HK/CIFX 板卡作为 EtherCAT 主站的基本使用举例

——万彬，技术支持工程师，2021.07.30

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 EtherCAT 主站，并与多个从站 IO 模块进行通信测试，其中包含三个 EIM 模块以及三个倍福的模块。

HK/CIFX 板卡 EtherCAT Master (V4)版本固件支持功能如下：

Parameter	Description
Maximum number of EtherCAT slaves	Maximum of 388 slaves, if RCX_GET_SLAVE_HANDLES_REQ service is used for determining number of slaves. The number of usable slaves depends on several parameters: the available memory for the configuration file (see 'configuration file' below), used cycle time, frame propagation time.
Maximum number of cyclic input data	Appr. 4600 bytes, if no LRW command (Logical Read Write) is used for process data
Maximum number of cyclic output data	Appr. 4600 bytes, if no LRW command (Logical Read Write) is used for process data
Acyclic communication	CoE (CANopen over EtherCAT): SDO, SDOINFO, Emergency FoE (File Access over EtherCAT) SoE (Servo Drive Profile over EtherCAT) EoE (Ethernet over EtherCAT) Configurable with SYCON.net: CoE If the file ETHERCAT.XML contains the appropriate configuration information (e.g. created with "EtherCAT Configurator"), following functions can be used: CoE, SoE, EoE
Mailbox protocols	CoE, EoE, FoE, SoE
Functions	Distributed Clocks Redundancy Slave diagnostics Bus scan
Minimum bus cycle time	250 µs, depending on the used number of slaves and the used number of cyclic input data and output data.
Topology	Line or ring
Slave station address range	1 – 14335
Data transport layer	Ethernet II, IEEE 802.3, 100 MBit/s, full-duplex
Configuration file (ETHERCAT.XML or CONFIG.NXD)	Maximum 1 MByte
Synchronization via ExtSync	Supported (not configurable with SYCON.net)
ENI Slave-to-Slave copy infos	Supported (not configurable with SYCON.net)
Hot Connect	Supported (not configurable with SYCON.net)
EoE (Ethernet over EtherCAT)	Via NDIS
Limitations	The size of the bus configuration file is limited by the size of the RAM disk (1 MByte) or Flash disk (3 MByte). Store-and-forward switches cannot be used within network topology due to hard receive timing model RCX_GET_SLAVE_HANDLES_REQ can only communicate up to 388 slaves. Process data is restricted by the dual-port memory to 5760 bytes.
Reference to firmware / stack version	V4.4

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 48 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 09 EN》。

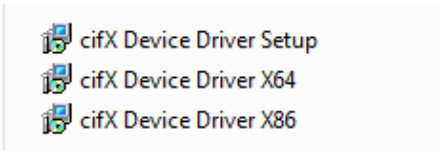


图 2.2，CIFX 的 Windows 驱动

路径：Communication_Solutions_2023-07-1\Driver_&_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。

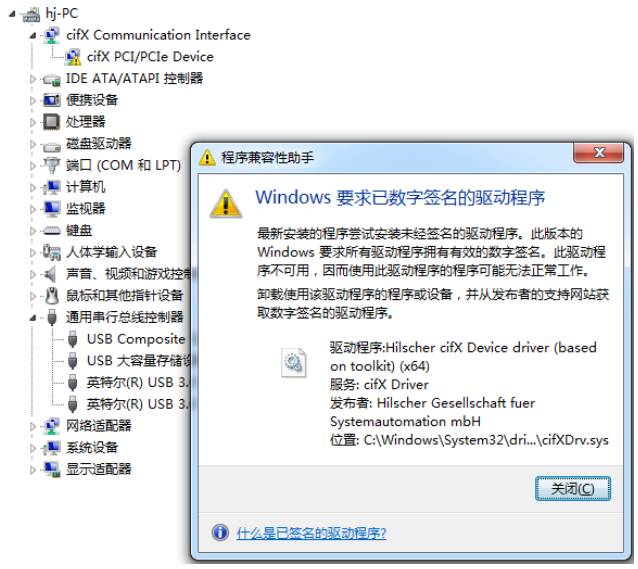


图 2.4，未经签名的驱动程序

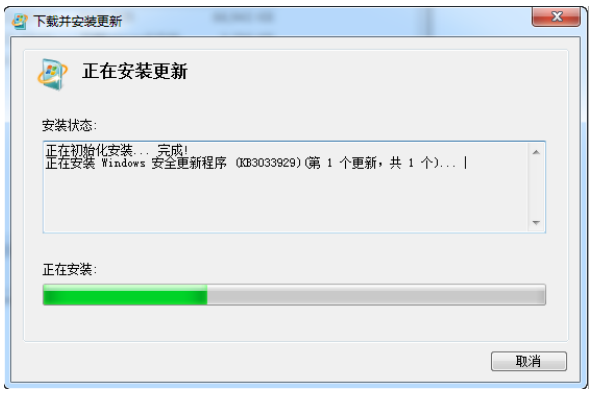
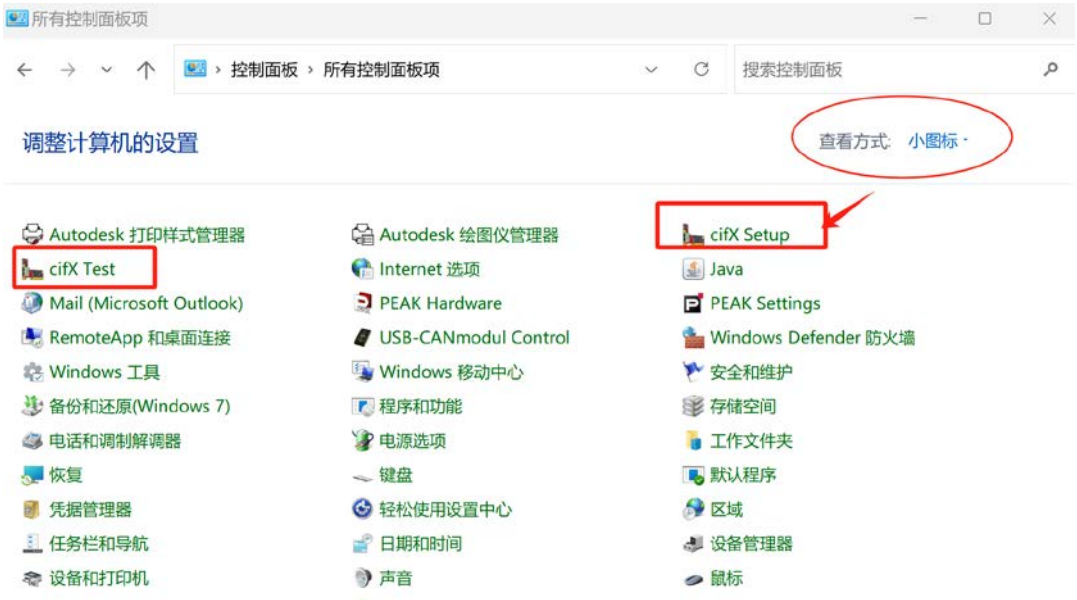


图 2.5，补丁 KB3033929 安装

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。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。

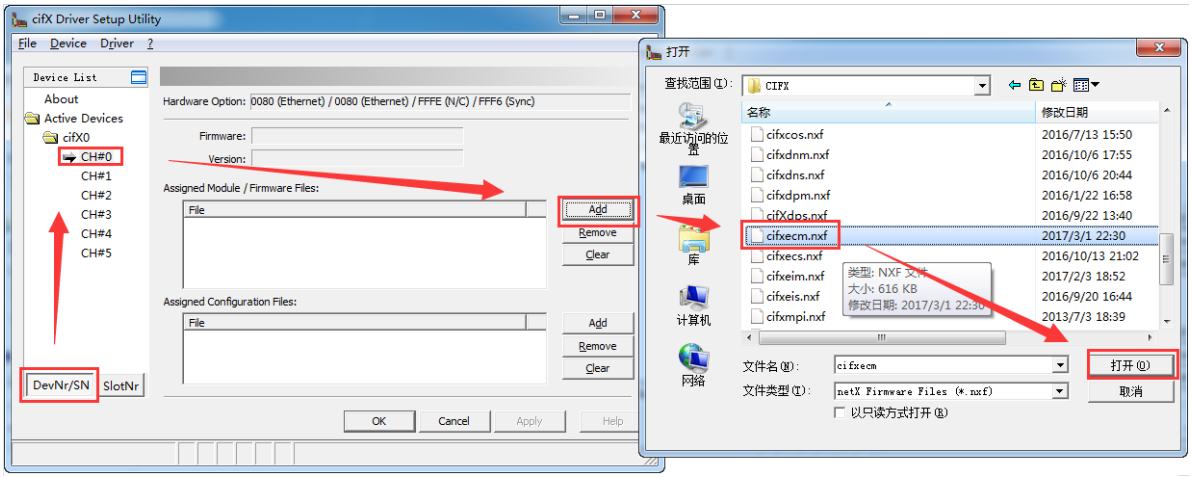


图 3.1 加载固件

固件最新路径：

Communication_Solutions_2023-07-1\Firmware,_EDS,_Examples,_Webpages\Firmware_&_EDS\COMSOL-ECM V4.5.0.9\Firmware\cifX

3.2 cifX Test

打开 cifX Test 工具，点选 Device——Open——cifX0——Channel0——Open，打开对应的通道，如图 3.2。选择 Information 可查看板卡的信息，可以看到板卡的 Channel0 已经加载了 EtherCAT Master 固件，版本为 4.4.0.0，如图 3.3。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）。

