USB 通信

1 初识 USB

1.1 简介

USB（UniversalSerialBus）是一种支持热插拔的高速串行传输总线，它使用差分信号来传输数据。在 USB1.0 和 USB1.1 版本中，只支持 1.5Mb/s 的低速模式和 12Mb/s 的全速模式，在 USB2.0 和 USB3.0 中，又分别加入了 480Mb/s 的高速模式和 5Gb/s 的超高速模式。USB2.0 被设计成为向下兼容的模式，当有全速（USB1.1）或者低速（USB1.0）设备连接到高速（USB2.0）主机时，主机可以通过分离传输来支持它们。一条 USB 总线上，可达到的最高传输速度等级由该总线上最慢的“设备”决定。

USB 是一种轮询总线，由 Host 发起所有的数据传输。大部分总线传输包含 3 个包（packet）。每个传输都由 Host 先发出令牌包（TokenPacket），明确传输类型、传输方向、USB 设备地址和端点号。对应地址的 USB 设备接收并解析包。一次传输可以由 Host 发向设备，也可以由设备发送至 Host，方向由令牌包说明。传输中的数据过程是可选的，即有的传输没有数据过程。在低速/全速设备中，一次传输由 4 个包组成。

1.2 USB 通信流程

从 USB 系统角度而言，一个逻辑上的 USB 设备是一个端点的集合。分组的端点构成一个接口。USB 系统通过默认控制管道来管理设备。客户端软件使用管道来管理接口，通过主机上的 Buffer 和 USB 设备上的端点来请求数据。主机控制器打包数据并将数据包发送出去，如图 1.2 所示。

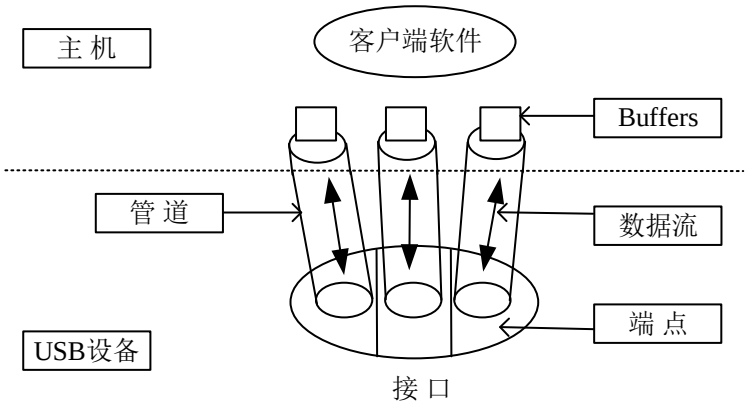

图 1.2 USB 数据通信流程

端点（Endpoint）是 USB 设备上可被独立识别的端口，是 Host 和 Device 通讯流的逻辑终点，是 USB 设备中可以进行数据收发的最小单元。一系列相互独立的端点在一起构成了 USB 逻辑设备。当设备连入 USB 总线时会被分配一个唯一的地址，设备上每一个端点有唯一的端点号。设备可决定每个端点的数据传输方向（输入或输出）。因此，使得每个端点可被唯一寻址。每个端点的属性包括总线访问频率、带宽、端点号、最大包长度、传输类型和数据传输方向。

每个 USB 设备必须要有一个端点 0，其作用为对设备枚举和对设备进行一些基本的控制功能，端点 0 也被称为控制端点。并且它与其他的端点还有一个不同之处在于端点 0 的数据传输方向是双向的，即端点 0 既可以给主机发送数据，也可以接收主机发送过来的数据，而其它端点均为单向。

除了控制端点以外，每个 USB 设备允许有一个或多个非 0 端点。低速设备最多只有两个非 0 端点。高速和全速设备最多支持 15 组端点。除了端点 0，其余的端点在设备配置之前不能与主机通信，只有向主机报告这些端点的特性并被确认后才能被激活。

管道（Pipe）是主机和设备端点之间数据传输的模型，共有两种类型的管道：无格式的流管道（StreamPipe）和有格式的信息管道（MessagePipe）。

流管道：数据从流通道一端流进的顺序与它们从流通道另一端流出时的顺序是一样的（先进先出），并且流通道中的通信流总是单向的。

信息管道：信息管道与端点的关系同流通道与端点的关系是不同的。首先，主机向 USB 设备发出一个请求;接着，就是数据的传送;最后，是一个状态阶段（这部分即一次命令请求的过程）。为了能够容纳请求/数据/状态的变化，信息管道要求数据有一个格式，此格式保证了命令能够被可靠地传送和确认。信息管道允许双方向的信息流。任何 USB 设备一旦上电就存在一个信息管道，即默认的控制管道，USB 主机通过该管道来获取设备的描述、配置、状态，并对设备进行配置。

USB 系统中的数据传输，宏观看是在 Host 和 USB 功能设备之间进行。微观看是在应用软件的 Buffer 和 USB 功能设备的端点之间进行。一般来说端点都有 Buffer，可以认为 USB 通讯就是应用软件 Buffer 和设备端点 Buffer 之间的数据交换，交换的通道称为管道。通常需要多个管道来完成数据交换，因为同一管道只支持一种类型的数据传输。用在一起来对设备进行控制的若干管道称为设备的接口，这就是端点、管道和接口的关系。

USB 采用“令牌包”-“数据包”-“握手包”的传输机制，在令牌包中指定数据包去向或者来源的设备地址和端点，从而保证了只有一个设备对被广播的数据包/令牌包作出响应。握手包表示了传输的成功与否。

USB 采用轮询的广播机制传输数据，所有的传输都由主机发起，任何时刻整个 USB 体系内仅允许一个数据包的传输，即不同物理传输线上看到的数据包都是同一被广播的数据包。

1.3 传输类型

USB 的传输模式有 4 种，分别是控制传输（ControlTransfer）、中断传输（InterruptTransfer）、批量传输或叫块传输（BulkTransfer）、实时传输或叫同步传输（IsochronousTransfer）。

（1）控制传输

控制传输是一种可靠的双向传输，是最重要也是最复杂的。一次控制传输分为三个(或两个)阶段：建立(Setup)、数据(DATA)(可能没有)以及状态(Status)。每个阶段都由一次或多次(数据阶段)事务(Transaction)传输组成。在 USB 设备初次接到主机后，主机通过控制传输来交换信息、设备地址和读取设备的描述符，使得主机识别设备，并安装相应的驱动程序。控制传输是双向的传输，必须有 IN 和 OUT 两个方向上的特定端点号的控制端点来完成两个方向上的控制传输。

