《USB2.0》设计

至芯科技教研部

李昭

2017-7-9

联系 QQ:984530288

至芯科技官网：www.zxopen.com

至芯科技技术论坛：www.fpgaw.com

至芯科技淘宝网址：
https://shop101836044.taobao.com/?spm=a230r.7195193.1997079397.2.9gJ436

至芯科技腾讯课堂：
https://ke.qq.com/course/list/%E8%87%B3%E8%8A%AF%E7%A7%91%E6%8A%80

至芯科技-fpag 交流群(QQ)：282124839

至芯科技 fpga 就业班火爆招生中，全国统一咨询热线：400-6810708

《USB2.0》设计

设计背景：

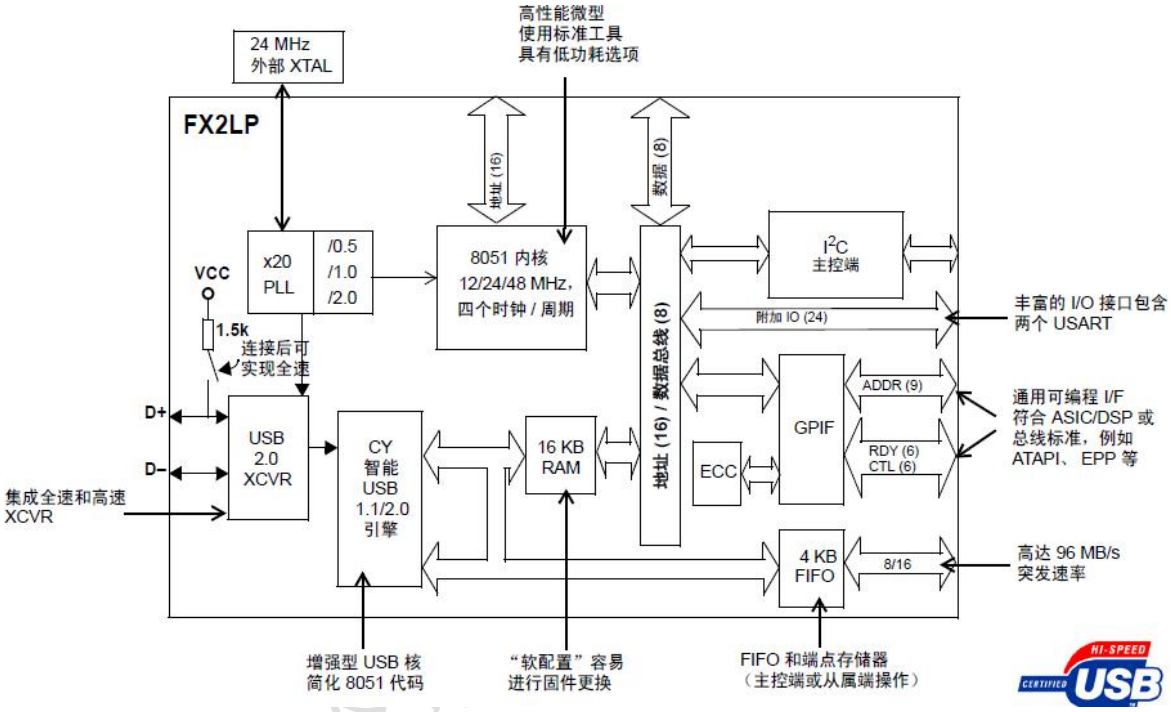
USB(Universal Serial Bus2.0，通用串行总线)是一种应用在计算机领域的新型接口技术。USB 接口具有传输速度更快，支持热插拔以及连接多个设备的特点。目前已经在各类外部设备中广泛的被采用。USB 接口有三种:USB1.1，USB2.0 和 USB3.0。理论上 USB1.1 的传输速度可以达到 12Mbps，而 USB2.0 则可以达到速度 480Mbps，并且可以向下兼容 USB1.1。

设计原理：

本次的设计主要设计我们用用的开发板是我们至芯出的第一代开发板,其中的 USB 芯片 C 是 ypress 的 FX2LP 系列中的 CY7C68013A 代，详细的介绍不多说，见 Cypress 的官网：在这里我就简化的给大家做一个简单介绍。