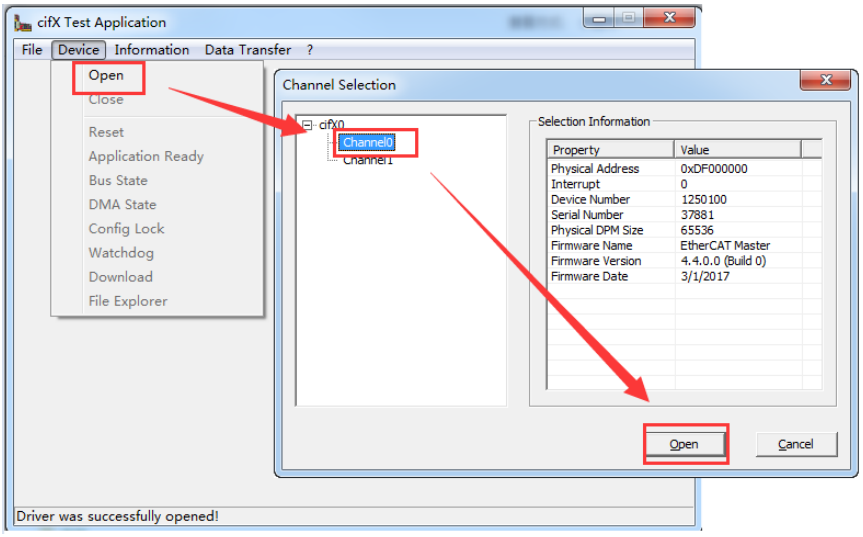


图 3.2，cifX Test 打开板卡通道

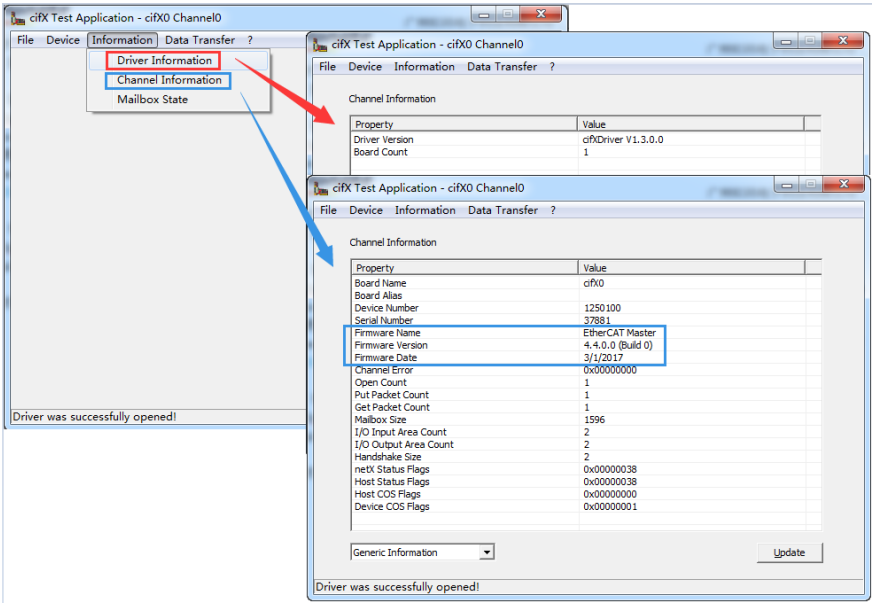


图 3.3，cifX Test 查看板卡固件信息

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 EtherCAT 主站，需要用 SYCON.net 软件进行 EtherCAT 网络的组态，并下载给 CIFX 板卡，让 CIFX 板卡知道网络中各个 EtherCAT 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，如切换网络状态，CoE 通信，点亮某个 DO，采集某个 DI 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_2017-08-1_V1_400_170125_19044\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。

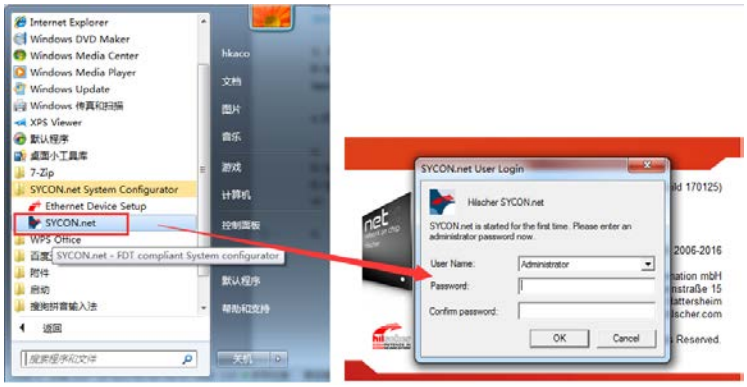


图 5.1，打开 SYCON.net 软件

5.2 添加从站 ESI 文件到 SYCON.net 软件

即添加从站的 ESI 文件（.xml 文件）到 Device Catalog 中，点选菜单栏 Network——Import Device Descriptions...——选择.xml 文件——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。

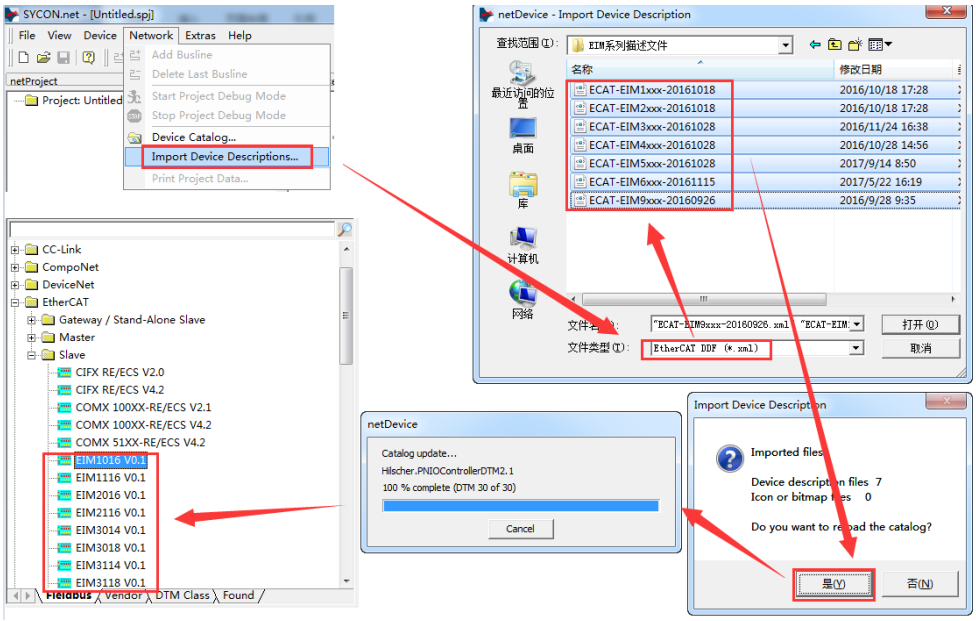


图 5.2,添加从站.xml 文件

5.3 添加 CIFX 板卡并扫描从站

在 Device Catalog 中找到 EtherCAT——Master——CIFX RE/ECM，并用鼠标左键将 CIFX 拖拉到界面中间的灰线上，如图 5.2。

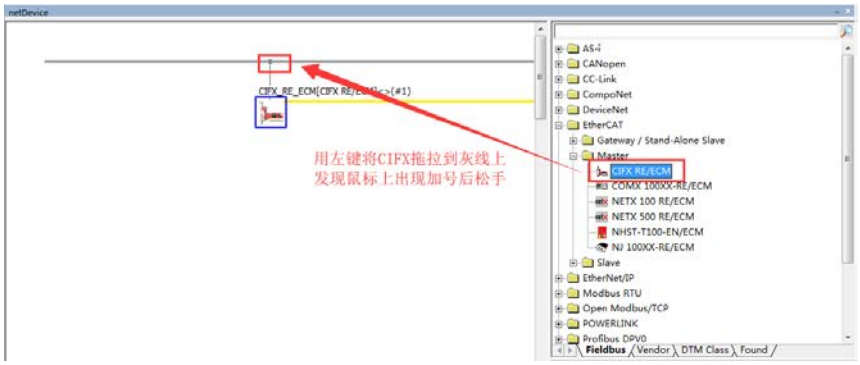


图 5.3，添加 CIFX 板卡

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。

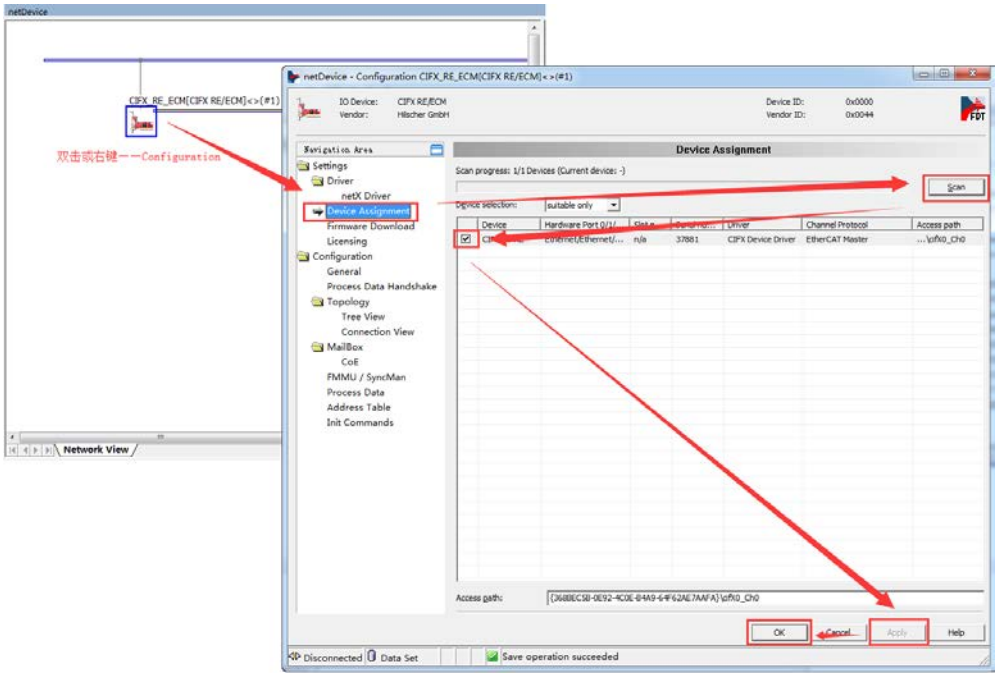


图 5.4，Device Assignment

在 CIFX 图标上右键——Network Scan，扫描出从站，注意在 Network Scan 之前应先给从站上电，并用网线串接到 CIFX 板卡上（接在靠近指示灯的那个口上），如图 5.5。如果扫描出来后无法识别从站的信息，那可能是因为步骤 5.2 漏了，也可能是因为加载的从站.xml 文件不包含这一从站的信息，需要联系从站厂家拿最新的.xml 文件。