控制传输通过控制管道在应用软件和 Device 的控制端点之间进行，控制传输过程中传输的数据是有格式定义的，USB 设备或主机可根据格式定义解析获得的数据含义。其他三种传输类型都没有格式定义。控制传输对于最大包长度有固定的要求。对于高速设备该值为 64Byte，对于低速设备该值为 8Byte，全速设备可以是 8/16/32/64Byte。

最大包长度表征了一个端点单次接收/发送数据的能力，实际上反应的是该端点对应 Buffer 的大小。Buffer 越大，单次可接收/发送的数据包越大，反之亦反。当通过一个端点进行数据传输时，若数据的大小超过该端点的最大包长度时，需要将数据分成若干个数据包传输。并保证除最后一个包外，所有的包长度均等于该最大包长度。这也就是说如果一个端点收到/发送了一个长度小于最大包长度的包，即意味着数据传输结束。

控制传输在访问总线时也受到一些限制,如高速端点的控制传输不能占用超过 20%的微帧，全速和低速的则不能超过 10%。在一帧内如果有多余的未用时间，并且没有同步和中断传输，可以用来进行控制传输。

与批量传输相比，在流程上并没有多大区别，区别只在于该事务传输发生的端点不一样、支持的最大包长度不一样、优先级不一样等这样一些对用户来说透明的东西。

（2）中断传输

中断传输是一种轮询的传输方式，是一种单向的传输。Host 通过固定的间隔对中断端点进行查询，若有数据传输或可以接收数据则返回数据或发送数据。否则返回 NAK，表示尚未准备好。中断传输的延迟有保证，但并非实时传输，它是一种延迟有限的可靠传输，支持错误重传。对于高速/全速/低速端点，最大包长度分别可以达到 1024/64/8Bytes。高速中断传输不得占用超过 80%的微帧时间，全速和低速不得超过 90%。中断端点的轮询间隔由在端点描述符中定义，全速端点的轮询间隔可以是 1~255ms，低速端点为 10~255ms，高速端点为$(2^{interval}-1)\times125us$，其中 interval 取 1 到 16 之间的值。

主机在排定中断传输任务时，会根据对应中断端点描述符中指定的查询间隔发起中断传输。中断传输有较高的优先级，仅次于同步传输。同样中断传输也采用 PID 翻转的机制来保证收发端数据同步。

中断传输方式总是用于对设备的查询，以确定是否有数据需要传输。因此中断传输的方向总是从 USB 设备到主机。

除高速高带宽中断端点外，一个微帧内仅允许一次中断事务传输。高速高带宽端点最多可以在一个微帧内进行三次中断事务传输，传输高达 3072 字节的数据。

所谓单向传输，并不是说该传输只支持一个方向的传输。而是指在某个端点上该传输仅支持一个方向，或输出、或输入。如果需要在两个方向上进行某种单向传输，需要占用两个端点，分别配置成不同的方向，可以拥有相同的端点编号。

中断传输由 OUT 事务和 IN 事务构成，用于键盘、鼠标等 HID 设备的数据传输。中断传输在流程上除不支持 PING 之外，其他的跟批量传输是一样的。他们之间的区别也仅在于事务传输发生的端点不一样、支持的最大包长度不一样、优先级不一样等这样一些对用户来说透明的东西。

（3）批量传输

批量传输由 OUT 事务和 IN 事务构成，是一种可靠的单向传输，但延迟没有保证，它尽量利用可以利用的带宽来完成传输，适合数据量比较大的传输。低速 USB 设备不支持批量传输，高速批量端点的最大包长度为 512，全速批量端点的最大包长度可以为 8、16、32、64。用于传输大量数据，要求传输不能出错，但对时间没有要求，适用于打印机、存储设备等批量传输在访问 USB 总线时，相对其他传输类型具有最低的优先级，USB 主机总是优先安排其他类型的传输，当总线带宽有富余时才安排批量传输。高速的批量端点必须支持 PING 操作，向主机报告端点的状态。NYET 表示否定应答，没有准备好接收下一个数据包，ACK 表示肯定应答，已经准备好接收下一个数据包。它通过在硬件级执行“错误

检测”和“重传”来确保 Host 与 device 之间“准确无误”地传输数据，即可靠传输。一次事务由三种包组成：令牌包、数据包和握手包。

若数据量比较大，将采用多次批量事务传输来完成全部数据的传输，传输过程中数据包的 PID 按照 DATA0-DATA1-DATA0-…的顺序发送数据包，只有成功的事务传输才会导致 PID 翻转，也就是说发送段只有在接收到 ACK 后才会翻转 PID，发送下一个数据包，否则会重试本次事务传输。同样，若在接收端发现接收到的数据包不是按照此顺序翻转的，比如连续收到两个 DATA0，那么接收端认为第二个 DATA0 是前一个 DATA0 的重传。

此外，若成功则将错误次数计数器清 0，否则累加该计数器。USB 允许连续 3 次以下的传输错误，错误时会重试该传输，若成功则将错误次数计数器清零，否则累加该计数器。超过三次后，Host 认为该端点功能错误（STALL），放弃该端点的传输任务。一次批量传输由 1 次到多次批量事务传输组成。

（4）等时传输

等时传输是一种实时的、不可靠的传输，不支持错误重发机制。只有高速和全速端点支持等时传输，高速等时端点的最大包长度为 1024，低速的为 1023。由 OUT 事务和 IN 事务构成。有两个特殊地方：第一，在等时传输的 IN 和 OUT 事务中是没有返回包阶段的；第二，在数据包阶段所有的数据包都为 DATA0。等时传输由令牌包和数据包两种包组成。等时传输不支持“握手包”和“重传能力”，所以它是不可靠传输。等时传输适用于必须以固定速率抵达或在指定时刻抵达，可以容忍偶尔错误的数据上。实时传输一般用于麦克风、喇叭、UVCCamera 等设备。等时传输有最高的优先级除高速高带宽等时端点外，一个微帧内仅允许一次等时事务传输，高速高带宽端点最多可以在一个微帧内进行三次等时事务传输，传输高达 3072 字节的数据。全速等时传输不得占用超过 80%的帧时间，高速等时传输不得占用超过 90%的微帧时间。等时端点的访问也和中断端点一样,有固定的时间间隔限制。