FX2 的设计架构如下图，内嵌 480MBit/s 的收发器，锁相环 PLL，串行接口引擎 SIE——集成了整个 USB 2.0 协议的物理层。为适应 USB 2.0 的 480MBit/s 的速率，FIFO 端点可配置成 2，3，4 个缓冲区。 配置用的是“软配置”——USB 固件可由 USB 总线下载，片上不需集成

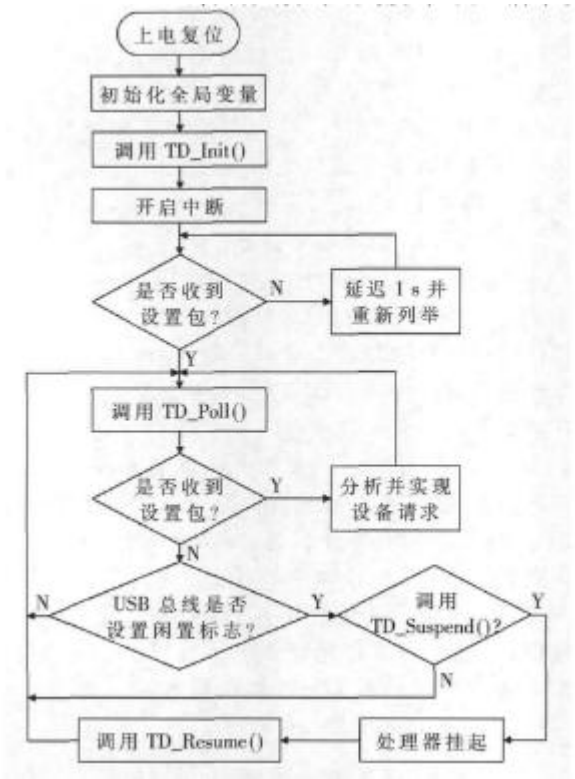
ROM。拥有四个 FIFO 接口，可工作在内部或外部时钟下。端点和 FIFO 接口的应用使外部逻辑和 USB 总线可高速连接。



基于 FX2LP 的 USB 开发，包括三部分：固件程序、驱动、上位机软件。

固件程序我们在 kiil 中写出来，然后配置到我们的芯片中，固件的开发对我们 FPGA 工程师来说是不用写的，是别的工程师配置好芯片我们拿来用的，其主要的配置过程如下图:先上电复位，然后初始化我们的寄存器变量，然后调用配置函数，打开中断后，判断是否接受到了我们的配置包，如果接收到了就调用 TD_POLL()函数，这个是函数是不停的执行扫描我们的端点等。然后判断我们的芯片是否挂

起，如果挂起就叫醒芯片，如果没有就一直调用 TD_POLL 函数，这样完成我们所需要的配置。



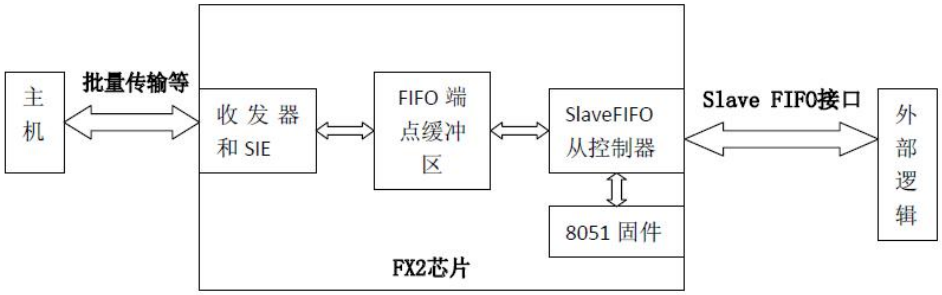
我们的项目是要把我们的 FX2 配置成从 FIFO 的模式， 配置为单片机工作时钟 24M，端点 2 输出，字节 1024，端点 6 输入，字节 1024，信号全设置为低电平有效等。我们的模块驱动时钟我们配置成内部输出时钟，也就是让 FX2 给我们的设计当做时钟源，输出一个最大的配置时钟 48M 的时钟。