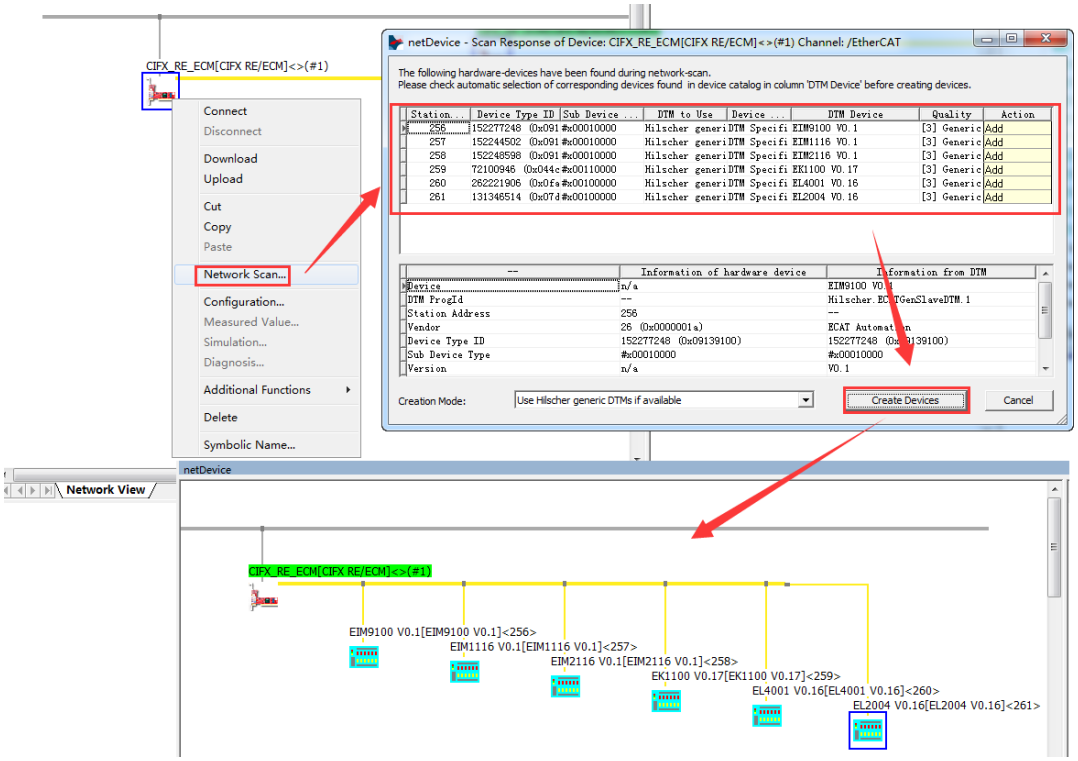


图 5.5，Network Scan

5.4 主从站 EtherCAT 配置

在 CIFX 图标上右键——Disconnect，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。

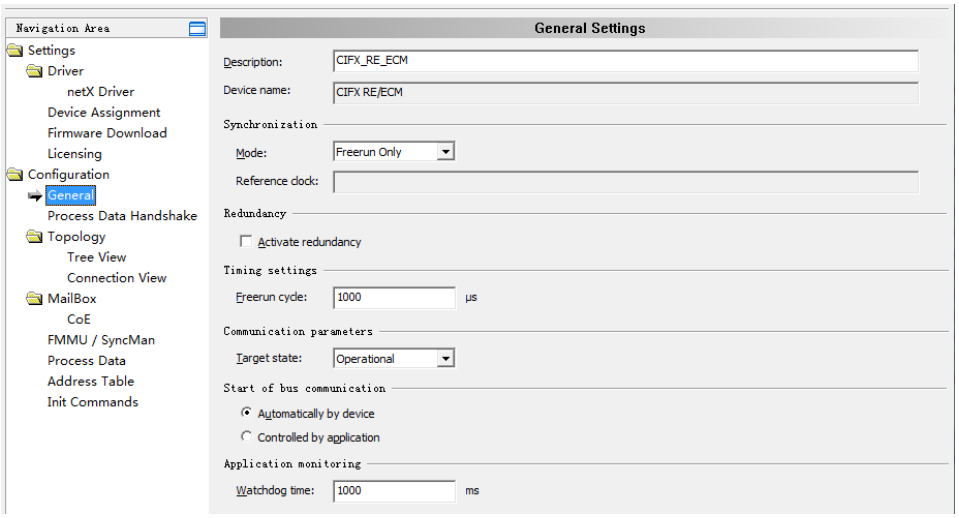


图 5.6，主站配置

除此之外，还可以看到网络拓扑，Mailbox，Address Table 等，如图 5.7。这里需要注意的是 Tree View 中的 Station address 是作为 CoE 通信的地址，而 Address Table 是作为 PI（Process Image，即实时通信数据）通信的地址，Address Table 把网络中所有从站的 PI 数据按照拓扑的顺序排列，即每个从站的每个 PI 数据在 Address Table 中都有唯一的 Address，且二次开发包在进行数据交换时也使用这一 Address 来确定通信的 PI 数据对象。

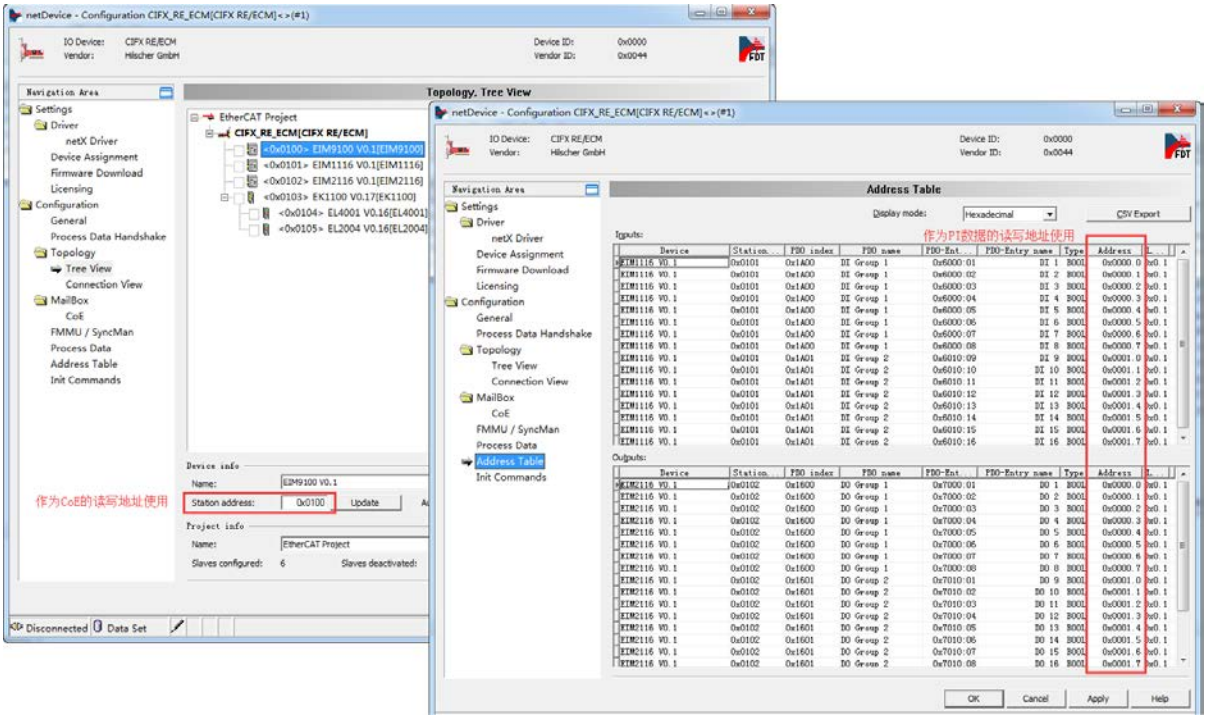


图 5.7，CoE 与 Address Table

主站配置完成后，双击每一个从站，可以对从站进行配置，其中包括分布时钟 DC 的设置，Process Data 中数据映射的设置等，如图 5.8，本例中不需要用到分布时钟，所以没有使能 DC 功能，需要时选择相应的 DC 模式即可。