2 如何使用 USB

2.1 FPGA 与 USB 通信

FPGA 与 USB 芯片通讯的接口原理图如图 2.1 所示，两者严格按照规定的 FIFO 握手协议进行数据传输。其中握手信号有：SLOE、SLRD、SLWR 为使能信号和读写控制信号；FLAGB 为空满的标志信号；IFCLK 提供 FPGA 写时钟信号；FIFOADR[1:0]为 USB 片内端点地址；FD[15:0]为 16 位传输数据。

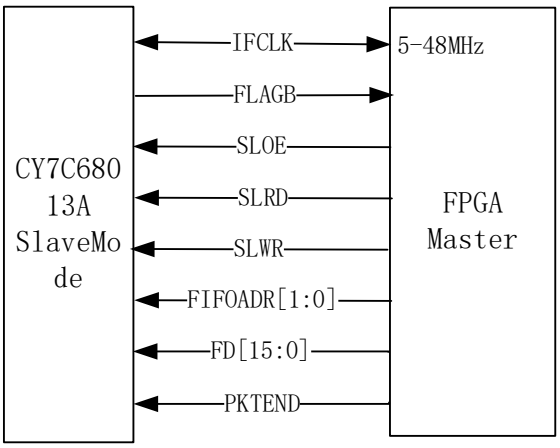

图 2.1 USB 接口原理图

其中 PKTEND、SLOE、SLWR、SLRD、FLAGB 的设置需要结合 FX2LP 固件代码中的引脚极性寄存器（FIFOPINPOLAR）的值。

为保证数据的准确传输，需在 FPGA 中进行 SlaveFIFO 状态机设计，SlaveFIFO 表示的是一种外部从属 FIFO，及通过片外的控制信号对片内的 RAM 进行存储操作。此处的通讯设计中，FPGA 与 USB 芯片就是属于一种从属关系，FPGA 提供数据与控制信号，控制 USB 芯片中的存储单元进行存储与提取。如图 2.2 所示为 SlaveFIFO 控制状态机（假定 FLAGB 等均低电平有效），在 FPGA 内部实现的硬件电路。IDLE 表示等待状态，当使能与端点地址已就绪，USB 芯片反馈为非空时，每个时钟进入一个 STATE 并进行一次 16 位有效数据的传输。

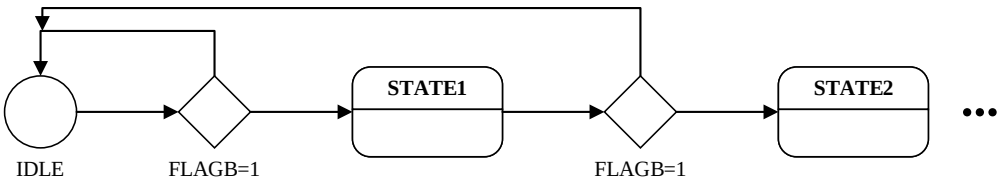

图 2.2 SlaveFIFO 控制状态机

参照官方实例，写出符合其固件代码的 FPGA 与 USB 通信代码，通过 CyConsole 软件可以检测 USB 芯片接收到 FPGA 输出的数据，判断数据是否成功上传至上位机。如图 2.3 所示为一个 USB 数据包，包中存有 512 字节的数据（0-255、0-255 的累加数），说明数据已经成功从 FPGA 转送至 USB 接口。

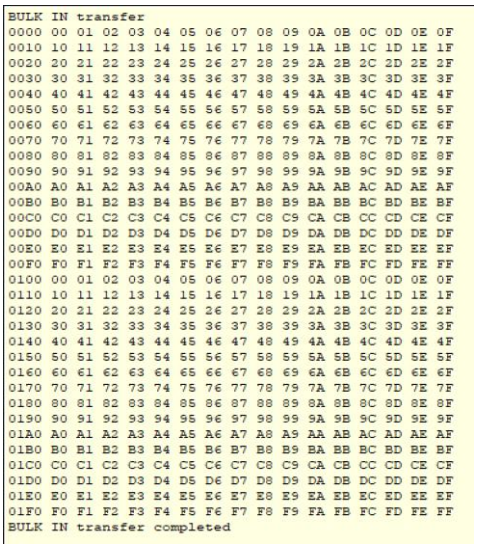

图 2.3 CyConsole 接收结果

2.2 USB 固件代码修改

USB 固件代码是 USB 通信三部曲之一，想要设计修改出符合自己要求的固件代码，需要了解其描述符、寄存器等重要参数。

2.2.1 常用描述符配置

USB 描述符也可以看作是 USB 设备的身份证明。常用的描述符有设备、配置、接口、端点和字符串。各个 USB 设备只有一个设备描述符。但是，一个设备可以有多种配置、接口、端点和字符串描述符。设备执行枚举时，终端阶段中有一个步是读取设备描述符，并选择需要使能的设备配置类型。每一次操作只能使能一种配置。

（1）设备描述符

设备描述符描述有关 USB 设备的一般信息。它包含全局适用于设备和所有设备配置的信息，如设备需要满足的 USB 规范、设备配置编号、设备支持的协议、供应商标识（VID，每个公司只能从 USB 实施者论坛获得唯一的 VID）、产品标识（PID，与数据包 ID 不同）、一个序列号（如果设备有）以及端点 0 的最大数据包大小等。

（2）配置描述符

配置描述符描述了关于特定设备配置的信息，如接口数量、供电方式和最大功耗等。描述符包含一个 bConfigurationValue 字段，该字段的值用作 SetConfiguration()请求的参数时，会使设备采用所描述的配置。描述符描述配置提供的接口数量，每个接口可以独立运行。配置完成后，设备可能会对配置进行有限的调整。如果某个特定接口具有备用设置，则可以在配置后选择备用设备。

（3）接口描述符

接口描述符描述配置中的特定接口，如端点数量、接口类别和协议等。一个配置提供一个或多个接口，每个接口具有零个或多个端点描述符，用于描述配置中的一组唯一端点。当配置支持多个接口时，特定接口的端点描述符将遵循 GetConfiguration()请求返回的数据中的接口描述符。接口描述符总是作为配置描述符的一部分返回。接口描述符不能通过 GetDescriptor()或 SetDescriptor()请求直接访问。