在这边说一下，我们的 FX2 的数据存储区叫端点，有 512,1024 字节两个存储大小之分。

从 FIFO 的说明：

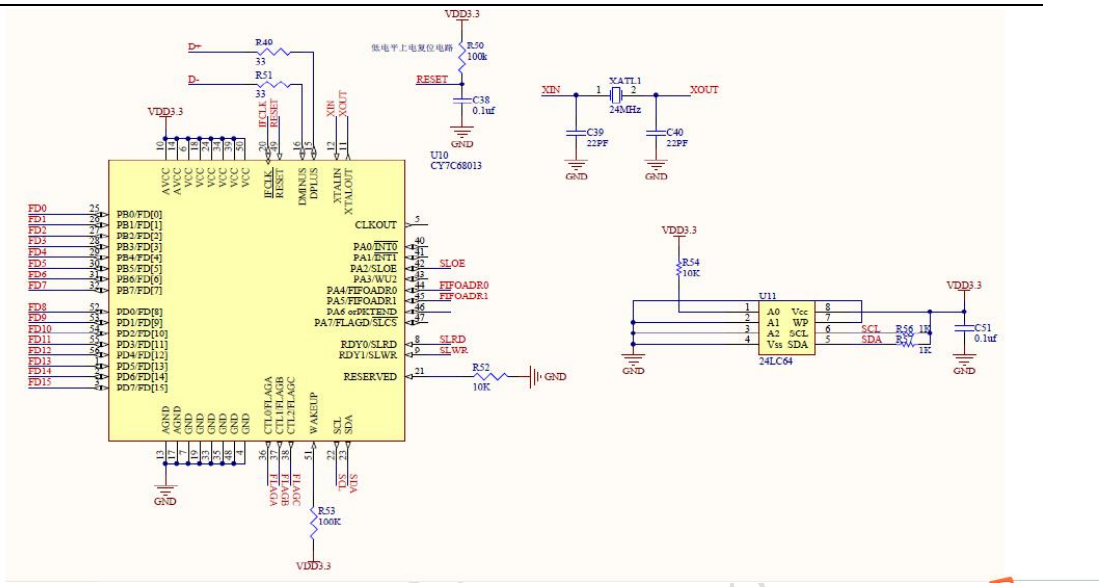
当有一个与FX2芯片相连的外部逻辑只需要利用FX2做为一个USB 2.0接口而实现与主机的高速通讯，而它本身又能够提供满足Slave FIFO要求的传输时序，可以做为Slave FIFO主控制器时，即可考虑用此传输方式。

Slave FIFO传输的示意图如下：



在这种方式下，FX2内嵌的8051固件的功能只是配置Slave FIFO相关的寄存器以及控制FX2何时工作在Slave FIFO模式下。一旦8051固件将相关的寄存器配置完毕，且使自身工作在Slave FIFO模式下后，外部逻辑（如FPGA）即可按照Slave FIFO的传输时序，高速与主机进行通讯，而在通讯过程中不需要8051固件的参与。

