IRSV-Net 红外图像采集卡

开发者手册 v1.0

北京红谱威视图像技术有限公司

2015 年 8 月

手册目录

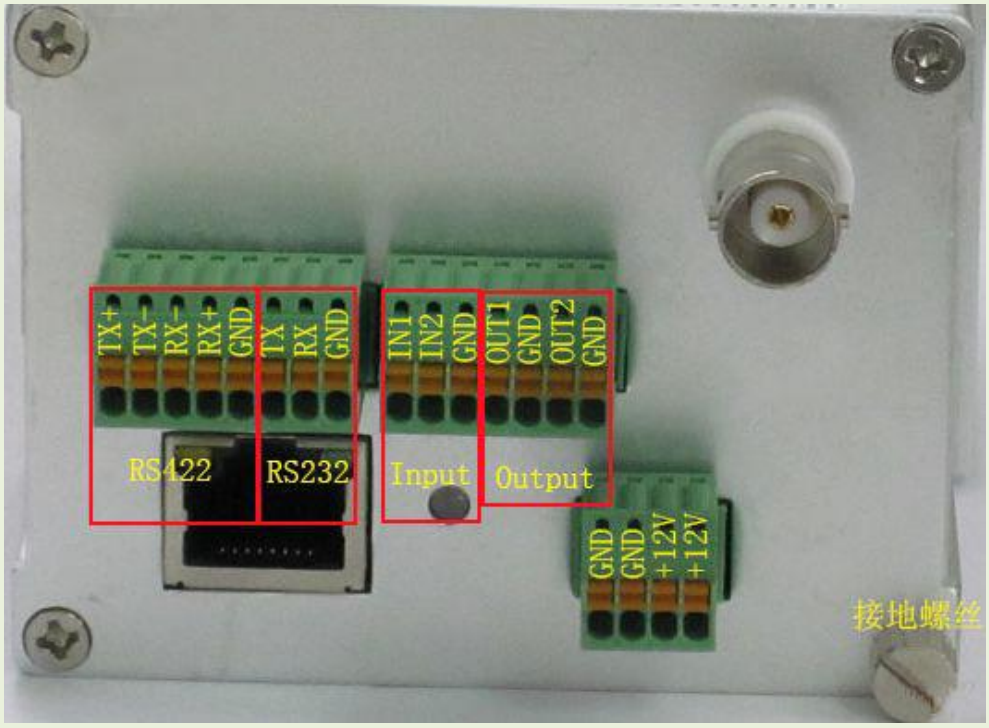
系统概述：

IRSV-Net 型红外相机采集卡设备连接Flir 的Photon 或者Tau 相机，使用其LVDS 总线采集红外相机的图像，然后通过网络传输给用户。

一、 系统连接：

1、硬件接口：

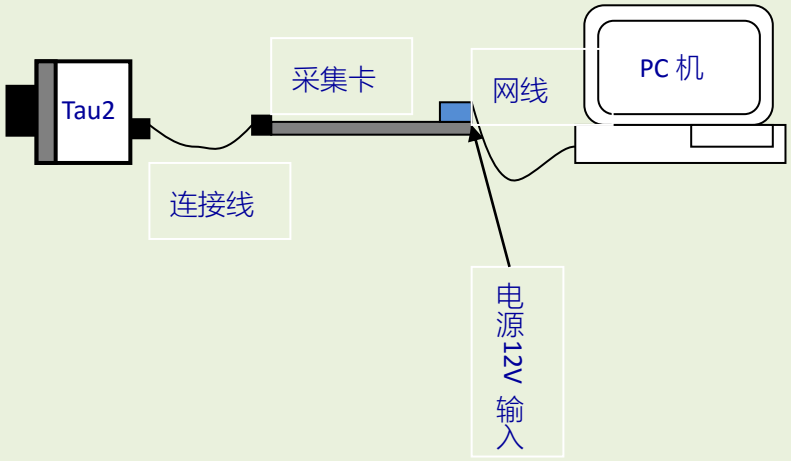
设备使用12V、1A 直流电源供电：下图左至右的顺序：GND、GND、+12V、+12V；
提供2 入2 出光隔离IO 信号：下图左至右的顺序：IN1、IN2、GND；
提供1 路隔离RS422 接口：下图左至右的顺序：TX+、TX-、RX-、RX+、GND；
提供1 路隔离RS232 接口：下图左至右的顺序：TX、RX、GND；
提供100M 网络传输图像和控制信号；