（4）端点描述符

在一个设备中所使用的全部端点都有自己的描述符，这些描述符会提供主机必须获取的端点信息，如端点方向、传输类型、最大数据包长度和轮询间隔等。一个端点描述符不能通过 GetDescriptor()或 SetDescriptor()请求直接访问，而是作为配置描述符的一部分返回。

（5）字符串描述符

字符串描述符是一种可选的描述符，它为用户提供了有关设备的可读信息。该描述符中所包含的信息显示了以下内容：设备名称、生产厂家、序列号或不同接口、配置的名称。如果设备没有使用字符串，必须将前面所述的所有描述符中的字符串附加字段的值设置为 00h。否则在枚举过程中，USB 主机会尝试去获取字符串描述符，如果没有，枚举就会失败。

2.2.2 常用寄存器配置

slavefifo 模式下常用寄存器如表 2.2 所示。

表 2.2 常用寄存器

CPUCS	FIFOPINPOLAR
PINFLAGSAB/CD	PORTACFG
IFCONFIG	EP2/4/6/8CFG
FIFORESET	EP2/4/6/8FIFOCFG
EP1OUTCFG	EP2/4/6/8AUTOINLENH/L
EP1INCFG	EP2/4/6/8ISOINPKTS

（1）CPUCS：CPU 控制和状态寄存器

CPUCS			CPU Control and Status				E600
b7	b6	b5	b4	b3	b2	b1	b0
0	**0**	**PORTCSTB**	**CLKSPD1**	**CLKSPD0**	**CLKINV**	**CLKOE**	**8051RES**
R	R	R/W	R/W	R/W	R/W	R/W	R
0	0	0	0	0	0	1	0

PORTCSTB：PORTC 访问产生 RD 和 WR 选通。

100 和 128 引脚 EZ-USB 封装有两个输出引脚 RD 和 WR，可用于同步 I/OPORTC 上的数据传输。当 PORTCSTB=1 时，启用此功能。对 PORTC 的任何读取都会激活 RD 选通，而对 PORTC 的任何写入都会激活 WR 选通。RD 和 WR 选通被置位两个 CLKOUT 周期；更新 PORTC 引脚后，WR 选通将置位两个 CLKOUT 周期。

如果设计使用 128 引脚 EZ-USB 并将片外存储器连接到地址和数据总线，则该位应设置为零。这是因为 RD 和 WR 引脚也是用于读写片外存储器的标准选通管，因此对 I/O 端口 C 的正常读/写会中断对该存储器的正常访问。

CLKSPD[1:0]：CPU 时钟速度。

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

这些位设置 CPU 时钟速度。在硬复位时，这些位默认为“00”（12MHz）。固件可以随时修改这些位

CLKINV：反转 CLKOUT 信号。0 不反转，1 反转。

CLKOE：驱动 CLKOUT 引脚。0 引脚悬空。1 引脚驱动。

8051RES：8051 重置。USB 主机向该位写入“1”以重置 8051，向该位写入“0”以运行 8051。只有 USB 主机可以写入该位（通过 0xA0 固件加载命令）。

（2）FIFOPINPOLAR：从设备 FIFO 接口引脚极性寄存器

FIFOPINPOLAR see Section 15.15			Slave FIFO Interface Pins Polarity				E609
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

SlaveFIFO 引脚极性设置均为 0 低有效，1 高有效。其中 PF 极性没有提供寄存器设置，为高有效。

（3）PINFLAGSAB/CD：从设备 FIFOFLAGA-FLAGD 引脚配置寄存器

PINFLAGSAB see Section 15.15			Slave FIFO FLAGA and FLAGB Pin Configuration				E602
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 see Section 15.15			Slave FIFO FLAGC and FLAGD Pin Configuration				E603
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	0	0	0	0	0	0	0

EZ-USB 有四个 FIFO 标志输出引脚，FLAGA、FLAGB、FLAGC 和 FLAGD。这些标志可以编程为使用每个 FIFO 的四个选择位来表示各种 FIFO 标志。PINFLAGSAB 寄存器控制 FLAGA 和 FLAGB 信号，PINFLAGSCD 寄存器控制 FLAGC 和 FLAGD 信号。所有四个标志的四位编码是相同的，如图 2.5 所示。在“FLAGx”符号中，“x”可以是 A、B、C 或 D。

FLAGx3	FLAGx2	FLAGx1	FLAGx0	Pin Function
0	0	0	0	FLAGA=PF, FLAGB=FF, FLAGC=EF, FLAGD=EP2PF (Actual FIFO is selected by FIFOADR[0,1] pins)
0	0	0	1	Reserved
0	0	1	0	
0	0	1	1	
0	1	0	0	EP2 PF
0	1	0	1	EP4 PF
0	1	1	0	EP6 PF
0	1	1	1	EP8 PF
1	0	0	0	EP2 EF
1	0	0	1	EP4 EF
1	0	1	0	EP6 EF
1	0	1	1	EP8 EF
1	1	0	0	EP2 FF
1	1	0	1	EP4 FF
1	1	1	0	EP6 FF
1	1	1	1	EP8 FF

图 2.5FIFO 标志引脚功能

注：FLAGD 默认为 EP2-PF。

（4）PORTACFG：I/OPORTA 备用配置寄存器

PORTACFG			I/O PORTA Alternate Configuration				E670
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

FLAGD：FlagD 备用配置。当 IFCFG[1:0]=11 时，将此位设置为“1”会将 PA7 引脚配置为 FLAGS，这是一个可编程的 FIFO 标志。

SLCS：$\overline{SLCS}$ 备用配置。如果 IFCFG[1:0]=11，将此位设置为“1”会将 PA7 引脚配置为 $\overline{SLCS}$，即从设备 FIFO 芯片选择。

INT[1:0]：备用配置启用中断。将这些位设置为“1”会将这些 PORTA 引脚配置为 INT1 或 INT0 引脚。

注：该寄存器的 b7、b6 都影响引脚 PA7。如果两个位都设置，则 FLAGD 优先。

（5）IFCONFIG：接口配置（端口、GPIF、从设备 FIFO）寄存器

IFCONFIG			Interface Configuration(Ports, GPIF, slave FIFOs)				E601
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	0	0	0	0	0	0	0

