AFDX 终端控制器

IP 核用户手册

（Ver：1.0）

珠 海 欧 比 特 控 制 工 程 股 份 有 限 公 司

地址: 广东省珠海市唐家东岸白沙路 1 号欧比特科技园　邮编: 519080

电话: 0756-3391979　传真: 0756-3391980　网址: www.myorbita.net

修改记录

版本变更	日期	修改内容
V1.0	2014-03-20	首次发布。

版权声明

免责声明

本文档仅提供阶段性信息，所含内容可根据产品的实际情况随时更新，恕不另行通知。如因文档使用不当造成的直接或间接损失，珠海欧比特控制工程股份有限公司不承担任何责任。

珠海欧比特控制工程股份有限公司

ZHUHAI ORBITA CONTROL ENGINEERING CO. , LTD

地址(Addr)：广东省珠海市唐家东岸白沙路 1 号欧比特科技园

Orbita Tech Park, 1 Baisha Road, Tangjia Dong'an, Zhuhai, Guangdong, China

邮编：519080

电话(Tel)：+86 756-3391979

传真(Fax)：+86 756-3391980

网址(web)：www.myorbita.net

目　录

1. OBT-AFDX 简介

AFDX终端控制器，以下简称OBT-AFDX。OBT-AFDX终端是根据ARINC664第7部分关于AFDX终端的协议规范设计完成的。 AFDX是在IEEE802.3以太网标准框架之内，强化了网络通信的实时性和确定性，用于航空应用的物理连接方式。AFDX 网络体系结构主要由航空电子子系统、端系统和AFDX 互连模块3个部分组成。

端系统是AFDX网络的重要组成部分，它嵌入在航空电子子系统中，将航空电子子系统与AFDX网络连接起来，实现航空电子子系统与AFDX网络发送和接收消息的目的，保证了航空电子子系统与AFDX网络上的其它子系统安全、可靠地进行数据交换。

AFDX 网络采用全双工访问，网络中通信节点是预先配置的，不能随机动态增加或删除。同时基于虚拟链路、带宽分配间隙和抖动等概念为每个链路安排了固定的传输带宽和最大传输时延，从源头上避免数据传输过程冲突的产生，从而使 AFDX 网络具有了实时性和确定性的特性。同时，通过采用双冗余网络，确保了 AFDX 通讯的可靠性。

1.1 OBT-AFDX 主要特征

OBT-AFDX 终端特性：

- 支持ARINC664规范；
- 提供2个10/100M的RJ45形式的端口，支持2个端口独立使用或2个端口互为冗余；
- 软件实现传输层(UPD)、网络层(IP)协议，硬件实现链路层(Virtual Link)及物理层(PHY)层协议 ；
- 高性能的AMBA总线与主机间的数据传输 ；
- 支持采样(Sampling)和列队(Queuing)端口；
- 支持发送和接收各128条虚拟链路(VL)；
- 处理器采用Virtex 5。

OBT-AFDX终端主要实现以下功能：

- 配置管理：负责对各个终端的各条虚链路进行参数配置，配置内容包括终端系统的网络ID、设备ID，还包括虚链路中的带宽分配间隙、最大帧长、冗余管理、端

口类型、状态等配置参数。

- 信息封装：对应用层信息进行UDP、IP和MAC的封装，封装的首部内容根据虚链路的配置参数确定。
- 流量整形：为了保证每个VL的带宽分配间隙，调整器对每个虚拟链路的数据帧在发送前按照BAG进行了流量调整。
- 虚拟链路调度：当发送ES具有多路VL时，调度器多路复用来自调整器的不同的数据帧，防止数据帧发送的冲突。
- 完整性检查：为了保证每条虚链路上信息传送的可靠性，接收终端必须进行完整性检查，在AFDX终端主要依靠对顺序号的检查来实现。
- 冗余管理：AFDX网络为了能够保证信息安全发送，防止信息传输过程中的丢失，采用了两个独立的冗余网络进行传输。

2. OBT-AFDX 功能结构框图

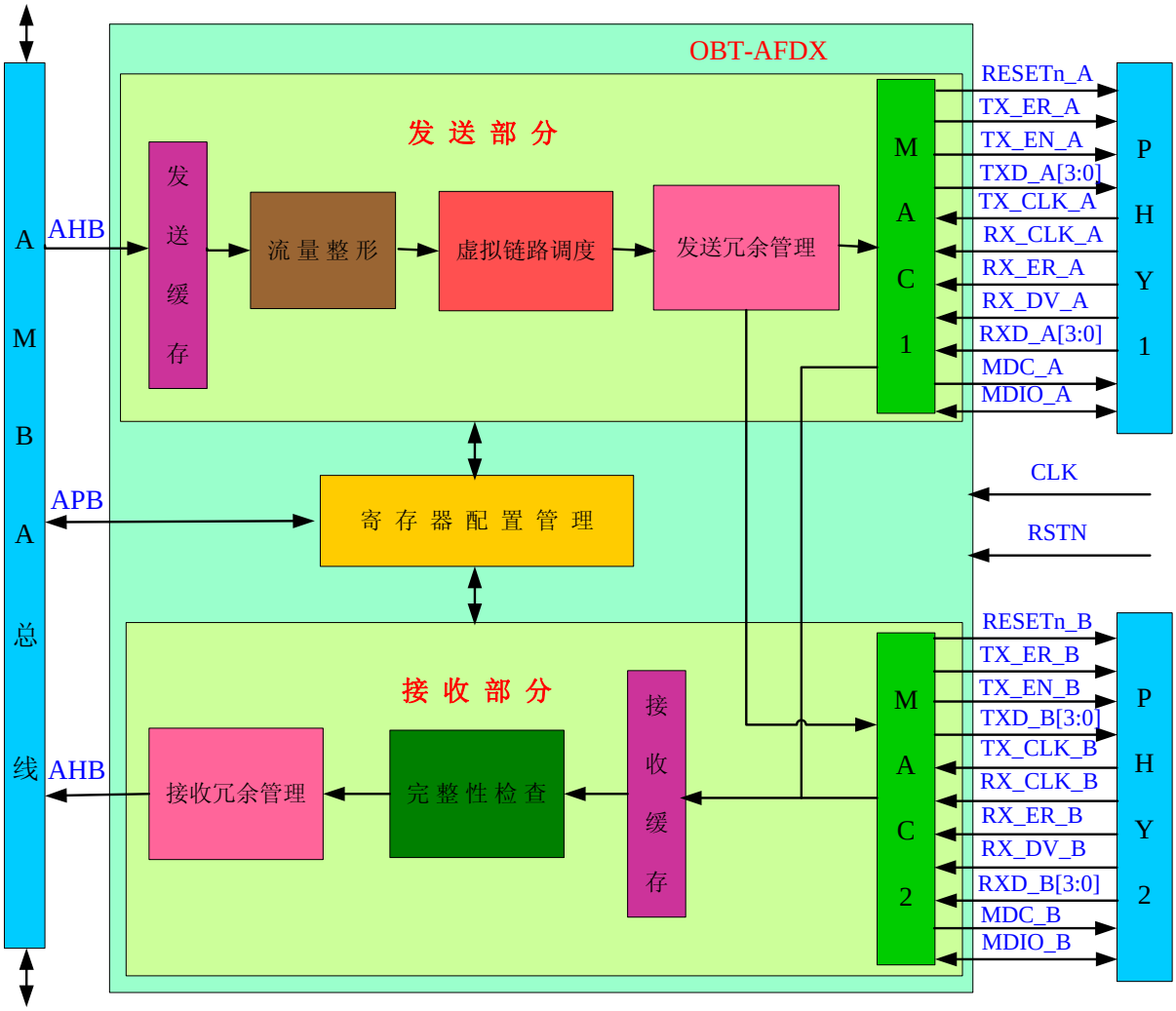

图 2-1 OBT-AFDX 功能结构框图

OBT-AFDX 结构框图如图 2-1 所示。各模块功能说明分别如下表 2-1 所示：

表 2-1 OBT-AFDX 各功能模块功能说明

序号	模块名称	模块描述
1	AMBA 总线	通过 AMBA 的 AHB 总线，实现 AFDX 和 SRAM 的数据交换；通过 AMBA 的 APB 总线，实现 AFDX 的寄存器配置。
2	发送缓存	AFDX 分为 128 个虚拟链路，每个虚拟链路数据单独开辟一个存储空间。发送缓存分成 128 个存储空间，来存储 128 个虚拟链路的发送数据。

序号	模块名称	模块描述
3	流量整形	流量整形是在单条虚拟链路上调整数据帧的发送时间，使每条虚拟链路在各自配置的时间间隔内，只发送一帧。
4	虚拟链路调度	虚拟链路的调度就是通过静态优先级的调度算法，将多条虚拟链路的数据帧，复用到一个通道（AFDX 网络的物理链路）进行传输。
5	发送冗余管理	AFDX 采用冗余网络增强数据的可靠性，发送冗余管理把要发送的数据进行复制，在两个互为冗余的网络同时传输帧。
6	寄存器配置管理	寄存器配置管理主要实现对 AFDX 功能的配置，能够间接反映 AFDX 的功能.
7	接收缓存	进过 MAC 层过来的数据，要先进行缓存，使数据帧在经过接收处理的同时，不影响数据帧的接收。
8	完整性检查	完整性检查是对同一虚拟链路连续接收的帧进行序列号的检查，判断虚拟链路接收的帧序列是否完整有效。
9	接收冗余管理	接收冗余管理是对接收到互为冗余的数据帧进一步处理，丢弃冗余帧。
10	MAC	MAC 模块主要实现 CRC 校验，配置 PHY 寄存器以及和 PHY 进行通讯功能。
11	PHY	AFDX 的网络接口。

表 2-2　OBT-AFDX 外部接口信号说明

序号	信号名称	信号方向	信号描述
1	CLK	I	系统输入时钟
2	RSTN	I	外部复位信号，低电平有效
3	AHB	I/O	接 AMBA.AHB2.0, 主要是地址线，数