FX2系列的有3种封装方式，我们我的开发板用的是56引脚的封装方式的电路图，其电路图如下所示：



端口介绍：

IFCLK：FX2 输出的时钟，可做为通讯的同步时钟；

SLCS：FIFO 的片选信号，外部逻辑控制，当 SLCS 输出高时，不可进行数据传输；

SLOE：FIFO 输出使能，外部逻辑控制，当 SLOE 无效时，数据线不输出有效数据；

SLRD：FIFO 读信号，外部逻辑控制，同步读时，FIFO 指针在 SLRD 有效时的每个 IFCLK 的上升沿递增。

SLWR：FIFO 写信号，外部逻辑控制，同步写时，在 SLWR 有效时的每个 IFCLK 的上升沿时数据被写入，FIFO 指针递增

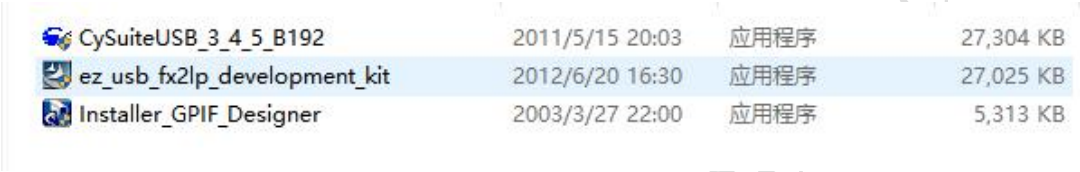
FD[15:0]：数据线；

FIFOADR[1:0]：选择四个 FIFO 端点的地址线，外部逻辑控制。

FLAGA,B,C 端点的空满标志位

我们的开发驱动大家可以在网上找，然后根据自己系统装上合适的驱动，或者在我们的至芯论坛上搜 EZ-USB，就可以看到我们老师发的帖子来讲解驱动的安装。

我们的上位机软件用的是官方的开发工具，只有如下的安装包然后安装第一个和第二个就好了。



设计代码：

读模块：

```
0 module usb_rd(pi_clk, pi_rst_n, pi_usb_flagb, pi_usb_flagc,
pio_usb_data,
1   po_usb_oe_n, po_usb_rd_n, po_usb_address, po_usb_wr_n, led);
2
3   input pi_clk;
4   input pi_rst_n;
5   input pi_usb_flagb;                          //端点 2 标志信号
6   input pi_usb_flagc;                        //端点 6 标志信号
7   inout [15:0]  pio_usb_data;                 //输入输出端口
8
9   output reg po_usb_oe_n;                     //读标志信号
10  output reg po_usb_rd_n;                     //写使能
11  output reg po_usb_wr_n;                     //读使能
12  output reg [1:0] po_usb_address;             //端点地址选择
13  output reg led;                               //接收标志正确指示灯
14
15  reg [15:0] temp_data;
16  reg [9:0] count;
17  reg [2:0] state;
18
```

```
19  assign pio_usb_data = (state == 10) ? 1 : 16'hzzzz;      //读数
    据，可以一直释放数据总线的控制权
20
21  always @ (posedge pi_clk or negedge pi_rst_n)
22      if(!pi_rst_n)
23          begin
24              state <= 0;
25              po_usb_oe_n <= 1;
26              po_usb_rd_n   <= 1;
27              count <= 0;
28              po_usb_wr_n <= 1;
29              temp_data <= 0;
30          end
31      else
32          case (state)
33              0   :   state <= 1;
34
35              1   :   begin
36                          po_usb_address <= 2'b00;  //地址指向端点 2
37                          state <= 2;
38                      end
39
40              2   :   if(!pi_usb_flagb)         //判断端点 2 已经满
41                          begin
42                              po_usb_rd_n   <= 0;
43                              state <= 3;
44                              po_usb_oe_n <= 0;
45                          end
46                      else
47                          state <= 2;
48
49              3   :   begin
50                      if(count < 512 - 1)          //接收 1024 字节的数
    据
51                          begin
52                              count <= count + 1'b1;
53                          end
54                      else
55                          begin
56                              count <= 0;
```

```
57                          state <= 4;
58                      end
59                  if (count == 2)
60                      begin
61                          temp_data <= pio_usb_data;
62                      end
63                  end
64
65              4   :   begin
66                      po_usb_rd_n   <= 1;
67                      po_usb_oe_n <= 1;
68                      state <= 0;
69                  end
70
71              default:   state <= 0;
72          endcase
73
74  always @ (*)
75      if(!pi_rst_n)
76          led <= 1;
77      else if (temp_data == 16'h33ff)  //判断我们接收数据是否正确
78          led <= 0;
79
80  endmodule
```