IFCLKSRC：FIFO 时钟内部/外部时钟源选择，0 外部时钟源，1 内部时钟源。

3048MHZ：内部 FIFO/GPIP 模式下的时钟频率，0 代表 30MHz，1 代表 48 MHz。

IFCLKOE：IFCLK 时钟输出使能，0 关闭，1 打开。

IFCLKPOL：IFCLK 输出反转使能，0 不反转，1 反转。

ASYNC：SlaveFIFO 同步/异步工作方式选择，0 同步，1 异步。

GSTATE：选择是否将 GSTATE[2:0]在 PORTE[2:0]输出，0 关闭，1 使能。

IFCFG[1:0]：选择接口模式（端口、GPIF 或从设备 FIFO）。00：I/O 方式；01：reserved；11：SlaveFIFO 方式；10：GPIF 方式。

（6）EP2/4/6/8CFG：端点 2、4、6 和 8 配置寄存器

EP2CFG	Endpoint 2 Configuration						E612
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	Endpoint 4 Configuration						E613
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	Endpoint 6 Configuration						E614
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	Endpoint 8 Configuration						E615
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：激活端点。0 停用端点，1 激活端点（默认）。

DIR：设置端点方向。0 为 OUT，1 为 IN。

TYPE[1:0]：定义端点类型。00：无效；01：等时；10：批量（默认）；11：中断。

SIZE：设置端点缓冲区的大小。端点 4 和 8 只能是 512 字节。端点 2 和 6 是可选择。0：512 字节；1：1024 字节

BUF[1:0]：缓冲类型/数量。00：四倍；01：无效；10：双倍；11：三倍。

注：当 EZUSB 分配缓冲区空间时，将忽略 Valid 位（例如，BUF[1:0]优先于 Valid 位）。当您没有使用端点配置中的所有端点时，禁用未使用的通过向相应 EPxCFG 寄存器的“有效”位写入零，而不会干扰寄存器中其他位的默认状态。

（7）FIFORESET：FIFO 复位寄存器

FIFORESET see Section 15.15	Restore FIFOs to Default State						E604
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：NAK 所有来自主机的传输。

EP[3:0]：端点号。通过写入所需的端点编号(2,4,6,8)，EZ-USB 逻辑重置单个端点。

通过向该寄存器写入 0x80 可以对来自主机的所有传输进行 NAK，然后写入 0x82、0x84、0x86 或 0x88 以继续 NAKALL 并复位单个 FIFO。这会将端点 FIFO 标志和字节计数恢复到它们的默认状态。写入 0x00 可以恢复正常操作。

（8）EP2/4/6/8FIFOCFG：端点 2、4、6 和 8/从设备 FIFO 配置寄存器

EP2FIFOCFG see Section 15.15	**Endpoint 2/Slave FIFO Configuration**	**E618**
EP4FIFOCFG see Section 15.15	**Endpoint 4/Slave FIFO Configuration**	**E619**
EP6FIFOCFG see Section 15.15	**Endpoint 6/Slave FIFO Configuration**	**E61A**
EP8FIFOCFG see Section 15.15	**Endpoint 8/Slave FIFO Configuration**	**E61B**

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

NFM1：FIFO 状态标志是否提前一个字节有效选择，IN 端点满减 1， 1 使能， 0 非使能。

OEP1：FIFO 状态标志是否提前一个字节有效选择，OUT 端点空加 1， 1 使能， 0 非使能。

AUTOOUT：在前面，我们说 SlaveFIFO 方式下的数据传输过程不需要 FX2 固件的参与，实际上是不确切的，应该说，FX2 固件可以不参与数据传输过程，也可以参与。AUTOOUT 即可设置。如果设置 AUTOOUT 为 1，则就如上面所说的，FX2 固件只需要完成初始化工作，真正的数据传输是不需要 FX2 固件的参与的，具体的说，当 FX2 从主机收到一包数据时，外部逻辑即可看到 FIFO 端点缓冲区状态的改变，然后从中取数。如果设置 AUTOOUT 为 0，则数据传输过程就需要 FX2 参与了，此时当 FX2 从主机收到一包数据时，FIFO 端点缓冲区状态的改变并不会立刻在端口显现，而是固件先看到 FIFO 端点状态的改变，此时，FX2 固件可以做三件事情：

a. 向 OUTPKTEND 中的 SKIP 位写 0，使 FIFO 端点状态的改变在端口显现，从而使外部逻辑可以从 FIFO 端点中读取数据；

b. 向 OUTPKTEND 中的 SKIP 位写 1，丢掉这包数据，这样就相当于主机从来就没有发送这一包数据，外部逻辑当然也不能从 FIFO 端点中读到这一包数据了；

c. 从新编辑这一包数据，设置完全重写整个包的数据，再写 EPxBC 寄存器，把数据传给外部逻辑。

在 FX2 复位之后，如果其 OUT 端点缓冲区内有一包数据未处理，这包数据并不会自动传给外部逻辑。所以，为保证 OUT 端点缓冲区内没有未处理数据，在 resetFX2 后，要清空一下 OUT 端点缓冲区,具体做法就是向 SKIP 位写 1（OUT 端点缓冲区有几个缓冲区就写几次）。

AUTOIN：AutoIN 和 AutoOUT 有一点不同，在 AutoOUT 里，包的大小只能是 512 或 1024，而在 AutoIN 里，包的大小可以任意设定，甚至可以是 0 字节，这可以通过 EPxAUTOINLENTH/L 设置。

和 AUTOOUT 类似，当设置 AUTOIN＝0 时，FX2 固件可以传输，丢弃，修改外部逻辑传过来的数据，这通过向 INPTKEND 寄存器的 SKIP 写不同的值实现。

ZEROLENIN：是否允许传输 0 字节,1 使能， 0 非使能。

WORDWIDE：8Bit，16Bit 选择。当选择 8Bit 模式时，PortB 将是 FD[7：0]；当选择 16Bit 模式时，PortD 将是 FD[15:8],1 则为 16 位，0 则为 8 位。

（9）EP2/4/6/8AUTOINLENH/L 端点 AUTOIN 长度设置（仅 IN 端点有效）

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

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

PL[10:0]设置 AUTOIN 时自动传输的包大小。

注：包大小不能大于 IN 端点的缓冲区的大小。PL10 仅端点 2 和 6 有效。所有端点的默认数据包大小为 512 字节。注意 EP2 和 EP6 的最大长度为 1024 字节，EP4 和 EP8 的最大长度为 512 字节，与端点结构保持一致