序号	信号名称	信号方向	信号描述
			据线，读写控制线，数据交互信号线
4	APB	I/O	接 AMBA.APB2.0，主要是地址线，数据线，读写控制线，寄存器配置信号线
5	RESETn_A	O	通道 A 复位 PHY 信号，低电平有效
6	TX_ER_A	O	通道 A 发送数据错误
7	TX_EN_A	O	通道 A 发送使能，当使能时告诉 PHY 数据有效
8	TXD_A[3:0]	O	通道 A 发送数据信号
9	TX_CLK_A	I	通道 A 发送单元时钟，PHY 提供
10	RX_CLK_A	I	通道 A 接收单元时钟，PHY 提供
11	RX_ER_A	I	通道 A 接收错，PHY 在当前传输帧提示传输媒质错误
12	RX_DV_A	I	通道 A 接收数据有效，PHY 告知 MAC 数据信号有效
13	RXD_A[3:0]	I	通道 A 接收数据信号
14	MDC_A	I	通道 A 管理数据时钟，MAC 提供给 PHY
15	MDIO_A	I/O	通道 A 管理数据的输入输出，双向数据通道用于 MAC 和 PHY 的数据交换
16	RESETn_B	O	通道 B 复位 PHY 信号，低电平有效
17	TX_ER_B	O	通道 B 发送数据错误
18	TX_EN_B	O	通道 B 发送使能，当使能时告诉 PHY 数据有效
19	TXD_B[3:0]	O	通道 B 发送数据信号
20	TX_CLK_B	I	通道 B 发送单元时钟，PHY 提供
21	RX_CLK_B	I	通道 B 接收单元时钟，PHY 提供
22	RX_ER_B	I	通道 B 接收错，PHY 在当前传输帧提

序号	信号名称	信号方向	信号描述
			示传输媒质错误
23	RX_DV_B	I	通道 B 接收数据有效，PHY 告知 MAC 数据信号有效
24	RXD_B[3:0]	I	通道 B 接收数据信号
25	MDC_B	I	通道 B 管理数据时钟，MAC 提供给 PHY
26	MDIO_B	I/O	通道 B 管理数据的输入输出，双向数据通道用于 MAC 和 PHY 的数据交换

3. OBT-AFDX 寄存器描述

3.1 OBT-AFDX 寄存器地址

表 3-1 寄存器地址分配

地址	读/写	有效位宽	寄存器描述
0x8000 0F00	RD/WR	32	控制寄存器（CTRL）
0x8000 0F04	RD/WR	32	状态寄存器（STATUS）
0x8000 0F08	RD/WR	32	配置 PHY 寄存器 A（MDIO_A）
0x8000 0F0c	RD/WR	32	配置 PHY 寄存器 B（MDIO_B）
0x8000 0F10	RD/WR	32	发送描述符寄存器（Tx Descriptor）
0x8000 0F14	RD/WR	32	接收描述符寄存器（Rx Descriptor）
0x8000 0F18	RD/WR	32	最大抖动值寄存器（Jittermax）
0x8000 0F1c	RD/WR	32	发送虚拟链路号寄存器（Tx_Id）
0x8000 0F20	RD/WR	32	发送虚拟链路参数寄存器（Tx_Regs）
0x8000 0F24	RD/WR	32	接收虚拟链路号寄存器（Rx_Id）
0x8000 0F28	RD/WR	32	接收虚拟链路参数寄存器（Rx_Regs）

3.2 OBT-AFDX 寄存器说明

3.2.1 控制寄存器（CTRL）

表 3-2 控制寄存器(CRTL)

位(BIT)	描述(DESCRIPTION)
31-6	保留
5	速度(SP)，设置现在的速度模式。0 = 10 Mbit, 1 = 100 Mbit。只在 RMII 模式使用(rmii = 1)，默认值自动在复位后从 PHY 读取
4	复位(RS)，该位写 1 后，将复位这个核
3	接收中断(RI)，使能接收器中断。当这位为 1 时，每次接收到一个包，将会产生一个中断。不管这个包正确接收还是因为错误而终止接收，这个中断都会产生

位(BIT)	描述(DESCRIPTION)
2	发送中断(TI)，使能发送器中断。当这位为 1 时，每次一个包发出，将会产生一个中断。不管这个包正确传输还是因为错误而终止发送，这个中断都会产生
1	接收使能(RE)，在每次新的描述符被使能后应该置‘1’。只要这位是 1，将读取新的描述符，直到它遭遇一个未被使能的描述符，此时它将停止直到 RE 被重新设定。这一位应该在新的描述符被使能后才写入 1
0	发送使能(TE)，应该在每次新的描述器被使能后写入 1。只要这位是 1，OBTETH 将读取新的描述符，直到它遭遇一个未被使能的描述符，此时它将停止直到 TE 被重新设定。这一位应该在新的描述符被使能后才写入 1

3.2.2 状态寄存器（STATUS）

表 3-3 状态寄存器(STATUS)

位(BIT)	描述(DESCRIPTION)
31-8	保留
7	无效地址(IA)，MAC 接收到一个不接受地址的包。当写‘1’时清除
6	过小(TS)，接收到一个包。这个包的大小比允许的最小值小。当写入‘1’时清除
5	发送器 AHB 错误(TA)，在发送器 DMA 器中发生 AHB 错误。当写入‘1’时被清除
4	接收器 AHB 错误(RA)，在接收器 DMA 器中发生 AHB 错误。当写入‘1’时被清除
3	发送器中断(TI)，一个包被正确发送。当写入‘1’被清除
2	接收器中断(RI)，一个包被正确接收。当写入‘1’被清除
1	发送器错误(TE)，一个包将被发送，但是发送以错误结束。当写入‘1’清除
0	接收器错误(RE)，一个包将被接收，但是接收以错误结束。当写入‘1’清除

3.2.3 配置 PHY 寄存器 A（MDIO_A）

表 3-4 配置 PHY 寄存器 A(MDIO_A)

位(BIT)	描述(DESCRIPTION)
31-16	数据(DATA)，包含读操作时的数据或者需要发送的数据
15-11	PHY 地址(PHY ADDRESS)，这个领域包含要访问的 PHY 地址
10-6	寄存器地址(REGESTER ADDRESS)，这个领域包含要访问的寄存器地址
5-4	保留
3	忙(BU)，当操作启动，这位被设定为‘1’。一旦操作完成，管理连接空闲，这位清零
2	连接失败(LF)，当操作完成（BUSY = 0），如果一个功能性管理连结没有检测到这位置位
1	读(RD)，在操作接口开始读操作。数据存储在数据领域
0	写(WR)，在操作接口开始写操作。从资料领域取数

3.2.4 配置 PHY 寄存器 B（MDIO_B）

表 3-5 配置 PHY 寄存器 B(MDIO_B)

位(BIT)	描述(DESCRIPTION)
31-16	数据(DATA)，包含读操作时的数据或者需要发送的数据
15-11	PHY 地址(PHY ADDRESS)，这个领域包含要访问的 PHY 地址
10-6	寄存器地址(REGESTER ADDRESS)，这个领域包含要访问的寄存器地址
5-4	保留
3	忙(BU)，当操作启动，这位被设定为‘1’。一旦操作完成，管理连接空闲，这位清零
2	连接失败(LF)，当操作完成（BUSY = 0），如果一个功能性管理连结没有检

位(BIT)	描述(DESCRIPTION)
	测到这位置位
1	读(RD)，在操作接口开始读操作。数据存储在数据领域
0	写(WR)，在操作接口开始写操作。从资料领域取数

3.2.5 发送描述符寄存器（Tx Descriptor）

表 3-6 发送描述符寄存器(Tx Descriptior)

位(BIT)	描述(DESCRIPTION)
31-10	发送描述附表的基地址(TRANSMITTER DESCRIPTOR TABLE ADDRESS)
9-3	有效描述符偏移地址(DESCRIPTOR POINTER)，被以太网 MAC 自动增加
2-0	保留

3.2.6 接收描述符寄存器（Rx Descriptor）

表 3-7 接收描述符寄存器(Rx Descriptior)

位(BIT)	描述(DESCRIPTION)
31-10	接收描述附表的基地址(RECEIVE DESCRIPTOR TABLE ADDRESS)
9-3	效描述符偏移地址(DESCRIPTOR POINTER)，被以太网 MAC 自动增加
2-0	保留

3.2.7 最大抖动值寄存器（Jittermax）

表 3-8 最大抖动值寄存器(Jittermax)

位(BIT)	描述(DESCRIPTION)
31-16	保留
15-0	虚拟链路抖动最大时间设定值(JITTERMAX)

3.2.8 发送虚拟链路号寄存器（Tx_Id）

表 3-9 发送虚拟链路号寄存器(Tx_Id)

位(BIT)	描述(DESCRIPTION)
31-19	保留
18-12	发送端内部虚拟链路号(TX_LIST_ID)
11-0	发送端外部虚拟链路号(TX_VL_ID)

3.2.9 发送虚拟链路参数寄存器（Tx_Regs）

表 3-10 发送虚拟链路参数寄存器(Tx_Regs)

位(BIT)	描述(DESCRIPTION)
31-17	保留
16	相应虚拟链路是否为采样端口(SAMP)，0 为队列端口，1 为采样端口
15-13	发送端相应虚拟链路 bag 值(TX_BAG)
12-11	发送端相应虚拟链路冗余发送控制(TX_RM)，0x00、0x11 冗余发送，0x01 只有 A 路发送，0x10 只有 B 路发送
10-0	发送端相应虚拟链路的最大帧长(TX_LENMAX)

3.2.10 接收虚拟链路号寄存器（Rx_Id）

表 3-11 接收虚拟链路号寄存器(Rx_Id)

位(BIT)	描述(DESCRIPTION)
31-19	保留
18-12	接收端内部虚拟链路号(RX_LIST_ID)

位(BIT)	描述(DESCRIPTION)
11-0	接收端外部虚拟链路号(RX_VL_ID)

3.2.11 接收虚拟链路参数寄存器（Rx_Regs）

表 3-12 接收虚拟链路参数寄存器(Rx_Regs)

位(BIT)	描述(DESCRIPTION)
31-21	保留
10-19	接收端相应虚拟链路冗余控制(RX_RM)，0x00 开启冗余管理，其他值关闭
18-0	接收端相应虚拟链路 skewmax 值(SKEWMAX)

4. OBT-AFDX 应用案例

本公司设计的板卡 OBT-AFDX，要符合国际通用 AFDX 协议 ARINC 664 Part 7 中规定的 AFDX 标准。要想验证本公司的 OBT-AFDX 板卡是否满足设计和使用要求，就要保证其和经过国际认证的大公司设计的 AFDX 板卡正常通信。

4.1 外围接口

OBT-AFDX 外围接口电路原理图主要说明 OBT-AFDX 外部接线关系，这里主要是和 88E1111 型号的 PHY 芯片的接口电路关系。OBT-AFDX 的 FPGA 型号为 Xilinx 公司的 XC5VLX110T，由于内部 Memory 容量有限，需要大容量 Memory 可以选择适当 FPGA 型号。两个 PHY 芯片要共用一个外部晶振，因为 FPGA 和 PHY 芯片的电源电压不同，所以需要一个电源转换芯片。

