HK/CIFX 板卡作为 PROFINET 主站的基本使用举例

——万彬，技术支持工程师，2020.04.02

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。

1. 插板卡 2
2. 装驱动 2
3. 加固件 3
3.1 cifX Setup 3
3.2 cifX Test 4
4. 安装 SYCON.net 软件 4
5. 在 SYCON.net 软件中配置板卡 4
5.1 打开 SYCON.net 软件 4
5.2 添加从站 ESI 文件到 SYCON.net 软件 5
5.3 添加 CIFX 板卡并扫描从站 5
5.4 主从站 PROFINET 配置 7
6. 下载配置到板卡 9
7. 用 SYCON.net 软件进行监控与测试 9
7.1 状态监控 9
7.2 数据交换测试 10
7.2.1 用 Packet Monitor 测试 PROFINET 异步通讯服务 10
7.2.2 用 IO Monitor 测试 IO 周期数据收发 14
8. CIFX 的二次开发包 15
9. 写在最后 22

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》。

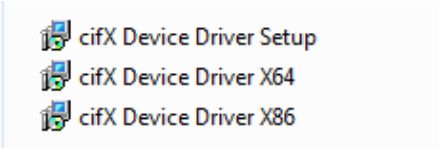


图 2.2，CIFX 的 Windows 驱动

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。

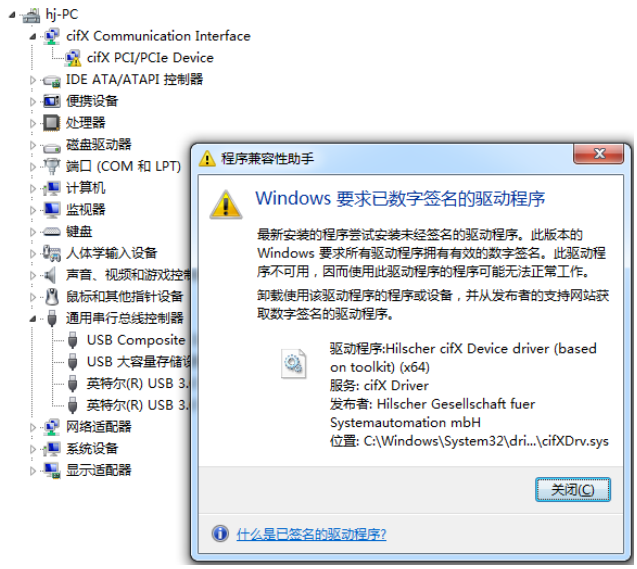


图 2.4，未经签名的驱动程序

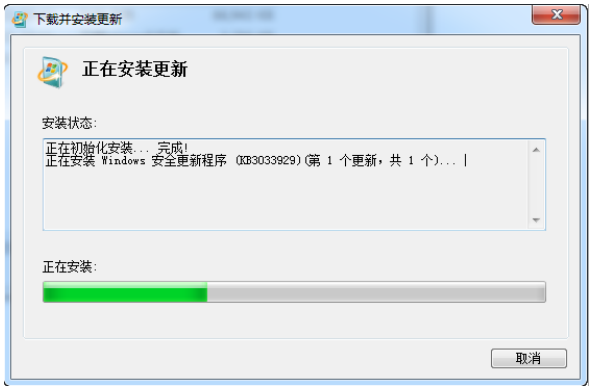


图 2.5，补丁 KB3033929 安装

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）——Apply。如图 3.1。

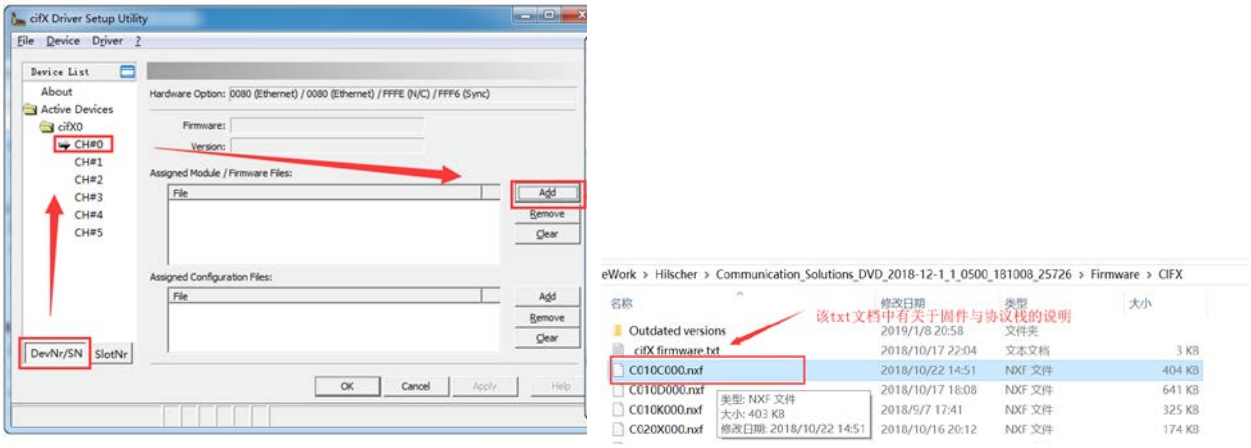


图 3.1 加载固件

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

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。

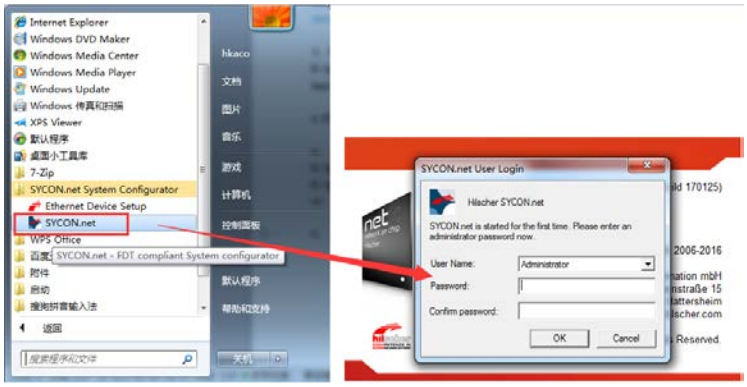
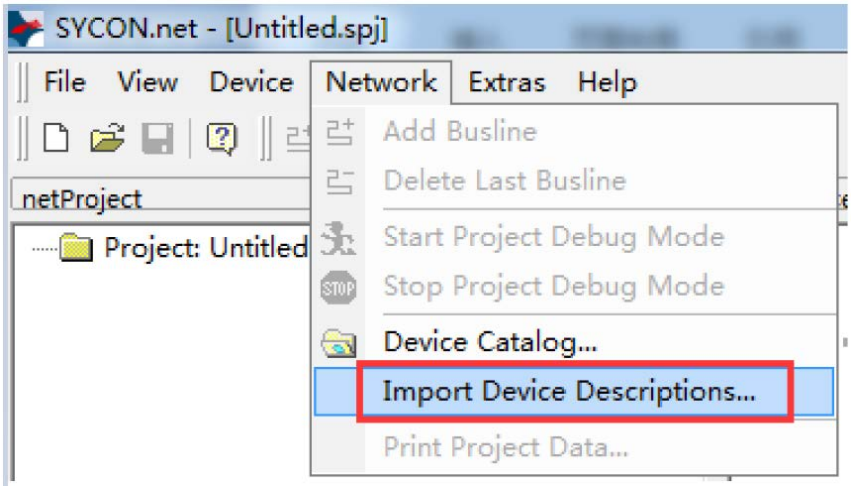