（10）EP2/4/6/8ISOINPKTS：每个帧寄存器的端点 2ISOIN 数据包

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

AADJ：自动调整。如果 AADJ 设置为“1”，则 FX2LP 自动管理高速、高带宽等时 IN 端点的数据 PID 排序，这些端点需要每个微帧中的额外事务。收到微帧中的第一个 IN 令牌后，FX2LP 逻辑评估 INFIFO 中已提交数据包的填充度。如果逻辑检测到包含少于 1024 字节的提交短数据包，则不会在微帧中发送超出短数据包的其他数据包。如果 INFIFO 没有提交的数据包，则在微帧中发送单个零长度数据包。如果 INFIFO 充满 1024 字节的数据包，则微帧中发送的数据包数量受 INPPF[1:0]设置的限制。在高速和全速模式下，如果 INFIFO 没有提交的数据包，EZUSB 会发送一个零长度的 IN 数据包。如果 AADJ 设置为“0”，对于微帧内的 IN 事务，FX2LP 始终以与 INPPF[1:0]中指定的每个微帧的数据包数量相对应的数据 PID 开始。例如，如果 INPPF[1:0]=10（每微帧两个数据包），即使数据包很短，或者没有数据，FX2LP 也会为微帧中的第一个 IN 事务返回 DATA1 的

数据 PID 可用于微帧中的下一个 IN 事务。在全速模式下，EZ-USB 每帧仅发送一个数据包，而不管 EPxISOINPKTS 寄存器设置如何。

INPPF[1:0]：每帧 IN 数据包。如果 EP2 是 ISOCHRONOUSIN 端点，则这些位确定每个微帧（高速模式）要发送的数据包数量。允许的值为 1、2 或 3。

（11）EP1OUT/INCFG：端点 1-OUT/IN 配置寄存器

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

VALID：激活端点。0 停用端点，1 激活端点（默认）

TYPE[1:0]：定义端点类型。00：无效；01：等时；10：批量（默认）；11：中断。

2.2.3 固件代码修改

在详细了解 USB 固件代码中常用的描述符和寄存器配置信息之后，在赛普拉斯官方实例的基础上对其进行修改，使其能满足实际使用需求。FX2LP 固件模块的组合情况如图 2.6 所示。

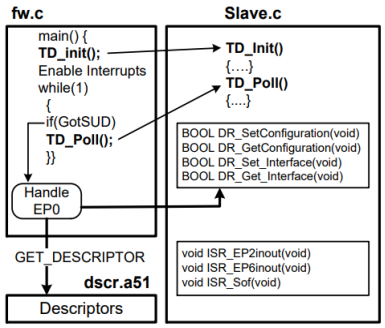

图 2.6 FX2LP 固件模块组合情况

fw.c 为赛普拉斯框架文件，其包含 main 函数，执行了 USB 维持的大部分操作（如进行枚举），并且每当需要自定义时，它将调用应用代码（Slave.c）中特定名称的外部函数。执行各个日常操作的步骤后，该函数将调用 Slave.c 所提供的外部函数，即 TD_Init。（前缀 TD 表示“任务调度”）然后，它进入一个无限循环，以通过 CONTROL 端点 0 检查 SETUP 数据包的到来。该循环还会检查 USB 暂停事件，但从设备 FIFO 应用不会使用该循环。每次进入该循环时，该函数都将调用 Slave.c 文件中提供的外部函数 TD_Poll。

dscr.a51 为描述符文件，是一个包含用于特定 USB 器件的描述符数据的 8051 汇编语言模块，主要负责修改端点方向、传输类型等参数。

slave.c 是从设备 FIFO 应用的用户代码文件，主要包括 TD_Init 和 TD_Poll 等函数，可对从设备 FIFO 参数进行初始化设置等操作。

对于用户来说，主要通过修改描述符文件中的端点描述符和用户代码中的 TD_Init 和 TD_Poll，来满足开发需求。值得注意的是，在从设备 FIFO 同步读写应用中，当内部时钟源 IFCLK 有 FPGA 端提供时，需要在 TD_Poll 函数中通过更改 IFCONFIG.7 获取来自 FPGA 的 IFCLK。

修改固件代码所用软件为 Keil，在修改之前需要对开发环境进行相应配置。当需要将固件代码烧录至子板所带 EPPROM 中时，需要将生成的.hex 文件改为.iic 文件。具体操作步骤如下：

在工程文件夹下右键或者选择 Project—>OptionsforTarget'Targer1',选择 Output 选项，选中 RunUserProgram#1 选择如图 2.7 所示。

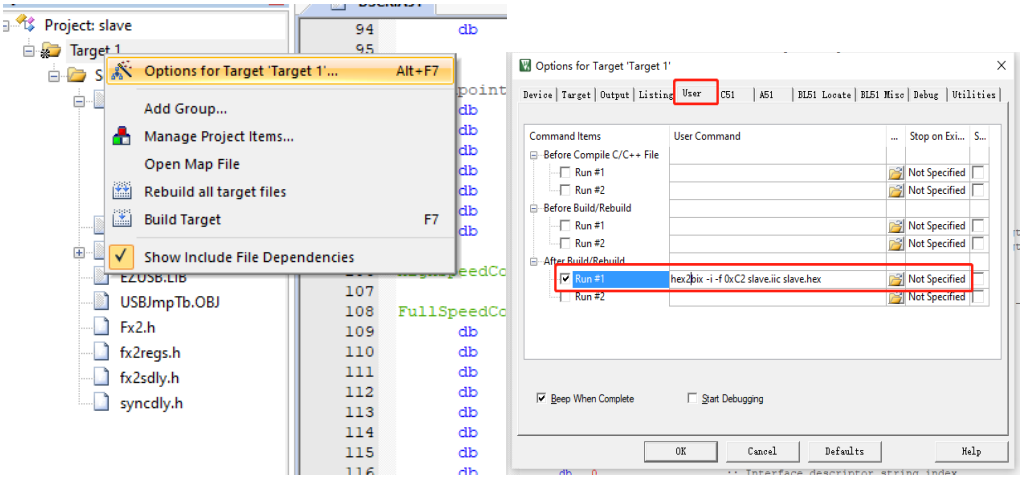

图 2.7 实际操作示意图

在上图中，需要对 RunUserProgramRun#1 中进行写下面命令：

xxxx（安装套件中的 bin 文件夹下 hex2bin 软件路径）-i-f0xC2-obulkloop.iic bulkloop.hex

命令中中-i 表示输出为.iic 文件；–f0xC2，表示烧写后，将 USB 设备再次插入到主机设备上之后，采用 C2 的启动方式（从外部 EEPROM 中启动，读取 VID,PID 等）；-o 表示将输入文件.hex 转换成.iic 文件。

