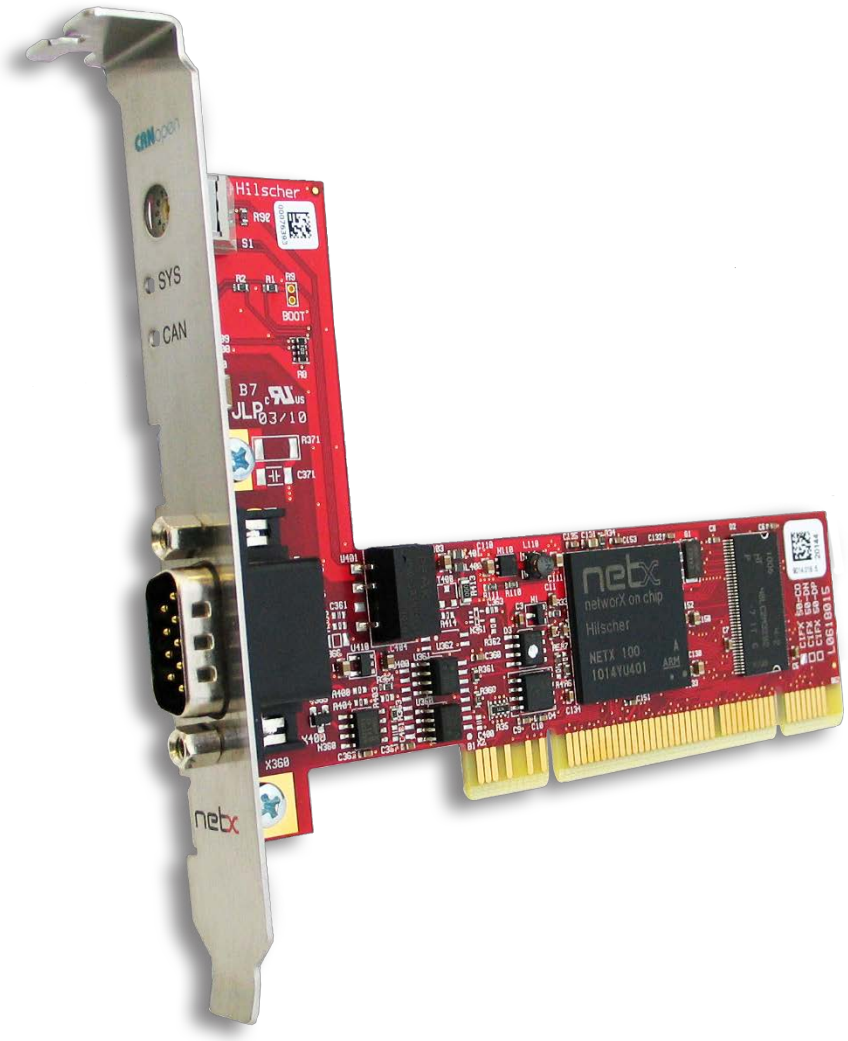
# HK-CIFX 操作指南

## HK-CIFX 板卡作为 PROFINET 主站的基本使用举例



## 概述

本文档的用意在于让初次接触HK/CIFX板卡的使用者了解该板卡的安装，配置，调试，以及二次开发包的使用。通过该文档的引导，使用者可以让CIFX板卡正常运行起来，并与其它设备进行基本的通信测试，也可采用二次开发包编写自己的应用程序。

文档中使用的HK/CIFX板卡型号为HK/CIFX 50-RE/+ML，PCI接口，可作为工业实时以太网协议的主站或从站，如Profinet主从站，Ethernet/IP主从站，EtherCAT主从站，详细的介绍请查看板卡的简介资料与说明手册。在本文中实现的功能是让HK/CIFX 50-RE/+ML作为Profinet主站，并与HK/CIFX 50-RE作Profinet从站进行通信测试。

HK/CIFX板卡PROFINET IO-Controller (V3)版本固件支持功能如下：

| Parameter | Description |
|---|---|
| Maximum number of ARs (Application Relation) | 128 for RT communication<br>64 for IRT communication |
| Maximum number of cyclic input data | 5652 bytes, including provider and consumer status |
| Maximum number of cyclic output data | 5700 bytes, including provider and consumer status |
| Send clock | 1 ms, 2 ms, 4 ms for RT mode<br>250 µs, 500 µs, 1 ms, 2 ms, 4 ms for IRT mode |
| Performance limits of ARs | Max. 8 ARs, if a send clock < 500 µs<br>Max. 16 ARs, if a send clock < 1 ms<br>Max. 64 ARs, if a send clock < 2 ms |
| Maximum number of submodules | 2048 |
| Maximum amount of data per IOCR | 1440 bytes |
| Number of IOCRs per AR | 1 Input IOCR<br>1 Output IOCR |
| Maximum amount of data for acyclic read/write record access | 65536 bytes |
| Maximum amount of record data per AR | 16384 bytes |
| Alarm processing (configurable) | Stack processes alarms automatically<br>Applikation processes alarms |
| Maximum number of ARVendorBlock | 256 |
| Maximum size of ARVendorBlockData | 512 bytes |
| Device Access AR CMI Timeout | 20 s |
| Functions | Automatic Name Assignment<br>Media Redundancy Client<br>Media Redundancy Manager (requires license) |
| DCP function API | Name Assignment IO-Devices (DCP SET NameOfStation)<br>Set IO-Devices IP (DCP SET IP)<br>Signal IO-Device (DCP SET SIGNAL)<br>Reset IO-Device to factory settings (DCP Reset FactorySettings)<br>Bus scan (DCP IDENTIFY ALL)<br>DCP GET |
| PROFINET specification | Implemented according to V2.3 ED2 MU3<br>Legacy Startup supported according to PROFINET specification V2.2 |

# 目录

## 1. 插板卡

第一步是在工控机未上电的情况下，将CIFX板卡插入到对应的PCI插槽中并固定住，确保板卡的金手指部分与PCI插槽是充分接触的。详细的接口定义可查看光盘中的文档《PC Cards CIFX 50 50E 70E 100EH UM 51 EN》。

## 2. 装驱动

给工控机上电，打开设备管理器，会发现新的PCI设备，如图2.1。


图2.1，已插入的PCI设备

在以下光盘路径中找到对应的Windows驱动并双击运行安装，如图2.2，详细安装过程可参考文档《cifX Device Driver Installation for Windows OI 10 EN》。


图2.2，CIFX的Windows驱动

路径：Communication_Solutions_DVD_2018-12-1_1_0500_181008_25726 \Driver and Toolkit\Device Driver (NXDRV-WIN)\Installation

驱动软件安装完成后建议重启工控机，CIFX板卡会自行寻找驱动并安装，安装完成后如图2.3。


图2.3，驱动安装完成

注：如在安装过程中提示驱动程序未经签名，如图2.4，请先自行下载并更新Windows补丁文件KB3033929，下载链接如下：https://docs.microsoft.com/en-us/security-updates/SecurityAdvisories/2015/3033929 补丁（或从此百度网盘链接下载：链接：https://pan.baidu.com/s/1PXi96eES5AcVdCIDMh0cZQ 提取码：n6ya ）更新过程如图2.5，更新补丁之后请重启工控机。

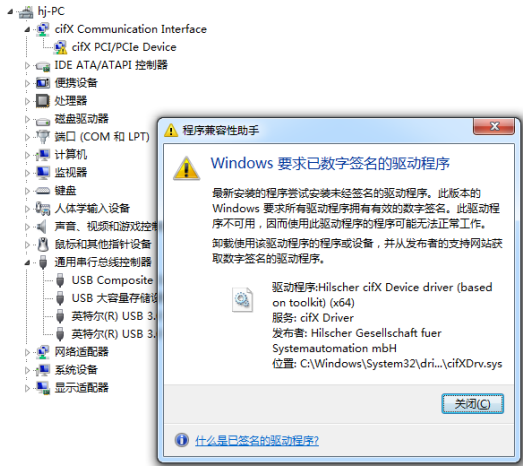



图2.4，未经签名的驱动程序

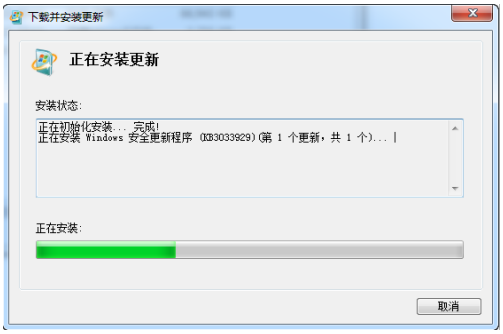



图2.5，补丁KB3033929安装

# 3. 加固件

板卡驱动成功安装后，在Windows的控制面板中会出现cifX Setup以及cifX Test两个工具，其中cifX Setup可为板卡加载所需的固件，固件决定了板卡的协议类型以及主从站类型。另外也可使用SYCON.net配置软件来加载和烧写固件。而cifX Test可查看板卡的基本信息，也可进行简单的数据交换测试。

## 3.1 cifX Setup

打开cifX Setup工具，点选DevNr/SN——Active Devices——cifX0——CH#0——Add（选择所需固件，这里以V2版本固件为例）——Apply。如图3.1。