图 5.1，打开 SYCON.net 软件

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。



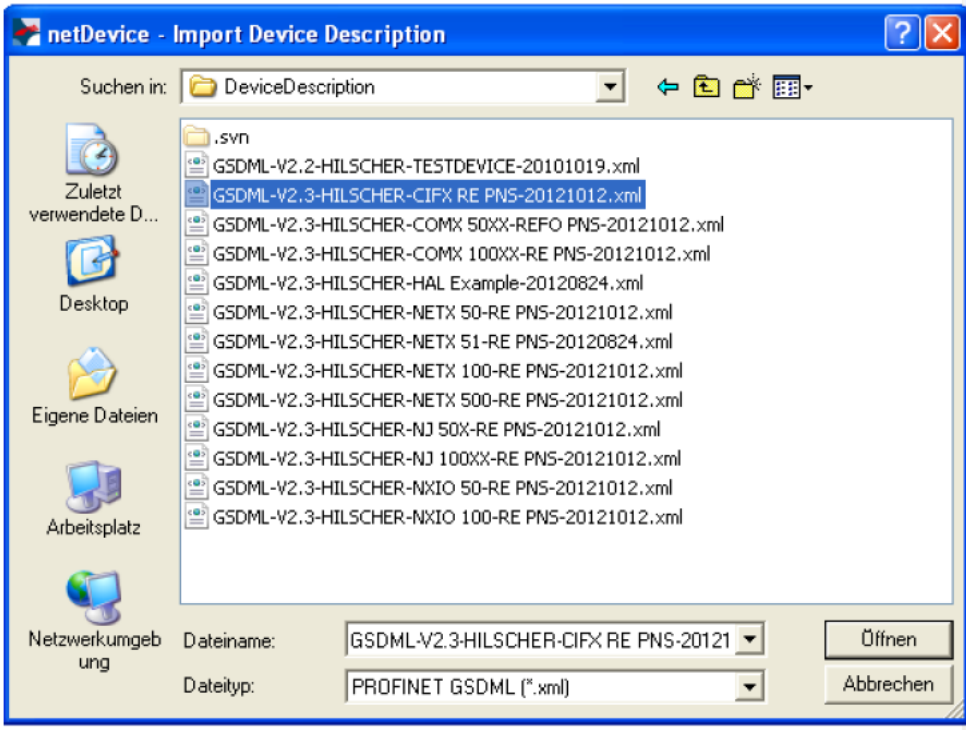


图 5.2, 添加从站 GSDML 文件

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。

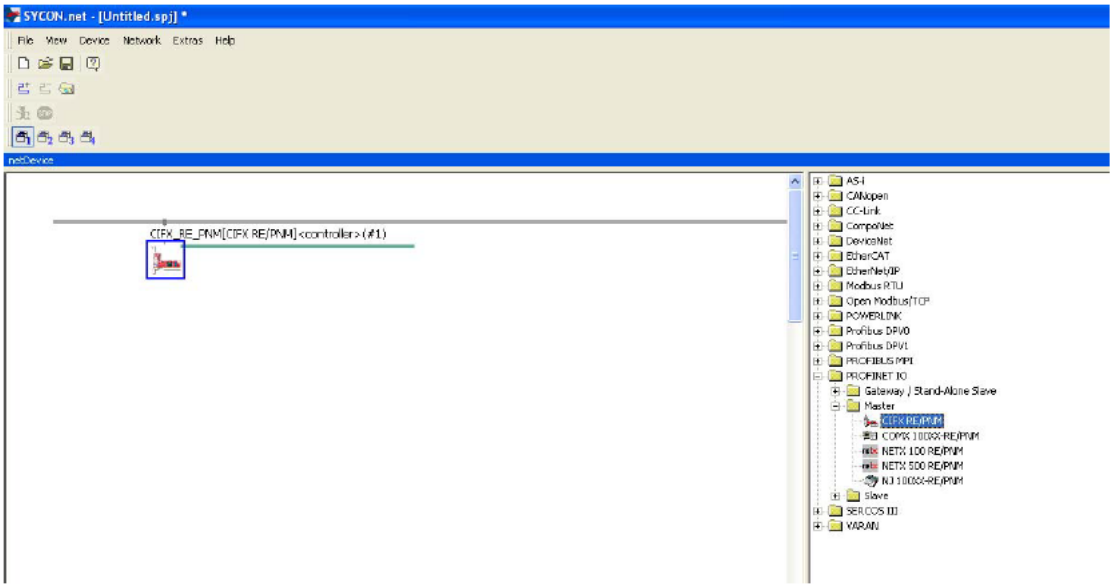


图 5.3，添加 CIFX 板卡

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。