设置完之后，点击编译即可输出.iic 文件。

2.3USB 上位机

由于电路系统使用的是基于 EZ-USBFX2LP 系列的 USB2.0 芯片，可通过使用 CyAPI 函数类在上位机中对 USB 芯片进行通信。

由于在校准过程中，数据连续上传且不间断，因此单独创建一个线程用于 USB 数据通信，如图 2.4 所示为 USB 通信程序。进入 USB 通讯线程，首先获取 USB 设备句柄与 USB 端点数目，再开启线程函数准备开始接收数据。在数据接收

的过程中，需要将数据从应用软件 Buffer 中提取出来作后续集中处理。待存满一次实验所需数据后，停止数据接收并关闭线程函数，USB 通信程序结束。

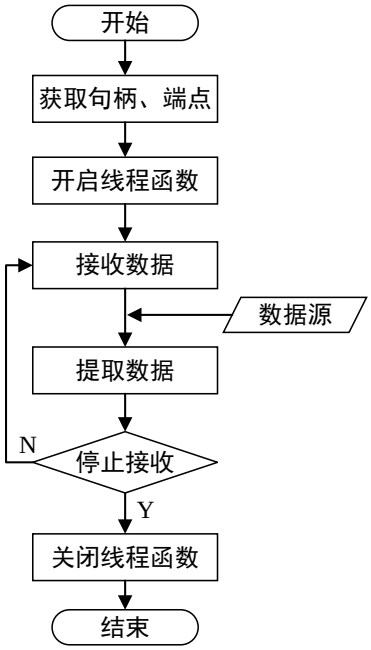

图 2.4 USB 通信程序流程图

需要注意的是，CyAPI 函数类负责接收数据函数有同步 XferData 方法和异步 BeginDataXfer/WaitForXfer/FinishDataXfer 方法。针对小数据、数据不连续的情况下，使用同步 XferData 方法更为适用；而对于连续大数据，需要使用多次传输事务的情况下，使用异步 BeginDataXfer/WaitForXfer/FinishDataXfer 方法更好。

3USB 丢数实验

针对组内 32bit 数据连续批量传输时，传输结果出现数据丢失的现象，对批量传输进行了测试，之后有对同样可用于大数据传输的等时传输进行了测试，与批量传输进行比对。

3.1 批量（BULK）传输

针对批量传输的特性，认为在整个系统的数据传输过程中，USB 与上位机间的数据传输是可靠的，丢数源应在 FPGA 与 USB 间的数据通信阶段。

系统数据传输的过程为：首先将 FPGA 产生的数据源会源源不断的写入 USB 端点的缓存区，当上位机下达传输指令时，如果传输成功，则会将端点缓冲区内数据写入上位机接收缓冲区内，上位机在进行数据处理前需要先将接收缓冲区内的数据取出放入线程中进行处理，并且在取出后会准备下一阶段的数据传输。

在这一过程中，有几个关键点：

（1）USB 端点的缓冲区最大可容纳 4kB 数据，因此当数据流过大时，端点缓冲区会出现不可用状态，只有当其有空余空间时，才会允许数据写入。

（2）上位机的接收缓冲区相当于“一个大小固定，只有一个门的封闭空间”，因此在接收数据时，无法对其进行任何操作，只有接收完成后，才能对其进行相应处理。

因此，对于 USB 来说，其写入数据是连续的，读出数据是间断的。同时根据 BULK 传输速率不恒定的特点，其相邻两次读出数据事务的间隔时间也不恒定，当间隔时间过长，则会出现 USB 端点缓冲区数据溢出情况，即丢数现象。

对于 BULK 传输方式的丢数，只要保证其在最差的传输情况下，FPGA 产生数据源与 USB 端点间的外加缓冲区不会被写满，就可以避免丢数。为此，采用 FPGA 片内 FIFO IP 核作为外加缓冲区，设置其工作模式为异步，大小为 128kB。同时 FPGA 与 USB 的通信方式为从设备 FIFO（Slave FIFO）模式，上位机工作模式为多线程，一个线程负责 USB 接收数据，一个线程负责数据处理（进制转换、保存 TXT 文件）。

