User Manual

EtherCAT 从站参考手册

AWorksLP

V0.0.0 Date:2025/01/13

类别	内容
关键词	DPort-ECT，EtherCAT，AWorks
摘要	EtherCAT 从站参考手册

修订历史

版本	日期	原因

目 录

1. 名词/缩写解释

SSC EtherCAT Slave Stack Code Tool

ESI EtherCAT Slave Information

PDI Process Data Interface

ESC EtherCAT Slave Controller

DC Distributed Clock

2. SSC 的配置及其使用

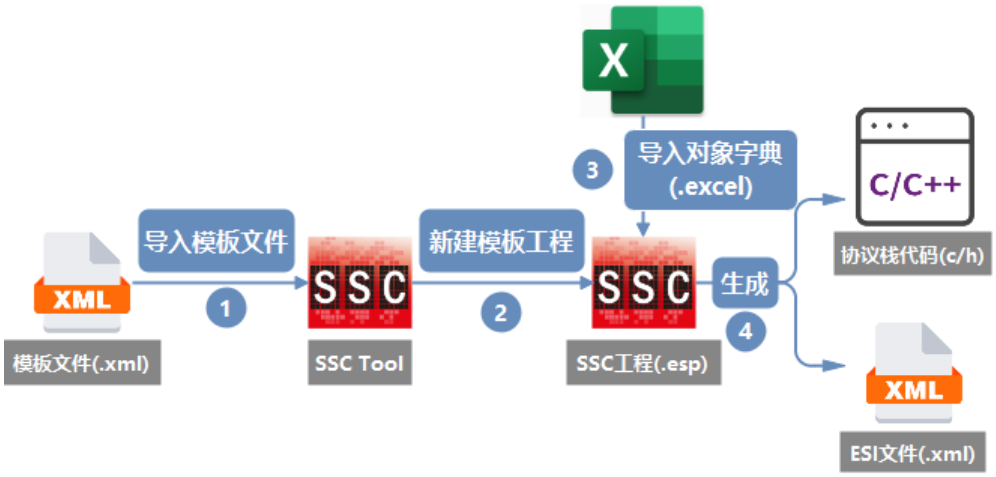


图 2.1　SSC 开发流程框图

注意

- 此文档以 EPC103-DP EtherCAT 评估板为例;
- DPort-ECT 相关例程均使用 **SSC_V5i13** 版本, 注意核对;
- 倍福官方 SSC 工具下载链接[1] 。

[1] https://www.beckhoff.com.cn/zh-cn/support/download-finder/search-result/?search=slave%20stack

2.1 SSC 选项栏使用方法

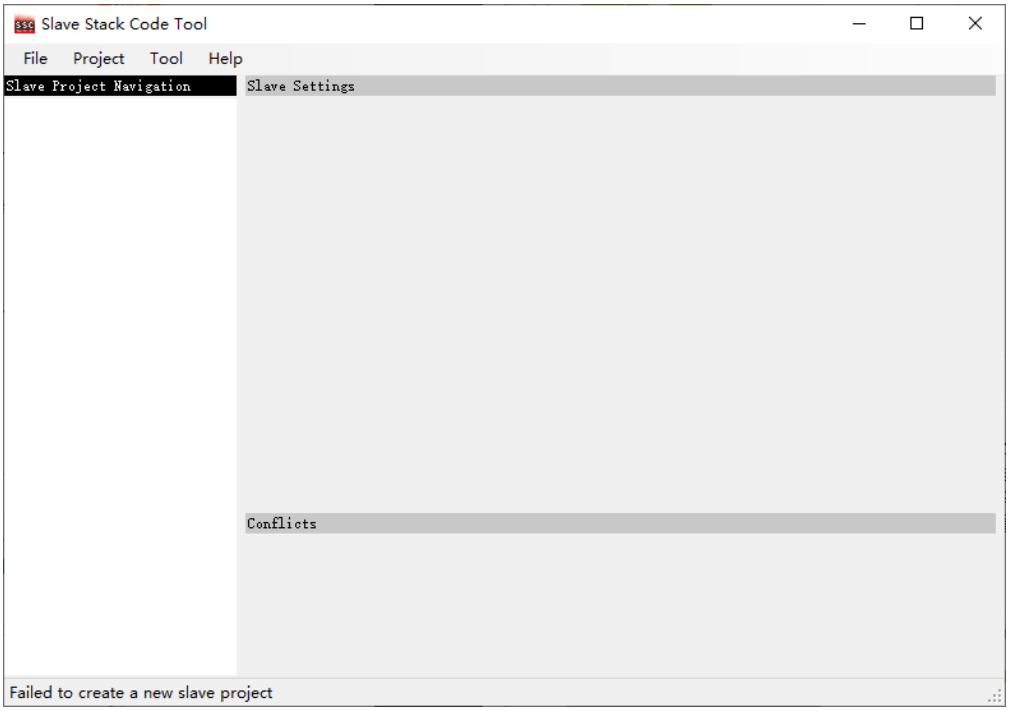


图 2.2 SSC 主界面

2.1.1 File

- New：新建工程；
- Open：打开已有的工程；
- Save：保存工程配置；
- Save As：将工程另存为；
- Exit：退出。

2.1.2 Project

- Find Setting：查找配置项；

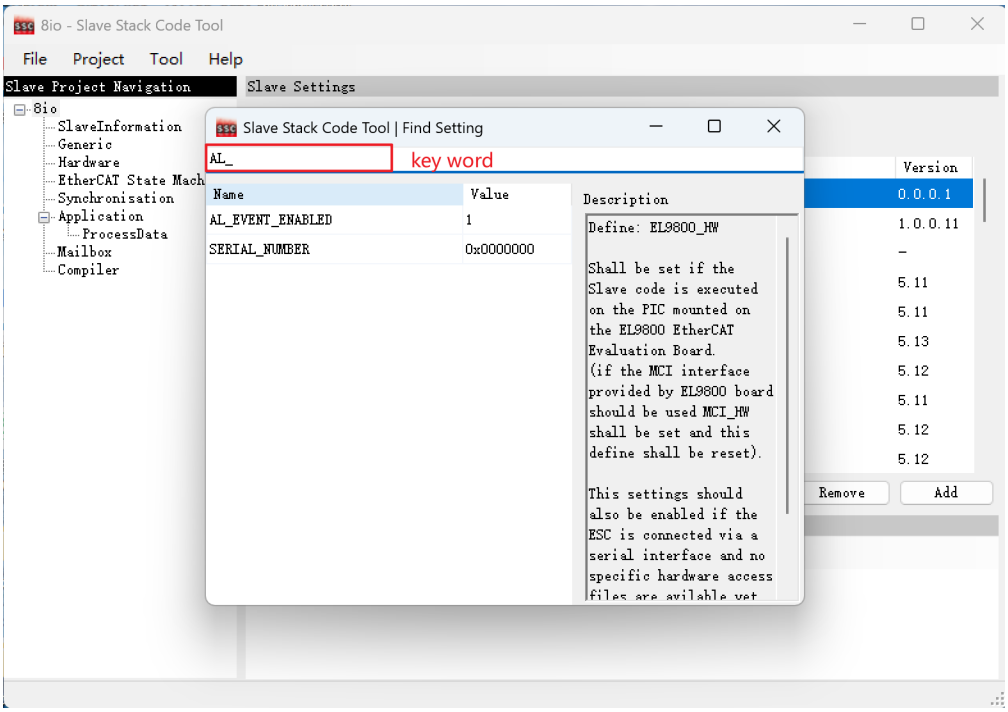


图 2.3　Find Setting

- Create new Slave Files：生成源码和 ESI 文件等

2.1.3　Tool

- Show Confict Window：显示信息提示窗口

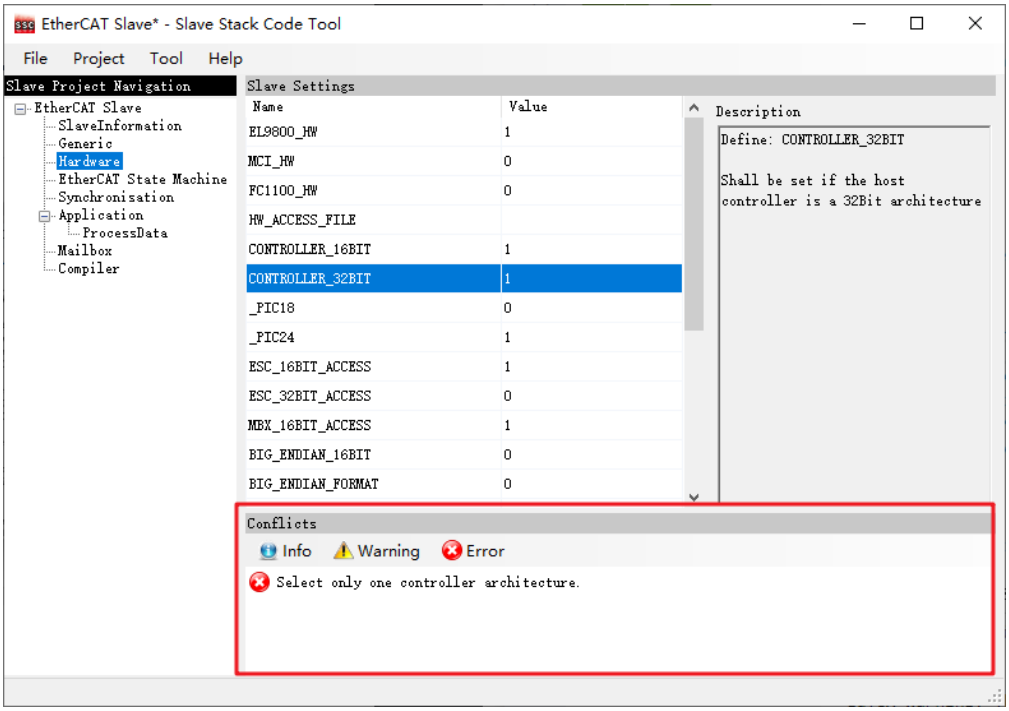


图 2.4　Show Confict Window

- Options：

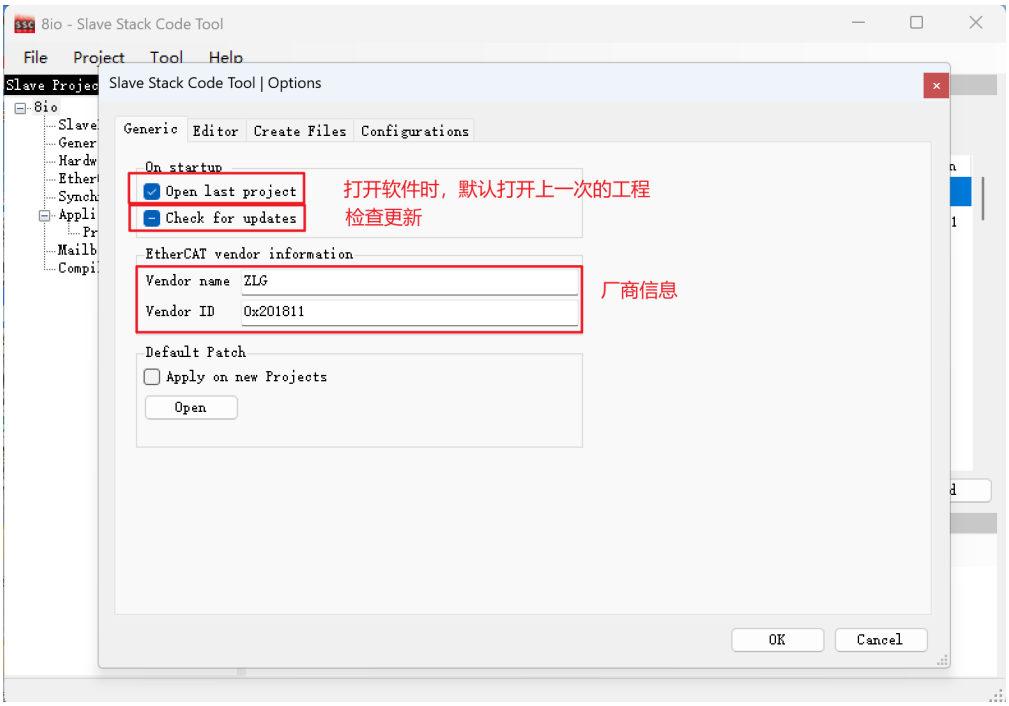


图 2.5　Generic

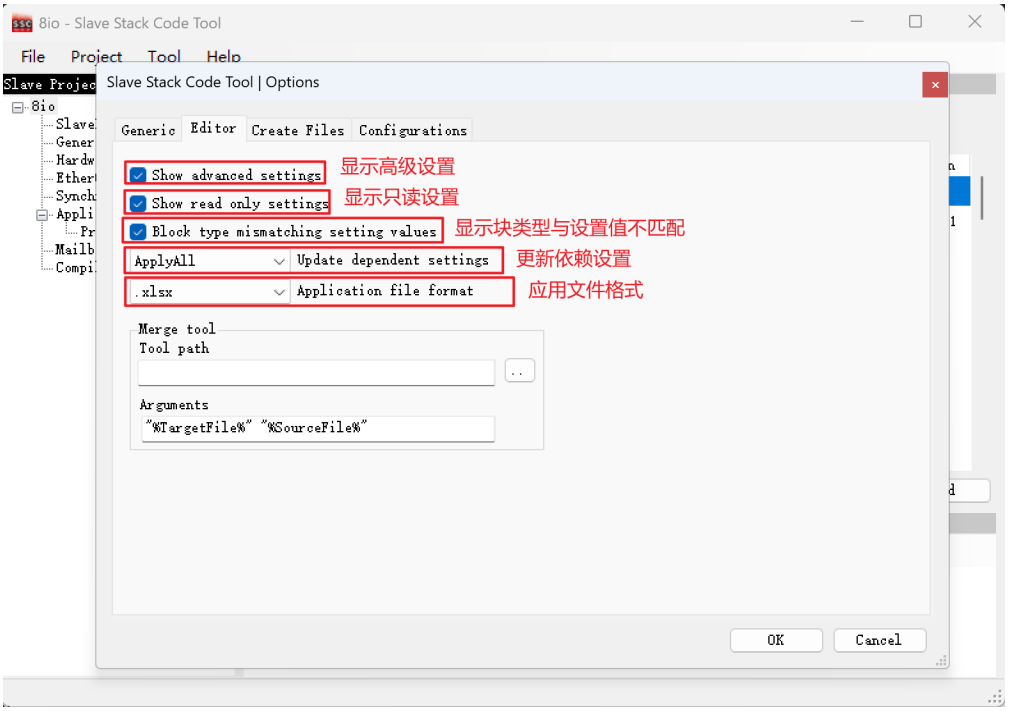


图 2.6　Editor

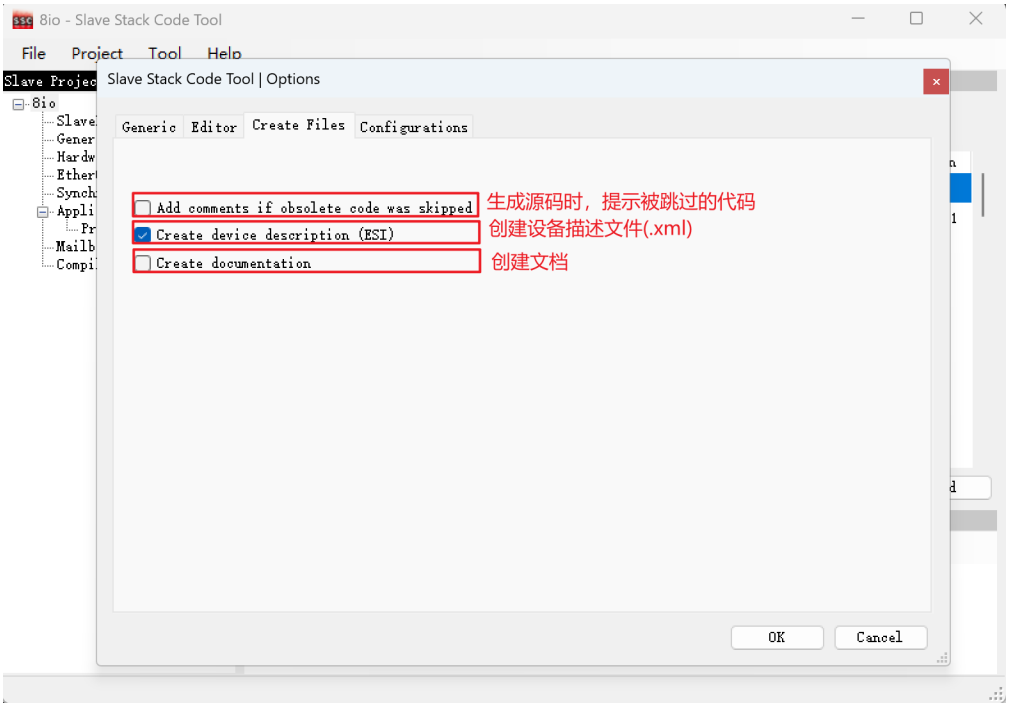


图 2.7　Create Files

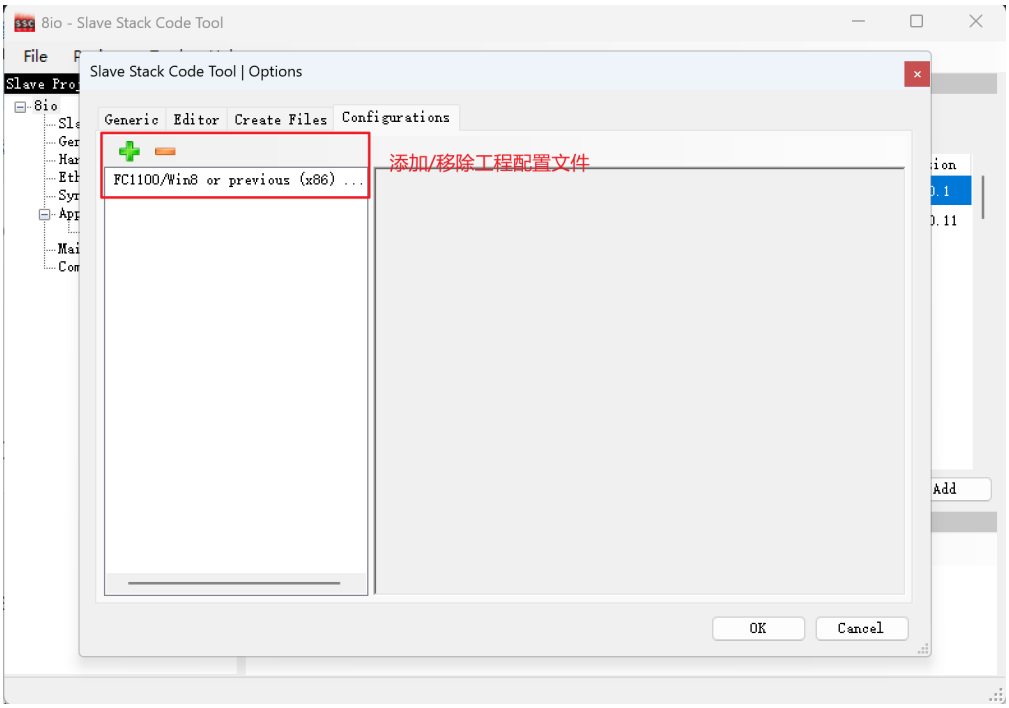


图 2.8　Configurations

- EEPROM Programmer：EEPROM ESI 烧录

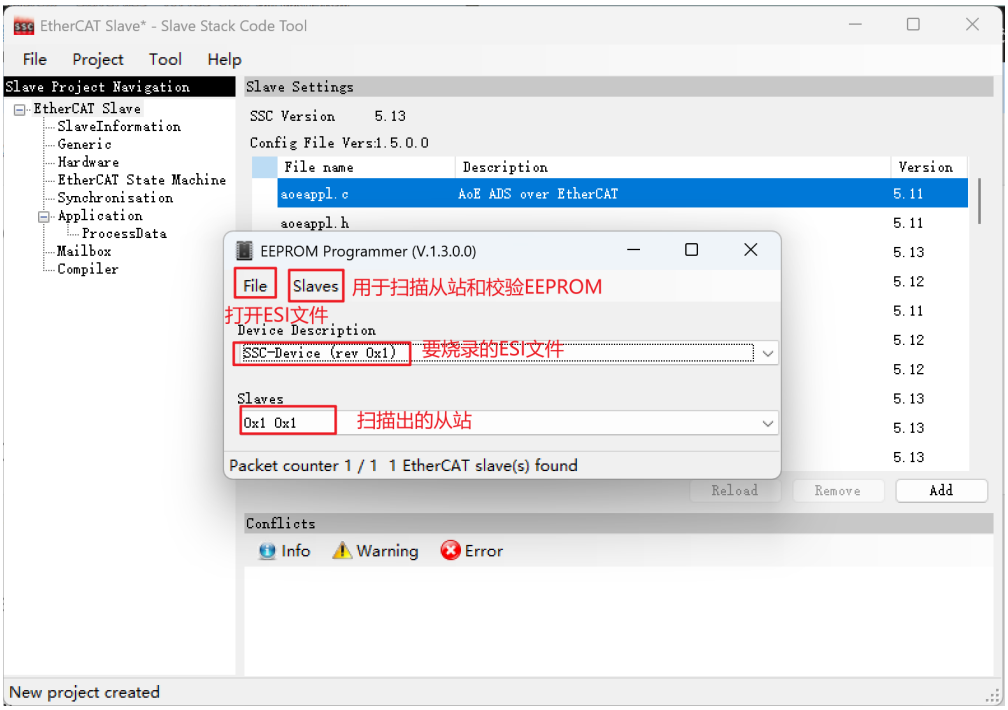


图 2.9 EEPROM Programmer

- Application：导入或新建对象字典文件 (之后章节会详细介绍)

2.1.4 help

- About：SSC 版本相关信息

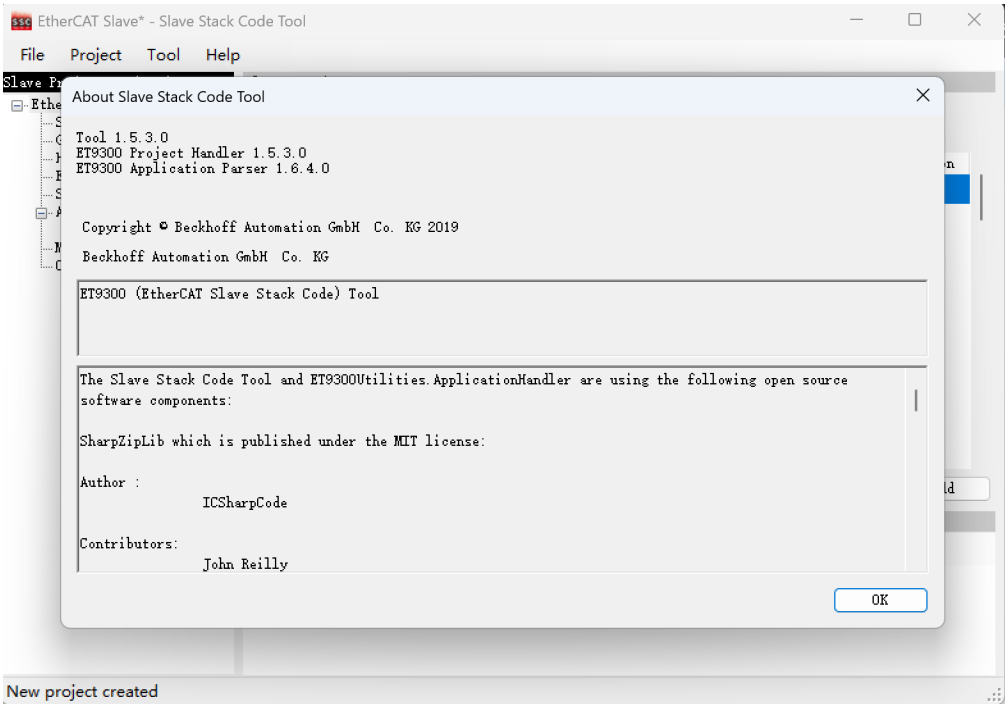


图 2.10 About

- Contact：倍福官方联系方式
- Documentation：SSC 相关文档
- Check for Updates：检查更新

2.2 导入模板文件

2.2.1 模板文件 (.xml)

