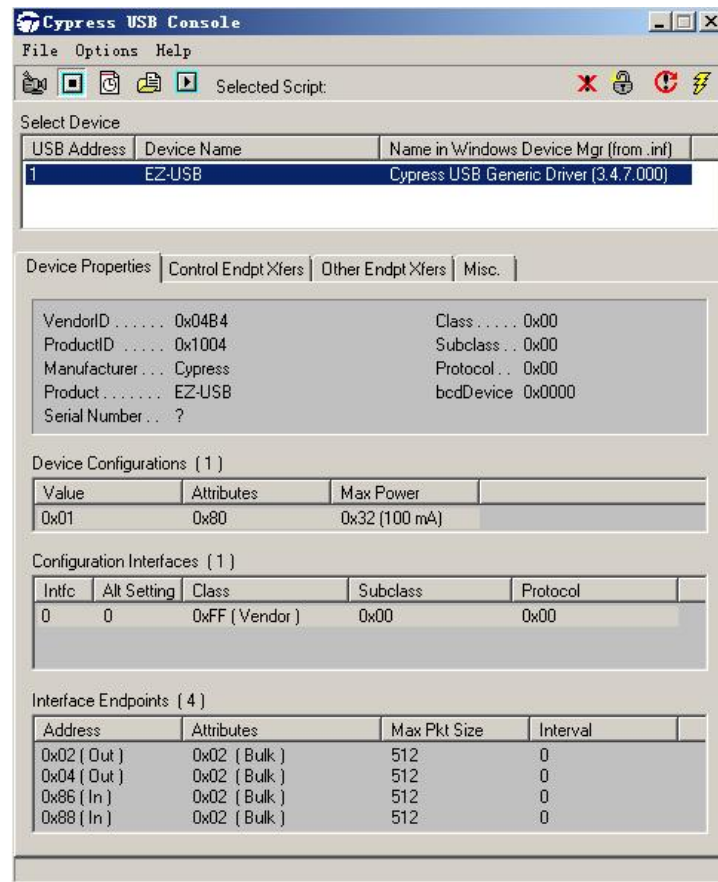
```

Save the project after the modification is complete and recompile the generated .hex file and .iic file.

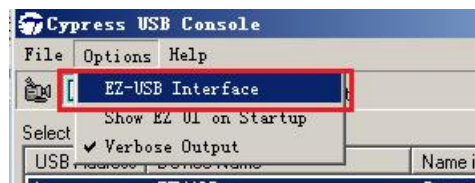
## Part 6: Firmware Download

Next we will download the generated bulkloop.iic file to the configuration EEPROM of the CY7C68013 on the development board. Connect the USB port of the PC and the USB port of the development board, and open the Cypress USB Console tool. At this time, we can see that the PC has detected the USB device of the development board.

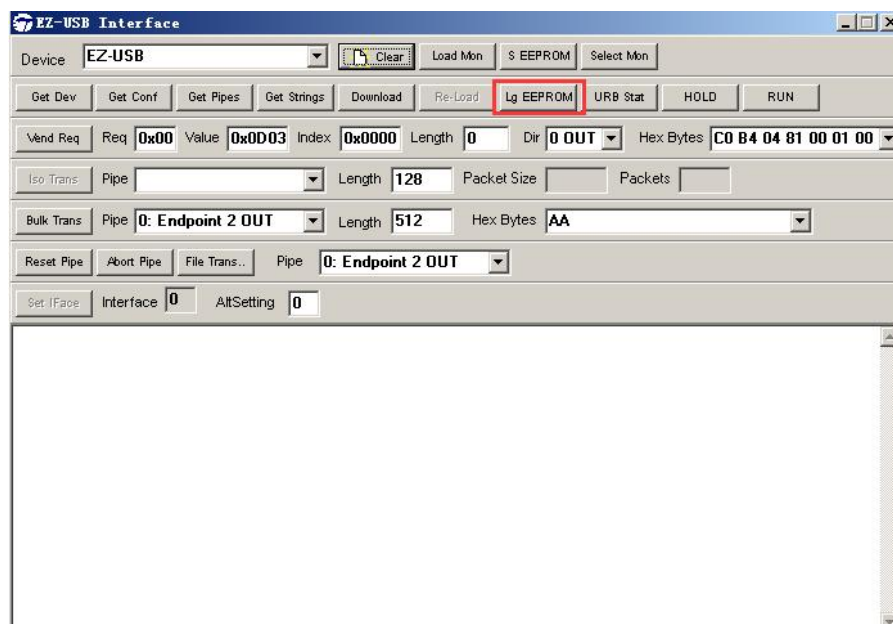




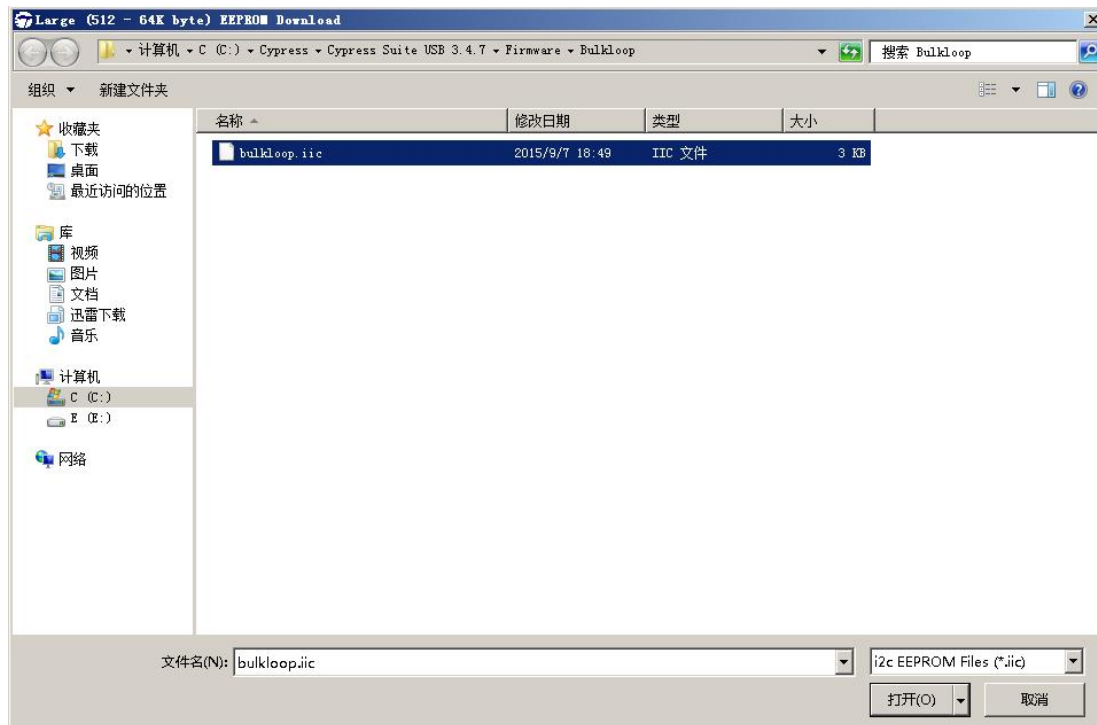
Select “Options->EZ-USB Interface”



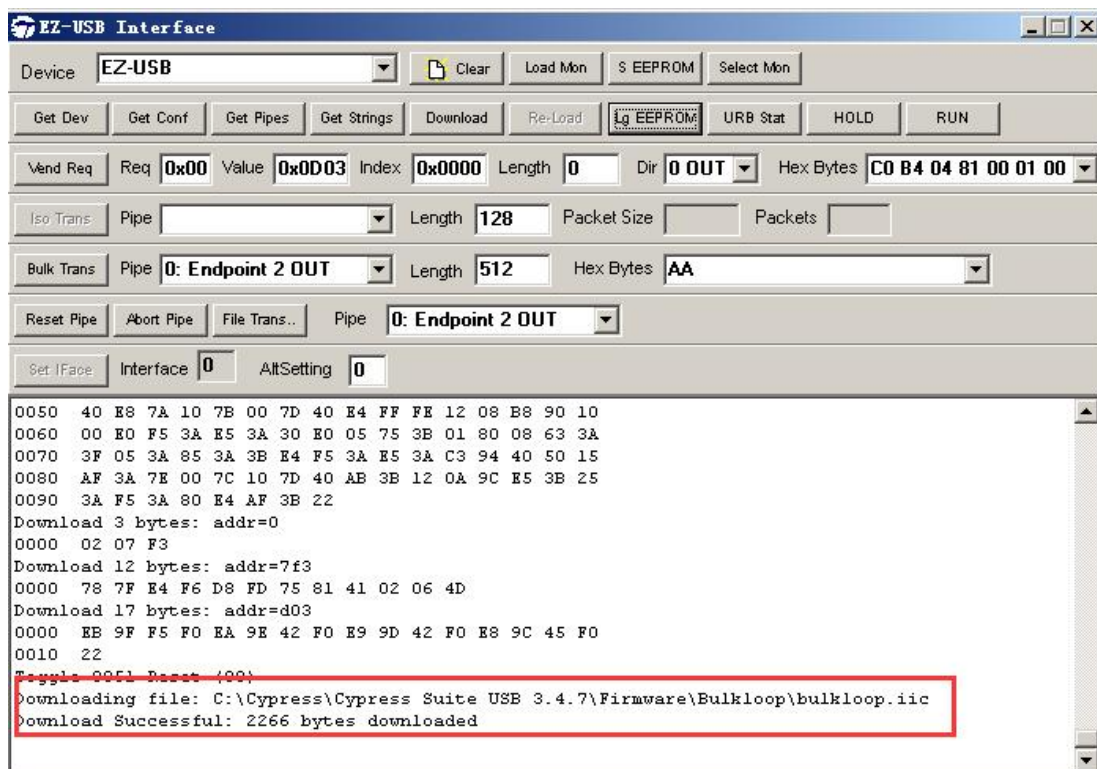
Select the **Lg EEPROM** button in the **EZ-USB Interface** interface.



Select the `bulkloop.iic` file you just created to open it.



The following prompt appears to indicate that the program has been downloaded to the EEPROM.