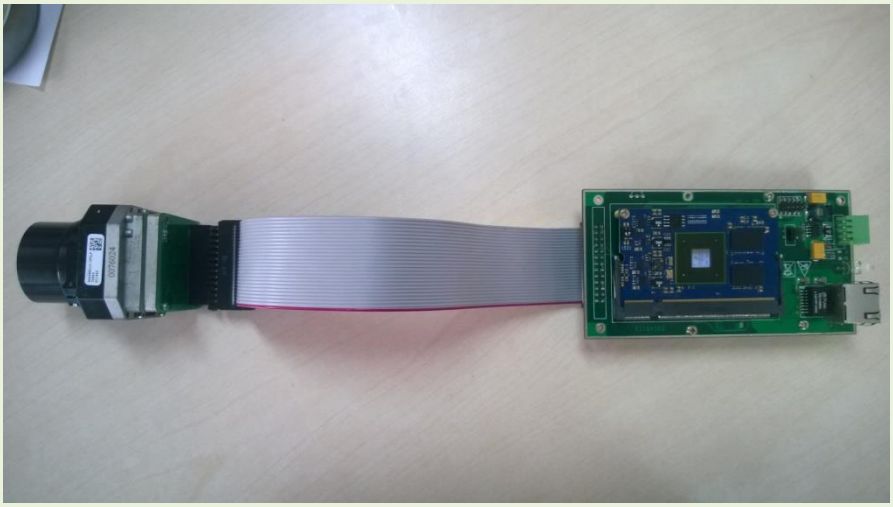
后端接口图

注意：其中，IN1 和 IN2 共地。

2、连接图：



实物连接图：



警告：注意红外机芯和采集卡的排线顺序。不能弄错！

3、**IRSV-Net** 采集卡内部构架与接口原理：

（**1**）、硬件接口顺序：

Flir Tau2 -> Altera EP4C FPGA -> Cortex A9 工业级 ARM 处理器->网络->Windows 系统

（**2**）、接口流程：

设备前端连接 Flir 的 Tau2 系列红外相机（自适应连接 Tau2 168、Tau2 324、Tau2 336、Tau2 640 系列相机），使用抗干扰能力强、高可靠性的 LVDS 信号，采集红外相机的 8 位或者 14 位数字图像，经过 FPGA 解析，把图像转换为 Cortex A9 工业级 ARM CPU 识别的并行图像信号，输出到 CPU。

CPU 上运行嵌入式 Linux 系统，通过 V4L2 视频驱动程序，采集 FPGA 传输的 8 位或者 14 位数字图像，然后通过网络服务程序，传输到 windows 客户端。

Linux 运行 2 个 server 程序，一个 UDP server 程序用户查找设备 IP 地址、修改设备 IP 地址等；一个是 TCP server 程序，用于图像数据的传输、Flir 红外相机的控制等。

Windows 客户端运行客户端网络程序，与设备上的 server 程序通讯，用多缓冲区机制实现图像的获取、相机的控制功能。上述功能封装为 C 用户的 SDK 库，提供用户开发手册。

Demo 程序使用 SDK 函数，实时接收 8 位或者 14 位红外图像，并把图像显示出来。如果是 14 位图像 ,Demo 程序使用平台直方图算法把 14 位图像转换为 8 位图像并显示。