点击 **File -> New** 新建工程，会出现如下界面：

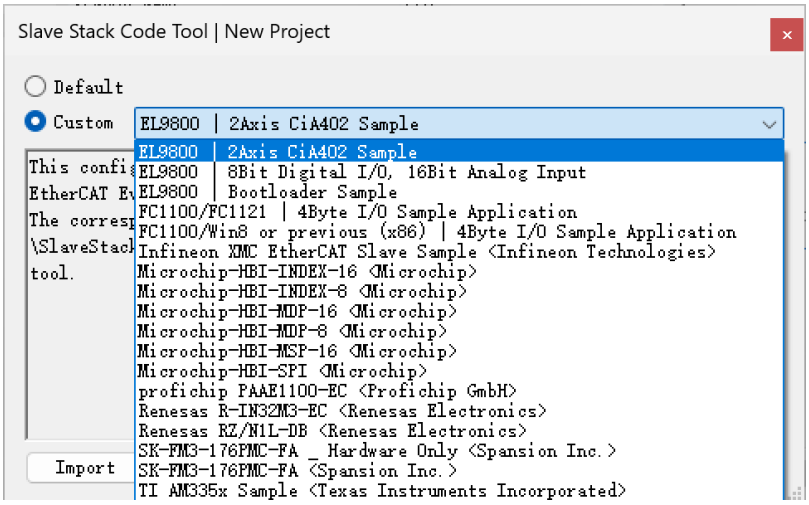


图 2.11 SSC 默认工程模板

如图所示，下拉菜单中的各种选项是 ESC 厂商根据自身硬件平台所提供的模板例程，如果所选模板平台与自身需求吻合，便可进行复用，

我们也向用户提供了基于 EPC103-DP/EPC6450-DP 平台的模板文件，以便参考。

打开 EPC103-DP 评估板中 **{例程}/ EtherCAT-Slave-Adapter/ DPort-ECT_SSC_Config.xml** 文件，内容如图下所示。

程序清单 2.1 DPort-ECT_SSC_Config.xml 中的部分内容

```
.....................
<!--EtherCAT State Machine-->
<Setting CodeDefine="BOOTSTRAPMODE_SUPPORTED" Value="0"/>
<Setting CodeDefine="OP_PD_REQUIRED" Value="1"/>
<Setting CodeDefine="PREOPTIMEOUT" Value="0x7D0"/>
<Setting CodeDefine="SAFEOP2OPTIMEOUT" Value="0x2328"/>
<Setting CodeDefine="CHECK_SM_PARAM_ALIGNMENT" Value="0"/>
<!--Synchronisation-->
.....................
```

对比 SSC 中的配置项：

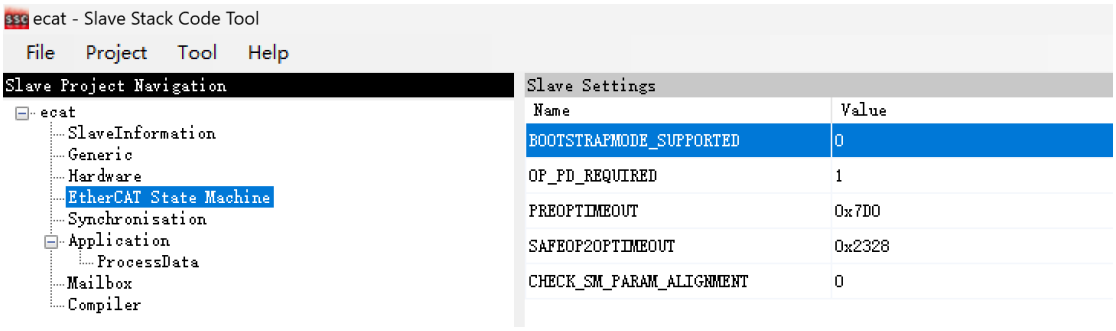


图 2.12　SSC 配置界面对比

可以发现，**DPort-ECT_SSC_Config.xml** 中的内容对应着 SSC 中的配置项，当新建工程时，SSC 的配置项会按照所选的工程模板中的键值进行设置。

注意

DPort-ECT_SSC_Config.xml 文件必须与 EtherCAT-Slave-Adapter 驱动组件配套使用。

2.2.2　导入步骤

- 1. 打开 SSC 软件，点击 **Tool -> Option -> Configurations**，出现如下界面：

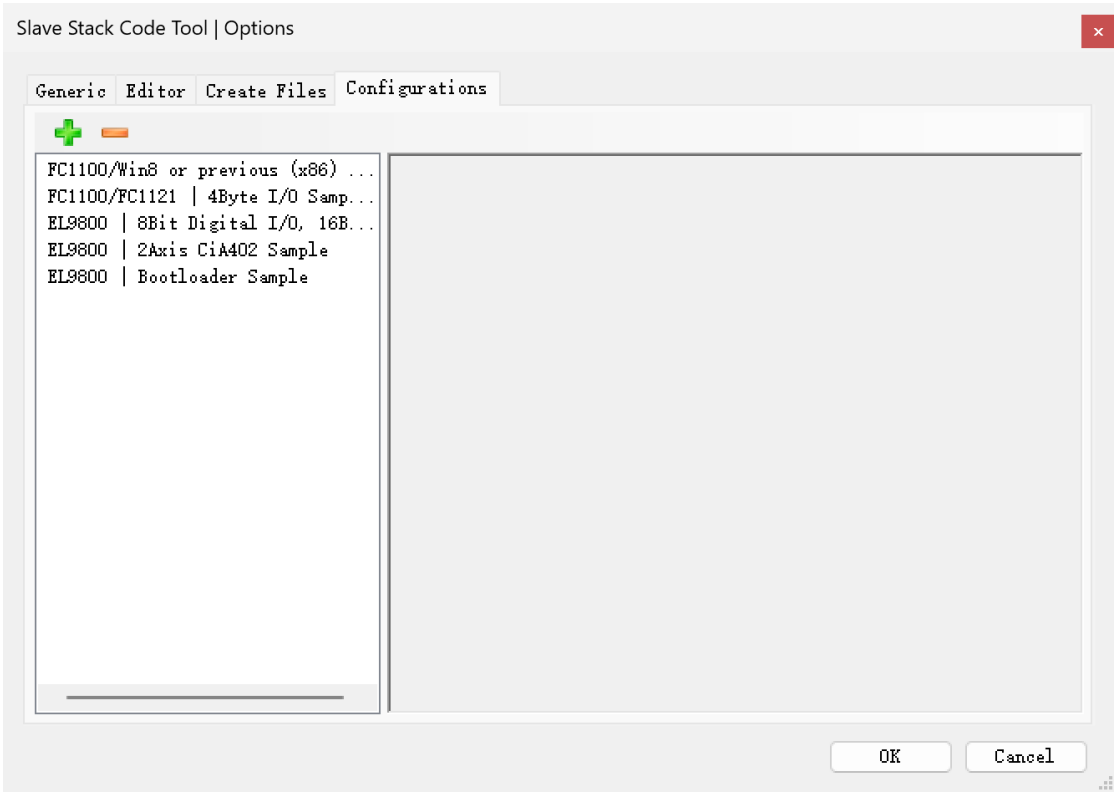


图 2.13　工程模板导入界面

- 2. 点击符号 +，选择例程下的模板文件，以 EPC103-DP 中 8io 为例，导入 DPort-ECT_SSC_Config.xml 文件。

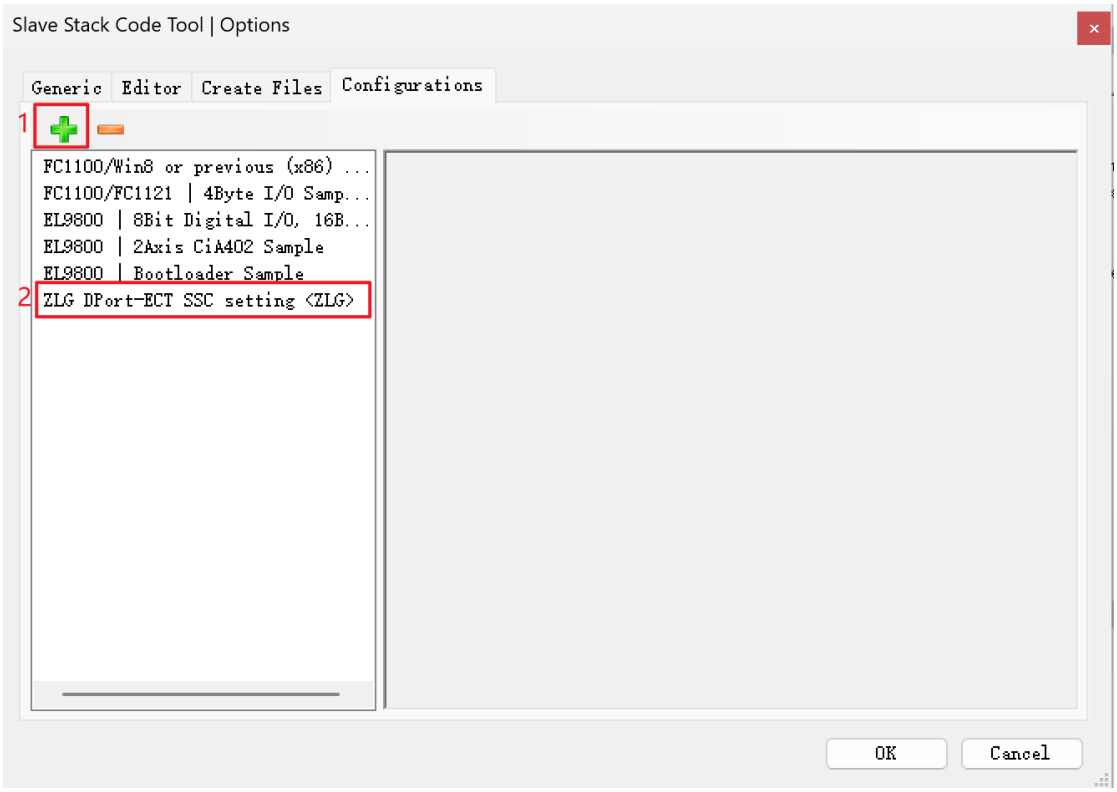


图 2.14　工程模板成功添加界面

- 3. 新建工程查看是否存在添加的模板，**File->New** 。

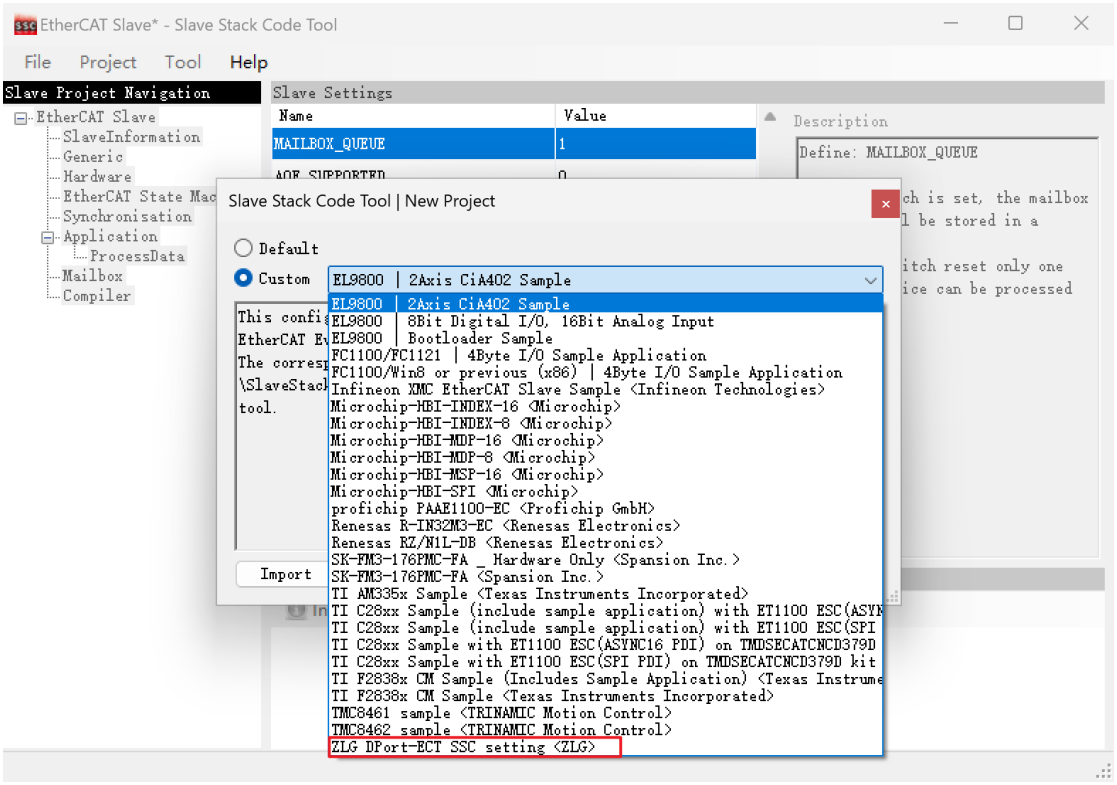


图 2.15　新建测试模板工程

2.3 SSC 配置项

2.3.1 SlaveInformation

SSC 配置项	含义	注解
VENDOR_ID	厂商 id	由 EtherCAT 官方进行分配
VENDOR_NAME	厂商名称	
VENDOR_IMAGE	厂商图标	bmp 格式 (16x14)px
GROUP_NAME	组别名称	
GROUP_IMAGE	组别图标	bmp 格式 (16x14)px
DEVICE_IMAGE	设备图标	bmp 格式 (16x14)px
PRODUCT_CODE	产品码	
REVISION_NUMBER	修订号	
SERIAL_NUMBER	序列号	
DEVICE_PROFILE_TYPE	设备描述类型	
DEVICE_NAME	设备名称	
DEVICE_HW_VERSION	设备硬件版本	
DEVICE_SW_VERSION	设备软件版本	

注意

此为设备版本相关的描述信息，需要由用户自定义，有的主站会对厂商信息产品码等有硬性要求，在 init 阶段会进行检查，若检查失败则可能会导致状态转换的失败。

2.3.2 generic

表 2.2 SYSTEM_HEADER_FILE

SSC 配置项	含义
SYSTEM_HEADER_FILE	系统头文件

注：通常格式为 *#include “xxx.h”*，头文件将会包含在 *ecat_def.h* 文件中，协议栈的所有代码皆可包含到 *xxx.h*。若使用 *EtherCAT-Slave-Adapter* 组件中的 *DPort-ECT_SSC_Config.xml* 模板文件时，此配置项默认为 *#include “aw_ecat_ssc.h”*。

表 2.3 EXPLICIT_DEVICE_ID

SSC 配置项	含义
EXPLICIT_DEVICE_ID	显示从站 ID

注：*EXPLICIT_DEVICE_ID* 用于设置寻址，在链路初始化阶段，从站协议站会调用 *UINT16 APPL_GetDeviceID(void)* 函数 (需用户实现) 获取用户设置的设备 ID，并将其作为此从站寻址依据，这就意味着在设置寻址下，从站的地址与从站的物理连接位置无关。

注意

使用此功能时需要主站设置相应的热拔插 (*«TwinCAT* 操作及使用 » 章节介绍)。

表 2.4 ESC_SM_WD_SUPPORTED

SSC 配置项	含义
ESC_SM_WD_SUPPORTED	同步管理器看门狗设置

注: 用于检测是否在规定时间内接收到周期数据，与之关联的错误码为 *ALSTATUSCODE_SMWATCHDOG*。

注意

在进入 op 阶段时，若出现错误码为 *ALSTATUSCODE_SMWATCHDOG* 而导致的状态转换失败，其原因在于规定时间内未接受到周期数据，此时需要检测一下 PDI 中断的配置情况。

表 2.5 STATIC_OBJECT_DIC

SSC 配置项	含义
STATIC_OBJECT_DIC	对象存储方式

注: 此宏只会在 *EL9800_APPLICATION* 中使用，无实际配置作用。

表 2.6 ESC_EEPROM_ACCESS_SUPPOT

SSC 配置项	含义
ESC_EEPROM_ACCESS_SUPPOT	是否支持访问 EEPROM

注: ESC 若支持访问 EEPROM，则会提供访问接口：

- *UINT16 ESC_EepromAccess(UINT32 wordaddress, UINT16 wordsize, UINT16 MBXMEM *pData, UINT8 access)*;
- *UINT32 ESC_EepromWriteCRC(void)*.

注意

EEPROM 读写规则

主站可以通过操作 *ESC EEPROM* 控制器访问内部的数据，而从站操作 *EEPROM* 则需要主站赋予操作权限 (由寄存器 0x500 体现，对于从站是只读的)，在以下情况 EtherCAT 主站会通过写 0x500 将访问权限交给 *PDI*：

- 1. 在 *init -> preop* 转换时;
- 2. 在 *init -> boot* 转换时，包括处于 *boot* 状态下;
- 3. 在 ESI 文件中定义 *AssignToPdi* 元素，除 init 外，主站会将权限交付给 *PDI*。

2.3.3 Hardware

SSC 配置项	含义
EL9800_HW	倍福官方开发板驱动
MCI_HW	
FC1100_HW	

SSC 配置项	含义
HW_ACCESS_FILE	硬件接口文件

注: 格式为 *#include “xxx.h”* ，指定协议栈硬件接口文件 (需要由用户提供)，若使用 *EtherCAT-Slave-*

Adapter 组件中的 *DPort-ECT_SSC_Config.xml* 模板文件时，此配置项默认为 *#include “aw_ecat_ssc.h”*，此文件会提供例如下的 IO 读写等函数：

- *HW_EscRead(pData, Address, Len);*
- *HW_EscWrite(pData, Address, Len);*
- ……

SSC 配置项	含义
CONTROLLER_16BIT	指定微控制器架构:16Bit or 32Bit
CONTROLLER_32BIT	

SSC 配置项	含义
MEMORY_UNIT_16BIT	内存单元最小是否为 16bit

注：常见的 32 位芯片最小寻址单元为 1Byte(8Bit)，但是在一些 DSP 芯片中，内部最小寻址单元就是 16bit。

SSC 配置项	含义
_PIC18	倍福基于 PIC 芯片做的示例
_PIC24	

SSC 配置项	含义
ESC_16BIT_ACCESS	ESC 内存访问对齐限制:16Bit or 32Bit
ESC_32BIT_ACCESS	

注意

大部分 ESC 芯片对 EtherCAT 内核部分寄存器的访问并没有限制，但是部分集成了 EtherCAT 外设的 MCU(On-Chip-Bus) 对 RAM 区域的访问必须是 4 字节对齐，协议栈在配置时也需要指定此处为四字节对齐。

SSC 配置项	含义
MBX_16BIT_ACCESS	微控制器邮箱区域内存 16bit 对齐访问限制
BIG_ENDIAN_16BIT	大端 16bit 模式
BIG_ENDIAN_FORMAT	大端模式
EXT_DEBUGER_INTERFACE	协议栈中 _PIC24 例程所使用的宏, 启用 debug 信息输出

SSC 配置项	含义
UC_SET_ECAT_LED	从站状态指示灯 (微控制器控制)

注：用户需实现 *void HW_SetLed(UINT8 RunLed, UINT8 Errled)* 函数，由协议栈调用，由微控制器需控制两个 led 的亮灭：

- RunLed == 1：run 灯亮;
- Errled == 1：err 灯亮;
- RunLed == 0：run 灯灭;
- Errled == 0：err 灯灭。

在 EPC103-DP 例程中实现为：

```
void __us_set_stack_led(uint8_t run_led, uint8_t err_led)
{
    run_led ? LL_GPIO_ResetOutputPin(RUN_GPIO_Port, RUN_Pin):
            LL_GPIO_SetOutputPin(RUN_GPIO_Port, RUN_Pin);
    err_led ? LL_GPIO_ResetOutputPin(ERR_GPIO_Port, ERR_Pin):
            LL_GPIO_SetOutputPin(ERR_GPIO_Port, ERR_Pin);
}
```

2.3.4 EtherCAT State Machine

SSC 配置项	含义
BOOTSTRAPMODE_SUPPORTED	boot 状态支持
OP_PD_REQUIRED	转换到 op 状态时，指定是否需要接收到周期数据
PREOPTIMEOUT	Init -> preop 状态转化超时时间
SAFEOP2OPTIMEOUT	safeop -> op 状态转换超时时间
CHECK_SM_PARAM_ALIGNMENT	检查 SM 的相关参数是否按要求对齐

2.3.5 Synchronisation

SSC 配置项	含义
AL_EVENT_ENABLED	PDI 中断使能
DC_SUPPORTED	ESC 是否支持分布式时钟

注：*AL_EVENT_ENABLED* 与 *DC_SUPPORTED* 不同组合会形成不同的通信模式：

	AL_EVENT_ENABLED	DC_SUPPORTED
freerun	0	0
SM-Sync	1	0
DC-Sync	0	1
SM-DC-Sync	1	1

SSC 配置项	含义
ECAT_TIMER_INT	是否使用定时器中断

注：*ECAT_TIMER_INT* 置一后需要为协议栈提供一个 1ms 的定时器中断，在中断回调内调用协议栈函数：*void ECAT_CheckTimer(void)*，STM32 代码示例如下：

```
void TIM2_IRQHandler(void)
{
    if(LL_TIM_IsActiveFlag_UPDATE(TIM2) == SET)
    {
        LL_TIM_ClearFlag_UPDATE(TIM2);
#if ECAT_TIMER_INT
        ECAT_CheckTimer();
        __g_ecat_timer_inc++;
#endif
```

```
    }
}
```

若不使用定时器中断则需要向协议栈提供以下函数和宏：

- fn *void HW_ClearTimer(void)*：清除当前的时间计数值
- fn *UINT16 HW_GetTimer(void)*：获取当前的时间计数值
- macro *ECAT_TIMER_INC_P_MS*：每毫秒有多少计数值

SSC 配置项	含义
MIN_PD_CYCLE_TIME	周期数据最小的循环时间
MAX_PD_CYCLE_TIME	周期数据最大的循环时间
PD_OUTPUT_DELAY_TIME	周期数据最小输出延时时间
PD_OUTPUT_CALC_AND_COPY_TIME	周期 OUTPUT 数据最小计算拷贝时间
PD_INPUT_CALC_AND_COPY_TIME	周期 INPUT 数据最小计算拷贝时间
PD_INPUT_DELAY_TIME	周期数据最大输出延时时间

2.3.6 Application

SSC 配置项	含义
TEST_APPLICATION	测试例程
TEST_APPLICATION_REDUCED_MEMORY	
EL9800_APPLICATION	EL9800 设备例程
CiA402_SAMPLE_APPLICATION	cia402 电机例程
SAMPLE_APPLICATION	独立于硬件的示例例程
SAMPLE_APPLICATION_INTERFACE	
BOOTLOADER_SAMPLE	boot loader 示例