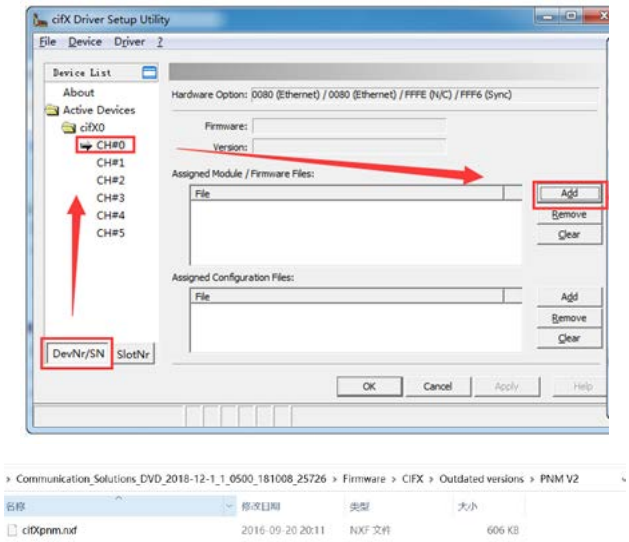



图3.1加载固件

### 3.2 cifX Test

打开cifX Test工具，点选Device——Open——cifX0——Channel0——Open，打开对应的通道，选择Information可查看板卡的信息，可以看到板卡的Channel0已经加载了的固件。选择Data Transfer可进行数据交换测试（该测试功能也可以后续的SYCON.net软件中进行，这里不做讲解）。

## 4. 安装 SYCON.net 软件

SYCON.net软件用于赫优讯全部系列产品的配置，并可进行状态监控和通信测试。

本例中CIFX板卡作为Profinet主站，需要用SYCON.net软件进行Profinet网络的组态，并下载给CIFX板卡，让CIFX板卡知道网络中各个Profinet从站的信息。下载完网络配置信息后也可用该软件监控板卡的状态，并测试通信过程，如Profinet异步通讯测试、周期数据通信测试等。

使用者可在产品光盘下找到SYCON.net软件的安装包。路径如下：

Communication_Solutions_DVD_2018-12-1_1_0500_181008_25726

\Software\SYCON.net\SYCON.net


虹科 — 您的解决方案合作伙伴 | info@intelnect.com

# 5. 在 SYCON.net 软件中配置板卡

根据应用的需要在SYCON.net软件中配置板卡并下载，具体过程如下。

## 5.1 打开 SYCON.net 软件

在开始菜单下找到SYCON.net并打开，默认密码为空，使用者可自行添加，如图5.1。

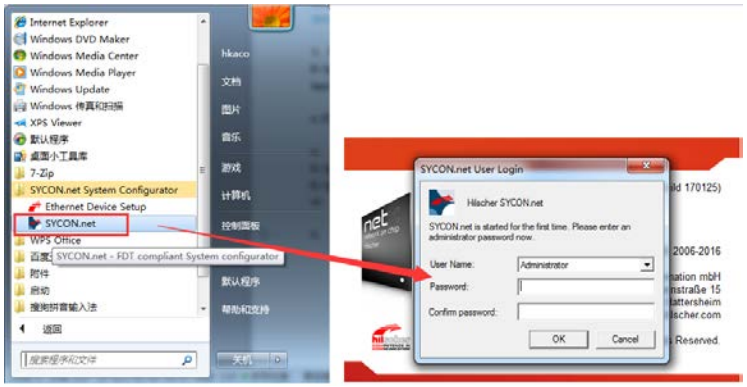
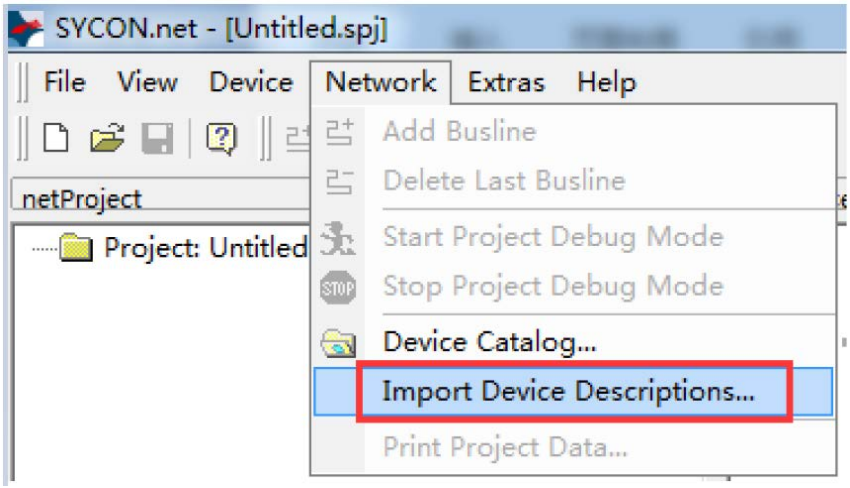



图5.1，打开SYCON.net软件

## 5.2 添加从站 ESI 文件到 SYCON.net 软件

即添加从站的ESI文件（.gsd文件）到Device Catalog中，点选菜单栏Network——Import Device Descriptions...——选择GSDML文件——注意文件类型选Profinet——打开——是，如图5.3，最后在界面右侧的Device Catalog中可以找到已经加载的从站，如界面右侧无Device Catalog，可在菜单栏Network——Device Catalog...中打开。



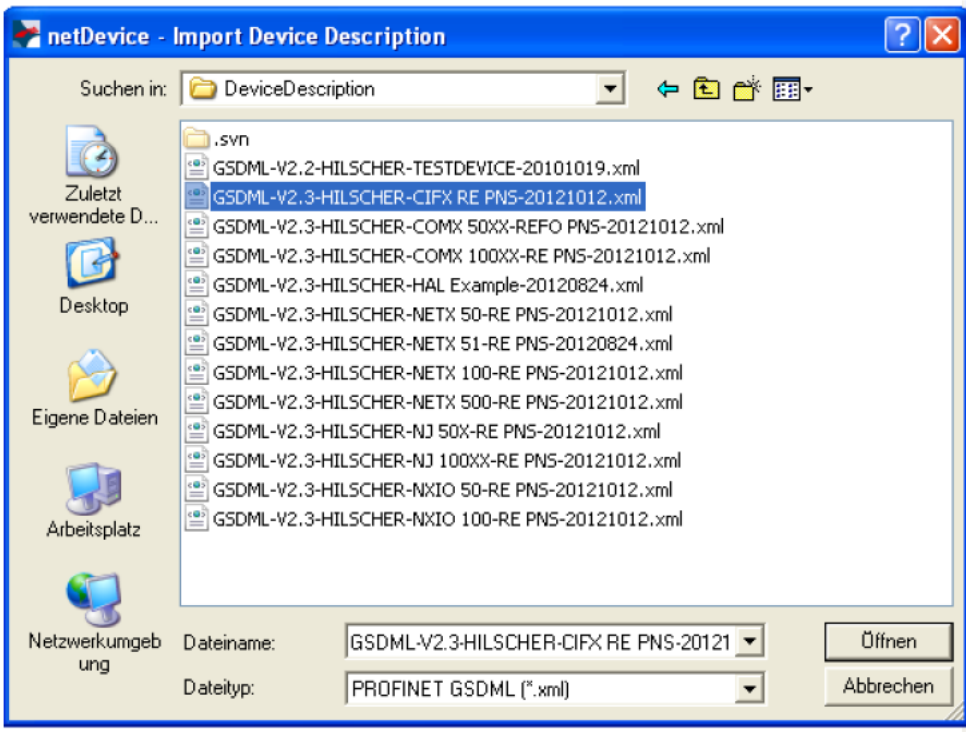



图5.2，添加从站GSDML文件

### 5.3 添加 CIFX 板卡并配置从站

在Device Catalog中找到PROFINET——Master——CIFX RE/PNM，并用鼠标左键将CIFX拖拉到界面中间的灰线上，如图5.3。

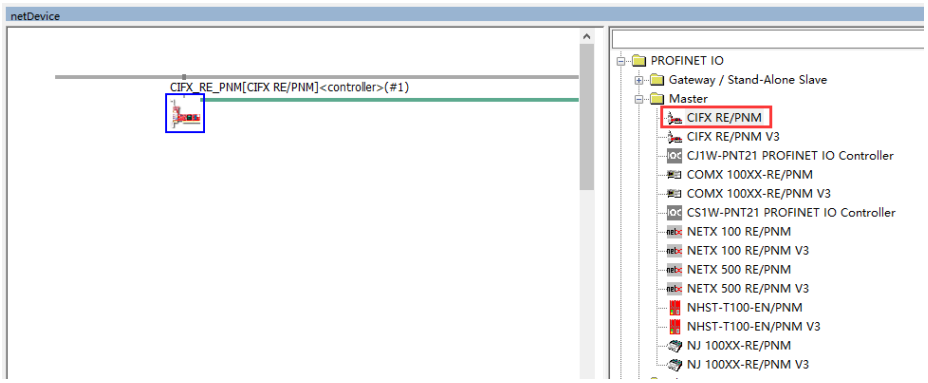



图5.3，添加CIFX板卡

双击拖拉出来的CIFX图标，或在图标上右键——Configuration打开配置界面，点开Device Assignment——Scan，找到CIFX 50-RE并打勾，最后Apply——OK关闭窗口，如图5.4。