写模块：

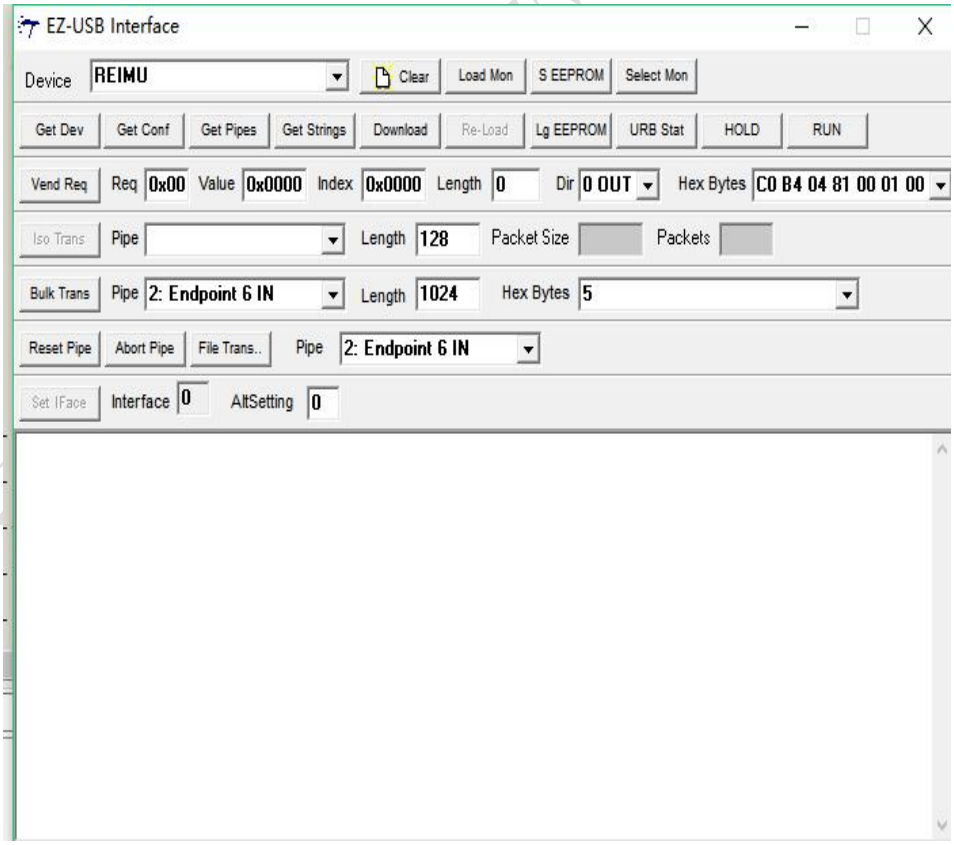
```
0  module usb_wr(pi_clk, pi_rst_n, pi_usb_flagb, pi_usb_flagc,
pio_usb_data,
1    po_usb_oe_n, po_usb_wr_n, po_usb_address, po_usb_rd_n);
2
3    input pi_clk;
4    input pi_rst_n;
5    input pi_usb_flagb;                    //端点 2 标志信号
6    input pi_usb_flagc;                    //端点 6 标志信号
7    inout [15:0] pio_usb_data;                    //输入输出端口
8
9    output reg po_usb_oe_n;                //读标志信号
10   output reg po_usb_wr_n;                //写使能
11   output reg po_usb_rd_n;                //读使能
```

```
12  output reg [1:0] po_usb_address;        //端点地址选择
13
14  reg [15:0] temp_data;
15  reg [2:0] state;
16
17  //在状态的 3，拿回数据总线控制全，给写入数据
18  assign pio_usb_data = (state == 3) ? temp_data : 16'hzzzz;
19
20  always @ (posedge pi_clk or negedge pi_rst_n)
21      if(!pi_rst_n)
22          begin
23              state <= 0;
24              po_usb_oe_n <= 1;
25              po_usb_wr_n   <= 1;
26              temp_data <= 0;
27              po_usb_rd_n <= 1;
28          end
29      else
30          case (state)
31              0   :   state <= 1;
32
33              1   :   begin
34                          po_usb_address <= 2'b10;  //地址指向端点 6
35                          state <= 2;
36                      end
37
38              2   :   if(!pi_usb_flagc)         //判断端点 6 已经空
39                          begin
40                              po_usb_wr_n   <= 0;
41                              state <= 3;
42                          end
43                      else
44                          state <= 2;
45
46              3   :   if(temp_data < 256 - 1)             //发送 1024
    字节的数据
47                          temp_data <= temp_data + 1'b1;
48                      else
49                          begin
50                              temp_data <= 0;
```

```
51                          state <= 4;
52                      end
53
54              4   :   begin
55                          po_usb_wr_n    <= 1;
56                          state <= 0;
57                      end
58
59              default:    state <= 0;
60          endcase
61
62  endmodule
```