SSC 配置项	含义
APPLICATION_FILE	应用程序头文件

注: 通常格式为 *#include “xxx.h”* 。在导入时 *Excel* 文件后，此配置项的值会自动设置为含 *Excel* 的名称的头文件，例如在例程 *8io* 当中，当导入 *8io.xlsx* 后，此配置项就会变为 *#include “8io.h”* 。

reason：

在导入 *Excel* 文件后，SSC 会在源码中生成三个带有 *Excel* 名称的应用相关的文件。以 *8io* 例程为例，当导入 *8io.xlsx* 后，Src 中会生成一系列的源码文件，其中除 *8io.h*、*8io.c*、*8ioObjects.h* 外，其余源码皆是与协议栈相关的 (内容是固定的)：

- *8io.c/8io.h*： 用户应用文件，用户需实现几个接口函数 (见 << 应用程序与接口函数 >>)，用于周期数据处理；
- *8ioObjects.h*： *Excel* 中编写的数据对象会在此文件以 c 语言的方式体现出来。

SSC 配置项	含义
USE_DEFAULT_MAIN	协议栈提供 main 函数

注: 协议栈会提供一个默认的 main 函数，方便用户调用，大致结构如下所示。

```
void main(void)
{
    /* initialize the Hardware and the EtherCAT Slave Controller */
    HW_Init();
    MainInit();
    bRunApplication = TRUE;
    do
    {
        MainLoop();
    } while (bRunApplication == TRUE);
    HW_Release();
}
```

- ProcessData

SSC 配置项	含义
MIN_PD_WRITE_ADDRESS	OUTPUT 周期数据在 DPROM 中映射的最小起始地址
DEF_PD_WRITE_ADDRESS	OUTPUT 周期数据在 DPROM 中映射的默认地址
MAX_PD_WRITE_ADDRESS	OUTPUT 周期数据在 DPROM 中映射的最大终止地址
MIN_PD_READ_ADDRESS	INPUT 周期数据在 DPROM 中映射的最小起始地址
DEF_PD_READ_ADDRESS	INPUT 周期数据在 DPROM 中映射的默认地址
MAX_PD_READ_ADDRESS	INPUT 周期数据在 DPROM 中映射的最大终止地址
MAX_PD_INPUT_SIZE	INPUT 周期数据最大 Size
MAX_PD_OUTPUT_SIZE	OUTPUT 周期数据最大 Size

2.3.7 Maibox

SSC 配置项	含义
MAILBOX_QUEUE	使能邮箱队列
AOE_SUPPORTED	AOE 协议使能
COE_SUPPORTED	COE 协议使能
COMPLETE_ACCESS_SUPPORTED	完全访问
SEGMENTED_SDO_SUPPORTED	sdo 分段是否支持
SDO_RES_INTERFACE	
USE_SINGLE_PDO_MAPPING_ENTRY_DESCR	
BACKUP_PARAMETER_SUPPORTED	
STORE_BACKUP_PARAMETER_IMMEDIATELY	
DIAGNOSIS_SUPPORTED	
MAX_DIAG_MSG	
EMERGENCY_SUPPORTED	
MAX_EMERGENCIES	
VOE_SUPPORTED	VOE 协议使能
SOE_SUPPORTED	SOE 协议使能

续上表

SSC 配置项	含义
EOE_SUPPORTED	EOE 协议使能
STATIC_ETHERNET_BUFFER	以太网 buffer 存储形式
FOE_SUPPORTED	FOE 协议栈使能
MAILBOX_SUPPORTED	邮箱使能
MIN_MBX_SIZE	邮箱最小 size(ESC RAM 区域)
DEF_MBX_SIZE	邮箱默认 size(ESC RAM 区域)
MAX_MBX_SIZE	邮箱最大 size(ESC RAM 区域)
MIN_MBX_WRITE_ADDRESS	写邮箱区域最小起始地址
DEF_MBX_WRITE_ADDRESS	写邮箱区域默认起始地址
MAX_MBX_WRITE_ADDRESS	写邮箱区域最大起始地址
MIN_MBX_READ_ADDRESS	读邮箱区域最小起始地址
DEF_MBX_READ_ADDRESS	读邮箱区域默认起始地址
MAX_MBX_READ_ADDRESS	读邮箱区域最大起始地址

2.3.8 Compiler

- 略

2.4 新建工程

新建模板工程，**File->New** 选择所要使用的模板，并点击 OK。

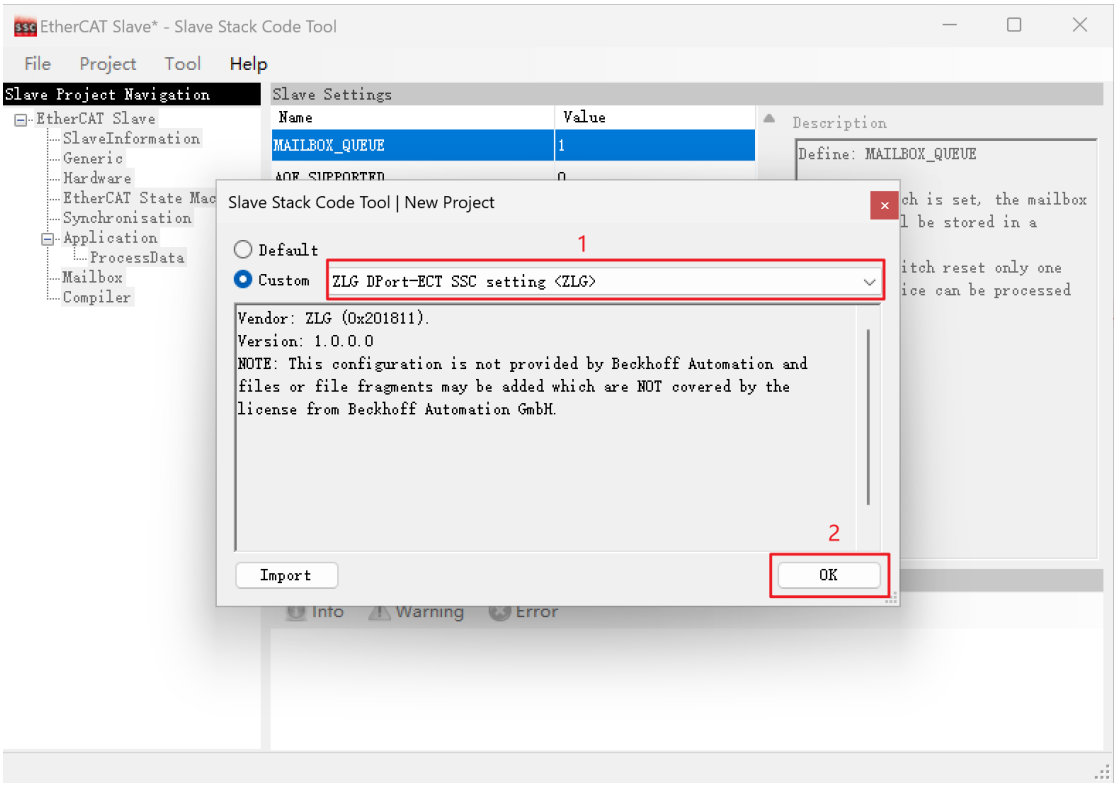


图 2.16 新建工程

新工程的配置就会和模板工程中的一致，且支持修改配置。

2.5 新建/导入对象字典

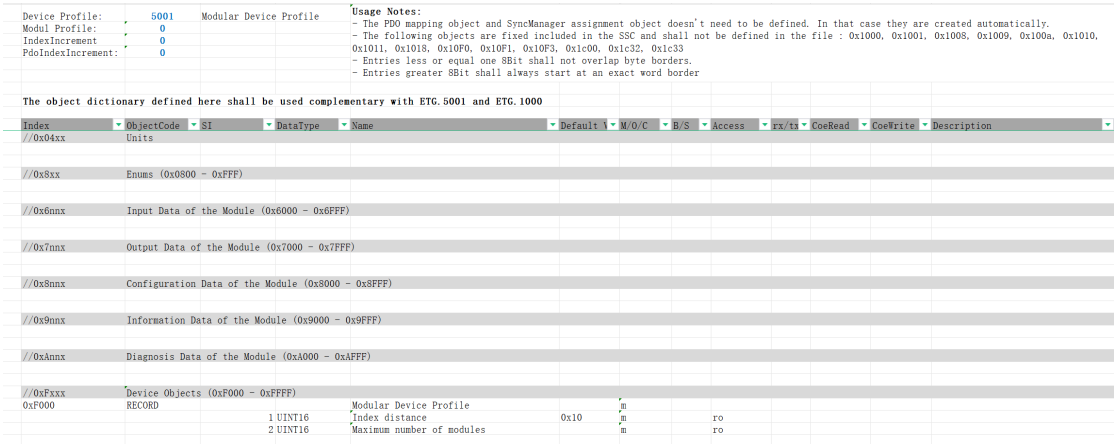
注意

SSC 中对象字典是在 Excel(CSV 也可但不常用) 中定义的，在编辑前建议补充一些 CanOpen 协议的内容和概念。

2.5.1 新建/导入 Excel 步骤

新建：Tool -> Application -> Create new：

创建新的空白的对象字典文件，需要用户编辑并添加自己的数据对象，创建的对象字典如下所示。



Device Profile:	5001	Modular Device Profile	Usage Notes:									
Modul Profile:	0		- The PDO mapping object and SyncManager assignment object doesn't need to be defined. In that case they are created automatically.									
IndexIncrement	0		- The following objects are fixed included in the SSC and shall not be defined in the file : 0x1000, 0x1001, 0x1008, 0x1009, 0x100a, 0x1010, 0x1011, 0x1018, 0x10F0, 0x10F1, 0x10F3, 0x1c00, 0x1c32, 0x1c33									
PdoIndexIncrement:	0		- Entries less or equal one 8Bit shall not overlap byte borders.									
			- Entries greater 8Bit shall always start at an exact word border									
The object dictionary defined here shall be used complementary with ETG.5001 and ETG.1000												
Index	ObjectCode	SI	DataType	Name	Default	M/O/C	B/S	Access	rx/tx	CoeRead	CoeWrite	Description
//0x04xx	Units											
//0x8xx	Enums (0x0800 - 0xFFF)											
//0x6nnx	Input Data of the Module (0x6000 - 0x6FFF)											
//0x7nnx	Output Data of the Module (0x7000 - 0x7FFF)											
//0x8nnx	Configuration Data of the Module (0x8000 - 0x8FFF)											
//0x9nnx	Information Data of the Module (0x9000 - 0x9FFF)											
//0xAnnx	Diagnosis Data of the Module (0xA000 - 0xAFFF)											
//0xFxxx	Device Objects (0xF000 - 0xFFFF)											
0xF000	RECORD			Modular Device Profile		m						
		1	UINT16	Index distance	0x10	m		ro				
		2	UINT16	Maximum number of modules		m		ro				

图 2.17 Excel 空白文件

导入：Tool -> Application -> Import：

若已有 Excel 对象文件，则只需要导入即可。

2.5.2 Excel 文件编写规则

Index

在 CANopen 协议中，索引 (Index) 是对象字典中的一个关键概念。对象字典是 CANopen 设备中用于存储配置和过程数据的表，每个条目都由一个 16 位的索引和一个 8 位的子索引 (Sub-index) 组成。

1. 索引的作用：
   - 唯一标识：索引用于唯一标识对象字典中的一个特定对象；
   - 数据访问：通过索引和子索引，可以访问和修改设备的参数、状态或配置信息；
   - 标准化通信：CANopen 标准定义了一些必须实现的索引，以确保设备之间的兼容性和通信。
2. 索引的范围和分配：

   标准范围：CANopen 协议规定了一些标准的索引范围，例如：

   - 0x0001 - 0x001F：静态数据类型；

- 0x1000 - 0x1FFF：通信对象子协议区；
- 0x2000 - 0x5FFF：制造商特定子协议区。

制造商特定：制造商可以在指定的范围内定义自己的索引来满足特定设备的需求。

3. 应用实例：
   - 设备配置：例如，设备类型 (索引 0x1000) 和设备名称 (索引 0x1008) 等信息都存储在对象字典中。
   - 控制信号：某些索引可以用于发送控制信号，如启动或停止设备。

通过索引和子索引的组合，CANopen 协议实现了设备之间的高效通信和数据交换

ObjectCode

在 CANopen 协议中，**对象代码 (Object Code)** 用于描述对象字典中对象的内部结构类型。对象代码定义了对象的数据结构，以便于正确地访问和处理这些数据。以下是 CANopen 中定义的几种主要对象代码：

ObjectCode	描述
VARIABLE	表示一个简单的变量，通常用于存储单一的数据值。
ARRAY	表示一个数组结构，数组中的所有元素具有相同的数据类型。例如，一个数组可以包含多个相同类型的值，如多个温度读数。
RECORD	表示一个记录结构，类似于 C 语言中的结构体。记录中的各个元素可以有不同的数据类型。例如，一个记录可以包含温度、压力和湿度等不同类型的值。

对象代码在 CANopen 设备的电子数据表 (EDS) 文件中被定义为“对象类型”。通过这些定义，CANopen 协议能够确保设备之间的兼容性和数据交换的准确性。

SI

在 CANopen 协议中，**子索引 (Subindex)** 是用于进一步细分对象字典中某个索引 (Index) 下的数据结构的标识符。每个对象字典条目由一个 16 位的索引和一个 8 位的子索引组成。子索引的范围是从 0x00 到 0xFF。

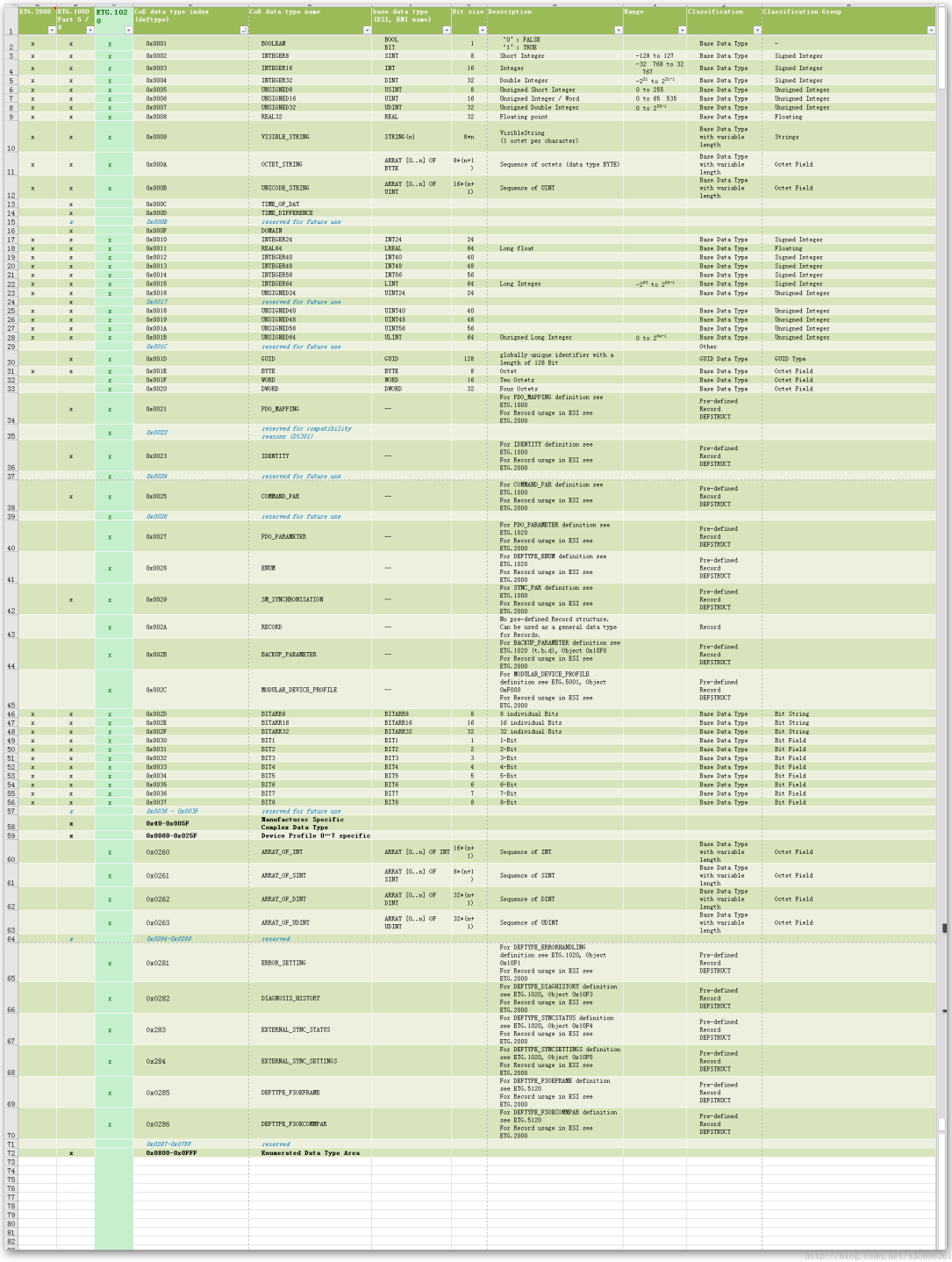
1. 子索引的作用：
   - 数据细分：对于复杂的数据结构，如数组或记录，子索引用于访问这些结构中的各个元素或字段；
   - 数量指示：在数组或记录中，子索引 0 通常用于指示该数据结构中元素的数量；
   - 唯一标识：每个索引和子索引的组合在对象字典中是唯一的，确保了对特定数据的准确访问。
2. 应用实例：
   - 数组：例如，一个设备可能有一个温度传感器数组，索引为 0x2000，子索引 0x00 表示数组中元素的数量，而子索引 0x01、0x02 等分别表示每个传感器的温度值；
   - 记录：对于一个记录结构，如设备状态信息，索引为 0x2001，子索引 0x01 可能表示温度，子索引 0x02 表示湿度。

通过使用子索引，CANopen 协议能够有效地管理和访问设备对象字典中的复杂数据结构，从而实现设备之间的高效通信和配置.

DataType

在 CANopen 协议中，**数据类型 (DataType)** 用于定义对象字典中对象的数据结构和格式。数据类型可以分为基本数据类型和复合数据类型：

1. **基本数据类型：**
    - BOOLEAN(0x01)：布尔型，表示一个逻辑值 (真或假)；
    - INTEGER8(0x02)：8 位有符号整型；
    - INTEGER16(0x03)：16 位有符号整型；
    - INTEGER32(0x04)：32 位有符号整型；
    - INTEGER64(0x15)：64 位有符号整型；
    - UNSIGNED8(0x05)：8 位无符号整型；
    - UNSIGNED16(0x06)：16 位无符号整型；
    - UNSIGNED32(0x07)：32 位无符号整型；
    - UNSIGNED64(0x1B)：64 位无符号整型；
    - REAL32(0x08)：单精度浮点型，32 位；
    - REAL64(0x11)：双精度浮点型，64 位。
    - **……**
2. **复合数据类型：**
    - VISIBLE_STRING(0x09)：可见字符串，由 8 位有符号字符组成的字符串；
    - OCTET_STRING(0x0A)：八位字符串，由 8 位无符号字符组成的字符串；
    - UNICODE_STRING(0x0B)：Unicode 字符串，由 16 位无符号字符组成的字符串；
    - TIME_OF_DAY(0x0C)：时间类型，表示一天中的时间；
    - TIME_DIFFERENCE(0x0D)：时间差类型，表示两个时间点之间的差值；
    - DOMAIN(0x0F)：域类型，用于传输大量数据。
    - **……**
3. SSC(Excel) 中对数据类型的实现为：