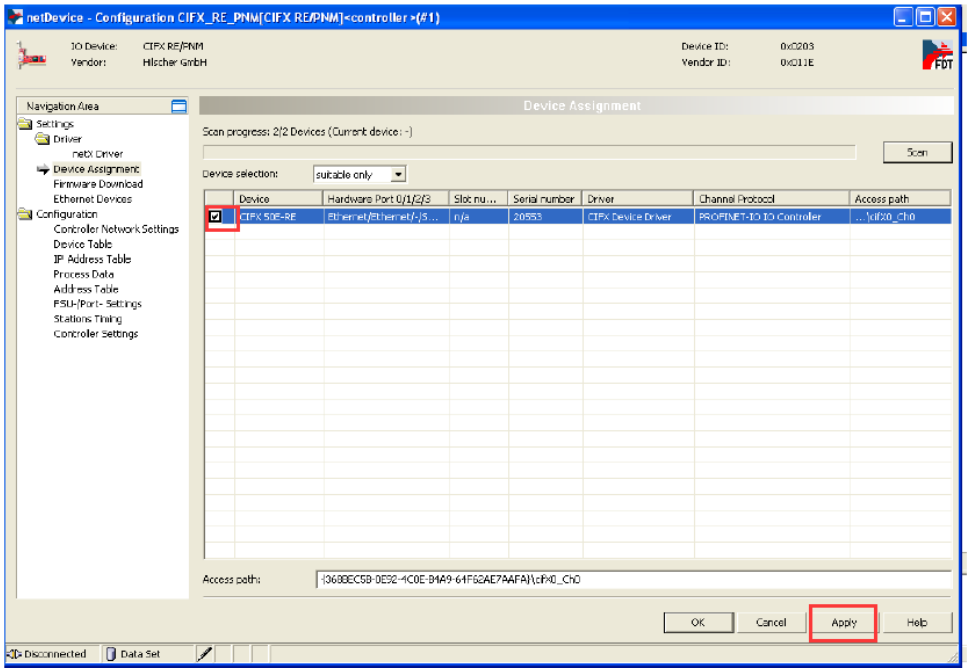



图5.4，Device Assignment

给网络中手动添加从站，操作同上，不过是选择PROFINET——Slave——CIFX_RE_PNS_V3.4.19 – V3.4.x。

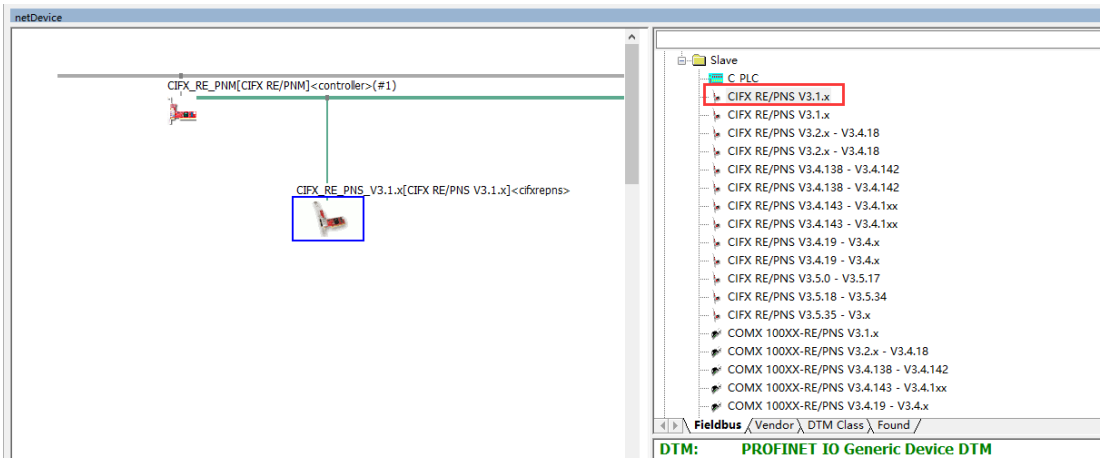



图5.5，Add Slaves

## 5.4 PROFINET 主从站配置

双击CIFX图标打开主站配置，其中可以配置主从站IP地址，从站设备名称，数据更新周期等，如图5.6，5.7，5.8。注意所有配置更高需要电机界面右下角“Apply”。

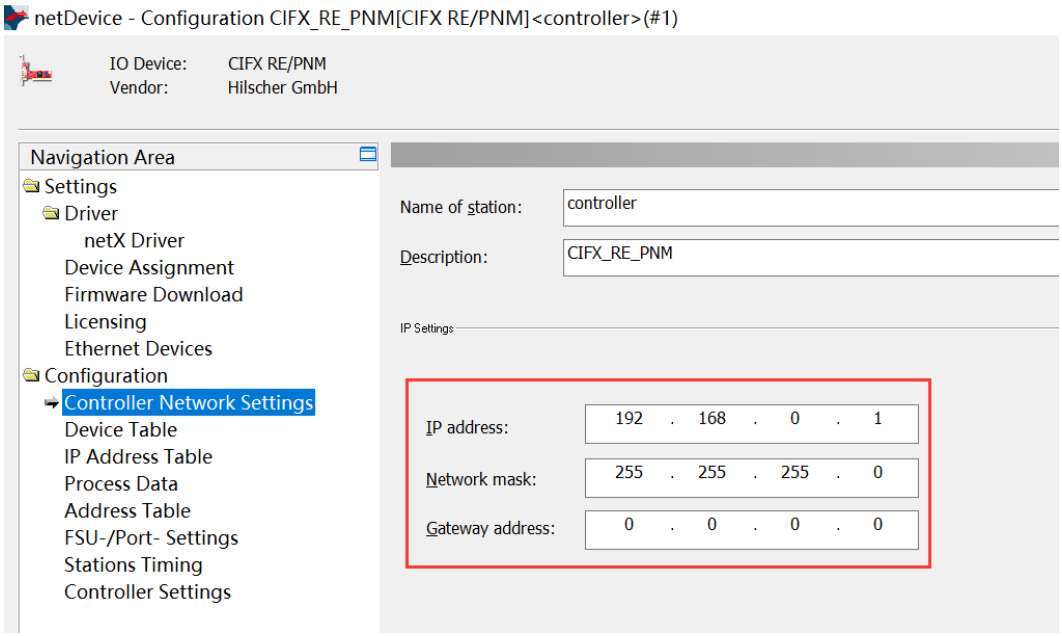



图5.6，主站配置

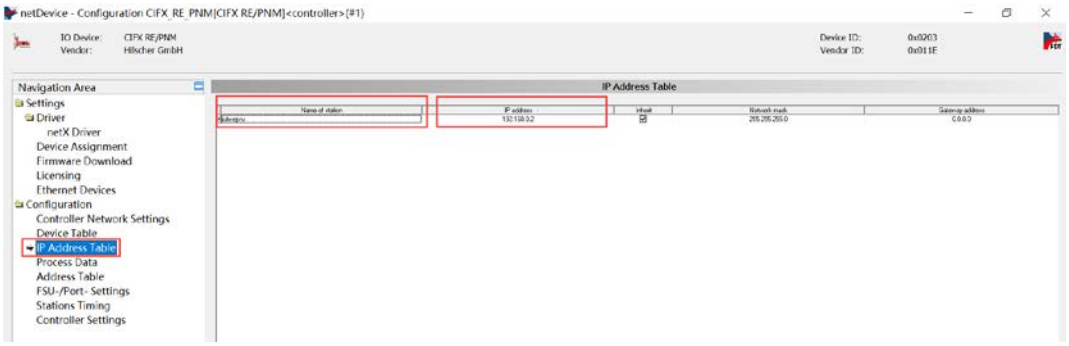



图5.7，从站设备名称和IP地址配置

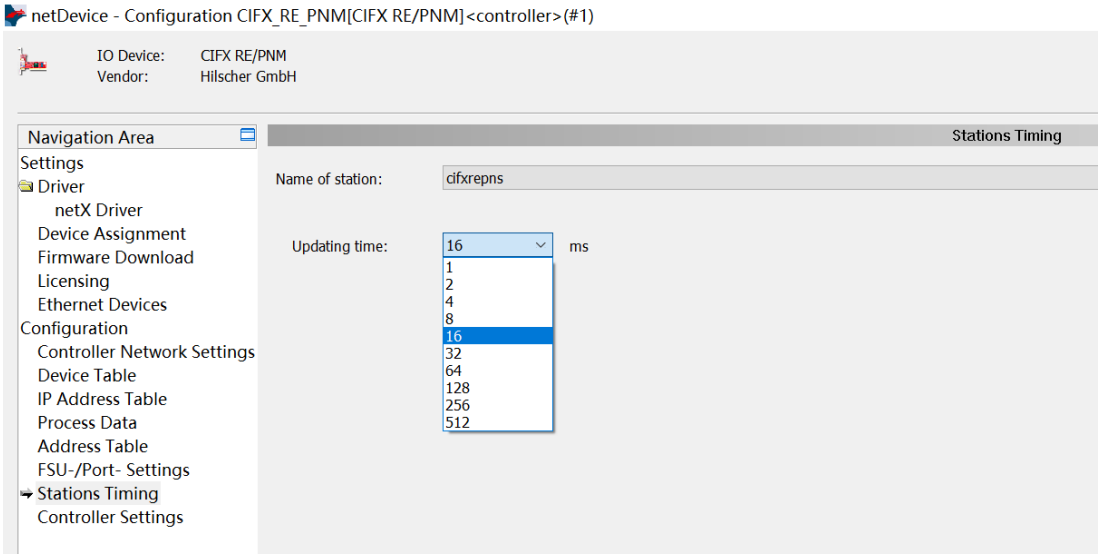



图5.7，更新周期

主站配置完成后，双击每一个从站，可以对从站进行配置如添加从站板卡的数据输入输出（这里以分别添加2 Bytes Input/Output为例），然后Apply，实际数据量配置根据具体应用而定。

