User Manual

倍福从站移植手册

AWorksLP

V0.0.0 Date:2024/05/23

类别	内容
关键词	倍福，EtherCat，AWorks
摘要	倍福从站移植手册

修订历史

版本	日期	原因

目 录

1. demo 添加与使用

这里以 EPC6450-DP 平台为例 (EPC103-DP 采用 keil 和 stm32cubeIDE)。

准备资料：

- 1 AWorksLP HPM6450 General SDK[1]
- 2 EtherCAT_SSC_demo[2]

其中 EtherCAT_SSC_demo 的位置在 EPC6450-DP 评估板.zip 中，如下图所示：

图 1.1　文件来源

DPort 相关下载链接依照 AXPI 仓库中 readme[3] 所示。

EPC6450-DP 的例程是以 SDK 补丁例程的形式存在，在这里我们需要将例程添加到 SDK 中进行编译构建，步骤如下所示。

- 1. 下载并解压 EPC6450-DP 评估版.zip 附件文件;
- 2. 将压缩包内 EPC6450-DP 文件夹拷贝至 {SDK}/boards 目录下，若不存在 boards 目录，手动创建即可；
- 3. 将 EtherCAT_SSC_demo 例程解压至 {SDK} 目录下任意位置；
- 4. 按照通用例程操作方式编译运行例程即可。

EtherCAT_SSC_demo 目录结构：

[1] https://axpi.zlgcloud.com/#/Container/Product/ProductDetail?id=11
[2] https://axpi.zlgcloud.com/#/Container/Product/ProductDetail?id=93
[3] https://axpi.zlgcloud.com/#/Container/Product/ProductDetail?id=93

图 1.2　工程目录

src 内部由三部分：

图 1.3　src

adapter 目录中里面有五个文件，hw_access 则是为协议栈提供相关 ESC 访问函数的，其余的则是 hpm 平台相关驱动文件。

图 1.4　adapter

app 中包含了主函数，以及用户自定义的应用程序文件：

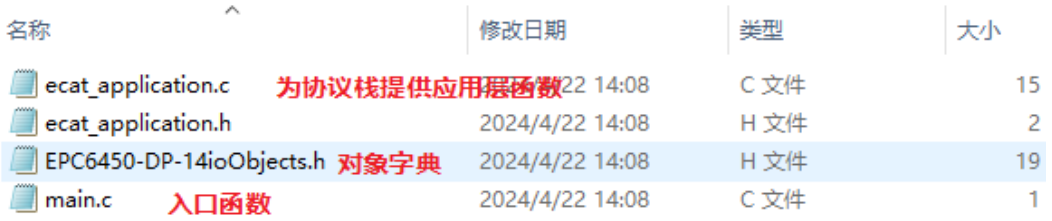


图 1.5　app

ssc_files 则是包含了倍福完整的协议栈代码：

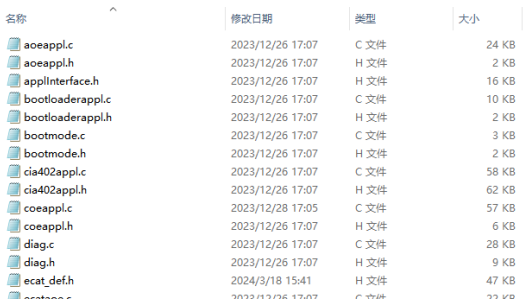


图 1.6 ssc_files

2. SSC 配置协议栈

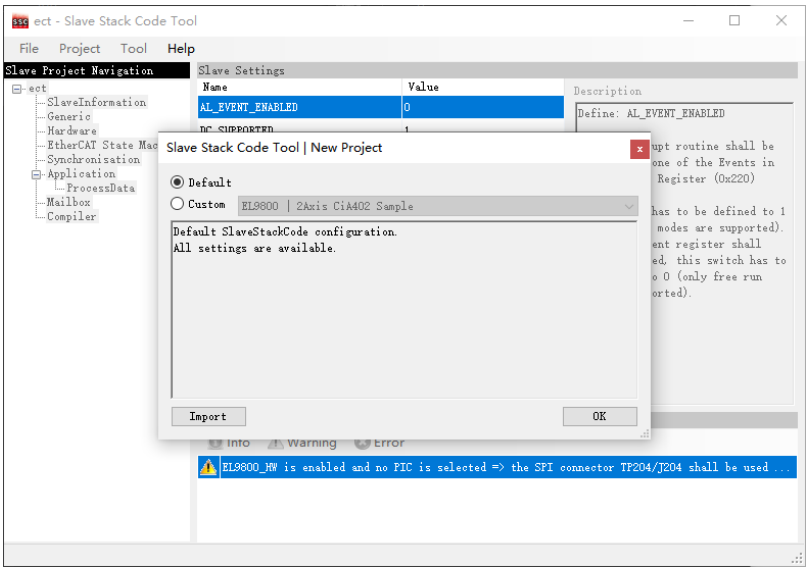
对于源码的使用我们有两种方式：- 1. 使用完整版的从站源码，通过配置 ecat_def.h 文件，对协议栈进行配置；- 2. 通过倍福提供的 Slave Stack Code Tool(SSC) 配置生成源码，对象字典以及 ESI 文件。