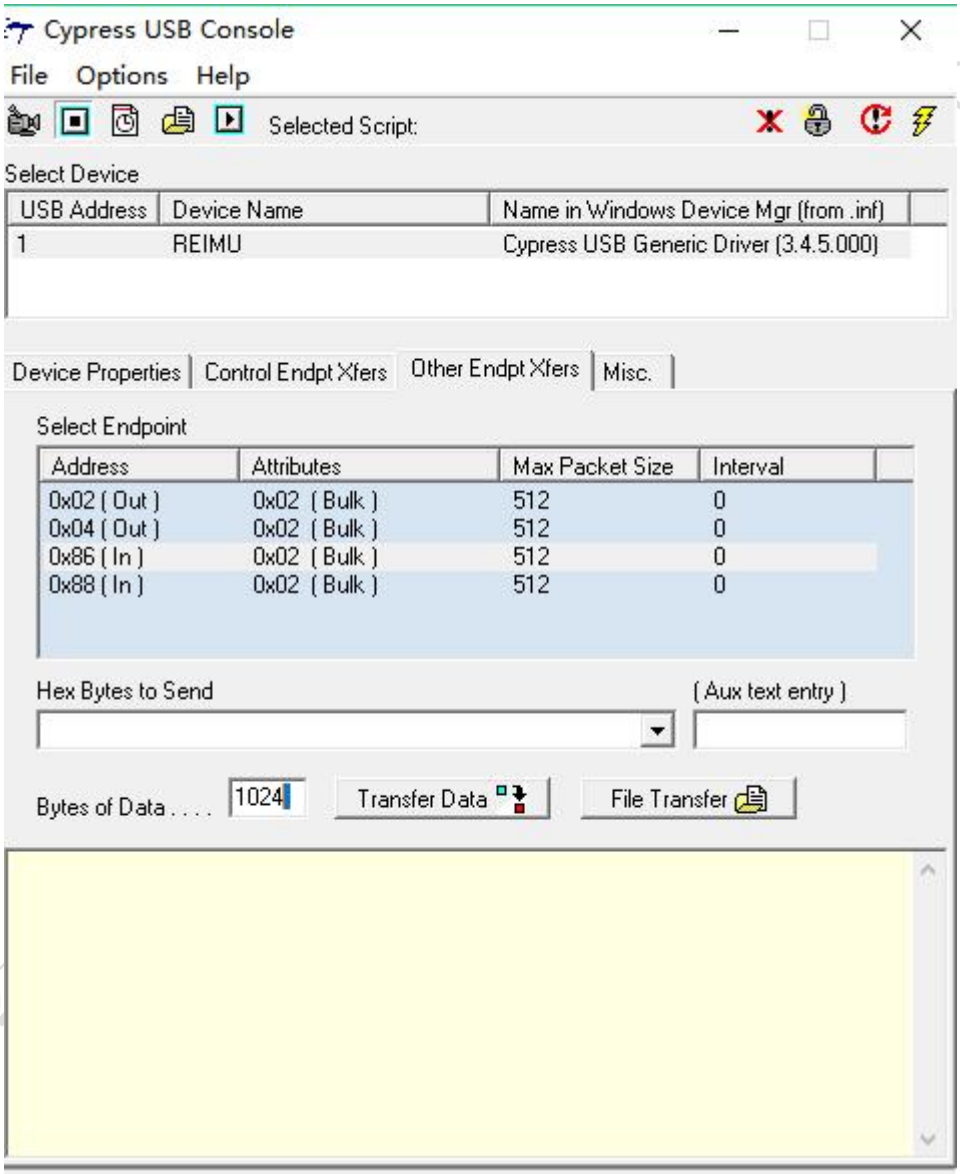
上位机测试：

我们安装好驱动和下载的上位机软件，然后在下面的界面中，点击”LGEEPROM”按钮，下载我们写好的的.IIC 固件



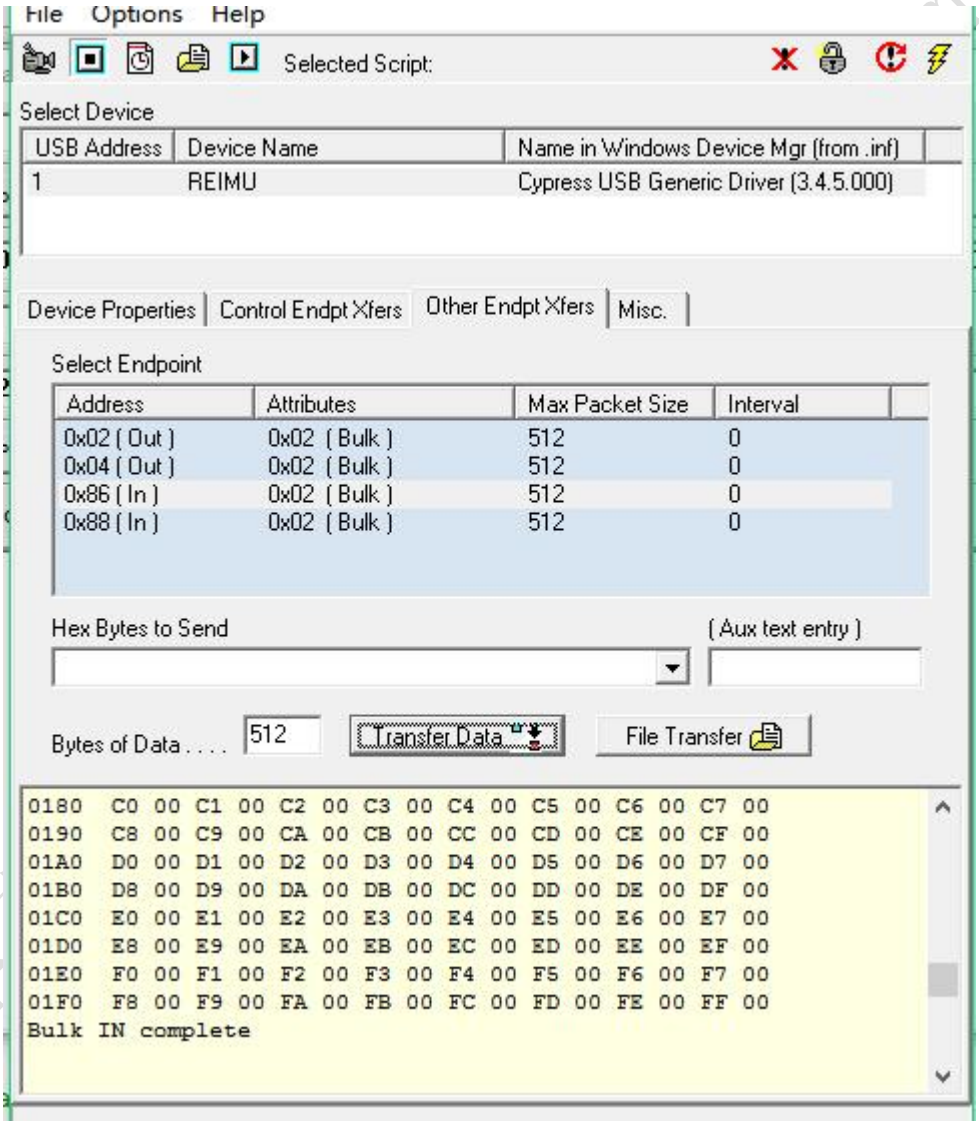
然后在下面的页面中会出现先选择 other endpt xfers 选项

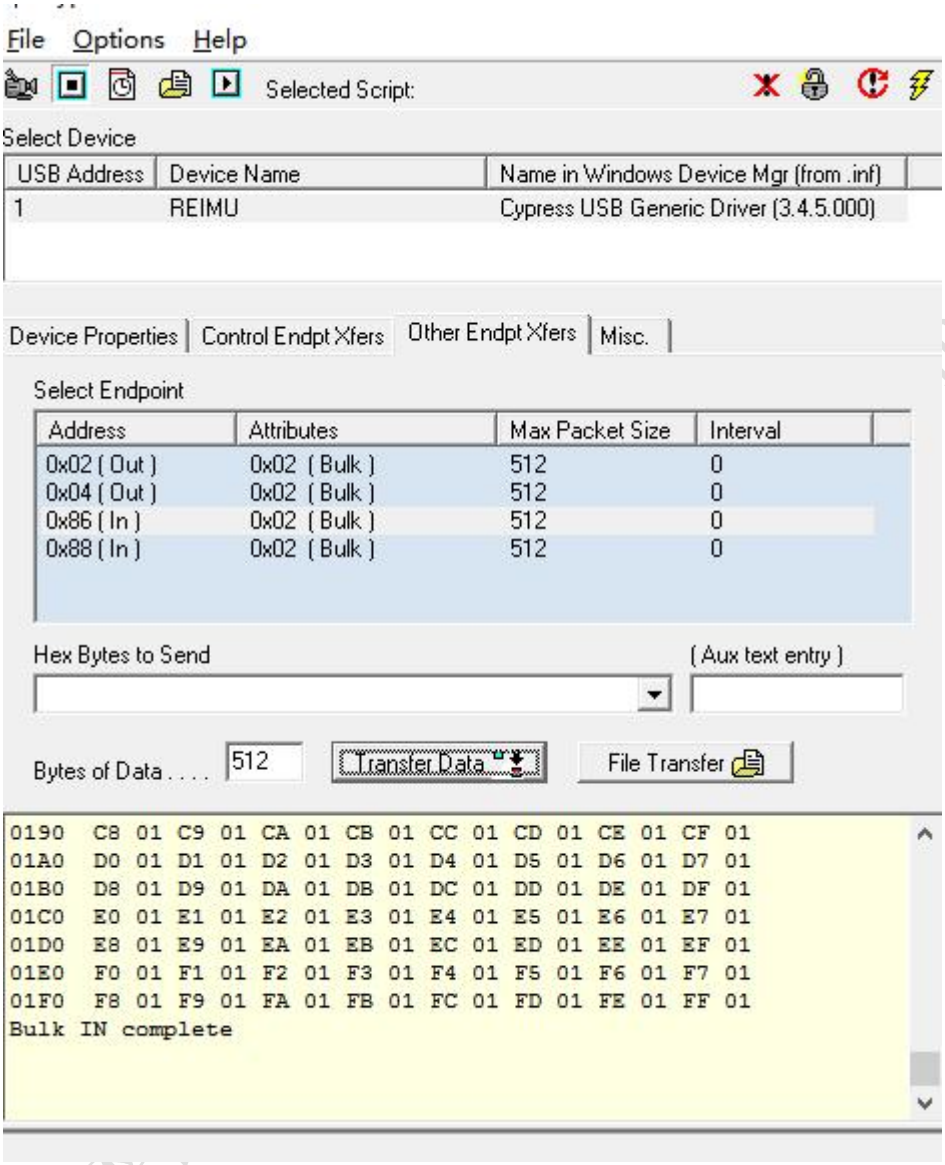
会出现我们的 4 个端点，然后我们选择写入的端点或者读的端点执行读写操作



写的端点是 6 端点，我们选择这个端点，我们的写入端点是 1024 个字节，我设置的是 512 字节，也就是写入 2 次就可以写满了，

如下图，和我们代码中写入数据值是一样的。




File Options Help
Selected Script:
Select Device
USB Address
Device Name
Name in Windows Device Mgr (from .inf)