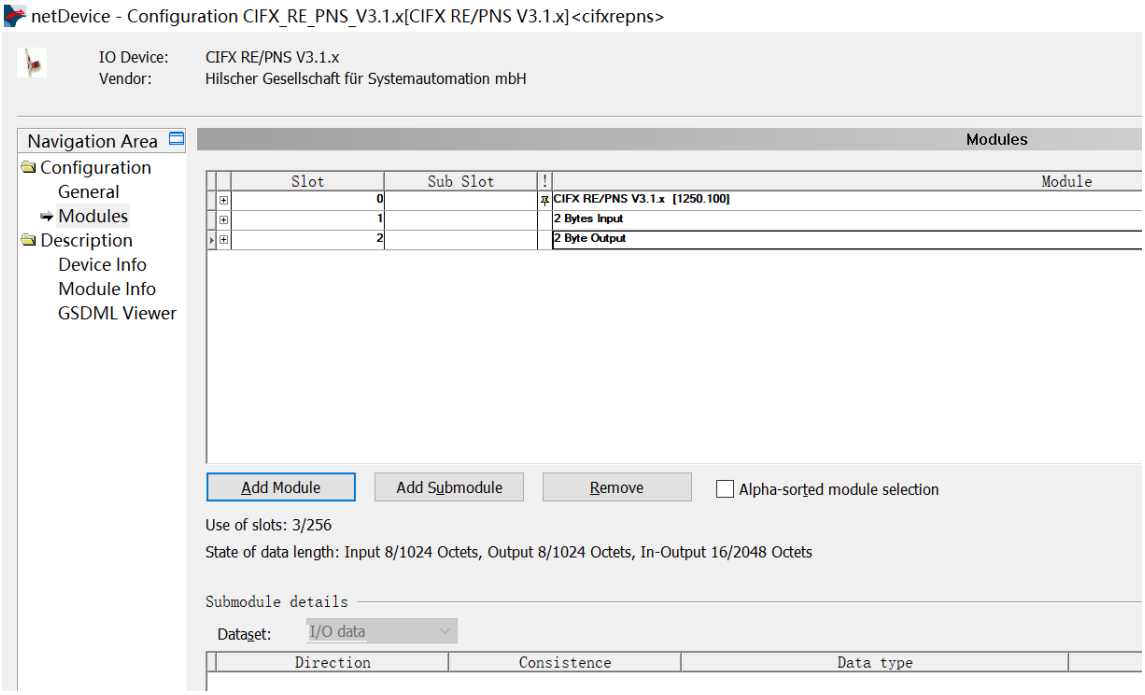



图5.9，从站Module配置

## 6. 下载配置到板卡

主从站配置完成之后，右键CIFX图标——Download即可把配置信息下载到板卡上。或是先右键——Connect，再右键——Download也行，弹出提示后直接确认下载即可，下载完成后在界面下方的Output Windows窗口会提示下载完成，如图6所示。

特别注意下载这一步骤是不可省略的，SYCON.net上的任何修改，只有下载到板卡上再可真正生效。

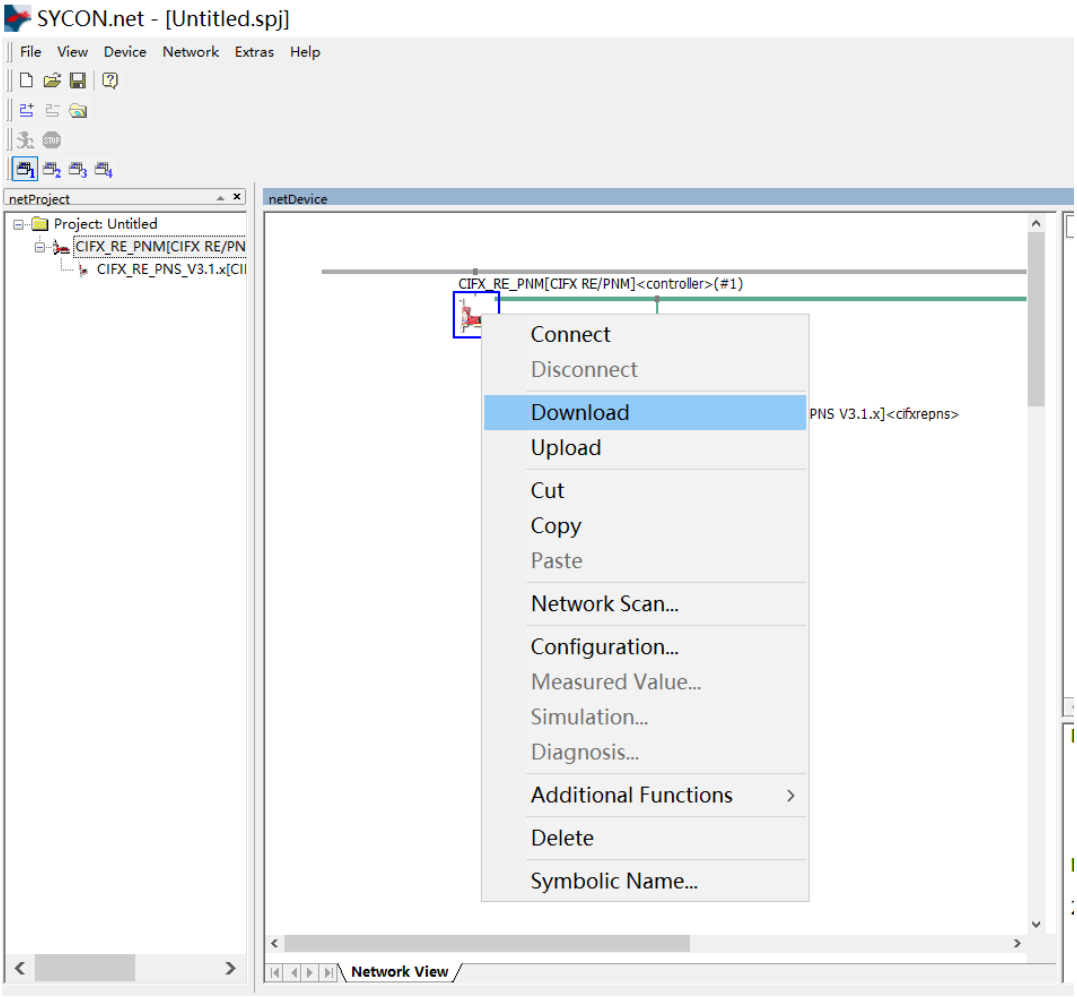



图6，下载配置

# 7. 用 SYCON.net 软件进行监控与测试

## 7.1 状态监控

下载配置完成后，板卡就根据配置运行起来了。此时，右键CIFX图标——Connect，再右键CIFX图标——Diagnosis...可以监控板卡的运行情况，如图7.1所示。

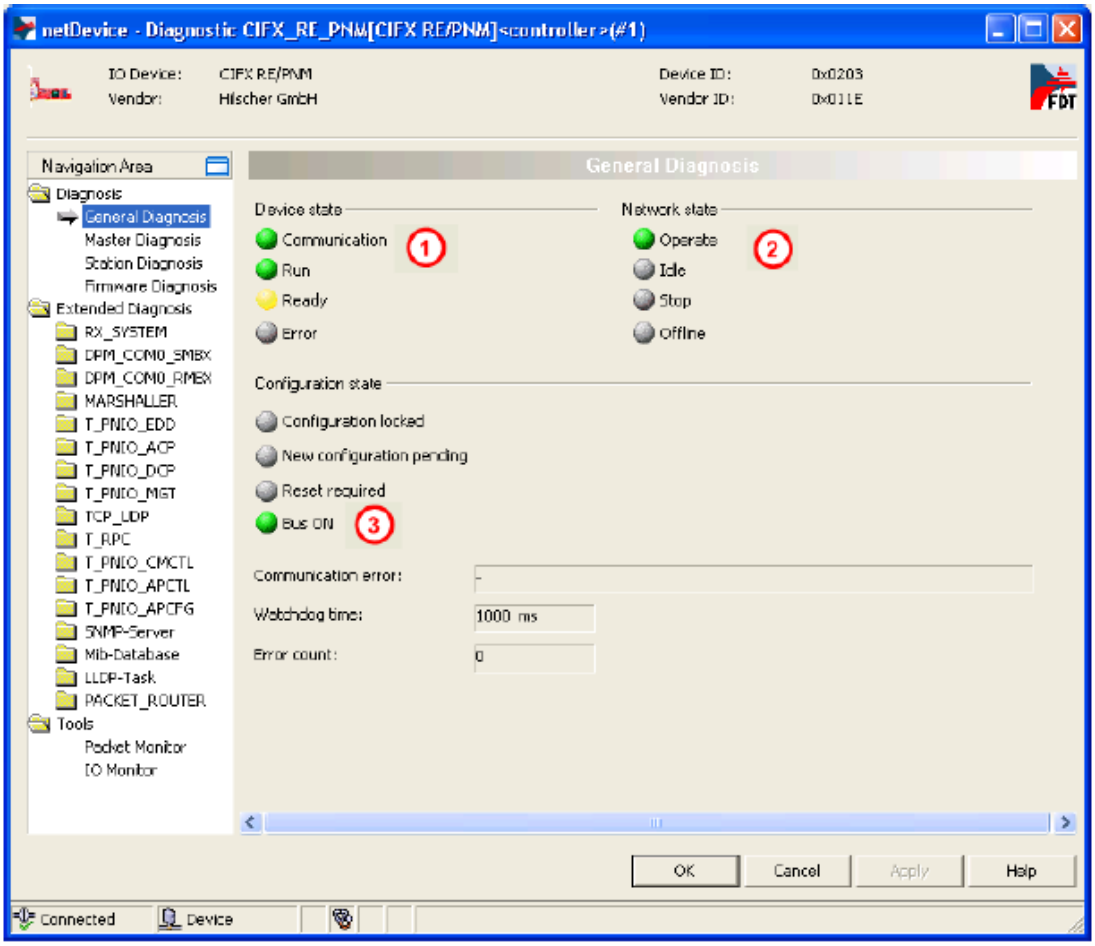



图7.1，Diagnosis

可以看到，板卡当前的状态是Communication+Run，表示板卡已经运行，且正常进行数据交换，这与图5.6中的主站配置有关。

从图5.6中可以看到，Controller Settings选项start of bus communication选择的是Automatically by device，表示由板卡自行运行，所以板卡上电后会自动Run，当需要由自己写的应用程序来启动板卡时，可以选择Controlled by application。

## 7.2 数据交换测试

首先讲一下CIFX板卡用于数据交换的两种机制，Packet与IO。其中Packet用于非周期数据的收发，应用程序每发一条Packet出去，都会得到一条对应的Packet反馈，一般用于PROFINET的

异步通信服务；而IO用于周期数据的收发，应用程序通过图5.7中的Address Table的唯一地址与从站中对应的PI数据进行交互，如写DO，读DI等。

在Diagnosis——Tools中有三个工具，其中Packet Monitor用于测试非周期数据， Process Image Monitor用于查看每个从站PI数据的当前值。