Demo 同时具有保存 14 位原始 RAW 数据和 8 位 BMP 图像的功能。

二、 Demo 软件操作：

1、安装**:**

把IRSV-Net_Demo 目录拷贝到Windows 系统下的任意目录即可。

IRSV-Net_Demo 目录包括下面6 个文件和3 个目录：

irCamDemo.exe

irCamTool.exe

ir_cam.dll

pthreadVC2.dll

qImgLib.dll

pst.ini

\para

\SaveImg_8

\SaveImg_14

2、设置计算机 **ip** 值：

设备的出厂IP 地址默认是192.168.0.201，用户根据自己的系统IP 设置情况来修改设备的IP 地址，要求与系统中的其它设备的IP 地址不一样。修改IP 地址的方法：

（1）、设置 Windows 系统的 IP 地址为 192.168.0.xxx；

3、双击**“irCamDemo.exe”**图标，即可进入界面：

4、**IRSV-Net** 图像采集卡后面板指示灯状态说明：

状态	意义
红色	设备上电
绿色	内部 CPU 系统正常启动
绿色闪烁	系统识别出红外相机，开始正常采集图像

5、**Demo** 软件功能介绍：

具体内容见“IRSV-Net图像采集卡手册v1.1”，示例程序为” base_demo_JPEG” ，光盘中提供该示例程序源码，供开发者参考。编程环境”Visual C++ 2012”。

注意：提供 Demo 程序的版本不同，可能功能略有差异！

三、 IRSV-Net 软件开发包文档（VC++）

1、说明

红外相机处理设备控制FIir 的Photon 或者Tau 相机，使用其LVDS 总线采集红外相机的图像，然后通过网络传输给用户。用户使用本文档描述的SDK 可以对红外相机进行控制、获取图像。

