

SG-PCIE-PN-200S二次开发文档

PNDEV接口说明

版本记录

时间	版本	说明
2025.11	1.0	添加PROFINET从站功能
2026.3	1.1	添加PROFINET主站功能
2026.4	1.2	增加Linux调用说明
2026.5	1.3	增加EtherNet/IP从站功能

功能简介

SG-PCIE-PN-200S 是一款高性能的PCIe接口卡，使PC机通过PCIe接口连接至PROFINET、EtherNet/IP网络，板卡支持PROFINET主站、PROFINET从站、EtherNet/IP从站协议（[可通过网页选择其工作模式](#)），支持Windows、Linux下进行二次开发，以动态连接库的方式驱动。

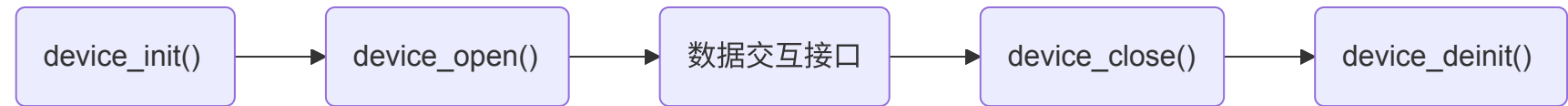
适用平台

平台	架构	编译器	支持状态
Linux	x86_64	GCC 9+	待支持
Windows	x86_64	MSVC 2019+	支持

Windows 端

开发编程时，直接加载 **PN-DEV.dll** 即可，接口描述文件位于 **PNDEV.h**,dll文件放置在可执行文件同级目录下即可，根据自身实际使用场景，选择对应章节开发说明。

库函数调用流程



通用部分

板卡支持PROFINET主、从站，部分数据结构跟库接口是通用的，数据交互接口是有差异的，可根据自身使用场景，查看对应章节。

通用数据结构说明

DEVICE_NET_T

```
typedef struct {
    BYTE name[16];
    BYTE mac[32];
    BYTE ip[16];
    BYTE mask[16];
    BYTE gw[16];
} DEVICE_NET_T, * PDEVICE_NET_T;
```

name: 网卡名称（固定名称，网口1:eth0 网口2:eth1）

mac: 网卡MAC（只读）

ip: 网卡IP地址

mask: 网卡子网掩码

gw: 网卡默认网关地址

通用库接口说明

初始化PCIe设备

```
int device_init();
```

C

说明: 最多支持4块板卡，对应设备索引0~4
返回值: 可操作设备数量，大于等于0为成功

注销PCIe设备

```
BOOL device_deinit();
```

C

说明: 用于关闭 `device_init()` 接口打开的资源
返回值: TRUE / FALSE

打开设备

```
int device_open(int index, int type);
```

C

说明: 用于打开设备，连接板卡
index: 设备索引号，有一个设备时索引号为0，有两个可以是0或1
type: 设备类型 0:PROFINET从站 1: PROFINET主站 2:EtherNet/IP从站 3:未知
返回值: 0为成功

关闭设备

```
BOOL device_close(int index);
```

C

说明: 用于关闭 `device_open()` 打开的资源
index: 设备索引号
返回值: TRUE / FALSE

获取网络信息

```
BOOL device_net_get(int index, PDEVICE_NET_T p);
```

C

说明: 用于获取板卡IP等网络参数
index: 设备索引号
p: 结构体指针
返回值: TRUE / FALSE

设置网络信息

```
BOOL device_net_set(int index, PDEVICE_NET_T p);
```

C

说明: 用于设置板卡IP等网络参数
index: 设备索引号
p: 结构体指针
返回值: TRUE / FALSE

PROFINET 从站部分

需登录网页，将板卡工作模式选择为 `PROFINET从站`，板卡默认IP为 `192.168.0.45`

数据结构说明

PNIO_SLAVE_MODULE_T

```
typedef struct _VEVICE_MODULE_T {
    DWORD status;
    BYTE name[32];
    DWORD type;    //in/out
    DWORD rec_len;
    BYTE record[4];
    DWORD in_size;
    DWORD out_size;
    BYTE in_buff[128];
    BYTE out_buff[128];
} PNIO_SLAVE_MODULE_T, * PPNIO_SLAVE_MODULE_T;
```