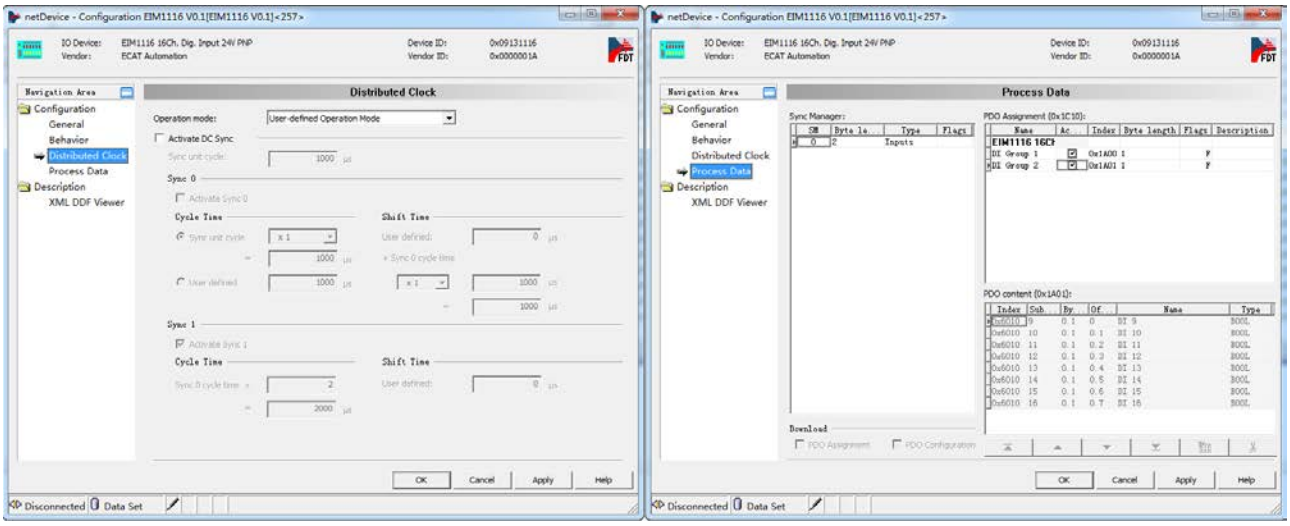


图 5.8，从站分布时钟与过程数据设置

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。

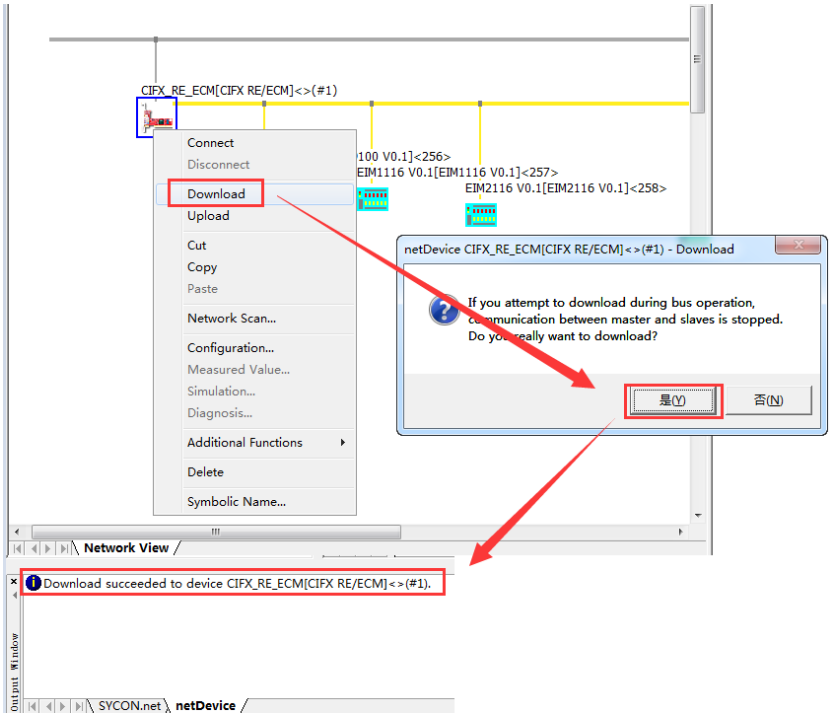


图 6，下载配置

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。

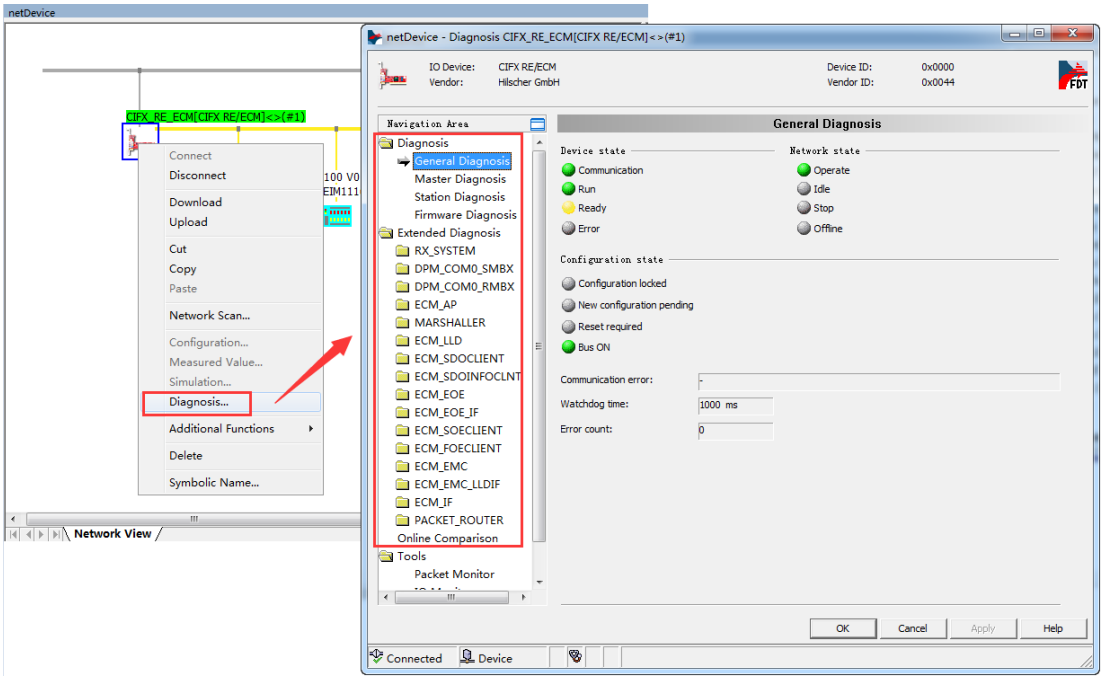


图 7.1，Diagnosis

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，Start of bus communication 选项选择的是 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。

再看图 5.6 中 Communication parameters 选项选择的是 Operational，表示板卡 Run 之后自动把 EtherCAT 网络切换到 OP 模式，所以此时的监控状态中可以看到是 Communication+Run。另外也可以看到 Network state 为 Operate，说明网络已经在进行数据交换了，而当 Network state 为 Idle 时，表示 EtherCAT 网络不在 OP 状态。

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 EtherCAT 状态的切换，CoE，FoE 等通信；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，如切换 EtherCAT 网络状态与 CoE 通信；IO Monitor 用于测试周期数据，如点亮 DO，采集 DI；Process Image Monitor 用于查看每个从站 PI 数据的当前值。

7.2.1 用 Packet Monitor 测试切换 EtherCAT 状态

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 12 EN》第 80 页，这里不做过多解释。

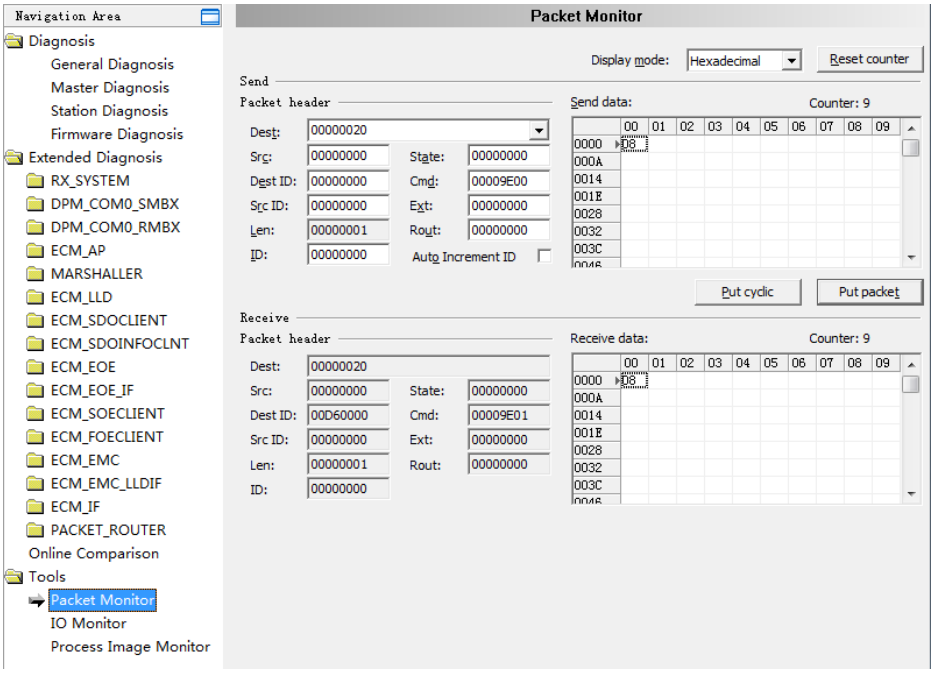


图 7.2，Packet Monitor

但需要特别注意的是：

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，该测试是为了切换 EtherCAT 网络的状态，即 Init，PREOP，SAFEOP，OP，Cmd 的值为 0x9E00，可参考光盘文档《EtherCAT Master V4 Protocol API 05 EN》第 29 页；
3. Data 的值定义：0x01 = INIT，0x02 = PREOP，0x04 = SAFEOP，0x08 = OP；
4. 由于 Data 的长度 1 个字节，所以 Len = 0x01。

如图 7.3 所示，由于板卡上电后自动 Run 并进 OP，所以 Device state 为 Communication+Run，Network state 为 Operate，且板卡的 COM0 灯和 IO 模块 RUN 灯常亮。

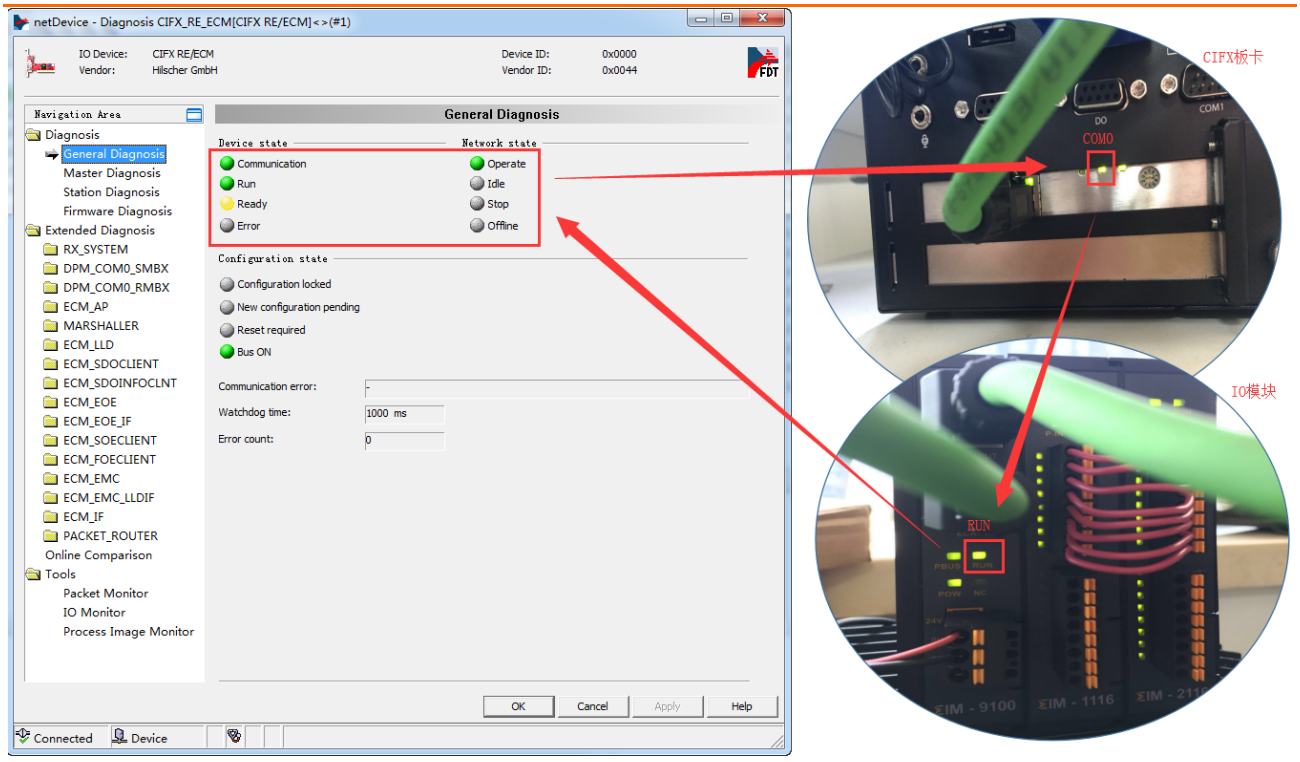


图 7.3，OP 状态下

接着把 Send data 的值设置为 0x01，然后发送 Packet，让网络进入 INIT，如图 7.4。成功切换状态后会发现 Device state 为 Run，Network state 为 Idle，且板卡的 COM0 灯和 IO 模块 RUN 灯常暗，如图 7.5。