After the development board is powered on, the downloaded USB firmware will take effect.



## Part 7: CY7C68013 Partial Register Description

### 1. System configuration register

#### 1.1: CPU control and status register

CPUS, PU Control and Status Register

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 Read and write port C generates RD# and WR# signals, 0 does not generate read/write signals.

**CLKSPD1, CLKSPD0:** CPU clock setting, the default setting is 12M, the setting is shown in Table 7-1.

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

Table 7-1: Clock Configuration

CLKINV: Clock state reversal

CLKOE: clock enable

#### 1.2: Interface configuration register

IFCONFIG, configuration register

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 external clock source, 1 internal clock source

3048MHZ: 0 IFCLK clock 30M, 1 IFCLK clock 48M

IFCLOKE: IFCLK clock output enable, 0 off, 1 on

IFCLKPOL: IFCLK signal inversion, 0 no inversion, 1 inversion

ASYNC: GPIF synchronous or asynchronous operation, 0 sync, 1 asynchronous

GSTATE: GPIF status output enable, 0 off, 1 enable, pin PE0, PE1, PE2 and GPIF status

GSTATE0, GSTATE1, GSTATE2, GPIF can be used during debugging.

IFCFG0, IFCFG1: Mode setting, mode determines the status of multiple pins, as shown in Table 7-2

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

Table 7-2: Module Configuration

### 1.3: Slave FIFO mode FLAGA/B/C/D pin configuration register

PINFLAGSAB, FIFO FLAGA and FLAGB configuration registers

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 and FLAGD configuration registers

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

In the Slave FIFO mode, the FLAGA to FLAGD pins are used to define the state of the endpoint FIFO, and FLAGx can be flexibly programmed, as shown

in Table 7-3.