OBT-AFDX 外围接口电路原理图见附录二。

4.2 Ballard AFDX 板卡通信

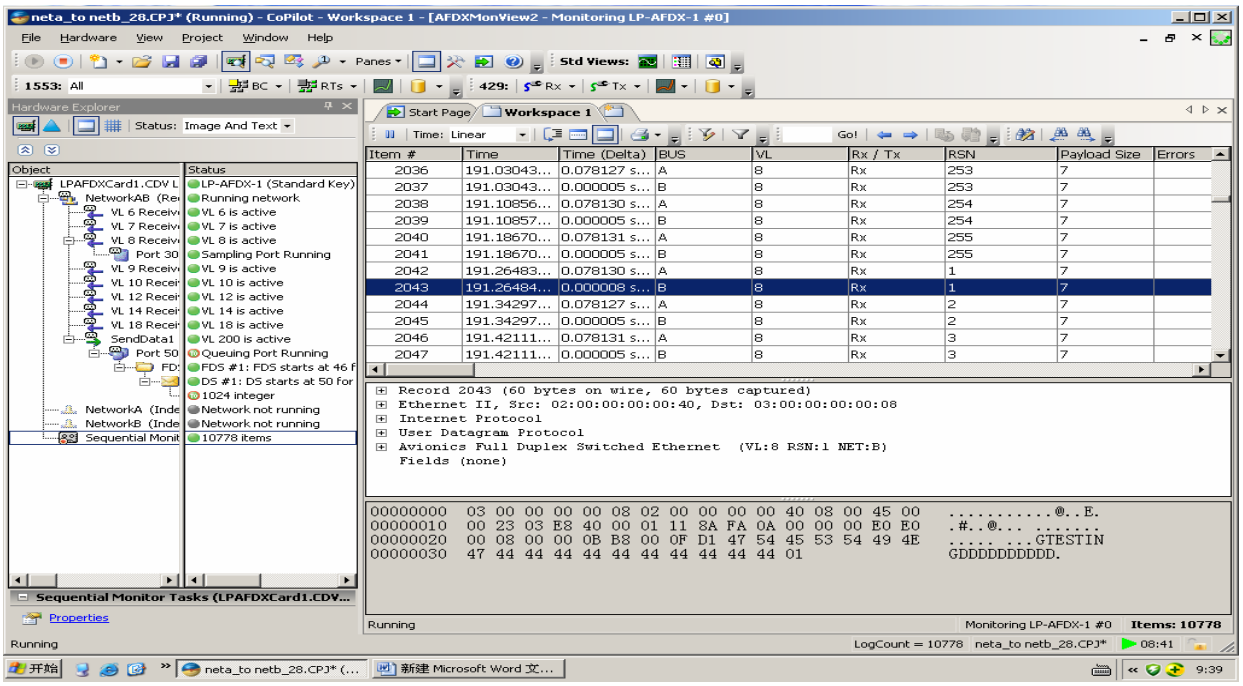

图 4-1 Ballard AFDX 板卡测试软件

Ballard 是美国一家 AFDX 板卡做的比较成熟的公司，其 AFDX 板卡已经的到国际认证，且其 AFDX 板卡在军事和民用领域都得到了广泛应用。将本公司 OBT-AFDX 和 Ballard AFDX 板卡进行通信测试，结果两个板卡之间收发数据正常，证明本公司本公司 OBT-AFDX 功能正确，满足设计和使用要求。

Ballard 公司有专门的 AFDX 板卡测试软件(CoPilot AFDX)，操作界面如图 4-1 所示，具体操作流程请参考手册 Ballard-- CoPilot Users Manual，这里只对本公司 OBT-AFDX 的测试应用程序进行说明。

OBT-AFDX 测试程序内容：包括寄存器配置测试，中断测试，发送功能测试，接收功能测试，配套软件测试，程序流程图如图 4-2 所示。

寄存器配置测试：要测试所有寄存器读写是否正常，其中有些寄存器要配置发送和接收各 128 条虚拟链路的参数设置。主要是通过以下几个寄存器完成虚拟链路参数设置功能，最大抖动值寄存器（Jittermax）、发送虚拟链路号寄存器（Tx_Id）、发送虚拟链路参数寄存器（Tx_Regs）、接收虚拟链路号寄存器（Rx_Id）、接收虚拟链路参数寄存器（Rx_Regs）。