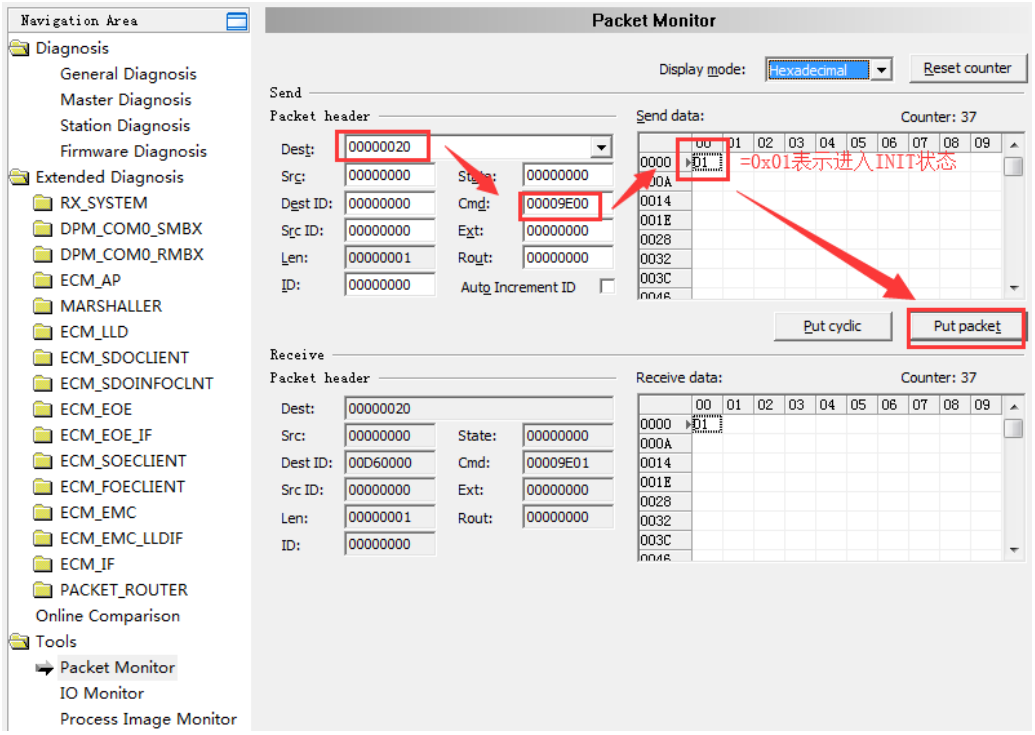


图 7.4，发送 Packet 进入 INIT 状态

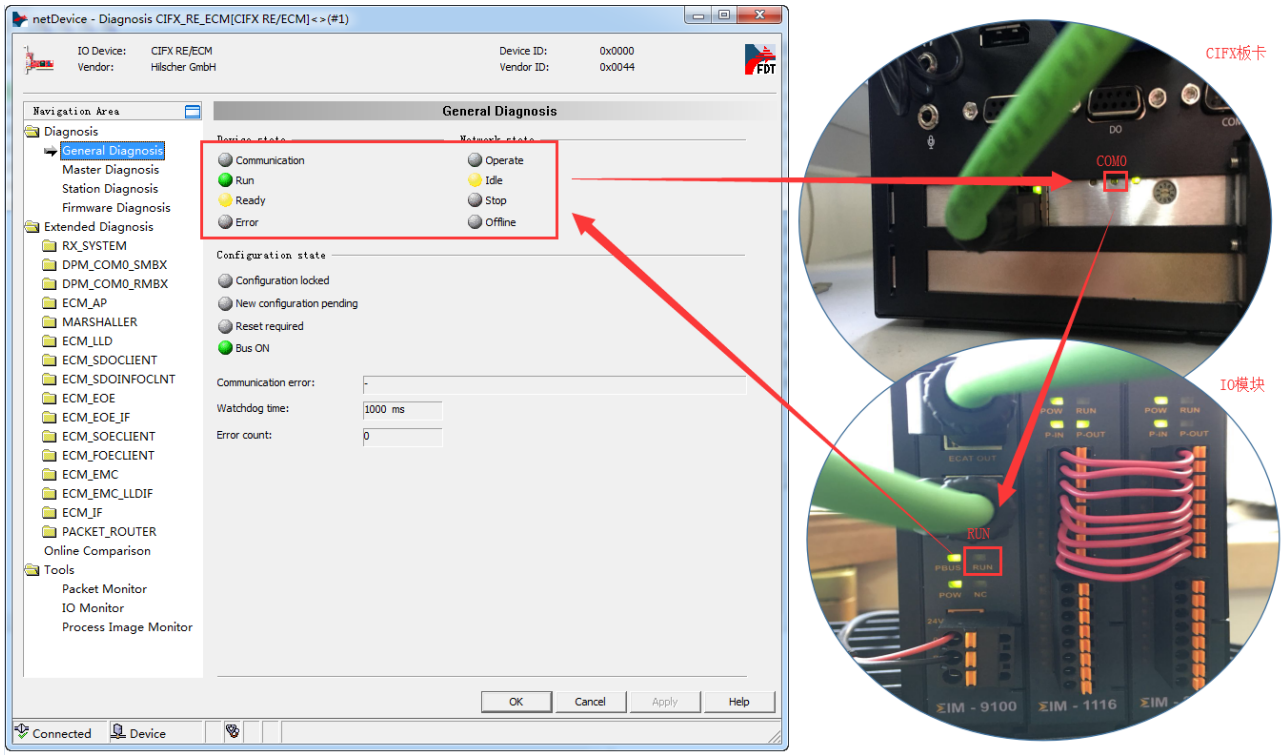


图 7.5，INIT 状态下

至此，用 Packet Monitor 测试切换 EtherCAT 状态已结束，如果想了解更多的非周期通信功能，请查看文档《EtherCAT Master V4 Protocol API 05 EN》。

7.2.2 用 Packet Monitor 测试 CoE 通信

与上一小节的状态切换测试类似，CoE 通信也是通过 Packet 来实现的，只是 CoE 用的 Cmd，Len，Data 等不同而已。

本例中，先读一下模块 EL4001 的 Index 0x8000 sub-Index 0x01 的值，发现默认值为 0x00，然后再给这个对象写值 0x01，再读回来，看是否为 0x01。如图 7.6 为读 0x8000:01 时的 Packet 定义与反馈的 Packet。

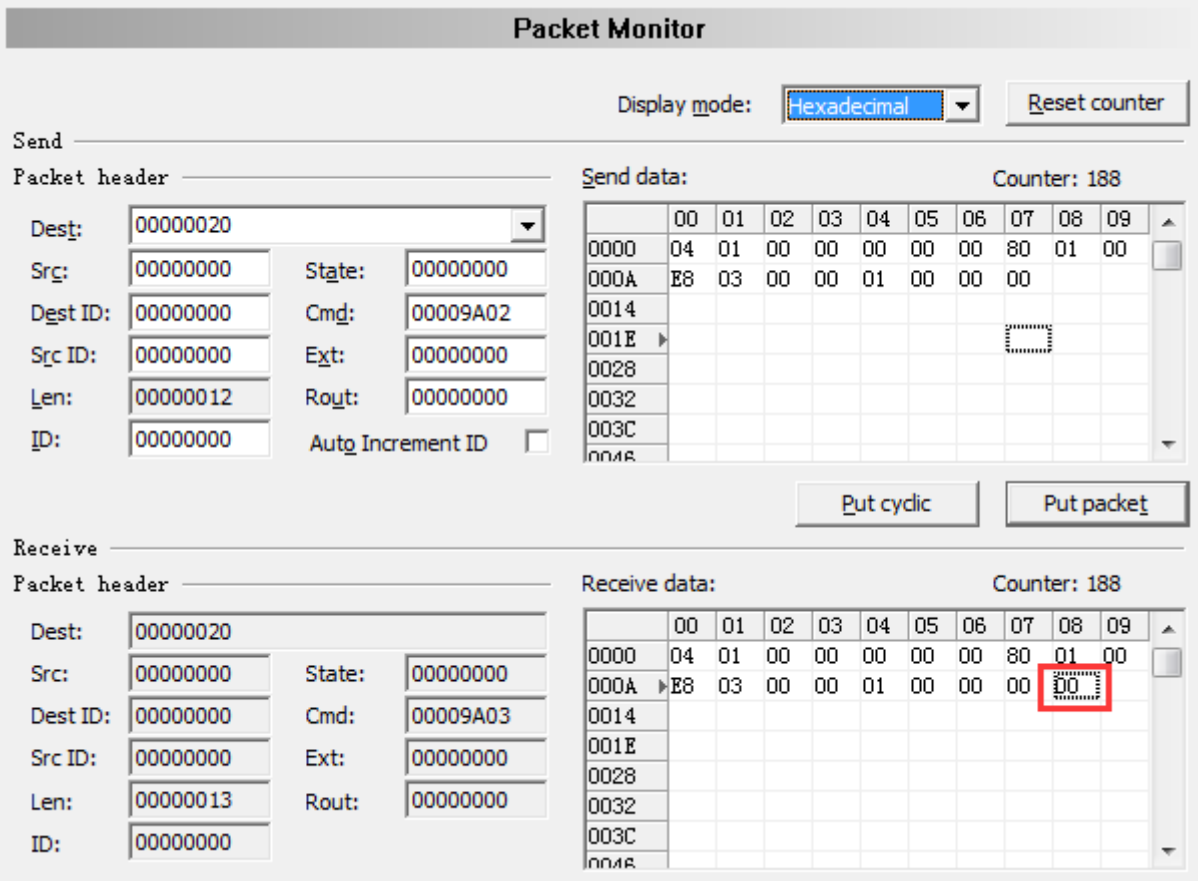


图 7.6，读 0x8000:01 的值为 0x00

1. Dest = 0x20；
2. Cmd = 0x9A02 表示 CoE read；
3. Data 各部分定义如下表:

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11
Station Address =0x0104		TransportType =0x00		AoEPort		ObjIndex =0x8000		SubIndex =0x01	CompleteAccess	TimeoutMs=0x03E8=1000ms				MaxTotalBytes=0x01=1Byte			
04	01	00	00	00	00	00	80	01	00	E8	03	00	00	01	00	00	00