	ETG.2000	ETG.1000 Part 5 / F	ETG.1020	CoE data type index (deftype)	CoE data type name	base data type (ESI, EMI name)	Bit size	Description	Range	Classification	Classification Group
2	x	x	x	0x0001	BOOLEAN	BOOL BIT	1	'0' : FALSE '1' : TRUE		Base Data Type	-
3	x	x	x	0x0002	INTEGER8	SINT	8	Short Integer	-128 to 127	Base Data Type	Signed Integer
4	x	x	x	0x0003	INTEGER16	INT	16	Integer	-32 768 to 32 767	Base Data Type	Signed Integer
5	x	x	x	0x0004	INTEGER32	DINT	32	Double Integer	-2^{31} to $2^{31}-1$	Base Data Type	Signed Integer
6	x	x	x	0x0005	UNSIGNED8	USINT	8	Unsigned Short Integer	0 to 255	Base Data Type	Unsigned Integer
7	x	x	x	0x0006	UNSIGNED16	UINT	16	Unsigned Integer / Word	0 to 65 535	Base Data Type	Unsigned Integer
8	x	x	x	0x0007	UNSIGNED32	UDINT	32	Unsigned Double Integer	0 to $2^{32}-1$	Base Data Type	Unsigned Integer
9	x	x	x	0x0008	REAL32	REAL	32	Floating point		Base Data Type	Floating
10	x	x	x	0x0009	VISIBLE_STRING	STRING(n)	8*n	VisibleString (1 octet per character)		Base Data Type with variable length	Strings
11	x	x	x	0x000A	OCTET_STRING	ARRAY [0..n] OF BYTE	8*(n+1)	Sequence of octets (data type BYTE)		Base Data Type with variable length	Octet Field
12	x	x	x	0x000B	UNICODE_STRING	ARRAY [0..n] OF UINT	16*(n+1)	Sequence of UINT		Base Data Type with variable length	Octet Field
13		x		0x000C	TIME_OF_DAY						
14		x		0x000D	TIME_DIFFERENCE						
15		x		*0x000E*	*reserved for future use*						
16		x		0x000F	DOMAIN						
17	x	x	x	0x0010	INTEGER24	INT24	24			Base Data Type	Signed Integer
18	x	x	x	0x0011	REAL64	LREAL	64	Long float		Base Data Type	Floating
19	x	x	x	0x0012	INTEGER40	INT40	40			Base Data Type	Signed Integer
20	x	x	x	0x0013	INTEGER48	INT48	48			Base Data Type	Signed Integer
21	x	x	x	0x0014	INTEGER56	INT56	56			Base Data Type	Signed Integer
22	x	x	x	0x0015	INTEGER64	LINT	64	Long Integer	-2^{63} to $2^{63}-1$	Base Data Type	Signed Integer
23	x	x	x	0x0016	UNSIGNED24	UINT24	24			Base Data Type	Unsigned Integer
24		x		*0x0017*	*reserved for future use*						
25	x	x	x	0x0018	UNSIGNED40	UINT40	40			Base Data Type	Unsigned Integer
26	x	x	x	0x0019	UNSIGNED48	UINT48	48			Base Data Type	Unsigned Integer
27	x	x	x	0x001A	UNSIGNED56	UINT56	56			Base Data Type	Unsigned Integer
28	x	x	x	0x001B	UNSIGNED64	ULINT	64	Unsigned Long Integer	0 to $2^{64}-1$	Base Data Type	Unsigned Integer
29				*0x001C*	*reserved for future use*					Other	
30		x	x	0x001D	GUID	GUID	128	globally unique identifier with a length of 128 Bit		GUID Data Type	GUID Type
31	x	x	x	0x001E	BYTE	BYTE	8	Octet		Base Data Type	Octet Field
32			x	0x001F	WORD	WORD	16	Two Octets		Base Data Type	Octet Field
33			x	0x0020	DWORD	DWORD	32	Four Octets		Base Data Type	Octet Field
34		x	x	0x0021	PDO_MAPPING	--		For PDO_MAPPING definition see ETG.1000 For Record usage in ESI see ETG.2000		Pre-defined Record DEFSTRUCT	
35			x	*0x0022*	*reserved for compatibility reasons (DS301)*						
36		x	x	0x0023	IDENTITY	--		For IDENTITY definition see ETG.1000 For Record usage in ESI see ETG.2000		Pre-defined Record DEFSTRUCT	
37			x	*0x0024*	*reserved for future use*						
38		x	x	0x0025	COMMAND_PAR	--		For COMMAND_PAR definition see ETG.1000 For Record usage in ESI see ETG.2000		Pre-defined Record DEFSTRUCT	
39			x	*0x0026*	*reserved for future use*						
40			x	0x0027	PDO_PARAMETER	--		For PDO_PARAMETER definition see ETG.1020 For Record usage in ESI see ETG.2000		Pre-defined Record DEFSTRUCT	
41			x	0x0028	ENUM	--		For DEFTYPE_ENUM definition see ETG.1020 For Record usage in ESI see ETG.2000		Pre-defined Record DEFSTRUCT	
42		x	x	0x0029	SM_SYNCHRONISATION	--		For SYNC_PAR definition see ETG.1000 For Record usage in ESI see ETG.2000		Pre-defined Record DEFSTRUCT	
43			x	0x002A	RECORD	--		No pre-defined Record structure. Can be used as a general data type for Records.		Record	
44			x	0x002B	BACKUP_PARAMETER	--		For BACKUP_PARAMETER definition see ETG.1020 (t.b.d), Object 0x10F0 For Record usage in ESI see ETG.2000		Pre-defined Record DEFSTRUCT	
45			x	0x002C	MODULAR_DEVICE_PROFILE	--		For MODULAR_DEVICE_PROFILE definition see ETG.5001, Object 0xF000 For Record usage in ESI see ETG.2000		Pre-defined Record DEFSTRUCT	
46	x	x	x	0x002D	BITARR8	BITARR8	8	8 individual Bits		Base Data Type	Bit String
47	x	x	x	0x002E	BITARR16	BITARR16	16	16 individual Bits		Base Data Type	Bit String
48	x	x	x	0x002F	BITARR32	BITARR32	32	32 individual Bits		Base Data Type	Bit String
49	x	x	x	0x0030	BIT1	BIT1	1	1-Bit		Base Data Type	Bit Field
50	x	x	x	0x0031	BIT2	BIT2	2	2-Bit		Base Data Type	Bit Field
51	x	x	x	0x0032	BIT3	BIT3	3	3-Bit		Base Data Type	Bit Field
52	x	x	x	0x0033	BIT4	BIT4	4	4-Bit		Base Data Type	Bit Field
53	x	x	x	0x0034	BIT5	BIT5	5	5-Bit		Base Data Type	Bit Field
54	x	x	x	0x0035	BIT6	BIT6	6	6-Bit		Base Data Type	Bit Field
55	x	x	x	0x0036	BIT7	BIT7	7	7-Bit		Base Data Type	Bit Field
56	x	x	x	0x0037	BIT8	BIT8	8	8-Bit		Base Data Type	Bit Field
57		x		*0x0038 - 0x003F*	*reserved for future use*						
58		x		**0x40-0x05F**	**Manufacturer Specific Complex Data Type**						
59		x		**0x0060-0x025F**	**Device Profile 0~7 specific**						
60			x	0x0260	ARRAY_OF_INT	ARRAY [0..n] OF INT	16*(n+1)	Sequence of INT		Base Data Type with variable length	Octet Field
61			x	0x0261	ARRAY_OF_SINT	ARRAY [0..n] OF SINT	8*(n+1)	Sequence of SINT		Base Data Type with variable length	Octet Field
62			x	0x0262	ARRAY_OF_DINT	ARRAY [0..n] OF DINT	32*(n+1)	Sequence of DINT		Base Data Type with variable length	Octet Field
63			x	0x0263	ARRAY_OF_UDINT	ARRAY [0..n] OF UDINT	32*(n+1)	Sequence of UDINT		Base Data Type with variable length	Octet Field
64		x		*0x0264-0x0280*	*reserved*						
65			x	0x0281	ERROR_SETTING			For DEFTYPE_ERRORHANDLING definition see ETG.1020, Object 0x10F1 For Record usage in ESI see ETG.2000		Pre-defined Record DEFSTRUCT	
66			x	0x0282	DIAGNOSIS_HISTORY			For DEFTYPE_DIAGHISTORY definition see ETG.1020, Object 0x10F3 For Record usage in ESI see ETG.2000		Pre-defined Record DEFSTRUCT	
67			x	0x283	EXTERNAL_SYNC_STATUS			For DEFTYPE_SYNCSTATUS definition see ETG.1020, Object 0x10F4 For Record usage in ESI see ETG.2000		Pre-defined Record DEFSTRUCT	
68			x	0x284	EXTERNAL_SYNC_SETTINGS			For DEFTYPE_SYNCSETTINGS definition see ETG.1020, Object 0x10F5 For Record usage in ESI see ETG.2000		Pre-defined Record DEFSTRUCT	
69			x	0x0285	DEFTYPE_FSOEFRAME			For DEFTYPE_FSOEFRAME definition see ETG.5120 For Record usage in ESI see ETG.2000		Pre-defined Record DEFSTRUCT	
70			x	0x0286	DEFTYPE_FSOECOMMPAR			For DEFTYPE_FSOECOMMPAR definition see ETG.5120 For Record usage in ESI see ETG.2000		Pre-defined Record DEFSTRUCT	
71				*0x0287-0x07FF*	*reserved*						
72		x		**0x0800-0x0FFF**	**Enumerated Data Type Area**						

图 2.18　数据类型规范

4. 应用:

这些数据类型在 CANopen 设备的对象字典中被广泛使用，用于定义设备的各种参数和状态信息。例如，设备温度可能使用 REAL32 类型来表示，而设备状态标志可能使用 BOOLEAN 类型。通过这些数据类型，CANopen 协议能够确保设备之间的数据交换是准确和一致的。

Name

在 CANopen 协议中，”name”通常指的是对象字典中的对象名称。对象的名称用于描述该对象的功能或用途，例如设备类型、错误寄存器、制造商设备名称等。例如，在对象字典中，索引 1008h 的对象名称是”Manufacturer device name”，表示制造商设备名称。这些名称在 EDS 文件 (Electronic Data Sheet) 中也有详细的定义和描述。

Default

在 CANopen 协议中，”default”通常指的是设备或网络的默认设置或参数值。这些默认值在设备启动或重置时被使用，以确保设备能够以一种已知和预期的状态运行。

以下是一些常见的 CANopen 默认设置：

- 节点 ID：节点 ID 的默认值通常由设备制造商设定，但可以通过 LSS(Layer Setting Services) 进行更改。
- PDO(过程数据对象) 参数：
- 传输类型：PDO 的传输类型也有默认设置，例如默认为事件触发 (Type 255)。

这些默认设置确保了设备在启动时能够以一种标准化的方式运行，并且可以通过网络管理工具进行配置和优化。

M/O/C

M	Mandatory(强制的)
O	Optional(可选的)
C	Conditional(条件性的)

表示一个索引的实现属性，例如在 ETG5001 规范中，索引 0xF000 为必须实现的对象 (Mandatory)

Table 5: Object Areas of the Device

Index	Name	Fieldbus Gateway	Modular Device	Module Device
0xF000	Modular Device Profile	M	M	M
0xF002	Detect Modules Command	C	C	-
0xF00E	Module PDO Group Mapping Alignment PDO Number	C	C	-
0xF00F	Module PDO Group Mapping Alignment	C	C	-
0xF010…0xF01F	Module Profile List Profile information of the modules	C	C	C M if modules have different profiles
0xF020…0xF02F	Address list of the configured Modules	M	C	-
0xF030…0xF03F	Module Ident list of the configured Modules		O	
0xF040…0xF04F	Address list of detected Modules	C M if 0xF002 is supported	C	-
0xF050…0xF05F	Module Ident list of detected Modules		O	
0xF060…0xF06F	Profile Version list of detected Modules	O	O	O
0xF100…0xF10F	Device Status (TxPDO mappable)	C	C	C
0xF110…0xF11F	Device Diagnosis	C	C	C
0xF120…0xF12F	Device Statistics	C	C	C
0xF200…0xF20F	Device Control (RxPDO mappable)	C	C	C
0xF500…0xF5FF	Vendor/manufacturer specific device objects	O	O	O
0xF600…0xF6EF	Input Area of device itself (TxPDO mappable)	C	C	C
0xF6F0…0xF6FF	Input Area of device itself (TxPDO mappable) according ETG.5003.1	O	O	O
0xF700…0xF7EF	Output Area of device itself (RxPDO mappable)	C	C	C

B/S

暂时未知

Access

在 CANopen 协议中，指的是对设备对象字典 (Object Dictionary) 中参数的访问权限。

在 EtherCAT 中的访问权限类型:

权限	描述
ro	只读
wr	只写
rw	可读可写
preop_ro/wr/rw	只能在 preop 阶段进行，只读/只写/可读可写
safeop_ro/wr/rw	只能在 safeop 阶段进行，只读/只写/可读可写
op_ro/wr/rw	只能在 op 阶段进行，只读/只写/可读可写

1. 访问方式：

   - SDO(服务数据对象)：用于访问对象字典中的参数。SDO 允许设备之间进行点对点的通信，以读取或写入对象字典中的数据。例如，一个主设备可以通过 SDO 读取从设备的参数值，或者修改从设备的参数设置；
   - PDO(过程数据对象)：用于传输实时数据，如传感器读数或执行器控制信号。PDO 的数据传输是基于事件或周期性的。

2. 访问控制：

   - NMT(网络管理)：负责管理设备的状态和访问权限。NMT 可以控制设备的启动、停止和复位等操作；
   - 安全机制：某些设备可能实现额外的安全机制，以限制对特定参数的访问。例如，只有在设备处于特定的 NMT 状态时，才能修改某些关键参数。

通过这些访问权限和方式，CANopen 协议确保了设备之间的有效通信和数据交换，同时提供了灵活的设备管理和配置手段。

rx/tx

表示此对象是否可以进行 PdoConfig。

- rx：Output 数据标识；
- tx：Input 数据标识。

标识过的对象，可以在主站上进行 Input 或 Output 的数据分配。

CoeRead/CoeWrite

coe 读写函数回调定义，在 Excel 中定义 CoeRead/CoeWrite 的函数名称，例如：

Index	ObjectCode	SI	DataType	Name	Default \	M/O/C	B/S	Access	rx/tx	CoeRead	CoeWrite	Description
//0x04xx	Units											
//0x8xx	Enums (0x0800 - 0xFFF)											
//0x6nnx	Input Data of the Module (0x6000 - 0x6FFF)											
//0x7nnx	Output Data of the Module (0x7000 - 0x7FFF)											
0x7000	RECORD										Coe0x7000Re	Coe0x7000Write
		1	UINT8	var1								
//0x8nnx	Configuration Data of the Module (0x8000 - 0x8FFF)											
//0x9nnx	Information Data of the Module (0x9000 - 0x9FFF)											
//0xAnnx	Diagnosis Data of the Module (0xA000 - 0xAFFF)											
//0xFxxx	Device Objects (0xF000 - 0xFFFF)											
0xF000	RECORD			Modular Device Profile		m						
		1	UINT16	Index distance	0x10	m		ro				
		2	UINT16	Maximum number of modules		m		ro				

生成代码时，会以定义的函数名称生成一个空白函数 (需用户实现)，在 coe 进行读写时便会调用此函数以代替默认的 coe 读写规则。

```
//////////////////////////////////////////////////////////////////////////////
↪//////
/**
\param     index               index of the requested object.
\param     subindex            subindex of the requested object.
\param     objSize             size of the requested object data, calculated with␣
↪OBJ_GetObjectLength
\param     pData               Pointer to the buffer where the data can be copied to
\param     bCompleteAccess     Indicates if a complete read of all subindices of the
                               object shall be done or not
```

```
 \return    result of the read operation (0 (success) or an abort code (ABORTIDX_...
↪. defined in
           sdosrv.h))
 *////////////////////////////////////////////////////////////////////////////////
↪//////
UINT8 Coe0x7000Read(UINT16 index, UINT8 subindex, UINT32 dataSize, UINT16 MBXMEM *␣
↪pData, UINT8 bCompleteAccess) {
#if _WIN32
#pragma message ("Warning: Implement CoE read callback")
#else
 #warning "Implement CoE read callback"
#endif
 return 0;
}


////////////////////////////////////////////////////////////////////////////////
↪//////
/**
\param     index               index of the requested object.
\param     subindex            subindex of the requested object.
\param     objSize             size of the requested object data, calculated with␣
↪OBJ_GetObjectLength
\param     pData               Pointer to the buffer where the data can be copied to
\param     bCompleteAccess     Indicates if a complete read of all subindices of the
                               object shall be done or not

 \return    result of the read operation (0 (success) or an abort code (ABORTIDX_...
↪. defined in
           sdosrv.h))
 *////////////////////////////////////////////////////////////////////////////////
↪//////
UINT8 Coe0x7000Write(UINT16 index, UINT8 subindex, UINT32 dataSize, UINT16 MBXMEM *␣
↪pData, UINT8 bCompleteAccess) {
#if _WIN32
#pragma message ("Warning: Implement CoE write callback")
#else
 #warning "Implement CoE write callback"
#endif
 return 0;
}
```

注: 函数类型: xxxRead/xxxWrite(UINT16 index, UINT8 subindex, UINT32 dataSize, UINT16 MBXMEM * pData, UINT8 bCompleteAccess) 例:

Index 0x7000		
Subindex	DataType	Name
1	UINT8	Var1

续上表

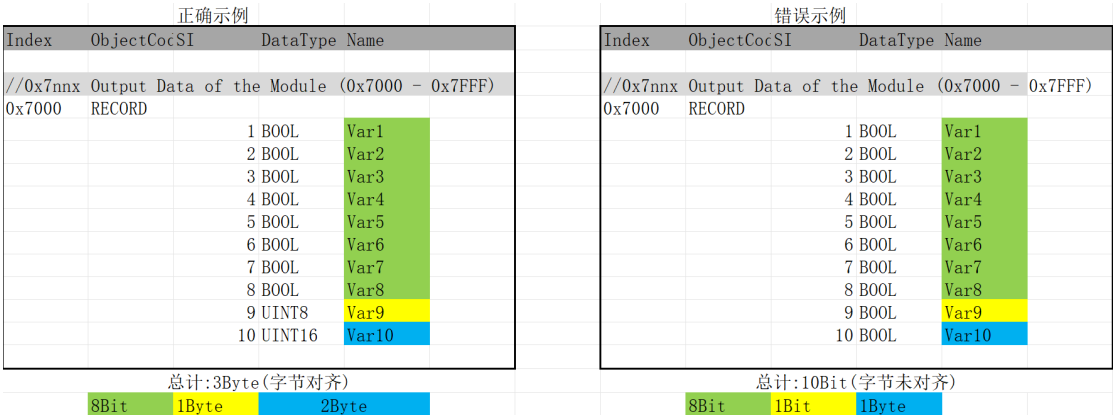
Index 0x7000		
2	UINT8	Var2
3	UINT8	Var3
4	UINT8	Var4

主站通过 sdo 对 0x7000 的 0x03 索引写值时，协议栈会调用 xxxWrite，此时函数参数

- index：0x7000
- subindex：0x03
- dataSize：sizeof(UINT8)
- pData：写的值
- bCompleteAccess：是否完全访问

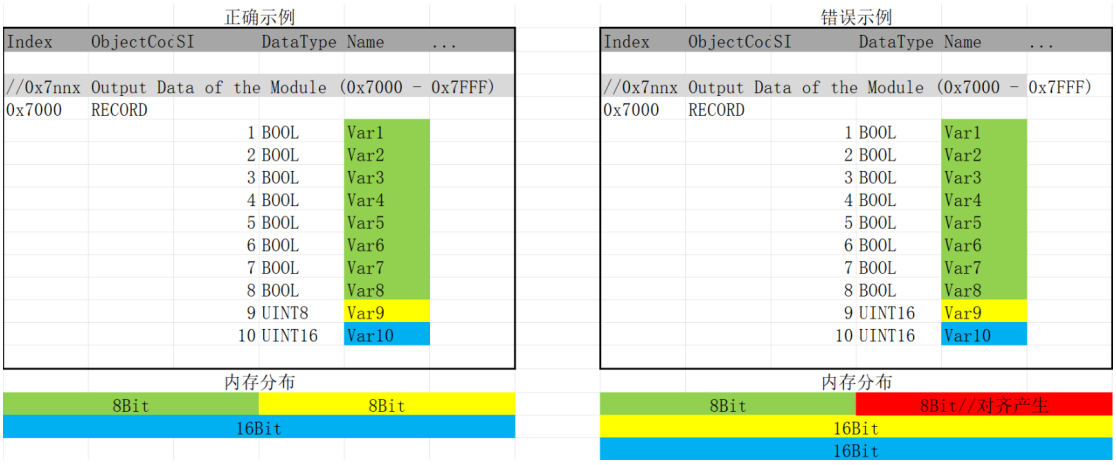
2.5.3 周期数据 (Input/Output)

1. 周期数据的总数据量必须为 Byte 的整数倍，例如：



如图所示，正确实例中的总数据量为 3Byte 是 Byte 的整数倍故，而错误示例中的总数据量为 10Bit 数据长度未对齐 Byte. 错误的长度定义会导致 SSC 工具检查的报错。

2. 同一个主索引下的周期数据存在对齐 (以最大的数据类型)，对齐过程中不允许产生多余字节：



错误示例中由于前八个成员总数居量为 8Bit，而第九个成员数据宽度为 16Bit，此时会因对齐产生多余的 8Bit，此举在 SSC 中同样不允许。

Description：

- 描述

注意

Excel 编辑对象是基于 CanOpen 协议规则

2.6 源码和 ESI 文件生成

保存项目后点击 **Project -> Create new Slave Files**，最后会生成一份 xml 文件 (ESI) 和 Src 源码，若要修改对象字典只需重新打开工程，重复之前步骤 Import 要导入的 Excel 文件即可。

2.7 应用程序与接口函数

除固定协议栈代码以外，SSC 还会依据用户导入的 Excel 生成相关的应用文件 (源码名会与 Excel 文件名相关)，以 8io 为例：

应用相关的代码会生成于 8io.c/8io.h/8ioObjects.h 这些文件中，8io.c 中存在三个已定义但内容未实现的应用函数 (需要用户实现)，这三个函数会在协议栈处理数据阶段调用。

2.7.1 void APPL_InputMapping(UINT16* pData)

- **Description：** 数据发送时的回调函数 (EtherCAT Slave to Master)
- **Parameter(pData)：** 指向发送缓冲区的指针。
- **Usage：** pData 是用作数据发送的缓冲区，用户将数据填充入 pData 后，协议栈会将其发送往主站.pData 的长度在 safeop 阶段就会被计算，并将其长度信息存储在变量 nPdInputSize 中。

```
void APPL_InputMapping(UINT16* pData)
{
    extern TOBJ6000 PDIChannel0x6000;
    memcpy(
    pData,
    (uint8_t*)(&PDIChannel0x6000) + sizeof(PDIChannel0x6000.u16SubIndex0),
    1);
}
```

2.7.2 void APPL_OutputMapping(UINT16* pData)

- **Description：** 数据接收时的回调函数 (EtherCAT Master to Slave)。
- **Parameter(pData)：** 指向接收缓冲区的指针。
- **Usage：** pData 是用作数据接收的缓冲区，协议栈将数据从 ESC 中取出后会存储在 pData 中等待处理.pData 的长度在 safeop 阶段就会被计算，并将其长度信息存储在变量 nPdOutputSize 中。

```
void APPL_OutputMapping(UINT16* pData)
{
    /* PDOChannel0x7000 include a variable in type UINT16,
    * boolean variables are not allow to take address,
    * so we move two bytes to the address of PDOChannel0x7000.LED1,
    * attention:
    * Guaranteed byte alignment,if PDO use a boolean type
    */
    extern TOBJ7000 PDOChannel0x7000;
    memcpy(
    (uint8_t*)(&PDOChannel0x7000) + sizeof(PDOChannel0x7000.u16SubIndex0),
    pData,
    1);
}
```

2.7.3 void APPL_Application(void)

- **Description：** 周期性循环函数。
- **Usage：** 此函数会被周期性调用，在于同步模式下，APPL_Application 的调用会处于 APPL_OutputMapping 与 APPL_InputMapping 之间，通常用于数据接收后进行硬件输入并采集硬件的反馈。

```
void APPL_Application(void)
{
    HAL_GPIO_WritePin(LED1_GPIO_Port,LED1_Pin,!(PDOChannel0x7000.LED1));
    HAL_GPIO_WritePin(LED2_GPIO_Port,LED2_Pin,!(PDOChannel0x7000.LED2));
    HAL_GPIO_WritePin(LED3_GPIO_Port,LED3_Pin,!(PDOChannel0x7000.LED3));
    HAL_GPIO_WritePin(LED4_GPIO_Port,LED4_Pin,!(PDOChannel0x7000.LED4));
    HAL_GPIO_WritePin(LED5_GPIO_Port,LED5_Pin,!(PDOChannel0x7000.LED5));
    HAL_GPIO_WritePin(LED6_GPIO_Port,LED6_Pin,!(PDOChannel0x7000.LED6));
    HAL_GPIO_WritePin(LED7_GPIO_Port,LED7_Pin,!(PDOChannel0x7000.LED7));
    HAL_GPIO_WritePin(LED8_GPIO_Port,LED8_Pin,!(PDOChannel0x7000.LED8));

    memcpy(
    (uint8_t*)(&PDIChannel0x6000) + sizeof(PDIChannel0x6000.u16SubIndex0),
    (uint8_t*)(&PDOChannel0x7000) + sizeof(PDOChannel0x7000.u16SubIndex0),
    1);
}
```

注意

- 协议栈配置好后，SSC 会将不必要的配置项和文件过滤，最后生成一份“剪裁”后的代码，所以当修改协议栈的配置项或者是对象字典时，需要由 SSC 重新生成, 而不是修改 **ecat_def.h** 文件;
- 代码和 ESI 文件需要配套使用 (指同时生成的一套);
- 查看协议栈的完整源码可参考安装包目录下的 **SlaveFiles.zip** 压缩包。

3. TwinCAT 操作及使用

注意

- DPort-ECT 相关所有例程测试主站均为 **TwinCAT3**
- 倍福 TwinCAT3 官方下载链接[2]

[2] https://www.beckhoff.com.cn/zh-cn/support/download-finder/search-result/?download_group=97028248

3.1 TwinCAT 下载及安装

3.1.1 TwinCAT3 硬件配置要求

实时模式下，TwinCAT 对 PC 的处理器以及网卡型号有特殊要求，处理器和网卡要须是 Intel，具体网卡型号可通过倍福官方网站进行验证。网卡型号验证链接地址[3]

如果只是使用 TwinCAT 做一些非实时性的验证，大部分处理器和网卡也是支持的。

3.1.2 TwinCAT3 安装步骤

网页链接：TwinCAT3.1(4026) 官方安装文档[4]