发送功能测试：中断发送数据、网络号控制功能、SN 控制功能、冗余发送功能、流量整形功能、采样和队列端口功能、相同优先级调度功能、不同优先级调度功能。

接收功能测试：中断接收功能、完整性检查功能、冗余管理功能、SKEWMAX 功能。

软件功能测试：UDP 头添加、IP 头添加、MAC 头添加、虚拟链路号添加。

OBTAFDX 示例程序见附录一。

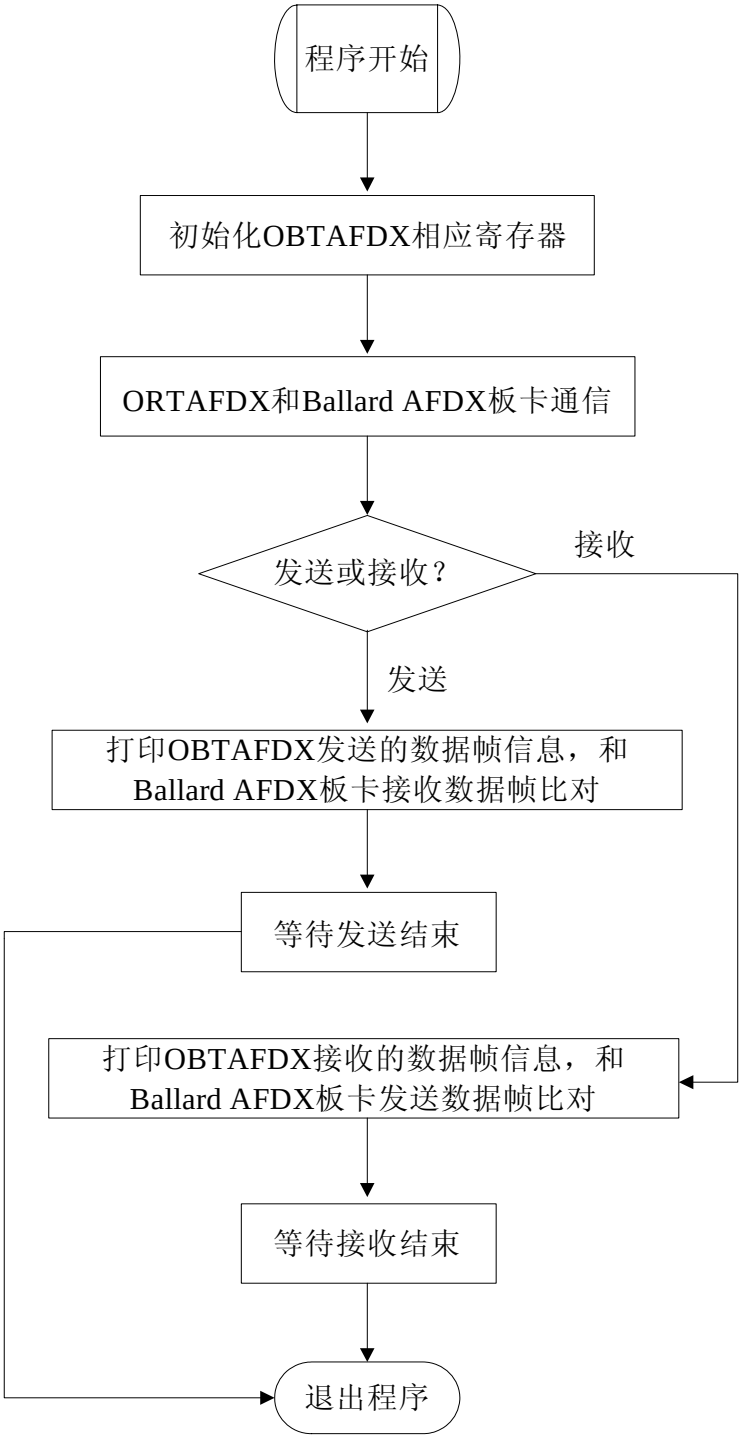

图 4-2 OBT-AFDX 程序流程图

5. 附录

5.1 附录一：OBT-AFDX 示例程序

greth.h

```
#ifndef _GR_ETH_
#define _GR_ETH_
#define GRETH_REGLOAD(x) (x)
#define GRETH_REGSAVE(x,y) (x = (y))
#define GRETH_REGORIN(x,y) (x = ((x) | (y)))
#define GRETH_REGANDIN(x,y) (x = ((x) & (y)))
typedef unsigned int unsigned32;
/* Add udp head*/
/*typedef struct _udp_head
{
 unsigned char sourceip[4];
 unsigned char destip[4];
 unsigned char flag;
 unsigned char deal;
 unsigned short datalen;
 unsigned short sourceport;
 unsigned short destport;
 unsigned short totlelen;
 unsigned short checksum;
};
*/
typedef struct _mac_head
{
 unsigned char destmac[6];
 unsigned char sourcemac[6];
};
/*typedef struct _ip_head
{
 unsigned char iptype[2];
 unsigned char ipheadstruct[20];
};
*/
typedef struct _ip_head
{
 unsigned char tyhead;                  //前 4 个 bit 为版本号，后 4 个 bit 为首部长度
 unsigned char diffser;                 //区分服务
 unsigned short iptotlelen;             //ip 包总长度
 unsigned short identification;     //标识
```

```
 unsigned short piece_excu;          //前 3 个 bit 为标志号，后 13 个 bit 为片偏移
 unsigned char ttl;                  //生存时间
 unsigned char datatype;             //数据部分协议
 unsigned short headchecksum;        //首部检验和
 unsigned char sourceip[4];
 unsigned char destip[4];
};
typedef struct _udp_head
{
 unsigned short sourceport;
 unsigned short destport;
 unsigned short udplen;
 unsigned short udpchecknum;
};
typedef struct _udp_fake_head
{
 unsigned char sourceip[4];
 unsigned char destip[4];
 unsigned char flag;
 unsigned char deal;
 unsigned short udplen;
};
typedef struct _head
{
 unsigned char destmac[6];
 unsigned char sourcemac[6];
 unsigned char iptype[2];
 unsigned char tyhead;               //前 4 个 bit 为版本号，后 4 个 bit 为首部长度
 unsigned char diffser;              //区分服务
 unsigned short iptotlelen;          //ip 包总长度
 unsigned short identification;      //标识
 unsigned short piece_excu;          //前 3 个 bit 为标志号，后 13 个 bit 为片偏移
 unsigned char ttl;                  //生存时间
 unsigned char datatype;             //数据部分协议
 unsigned short headchecksum;        //首部检验和
 unsigned char sourceip[4];
 unsigned char destip[4];
 unsigned short sourceport;
 unsigned short destport;
 unsigned short udplen;
 unsigned short udpchecknum;
};
/* Configuration Information */
```

```
typedef struct {
   unsigned32                 base_address;
   unsigned32                 vector;
   unsigned32                 txd_count;
   unsigned32                 rxd_count;
} greth_configuration_t;
/* Ethernet buffer descriptor */
typedef struct _greth_bd {
   volatile unsigned32 stat;
   volatile unsigned32 *addr;                /* Buffer address */
} greth_bd;
/* Ethernet configuration registers */
typedef struct _greth_regs {
     volatile unsigned32 control;          /* Ctrl Register */
     volatile unsigned32 status;           /* Status Register */
     volatile unsigned32 a_mdio;           /* MDIO phy_a control and status */
     volatile unsigned32 b_mdio;           /* MDIO phy_b control and status */
     volatile unsigned32 tx_desc_p;        /* Transmit descriptor pointer */
     volatile unsigned32 rx_desc_p;        /* Receive descriptor pointer */
     volatile unsigned32 jittermax;       /* afdx jittermax */
     volatile unsigned32 afdx_tx_id;      /*afdx_tx_regs list_id vl_id */
     volatile unsigned32 afdx_tx_regs; /* afdx_tx_regs    bag redundance lamx*/
     volatile unsigned32 afdx_rx_id;      /* afdx_rx_regs list_id vl_id*/
     volatile unsigned32 afdx_rx_regs; /* afdx_rx_regs skewmax*/
} greth_regs;
/*
 * Per-device data
 */
struct greth_softc
{
   volatile greth_regs *regs;
   volatile greth_bd *tx_bd_base;                  /* Address of Tx BDs. */
   volatile greth_bd *rx_bd_base;                  /* Address of Rx BDs. */
   unsigned int tx_next;                           /* Next buffer to be sent */
   unsigned int tx_last;                           /* Next buffer to be checked if packet sent */
   unsigned int tx_full;
   unsigned int rx_cur;                            /* Next buffer to be checked if packet received */
   /* configuration */
   unsigned char ac_enaddr[6];    // used
   unsigned char ipaddr[4];
   unsigned char dripaddr[4];     // used
   unsigned char maskaddr[4];     // used
   /*
    * Statistics
```

```
     */
    unsigned long rx_length_errors;
    unsigned long rx_frame_errors;
    unsigned long rx_crc_errors;
    unsigned long rx_packets;
    unsigned long tx_packets;
};
int libio_uip_greth_init(struct greth_softc *o,void (*func_ip) (void),void (*func_udp) (void));
unsigned int greth_read(void);
#define CPU_HANDLE_GRETH_HANDLE 0
#define GRETH_CR_RI 0x8
#define GRETH_CR_TI 0x4
#define GRETH_IRQ_NO     12
#define GRETH_EN_RX_IRQ 0x2000
#define GRETH_FD 0x10
#define GRETH_RESET 0x40
#define PHY_ADDR 0x7
#define GRETH_MII_BUSY 0x8
#define GRETH_MII_NVALID 0x10
#define GRETH_BD_EN 0x2800        // 原来是 0x800,现在是配置中断使能和接收使能
#define GRETH_BD_WR 0x1000
#define GRETH_BD_IE 0x2000
#define GRETH_BD_LEN 0x7FF
#define GRETH_INT_TX 0x8
#define GRETH_TXEN 0x1
#define GRETH_TXI 0x4
#define GRETH_TXBD_STATUS 0xFFFFC000
#define GRETH_TXBD_ERR_UE 0x4000
#define GRETH_TXBD_ERR_AL 0x8000
#define GRETH_TX_BUF_SIZE 2048
//#define GRETH_TXBD_NUM 2
#define GRETH_TXBD_NUM 100
#define GRETH_TXBD_NUM_MASK GRETH_TXBD_NUM-1
#define GRETH_INT_RX 0x4
#define GRETH_RXEN 0x2
#define GRETH_RXI 0x8
#define GRETH_RXBD_STATUS 0xFFFFC000
#define GRETH_RXBD_ERR_AE 0x4000
#define GRETH_RXBD_ERR_FT 0x8000
#define GRETH_RXBD_ERR_CRC 0x10000
#define GRETH_RXBD_ERR_OE 0x20000
#define GRETH_RX_BUF_SIZE 2048
#define UIP_BUFSIZE 2048
//#define GRETH_RXBD_NUM 2
```

```
#define GRETH_RXBD_NUM 100
#define GRETH_RXBD_NUM_MASK GRETH_RXBD_NUM-1
#define IRQMP_BASE                  0x80000200
#define IRQMP_P0_MASK_START IRQMP_BASE+0x40
#define MASK_IRQ(cpu,irq)         (*(volatile unsigned int *)(IRQMP_P0_MASK_START+cpu*4))|=(1<<irq)
#define UN_MASK_IRQ(cpu,irq)    (*(volatile unsigned int *)(IRQMP_P0_MASK_START+cpu*4))&=(~(1<<irq))
#define IRQMP_ICLEAR                  *(volatile unsigned int *)(IRQMP_BASE+0xc)
#define IRQ_CLEAR(irq)              IRQMP_ICLEAR|=(1<<irq)
#endif
/*****************************************************************/
#ifdef SYS_SMP_SUPPORT                        //如果定义 SYS_SMP_SUPPORT

void disable_irq(char irq)
{
 int _cpu_no_;
      for(_cpu_no_=0;_cpu_no_<SYS_SMP_CPU_MAX;_cpu_no_++)
            UN_MASK_IRQ(_cpu_no_,irq);
}
void    enable_irq(char irq)
{

 int _cpu_no_;

 for(_cpu_no_=0;_cpu_no_<SYS_SMP_CPU_MAX;_cpu_no_++)
        MASK_IRQ(_cpu_no_,irq);
}
#define disable_cpu_irq(cpu,irq)    UN_MASK_IRQ(cpu,irq)
#define enable_cpu_irq(cpu,irq)            MASK_IRQ(cpu,irq)
void    irq_set_cpu(char    cpu,char irq)
{
 int _cpu_no_;
 for(_cpu_no_=0;_cpu_no_<SYS_SMP_CPU_MAX;_cpu_no_++)
 {
        if(_cpu_no_==cpu)                    //如果是指定的 CPU，则开中断
        {
              MASK_IRQ(_cpu_no_,irq);
        }
        else                                 //如果不是指定的 CPU，则关中断
        {
              UN_MASK_IRQ(_cpu_no_,irq);
        }
 }
```

```
}
#else                                   //如果没有定义 SYS_SMP_SUPPORT,则默认用 cpu0 作为工作 CPU
#define disable_irq(irq)     UN_MASK_IRQ(0,irq)
#define enable_irq(irq)       MASK_IRQ(0,irq)
#endif
```