注：从图 5.7 的主站配置中可以看到，模块 EL4001 的 Station address 为 0x0104。其它参数可参考 EtherCAT Master V4 Protocol API 05 EN》第 86 页；

4. Len 为 0x12，即 18 个字节，由 Data 的长度决定。

从图 7.6 中也可以看到，反馈回来的 Packet 中，Data 部分比 Send data 多了最后一个字节，这一字节即为读回来的数值，为 0x00，即对象 0x8000:01 的当前值为 0x00。

接着给对象 0x8000:01 写值 0x01，Packet 定义与与反馈的 Packet 如图 7.7 所示。

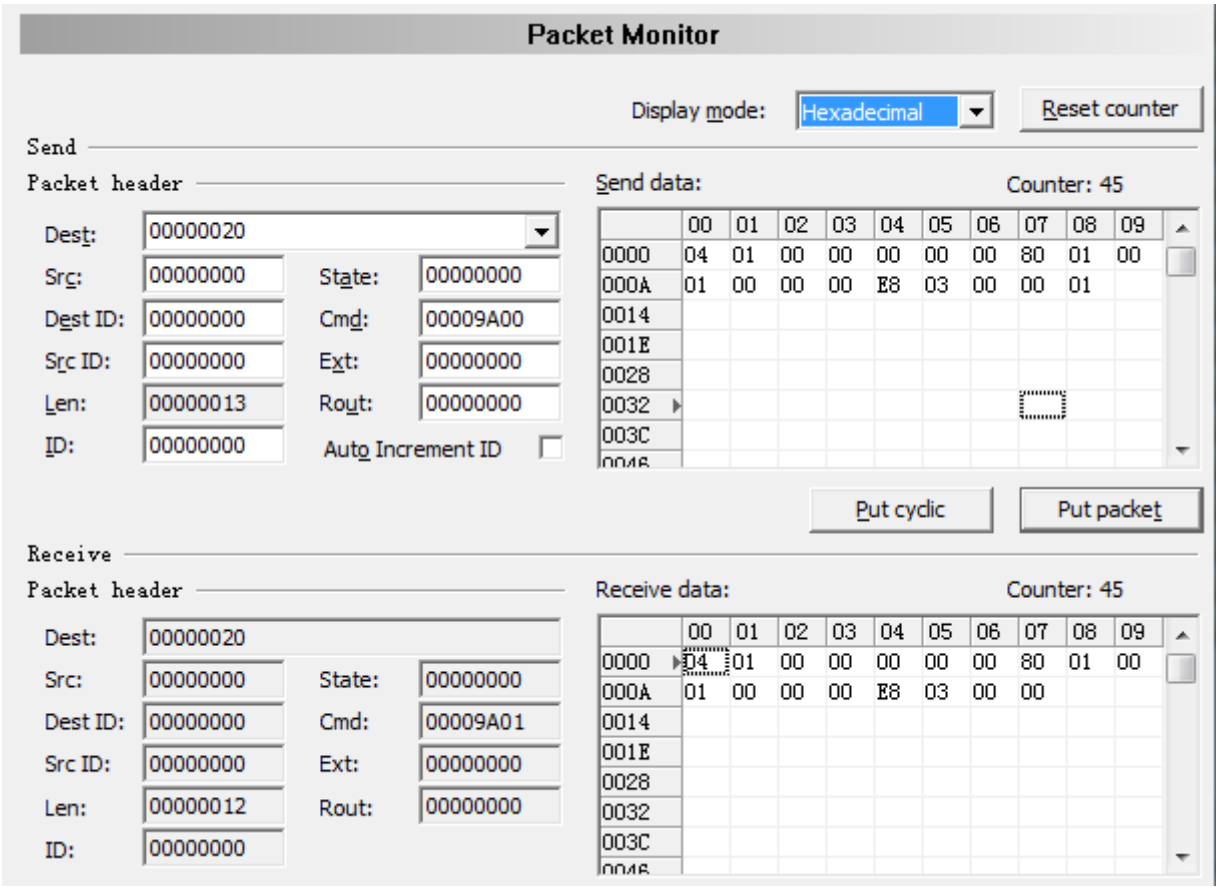


图 7.7，写 0x8000:01 的值为 0x01

1. Dest = 0x20；
2. Cmd = 0x9A00 表示 CoE write；
3. Data 各部分定义如下表：

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12
Station Address =0x0104		TransportType =0x00		AoE Port		ObjIndex=0x8000		SubIndex = 0x01	Complete Access	TotalBytes=0x01=1 Byte				TimeoutMs=0x03E8=1000ms				Data[n] =0x01
04	01	00	00	00	00	00	80	01	00	01	00	00	00	E8	03	00	00	01

注：格式与 CoE read 类似，且最后一个字节为需要给 0x8000:01 的数值。其它参数可参考 EtherCAT Master V4 Protocol API 05 EN》第 83 页；

4. Len 为 0x13，即 19 个字节，由 Data 的长度决定。

此时反馈的 Packet 无报任何错误，可以继续再用 CoE read 去读 0x8000:01 的值，发现它的值已经变成 0x01 了，如图 7.8，说明前面的 CoE write 已经成功写入了。