### 7.2.1 用 Packet Monitor 测试 PROFINET 异步通讯服务

板卡运行起来后，进入Diagnosis——Tools——Packet Monitor，可以看到有上下两个部分，上部分为Send，下部分为Receive，且每一部分都有自己的Packet header和data，如图7.2。每一个参数的具体定义可以参考光盘文档《netX Dual-Port Memory Interface DPM 14 EN》第38页，这里不做过多解释。

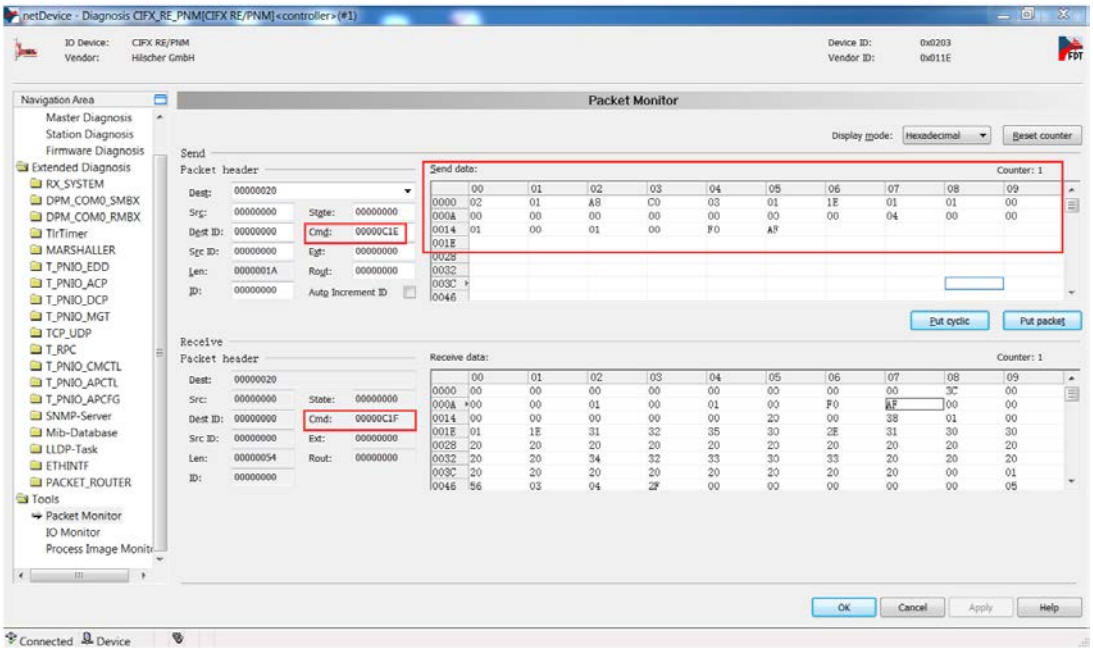



图7.2，Packet Monitor

但需要特别注意的是，所需发送的数据包根据《PROFINET IO Controller Protocol API 19 EN》或者在《PROFINET IO Controller V3 Protocol API 07 EN》中定义的来，对应的章节和页数如下：


虹科 — 您的解决方案合作伙伴 | info@intelnect.com

| Overview over Acyclic Request Packets of the PROFINET IO Controller Stack | | | |
|---|---|---|---|
| No. of section | Packet | Command code (REQ/CNF or IND/RES) | Page |
| 5.5.1 | Read Request | 0x0C1A | 126 |
| | Read Confirmation | 0x0C1B | 127 |
| 5.5.2 | Write Request | 0x0C1C | 129 |
| | Write Confirmation | 0x0C1D | 130 |
| 5.5.3 | Read Implicit Request | 0x0C1E | 132 |
| | Read Implicit Confirmation | 0x0C1F | 133 |
| 5.5.4 | Device Diagnosis Request | 0x0C22 | 135 |
| | Device Diagnosis Confirmation | 0x0C23 | 136 |
| 5.5.5 | Device is active and exchanges cyclic data with IO Controller | 0x2F0B | 139 |
| | Device is inactive | 0x2F0B | 140 |
| 5.5.6 | Common Status Service Request | 0x2F00 | 141 |
| | Common Status Service Confirmation | 0x2F01 | 142 |
| 5.5.7 | ModuleDiffBlock Request | 0x0C60 | 143 |
| | ModuleDiffBlock Confirmation | 0x0C61 | 144 |
| 5.5.8 | DCP Signal Request | 0x0C0E | 146 |
| | DCP Signal Confirmation | 0x0C0F | 148 |
| 5.5.9 | DCP SET Name Request | 0x0C0A | 150 |
| | DCP SET Name Confirmation | 0x0C0B | 152 |
| 5.5.10 | DCP SET IP Request | 0x0C0C | 154 |
| | DCP SET IP Confirmation | 0x0C0D | 156 |
| 5.5.12 | DCP IDENT ALL Request | 0x0C10 | 161 |
| | DCP IDENT ALL Confirmation | 0x0C11 | 162 |
| 5.5.14 | Alarm Acknowledge Request | 0x0C26 | 167 |
| | Alarm Acknowledge Confirmation | 0x0C27 | 169 |
| 5.5.15 | Release IO-Device Request | 0x0C2E | 171 |
| | Release IO-Device Confirmation | 0x0C2F | 172 |

图7.3，PROFINET异步服务

参数Dest的值一般给的是0x20，该值对于任何的非周期通信来说都是有效的；

参数Cmd的值决定了该Packet是什么功能，该测试是为了测试Read Implicit Request， Cmd的值为0x00000C1E，可参考光盘文档《PROFINET IO Controller Protocol API 19 EN》第132页；

Data的值定义，注意区分对应数据的高低字节顺序，比如这里IP地址为198.168.1.2，对应的十六进制数为C0.A8.01.02，在数据域的定义中02在第0个字节，01在第1个字节，以此类推：

| Structure Information | | | Type: Request |
|---|---|---|---|
| Variable | Type | Value / Range | Description 数据包HEADER定义 |
| **Structure TLR_PACKET_HEADER_T** | | | |
| ulDest | UINT32 | | Destination queue handle of APCTL-task process queue |
| ulSrc | UINT32 | | Source queue handle of AP-task process queue |
| ulDestId | UINT32 | 0 | Destination End Point Identifier not in use, set to zero for compatibility reasons |
| ulSrcId | UINT32 | 0 ... $2^{32}$ -1 | Source End Point Identifier, specifying the origin of the packet inside the Source Process. |
| ulLen | UINT32 | 26 | APIOC_READ_IMPL_REQ_DATA_T - Packet data length in bytes |
| ulId | UINT32 | 0 ... $2^{32}$ -1 | Packet identification as unique number generated by the source process of the packet |
| ulSta | UINT32 | 0 | Status unused for request, Set to zero |
| ulCmd | UINT32 | 0xC1E | PNIO_APCTL_CMD_READ_IMPL_REQ - Command |
| ulExt | UINT32 | 0 | Extension not in use, set to zero for compatibility reasons |
| ulRout | UINT32 | 0 | Routing not in use, set to zero for compatibility reasons |
| **Structure tData** (APIOC_READ_REQ_IMPL_DATA_T) | | | |
| ulIPAddress | UINT32 | | The IP-Address of the IO-Device to read from. |
| usDeviceId | UINT16 | | The Device ID of the IO-Device to read from. 数据域值定义 |
| usVendorId | UINT16 | | The Vendor ID of the IO-Device to read from. |
| usInstanceId | UINT16 | | The Instance ID of the IO-Device to read from (GSDML parameter). |
| usReserved | UINT16 | 0 | Reserved, set to zero. |
| ulApi | UINT32 | 0 ... $2^{32}$ -1 | Profinet API to read from. |
| ulDataLength | UINT32 | 0 .. 4096 | Maximum response size the requestor is able to handle. |
| usSlot | UINT16 | 0 ... $2^{15}$ -1 | Profinet Slot to read from. |
| usSubSlot | UINT16 | 0 ... 36863 | Profinet SubSlot to read from. |
| usIndex | UINT16 | 0 ... $2^{16}$ -1 | Profinet Index to read from. |

图7.4，PROFINET异步服务数据包结构（Request）

每一个异步服务请求都有对应的confirm响应的数据，在界面receive中可以看到，这里的相应数据为：

| Structure Information | | | Type: Confirmation |
|---|---|---|---|
| Variable | Type | Value / Range | Description |
| **Structure TLR_PACKET_HEADER_T** | | | |
| ulDest | UINT32 | | Destination queue handle of APCTL-task process queue |
| ulSrc | UINT32 | | Source queue handle of AP-task process queue |
| ulDestId | UINT32 | 0 | Destination End Point Identifier not in use, set to zero for compatibility reasons |
| ulSrcId | UINT32 | 0 ... $2^{32}$ -1 | Source End Point Identifier, specifying the origin of the packet inside the Source Process. |
| ulLen | UINT32 | 25 + n | APIOC_READ_IMPL_CNF_DATA_T - Packet data length + sizeof(IO-Devices Read-Response) -1 in bytes |
| ulId | UINT32 | 0 ... $2^{32}$ -1 | Packet identification as unique number generated by the source process of the packet |
| ulSta | UINT32 | See below | Status / error code. |
| ulCmd | UINT32 | 0xC1F | PNIO_APCTL_CMD_READ_IMPL_CNF - Command |
| ulExt | UINT32 | 0x0000<br>0x0080<br>0x00C0<br>0x0040 | Extension<br>No Sequenced Packet<br>First Packet of Sequence<br>Sequenced Packet<br>Last Packet of Sequence |
| ulRout | UINT32 | 0 | Routing not in use, set to zero for compatibility reasons |
| **Structure tData** (APIOC_READ_IMPL_CNF_DATA_T) | | | |
| ulPnio | UINT32 | 0 ... $2^{32}$ -1 | Profinet IO status of Read Request, RPC status. |
| ulApi | UINT32 | 0 ... $2^{32}$ -1 | Profinet API read from. |
| ulDataLength | UINT32 | 0 ... $2^{32}$ -1 | Total length of Read Response data. Needed in case that the packet is transferred via DPM Mailboxes that cut the packet in small segments. |
| usSlot | UINT16 | 0 ... $2^{15}$ -1 | Profinet Slot read from. |
| usSubSlot | UINT16 | 0 ... 36863 | Profinet SubSlot read from. |
| usIndex | UINT16 | 0 ... $2^{16}$ -1 | Profinet Index read from. |
| usAddVal1 | UINT16 | 0 ... $2^{16}$ -1 | Profinet Additional Value 1. |
| usAddVal2 | UINT16 | 0 ... $2^{16}$ -1 | Profinet Additional Value 2. |
| usReserved | UINT16 | 0 | Reserved, is set to zero. |
| abReadData[1] | UINT8[] | 0 ... $2^{8}$ -1 | Profinet Read Response. If more than 1 byte was read the remaining bytes of response follow this byte. |