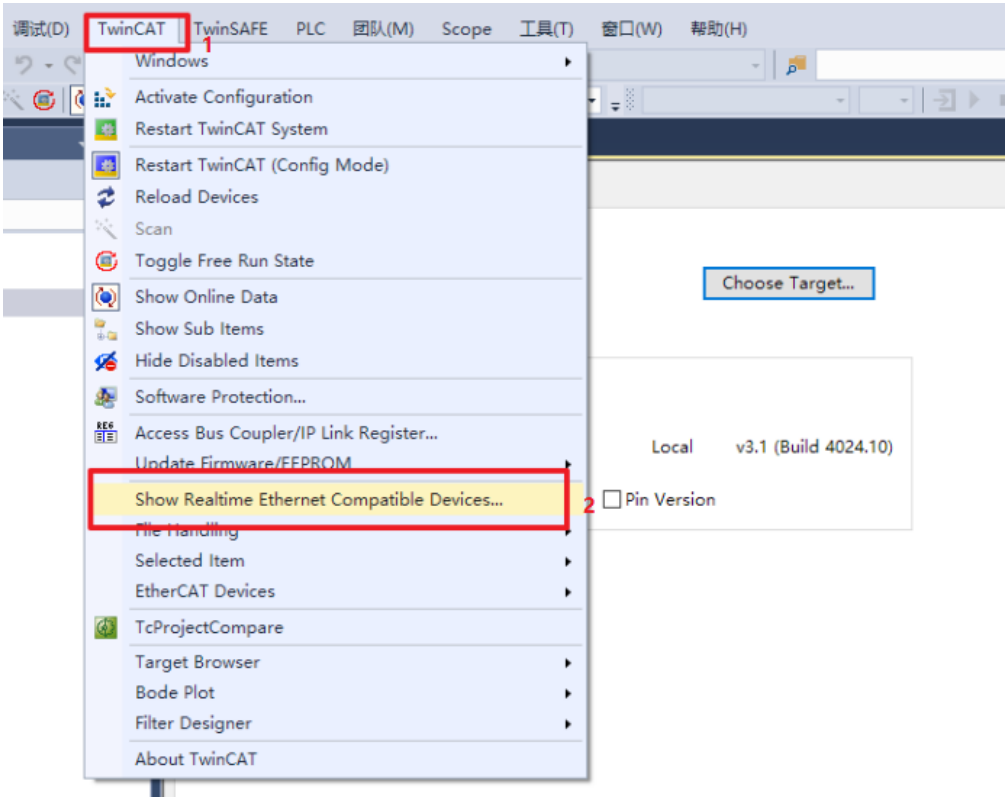
3.1.3 TwinCAT3 网卡配置

第一次安装 TwinCAT，需要对网卡进行设置，设置网卡步骤如下：

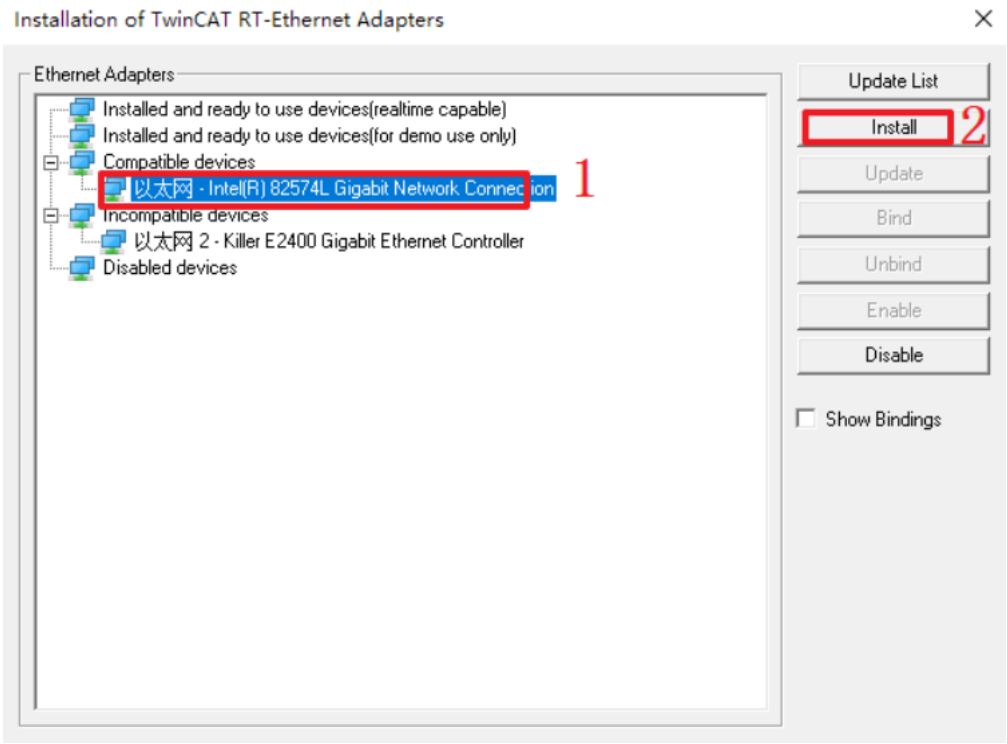
1. 点击 **TwinCAT -> show Realtime Ethernet Compatible Device** 。

[3] https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_overview/9309844363.html&id=1489698440745036069

[4] https://tr.beckhoff.com.cn/mod/resource/view.php?id=2900



2. 选择与从站连接的网卡，点击安装：



3.2 新建 TwinCAT 工程

- 1. 文件 -> 新建 -> 项目

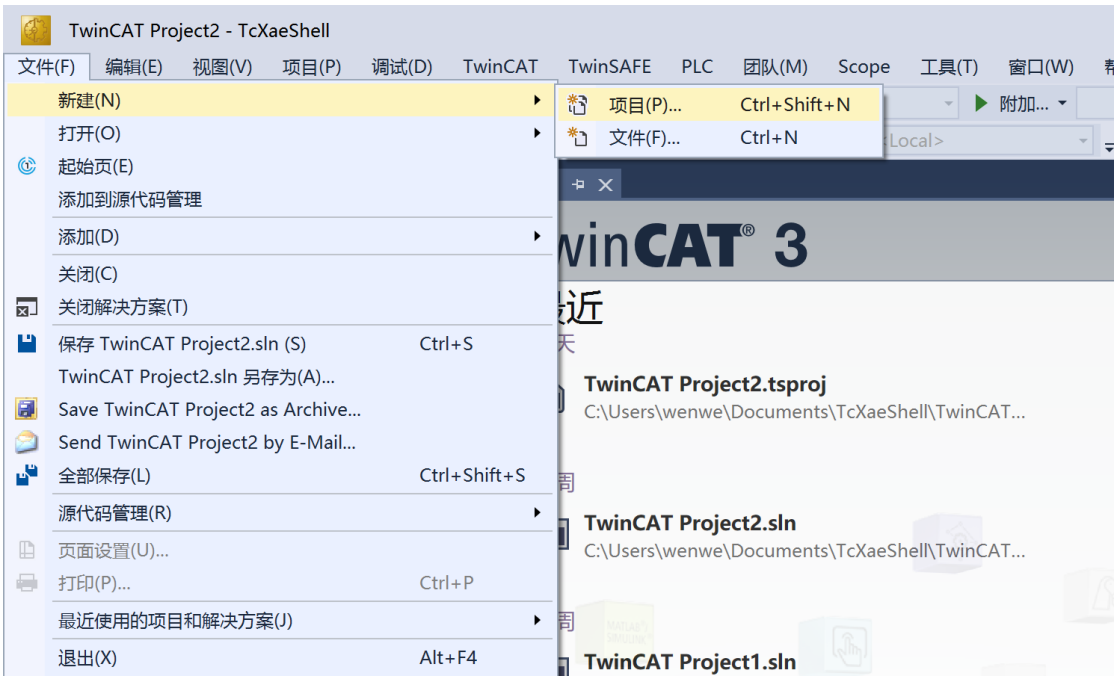


图 3.1 新建工程

- 2. **TwinCAT Projects -> TwinCAT XAE Project(XML format)**，修改项目名称与存放路径，点击确认。

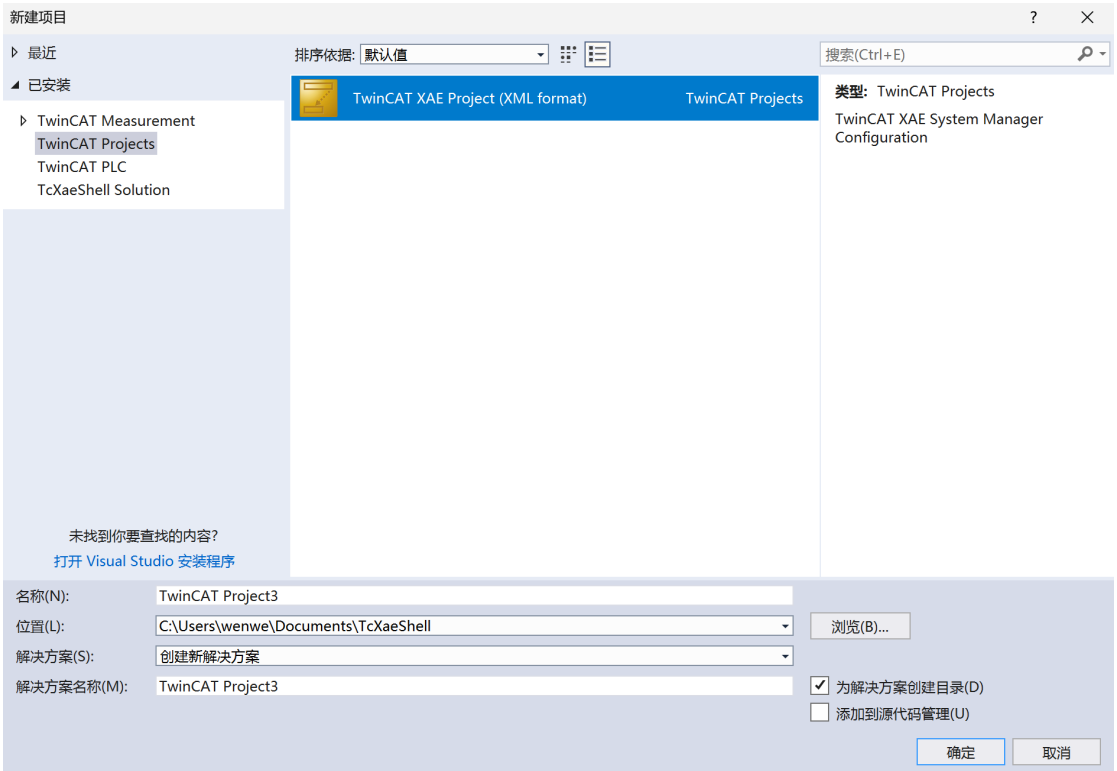


图 3.2 新建工程

- 3. 执行以上两个步骤，工程界面如下所示。

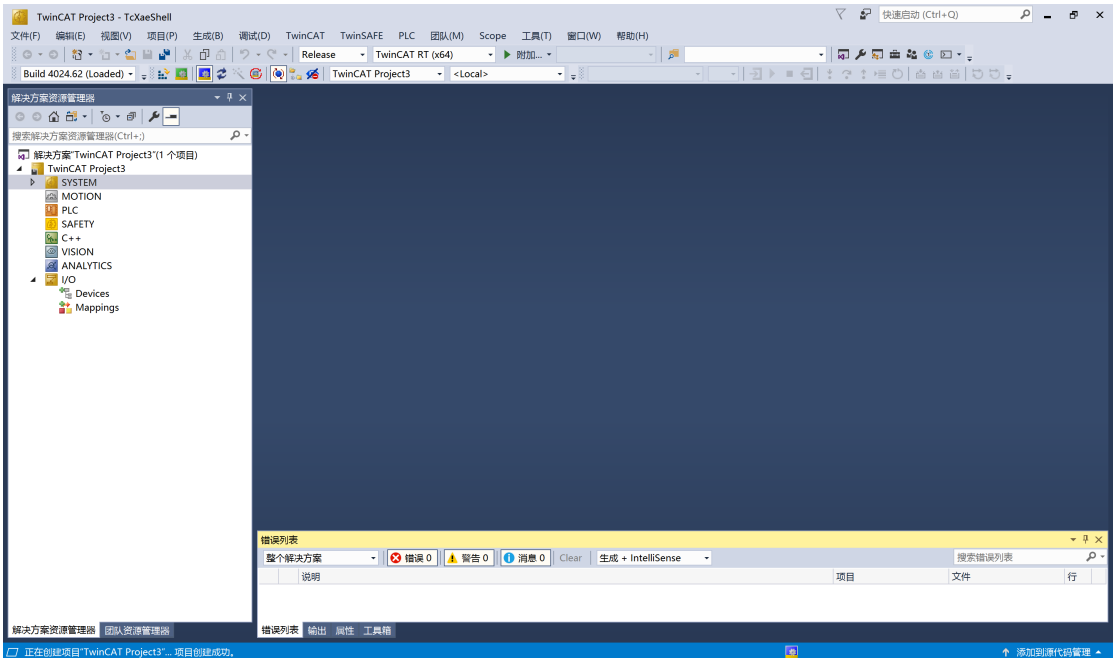


图 3.3　工程界面

3.3　试用版授权激活

- 1. 双击 **License**，弹出如下界面。

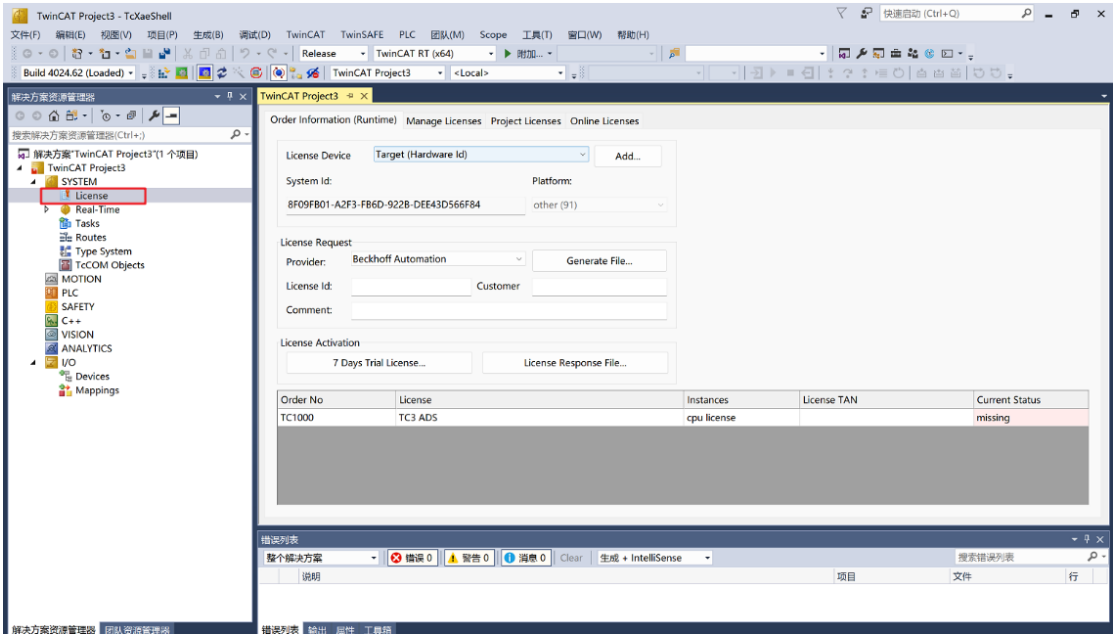


图 3.4　激活界面

- 2. 点击 **Manage License**，选择需要的 License 添加。

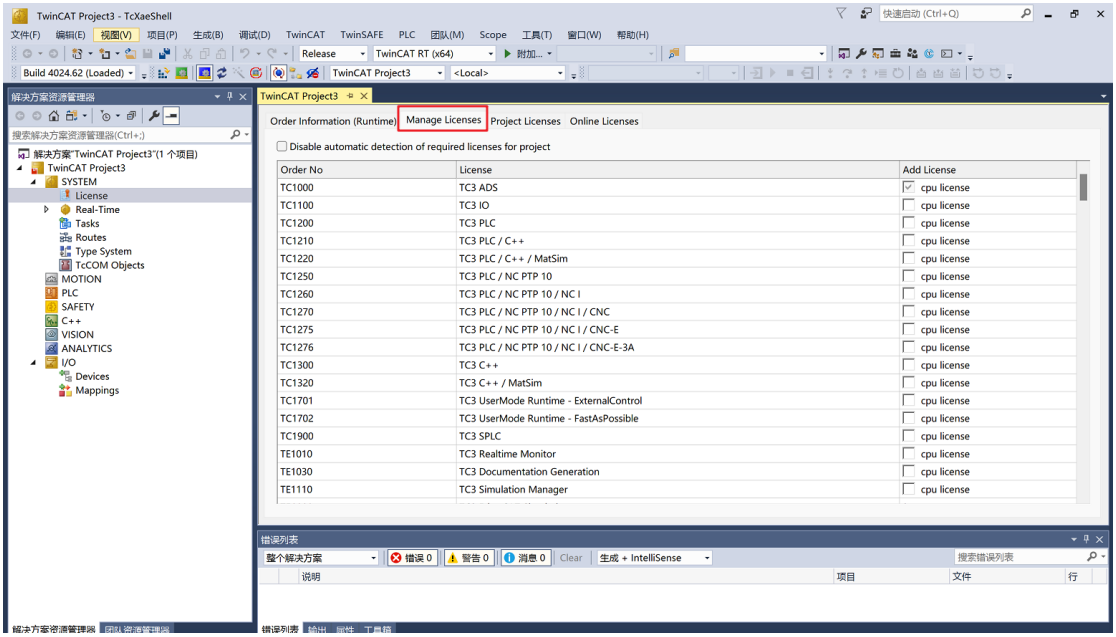


图 3.5　添加 License

- 3. 返回 **Order Information (Runtime)** 点击 **7 Days Trial License**

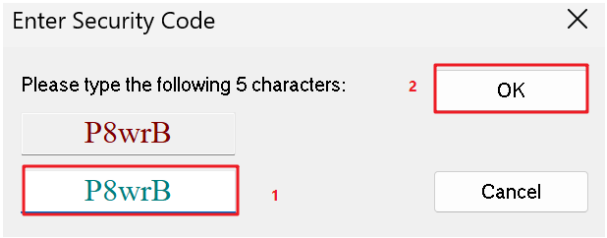


图 3.6　生成 7 天的 License

- 4. 出现如下界面则表示激活成功，超过使用期限后重复激活即可。

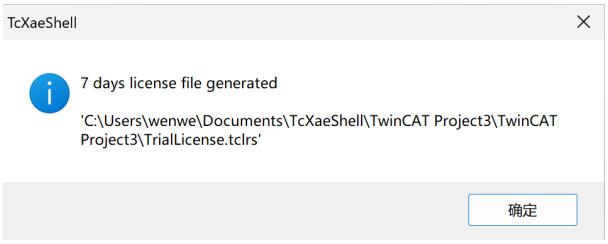


图 3.7　激活成功提示

3.4 TwinCAT ESI 文件烧录

- 1. 将 SSC 配套生成的 ESI(.xml) 文件放置在 **${TwinCAT 安装目录}/3.1/Config/Io/EtherCAT** 目录下。

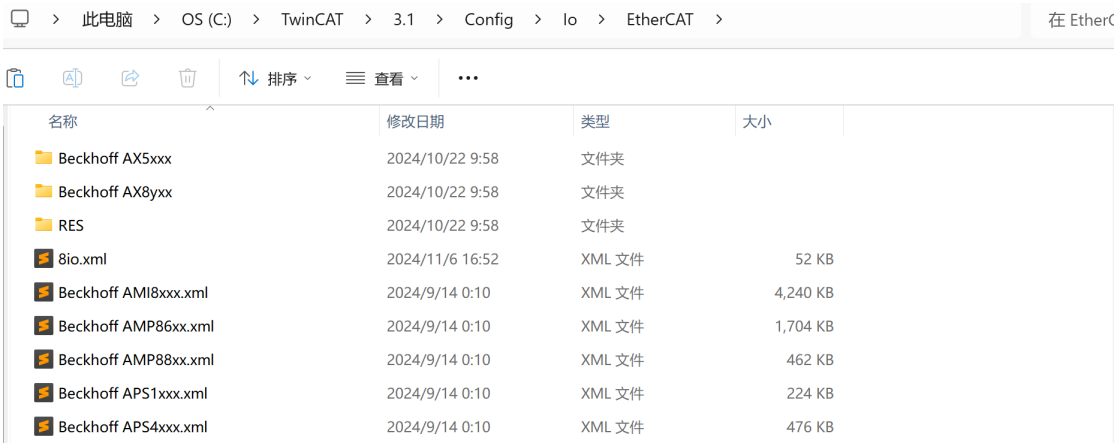


图 3.8　TwinCAT ESI 文件路径

- 2. 回到 TwinCAT 点击 **TwinCAT -> EtherCAT Devices -> Reload Device Description**，重新加载 ESI 文件。