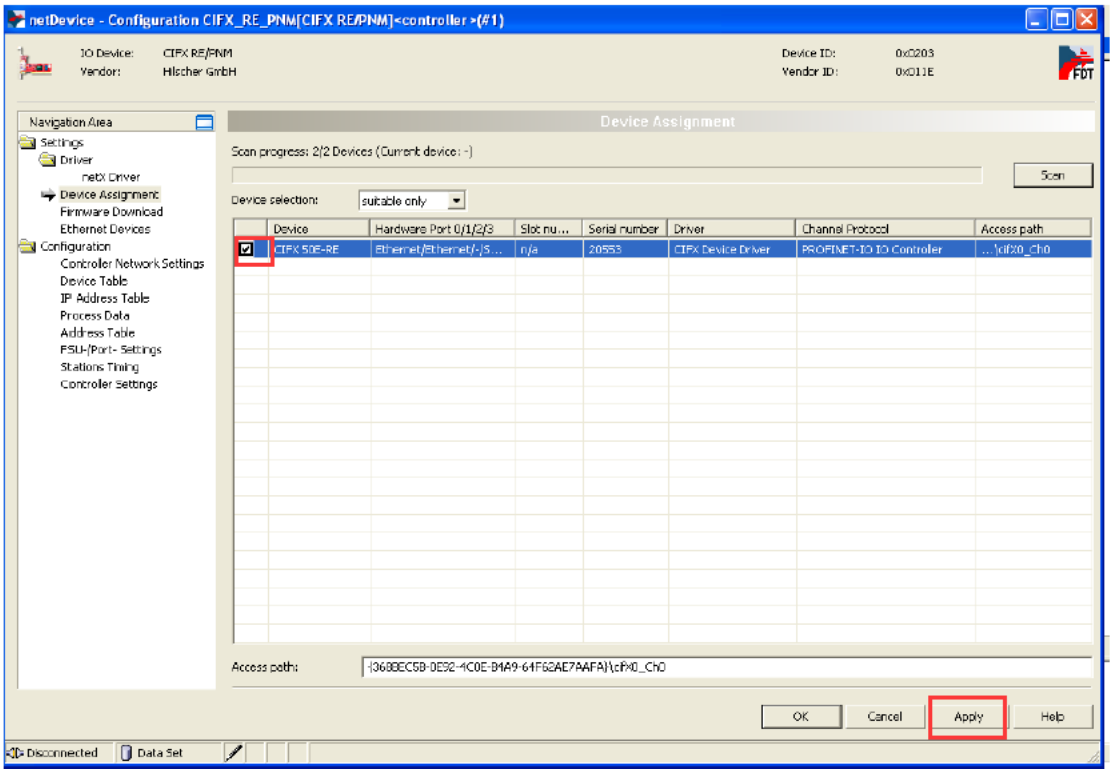


图 5.4，Device Assignment

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。

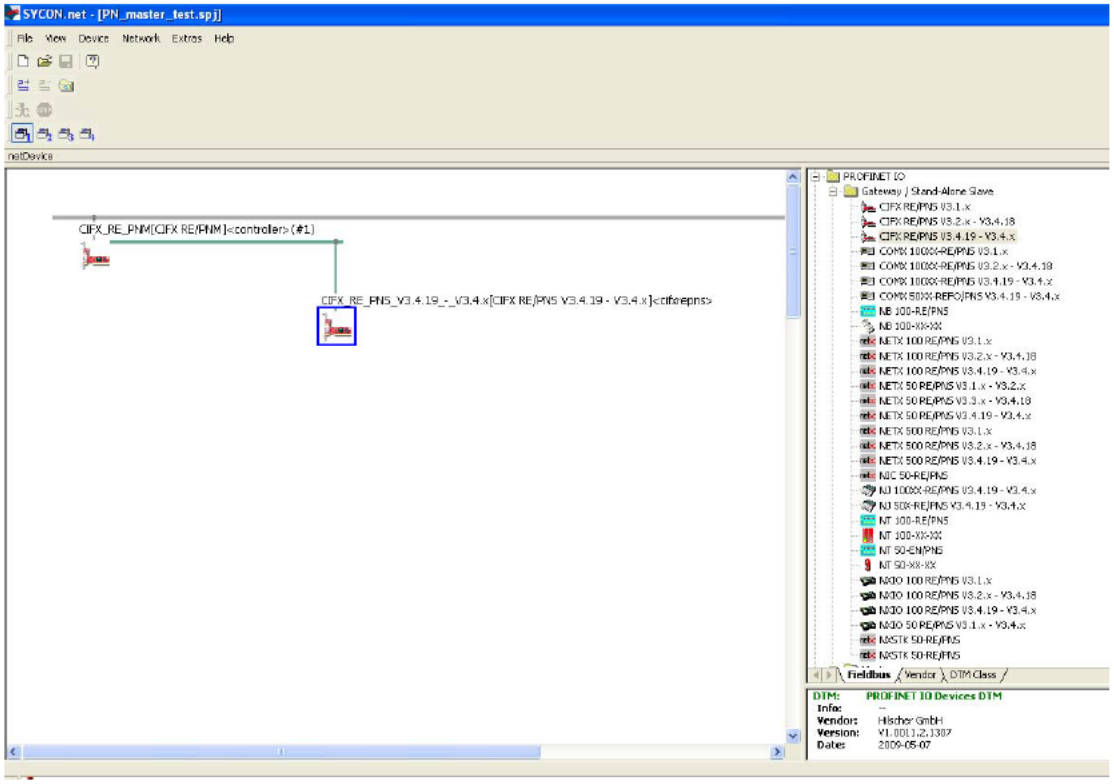


图 5.5，Add Slaves

5.4 PROFINET 主从站配置

双击 CIFX 图标打开主站配置，其中可以配置主站的 Cycle time，初始化状态，同步模式，通信触发，看门狗等，如图 5.6，5.7。

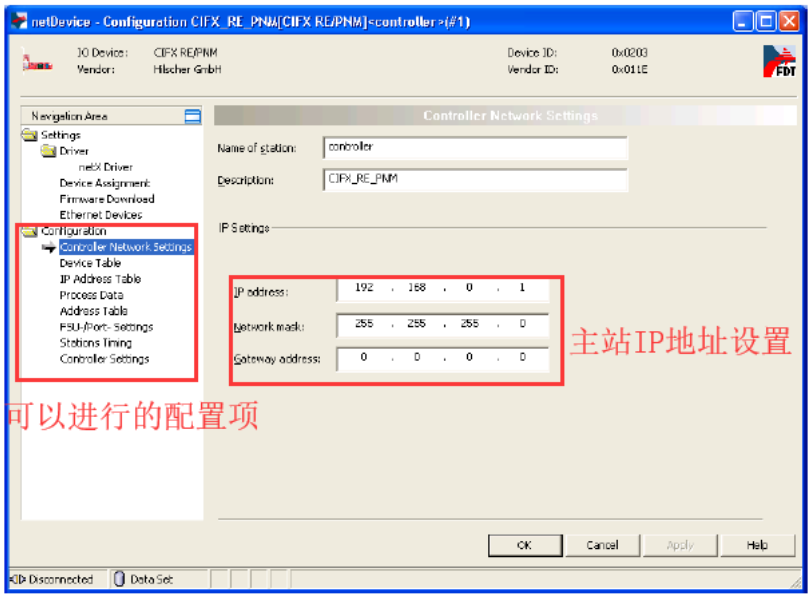


图 5.6，主站配置