greth.c

```
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include <string.h>
#define MEM_CONFIG1_REG         *(volatile unsigned int *)(0x80000000)
#define MEM_CONFIG2_REG         *(volatile unsigned int *)(0x80000004)
#define UIP_LLH_LEN 14
#define MAC_HEAD_LEN 12
#define IP_TYPE_LEN 2
#define IP_HEAD_LEN 20
#define UDP_HEAD_LEN 8
#define    IDENTIFICATION 1000
#define TOTAL_HEAD_LEN MAC_HEAD_LEN+IP_HEAD_LEN+UDP_HEAD_LEN+IP_TYPE_LEN
#define DEBUG (0) //-1&(~64))
#include "greth.h"
#define ARP_PERIOD_US 2
static struct greth_softc greth_config = {
0, 0, 0, 0, 0, 0, 0, {0,1,2,3,4,5}, {192,168,0,10}, {0,0,0,0}, {255, 255, 255, 0},0, 0, 0, 0, 0};
/*static struct _udp_head udp_head = {
 {192,168,0,153}, {192,168,0,101},17,0,15,5000,5001,0,0
};*/
static struct _mac_head mac_head = { 0x03, 0x00, 0x00, 0x00, 0x00, 0x00,0x02, 0x00, 0x00, 0x00, 0x0, 0x60 };
/*static struct _ip_head ip_head = {0x08, 0, 0xE, 0xf, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf, 0, 1};
*/
static unsigned char iptype[2] = {0x08,0x00};//协议
static struct _ip_head ip_head ={0x45,0x01,0x02,0x03,0x04,0x05,0x11,0x12,{10,0,0,0}, {224,224,0,8}};
static struct _udp_fake_head fake_head = {{10,0,0,0}, {224,224,0,8},0,17,15};
static struct _udp_head udp_head = {0, 3000, 0,0};
static unsigned char head_buf[TOTAL_HEAD_LEN];
static unsigned short buffer2[500];
static int datalen;
static short vlnum = 6;
static short recevnum = 0;
static snnumfirst = 0;
static snnumsecond = 0;
static struct _head head;
```

```
static char txdata[500] = {0x54,0x45,0x53,0x54,0x49,0x4e,0x47,0x00,0x01,0x02};

static char txdata_pri[500] = {0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf
};
/*
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,

 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
 0x00,0x1,0x2,0x3,0x4,0x5,0x6,0x7,0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf
 };
 */
```

```
unsigned short EndianCovert(unsigned short InputNum)
{
 unsigned char *p = (unsigned char *)&InputNum;
 return(((unsigned short)*p<<8)+((unsigned short)*(p+1)));
}
void chartoshort(unsigned char *buf, int nword)
{
 int i;
 for (i=0; i<nword/2; i++)
 {
        buffer2[i] = buf[i*2];
        buffer2[i] = buffer2[i]<<8;
        buffer2[i] += buf[i*2+1];
 }
unsigned short checksum(unsigned short *buf, int nword)
{
 unsigned long sum = 0;
 for ( ; nword>0; nword--)
 {
        sum += *buf++;
 }
 sum = (sum>>16) + (sum&0xffff);
 sum += (sum>>16);
 return sum;
}
void ch_udp_head(struct _udp_fake_head *fakehead, struct _udp_head *udph, char *data)
{
 int len, i;
 unsigned char buffer1[600];
 unsigned short result;
 //len = strlen(data)+8;
 len = datalen+8;
 udph->udplen = len;
 udph->udpchecknum = 0;
 fakehead->udplen = len;
 unsigned char *fakeh = (unsigned char *)fakehead;
 unsigned char *uh = (unsigned char *)udph;
 for( i=0; i<12; i++)
 {
        buffer1[i] = fakeh[i];
 }
 for(; i<20; i++)
 {
        buffer1[i] = uh[i-12];
```

```
    }
    for (; i<(20+len-8); i++)
    {
        buffer1[i] = data[i-20];
    }
    /*for(i=0; i<(20+len-8); i++)
    {
        printf("%x\n",buffer1[i]);
    }
    */
    if(len%2)
    {
        buffer1[20+len-8]=0x00;
        len = len+1;
    }
    chartoshort(buffer1, 20+len-8);
    /*for(i=0; i<(20+len-8)/2; i++)
    {
        printf("%x\n",buffer2[i]);
    }
    */
    result = ~(checksum(buffer2, (20+len-8)/2));
    //p2 = (unsigned char *)&result;
    //udph[6] =*p2;
    //udph[7] =*(p2+1);
    udph->udpchecknum = result;
}
void ch_ip_head(struct _ip_head *iph)
{
    int len, i=0;
    unsigned short result;
    unsigned short piandex;
    unsigned short excursion = 0;
    unsigned char tyhe;
    len = datalen+8+20;
    tyhe = 4;
    tyhe = tyhe<<4;
    tyhe += 5;
    iph->tyhead = tyhe;
    iph->diffser = 0;
    iph->iptotlelen = len;
    iph->identification = IDENTIFICATION;
    iph->headchecksum = 0;
    piandex = 2;
```

```
 piandex = piandex<<13;
 piandex += excursion;
 iph->piece_excu = piandex;
 iph->ttl = 1;
 iph->datatype = 17;
 unsigned char *ipbuf1 = (unsigned char *)iph;
 chartoshort(ipbuf1,20);
 /*for(i=0; i<10; i++)
 {
        printf("%x\n",buffer2[i]);
 }
 */
 result = ~(checksum(buffer2,10));
 iph->headchecksum = result;
 //printf("%x\n",result);
 //iph->headchecksum = EndianCovert(iph->headchecksum);
}

void ch_mac_head(struct _mac_head *mach)
{
 mach->destmac[4] = (char)(vlnum>>8);
 mach->destmac[5] = (char)(vlnum);
 //printf("\nmach->destmac[4]:%x\n",mach->destmac[4]);
 //printf("\nmach->destmac[5]:%x\n",mach->destmac[5]);
 //vlnum++;
}
void add_mac_head(unsigned char *mac_buf)
{
 int i;
 for(i = 0; i<MAC_HEAD_LEN; i++)
 {
        head_buf[i] = mac_buf[i];
 }
}
void add_ip_type(unsigned char *ip_type)
{
 int i;
 for (i = MAC_HEAD_LEN; i<MAC_HEAD_LEN+IP_TYPE_LEN; i++)
 {
        head_buf[i] = ip_type[i-MAC_HEAD_LEN];
 }
}
void add_ip_head(unsigned char *ip_buf)
{
```

```
 int i;
 for (i = MAC_HEAD_LEN+IP_TYPE_LEN; i<MAC_HEAD_LEN+IP_TYPE_LEN+IP_HEAD_LEN; i++)
 {
        head_buf[i] = ip_buf[i-MAC_HEAD_LEN-IP_TYPE_LEN];
 }
}
void add_udp_head(unsigned char *udp_buf)
{
 int i;
 for(i=MAC_HEAD_LEN+IP_TYPE_LEN+IP_HEAD_LEN;
      i<MAC_HEAD_LEN+IP_TYPE_LEN+IP_HEAD_LEN+UDP_HEAD_LEN; i++)
 {
        head_buf[i] = udp_buf[i-MAC_HEAD_LEN-IP_TYPE_LEN-IP_HEAD_LEN];
 }
}
static struct greth_softc *greth;
static unsigned char rxbuf[GRETH_RXBD_NUM*GRETH_RX_BUF_SIZE] = {1};
static unsigned char txbuf[GRETH_TXBD_NUM*GRETH_TX_BUF_SIZE] = {1};
static char rxbufr[1024*2] = {1};
static char txbufr[1024*2] = {1};
//中断处理函数
void greth_int_handle(int irq)
{
 unsigned char temp;
#ifdef SYS_SMP_SUPPORT
 disable_cpu_irq(CPU_HANDLE_GRETH_HANDLE,GRETH_IRQ_NO);
#ifdef PRINT_WHICH_CPU_HANDLE_IRQ
 printf("handle the greth interrupt at P%d\n",HAL_SMP_CPU_THIS());
#endif
#else
 disable_irq(GRETH_IRQ_NO);
#endif
 IRQ_CLEAR(GRETH_IRQ_NO);
 greth->regs->status = 0xffff;
 greth_read();
#ifdef SYS_SMP_SUPPORT
 enable_cpu_irq(CPU_HANDLE_GRETH_HANDLE,GRETH_IRQ_NO);
#else
 enable_irq(GRETH_IRQ_NO);
#endif
}
//启动 greth  中断
void greth_start_irq()
{
```

```
        catch_interrupt(greth_int_handle,GRETH_IRQ_NO);       //捕获中断
      #ifdef SYS_SMP_SUPPORT
        printf("\nstart greth test,smp support and set the p%d handle the greth interrupt %d\n",CPU_HANDLE_GRETH_HANDLE,GRETH_IRQ_NO);
        irq_set_cpu(CPU_HANDLE_GRETH_HANDLE,GRETH_IRQ_NO);   //开 CPU 中断
      #else
        enable_irq(GRETH_IRQ_NO);
        printf("***the cpu interrupt was open***");
        printf("\n");
      #endif
      }
      static void set_greth_regs(greth_regs *regs)
      {
       /* Clear all pending interrupts
          */
        GRETH_REGSAVE(regs->status,0);
        /* Enable receiver and transmiter
         */
        unsigned32 jittermax = 1200;
        unsigned32 tx_id;
        unsigned32 tx_regs;
        unsigned32 tx_vl_id = 0;// [11：0]发送端外部虚拟链路号
        unsigned32 tx_list_id = 0;// [18：12]发送端内部虚拟链路号
        unsigned32 tx_bag = 1; // [15：13]发送端相应虚拟链路 bag 值
        unsigned32 tx_rm = 0;   // [12：11]发送端相应虚拟链路冗余发送控制，00、11 冗余发送，01 只有 A 路发送，10 只有 B 路发送
        unsigned32 tx_lenmax = 1499;// [10：0]发送端相应虚拟链路的最大帧长
        unsigned32 rx_id;
        unsigned32 rx_regs;
        unsigned32 rx_vl_id = 0;
        unsigned32 rx_list_id = 0;
        unsigned32 rx_rm = 3;
        unsigned32 rx_skewmax = 65530;
        int i,j;
        GRETH_REGSAVE(regs->jittermax, jittermax);
        for (i = 0; i<128; i++ )
        {
         for (j=0; j<32; j++)
         {
                tx_id = tx_vl_id + (tx_list_id<<12);
                GRETH_REGSAVE(regs->afdx_tx_id, tx_id);
                tx_regs = tx_lenmax + (tx_rm<<11) + (tx_bag<<13);
                GRETH_REGSAVE(regs->afdx_tx_regs, tx_regs);
```

```
                rx_id = rx_vl_id + (rx_list_id<<12);
                GRETH_REGSAVE(regs->afdx_rx_id, rx_id);
                rx_regs = rx_skewmax + (rx_rm<<19);
                GRETH_REGSAVE(regs->afdx_rx_regs, rx_regs);
                //printf("regs->afdx_tx_id:0x%x\n",regs->afdx_tx_id);
                //printf("regs->afdx_rx_id:0x%x\n",regs->afdx_rx_id);
                tx_vl_id ++;
                rx_vl_id ++;
            }
            rx_list_id++;
            tx_list_id++;
           }
            printf("jittermax:0x%x\n", regs->jittermax);
      }
      static void set_greth_regs_temp(greth_regs *regs)
      {
        GRETH_REGSAVE(regs->status,0);
         /* Enable receiver and transmiter
          */
      //发送功能测试配置
      //6，7，8 路的 bag 值应相同，用于测试不同优先级的流量整形和调度
      //8 路和 10 路的 bag 值不同，用于测试相同优先级的流量整形和调度
      //11 路设置符合中断发送数据,网络号控制功能，SN 控制功能，冗余发送功能测试要求
         unsigned32 jittermax = 1200;
         unsigned32 tx_id;
         unsigned32 tx_regs;
         unsigned32 tx_vl_id = 0;       // [11：0]发送端外部虚拟链路号
         unsigned32 tx_list_id = 0;   // [18：12]发送端内部虚拟链路号