status: 模块状态 0:未插入 1:已插入

name: 模块名称

type: 模块类型 0:输入 1:输出 2:输入输出

rec_len: 不关注

record: 不关注

in_size: 模块输入字节数

out_size: 模块输出字节数

in_buff: 模块输入缓冲区

out_buff: 模块输出缓冲区

PNIO_SLAVE_INFO_T

```
typedef struct {
    BYTE app_ver[16];
    BYTE fw_ver[16];
    DEVICE_NET_T net[2];
    DWORD max_slot_num;
    DWORD cur_slot_num;
    DEVICE_MODULE_T module[64];
} PNIO_SLAVE_INFO_T, * PPNIO_SLAVE_INFO_T;
```

app_ver: 库版本

fw_ver: 固件版本

net: 网口参数

max_slot_num: 支持最大插槽数量（64）

cur_slot_num: 已使用插槽数量

module: 插槽中对应模块参数

接口函数说明

获取板卡全部参数

```
BOOL pnio_s_info_get(int index, PPNIO_SLAVE_INFO_T p);
```

说明: 用于获取板卡全部参数

index: 设备索引号

p: 结构体指针

返回值: TRUE / FALSE

获取板卡全部插槽的参数

```
BOOL pnio_s_module_info_all_get(int index, int solt, PDEVICE_MODULE_T p);
```

说明: 用于获取从站全部插槽参数

index: 设备索引号

p: 结构体指针 （此处结构体数组大小应为64）

返回值: TRUE / FALSE

获取板卡单个插槽的参数

```
BOOL pnio_s_module_info_get(int index, int slot, PDEVICE_MODULE_T p);
```

说明: 用于获取从站单个插槽参数

index: 设备索引号

p: 结构体指针

返回值: TRUE / FALSE

获取输入缓冲区数据

```
BOOL pnio_s_module_data_get(int index, int slot, int type, int offset, int size, unsigned char* buff);
```

说明: 用于获取从站单个插槽参数

index: 设备索引号

slot: 第几个插槽

type: 输入或输出 0: 输入 1: 输出

offset: 偏移地址

size: 读取字节数

buff: 缓存数据的字节数组，不得小于size大小

返回值: TRUE / FALSE

设置输出缓冲区数据

```
BOOL pnio_s_module_data_set(int index, int slot, int type, int offset, int size, unsigned char* buff);
```

说明: 用于设置从站单个插槽参数

index: 设备索引号

slot: 第几个插槽

type: 固定值1

offset: 偏移地址

size: 读取字节数

buff: 缓存数据的字节数组，不得小于size大小

返回值: TRUE / FALSE

PROFINET主站部分

需登录网页，将板卡工作模式选择为 **PROFINET主站**，板卡默认IP为 **192.168.0.45**。主站需通过网页端上传TIA Portal（博图）生成的组态文件后方可正常工作，相关操作参考说明书部分。

数据结构说明

PNIO_MASTER_LIST_T

```
typedef struct {
    int online;
    int station;
    int slot;
    int subslot;
    int type;
    int addr;
    int len;
} PNIO_MASTER_LIST_T, *PPNIO_MASTER_LIST_T;
```

online: 在线状态 0: 离线 1: 在线

station: 从站编号

slot: 插槽编号

subslot: 子插槽编号

type: 类型 0: 输入 1: 输出 2: 输入输出

addr: 起始地址

len: 数据长度

PNIO_MASTER_CONFIG_T

```
typedef struct {  
    int station_cnt;  
    int online_cnt;  
    int reserve;  
    int item_cnt;  
    PNIO_MASTER_LIST_T item[200];  
}PNIO_MASTER_CONFIG_T, *PPNIO_MASTER_CONFIG_T;
```