当数据流为 9.6Mbps 时，相邻两数据作差曲线如图 3.1 所示，有明显的丢数现象发生。

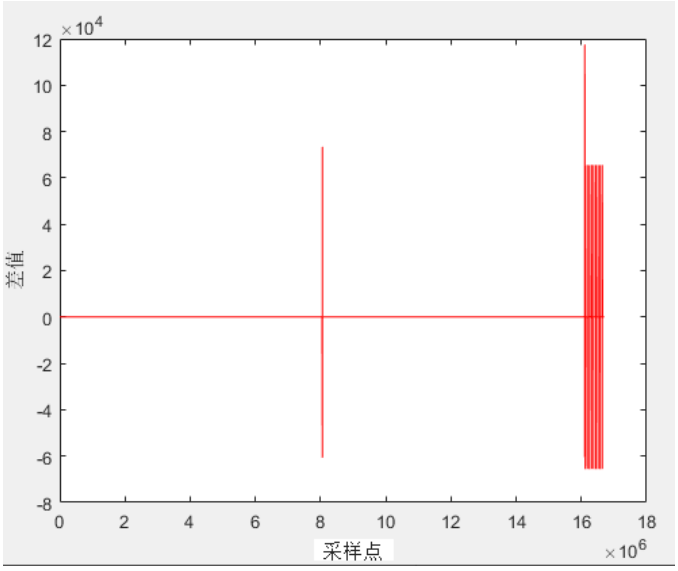

图 3.1 相邻两数据作差曲线

当数据流为 6.4Mbps 时，相邻两数据作差曲线如图 3.2 所示，没有发生丢数现象。

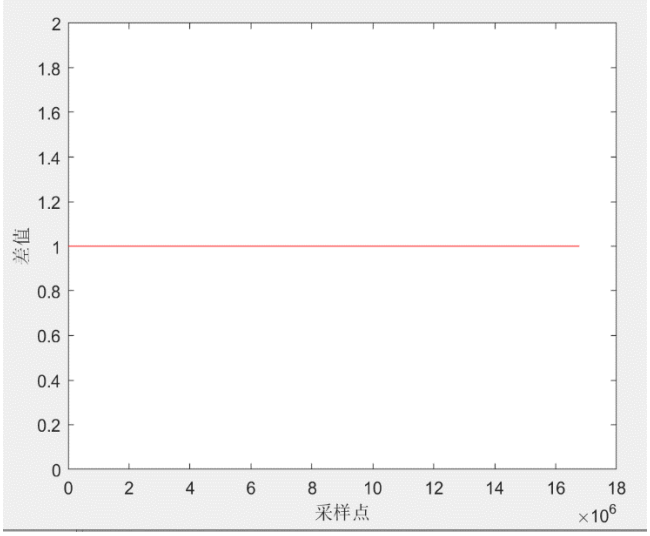

图 3.2 相邻两数据作差曲线

3.2 等时（ISOC）传输

根据 ISOC 传输特点，其丢数源即可能出现在 FPGA 与 USB 通信阶段，也可能发生在 USB 与上位机通信阶段。

基本流程与 BULK 类似，唯一区别在于该过程没有握手判断，数据包以恒定速率传输，不论传输是否成功，都会发送新的数据包。此外，ISOC 传输要求接收缓冲区长度和端点的传输大小（SetXferSize）必须是端点的 MaxPktSize 的 8 倍。因为 USB 2.0 规范允许单个设备每微帧请求最多三个同步数据包，这将使微帧中的最大带宽为 3072 字节。因此当端点的 MaxPktSize 设置为 3072 字节时，其接收缓冲区大小应为 24576 字节。

针对 ISOC 传输的数据不可靠性，需要先确保数据在 USB 与上位机通讯前没有因为前端的缓冲区溢出而丢数，考虑到所设置的外加缓冲区 FIFO IP 核深度为 128kB，加上 USB 端点缓冲区 3072B，总共可保证写入 33536 个 32bit 数。该实验结果如表 3.1 所示。

表 3.1 实验结果

数据流(Mbps)	传输数据量（个 32bit 数）	结果
6.4	33536	未丢数
9.6	33536	未丢数
12.8	33536	未丢数
32	33536	未丢数
320	33536	未丢数
3200	33536	未丢数

根据表中数据，可以初步判定，只要 FPGA 与 USB 通信阶段不出现因数据溢出造成的数据丢失，则整个系统就不会出现数据丢失。

为了对比 BULK 传输效果，当数据流为 6.4Mbps 时，发送 10444800 个 32bit 数据时，，ISOC 实际只接受到了 40856576 字节数据（即 10214144 个 32bbit 数据），如图 3.3（a）所示。与之相比，BULK 传输实际接收了 41779200 字节数据（即 10444800 个 32bit 数据）符合发送的数据量，如图 3.3（b）所示。

图 3.3（a） ISOC 模式实际接收情况　　（b）BULK 模式实际接收情况

3.3 补充实验

之前的批量传输实验上位机采用的是单次批量事务（一次 1 个任务队列，每个队列一个包），这里结合等时传输实验，参照官方测速实例，采用多次批量事务（一次 64 个任务队列，每个队列 64 个包）进行了补充实验，并加上了曲线绘制功能（每 51200 个数据绘制一次图形）结果如图 3.4 所示。

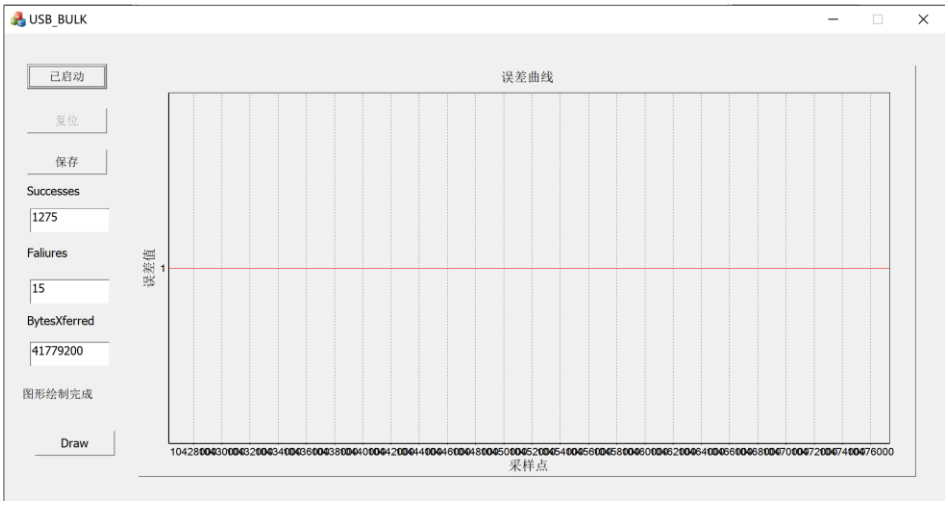

图 3.4 6.4Mbps 接收情况

图 3.4 中 Successes 表示接收成功次数，Failures 表示接收失败次数，BytesXferred 表示成功接收的总字节数，误差曲线为相邻接收数据的作差值。此外，接收总字节数与实际发送的字节数相同，并且实际测试过程中误差曲线也没有突发调变，一直是误差值为 1 的直线。最终得出通过调整传输任务队列的个数和每次传输操作的数据包数量可以提高数据速率。

4 结论

数据丢失的原因多种多样，不仅在等时（ISOC）传输是这样，在任何 USB 传输中也是如此。硬件和软件故障，外部条件都是造成数丢失的因素之一。 但是，在其他 USB 传输中，比如 BULK 传输模式由于其具有握手包及数据重传机制，可以纠正损坏，但在等时（ISOC）传输中不可能。

本文叙述了一种解决 USB 传输丢数的方案，即将数据丢失的重心集中在 FPGA 与 USB 通信阶段，通过扩大数据发送端的缓冲区大小，来提升数据稳定传输的极限值，并通过实验验证了其可行性。但是本文实验内容上存在一定局限性，由于 FPGA 片内 FIFO 存储空间有着一定的限制，只能测试 128k 大小的外加缓冲区下的情况，后续可以考虑利用外部存储芯片（如 SDRAM,DDR2 等）进行更细致的实验，给出更加具体可行性的相关参数指标。