对有需求的用户，红外相机处理设备还包含了2 入2 出的光隔离IO，以及隔离的RS232 串口和隔离的RS422 接口，使用本SDK 可以对这些接口进行控制。

2、SDK 包含的文件

头文件

ir_types.h 基本数据类型的定义

ir_errinfo.h 错误信息的定义

ir_cam.h 控制码、数据结构、函数的定义

FlirCam_common.h 控制红外相机的数据结构

VC 编译库

ir_cam.lib

运行库

ir_cam.dll

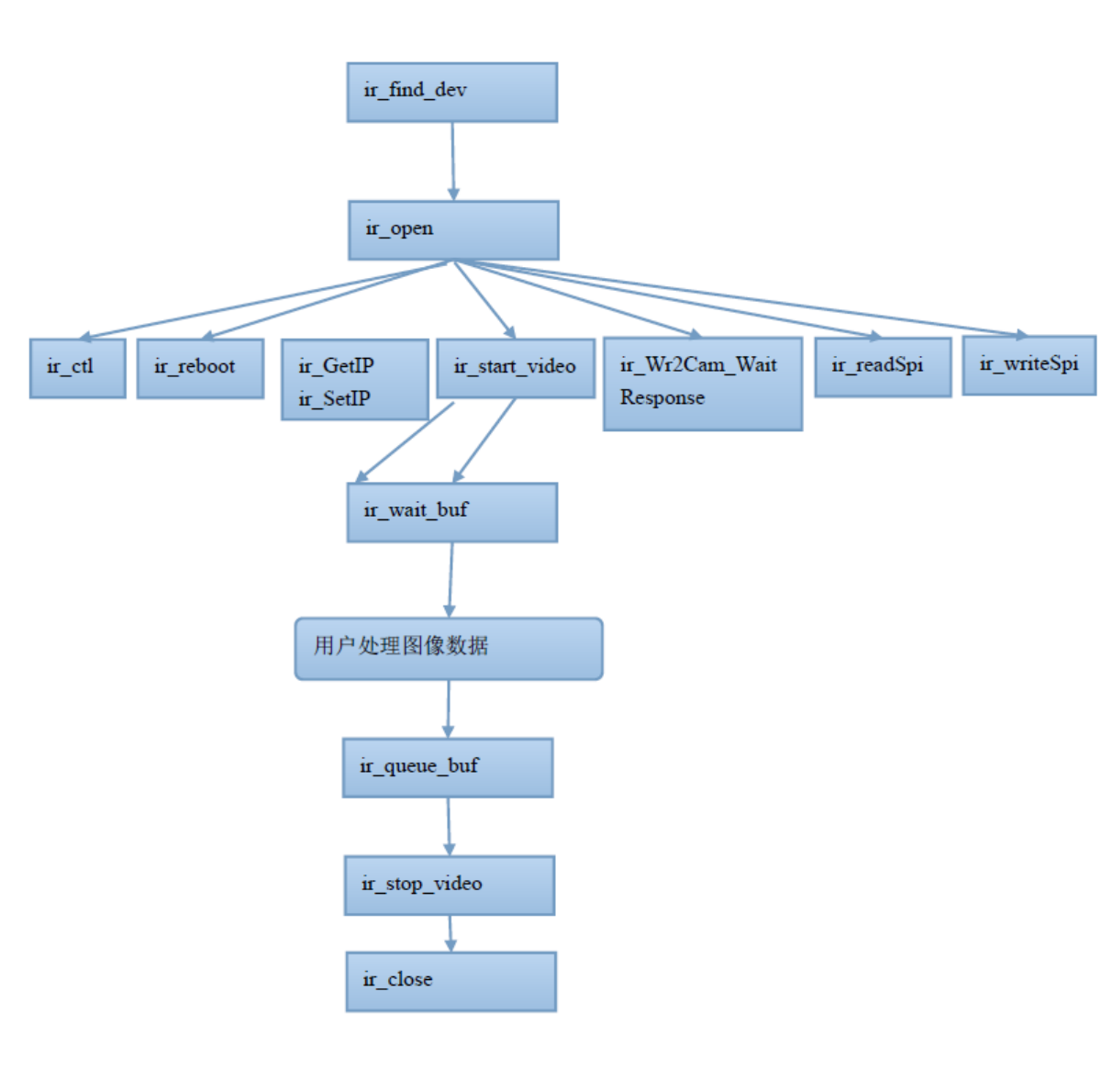
pthreadVC2.dll

3、操作相机的基本步骤

查找设备-> 打开设备-> 控制设备参数-> 启动采集-> 等待图像-> 处理图像后把缓冲区加入队列-> 循环等待图像-> 停止采集-> 关闭设备。

打开设备后，可以对设备进行控制操作。

可以参考示例程序，了解具体函数的使用情况。



4、函数说明

ir_find_dev

函数：int32 __stdcall ir_find_dev(ir_dev_info_t *pdev, int32 *total_num,char *lpBroadcastIP);

参数：

pdev： 返回设备列表。

total_num： 网络中的设备总数目。

lpBroadcastIP：广播地址。一般是"255.255.255.255"。

返回值：

0： 正确；

<0： 错误。错误值的定义参考：ir_errinfo.h

说明：

首先要使用该函数查找网络中的设备，然后才能对其中的一个设备进行后续操作。

示例：

```
void CIrCamDemoView::OnDevOpen()
{
int MAX_DEV_NUM=128;
ir_dev_info_t device[MAX_DEV_NUM];
int count = 0;
memset(device,0,MAX_DEV_NUM*sizeof(ir_dev_info_t));
//(1)
int ret = ir_find_dev(device,(long*)&count,m_lpBroadcastIP);
if(ret < 0)
{
return;
}
int nIndex_Selected_Dev = 0; //打开第一个设备
//(3)
ret = ir_open (&ir_handle, &device[nIndex_Selected_Dev],
(ir_net_status_callback_func_t)ir_net_status,(void*)this->GetSafeHwnd());
if(ret != 0)
{
return;
}
}
```

ir_open

函数：

```
int32 __stdcall ir_open (ir_dev_handle_t *hd, ir_dev_info_t *pdev,ir_net_status_callback_func_t
callback_func, void *user_data);
```

参数：

hd： 输出打开的设备句柄。以后的函数操作，都是对该设备的句柄进行操作。

pdev： 输入要打开的一个设备。该设备信息通常是由函数ir_find_dev 得到。

callback_func：网络通断变化时的回调函数。

user_data：回调函数的参数。

返回值：

0： 正确；

<0： 错误。错误值的定义参考：ir_errinfo.h

说明：

查找完网络中的设备列表后，使用该函数打开列表中的一个设备，然后进行操作。

示例：

无。

ir_close

函数：int32 __stdcall ir_close (ir_dev_handle_t hd);