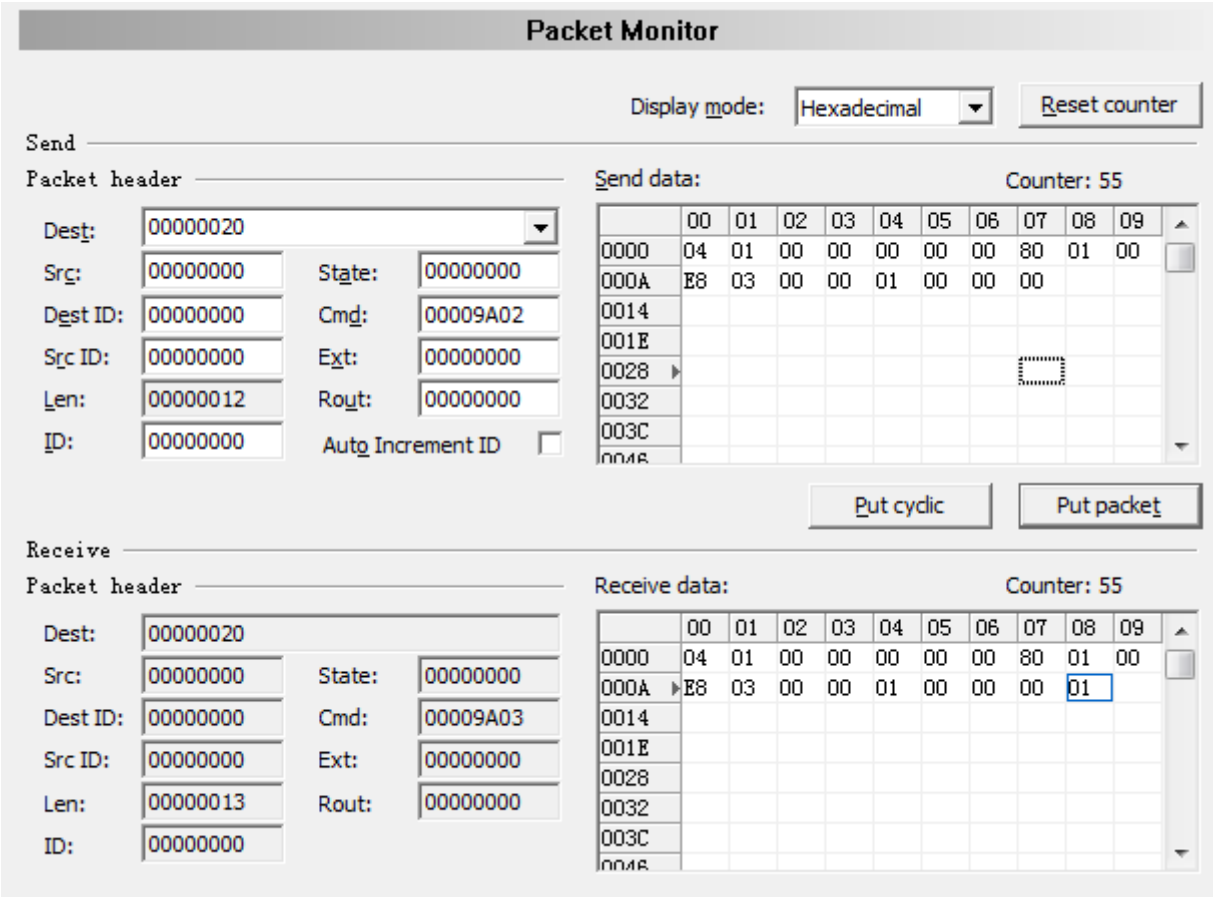


图 7.8，读 0x8000:01 的值为 0x01

至此，用 Packet Monitor 测试 CoE 通信已结束。

7.2.3 用 IO Monitor 测试 PI 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。

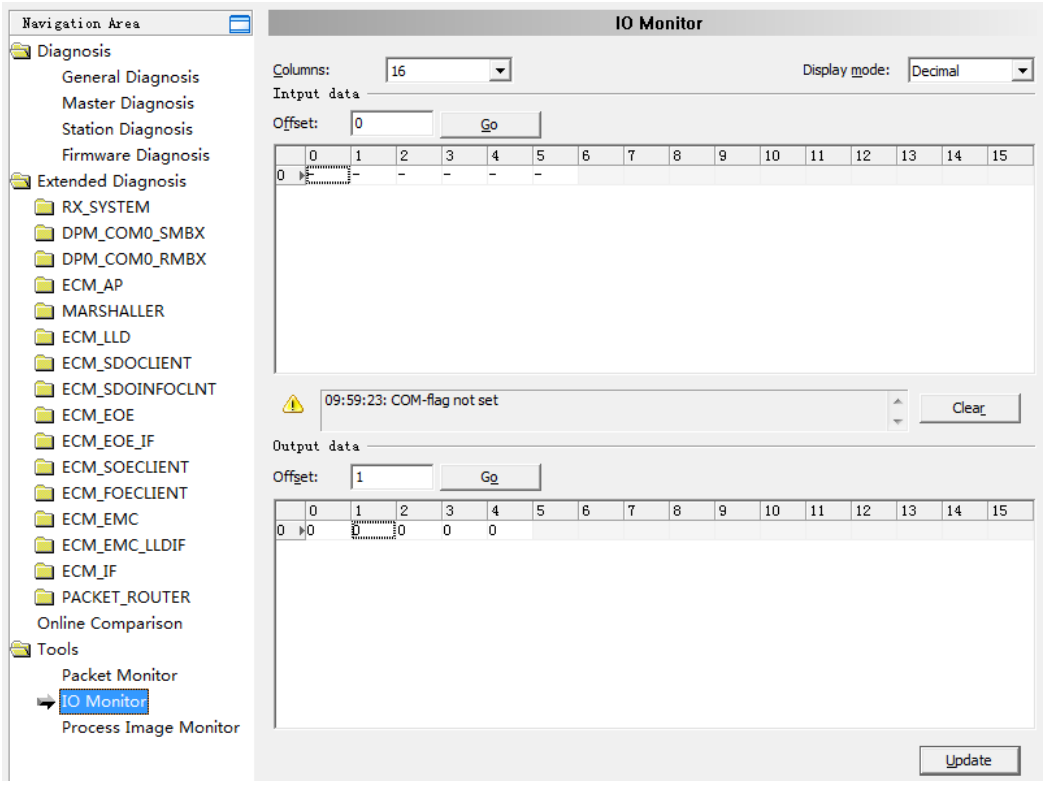


图 7.9，IO Monitor

图 7.9 中 Input 部分出现叹号，提示“COM-flag not set”，那是因为板卡的 Device state 未处于 Communication+Run 状态，通过右键 CIFX 图标——Additional Functions——Service——Start Communication 可以让板卡 RUN，再通过 Packet 切换状态机让板卡与 IO 进入 Communication 状态。板卡进入状态后再回到 IO Monitor，点击 Clear 可以清除提示。

接着来看一下数据的对应关系，这里以 EIM1116 和 EIM2116 为例，EIM1116 为 16 通道的 DI，EIM2116 为 16 通道的 DO，其中，EIM1116 的前 8 个 DI 与 EIM2116 的前 8 个 DO 相连，所以当板卡把 EIM2116 的前 8 个 DO 置 1 时，EIM1116 的前 8 个 DI 也会采集到置 1 信号。除此之外，章节 5.4 提示一个 Address Table，它把所有 PI 数据都罗列在表中了，而根据表中的每个唯一地址去访问 PI 数据，所以 EIM1116 的 DI 和 EIM2116 的 DO 也在列表中，且 EIM1116 的 16 个 DI 排在 Inputs 的最前面，即 0x0000.0~0.0001.7，而 EIM2116 的 16 个 DO 也排在 Outputs 的最前面，即 0x0000.0~0.0001.7。所以在 IO Monitor 中，Input data 的前两个字节为 EIM1116 的 16 个 DI，且第一个字节为 DI1~DI8，而 Output data 的前两个字节为 EIM2116 的 16 个 DO，且第一个字节为 DO1~DO8，如图 7.10 所示。

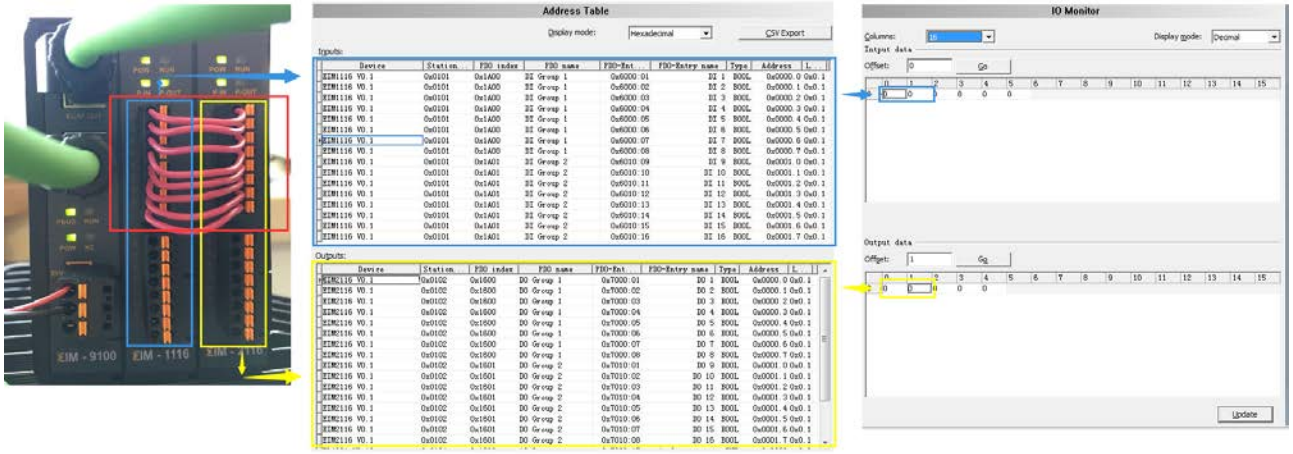


图 7.10，数据对应关系

明白了对应关系之后，在 IO Monitor 的 Output data 的前两个字节都输入 255，即 0xFF，再点 Update，

可以看到 Input 的第一个字节也更新为 255，说明数值 255 已经由板卡主站写给了 EIM2116，且 EIM1116 的值也已经传回了板卡，且 IO 模块对应的 IO 点也已经点亮，如图 7.11 所示。

图 7.11，点亮 IO 模块

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，如切换状态，CoE，PI update 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_2017-08-1_V1_400_170125_19044\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 04 EN》第 11 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，该例程可联系虹科获取。

该例程调用 API 的流程如图 8 所示。其中黄色部分表示 main 函数，绿色部分表示 main 函数内包含的几个调用块，蓝色部分为每个调用块内用到的 API 函数，灰色部分为对 API 函数实现的功能说明。

EnumBoardDemo 和 SysdeviceDemo 这两部分以及内部的 API 实现的只是一些基本调用，如找开设备，查看设备信息等，与协议和数据通信无关，使用者可以参考调用即可。而 ChannelDemo 部分刚实现了周期数据收发（PI 数据）和非周期数据收发（切换状态，CoE，FoE 等）的所有过程，是使用者应该重点考虑的。


Main
EnumBoardDemo
xDriverOpen
xDriverEnumBoards
xDriverEnumChannels
SysdeviceDemo
xSysdeviceOpen
xSysdeviceInfo
xSysdeviceGetMBXState
xSysdevicePutPacket
xSysdeviceGetMBXState
xSysdeviceGetPacket
xSysdeviceGetMBXState
xSysdeviceClose
ChannelDemo
xChannelOpen
xChannelInfo
xChannelHostState
xChannelBusState
Switch on the bus=Run
xChannelPutPacket
xChannelGetPacket
Switch to OP state
xChannelPutPacket
xChannelGetPacket
CoE wirte
xChannelPutPacket
xChannelGetPacket
CoE read
xChannelIORead
PI read,in cycle
xChannelIOWrite
PI write,in cycle
xChannelBusState
Switch off the bus
xChannelHostState
xChannelClose
xDriverGetErrorDescription
print error when happened
xDriverClose

图 8，API 调用流程

8.1 Main 函数代码部分

```
/*****************************************************************************/
/*! The main function
*   \return 0 on success                                                     */
/*****************************************************************************/
int main(int argc, char* argv[])
{
  HANDLE hDriver  = NULL;
  long lRet = CIFX_NO_ERROR;

  UNREFERENCED_PARAMETER(argc);
  UNREFERENCED_PARAMETER(argv);

  /* Open the cifX driver */
  lRet = xDriverOpen(&hDriver);

  if(CIFX_NO_ERROR != lRet)
  {
    printf("Error opening driver. lRet=0x%08X\r\n", lRet);
  } else
  {
    /* Example how to find a cifX/comX board */
    EnumBoardDemo(hDriver);

    /* Example how to communicate with the SYSTEM device of a board */
    SysdeviceDemo(hDriver, "cifX0");

    /* Example how to communicate with a communication channel on a board */
    ChannelDemo(hDriver, "cifX0", 0);

    /* Close the cifX driver */
    xDriverClose(hDriver);
  }

    return 0;
}
```

8.2 切换网络状态部分

Packet 定义，与 7.2.1 章节测试时定义的 Packet 一致。

```
/*  Definiton of Transfer Packet  for setting master state
/*Set state to OP
*   ulDest = 0x20
*   ulLen  = 0x01
*   ulCmd  = 0x9E00
*   abData = 0x01 for INIT
*   abData = 0x02 for PREOP
*   abData = 0x04 for SAFEOP
*   abData = 0x08 for OP           */
/*CIFX_PACKET                  {ulDest,   ulSrc,  ulDestId,   ulSrcId,    ulLen,  ulId,   ulSta,  ulCmd,  ulExt,  ulRout, abData          }
CIFX_PACKET        tSendState = {0x20,     0,      0,          0,          1,      0,      0,      0x9E00, 0,      0,      {0x08}          };
```

发送 Packet 实现 INIT 至 OP 状态的切换

```
/* Switch from INIT to OPERATION state*/
if(CIFX_NO_ERROR != (lRet = xChannelPutPacket(hChannel, &tSendState, PACKET_WAIT_TIMEOUT)))
  {
    printf("Error sending packet to device!\r\n");
  } else
  {
    printf("Swith state:Send Packet:\r\n");
    DumpPacket(&tSendState);

    if(CIFX_NO_ERROR != (lRet = xChannelGetPacket(hChannel, sizeof(tGetPkt), &tGetPkt, PACKET_WAIT_TIMEOUT)) )
    {
      printf("Error getting packet from device!\r\n");
    } else
    {
      printf("Swith state:Received Packet:\r\n");
      DumpPacket(&tGetPkt);
    }
  }
  Sleep(1000);
```