station_num: 从站数量

online_num: 在线从站数量

reserve: 保留

item_cnt: item数组有效长度

item: item数组

PNIO_MASTER_INFO_T

```
typedef struct {  
    char app_ver[16];  
    char fw_ver[16];  
    DEVICE_NET_T net[2];  
    PNIO_MASTER_CONFIG_T cfg;  
} PNIO_MASTER_INFO_T, *PPNIO_MASTER_INFO_T;
```

app_ver: 库版本号

fw_ver: 固件版本号

net: 网络参数

cfg: 主站配置参数

接口函数说明

获取板卡全部参数

```
BOOL pnio_m_info_get(int index, PPNIO_MASTER_INFO_T p);
```

说明: 用于获取从站全部插槽参数

index: 设备索引号

p: 结构体指针

返回值: TRUE / FALSE

获取输入缓冲区数据

```
BOOL pnio_m_module_data_get(int index, int slot, int type, int offset, int size, unsigned char* buff);
```

说明: 用于获取从站单个插槽参数

index: 设备索引号

slot: 第几个插槽

type: 输入或输出 0: 输入 1: 输出

offset: 偏移地址
size: 读取字节数
buff: 缓存数据的字节数组，不得小于size大小
返回值: TRUE / FALSE

设置输出缓冲区数据

```
BOOL pnio_m_module_data_set(int index, int solt, int type, int offset, int size, unsigned char* buff);
```

说明: 用于设置从站单个插槽参数
index: 设备索引号
slot: 第几个插槽
type: 固定值1
offset: 偏移地址
size: 读取字节数
buff: 缓存数据的字节数组，不得小于size大小
返回值: TRUE / FALSE

EtherNet/IP从站部分

需登录网页，将板卡工作模式选择为 **EtherNet/IP从站**，板卡默认IP为 **192.168.0.45**。

数据结构说明

EIP_SLAVE_INFO_T

```
typedef struct {
    char app_ver[16];
    char fw_ver[16];
    DEVICE_NET_T net[2];
    int in_size;
    int out_size;
} EIP_SLAVE_INFO_T, *PEIP_SLAVE_INFO_T;
```

app_ver: 库版本号
fw_ver: 固件版本号
net: 网络参数
in_size: 输入大小（字节）
out_size:输出大小（字节）

接口函数说明

获取板卡全部参数

```
BOOL eip_s_info_get(int index, PEIP_SLAVE_INFO_T p);
```

说明: 用于获取从站全部参数
index: 设备索引号
p: 结构体指针
返回值: TRUE / FALSE

获取输入缓冲区数据

```
BOOL eip_s_module_data_get(int index, int type, int offset, int size, unsigned char* buff);
```

说明: 用于获取从站缓冲区数据
index: 设备索引号
type: 输入或输出 0: 输入 1: 输出

offset: 偏移地址
size: 读取字节数
buff: 缓存数据的字节数组，不得小于size大小
返回值: TRUE / FALSE

设置输出缓冲区数据

```
BOOL eip_s_module_data_set(int index, int type, int offset, int size, unsigned char* buff);
```

说明: 用于设置从站缓冲区数据
index: 设备索引号
type: 固定值1
offset: 偏移地址
size: 读取字节数
buff: 缓存数据的字节数组，不得小于size大小
返回值: TRUE / FALSE

接口库使用说明

将库函数文件都放置在工作目录下即可，库函数文件有三个：`PNDEV.h`、`PNDEV.lib`、`PNDEV.dll`。

VC调用动态库方法

- 在扩展名为.cpp的文件中包含PNDEV.h头文件。如：`#include "PNDEV.h"`
- 在工程的链接器设置中链接到`PNDEV.lib`文件。如：在VS环境下，在项目属性页里的配置属性-->连接器-->输入-->附加依赖项 中 添加 `PNDEV.lib`。