参数：

hd： 由函数ir_open 得到的设备句柄。是输入参数。

返回值：

0： 正确；

<0： 错误。错误值的定义参考：ir_errinfo.h

说明：

调用该函数关闭打开的设备。

示例：

无

ir_start_video

函数：int32 __stdcall ir_start_video(ir_dev_handle_t hd, void *user_data);

参数：

hd： 由函数ir_open 得到的设备句柄。

user_data： 用户传入自定义数据，在得到数据缓冲区时再返回该数据。可以是NULL。

返回值：

0： 正确；

<0： 错误。错误值的定义参考：ir_errinfo.h

说明：

启动图像采集。

示例：

无。

ir_stop_video

函数：int32 __stdcall ir_stop_video(ir_dev_handle_t hd);

参数：

hd： 由函数ir_open 得到的设备句柄。

返回值：

0： 正确；

<0： 错误。错误值的定义参考：ir_errinfo.h

说明：

停止采集图像。

ir_ctl

函数：int32 __stdcall ir_ctl (ir_dev_handle_t hd, ir_ctl_code_t ctl_code, void *ctl_para);

参数：

hd： 由函数ir_open 得到的设备句柄。

ctl_code： 操作码，参考控制码的定义：ir_ctl_code_t

ctl_para： 要设置的参数或者得到的参数。

返回值：

0： 正确；

<0： 错误。错误值的定义参考：ir_errinfo.h

说明：

对设备进行控制。使用该函数可以获取和设置设备参数。

示例：

无。

ir_wait_buf

函数：int32 __stdcall ir_wait_buf(ir_dev_handle_t hd, ir_img_buf_t* pimg, int32 timeout_s);

参数：

hd： 由函数ir_open 得到的设备句柄。

pimg： 得到的图像数据，包括图像格式、宽高信息以及数据地址和用户由函数ir_start_video 传入的自定义数据。参考结构ir_img_buf_t 的定义。

timeout_s： 等待的超时时间。单位秒。

返回值：

0： 正确；

<0： 错误。错误值的定义参考：ir_errinfo.h

说明：

获取采集到的图像数据。

一般用法：

用户使用线程来获取图像：

(1)调用函数ir_wait_buf 获取图像

(2)当缓冲区状态是正常时（函数ir_wait_buf 返回0），处理图像

(3)调用函数ir_queue_buf，把该缓冲区排队到采集图像时使用的缓冲区队列中。

示例：

无。

ir_queue_buf

函数：int32 ir_queue_buf(ir_dev_handle_t hd);

参数：

hd： 由函数ir_open 得到的设备句柄。

返回值：

0： 正确；

<0： 错误。错误值的定义参考：ir_errinfo.h

说明：

把用户处理使用后的缓冲区排队到采集图像使用的缓冲区队列中。

一般用法：

用户使用线程来获取图像：

(1)调用函数ir_wait_buf 获取图像

(2)当缓冲区状态是正常时（函数ir_wait_buf 返回0），处理图像

(3)调用函数ir_queue_buf，把该缓冲区排队到采集图像时使用的缓冲区队列中。

示例：

无

ir_readSpi

函数：int32 ir_readSpi(ir_dev_handle_t fd,int32 nReg,int32 *pData);

参数：

fd： 由函数ir_open 得到的设备句柄。

nReg： spi 寄存器地址（对应的FPGA 的寄存器地址*2 ）

pData： 读取的16 位数据

返回值：

0： 正确；

<0： 错误。错误值的定义参考：ir_errinfo.h

说明：

通过spi 从地址nReg 读取一个16 bits 的数据。保留接口，用户一般不使用。（对有需求的客户，一般用于读取设备内部的环境温度。）

示例：

```
int nReg=6;
int nR=0;
ir_readSpi(ir_handle,nReg,(long*)&nR);
```

ir_writeSpi

函数：int32 ir_writeSpi(ir_dev_handle_t fd,int32 nReg,int32 nData);