8.3 CoE write 与 CoE read 部分

Packet 定义，与 7.2.2 章节测试时定义的 Packet 一致。

```
/*  Definiton of Transfer Packet  for CoE Write,wirte data 0x01 to 0x1800:01 of slave with station address 0x0104
*   ulDest = 0x20
*   ulLen  = 0x13(=0x12+n,n=length of adData,in this case,n=1)
*   ulCmd  = 0x9A00
*   abData[ Station Address = 0x0104,
            TransportType   = 0x0000,
            AoEPort         = 0x0000,
            ObjIndex        = 0x8000,
            SubIndex        = 0x01,
            CompleteAccess  = 0x00,
            TotalBytes      = 0x00000001,
            TimeoutMs       = 0x000003E8,
            Data[n]         = 0x01]
        */
/*CIFX_PACKET                   {ulDest,   ulSrc,  ulDestId,   ulSrcId,    ulLen,  ulId,   ulSta,  ulCmd,  ulExt,  ulRout, abData         }                                */
CIFX_PACKET        tSendCoEw  = {0x20,     0,      0,          0,          0x13,   0,      0,      0x9A00, 0,      0,      \
/*                             Station Address,   TransportType,  AoEPort,        ObjIndex,       SubIndex,   CompleteAccess, TotalBytes,                    TimeoutMs,                      Data[n]*/
                               {{0x04},{0x01},    {0x00},{0x00},  {0x00},{0x00},  {0x00},{0x80},  {0x01},     {0x00},         {0x01},{0x00},{0x00},{0x00},    {0xE8},{0x03},{0x00},{0x00},    {0x01}}         };

/*  Definiton of Transfer Packet  for CoE Read,read value of 0x1800:01 of slave with station address 0x0104
*   ulDest = 0x20
*   ulLen  = 0x12
*   ulCmd  = 0x9A02
*   abData[ Station Address = 0x0104,
            TransportType   = 0x0000,
            AoEPort         = 0x0000,
            ObjIndex        = 0x8000,
            SubIndex        = 0x01,
            CompleteAccess  = 0x00,
            TimeoutMs       = 0x000003E8,
            MaxTotalBytes   = 0x00000001]
        */
/*CIFX_PACKET                   {ulDest,   ulSrc,  ulDestId,   ulSrcId,    ulLen,  ulId,   ulSta,  ulCmd,  ulExt,  ulRout, abData         }                                */
CIFX_PACKET        tSendCoEr  = {0x20,     0,      0,          0,          0x12,   0,      0,      0x9A02, 0,      0,      \
/*                             Station Address,   TransportType,  AoEPort,        ObjIndex,       SubIndex,   CompleteAccess, TimeoutMs,                     MaxTotalBytes*/
                               {{0x04},{0x01},    {0x00},{0x00},  {0x00},{0x00},  {0x00},{0x80},  {0x01},     {0x00},         {0xE8},{0x03},{0x00},{0x00},    {0x01},{0x00},{0x00},{0x00},}           };
```

发送 Packet 实现 CoE write 与 CoE read

```
/* CoE write*/
if(CIFX_NO_ERROR != (lRet = xChannelPutPacket(hChannel, &tSendCoEw, PACKET_WAIT_TIMEOUT)))
{
  printf("Error sending packet to device!\r\n");
} else
{
  printf("CoE write:Send Packet:\r\n");
  DumpPacket(&tSendCoEw);

  if(CIFX_NO_ERROR != (lRet = xChannelGetPacket(hChannel, sizeof(tGetPkt), &tGetPkt, PACKET_WAIT_TIMEOUT)) )
  {
    printf("Error getting packet from device!\r\n");
  } else
  {
    printf("CoE write:Received Packet:\r\n");
    DumpPacket(&tGetPkt);
  }
}
Sleep(1000);

/* CoE read*/
if(CIFX_NO_ERROR != (lRet = xChannelPutPacket(hChannel, &tSendCoEr, PACKET_WAIT_TIMEOUT)))
{
  printf("Error sending packet to device!\r\n");
} else
{
  printf("CoE read:Send Packet:\r\n");
  DumpPacket(&tSendCoEr);

  if(CIFX_NO_ERROR != (lRet = xChannelGetPacket(hChannel, sizeof(tGetPkt), &tGetPkt, PACKET_WAIT_TIMEOUT)) )
  {
    printf("Error getting packet from device!\r\n");
  } else
  {
    printf("CoE read:Received Packet:\r\n");
    DumpPacket(&tGetPkt);
  }
}
Sleep(10000);
```

8.4 PI 周期数据更新部分

定义

```
/* Read and write I/O data (32Bytes). Output data will be incremented each cyle */
unsigned char       abSendPIData[2] = {0};
unsigned char       abRecvPIData[1] = {0};
unsigned long       ulCycle         = 0;
uint32_t            ulState         = 0;
```

通过 xChannelIORead()和 xChannelIOWrite()函数实现周期数据更新，可以看到，发送的数据 abSendPIData 与 7.2.3 章节测试时保持一致。

```
/* Do I/O Data exchange until a key is hit */
while(!kbhit())
{
  if(CIFX_NO_ERROR != (lRet = xChannelIORead(hChannel, 0, 0, sizeof(abRecvPIData), abRecvPIData, IO_WAIT_TIMEOUT)))
  {
    printf("Error reading IO Data area!\r\n");
    break;
  } else
  {
    printf("IORead Data:");
    DumpData(abRecvPIData, sizeof(abRecvPIData));

    abSendPIData[0] = 0xFF;
    abSendPIData[1] = 0xFF;

    if(CIFX_NO_ERROR != (lRet = xChannelIOWrite(hChannel, 0, 0, sizeof(abSendPIData), abSendPIData, IO_WAIT_TIMEOUT)))
    {
      printf("Error writing to IO Data area!\r\n");
      break;
    } else
    {
      printf("IOWrite Data:");
      DumpData(abSendPIData, sizeof(abSendPIData));

      memset(abSendPIData, ulCycle + 1, sizeof(abSendPIData));
    }
  }
```

8.5 Debug 结果

可以看到，Debug 的结果与 7.2 章节的测试结果是一致的。

```
---------- Board/Channel enumeration demo ----------
Found Board cifX0
 DeviceNumber : 1250100
 SerialNumber : 37881
 Board ID     : 0
 System Error : 0x00000000
 Channels     : 2
 DPM Size     : 65536
 - Channel 0:
    Firmware : EtherCAT Master
    Version  : 4.4.0 build 0
    Date     : 03/01/2017
 - Channel 1:
    Firmware : Ethernet Interface
    Version  : 4.2.6 build 0
    Date     : 04/07/2014
-------------------------------------------------
---------- System Device handling demo ----------
System Channel Info Block:
DPM Size          : 32000
Device Number     : 1250100
Serial Number     : 37881
Manufacturer      : 1
Production Date   : 4610
Device Class      : 3
HW Revision       : 6
HW Compatibility : 0
System Mailbox State: MaxSend = 64, Pending Receive = 0
Send Packet:
Dest    : 0x00000000      ID    : 0x00000000
Src     : 0x00000000      Sta   : 0x00000000
DestID : 0x00000000       Cmd   : 0x00000000
SrcID   : 0x00000000      Ext   : 0x00000000
Len     : 0x00000000      Rout : 0x00000000
Data:
System Mailbox State: MaxSend = 64, Pending Receive = 1
Received Packet:
Dest    : 0x00000000      ID    : 0x00000000
Src     : 0x00000000      Sta   : 0xC0000004
DestID : 0x00000000       Cmd   : 0x00000001
SrcID   : 0x00000000      Ext   : 0x00000000
Len     : 0x00000000      Rout : 0x00000000
Data:
System Mailbox State: MaxSend = 64, Pending Receive = 0
 State = 0x00000000
```

```
---------- Communication Channel demo ----------
Communication Channel Info:
Device Number     : 1250100
Serial Number     : 37881
Firmware          : EtherCAT Master
FW Version        : 4.4.0 build 0
FW Date           : 03/01/2017
Mailbox Size      : 1596
Swith state:Send Packet:
Dest    : 0x00000020      ID    : 0x00000000
Src     : 0x00000000      Sta   : 0x00000000
DestID : 0x00000000       Cmd   : 0x00009E00
SrcID   : 0x00000000      Ext   : 0x00000000
Len     : 0x00000001      Rout : 0x00000000
Data:
08
Swith state:Received Packet:
Dest    : 0x00000020      ID    : 0x00000000
Src     : 0x00000000      Sta   : 0x00000000
DestID : 0x00D60000       Cmd   : 0x00009E01
SrcID   : 0x00000000      Ext   : 0x00000000
Len     : 0x00000001      Rout : 0x00000000
Data:
08
```

```
CoE write:Send Packet:
Dest    : 0x00000020      ID   : 0x00000000
Src     : 0x00000000      Sta  : 0x00000000
DestID  : 0x00000000      Cmd  : 0x00009A00
SrcID   : 0x00000000      Ext  : 0x00000000
Len     : 0x00000013      Rout : 0x00000000
Data:
04 01 00 00 00 00 00 80 01 00 01 00 00 00 E8 03
00 00 01
CoE write:Received Packet:
Dest    : 0x00000020      ID   : 0x00000000
Src     : 0x00000000      Sta  : 0x00000000
DestID  : 0x00000000      Cmd  : 0x00009A01
SrcID   : 0x00000000      Ext  : 0x00000000
Len     : 0x00000012      Rout : 0x00000000
Data:
04 01 00 00 00 00 00 80 01 00 01 00 00 00 E8 03
00 00
CoE read:Send Packet:
Dest    : 0x00000020      ID   : 0x00000000
Src     : 0x00000000      Sta  : 0x00000000
DestID  : 0x00000000      Cmd  : 0x00009A02
SrcID   : 0x00000000      Ext  : 0x00000000
Len     : 0x00000012      Rout : 0x00000000
Data:
04 01 00 00 00 00 00 80 01 00 E8 03 00 00 01 00
00 00
CoE read:Received Packet:
Dest    : 0x00000020      ID   : 0x00000000
Src     : 0x00000000      Sta  : 0x00000000
DestID  : 0x00000000      Cmd  : 0x00009A03
SrcID   : 0x00000000      Ext  : 0x00000000
Len     : 0x00000013      Rout : 0x00000000
Data:
04 01 00 00 00 00 00 80 01 00 E8 03 00 00 01 00
00 00 01
```

```
IORead Data:
FF
IOWrite Data:
FF FF
IORead Data:
FF
IOWrite Data:
FF FF
IORead Data:
```

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。