FLAGx3	FLAGx2	FLAGx1	FLAGx0	Pin function
0	0	0	0	FLGA=PF, FLAGB=EF  FLGA=EF, FLAGD=EP2PF  (Except for FLAGD, the endpoint FIFOs  corresponding to FLGA and FLAGC are  determined by pin FIFO[0,1]. See Table 3.4 for  details)
0	0	0	1	Reserved
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

Description: PF indicates FIFO programming flag, EF indicates FIFO is empty, FF indicates FIFO is full

## 2. Port configuration

### 2.1: Port A configuration

#### PORTACFG

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

Set to 1 to enable port A multiplexed pin function

## 2.2: Port C configuration

### PORTCCFG

b7	b6	b5	b4	b3	b2	b1	b0
<b>GPIFA7</b>	<b>GPIFA6</b>	<b>GPIFA5</b>	<b>GPIFA4</b>	<b>GPIFA3</b>	<b>GPIFA2</b>	<b>GPIFA1</b>	<b>GPIFA0</b>
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: Port E configuration

### PORTECFG

b7	b6	b5	b4	b3	b2	b1	b0
<b>GPIFA8</b>	<b>T2EX</b>	<b>INT6</b>	<b>RXD1OUT</b>	<b>RXD0OUT</b>	<b>T2OUT</b>	<b>T1OUT</b>	<b>T0OUT</b>
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 mode signal valid register

### FIFOPINPOLAR

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

Used to set the Slave FIFO mode signal active level, 0 signal is active low,  
1 signal is active high

## 4. Breakpoint configuration register

### 4.1: Endpoint 1IN and 1OUT configurations

EP1OUTCFG, endpoint 1OUT configuration

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

EP1INCFG, endpoint 1IN configuration

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

Valid: 1 endpoint is valid, 0 endpoint is invalid

TYPE1, TYPE0: Endpoint type is shown in Table 7.5

TYPE 1	TYPE 0	Type
0	0	Invalid
0	1	Invalid
1	0	Batch (default)
1	1	Interrupt

Table 7.5: Endpoint 1IN and 1OUT type settings

## 4.2: Endpoints 2, 4, 6, 8 Configure

### EP2CFG, Endpoint 2 Configuration

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

### EP4CFG, Endpoint 4 Configuration

b7	b6	b5	b4	b3	b2	b1	b0
<b>VALID</b>	<b>DIR</b>	<b>TYPE1</b>	<b>TYPE0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>
R/W	R/W	R/W	R/W	R	R	R	R
1	0	1	0	0	0	0	0

### EP6CFG, Endpoint 6 Configuration

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

### EP8CFG, Endpoint 8 Configuration

b7	b6	b5	b4	b3	b2	b1	b0
<b>VALID</b>	<b>DIR</b>	<b>TYPE1</b>	<b>TYPE0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>
R/W	R/W	R/W	R/W	R	R	R	R
1	1	1	0	0	0	0	0

VALID: 0 Endpoint is invalid, 1 endpoint is valid

DIR: End direction, 0=OUT direction, 1=IN direction, default endpoint 2, 4 is IN, endpoint 6, 8 is OUT TYPE1, TYPE0

Endpoint type, see Table 7-6

TYPE 1	TYPE 0	Type
0	0	Invalid
0	1	Synchronize
1	0	Batch (default)
1	1	Interrupt

Table 7-6: Endpoint 2, 4, 6, 8 type

SIZE: buffer size (endpoint 2 and endpoint 6 only), 0=512 bytes, 1=1024 bytes

BUF1, BUF0: number of endpoint buffers (endpoint 2 and endpoint 6 only), see Table 7-7

BUF1	BUF0	Type
0	0	Four buffer
0	1	Invalid
1	0	Double buffer (default)
1	1	Three buffer

Table 7-7: Number of endpoint buffers

#### 4.3: Slave FIFO mode endpoints 2, 4, 6, 8 Configure

EP2FIFOCFG, EP4FIFOCFG, EP6FIFOCFG, EP8FIFOCF

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 end point full reduction 1

OEP1: OUT endpoint empty plus 1 enable

AUTOOUT: Auto OUT enable

AUTOIN: Automatic IN enable

ZEROLENIN: Zero-length IN endpoint packet submission enable

WORDWIDE: data width, 8 or 16 bits

**4.4:** Points 2, 4, 6 and 8 count low byte

EP2BCL, EP4BCL, EP6BCL, EP8BCL

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