参数：

fd： 由函数ir_open 得到的设备句柄。

nReg: spi 寄存器地址（对应的FPGA 的寄存器地址*2 ）

nData: 写的16 位数据

返回值：

0： 正确；

<0： 错误。错误值的定义参考：ir_errinfo.h

说明：

通过spi 向地址nReg 写一个16 bits 的数据。保留接口，用户一般不使用。

示例：

```
int nReg=6;
int nData=0x56ab;
ir_writeSpi(ir_handle,nReg,nData);
```

ir_Wr2Cam_WaitResponse

函数：int32 ir_Wr2Cam_WaitResponse(ir_dev_handle_t fd,stCamOpData* pCamData);

参数：

fd： 由函数ir_open 得到的设备句柄。

pCamData： 要写的控制信息、等待应答的信息。

返回值：

0： 正确；

<0： 错误。错误值的定义参考：ir_errinfo.h

说明：

通过网络，向相机写控制字、然后读取相机的应答信息。

示例：

```
int camTau2_Get_SERIAL_NUMBER(ir_dev_handle_t fd,DWORD *pCamSeri,DWORD*pSensorSeri)
{
//(1)
stCamOpData aCamData;
memset(&aCamData,0,sizeof(stCamOpData));
aCamData.nFuncCode = TAU2_SERIAL_NUMBER;
aCamData.nDataLen = 0;
aCamData.nWantRcvLen = FIXED_CMD_LEN +8;
//(2)
int nRet = ir_Wr2Cam_WaitResponse(fd,&aCamData);
if(nRet == IRCAM_OK)
{
DWORD* pdwData = (DWORD*)(aCamData.lpResponse);
//(3.2)
if(pCamSeri != NULL)
*pCamSeri = be32_to_le32(pdwData[0]);
```

```
//(3.3)
if(pSensorSeri != NULL)
*pSensorSeri = be32_to_le32(pdwData[1]);
return IR_E_OK;
}
else
return nRet;
}
```

ir_reboot

函数：int32 ir_reboot(ir_dev_handle_t fd);
参数：
fd： 由函数ir_open 得到的设备句柄。
返回值：
0： 正确；
<0： 错误。错误值的定义参考：ir_errinfo.h
说明：
重启设备。
示例：
无。

ir_GetIP

函数：int32 ir_GetIP(eth_conf_t* lpConf,int32 *p_total_num);
参数：
lpConf： 指向结构eth_conf_t 的指针。该指针是用户数据区，必须非空。
p_total_num：返回查找到的设备数目。
返回值：
0： 正确；
<0： 错误。错误值的定义参考：ir_errinfo.h
说明：
枚举网络中的设备的IP 地址、网络掩码、网关、DNS。可以跨网段查找设备。

示例：

```
eth_conf_t m_device[MAX_DEV_NUM];
int m_nDevNum=0;
int ret = ir_GetIP(m_device,(int32*)&m_nDevNum);
if(ret != 0)
{
printf("No Device!");
return;
}
```

ir_SetIP

函数：int32 ir_SetIP(eth_conf_t* lpConf);

参数：

lpConf： 指向结构eth_conf_t 的指针。该指针是用户数据区，必须非空。一般是调用函数ir_GetIP 得到的一个设备的IP 信息结构。

返回值：

0： 正确；

<0： 错误。错误值的定义参考：ir_errinfo.h

说明：

设置设备的IP 地址、网络掩码、网关、DNS。设置后，设备下次重启后，修改的IP 地址才能起作用。

设备出厂的默认IP 地址是192.168.0.201，用户需要根据自己的网络环境来修改设备的IP 地址。

示例：

```
eth_conf_t m_device[MAX_DEV_NUM];
int m_nDevNum=0;
int ret = ir_GetIP(m_device,(int32*)&m_nDevNum);
if(ret == 0)
{
eth_conf_t aConf = m_device[0];
ret = ir_SetIP(&aConf);
if(ret != 0)
```

```
{
printf("设置IP 失败.");
}
return;
}
```

ir_reboot_UDP

函数：int32 ir_reboot_UDP(eth_conf_t* lpConf);