Linux 端

本目录包含库和示例程序，库函数调用流程及接口库参考Windows端即可。

📁 目录结构

```
linux-rc/
├── README.md           # 本文档
├── docs/               # 文档目录（预留）
├── driver/            # 内核驱动程序
│   ├── kernel4.19/    # Linux Kernel 4.19 版本驱动
│   │   ├── Makefile   # 内核模块编译脚本
│   │   ├── rk_pcie_rc.c # PCIe RC 主驱动源码
│   │   ├── rk_pcie_rc.h # 驱动头文件
│   │   ├── rk_pcie_type.h # 类型定义
│   │   └── ...        # 其他相关文件
│   ├── kernel5.10/    # Linux Kernel 5.10 版本驱动
│   │   └── ...        # 同上
│   └── kernel6.1/     # Linux Kernel 6.1 版本驱动
│       └── ...        # 同上
├── include/           # 用户空间头文件
│   └── pcie_dev.h      # PCIe 设备 API 接口定义
├── lib/               # 编译好的静态库
│   └── libpn.a         # PCIE_PN 静态库（包含所有依赖）
└── src/               # 用户空间示例程序
    ├── Makefile       # 示例程序编译脚本
    ├── build/         # 编译输出目录
    │   └── pcie_demo  # 演示程序可执行文件
    └── pcie_demo.c    # 交互式演示程序源码
```

📁 详细说明

driver/ - 内核驱动程序

包含内核模块的源代码，支持多个 Linux 内核版本：

子目录	适用内核	说明
kernel4.19/	Linux 4.19.x	适用于旧版稳定内核
kernel5.10/	Linux 5.10.x	适用于 LTS 长期支持内核
kernel6.1/	Linux 6.1.x	适用于最新稳定内核

编译内核模块：

```
cd driver/kernel$(uname -r | cut -d'.' -f1-2)
make
sudo insmod rk_pcie_rc.ko
```

bash

若没有insmod命令，请使用 `sudo apt install kmod`

include/ - 用户空间头文件

提供用户空间程序访问 PCIe 设备所需的 API 接口定义。

主要文件：

- `pcie_dev.h` - 核心 API 接口定义
 - 设备管理函数声明
 - 数据结构定义（DEVICE_NET_T, PNIO_SLAVE_INFO_T 等）
 - Master/Slave 模式接口

使用示例：

```
#include "pcie_dev.h"

int main() {
    device_init();
    // ...
}
```

C

lib/ - 静态库

包含编译好的静态库文件，可直接链接到用户程序。

库文件：

- `libpn.a` - PCIE_PN 静态库
 - 包含项目核心功能代码
 - 自包含所有依赖，无需额外链接
 - 库大小：~324 KB（包含所有依赖）

src/ - 用户空间示例程序

包含使用 PCIE_PN 库的示例应用程序。

文件列表：

- `pcie_demo.c` - 交互式演示程序
 - 完整的菜单驱动界面
 - 演示所有 API 使用方法
 - 支持设备初始化、配置、数据读写等操作
- `Makefile` - 示例程序编译脚本
 - 自动依赖 `../include` 和 `../lib`
 - 支持多文件自动编译
 - 提供 clean、run、help 等目标

编译示例：


```
cd src
make
sudo ./build/pcie_demo
```

bash

快速开始

1. 编译内核驱动

```
# 选择对应内核版本的驱动
cd driver/kernel5.10

# 编译
make

# 加载模块
sudo insmod rk_pcie_rc.ko

# 验证
lsmod | grep rk_pcie
dmesg | tail -20
```

bash

2. 使用静态库

方法一：直接使用 lib 目录的库

```
gcc -I../include -L../lib your_app.c -lpn -o your_app
```

bash

方法二：从主项目重新编译

```
cd src/
make
```

bash

3. 运行示例程序

```
cd src
make
sudo make run
```

bash