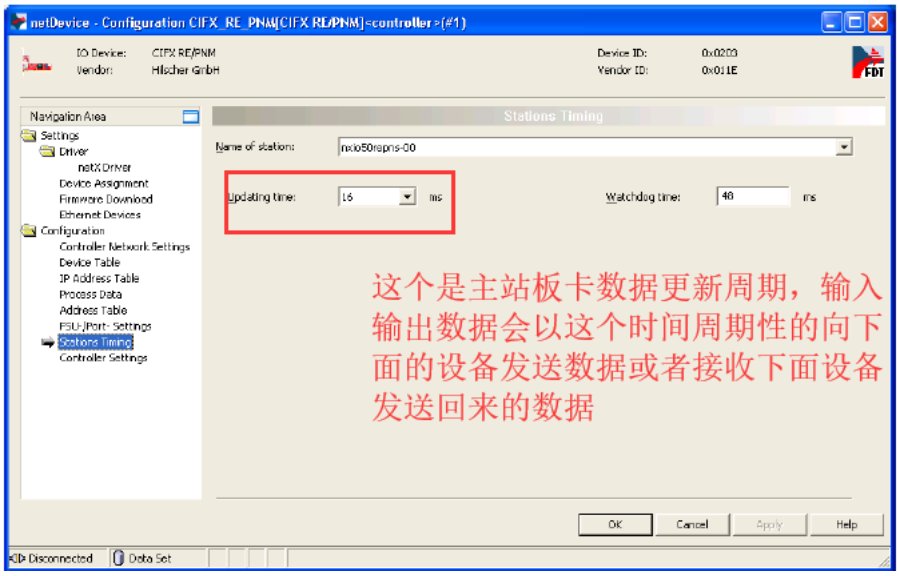


图 5.7，更新周期

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。

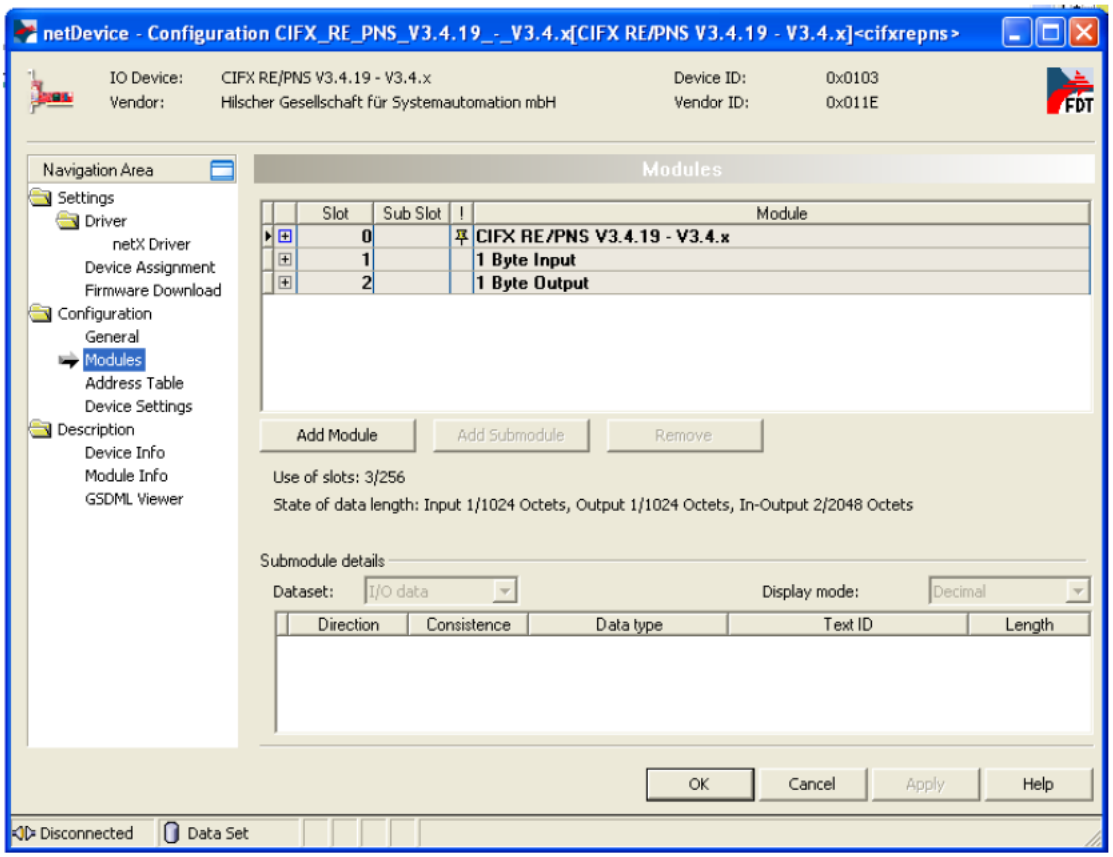


图 5.8，从站 Module 配置

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。

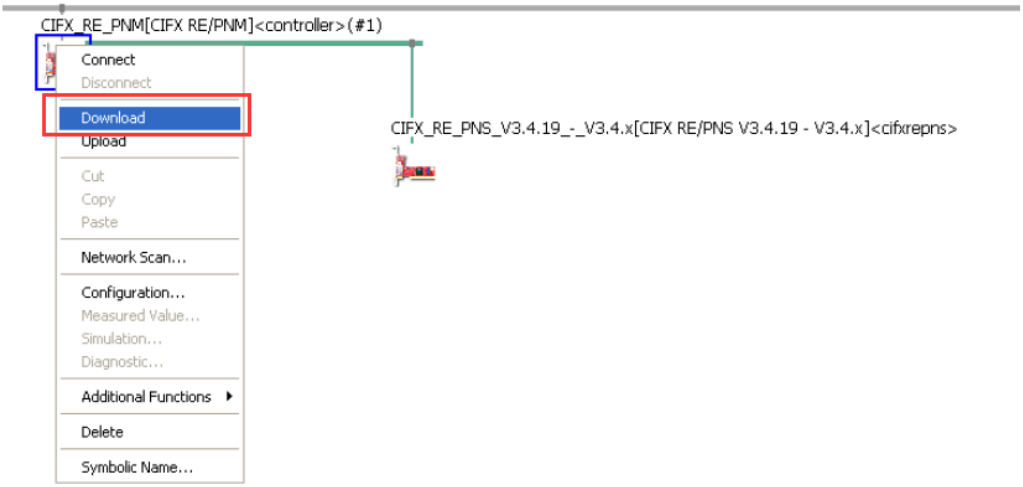


图 6，下载配置

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。