图7.5，PROFINET异步服务数据包结构（Confirmation）


虹科 — 您的解决方案合作伙伴 | info@intelnect.com

至此，用Packet Monitor测试Read Implicit Service已结束，如果想了解更多的非周期通信功能，请查看文档《PROFINET IO Controller Protocol API 19 EN》《PROFINET IO Controller V3 Protocol API 07 EN》。

### 7.2.2 用 IO Monitor 测试 IO 周期数据收发

板卡运行起来后，进入Diagnosis——Tools——IO Monitor，同样可以看到上下两部分，如图7.9，上部分为Input data，即主站读从站的数据，下部分为Output data，即主站写从站的数据。可以通过这个功能对网络中连接设备根据具体变量偏移和大小，进行IO数据的读写：

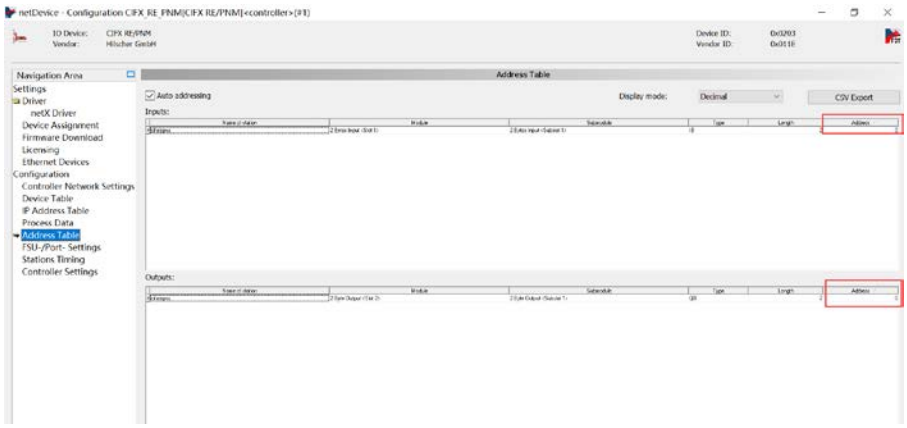


图7.6，确认相关数据偏移位置

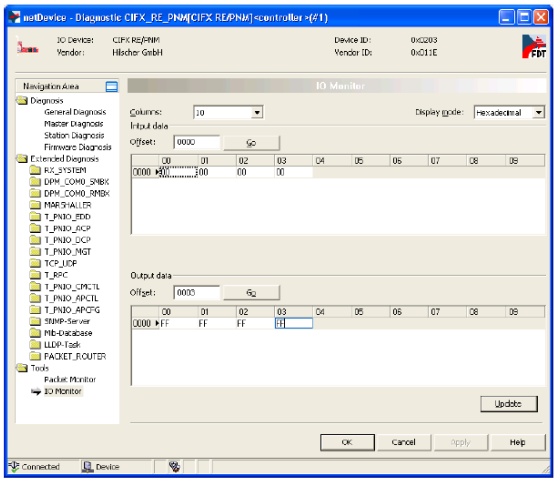


图7.7，IO Monitor监测收发周期性IO数据

至此，用IO Monitor测试PI数据收发已结束。

# 8. CIFX 的二次开发包

通过7.2章节的测试，相信每一个使用者对CIFX的通信机制都有一定的了解了，但如果使用者想基于CIFX板卡写应用程序，那么就需要用CIFX的二次开发包，二次开发包提供的API可实现7,2章节用到的所有通信过程，如异步通信服务等。

CIFX的API以库的方式提供，该库可以在以下的光盘路径下找到。

路径：Communication_Solutions_DVD_2018-12-1_1_0500_181008_25726\Driver and Toolkit\Device Driver (NXDRV-WIN)\API

CIFX的API主要分三部分的内容，Driver Functions，System Device Functions，Communication Channel Functions。

Driver Functions：带前缀xDriver的函数，用于选定某个板卡，一台PC可连接了多个板卡；

System Device Functions：带前缀xSysdevice的函数，与系统重置，下载，设备信息等相关的函数；

Communication Channel Functions：带前缀xChannel的函数，与协议，数据通信相关的函数。

每一部分Functions的使用方法是先打开，如xDriverOpen()，再调用别的API。详细的API说明可以参考《cifX API PR 05 EN》第14页。

接下来通过一个简单的例程来介绍开发包的基本使用流程与方法，C++示例代码路径：Communication_Solutions_DVD_2018-12-1_1_0500_181008_25726\ Examples and API\ 2. Application Examples\ PROFINET IO Controller\ Acyclic Services\。当您看完本示例后，如果需要以C语言方式来编写，可以参考C语言示例的基础上来写,,

Communication_Solutions_DVD_2018-12-1_1_0500_181008_25726\ Driver and Toolkit\ Device Driver (NXDRV-WIN)\ Examples\ cifXDemo，但注意添加必要的头文件和库。

在工程中首先是需要将开发板提供的二次开发包需要的头文件以及库文件cifx32dll.lib添加进工程中，在头文件中重点应用需要的是以下红框中的文件，这几个头文件是程序编写中需要重点参考的文件，如各种异步通信服务package包结构体的定义、cmd命令定义等，但建议其他头文件也添加进工程中，避免编译出错：

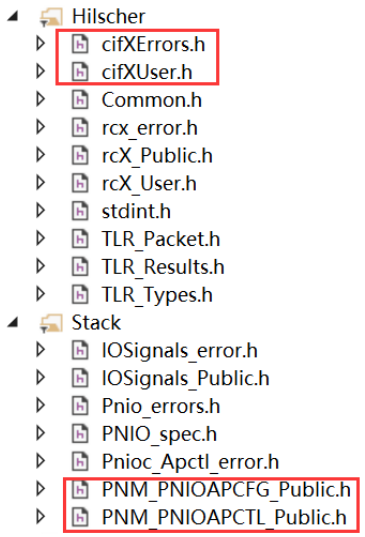



图8.1，头文件

在本演示例程中，APP_Main（）主函数中演示了板卡运行的基本执行流程、一些异步通信服务的调用（调用的是函数，具体数据包的定义在函数中，之后通过xChannelPutPacket()或xChannelGetPacket()函数将包发送或接收数据包）、以及同步通信服务（调用API函数xChannelIORead()或xChannelIOWrite()）等。总的来讲，程序常用的API函数有：

**Communication Channel functions:**

| Function | Description |
|---|---|
| xChannelOpen | Opens a connection to a communication channel |
| xChannelClose | Closes a connection |
| **Asynchronous services (Packets)** | |
| xChannelGetMBXState | Retrieve the channels mailbox state |
| xChannelGetPacket | Retrieve a pending packet from the channel mailbox |
| xChannelPutPacket | Send a packet to the channel mailbox |
| xChannelGetSendPacket | Read back the last sent packet |
| **Device Administrational/Informational functions** | |
| xChannelDownload | Download a file/configuration to the channel |
| xChannelReset | Reset the channel |
| xChannelInfo | Retrieve channel specific information |
| xChannelWatchdog | Activate/Deactivate/Trigger the channel Watchdog |
| xChannelHostState | Set the application state flag in the application COS flags, to signal the hardware if an application is running or not |
| xChannelBusState | Set the bus state flag in the application COS state flags, to start or stop fieldbus communication. |
| xChannelControlBlock | Access the channel control block |
| xChannelCommonStatusBlock | Access to the common status block |
| xChannelExtendedStatusBlock | Access to the extended status block |
| xChannelUserBlock | Access user block (not implemented yet!) |
| **Cyclic Data services (I/O's)** | |
| xChannelIORead | Instructs the device to place the latest data into the DPM and passes them to the user |
| xChannelIOWrite | Copies the data to the DPM and waits for the firmware to retrieve them |
| xChannelIOReadSendData | Reads back the last send data |
| **Cyclic Data services (I/O's, PLC optimized)** | |

本样例中程序基本流程以通信服务简单解析：

主函数APP_Main（）代表了程序中涉及的操作与流程：

（1、其中参数szBoardName需根据在cifxsetup中查看到的来定义，如果是cifx0，则定义为

cifx0，如果为其他的也需要定义为对应的：

```
TLR_RESULT App_Main(APPLICATION_T* ptApp)
{
  /* return value of main function */
  TLR_RESULT tResult = TLR_S_OK;

  /* name of board to use */
  char* szBoardName = "CIFx0";

  /* number of channel to use */
  TLR_UINT32 ulChannel = 0;

  /* general reference to driver */
  CIFXHANDLE  hDriver  = NULL;
```