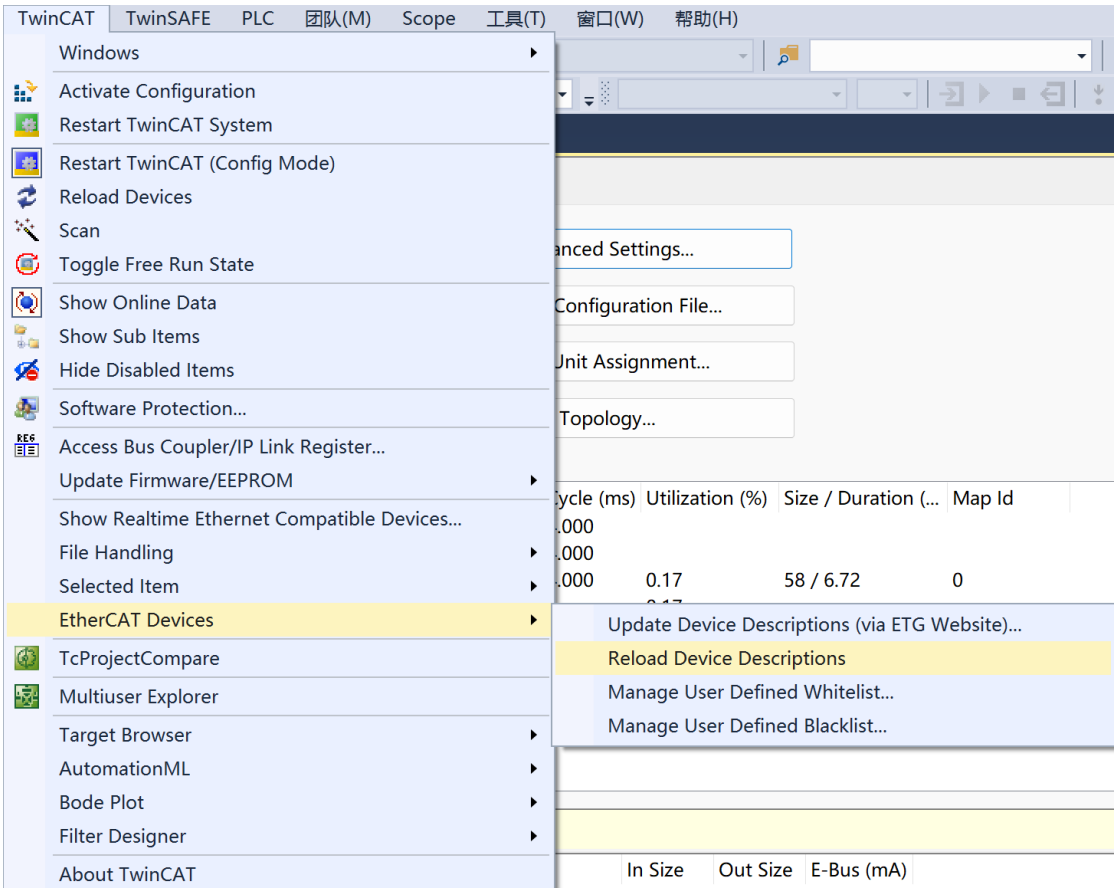


图 3.9　重加载 ESI 文件

- 3. 点击 **右键点击 Device -> Scan**，出现任意方框点击是即可。

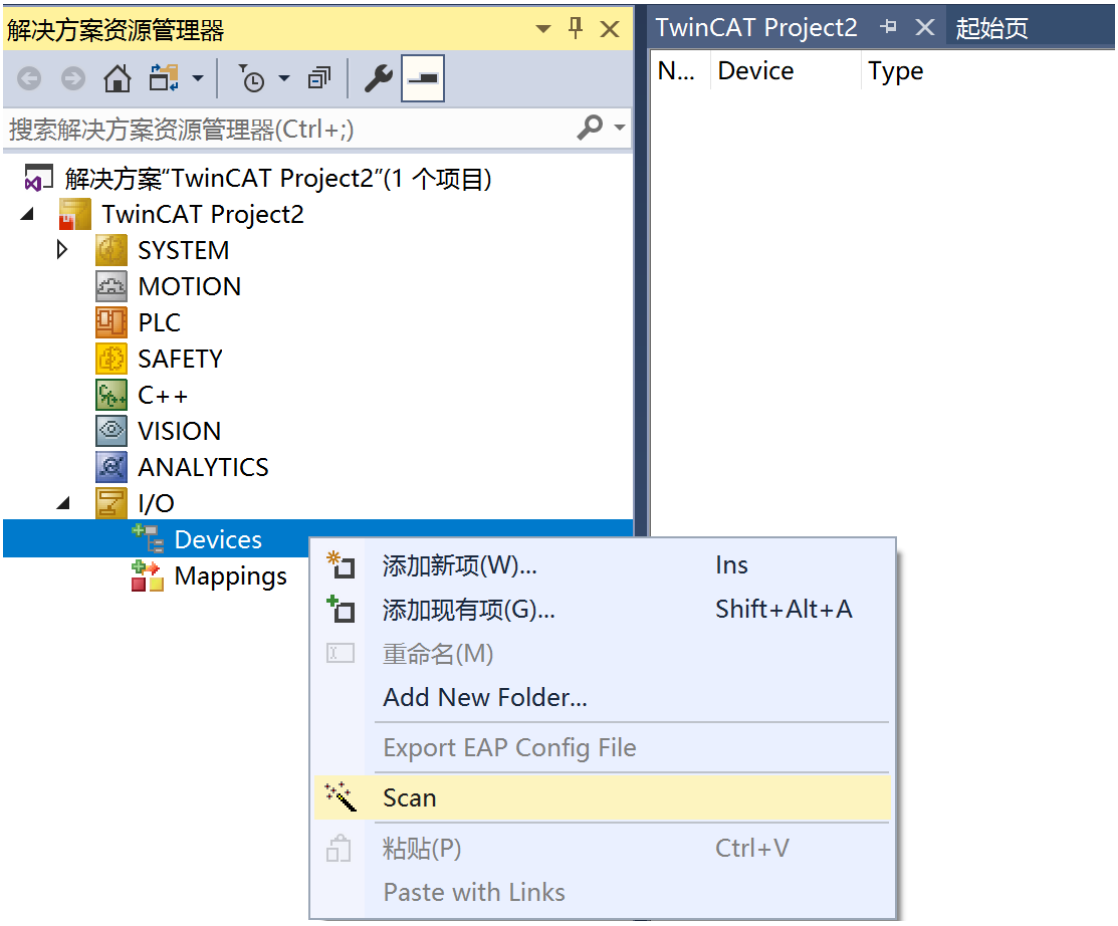


图 3.10　扫描设备

- 4. 弹出网卡扫描框后，若物理接线网卡出现 √，则表示扫描到了设备，否则请检查 TwinCAT 的网卡设置，成功检测后继续点击 OK。

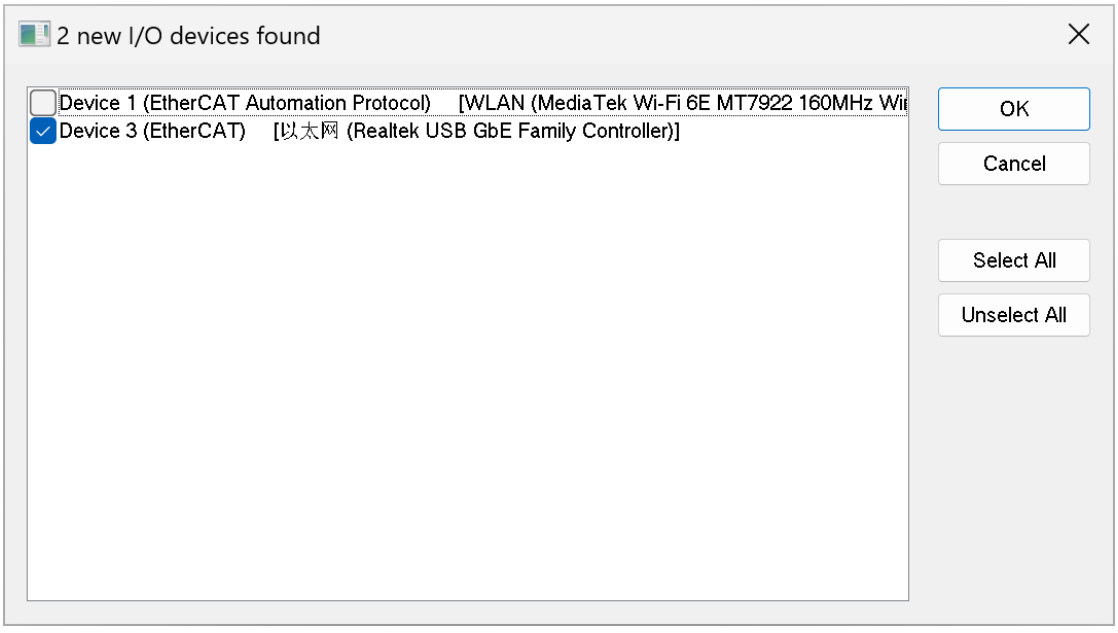


图 3.11　网卡扫描结果

出现 **Activate Free Run**，点击是 (Y)

- 5. 此时界面如图下所示，对比从站名称，若与 ESI 文件中记载相同，则 ESI 文件成功烧录。

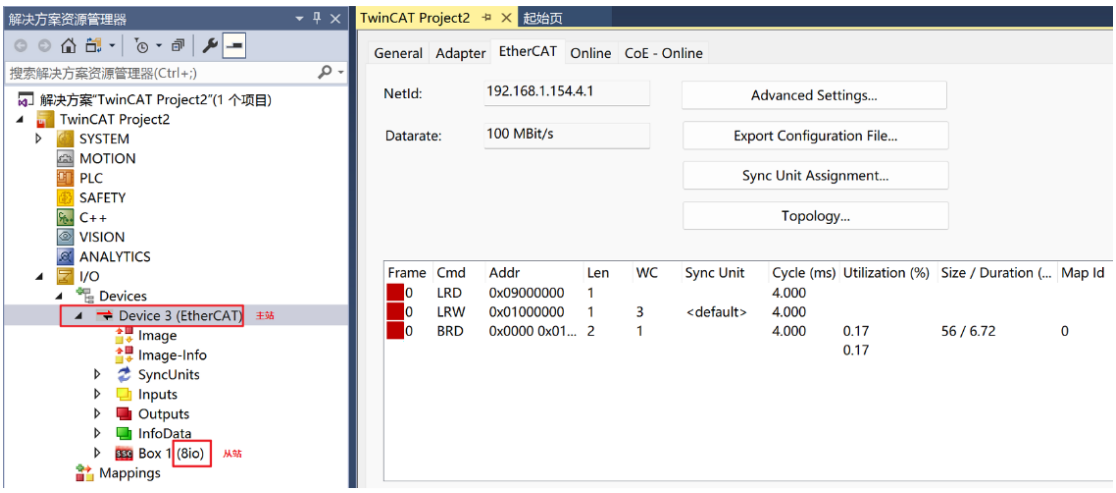


图 3.12　设备

3.5　TwinCAT 主从栈通信

3.5.1　EtherCAT 状态机详解

在扫描出从站后，TwinCAT 会自动与从站建立通讯，点击 **box -> Online** 查看从站当前的状态。

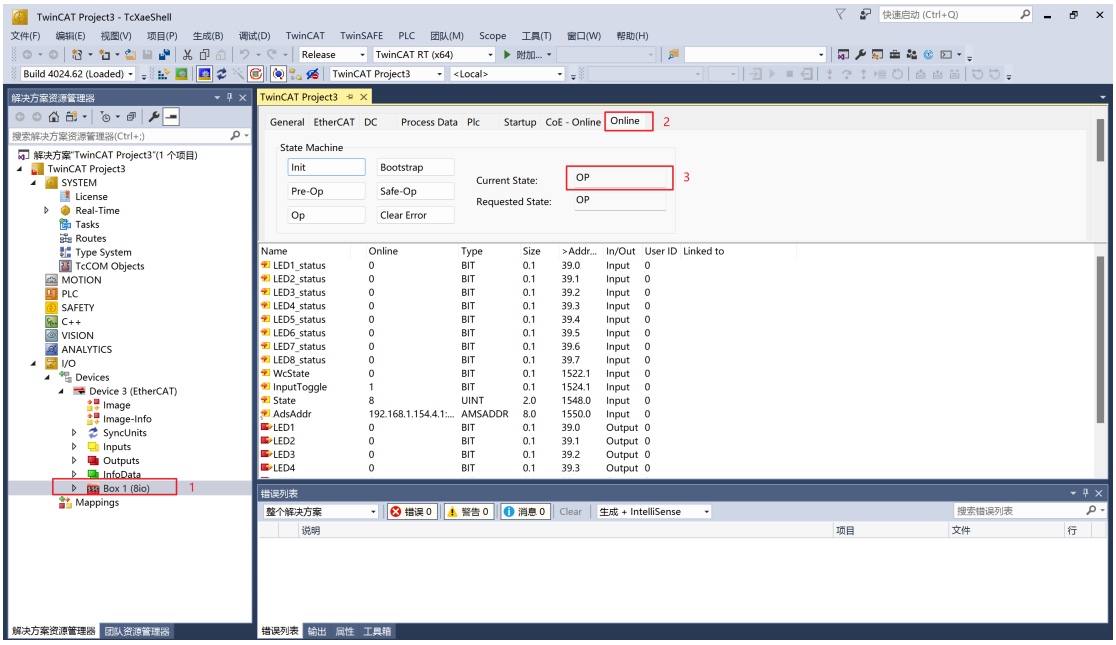


图 3.13　从站状态

若成功到达 OP，则表示通信可以正常进行。

EtherCAT 的状态转换如下：

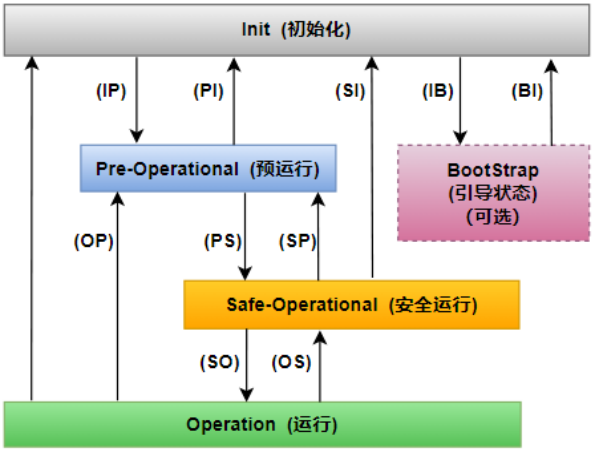


图 3.14　从站状态

EtherCAT 状态机 (EtherCAT State Machine, ESM) 用于协调主站和从站应用程序在初始化和运行时的状态关系及转换。

以下是 EtherCAT 状态机的各个状态及其功能：

- **Init(初始化)：** 从设备上电后处于此状态。主站可以读取从站的设备信息并做一些准备进入预运行状态的配置。可以清除从站 FMMU 和 SyncManagers 配置，设置从站的地址和 mailbox 信息等。如果使用 DC(分布式时钟)，可以设置 DC 的传输延时和启动的偏移时间，并进行若干次的时钟同步。
- **Pre-Operational(预运行)：** 邮箱通信可用，但过程数据通信不可用。主站可以初始化用于过程数据的同步管理器通道、FMMU 通道，以及 (如果从站支持可配置映射)PDO 映射或同步管理器 PDO 分配。在此状态下，过程数据传输的设置以及可能与默认设置不同的终端特定参数也会被传输。
- **Safe-Operational(安全运行)：** 邮箱通信和过程 (输入) 数据通信可用。在此状态下，只有输入被评估；输出保持在“安全”状态。从站会检查用于过程数据通信的同步管理器通道是否正确，如果需要，还会检查分布式时钟设置是否正确。
- **Operational(运行)：** 过程数据输入和输出有效。从站程序读取输入数据，主站应用程序发出输出数据，从站设备产生输出信号。
- **Bootstrap(引导状态)：** 可选状态，但推荐使用，特别是在需要固件更新时。无过程数据通信，仅通过邮箱在应用层通信可用。可以进行特殊的邮箱配置，例如更大的邮箱大小。通常使用 FoE(File Access over EtherCAT) 协议进行固件下载。

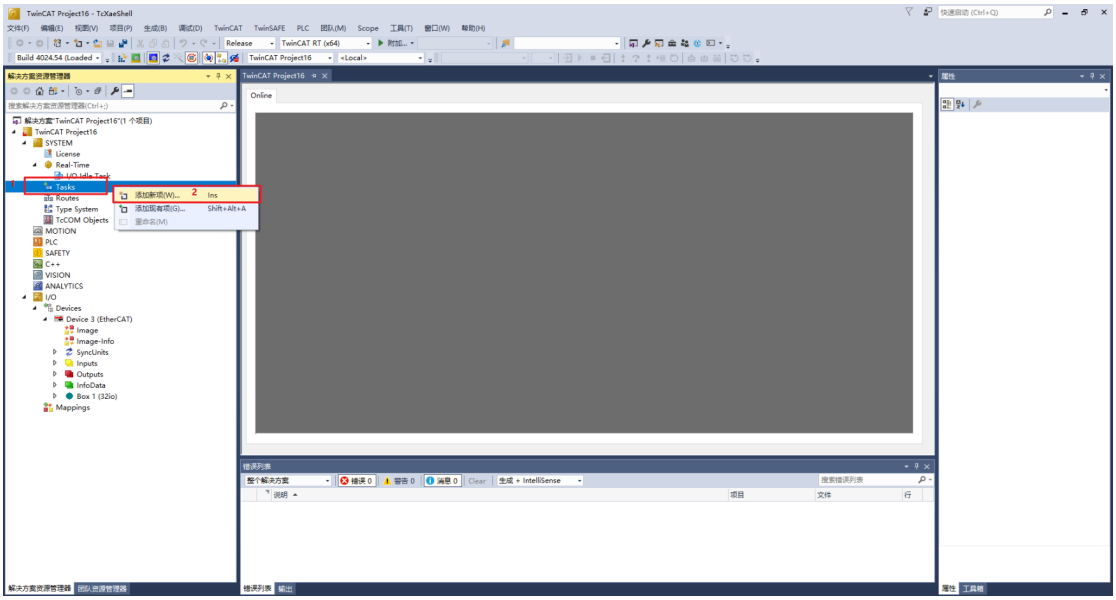
状态转换：

从初始化状态到运行状态的转换必须按照“初始化-> 预运行-> 安全运行-> 运行”的顺序进行。从运行状态返回时可以越级转换。所有的状态改变都由主站发起，主站向从站发送状态控制命令请求新的状态，从站响应此命令，执行所请求的状态转换，并将结果写入从站状态指示变量。如果请求的状态转换失败，从站将给出错误标志

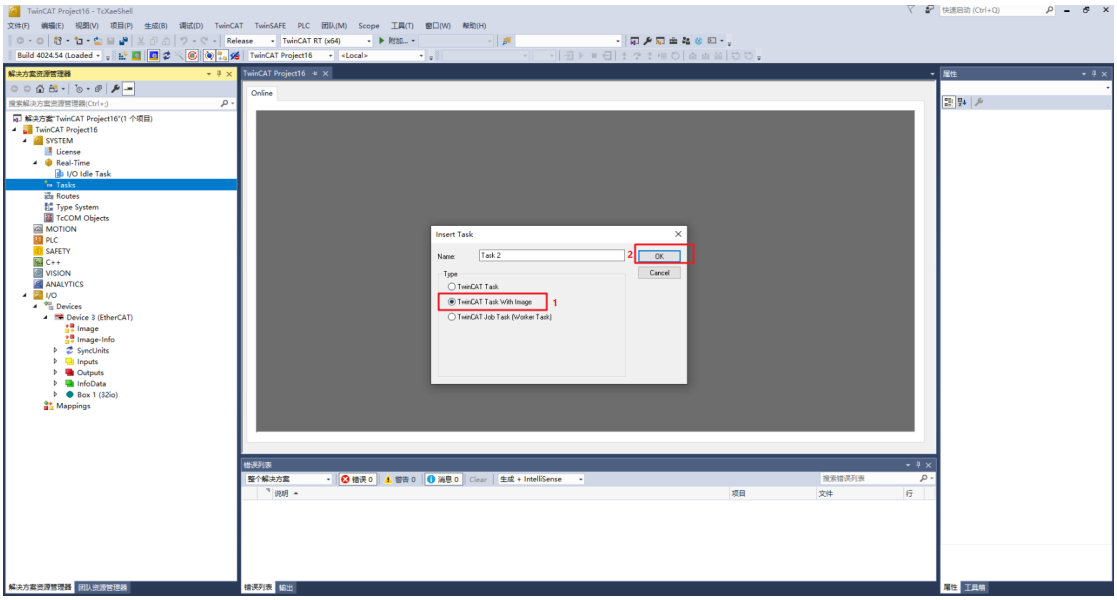
3.6 TwinCAT 实时模式

当 SM-Sync 模式到达 op 状态后，我们就可以开始测试 DC-Sync 模式了。

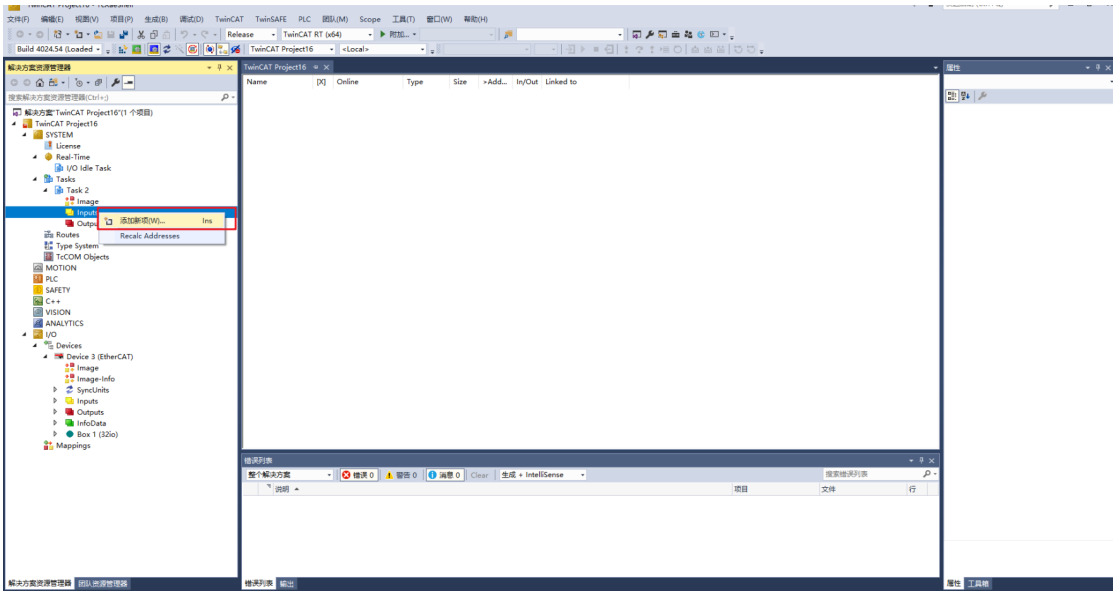
1. 右键点击 Tasks，选择添加新项



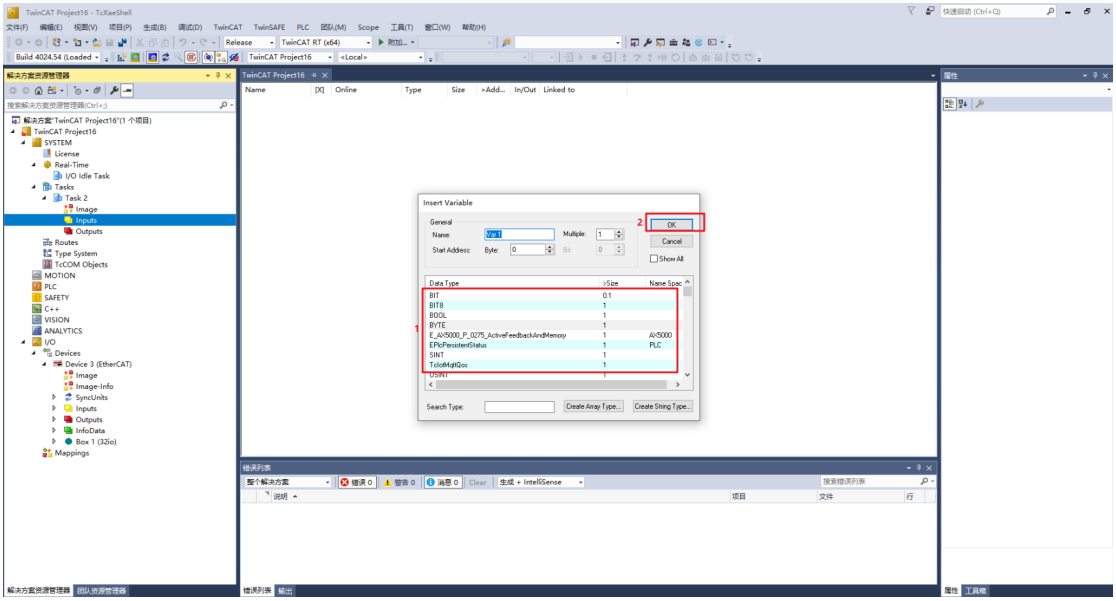
2. 选择 TwinCAT Task With Image 后点击 OK。



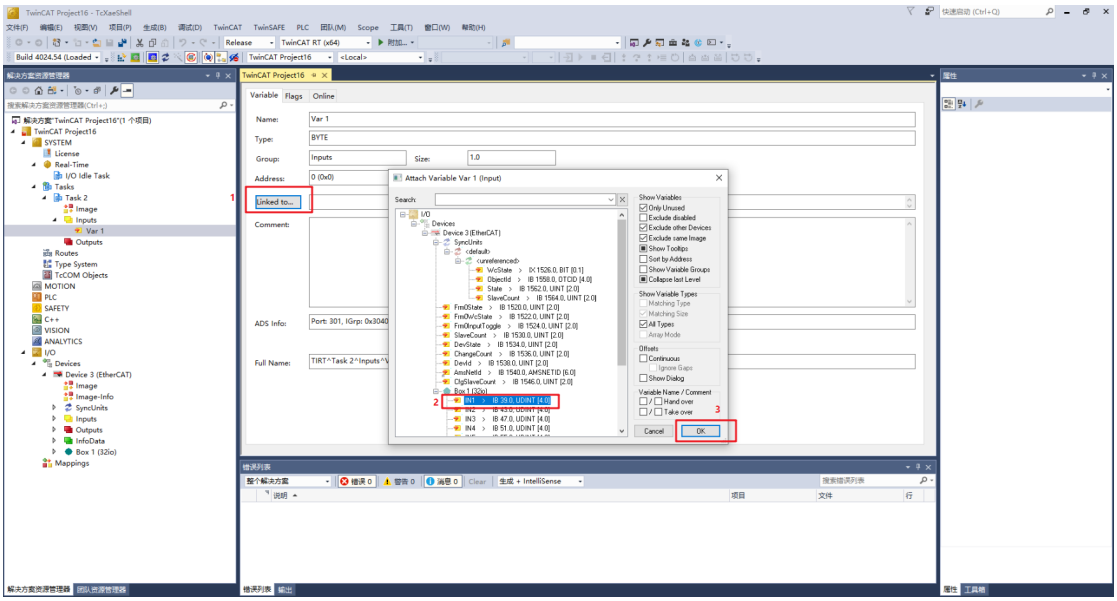
3. 点击 Input 或者 Output 添加新项 (这里以 Input 为例)，选择与要绑定数据相同的类型并点击 ok。