         unsigned32 tx_bag = 0;          // [15：13]发送端相应虚拟链路 bag 值
         unsigned32 tx_rm = 0;           // [12：11]发送端相应虚拟链路冗余发送控制，00、11 冗余发送，01 只有 A 路发送，
10 只有 B 路发送
         unsigned32 tx_lenmax = 1499;// [10：0]发送端相应虚拟链路的最大帧长
         unsigned32 rx_id;
         unsigned32 rx_regs;
         unsigned32 rx_vl_id = 0;
         unsigned32 rx_list_id = 0;
         unsigned32 rx_rm = 0;   //开接收冗余管理功能
         unsigned32 rx_skewmax = 65530;
         int i,j;
         GRETH_REGSAVE(regs->jittermax, jittermax);
         {
                tx_id = tx_vl_id + (tx_list_id<<12);
                GRETH_REGSAVE(regs->afdx_tx_id, tx_id);
```

```
            tx_regs = tx_lenmax + (tx_rm<<11) + (tx_bag<<13);
            GRETH_REGSAVE(regs->afdx_tx_regs, tx_regs);
            rx_id = rx_vl_id + (rx_list_id<<12);
            GRETH_REGSAVE(regs->afdx_rx_id, rx_id);
            rx_regs = rx_skewmax + (rx_rm<<19);
            GRETH_REGSAVE(regs->afdx_rx_regs, rx_regs);
    }
    tx_vl_id = 1;       // [11：0]发送端外部虚拟链路号
    tx_list_id = 1;    // [18：12]发送端内部虚拟链路号
    tx_bag = 0;          // [15：13]发送端相应虚拟链路 bag 值
{
            tx_id = tx_vl_id + (tx_list_id<<12);
            GRETH_REGSAVE(regs->afdx_tx_id, tx_id);
            tx_regs = tx_lenmax + (tx_rm<<11) + (tx_bag<<13);
            GRETH_REGSAVE(regs->afdx_tx_regs, tx_regs);
            rx_id = rx_vl_id + (rx_list_id<<12);
            GRETH_REGSAVE(regs->afdx_rx_id, rx_id);
            rx_regs = rx_skewmax + (rx_rm<<19);
            GRETH_REGSAVE(regs->afdx_rx_regs, rx_regs);
}
tx_vl_id = 2;       // [11：0]发送端外部虚拟链路号
tx_list_id = 2;    // [18：12]发送端内部虚拟链路号
tx_bag = 0;          // [15：13]发送端相应虚拟链路 bag 值
{
            tx_id = tx_vl_id + (tx_list_id<<12);
            GRETH_REGSAVE(regs->afdx_tx_id, tx_id);
            tx_regs = tx_lenmax + (tx_rm<<11) + (tx_bag<<13);
            GRETH_REGSAVE(regs->afdx_tx_regs, tx_regs);
            rx_id = rx_vl_id + (rx_list_id<<12);
            GRETH_REGSAVE(regs->afdx_rx_id, rx_id);
            rx_regs = rx_skewmax + (rx_rm<<19);
            GRETH_REGSAVE(regs->afdx_rx_regs, rx_regs);
}
tx_vl_id = 3;       // [11：0]发送端外部虚拟链路号
tx_list_id = 3;    // [18：12]发送端内部虚拟链路号
tx_bag = 0;          // [15：13]发送端相应虚拟链路 bag 值
{
            tx_id = tx_vl_id + (tx_list_id<<12);
            GRETH_REGSAVE(regs->afdx_tx_id, tx_id);
            tx_regs = tx_lenmax + (tx_rm<<11) + (tx_bag<<13);
            GRETH_REGSAVE(regs->afdx_tx_regs, tx_regs);
            rx_id = rx_vl_id + (rx_list_id<<12);
            GRETH_REGSAVE(regs->afdx_rx_id, rx_id);
```

```
                rx_regs = rx_skewmax + (rx_rm<<19);
                GRETH_REGSAVE(regs->afdx_rx_regs, rx_regs);
    }
    tx_vl_id = 4;       // [11：0]发送端外部虚拟链路号
    tx_list_id = 4;    // [18：12]发送端内部虚拟链路号
    tx_bag = 0;          // [15：13]发送端相应虚拟链路 bag 值
    {
                tx_id = tx_vl_id + (tx_list_id<<12);
                GRETH_REGSAVE(regs->afdx_tx_id, tx_id);
                tx_regs = tx_lenmax + (tx_rm<<11) + (tx_bag<<13);
                GRETH_REGSAVE(regs->afdx_tx_regs, tx_regs);
                rx_id = rx_vl_id + (rx_list_id<<12);
                GRETH_REGSAVE(regs->afdx_rx_id, rx_id);
                rx_regs = rx_skewmax + (rx_rm<<19);
                GRETH_REGSAVE(regs->afdx_rx_regs, rx_regs);
    }
       tx_vl_id = 5;       // [11：0]发送端外部虚拟链路号
       tx_list_id = 5;    // [18：12]发送端内部虚拟链路号
       tx_bag = 0;          // [15：13]发送端相应虚拟链路 bag 值
    {
                tx_id = tx_vl_id + (tx_list_id<<12);
                GRETH_REGSAVE(regs->afdx_tx_id, tx_id);
                tx_regs = tx_lenmax + (tx_rm<<11) + (tx_bag<<13);
                GRETH_REGSAVE(regs->afdx_tx_regs, tx_regs);
                rx_id = rx_vl_id + (rx_list_id<<12);
                GRETH_REGSAVE(regs->afdx_rx_id, rx_id);
                rx_regs = rx_skewmax + (rx_rm<<19);
                GRETH_REGSAVE(regs->afdx_rx_regs, rx_regs);
    }
    tx_vl_id = 6;       // [11：0]发送端外部虚拟链路号
    tx_list_id = 6;    // [18：12]发送端内部虚拟链路号
    tx_bag = 0;          // [15：13]发送端相应虚拟链路 bag 值
    {
                tx_id = tx_vl_id + (tx_list_id<<12);
                GRETH_REGSAVE(regs->afdx_tx_id, tx_id);
                tx_regs = tx_lenmax + (tx_rm<<11) + (tx_bag<<13);
                GRETH_REGSAVE(regs->afdx_tx_regs, tx_regs);
                rx_id = rx_vl_id + (rx_list_id<<12);
                GRETH_REGSAVE(regs->afdx_rx_id, rx_id);
                rx_regs = rx_skewmax + (rx_rm<<19);
                GRETH_REGSAVE(regs->afdx_rx_regs, rx_regs);
    }
    tx_vl_id = 7;       // [11：0]发送端外部虚拟链路号
```

```
tx_list_id = 7;   // [18：12]发送端内部虚拟链路号
tx_bag = 0;         // [15：13]发送端相应虚拟链路 bag 值
{
        tx_id = tx_vl_id + (tx_list_id<<12);
        GRETH_REGSAVE(regs->afdx_tx_id, tx_id);
        tx_regs = tx_lenmax + (tx_rm<<11) + (tx_bag<<13);
        GRETH_REGSAVE(regs->afdx_tx_regs, tx_regs);
        rx_id = rx_vl_id + (rx_list_id<<12);
        GRETH_REGSAVE(regs->afdx_rx_id, rx_id);
        rx_regs = rx_skewmax + (rx_rm<<19);
        GRETH_REGSAVE(regs->afdx_rx_regs, rx_regs);
}

   tx_vl_id = 8;      // [11：0]发送端外部虚拟链路号
   tx_list_id = 8;   // [18：12]发送端内部虚拟链路号
   tx_bag = 0;         // [15：13]发送端相应虚拟链路 bag 值
{
      tx_id = tx_vl_id + (tx_list_id<<12);
      GRETH_REGSAVE(regs->afdx_tx_id, tx_id);
      tx_regs = tx_lenmax + (tx_rm<<11) + (tx_bag<<13);
      GRETH_REGSAVE(regs->afdx_tx_regs, tx_regs);
      rx_id = rx_vl_id + (rx_list_id<<12);
      GRETH_REGSAVE(regs->afdx_rx_id, rx_id);
      rx_regs = rx_skewmax + (rx_rm<<19);
      GRETH_REGSAVE(regs->afdx_rx_regs, rx_regs);
}
tx_vl_id = 10;      // [11：0]发送端外部虚拟链路号
tx_list_id = 10;   // [18：12]发送端内部虚拟链路号
tx_bag = 7;         // [15：13]发送端相应虚拟链路 bag 值

{
     tx_id = tx_vl_id + (tx_list_id<<12);
     GRETH_REGSAVE(regs->afdx_tx_id, tx_id);
     tx_regs = tx_lenmax + (tx_rm<<11) + (tx_bag<<13);
     GRETH_REGSAVE(regs->afdx_tx_regs, tx_regs);
     rx_id = rx_vl_id + (rx_list_id<<12);
     GRETH_REGSAVE(regs->afdx_rx_id, rx_id);
     rx_regs = rx_skewmax + (rx_rm<<19);
     GRETH_REGSAVE(regs->afdx_rx_regs, rx_regs);
}
tx_vl_id = 11;      // [11：0]发送端外部虚拟链路号
tx_list_id = 11;   // [18：12]发送端内部虚拟链路号
tx_bag = 7;         // [15：13]发送端相应虚拟链路 bag 值
```

```
    {
        tx_id = tx_vl_id + (tx_list_id<<12);
        GRETH_REGSAVE(regs->afdx_tx_id, tx_id);
        tx_regs = tx_lenmax + (tx_rm<<11) + (tx_bag<<13);
        GRETH_REGSAVE(regs->afdx_tx_regs, tx_regs);
        rx_id = rx_vl_id + (rx_list_id<<12);
        GRETH_REGSAVE(regs->afdx_rx_id, rx_id);
        rx_regs = rx_skewmax + (rx_rm<<19);
        GRETH_REGSAVE(regs->afdx_rx_regs, rx_regs);
    }


    //接收功能测试配置
    //第 12 路和第 14 路用于测试 SKEWMAX 功能，两路都开启冗余管理功能，12 路的 skewmax 设为 0,14 路的 skewmax 设为 65530
    rx_vl_id = 12;      // [11：0]发送端外部虚拟链路号
    rx_list_id = 12;   // [18：12]发送端内部虚拟链路号
    rx_rm = 0; //开启接收冗余管理功能
    rx_skewmax = 0;

    {
            tx_id = tx_vl_id + (tx_list_id<<12);
        GRETH_REGSAVE(regs->afdx_tx_id, tx_id);
        tx_regs = tx_lenmax + (tx_rm<<11) + (tx_bag<<13);
        GRETH_REGSAVE(regs->afdx_tx_regs, tx_regs);
        rx_id = rx_vl_id + (rx_list_id<<12);
        GRETH_REGSAVE(regs->afdx_rx_id, rx_id);
        rx_regs = rx_skewmax + (rx_rm<<19);
        GRETH_REGSAVE(regs->afdx_rx_regs, rx_regs);
    }

    rx_vl_id = 14;      // [11：0]发送端外部虚拟链路号
    rx_list_id = 14;   // [18：12]发送端内部虚拟链路号
    rx_rm = 0; //开启接收冗余管理功能
    rx_skewmax = 65530;

    {
            tx_id = tx_vl_id + (tx_list_id<<12);
        GRETH_REGSAVE(regs->afdx_tx_id, tx_id);
        tx_regs = tx_lenmax + (tx_rm<<11) + (tx_bag<<13);
        GRETH_REGSAVE(regs->afdx_tx_regs, tx_regs);
        rx_id = rx_vl_id + (rx_list_id<<12);
```

```
            GRETH_REGSAVE(regs->afdx_rx_id, rx_id);
            rx_regs = rx_skewmax + (rx_rm<<19);
            GRETH_REGSAVE(regs->afdx_rx_regs, rx_regs);
 }
//第 20 路和第 21 路用于测试冗余管理功能，20 路开启冗余管理功能，21 路关闭冗余管理功能
 rx_vl_id = 20;       // [11：0]发送端外部虚拟链路号
 rx_list_id = 20;    // [18：12]发送端内部虚拟链路号

 rx_rm = 0; //开启接收冗余管理功能
 rx_skewmax = 65530;

 {
            tx_id = tx_vl_id + (tx_list_id<<12);
            GRETH_REGSAVE(regs->afdx_tx_id, tx_id);
            tx_regs = tx_lenmax + (tx_rm<<11) + (tx_bag<<13);
            GRETH_REGSAVE(regs->afdx_tx_regs, tx_regs);
            rx_id = rx_vl_id + (rx_list_id<<12);
            GRETH_REGSAVE(regs->afdx_rx_id, rx_id);
            rx_regs = rx_skewmax + (rx_rm<<19);
            GRETH_REGSAVE(regs->afdx_rx_regs, rx_regs);
 }
 rx_vl_id = 21;       // [11：0]发送端外部虚拟链路号
 rx_list_id = 21;    // [18：12]发送端内部虚拟链路号
 rx_rm = 3; //开启接收冗余管理功能
 rx_skewmax = 65530;
 {
            tx_id = tx_vl_id + (tx_list_id<<12);
            GRETH_REGSAVE(regs->afdx_tx_id, tx_id);
            tx_regs = tx_lenmax + (tx_rm<<11) + (tx_bag<<13);
            GRETH_REGSAVE(regs->afdx_tx_regs, tx_regs);