（2、板卡驱动的初始化以及状态切换，让板卡能够正常启动以让后续通讯正常：


虹科 — 您的解决方案合作伙伴 | info@intelnect.com

```
/* Open Driver */
DEBUG("Opening driver...\n");
tResult = xDriverOpen(&hDriver);

if (CIFX_NO_ERROR == tResult)
{
  /* place driver handle in application resources */
  ptApp->tCommon.hDriver = hDriver;

  /* Driver successfully opened */
  /* Open channel */
  DEBUG("Opening channel %d on board %s...\n", ulChannel, szBoardName);
  tResult = xChannelOpen(hDriver, szBoardName, ulChannel, &hChannel);

  if (CIFX_NO_ERROR == tResult)
  {
    /* place channel handle in application resources */
    ptApp->tCommon.hChannel = hChannel;
```

```
/* system restart successful */
if (CIFX_NO_ERROR == tResult)
{
  /* Toggle Application Ready State Flag */
  do
  {
    tResult = xChannelHostState(hChannel, CIFX_HOST_STATE_READY, &ulState, ulTimeout);
```

```
/* bus on */
DEBUG("Setting bus state on and wait for communication to be established...\n");
tResult = xChannelBusState(hChannel, CIFX_BUS_STATE_ON, &ulState, 20000);
```

（3、异步通信服务举例，跟前面在SYCON.net中的操作一样，这里以read implicit service为例说明：

A、read implicit service在光盘文档《PROFINET IO Controller Protocol API 19 EN》第132页有介绍，其包的结构为以下截图，cmd命令字决定改数据包代表的功能，tdata数据域规定了向profinet设备发送的数据，然后获得设备中特定的数据：

| Structure Information | | | Type: Request |
|---|---|---|---|
| **Variable** | **Type** | **Value / Range** | **Description** |
| **Structure TLR_PACKET_HEADER_T** | | | |
| ulDest | UINT32 | | Destination queue handle of APCTL-task process queue |
| ulSrc | UINT32 | | Source queue handle of AP-task process queue |
| ulDestId | UINT32 | 0 | Destination End Point Identifier not in use, set to zero for compatibility reasons |
| ulSrcId | UINT32 | 0 ... $2^{32}$ -1 | Source End Point Identifier, specifying the origin of the packet inside the Source Process. |
| ulLen | UINT32 | 26 | APIOC_READ_IMPL_REQ_DATA_T - Packet data length in bytes |
| ulId | UINT32 | 0 ... $2^{32}$ -1 | Packet identification as unique number generated by the source process of the packet |
| ulSta | UINT32 | 0 | Status unused for request, Set to zero |
| ulCmd | UINT32 | 0xC1E | PNIO_APCTL_CMD_READ_IMPL_REQ - Command |
| ulExt | UINT32 | 0 | Extension not in use, set to zero for compatibility reasons |
| ulRout | UINT32 | 0 | Routing not in use, set to zero for compatibility reasons |
| **Structure tData** (APIOC_READ_REQ_IMPL_DATA_T) | | | |
| ulIPAddress | UINT32 | | The IP-Address of the IO-Device to read from. |
| usDeviceId | UINT16 | | The Device ID of the IO-Device to read from. |
| usVendorId | UINT16 | | The Vendor ID of the IO-Device to read from. |
| usInstanceId | UINT16 | | The Instance ID of the IO-Device to read from (GSDML parameter). |
| usReserved | UINT16 | 0 | Reserved, set to zero. |
| ulApi | UINT32 | 0 ... $2^{32}$ -1 | Profinet API to read from. |
| ulDataLength | UINT32 | 0 .. 4096 | Maximum response size the requestor is able to handle. |
| usSlot | UINT16 | 0 ... $2^{15}$ -1 | Profinet Slot to read from. |
| usSubSlot | UINT16 | 0 ... 36863 | Profinet SubSlot to read from. |
| usIndex | UINT16 | 0 ... $2^{16}$ -1 | Profinet Index to read from. |

B、数据包发送后，会有对应的read implicit confirm数据返回：

| Structure Information | | | Type: Confirmation |
|---|---|---|---|
| **Variable** | **Type** | **Value / Range** | **Description** |
| **Structure TLR_PACKET_HEADER_T** | | | |
| ulDest | UINT32 | | Destination queue handle of APCTL-task process queue |
| ulSrc | UINT32 | | Source queue handle of AP-task process queue |
| ulDestId | UINT32 | 0 | Destination End Point Identifier not in use, set to zero for compatibility reasons |
| ulSrcId | UINT32 | 0 ... $2^{32}$ -1 | Source End Point Identifier, specifying the origin of the packet inside the Source Process. |
| ulLen | UINT32 | 25 + n | APIOC_READ_IMPL_CNF_DATA_T - Packet data length + sizeof(IO-Devices Read-Response) -1 in bytes |
| ulId | UINT32 | 0 ... $2^{32}$ -1 | Packet identification as unique number generated by the source process of the packet |
| ulSta | UINT32 | See below | Status / error code. |
| ulCmd | UINT32 | 0xC1F | PNIO_APCTL_CMD_READ_IMPL_CNF - Command |
| ulExt | UINT32 | 0x0000<br>0x0080<br>0x00C0<br>0x0040 | Extension<br>No Sequenced Packet<br>First Packet of Sequence<br>Sequenced Packet<br>Last Packet of Sequence |
| ulRout | UINT32 | 0 | Routing not in use, set to zero for compatibility reasons |
| **Structure tData** (APIOC_READ_IMPL_CNF_DATA_T) | | | |
| ulPnio | UINT32 | 0 ... $2^{32}$ -1 | Profinet IO status of Read Request, RPC status. |
| ulApi | UINT32 | 0 ... $2^{32}$ -1 | Profinet API read from. |
| ulDataLength | UINT32 | 0 ... $2^{32}$ -1 | Total length of Read Response data. Needed in case that the packet is transferred via DPM Mailboxes that cut the packet in small segments. |
| usSlot | UINT16 | 0 ... $2^{15}$ -1 | Profinet Slot read from. |
| usSubSlot | UINT16 | 0 ... 36863 | Profinet SubSlot read from. |
| usIndex | UINT16 | 0 ... $2^{16}$ -1 | Profinet Index read from. |
| usAddVal1 | UINT16 | 0 ... $2^{16}$ -1 | Profinet Additional Value 1. |
| usAddVal2 | UINT16 | 0 ... $2^{16}$ -1 | Profinet Additional Value 2. |
| usReserved | UINT16 | 0 | Reserved, is set to zero. |
| abReadData[1] | UINT8[] | 0 ... $2^{8}$ -1 | Profinet Read Response. If more than 1 byte was read the remaining bytes of response follow this byte. |

C、在代码实现，也就是要首先规定号数据包中的内容，然后通过API函数

xChannelPutPacket()发送这个数据包：


虹科 — 您的解决方案合作伙伴 | info@intelnect.com

```
123  void App_SendReadImplReq(APPLICATION_T* ptApp)
124  {
125      APIOC_READ_IMPL_REQ_T tPck;
126
127      char cBuf[128];
128
129      memset(&tPck, 0, sizeof(tPck));
130
131      tPck.tHead.ulCmd  = PNIO_APCTL_CMD_READ_IMPL_REQ;
132      tPck.tHead.ulDest = 0x20;
133      tPck.tHead.ulLen  = sizeof(tPck.tData);
134
135      tPck.tData.ulDataLength = 1024;
136
137      DEBUG("Specify IP-address of device to read from: ");
138      fgets(cBuf, 128, stdin);
139      tPck.tData.ulIPAddress = atol(cBuf);
140
141      DEBUG("Specify VendorId of device to read from: ");
142      fgets(cBuf, 128, stdin);
143      tPck.tData.usVendorId = atoi(cBuf);
144
145      DEBUG("Specify DeviceId of device to read from: ");
146      fgets(cBuf, 128, stdin);
147      tPck.tData.usDeviceId = atoi(cBuf);
```

```
149      DEBUG("Specify InstanceId of device to read from: ");
150      fgets(cBuf, 128, stdin);
151      tPck.tData.usInstanceId = atoi(cBuf);
152
153      DEBUG("What API shall be used? Enter API: ");
154      fgets(cBuf, 128, stdin);
155      tPck.tData.ulApi = atol(cBuf);
156
157      DEBUG("What slot shall be used? Enter slot: ");
158      fgets(cBuf, 128, stdin);
159      tPck.tData.usSlot = atoi(cBuf);
160
161      DEBUG("What subslot shall be used? Enter subslot: ");
162      fgets(cBuf, 128, stdin);
163      tPck.tData.usSubSlot = atoi(cBuf);
164
165      DEBUG("What index shall be read? Enter index: ");
166      fgets(cBuf, 128, stdin);
167      tPck.tData.usIndex = atoi(cBuf);
168
169
170      if (CIFX_NO_ERROR != xChannelPutPacket(ptApp->tCommon.hChannel, (CIFX_PACKET *)&tPck, ptApp->tCommon.ulTimeout))
171      {
172        DEBUG("Sending packet failed.\n");
173      }
174  }
```

D、然后，同样有confirm数据：

```
575  void App_HandleReadImplCnf(APPLICATION_T* ptApp, CIFX_PACKET* ptPacket)
576  {
577      APIOC_READ_IMPL_CNF_T* ptPck = (APIOC_READ_IMPL_CNF_T*)ptPacket;
578
579      if (TLR_S_OK != ptPck->tHead.ulSta || 0 != ptPck->tData.ulPnio)
580      {
581          DEBUG("ReadImpl Confirmation: rcX-Error 0x%08x, Profinet-Error 0x%08x\n", ptPck->tHead.ulSta, ptPck->tData.ulPnio);
582      }
583      else
584      {
585          DEBUG("ReadImpl API %u, Slot %u, Subslot %u, Index %u returned %u byte:\n", ptPck->tData.ulApi, ptPck->tData.usSlot, ptPc
586          DEBUG("%s\n", ptPck->tData.abReadData);
587      }
588  }
```