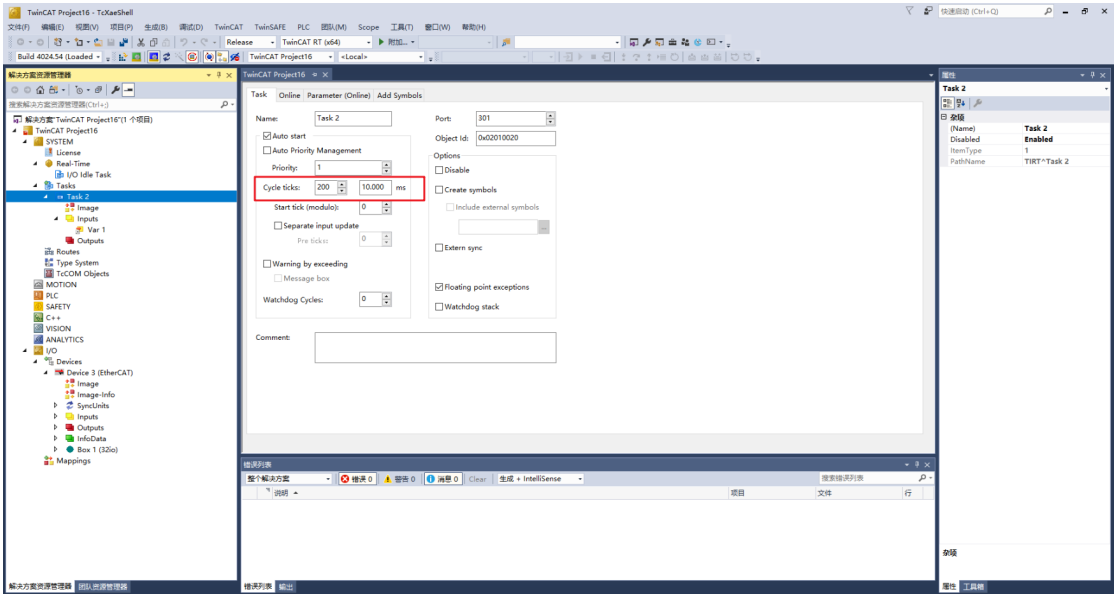
4. 点击 linked to 选择要绑定的周期数据。



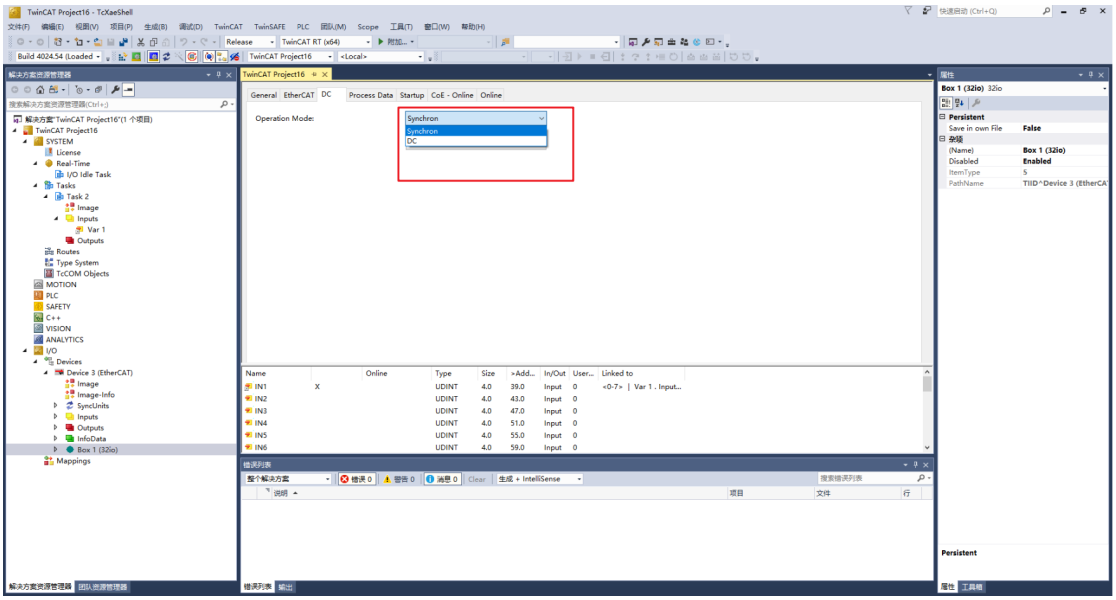
5. 数据绑定完毕后，我们可以根据自己的需求调整任务的发帧周期。



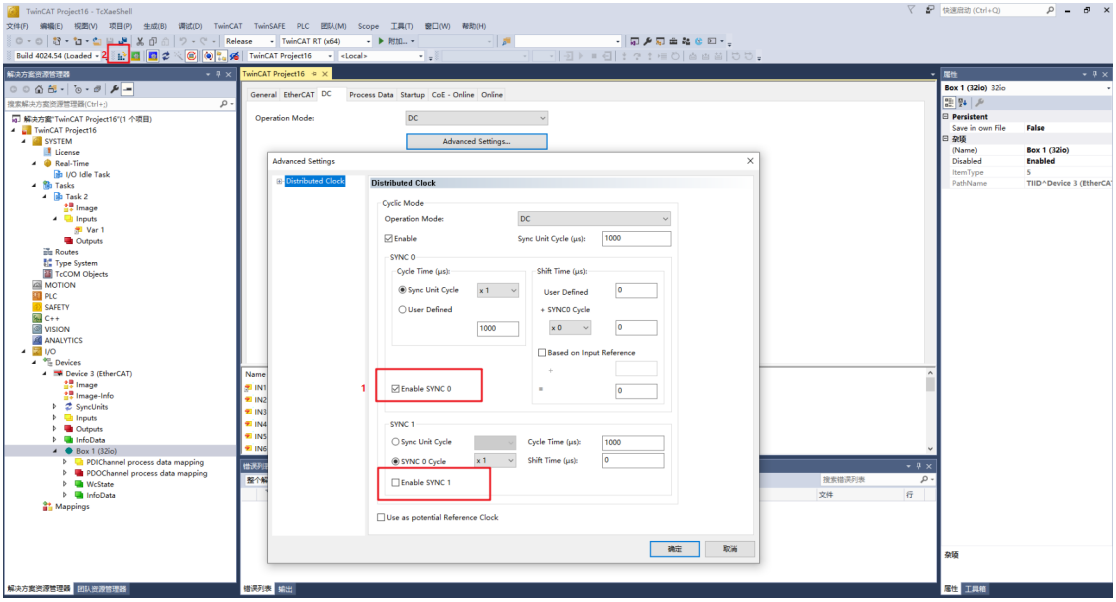
6. 回到从站界面，同步模式我们选择 DC。



5. 点击 Advanced Settings，此时会弹出 DC 的配置界面，首先需要配置 DC 的周期循环时间，配置后从站的 DC 就会按照这个循环时间周期性的触发中断，数据的处理和动作的同步。配置完毕后，勾选 Enable SYNC0 如果需要使用 SYNC1 同步信号，则一并勾选。



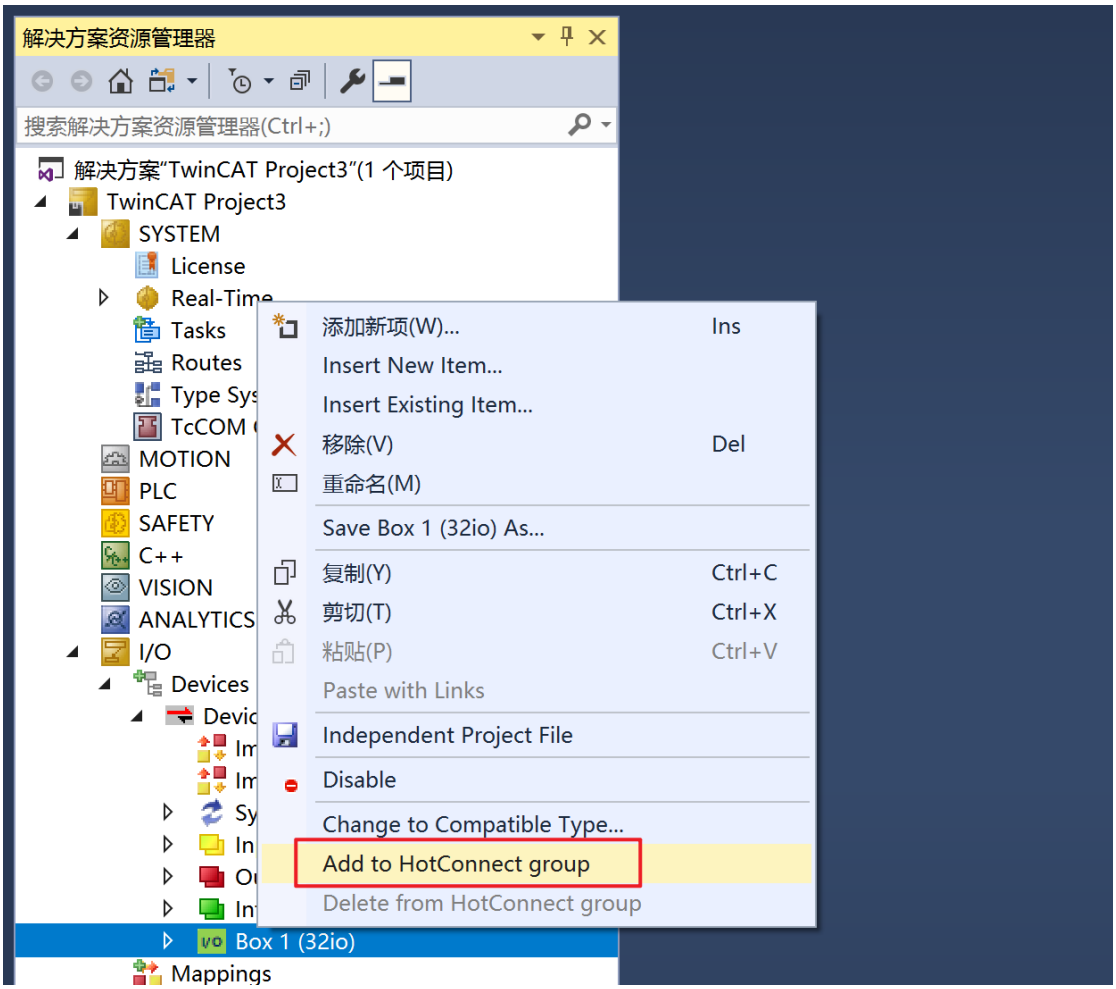
6. 勾选完毕后点击激活配置，此时 TwinCAT 会重新启动，启动成功后我们可以看见右下角图标变成绿色并且状态到达 op(配置模式下是蓝色)。



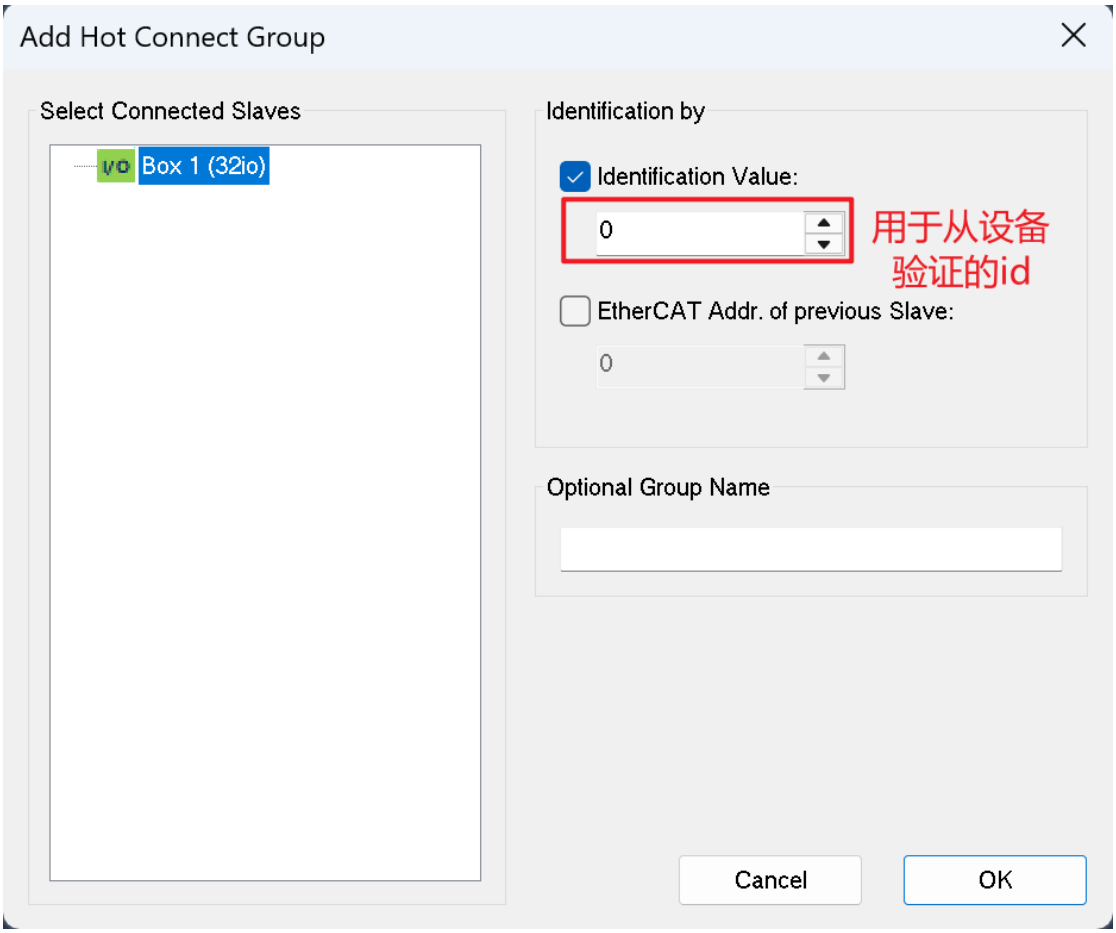
3.7 TwinCAT 设置 Box 热拔插

主站下添加从站项 (box)，或者以扫描的从站 (box)

1. 右键 **box** ，将 box 添加入热拔插组



2. 设置将用于校验从站 ID 的地址，若总线中检查到插入相应设备 ID 的从站，则会与其建立连接并到达 OP 状态



4. ESC 相关硬件配置

注意

- ESC 相关配置以 **DPort-ECT** 为例

4.1 DPort-ECT

DPort-ECT 内部芯片同属于 ET1100 系, 驱动方式与 ET1100 类似。

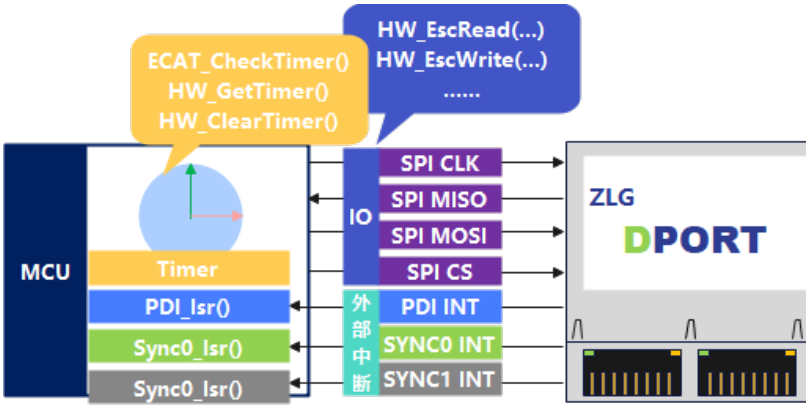


图 4.1　DPort-ECT 连接及协议栈拓扑图

事实上大部分外接 ESC 拓扑都基本相同, 除集成的 IP 核以外 (PDI,SYNC(0/1) 等信号由内核处理), 协议栈所需

- **PDI**：用于与 ESC 通信交互的接口，例如 SPI,HBI 等；
- **Sync Int**：由 ESC 产生中断信号，MCU 处理的三个同步中断 (PDI,SYNC0/1 中断)；
- **Timer**：为协议栈提供同步检测和超时处理。

4.2 协议栈硬件适配接口

SSC 协议栈所需用户实现的 ESC 访问相关函数如下：

Read Access：

- fn HW_GetALEventRegister()
- fn HW_GetALEventRegister_Isr()
- fn HW_EscRead(pData, Address, Size)
- fn HW_EscReadIsr(pData, Address, Size)
- fn HW_EscReadWord(WordValue, Address)
- fn HW_EscReadWordIsr(WordValue, Address)
- fn HW_EscReadDWord(DWordValue, Address)
- fn HW_EscReadDWordIsr(DWordValue, Address)
- fn HW_EscReadMbxMem(pData, Address, Size)

Write Access：

- fn HW_EscWrite(pData, Address, Size)
- fn HW_EscWriteIsr(pData, Address, Size)
- fn HW_EscWriteWord(WordValue, Address)
- fn HW_EscWriteWordIsr(WordValue, Address)
- fn HW_EscWriteDWord(DWordValue, Address)
- fn HW_EscWriteDWordIsr(DWordValue, Address)
- fn HW_EscWriteMbxMem(pData, Address, Size)

Key Words：

- HW_EscReadXXX：从 ESC 中读取数据；
- HW_EscWriteXXX：向 ESC 中写入数据；
- HW_EscXXXIsr：其中后缀带有 Isr 的函数表示此函数将会在中断管中调用，且函数的执行不能够被打断；
- HW_EscXXXWordXX：其中带有 Word 的函数，代表所操作的字节数为 2；
- HW_EscXXXDWordXX：其中带有 DWord 的函数，代表所操作的字节数为 4；
- Address：代表要操作的 ESC 寄存器或 DPRAM 地址；
- Size：代表要操作的字节数；
- pData：在读访问中代表用于存储从 ESC 中取出的数据的 buffer。在写访问中代表指向将要写入 ESC 的数据 buffer；
- WordValue/DWordValue：代表要进行读写的变量。

注: HW_EscXXXWordXXX 或 HW_EscXXXDWordXXX 比较特殊，其参数 WordValue/DWordValue 代表的是一个变量而不是一个指向 buffer 的指针。在 DPort-ECT 中通常会将其实现为 (附带 HW_GetALEventRegister() 的实现):

```
#define HW_EscReadWord(WordValue, Address)       \
        HW_EscRead(((MEM_ADDR *)&(WordValue)),((UINT16)(Address)),2)

#define HW_EscReadDWord(DWordValue, Address)     \
        HW_EscRead(((MEM_ADDR *)&(DWordValue)),((UINT16)(Address)),4)

#define HW_EscReadWordIsr(WordValue, Address)    \
        HW_EscReadIsr(((MEM_ADDR *)&(WordValue)),((UINT16)(Address)),2)

#define HW_EscReadDWordIsr(DWordValue, Address)  \
        HW_EscReadIsr(((MEM_ADDR *)&(DWordValue)),((UINT16)(Address)),4)

#define HW_EscWriteWord(WordValue, Address)      \
        HW_EscWrite(((MEM_ADDR *)&(WordValue)),((UINT16)(Address)),2)

#define HW_EscWriteDWord(DWordValue, Address)    \
        HW_EscWrite(((MEM_ADDR *)&(DWordValue)),((UINT16)(Address)),4)
```

```
#define HW_EscWriteWordIsr(WordValue, Address)    \
        HW_EscWriteIsr(((MEM_ADDR *)&(WordValue)),((UINT16)(Address)),2)

#define HW_EscWriteDWordIsr(DWordValue, Address) \
        HW_EscWriteIsr(((MEM_ADDR *)&(DWordValue)),((UINT16)(Address)),4)

/* 0x220U 为ESC内部的事件寄存器地址 */
void HW_GetALEventRegister(void)
{
    UINT16 ALEventReg;
    HW_EscRead(&ALEventReg, 0x220U, 2);
    return ALEventReg;
}

void HW_GetALEventRegister_Isr(void)
{
    UINT16 ALEventReg;
    HW_EscReadIsr(&ALEventReg, 0x220U, 2);
    return ALEventReg;
}
```

若是使能 (AL_EVENT_ENABLED == 1) 后，周期数据需要由中断处理，在中断中运行 void PDI_Isr(void) 函数。

若是使能 (DC_SUPPORTED == 1) 后，主站开启 DC 模式，从站则需要在中断中运行 void Sync0_Isr(void)/void Sync1_Isr(void) 函数。

SSC 协议栈所需用户实现的定时器相关函数如下：

未使用定时器中断时 (ECAT_TIMER_INT == 0) 或者使能 ESC_EEPROM_ACCESS_SUPPOTOM(EEPROM 的访问函数需要超时处理)，需要实现以下函数和宏：

- marco ECAT_TIMER_INC_P_MS：定时器计时 1ms 会有多少个 tick；
- fn UINT16 HW_GetTimer(void)：获取定时器当前 tick 值；
- fn void HW_ClearTimer(void)：清除定时器 tick 值。

若 (ECAT_TIMER_INT == 1) 时，需要提供一个 1ms 的定时器中断，周期性的运行协议栈提供的 void ECAT_CheckTimer(void) 函数。

4.3 EEPROM

EEPROM 的大小实际取决于 ESC 上电时的 bootstrap 配置：

表 4.1 Register EEPROM Control/Status (0x502[7])

Bit	Description
7	Selected EEPROM Algorithm: 0: 1 address byte (1Kbit - 16Kbit EEPROMs) 1: 2 address bytes (32Kbit - 4 Mbit EEPROMs)

通常 ESC 会提供外部引脚来于用户锁存 EEPROM 的配置信息，寻址模式 (1 address byte or 2 address byte) 需要结合实际的 EEPROM 的时序选择。