1
REIMU
Cypress USB Generic Driver (3.4.5.000)
Device Properties
Control Endpt Xfers
Other Endpt Xfers
Misc.
Select Endpoint
Address
Attributes
Max Packet Size
Interval
0x02 (Out)
0x02 (Bulk)
512
0
0x04 (Out)
0x02 (Bulk)
512
0
0x86 (In)
0x02 (Bulk)
512
0
0x88 (In)
0x02 (Bulk)
512
0
Hex Bytes to Send
(Aux text entry)
Bytes of Data . . . .
512
Transfer Data
File Transfer
0190 C8 01 C9 01 CA 01 CB 01 CC 01 CD 01 CE 01 CF 01
01A0 D0 01 D1 01 D2 01 D3 01 D4 01 D5 01 D6 01 D7 01
01B0 D8 01 D9 01 DA 01 DB 01 DC 01 DD 01 DE 01 DF 01
01C0 E0 01 E1 01 E2 01 E3 01 E4 01 E5 01 E6 01 E7 01
01D0 E8 01 E9 01 EA 01 EB 01 EC 01 ED 01 EE 01 EF 01
01E0 F0 01 F1 01 F2 01 F3 01 F4 01 F5 01 F6 01 F7 01
01F0 F8 01 F9 01 FA 01 FB 01 FC 01 FD 01 FE 01 FF 01
Bulk IN complete

读操作也就是要读我们的端点 2，我们先要给端点一个数，然后才能读我们的端点，我们写入我们图中显示的数，因为我们设计的是读出的数如果第三个数位 33ff 就让我们的灯亮，值得一说的是，我们上位机显示的时候是把低位显示到了前面，高位显示到了后面，我们一个包是 1024 字节，后面的数自动补零，读出数据后可以看到我们的 led 灯亮，验证出我们的设计正确。