程序中其他的异步通信服务App_SendBusScan(ptApp)、App_SendReadReq(ptApp)、

App_SendReadImplReq(ptApp)、App_SendWriteReq(ptApp)、App_GetHandles(ptApp)、

App_ReadOutputData(ptApp)、App_SendFactoryResetReq(ptApp)与以上操作过程类似。

异步通信服务主要是根据开发者的需要，自行决定需要采用哪些操作。

（4、周期性通信服务或者说IO数据更新，也就是实际的变量操作，主要是调用xChannelIORead（）和xChannelIOWrite（）来实现，相关定义如下，实际的变量偏移和长度可在SYCON.net的Address Table中看到，具体对哪个数据操作由此决定：

**Function call:**

```
int32_t xChannelIORead(   CIFXHANDLE      hChannel,
                          uint32_t        ulAreaNumber,
                          uint32_t        ulOffset,
                          uint32_t        ulDataLen,
                          void*           pvData,
                          uint32_t        ulTimeout)
```

**Arguments:**

| Argument | Data type | Description |
|---|---|---|
| hChannel | CIFXHANDLE | Handle of the channel. |
| ulAreaNumber | uint32_t | Number of the I/O Input area to get data from |
| ulOffset | uint32_t | Offset inside area to start reading data from |
| ulDataLen | uint32_t | Length of the data being retrieved |
| pvData | void* | Pointer to the return data buffer |
| ulTimeout | uint32_t | Timeout in ms to wait for I/O handshake completion of the channel (if configured) |

**Function call:**

```
int32_t xChannelIOWrite(  CIFXHANDLE      hChannel,
                          uint32_t        ulAreaNumber,
                          uint32_t        ulOffset,
                          uint32_t        ulDataLen,
                          void*           pvData,
                          uint32_t        ulTimeout)
```

**Arguments:**

| Argument | Data type | Description |
|---|---|---|
| hChannel | CIFXHANDLE | Handle of the channel. |
| ulAreaNumber | uint32_t | Number of the I/O Output area to send data to |
| ulOffset | uint32_t | Offset inside area to start writing data to |
| ulDataLen | uint32_t | Length of the data being send |
| pvData | void* | Pointer to the send data buffer |
| ulTimeout | uint32_t | Timeout in ms to wait for I/O handshake completion of the channel (if configured) |

在代码中，这里是用了内部定义的结构体来规定的，如果只是为了对具体的数据简单操作，可以直接规定对应的偏移和大小：

```
void App_HandleProcessData(APPLICATION_T* ptApp)
{
  /* return value of function */
  TLR_RESULT tResult = TLR_S_OK;

  /* cyclic io handing happens here */
  /* read RTD-Data */
  tResult = xChannelIORead(ptApp->tCommon.hChannel, 0, 0, ptApp->tCommon.ulReadBufferSize, ptApp->tCommon.pabReadBuffer, ptApp->tCommon.ulReadBufferSize);

  /* the action depends on the return value we have received */
  if ((CIFX_DEV_EXCHANGE_FAILED == tResult) || (CIFX_DEV_EXCHANGE_TIMEOUT == tResult))
  {
    DEBUG("Read failed with 0x%08x\n", tResult);
  }
  else if (CIFX_DEV_NO_COM_FLAG == tResult)
  {
    //DEBUG("Read failed. Device is not communicating\n");
  }
  else if (CIFX_NO_ERROR == tResult)
  {
    if (g_bInputData != ptApp->tCommon.pabReadBuffer[0])
    {
        DEBUG("Input data changed from 0x%02x to 0x%02x\n", g_bInputData, ptApp->tCommon.pabReadBuffer[0]);
    }
    g_bInputData = ptApp->tCommon.pabReadBuffer[0];

    //DEBUG("Read succeeded. Copy Inputs back to Outputs.\n");

#if 1
    /* we copy the memory we have read to the memory we want to send, */
    /* because we just want to mirror the data */
    memcpy (ptApp->tCommon.pabWriteBuffer, ptApp->tCommon.pabReadBuffer, ptApp->tCommon.ulWriteBufferSize);

    // write IO-Data
    tResult = xChannelIOWrite(ptApp->tCommon.hChannel, 0, 0, ptApp->tCommon.ulWriteBufferSize, ptApp->tCommon.pabWriteBuffer, ptApp->tCommon.ulWriteBufferSize);
```

## 9. 写在最后

CIFX板卡的功能非常强大，几乎支持市面上的所有协议，使用板卡作为其它协议时，只需要重新加载对应协议的固件，重新配置网络即可，且使用者本身无需对协议非常了解即可进行使用和二次开发。

本文所有内容经由本人测试与整理，如有歧义，请与英文原版说明书为准。


# 虹科云课堂

HongKe Online Acadamy

2020 年 2 月 21 日，虹科云课堂首次与大家见面，带来的第一节《CAN 总线基础之物理层篇》课程，就得到了各位工程师朋友们的热情支持与参与，当晚观看人数 4900+。我们非常感恩，愿不负支持与鼓励，致力将虹科云课堂打造成干货知识共享平台。

目前虹科云课堂的全部课程已经超过 200 节，如下表格是我们汽车相关的部分课程列表，大家通过微信扫描二维码关注公众号，点击免费课程直接进入观看，全部免费。

### 汽车以太网课程

智能网联下车载以太网的解决方案
SOME/IP协议介绍
基于CanEasy浅谈XCP
TSN/AVB 基于信用点的整形

### TSN技术课程

基于TSN的汽车实时数据传输网络解决方案
TSN时间敏感型网络技术综述
以太网流量模型和仿真
基于TSN的智能驾驶汽车E/E架构设计案例分享
IEEE 802.1AS 时间同步机制
TSN技术如何提高下一代汽车以太网的服务质量?

### CAN、CAN FD、CAN XL总线课程

CAN总线基础之物理层篇
CAN数据链路层详解篇
CAN FD协议基础
CAN总线一致性测试基本方法
CAN测试软件(PCAN-Explorer6)基本使用方法
CAN测试软件(PCAN-Explorer6)高级功能使用
浅谈CAN总线的最新发展：CAN FD与CAN XL
CAN线的各种故障模式波形分析

### LIN总线相关课程

汽车LIN总线基本协议概述
汽车LIN总线诊断及节点配置规范
LIN总线一致性测试基本方法
LIN自动化测试软件(LINWorks)基本使用方法
LIN自动化测试软件(LINWorks)高级功能使用
基于CANLIN总线的汽车零部件测试方案

### CAN高级应用课程

UDS诊断基础
UDS诊断及ISO27145
基于UDS的ECU刷写
基于PCAN的二次开发方法
CCP标定技术
J1939及国六排放
OBD诊断及应用（GB3847）
BMS电池组仿真测试方案
总线开发的流程及注意事项
车用总线深入解析

### 汽车测修诊断相关课程

汽车维修诊断大师系列-如何选择示波器
汽车维修诊断大师系列-巧用示波器
汽车维修诊断-振动异响（NVH）诊断方案

### 工业通讯协议基础课程

PROFINET协议基础知识
初识EtherCAT协议
初识CANopen协议
EtherNet/IP协议基础知识
IO-Link：工业物联网的现场基础
新兴工业级无线技术IO-Link Wireless

关注获取最新课程

汽车电子bilibili主页

工业智能互联
bilibili主页

# 智能通讯领域专业的
# 资源整合及技术服务落地供应商

## 关于虹科

虹科电子科技有限公司（前身是宏科）成立于1995年，总部位于中国南方经济和文化中心-广州；还在上海、北京、成都、西安、苏州、台湾、香港，韩国和日本设有分公司。

我们是一家高新技术公司，是广东省特批的两高四新、三个一批、专精特新和瞪羚企业，并与全球顶尖公司有多领域的深度技术合作，业务包括**工业自动化和数字化、汽车研发测试、自动驾驶等领域：医药和风电行业等的环境监测；半导体、轨道交通、航空航天等测试测量方案。**

虹科工程师团队致力于为行业客户提供创新产品和解决方案，全力帮助客户成功。

## 智能互联事业部

虹科是一家在通讯领域，尤其是汽车电子和智能自动化领域拥有超过 15 年经验的高科技公司，致力于为客户提供全方位的一站式智能互联解决方案。多年来，我们与全球行业专家深度合作，成为了行业内领先的通讯技术服务商。**我们提供全面的软硬件解决方案，包括【CAN/CAN FD、LIN、车载以太网、TSN、IO-Link/IO-Link wireless、OPC UA、CANopen、PROFINET、EtherNet/IP、EtherCAT】等各类通讯协议的解决方案、测试方案、培训和开发服务等。**

我们以满足客户需求为导向，以技术能力为基础，为国内外企业提供最适合的产品和最满意的服务。目前我们服务的客户已经超过 5000 家，我们自主研发的 EOL 测试系统、CCP/XCP 标定和 UDS 诊断服务开发服务以及 TSN 网络验证测试系统等也已经在业内完成超过 1000 次安装和测试。我们的方案覆盖了各行业知名企业，得到了包括蔚来，比亚迪，长城，联影，东芝三菱，安川等多个用户的一致好评。

**华东区（上海）销售**
**高印祺**

电话/微信: 136 6024 4187
邮箱: gao.yinqi@intelnect.com

**华东区（非上海）销售**
**林燕芬**

电话/微信: 135 1276 7172
邮箱: lin.yanfen@intelnect.com

**华南区销售**
**董欢**

电话/微信: 189 2224 3009
邮箱: dong.huan@intelnect.com

**华北区销售**
**张瑞婕**

电话/微信: 181 3875 8797
邮箱: zhang.ruijie@intelnect.com

**协议开发方案（全国）**
**郭泽明**

电话/微信: 189 2224 2268
邮箱: guo.zeming@intelnect.com

HongKe 虹科

虹科电子科技有限公司

www.intelnect.com
info@intelnect.com

广州市黄埔区开泰大道30号佳都PCI科技园6号楼

T (+86)400-999-3848

各分部：广州 | 成都 | 上海 | 苏州 | 西安 |
北京 | 台湾 | 香港 | 日本 | 韩国

版本：V1.0 - 22/11/14

获取工业行业资料　获取汽车行业资料