参数：

lpConf： 指向结构eth_conf_t 的指针。该指针是用户数据区，必须非空。一般是调用函数ir_GetIP 得到的一个设备的IP 信息结构。

返回值：

0： 正确；

<0： 错误。错误值的定义参考：ir_errinfo.h

说明：

让指定IP 地址的设备重启。

示例：

```
eth_conf_t m_device[MAX_DEV_NUM];
int m_nDevNum=0;
int ret = ir_GetIP(m_device,(int32*)&m_nDevNum);
if(ret == 0)
{
eth_conf_t aConf = m_device[0];
ret = ir_reboot_UDP(&aConf);
if(ret != 0)
{
printf("让设备重启失败.");
}
return;
}
```

5、数据结构说明

ComPara_t

```
typedef struct _COM_PARA
{
int32 nComID;
int32 nDataLen; //读写的数据长度
}ComPara_t;
```

说明：

单独读或者写串口的参数。

nComID： 串口ID。只能是USER_COM_3 或者USER_COM_4。

nDataLen： 读写的数据长度。一次最多256 个字节。

ComData_t

```
typedef struct _COM_DATA
{
ComPara_t aComPara;
unsigned char nData[MAX_COM_DATA_LEN];
}ComData_t;
```

说明：

nData： 读写串口的缓冲区。

Com_W_R_Data_t

```
typedef struct _COM_W_R_DATA
{
ComData_t wData;
ComData_t rData;
}Com_W_R_Data_t;
```

说明：

先写串口、然后再读取串口。

ir_version_info_t

typedef struct _ir_version_info

{

u_int32 fpga_ver;

u_int32 L_driver_ver;

u_int32 L_udp_server_ver;

u_int32 client_SDK_ver;

}ir_version_info_t;

说明：

获取设备的版本信息。

user_io_t

typedef struct _user_io

{

int32 nPort;

int32 nVal;

}user_io_t;

说明：

IO 操作的数据结构。

nPort：

输入时，只能是：USER_INPUT1_PORT 和USER_INPUT2_PORT。

输出时，只能是：USER_OUTPUT1_PORT 和USER_OUTPUT2_PORT。

nVal：

只能是0 或者1。

ir_cam_config_t

设备基本参数：

typedef struct _ir_cam_config

{

int watchdog_en;

```
int auto_reboot_en;
int auto_reboot_mode; //0: by time; 1: by frames
char auto_reboot_time[16]; //什么时间重启系统。时间格式：比如： 2:30:0
int auto_reboot_frames; //采集多少帧后自动重启系统
}ir_cam_config_t;
```

说明：

watchdog_en： 0 或者1。是否使能设备的看门狗功能。设置后，在设备下次启动时生效。

auto_reboot_en： 0 或者1。是否使能设备的自动重启功能。

auto_reboot_mode：在使能设备自动重启功能时，设置自动重启的条件。0：依据设备的时间；1：依据设备采集的图像总数。注意，由于设备内部没有电池记忆时间，每次设备启动后的时间复位到1970-1-1 ,0 :0 :0 ,当依据设备的时间时 ,时间值只能小于23:59:59。时间的字符串格式为 :hh:mm:ss

com_open_para_t

```
typedef struct _com_open_para
{
int32 nComID;
char lpBaud[8]; //e.g.: "57600"
char lpPar[4]; //e.g.: "N"
char lpBits[4]; //e.g.: "8"
char lpStopb[4]; //e.g.: "1"
}com_open_para_t;
```

说明：

用于打开设备上的串口时，设置串口参数。

nComID： 只能是USER_COM_3 或者USER_COM_4。

ir_video_palette_t

```
typedef enum _ir_video_palette
{
IR_VIDEO_R080 = FOURCC('R','0','8','0'), //8bit 数据
IR_VIDEO_R140 = FOURCC('R','1','4','0'), //14bit 数据
}ir_video_palette_t;
```