注意

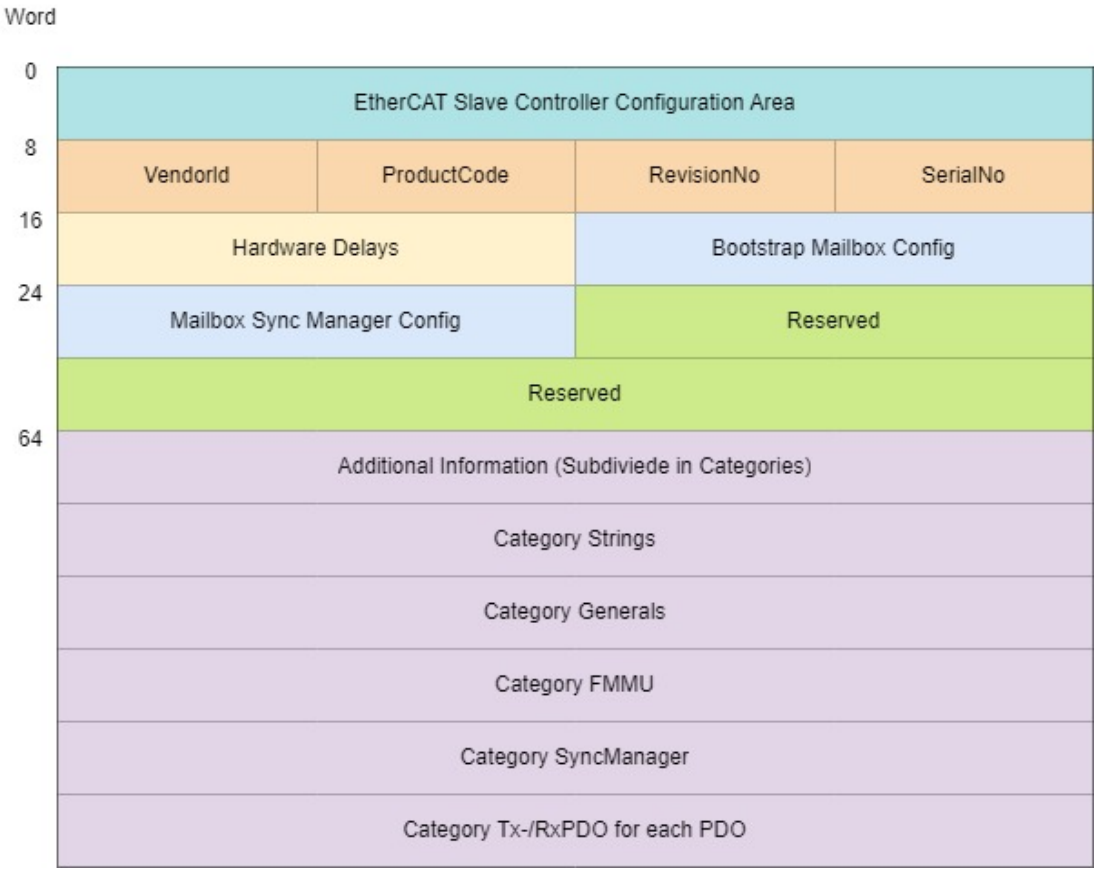
DPort-ECT 的 EEPROM 硬件连接的是 16Kbit，也就是说 DPort-ECT 的大小为 16Kbit(0x800 Byte)。

4.4 ESC Configuration Data(Handware Config)

Configuration data 定义于 ESI 文件中, 烧写 ESI 文件时, 会将其烧写在 EEPROM 中前八个 Word(16Byte)

程序清单 4.1 ESI 文件中的 ConfigData 字段

```
......
<Eeprom>
    <ByteSize>2048</ByteSize>
    <ConfigData>050C0344102700000000</ConfigData>
</Eeprom>
......
```



Configuration data 在 ESC 上电或者复位时加载进入对应的 ESC 寄存器中

Word Address	Parameter	Register Address
0x0	PDI Control/ESC Configuration	0x0140:0x0141
0x1	PDI Configuration	0x0150:0x0151
0x2	Pulse Length of SYNC Signals	0x0982:0x0983
0x3	Extended PDI Configuration	0x0152:0x0153
0x4	Configured Station Alias	0x0012:0x0013
0x5	Reserved	
0x6	Reserved	
0x7	Checksum	

4.4.1 PDI Control register(SPI interface select)

topology:

Bit	Description
7:0	Process data interface: 0x00:interface deactivated(no PDI) 0x01:4 Digital Input 0x02:4 Digital Output 0x03:2 Digital Input and 2 Digital Output 0x04:Digital I/O 0x05:SPI Slave 0x06:Oversampling I/O 0x07:EtherCAT Bridge(port3) 0x08:16 Bit asynchronous Microcontroller interface 0x09:8 Bit asynchronous Microcontroller interface 0x0A:16 Bit synchronous Microcontroller interface 0x0B:8 Bit synchronous Microcontroller interface 0x10:32 Digital Input and 0 Digital Output 0x11:24 Digital Input and 8 Digital Output 0x12:16 Digital Input and 16 Digital Output 0x13:8 Digital Input and 24 Digital Output 0x14:0 Digital Input and 32 Digital Output 0x80:On-chip bus Others:reserved

Usage:

用于选择 ESC 的通信接口 (PDI), 例如 DPort-ECT 的 PDI 只支持 SPI Slave。

注意

此寄存器的内容并非一成不变, 各家厂商会根据设计的 EtherCAT IP 外围电路添加些许内容, 典型的就是 MicroChip LAN925x 系列。

4.4.2 ESC Configuration register(SYNC0/1 Int Out enabled)

topology:

表 4.3 ESC Configuration register

Bit	Description
0	Device emulation (control of AL status): 0:AL status register has to be set by PDI 1:AL status register will be set to value Written to AL control register
1	Enhanced Link detection all port: 0:disabled (if bits[7:4]=0) 1:enabled at all port(overrides bits[7:4])
2	Distributed Clocks SYNC Out Unit: 0:disabled(power saving) 1:enabled
3	Distributed Clocks Latch In Unit: 0:disabled(power saving) 1:enabled
4	Enhanced Link port0: 0:disabled(if bit=0) 1:enabled
5	Enhanced Link port1: 0:disabled(if bit=0) 1:enabled
6	Enhanced Link port2: 0:disabled(if bit=0) 1:enabled
7	Enhanced Link port3: 0:disabled(if bit=0) 1:enabled

Usage:

- bit[0]:

set 0：主站和从站状态转换规律如下：

(1). 主机控制从机状态转换, 将目的状态写入从站 AL 控制位 (0x0120[0:3]);

(2). 从机读取新状态请求后, 检查自身状态:

a: 如果可以转换, 则将新状态写入状态机—实际状态位 (0x130[0:3]);

b: 如果不能转换, 则不改变实际状态位, 设置错误指示位 (0x130[4]), 并将错误码写入 (0x134~0x135);

(3). 主站读取状态机实际状态 (0x130):

a: 如果正常转换, 则执行下一步操作;

b: 如果转换出错, 主站读取错误码, 并写 AL 错误应答 (0x120[4]) 来清除 AL 错误指示;

set 1：AL status register 的值由控制 AL 控制寄存器的值决定

- bit[1]: 使能增强型来链路检测, 这里贴上官方概念解释

  Enhanced Link Detection

  For Ethernet, the enhanced MII link detection feature is a feature of link error detection and reaction. This has to be distinguished from the actual link detection, which tells the ESC if a physical link is available (i.e., the LINK_MII signal or the MI link detection and configuration mechanism). Enhanced MII link detection will additionally disconnect a link if at least 32 RX errors (RX_ER) occur in a fixed interval of time (~10 µs). The local loop is closed and the link partner is informed by restarting the Auto-Negotiation mechanism via the MII Management Interface. This informs the link partner of the error condition, and the link partner will close the loop. The ESC keeps the port closed until the link goes down during Auto-Negotiation and comes up again (the port remains closed if the link does not go down). The availability of Enhanced MII Link Detection depends on a supported PHY address configuration, otherwise it has to be disabled.

- bit[2]: 分布式时钟同步输出单元使能
- bit[3]: 分布式时钟锁存单元使能
- bit[4:7]: 端口增强型链路使能

4.4.3 PDI Configuration register(SPI mode and PDI Int Config)

topology:

表 4.4 PDI Configuration register

Bit	Description
1:0	SPI mode: 00:SPI mode0 01:SPI mode1 10:SPI mode2 11:SPI mode3 NOTE:SPI mode3 is recommended for Slave Sample Code NOTE:SPI status flag is not available in SPI modes0 and 2 with normal data out sample
3:2	SPI_IRQ output driver.polarity: 00: Push-Pull active low 01: Open Drain (active low) 10: Push-Pull active high 11: Open Source (active high)
4	SPI_SEL polarity: 0: Active low 1: Active high
5	Data Out sample mode: 0: Normal sample (SPI_DO and SPI_DI are sampled at the same SPI_CLK edge) 1: Late sample (SPI_DO and SPI_DI are sampled at different SPI_CLK edges)

续上表

Bit	Description
7:6	Reserved, set EEPROM value to 0

Usage:

- bit[1:0]: SPI 通信模式, 示例代码中为 mode3
- bit[3:2]: PDI 中断信号的有效极性
- bit[4]: SPI CS 引脚有效极性
- bit[5]: 数据输出采样模式

4.4.4 Pulse Length of SYNC Signals(SYNC0/1 Signals length)

topology:

表 4.5 PDI Configuration register

Bit	Description
15:0	ECAT PDI Reset Value Pulse length of SyncSignals (in Units of 10ns) 0: Acknowledge mode: SyncSignal will be cleared by reading SYNC[1:0] Status register

Usage: SYNC0/1 中断信号有效电平长度

4.5 ESC SPI Slave

ConfigData(050C0344102700000000) 对应着以下的硬件配置：

SPI:

- mode3(空闲时是高电平，从第二个跳变沿（下降沿）开始采样)；
- msb(最高位有效)；
- 8bit；
- 软件控制 CS 硬件；
- CS 低电平有效。

PDI/SYNC0/SYNC1 中断：

- 触发方式：下降沿触发。

表 4.6 SPI commands CMD0 and CMD1

CMD[2]	CMD[1]	CMD[0]	Command
0	0	0	NOP(no operation)
0	0	1	reserved
0	1	0	Read
0	1	1	Read with following Wait State bytes
1	0	0	Write
1	0	1	reserved
1	1	0	Address Externsion(3 address/command bytes)
1	1	1	reserved

spi 接口支持两种寻址模式,2byte addressing 和 3byte addressing 模式

注:

- 2byte mode(寻址：0~8k): 13bit address + 3bit command
- 3byte mode(寻址：0~64k): 16bit address + 6bit command + 2bit reserved

4.5.1 ESC SPI Read Access

表 4.7 Address modes for Read access with Wait state byte

Byte	2 Byte address mode	3 Byte address mode
0	A[12:5] address bit [12:5]	A[12:5] address bit [12:5]
1	A[4:0] address bit [4:0] CMD0[2:0] read command: 011b	A[4:0] address bit [4:0] CMD0[2:0] read command: 011b
2	0xFF wait state byte	A[4:0] address bit [4:0] CMD0[2:0] read command: 011b res[1:0] two reserved bits, set to 00b
3	D0[7:0] data byte 0	0xFF wait state byte
4	D1[7:0] data byte 1	D0[7:0] data byte 0
5 ff.	D2[7:0] data byte 2	D1[7:0] data byte 1

- Read Access

  在数据阶段, 从站会发送读取的数据到主站

- Read Wait State

  在最后一个地址/命令字节和第一个数据字节之前,SPI 主设备必须等待从站获取内部的数据, 后续字节读取自动预取, 不需再等待

  1. 在最后一个地址/命令字节和第一个数据字节之间,SPI 主设备需要等待最坏的内部读取时间

  2. 在最后一个地址/命令字节和第一个数据字节之间,SPI 主设备可以插入一个 0xFF 等待字节

- Read Termination

  在最后一个数据字节,SPI 主设备需要发送一个 0xFF 代表数据读取终止, 从站识别终止字节后, 就不会再预取内部数据。如果 SPI 主设备发送 0x00, 则代表从设备至少要预取一个字节的数据

4.5.2 ESC SPI Write Access

表 4.8 Write access for 2 and 4 Byte SPI Masters

Byte	2 Byte SPI Master	4 Byte SPI Master
0	A[12:5] address bit [12:5]	A[12:5] address bit [12:5]
1	A[4:0] address bit [4:0] CMD0[2:0] write command: 100b	A[4:0] address bit [4:0] CMD0[2:0] 3 byte addressing:110b
2	D0[7:0] data byte 0	A[15:13] address bit [15:13] CMD1[2:0] 3 byte addressing:110b res[1:0] two reserved bits, set to 00b

续上表

Byte	2 Byte SPI Master	4 Byte SPI Master
3	D1[7:0] data byte 1	A[15:13] address bit [15:13] CMD1[2:0] 3 byte addressing:100b res[1:0] two reserved bits, set to 00b
4	D2[7:0] data byte 2	D0[7:0] data byte 0
5	D3[7:0] data byte 3	D1[7:0] data byte 1
6	D4[7:0] data byte 4	D2[7:0] data byte 2
7	D5[7:0] data byte 5	D3[7:0] data byte 3

- Write Access

  在写访问的数据阶段,SPI 主设备向从站发送写入的数据, 在发送最后一个数据后, 需要将 SPI 片选释放, 至此 SPI 写访问终止

4.5.3 ESC Read/Write 通信时序

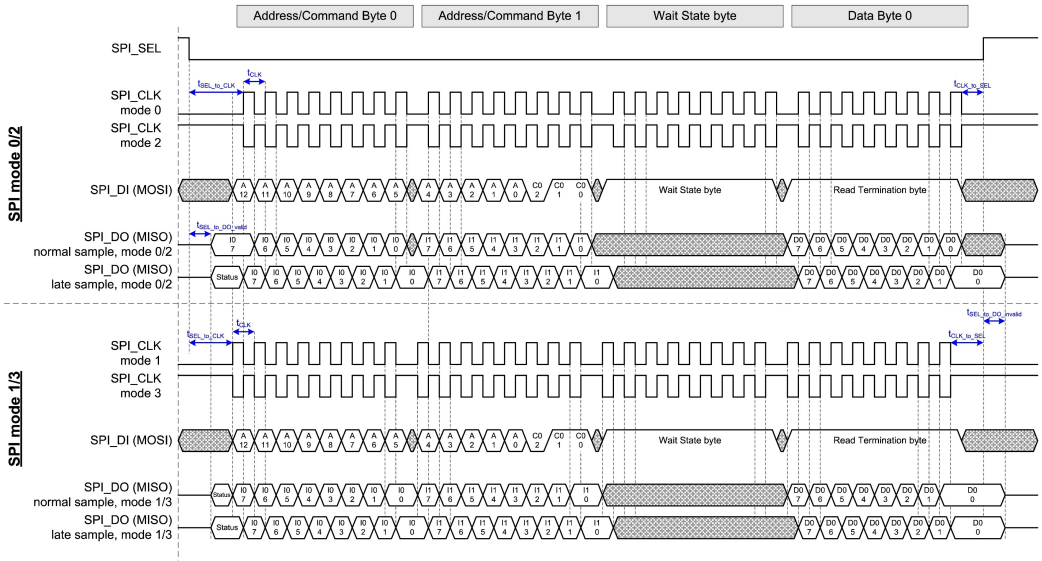


图 4.2 SPI read access (2 byte addressing,1 byte read data) with Wait State byte

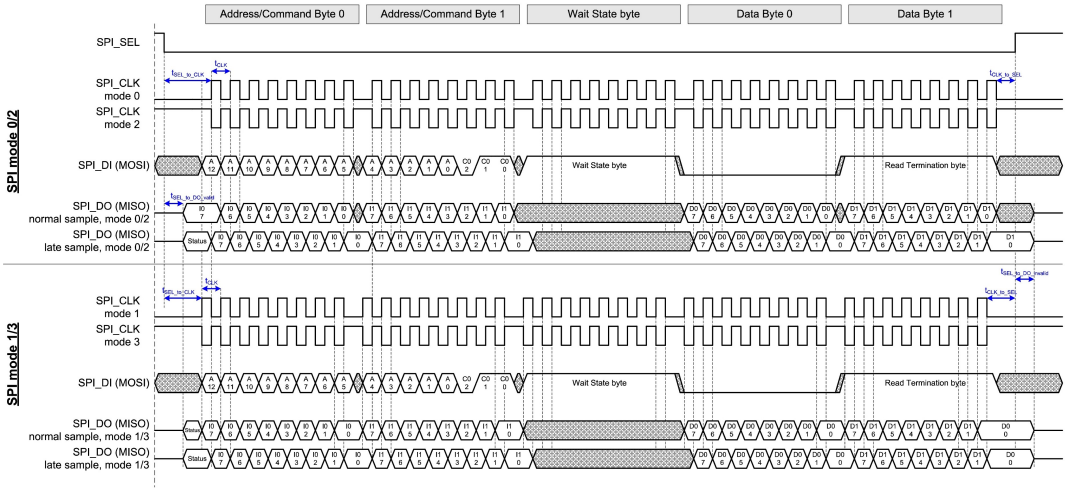


图 4.3　SPI read access (2 byte addressing,2 byte read data) with Wait State byte

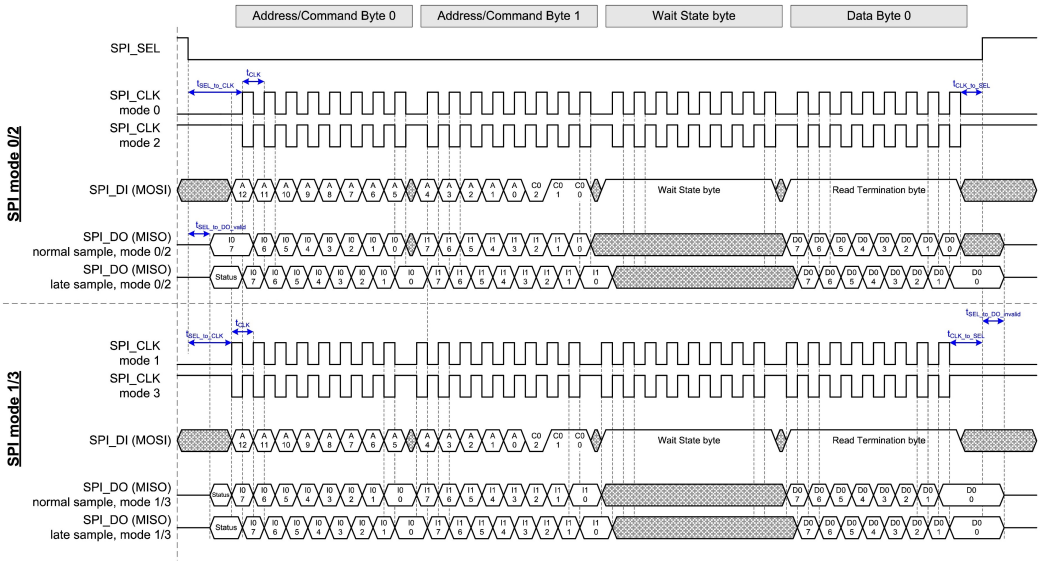


图 4.4　SPI write access (2byte addressing,1byte write data)

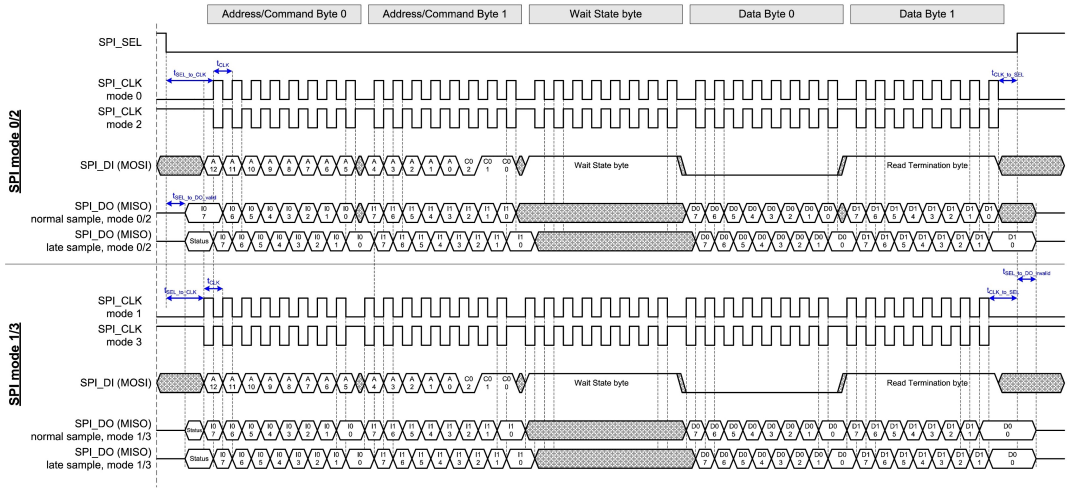


图 4.5　SPI write access (3byte addressing,1byte write data)

4.5.4　ESC 访问函数伪代码实现

伪代码实现 ESC 读写访问函数

```
/**
 * \brief send command and address
 */

static void __handle_setup_phase(uint16_t address, uint8_t command)
{
#define __ADDR_MODE_BYTE_MAX (4U)

    uint8_t tx_data[__ADDR_MODE_BYTE_MAX] = {0, 0, 0, 0}, tx_len = 0;

    tx_data[0] = address >> 5;
    tx_len++;
    /**
    * \brief check address is 2byte or 3byte mode
    */
    if((address + 8) & ~0x1fff)
    {
        tx_data[1] = (address << 3) + 0x06;
        tx_data[2] = ((address >> 8) & 0x0e) + (command << 2);
        tx_len += 2;
    }
    else
    {
        tx_data[1] = (address << 3) | command;
        tx_len++;
    }
    /**
    * \brief add wait state byte
```

```
    */
    if(__ESC_RWS == command)
    {
        tx_data[tx_len] = 0xff;
        tx_len++;
    }
    __g_esc_obj->pfn_spi_write(tx_data, tx_len);
}

void aw_ecat_esc_read(void *p_data, uint16_t address, uint16_t len)
{
    uint8_t tx_data = 0xFF;
    while(len-- > 0)
    {
        __g_esc_obj->pfn_spi_entry_critical();
        __g_esc_obj->pfn_spi_cs_select();
        /**
        * \brief send address and cmd
        */
        __handle_setup_phase(address,
#if /*if ESC is same as ET1100*/
                             __ESC_RWS
#else /*if ESC is same as ET1200*/
                             __ESC_RD
#endif
                             );
        /**
        * \brief read data one by one,
        * it can be interrupted by PDI_Isr/Syncx_Isr
        * for realtime
        */
        __g_esc_obj->pfn_spi_write_and_read(&tx_data, p_data, 1);
        __g_esc_obj->pfn_spi_cs_deselect();
        __g_esc_obj->pfn_spi_exit_critical();
        (uint8_t*)p_data++;
        address++;
    }
}

void aw_ecat_esc_read_isr(void *p_data, uint16_t address, uint16_t len)
{
    uint8_t tx_data = 0x00U;
    __g_esc_obj->pfn_spi_entry_critical();
    __g_esc_obj->pfn_spi_cs_select();
    /**
    * \brief send address and cmd
    */
    __handle_setup_phase(address,
#if /*if ESC is same as ET1100*/
```

```
                            __ESC_RWS
#else /*if ESC is same as ET1200*/
                            __ESC_RD
#endif
                            );
    /**
    * \brief read data
    */
    while(len-- > 1)
    {
        __g_esc_obj->pfn_spi_write_and_read(&tx_data, p_data, 1);
        (uint8_t*)p_data++;
    }
    /**
    * \brief send terminate data
    */
    tx_data = 0xFFU;
    __g_esc_obj->pfn_spi_write_and_read(&tx_data, (uint8_t*)p_data, 1);
    __g_esc_obj->pfn_spi_cs_deselect();
    __g_esc_obj->pfn_spi_exit_critical();
}

void aw_ecat_esc_write(void *p_data, uint16_t address, uint16_t len)
{
    while(len-- > 0)
    {
        __g_esc_obj->pfn_spi_entry_critical();
        __g_esc_obj->pfn_spi_cs_select();
        /**
        * \brief send address and cmd
        */
        __handle_setup_phase(address, __ESC_WR);
        __g_esc_obj->pfn_spi_write(p_data, 1);
        __g_esc_obj->pfn_spi_cs_deselect();
        __g_esc_obj->pfn_spi_exit_critical();
        (uint8_t*)p_data++;
        address++;
    }
}

void aw_ecat_esc_write_isr(void *p_data, uint16_t address, uint16_t len)
{
    __g_esc_obj->pfn_spi_entry_critical();
    __g_esc_obj->pfn_spi_cs_select();
    /**
    * \brief send address and cmd
    */
    __handle_setup_phase(address, __ESC_WR);
    __g_esc_obj->pfn_spi_write(p_data, len);
```

```
    __g_esc_obj->pfn_spi_cs_deselect();
    __g_esc_obj->pfn_spi_exit_critical();
}
```

诚信共赢，持续学习，客户为先，专业专注，只做第一

广州致远电子股份有限公司

更多详情请访问 www.zlg.cn

欢迎拨打全国服务热线 400-888-4005