            rx_id = rx_vl_id + (rx_list_id<<12);
            GRETH_REGSAVE(regs->afdx_rx_id, rx_id);
            rx_regs = rx_skewmax + (rx_rm<<19);
            GRETH_REGSAVE(regs->afdx_rx_regs, rx_regs);
 }
}
static void set_greth_regs_redundancy(greth_regs *regs)//冗余测试配置函数
{
  GRETH_REGSAVE(regs->status,0);
    /* Enable receiver and transmiter
      */
    unsigned32 jittermax = 1200;
    unsigned32 tx_id;
```

```
        unsigned32 tx_regs;
        unsigned32 tx_vl_id = 8;       // [11：0]发送端外部虚拟链路号
        unsigned32 tx_list_id = 8;   // [18：12]发送端内部虚拟链路号
        //unsigned32 tx_bag_temp[5] = {1,7,1,7,3};
        unsigned32 tx_bag = 7;          // [15：13]发送端相应虚拟链路 bag 值
        unsigned32 tx_rm = 3;           // [12：11]发送端相应虚拟链路冗余发送控制，00、11 冗余发送，01 只有 A 路发送，
10 只有 B 路发送
        unsigned32 tx_lenmax = 1499;// [10：0]发送端相应虚拟链路的最大帧长
        unsigned32 rx_id;
        unsigned32 rx_regs;
        unsigned32 rx_vl_id = 0;
        unsigned32 rx_list_id = 0;
        unsigned32 rx_rm = 0;
        unsigned32 rx_skewmax = 65530;
        int i,j;
        GRETH_REGSAVE(regs->jittermax, jittermax);
        {
                //tx_bag = tx_bag_temp[i];
                tx_id = tx_vl_id + (tx_list_id<<12);
                GRETH_REGSAVE(regs->afdx_tx_id, tx_id);
                tx_regs = tx_lenmax + (tx_rm<<11) + (tx_bag<<13);
                GRETH_REGSAVE(regs->afdx_tx_regs, tx_regs);
                rx_id = rx_vl_id + (rx_list_id<<12);
                GRETH_REGSAVE(regs->afdx_rx_id, rx_id);
                rx_regs = rx_skewmax + (rx_rm<<19);
                GRETH_REGSAVE(regs->afdx_rx_regs, rx_regs);
        }
    }
    static int read_mii_a(int addr, volatile greth_regs *regs)
    {
        while (GRETH_REGLOAD(regs->a_mdio) & GRETH_MII_BUSY) {}
        GRETH_REGSAVE(regs->a_mdio, (5 << 11) | ((addr&0x1F) << 6) | 2);
        while (GRETH_REGLOAD(regs->a_mdio) & GRETH_MII_BUSY) {}
        if (!(GRETH_REGLOAD(regs->a_mdio) & GRETH_MII_NVALID))
        {
            return (GRETH_REGLOAD(regs->a_mdio)>>16)&0xFFFF;
        }
        else
        {
            return -1;
        }
    }
    static int read_mii_b(int addr, volatile greth_regs *regs)
```

```
{
 while (GRETH_REGLOAD(regs->b_mdio) & GRETH_MII_BUSY) {}
 GRETH_REGSAVE(regs->b_mdio, (3 << 11) | ((addr&0x1F) << 6) | 2);
 while (GRETH_REGLOAD(regs->b_mdio) & GRETH_MII_BUSY) {}
 if (!(GRETH_REGLOAD(regs->b_mdio) & GRETH_MII_NVALID))
 {
        return (GRETH_REGLOAD(regs->b_mdio)>>16)&0xFFFF;
 }
 else
 {return -1;}
}
static void write_mii(int addr, int data, volatile greth_regs *regs)
{
 //while (GRETH_REGLOAD(regs->a_mdio) & GRETH_MII_BUSY) {}
   GRETH_REGSAVE(regs->a_mdio, ((data&0xFFFF)<<16) | (5 << 11) | ((addr&0x1F) << 6) | 1);
   while (GRETH_REGLOAD(regs->a_mdio) & GRETH_MII_BUSY) {}
   GRETH_REGSAVE(regs->b_mdio, ((data&0xFFFF)<<16) | (3 << 11) | ((addr&0x1F) << 6) | 1);
   while (GRETH_REGLOAD(regs->b_mdio) & GRETH_MII_BUSY) {}
}
/* Initialize the ethernet hardware */
static greth_initialize_hardware()
{
 int i, tmp = 0x100;
 int tmp1 = 0x100;
 greth_regs *regs = greth->regs;
 volatile greth_bd *tx_bd, *rx_bd;
 unsigned long txmem_addr = (unsigned long)&(txbuf);
 unsigned long rxmem_addr = (unsigned long)&(rxbuf);
   /* Reset the controller.    */
 GRETH_REGSAVE(regs->control,GRETH_RESET);
 for (i=0;i<32 ;i++ )
 {
        printf("wait");
        if (i % 16 == 15) { printf("\n");}
 }
 printf ("read two phy reg\n ");
 while (GRETH_REGLOAD(regs->control) & GRETH_RESET) {}
   /* Configure PHY    */
 read_mii_a(0,regs);
 read_mii_b(0,regs);
 //write_mii(0, 0xa100, regs);
 write_mii(0, 0x8100, regs);
 //write_mii(0, 0x6100, regs);
   for (i=0;i<32 ;i++ )//???
```

```
    {
     tmp = read_mii_a(i,regs);
     printf ("phy_a_5_reg[%x]: %x\n ",i,tmp);
     tmp1 = read_mii_b(i,regs);
     printf ("phy_b_3_reg[%x]: %x\n ",i,tmp1);
    }
    /* Initialize TXBD pointer      */
  greth->tx_bd_base = (volatile greth_bd *)((((unsigned int)&txbufr) + (1024-1)) & ~(1024-1));
  GRETH_REGSAVE(regs->tx_desc_p,(unsigned32)greth->tx_bd_base);
  tx_bd = (volatile greth_bd *) greth->tx_bd_base;
  printf("tx_bd:0x%x\n",tx_bd);
    /* Initialize RXBD pointer      */
  greth->rx_bd_base = (volatile greth_bd *)((((unsigned int)&rxbufr) + (1024-1)) & ~(1024-1));
  GRETH_REGSAVE(regs->rx_desc_p,(unsigned32)(greth->rx_bd_base));
  rx_bd = (volatile greth_bd *) greth->rx_bd_base;
  printf(" rx_bd:0x%x\n",rx_bd);
    /* Initialize pointers      */
  greth->rx_cur = 0;
  greth->tx_next = 0;
  greth->tx_last = 0;
  greth->tx_full = 0;
    /* Initialize TXBDs   */
  for(i = 0; i < GRETH_TXBD_NUM; i++)
  {
        GRETH_REGSAVE(tx_bd[i].stat, GRETH_BD_EN);
        GRETH_REGSAVE(tx_bd[i].addr, txmem_addr);
        txmem_addr += GRETH_TX_BUF_SIZE;
  }
        GRETH_REGORIN(tx_bd[GRETH_TXBD_NUM - 1].stat,GRETH_BD_WR);
    /* Initialize RXBDs.      */
  for(i = 0; i < GRETH_RXBD_NUM; i++)
  {
        GRETH_REGSAVE(rx_bd[i].stat, GRETH_BD_EN);
        GRETH_REGSAVE(rx_bd[i].addr,rxmem_addr);
        rxmem_addr += GRETH_RX_BUF_SIZE;
  }
        GRETH_REGORIN(rx_bd[GRETH_RXBD_NUM - 1].stat,GRETH_BD_WR);

  //GRETH_REGORIN(regs->control, GRETH_RXEN | GRETH_TXEN);
  //GRETH_REGORIN(regs->control, GRETH_CR_RI | GRETH_CR_TI);
  GRETH_REGORIN(regs->control, GRETH_RXEN | GRETH_TXEN);
  GRETH_REGORIN(regs->control, GRETH_CR_RI);
  //greth_start_irq();
 }
```

```
static int arp_ticks = 2;
unsigned int greth_read(void)
{
        //z printf("greth_read\n");
        recevnum++;
        //z printf("recever number is :%d\n\n",recevnum);
        unsigned char *recvhead = (unsigned char *)&head;
        unsigned char *recvdata;
        unsigned int bad,stat,len;
        unsigned char flags;
        unsigned short flagment;
        unsigned char *addr;
        int val1, val2, i, cur;
        val1 = 2;
        val2 = 1;
        cur = greth->rx_cur;
        stat = GRETH_REGLOAD(greth->rx_bd_base[cur].stat);
        //z printf("rx_bd_base:(0x%x)\n", greth->rx_bd_base);
        //z printf("cur:%d stat :(0x%x)\n", cur, stat);
        if (!((stat = GRETH_REGLOAD(greth->rx_bd_base[cur].stat)) & GRETH_BD_EN))
        {
                //z printf("cur:%d stat :(0x%x)\n", cur, stat);
                bad = 0;
                len = ((stat & GRETH_BD_LEN)<(UIP_BUFSIZE)?(stat & GRETH_BD_LEN):(UIP_BUFSIZE));
                // Check status for errors.
                if (stat & GRETH_RXBD_ERR_FT)
                {
                        greth->rx_length_errors++;
                        bad = 1;
                }
                if (stat & (GRETH_RXBD_ERR_AE | GRETH_RXBD_ERR_OE))
                {
                        greth->rx_frame_errors++;
                        bad = 1;
                }
                if (stat & GRETH_RXBD_ERR_CRC)
                {
                        greth->rx_crc_errors++;
                        bad = 1;
                }
                GRETH_REGANDIN(greth->rx_bd_base[cur].stat , ~GRETH_RXBD_STATUS);
                GRETH_REGORIN(greth->rx_bd_base[cur].stat, GRETH_BD_EN | ((cur ==
GRETH_RXBD_NUM_MASK) ? GRETH_BD_WR : 0));
```

```
GRETH_REGORIN(greth->regs->control,GRETH_RXEN);
greth->rx_cur = (greth->rx_cur + 1) % GRETH_RXBD_NUM;
if (!bad)
{
    addr = (unsigned char *)greth->rx_bd_base[cur].addr;
    //z printf(">");
    //z printf("Rx packet:0x%x (%d)\n",addr,len);

    //z printf("greth->rx_bd_base_p:0x%x \n",greth->rx_bd_base);
  /*
    for(i=0; i<42; i++)
    {
        recvhead[i] = addr[i];
    }
    for(; i<len; i++)
    {
        recvdata[i] = addr[i];
    }
    flags = (unsigned char)(head.piece_excu>>13);
    flagment = head.piece_excu & 0x1fff;
    printf("Flags: %02x\n",flags);
    printf("Fragment offset: %02x\n",flagment);
    printf("...................Ethernet II.............\n");
    printf("Destination Mac: ");
    for (i=0; i<6; i++)
    {
        printf("%02x",head.destmac[i]);
        printf(":");
    }
    printf("\n");
    printf("Source Mac: ");
    for (; i<12; i++)
    {
        printf("%02x",head.sourcemac[i-6]);
        printf(":");
    }
    printf("\n");
    printf("Type: ");
    for (; i<14; i++)
    {
        printf("%02x",head.iptype[i-12]);
    }
    printf("\n\n");
    printf("...................Internet Protocol.............\n");
```

```
printf("Version: 4 \n");
printf("Header Length: 20 bytes \n");
printf("Type of Server: 0x00 \n");
printf("Total Length: %d\n",head.iptotlelen);
printf("Identification: %d\n",head.identification);
printf("Time to Live: %d",head.ttl);
printf("Protocol: %d\n",head.datatype);
printf("Header Checksum: %d\n",head.headchecksum);
printf("Ip Source Addr: ");
for (i=30; i<34; i++)
{
    printf("%02x",head.sourceip[i-30]);
    printf(":");
}
printf("\n");
printf("Ip Destination Addr: ");
for (i=34; i<38; i++)
{
    printf("%02x",head.destip[i-34]);
    printf(":");