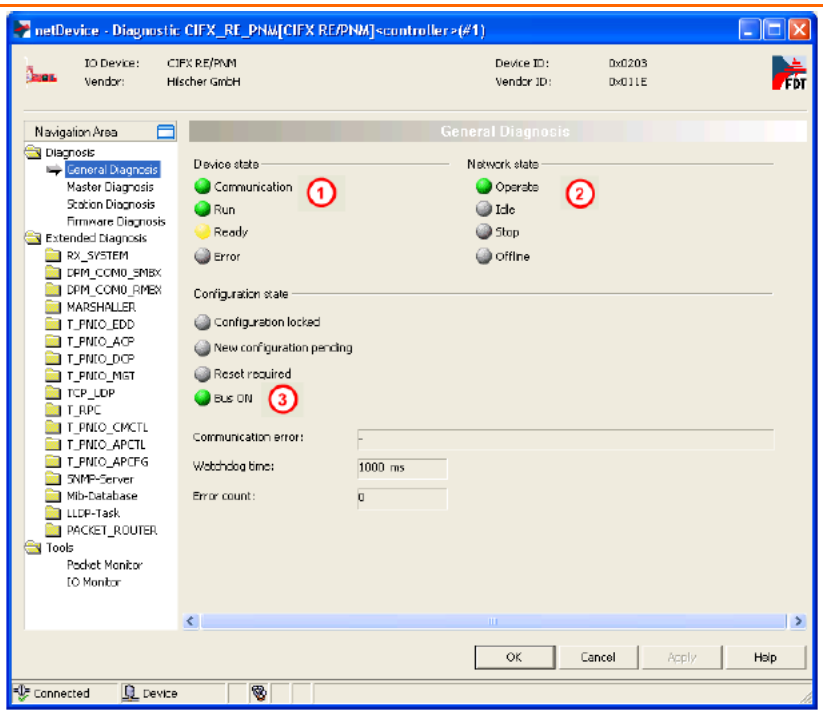


图 7.1，Diagnosis

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。

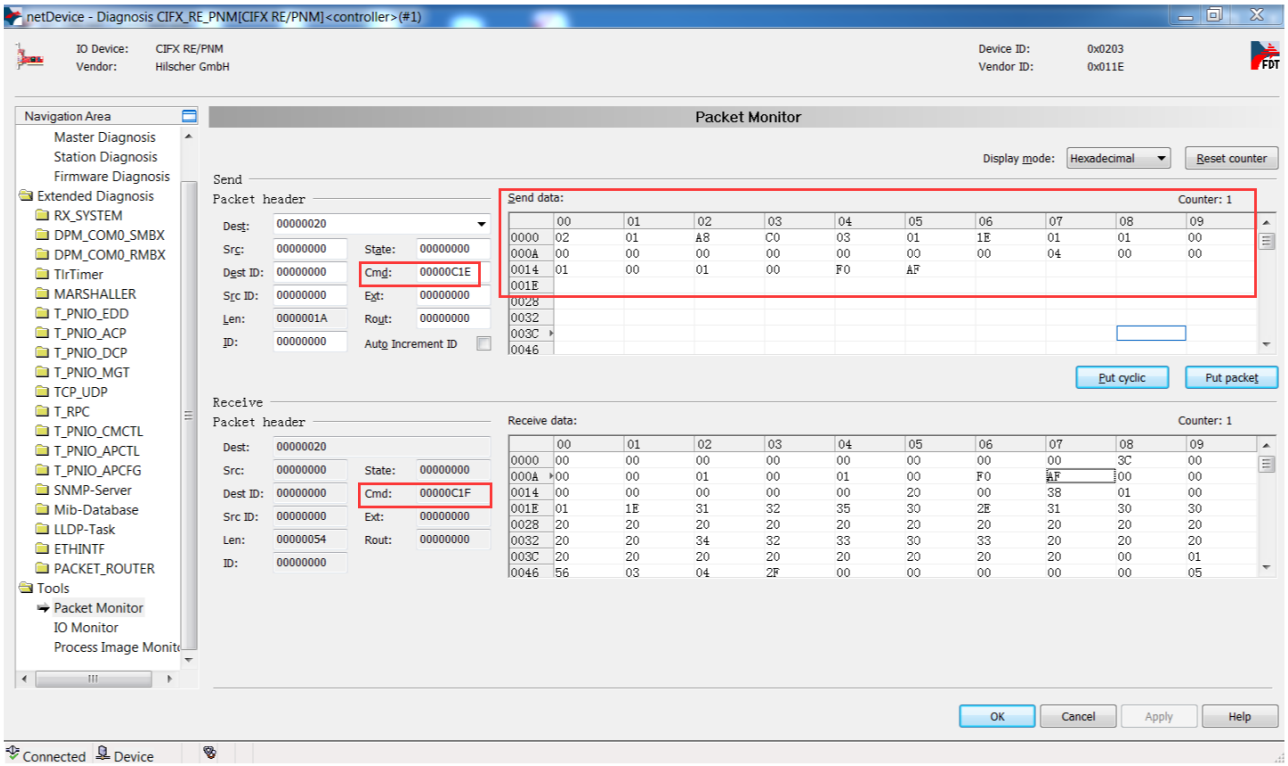


图 7.2，Packet Monitor

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：

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

1. 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；
2. 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；
3. 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）

4. 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 0x0080 0x00C0 0x0040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet 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.	