说明：用于设置设备采集数据的格式。

ir_video_format_t

```
typedef struct _ir_video_format
{
ir_rect_t rect;
ir_video_palette_t fourcc;
int32 bpps; ///< bits per pixel。改变fmt 时，内部改变该值。用户不应该改变该值。
int32 img_size; ///<得到的图像的大小。
}ir_video_format_t;
```

说明：

获取的图像的基本信息。

ir_img_buf_t

```
typedef struct _ir_img_buf
{
ir_video_format_t video_fmt;
struct timeval time; ///< buf 采集完成的时间: tv_sec and tv_usec
u_int8 *data; ///< 图像数据缓冲区
u_int32 detached; ///< 内部使用。0，可以用于采集；1，给用户处理
void *ptr_user; ///< 用户自定义的数据。调用函数ir_start_video()时传入，得到图像数据后再传出。
}ir_img_buf_t;
```

说明：

获取的图像的基本信息。

ir_dev_info_t

```
typedef struct _ir_dev_info
{
u_int32 dwManuID; ///< 'IRWD'
char dev_name[64]; ///< 设备名称
u_int32 ip_addr; ///< 设备IP 地址
u_int32 fpga_ver;
u_int32 driver_ver;
```

u_int32 libs_ver;

u_int16 nDeviceClassID; ///< 'ir'

u_int16 nDeviceID; ///< 01

u_int64 GUID; ///< GUID

}ir_dev_info_t;

说明：

调用函数ir_find_dev 得到的设备信息。然后函数ir_open 打开其中一个设备。

ir_net_status_t

typedef struct _ir_net_status

{

u_int32 status; ///< 0：正常；1：网络断开

struct timeval a_time; ///< 网络状态发生的时刻

void *ptr_user; ///< 用户传进来的数据。调用ir_open 时传入，网络状态改变时通过回调函数传出。

}ir_net_status_t;

说明：

网络状态改变时调用的回调函数返回的网络状态。

ir_ctl_code_t

///< 给用户使用的控制码。注意，控制码0x100~0x200 是内部使用，不能冲突。

typedef enum _ir_ctl_code

{

CTL_CODE_MIN = 1,

GET_VERSION_INFO = 10, //获取设备版本信息

OPEN_COM = 100, //打开设备上的串口

CLOSE_COM= 101, //关闭设备上的串口

READ_COM = 110, //从设备上的串口读取数据

WRITE_COM= 111, //向设备上的串口写数据

W_R_COM = 120, //先向设备上的串口写数据、然后再读数据

WR_IO = 130, //让设备的IO 输出

RD_IO = 131, //查询设备上的IO 状态

GET_CAM_VIDEO_FORCC = 150, //获取设备的图像格式

SET_CAM_VIDEO_FORCC= 160, //设置设备的图像格式

CTL_G_TIME = 0x1035, ///< 获取设备时间

CTL_S_TIME = 0x1036, ///< 设置设备时间

CTL_G_PARA = 0x1040, ///< 获取设备的ir_cam_config_t 参数

CTL_S_PARA = 0x1041, ///< 设置设备的ir_cam_config_t 参数

CTL_SAVE_PARA= 0x1042, ///< 存储设备的ir_cam_config_t 参数

CTL_CODE_MAX,

}ir_ctl_code_t;

说明：

对设备控制时的控制码。

注意：

（1）设置设备的时间后，设备下次启动后的时间复位到1970-1-1,0:0:0

（2）保存设备ir_cam_config_t 参数后，设备下次启动后生效。

（3）串口和IO 的操作控制码，只对有IO 和串口硬件接口板的用户有效。

eth_conf_t

typedef struct _eth_conf

{

char lpIP[32]; //设备的IP 地址

char lpMask[32]; //设备的网络掩码，一般是255.255.255.0。

char lpGateway[32]; //设备的网关

char lpDNS[32]; //设备的网络DNS

unsigned char lpMAC[8]; //设备的MAC

}eth_conf_t;

描述设备IP 属性的结构。