}
printf("\n\n");
printf(".....User Datagram Protocol.....\n");
printf("Source Port: %d\n",head.sourceport);
printf("Dest Port: %d\n",head.destport);
printf("Udp len: %d\n",head.udplen);
printf("Udp CheckSum: %d",head.udpchecknum);
printf("\n\n");
for (i=0; i<len-42; i++)
{
    printf("%02x ",recvdata[i]);
    if (i % 16 == 15)
    {
        printf("\n");
    }
}
*/
printf("\n");
//for (i = 0; i < len; i++)
for (i = 0; i < 12; i++)
{
     printf("%02x ",addr[i]);
    //if (i % 16 == 15)
    //{
```

```
                                        //z printf("\n");
                                //}
                        }
                          printf("\n");
                          printf("%02x ",addr[len-1]);
/*
                        snnumsecond = addr[len-1]-snnumfirst;
                        if(snnumsecond != 1)
                        {
                                printf("%02x ",addr[len-1]);
                        }
                        snnumfirst = addr[len-1];
*/
                        //z printf("\n");
                        greth->rx_packets++;
                }
        }
        return 0;
 }
 void greth_send(int uip_len, char *uip_buf, char *uip_appdata, int wait)
 {
        unsigned char *addr;
        unsigned int len,stat,i,cur,j;
        unsigned32 extra_vl;
        unsigned32 mac_vl1,mac_vl2,mac_vl3;
        unsigned char *mac_buf = (unsigned char *)&mac_head;
        datalen = uip_len;
        struct _udp_fake_head *udp_fake_buf = &fake_head;
        struct _udp_head *udp_buf = &udp_head;
        struct _ip_head *ip_headp = &ip_head;
        struct _mac_head *mach =&mac_head;
        cur = greth->tx_next;
        // wait for previous to finish
        while (GRETH_REGLOAD(greth->tx_bd_base[cur].stat) & GRETH_BD_EN) ;
        // don't send long packets
        if ( uip_len <= GRETH_TX_BUF_SIZE )
        {
                addr = (unsigned char *)greth->tx_bd_base[cur].addr;
                // change the heads
                ch_udp_head(udp_fake_buf, udp_buf, uip_buf);
                ch_ip_head(ip_headp);
                ch_ip_head(ip_headp);
                ch_mac_head(mach);
```

```
                //add_package_head(mac_buf, iptype, ip_headp, udp_buf);
                add_mac_head(mac_buf);
                add_ip_type(iptype);
                add_ip_head(ip_headp);
                add_udp_head(udp_buf);
                for (i=0; i<TOTAL_HEAD_LEN;i++)
                {
                        addr[i] = head_buf[i];
                }
                for(j=0; i < uip_len+TOTAL_HEAD_LEN;i++,j++)
                {
                        addr[i] = uip_buf[j];
                }

                //      deside the vl from the mac
                mac_vl1 = addr[4];
                mac_vl2 = addr[5];
                mac_vl1 = mac_vl1<<8;
                mac_vl3 = mac_vl1+mac_vl2;
                /*
                printf("mac_vl1:0x%x\n",mac_vl1);
                printf("mac_vl2:0x%x\n",mac_vl2);
                printf("mac_vl3:0x%x\n",mac_vl3);
                */
                extra_vl = mac_vl3%4096;
                extra_vl = extra_vl<<14;
                //printf("extra_vl:0x%x\n",extra_vl);
        /*
                printf("<");
                printf("Tx packet[%d]:0x%x (len:%d)\n",cur,addr,uip_len);
                for (i = 0; i < uip_len+TOTAL_HEAD_LEN; i++)
                {
                        printf("%02x ",addr[i]);
                        if (i % 16 == 15)
                        {
                                printf("\n");
                        }
                }
        */              // Clear all of the status flags.
                stat = GRETH_REGANDIN(greth->tx_bd_base[cur].stat,~GRETH_TXBD_STATUS);

                // write buffer descriptor length and status
                len = uip_len+TOTAL_HEAD_LEN;
                stat = (len & GRETH_BD_LEN) | GRETH_EN_RX_IRQ | GRETH_BD_EN | ((cur ==
```

```
GRETH_TXBD_NUM_MASK) ? GRETH_BD_WR |extra_vl    : extra_vl);
                    GRETH_REGORIN(greth->tx_bd_base[cur].stat,stat);
                    GRETH_REGORIN(greth->regs->control,GRETH_TXEN);
                    //printf("greth->tx_bd_base[cur].stat:0x%x\n",greth->tx_bd_base[cur].stat);
                    // wait for this one to finish
                    if (wait)
                    while (GRETH_REGLOAD(greth->tx_bd_base[cur].stat) & GRETH_BD_EN) ;
                    greth->tx_packets++;
                    greth->tx_next = (greth->tx_next + 1) % GRETH_TXBD_NUM;
                    //printf("\n\ngreth->tx_next:%d\n\n",greth->tx_next);
            }
    }
    static greth_init(struct greth_softc *cfg)
    {
            greth = cfg;
            // Initialize the device driver.
            greth_initialize_hardware();
    }
    static char txdata_vl[500] = {0, 1, 2, 3, 0, 0, 6, 7, 8, 9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
    0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf, 0, 1, 2, 3, 4, 5, 0, 0xa2,
    8, 9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf,
    0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xf};
    void delay(int a)
            {
                    int i, j,k;
                    for(k=0; k<a; k++)
                    {
                            for(i=0; i<3; i++)
                            {
                                    for(j=0; j<1111; j++);
                            }
                    }
            }
    //中断发送数据,网络号控制功能，SN 控制功能，冗余发送功能测试函数
    void int_send()
    {
     //greth_regs *regs = greth->regs;
     //set_greth_regs_temp(regs);
     while ((GRETH_REGLOAD(greth->regs->status) & GRETH_INT_TX) )
     {
            greth->regs->status = 0x8;
            vlnum = 11;
            delay(80);
            greth_send(10, &txdata, &txdata[16], 0);
```

```
 }
}
//流量整形和相同优先级调度测试函数
void bag_scheduing()
{
 while ((GRETH_REGLOAD(greth->regs->status) & GRETH_INT_TX) )
 {
        greth->regs->status = 0xffff;
        //if(vlnum == 7)
        vlnum = 8;
        delay(3);
        greth_send(7, &txdata, &txdata[16], 0);
        break;
 }
 while ((GRETH_REGLOAD(greth->regs->status) & GRETH_INT_TX) )
 {
        greth->regs->status = 0xffff;
        vlnum = 10;
        delay(3);
        greth_send(7, &txdata, &txdata[16], 0);
        break;
 }
}
//流量整形和不同优先级调度测试函数

void bag_pri_scheduing()
{
 //while ((GRETH_REGLOAD(greth->regs->status) & GRETH_INT_TX) )
 {
        //greth->regs->status = 0xffff;
        vlnum = 0;
        greth_send(224, &txdata_pri, &txdata[16], 0);
        vlnum = 1;
        greth_send(224, &txdata_pri, &txdata[16], 0);
        vlnum = 2;
        greth_send(224, &txdata_pri, &txdata[16], 0);
        vlnum = 3;
        greth_send(224, &txdata_pri, &txdata[16], 0);
        vlnum = 4;
        greth_send(224, &txdata_pri, &txdata[16], 0);
        vlnum = 5;
        greth_send(224, &txdata_pri, &txdata[16], 0);
        vlnum = 6;
        greth_send(224, &txdata_pri, &txdata[16], 0);
```

```
            vlnum = 7;
            greth_send(224, &txdata_pri, &txdata[16], 0);
            vlnum = 8;
            greth_send(224, &txdata_pri, &txdata[16], 0);
            //vlnum = 8;
            //break;
    }
    //while ((GRETH_REGLOAD(greth->regs->status) & GRETH_INT_TX) )
    {
            //greth->regs->status = 0xffff;
            //vlnum = 8;
            //delay(4);
            //greth_send(224, &txdata_pri, &txdata[16], 0);
            //vlnum = 7;
            //break;
    }
    //while ((GRETH_REGLOAD(greth->regs->status) & GRETH_INT_TX) )
    {
            //greth->regs->status = 0xffff;
            //delay(4);
            //greth_send(7, &txdata, &txdata[16], 0);
            //vlnum = 7;
            //break;
    }
}
////中断接收功能测试函数,完整性检查功能测试函数
void int_recv()
{
 greth_start_irq();
 while(1)
 {
 }
}
//冗余管理功能测试函数
void redundancy_manage()
{
 greth_start_irq();
 while(1)
 {
 }
}
//SKEWMAX 功能测试函数
void skewmax()
```

```
{
 greth_start_irq();
 while(1)
 {
 }
}
int main()
{
 //初始化 S698P4 配置寄存器
 MEM_CONFIG1_REG=0xe8f809ff;
 MEM_CONFIG2_REG=0xb801063;
 printf("***AFDX TEST-- INIT start ***\n");
 greth_config.regs = 0x80000f00;
 greth_init(&greth_config);
 printf("***AFDX TEST-- INIT end ***\n");
 int i,j;
 vlnum = 10;
 greth_send(7, &txdata, &txdata[16], 0);
 //vlnum = 8;
 while(1)
 {
        //int_send();//中断发送数据,网络号控制功能，SN 控制功能，冗余发送功能测试函数(用第 11 路发送)
        //bag_scheduing();//流量整形和相同优先级调度测试函数
        bag_pri_scheduing();//流量整形和不同优先级调度测试函数
        //int_recv();//中断接收功能测试函数,完整性检查功能测试函数（要求发送端以第 20 路发送）
        //redundancy_manage();//冗余管理功能测试函数（要求分别以第 20 路和第 21 路发送数据进行测试，第 20 路开启了接收冗余管理功能，第 21 路关闭了接收冗余管理功能）
        //skewmax();//SKEWMAX 功能测试函数(要求分别以第 12 路和第 14 路发送数据进行测试,第 12 路的 skewmax 设为 0,第 14 路的 skewmax 设为 65530）
 }
 return 0;
}
```

5.2 附录二：OBT-AFDX 外围电路原理图

附录二为超链接文件：OBT-AFDX外围电路原理图。