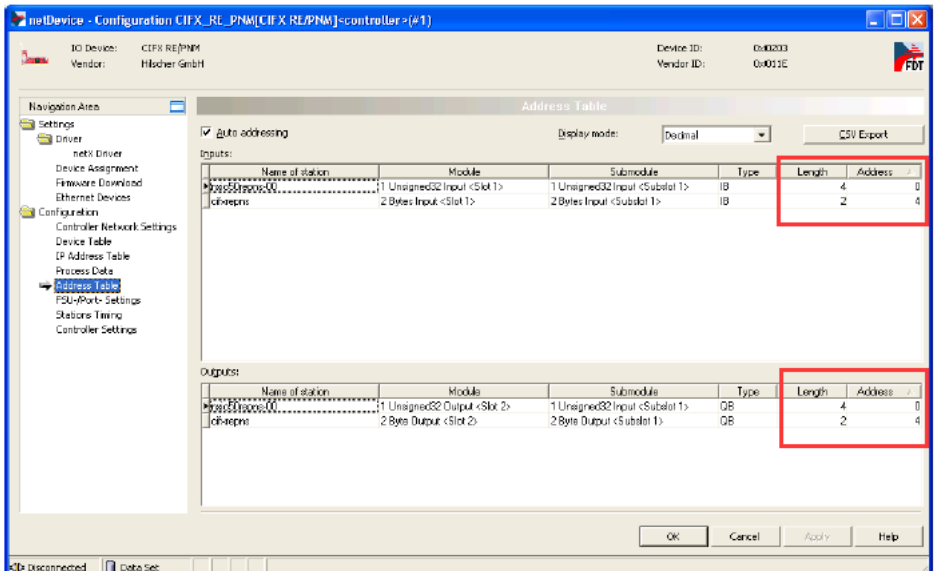
图 7.5，PROFINET 异步服务数据包结构（Confirmation）

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：



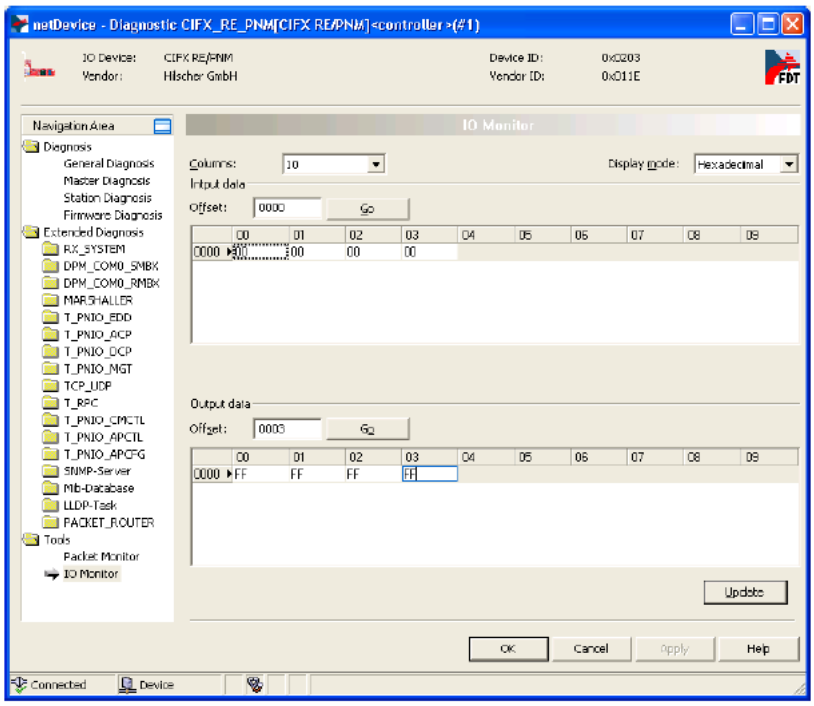


图 7.6，IO Monitor

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。

1. 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；
2. 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；
3. 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：

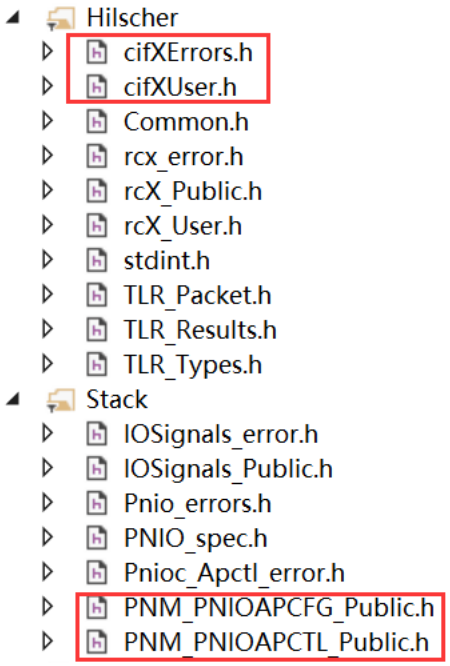


图 8.1，头文件

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：

1、 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：

```
  /* 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 0x0080 0x00C0 0x0040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet 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：

```
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, ptPcl
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：

```
750  void App_HandleProcessData(APPLICATION_T* ptApp)
751  {
752    /* return value of function */
753    TLR_RESULT tResult = TLR_S_OK;
754
755    /* cyclic io handing happens here */
756    /* read RTD-Data */
757    tResult = xChannelIORead(ptApp->tCommon.hChannel, 0, 0, ptApp->tCommon.ulReadBufferSize, ptApp->tCommon.pabReadBuffer, ptApp->tCommon.ulReadBufferSize);
758
759    /* the action depends on the return value we have received */
760    if ((CIFX_DEV_EXCHANGE_FAILED == tResult) || (CIFX_DEV_EXCHANGE_TIMEOUT == tResult))
761    {
762      DEBUG("Read failed with 0x%08x\n", tResult);
763    }
764    else if (CIFX_DEV_NO_COM_FLAG == tResult)
765    {
766      //DEBUG("Read failed. Device is not communicating\n");
767    }
768    else if (CIFX_NO_ERROR == tResult)
769    {
770      if (g_bInputData != ptApp->tCommon.pabReadBuffer[0])
771      {
772          DEBUG("Input data changed from 0x%02x to 0x%02x\n", g_bInputData, ptApp->tCommon.pabReadBuffer[0]);
773      }
774      g_bInputData = ptApp->tCommon.pabReadBuffer[0];
775
776      //DEBUG("Read succeeded. Copy Inputs back to Outputs.\n");
777
778  #if 1
779      /* we copy the memory we have read to the memory we want to send, */
780      /* because we just want to mirror the data */
781      memcpy (ptApp->tCommon.pabWriteBuffer, ptApp->tCommon.pabReadBuffer, ptApp->tCommon.ulWriteBufferSize);
782
783      // write IO-Data
784      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。