接下来会对这两种方式的使用进行详细的阐述。

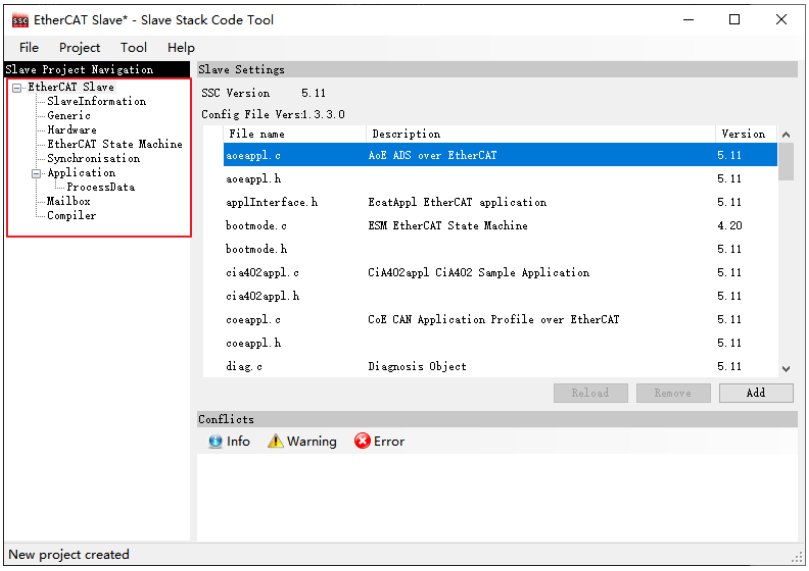
2.1 SSC 配置

SSC 是一个文件生成工具，除生成源码以外，还会生成描述从站信息的 ESI 文件，其中生成源码之前还需要对源码根据自己的需求进行配置。

在打开 SSC 工具后会出现如下界面，其中 Custom 是倍福基于自己的硬件做好的适配，与 defult 并无本质区别，这里选择 defult 即可。



可以看到左边的配置栏中有多个配置项：



接下来我们针对一些重要的配置分别解析其功能。

- **1.SlaveInfomation：**
  - **VENDOR_ID:** 供应商 id
  - **VENDOR_NAME:** 供应商名称
  - **PRODUCT_CODE:** 产品编码
- **2.Generic**
  - **EXPLICIT_DEVICE_ID:** 是否显示设备 ID
  - **ESC_SM_WD_SUPPORTED:** 是否使能硬件看门狗 (若不使能协议栈会使用软件来实现看门狗)，用于检测周期数据是否在更新
  - **ESC_EEPORM_ACCESS_SUPPORT:** 使能后从站可以通过 EEPROM 访问 ESC 内部的 eeprom。从站对 eeprom 的访问必须遵循以下原则：主站可以通过操作 ESC 的 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 提现，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
- **3.Hardware**
  - **EL9800_HW,MCI_HW,FC1100_HW,_PIC18,_PIC24：** 倍福自己硬件平台相关
  - **HW_ACCESS_FILES：** 适配头文件，需要提供相应的 ESC 访问函数供协议栈调用
  - **CONTROLLER_16BIT,CONTROLLER_32BIT:** 微控制器架构为 16 位还是 32 位
  - **ESC_16BIT_ACCESS,ESC_32BIT_ACCESS:** 微控制器访问 ESC 时支持 16 位还是 32 位
  - **MBX_16BIT_ACCESS:** 访问邮箱内存时，是否支持 16bit 访问，如不支持则默认 8bit 访问
  - **BIG_ENDIAN_FORMAT:** 指定大小端
  - **UC_SET_ECAT_LED:** 使能后需要为协议栈提供 HW_SetLed(uint8_t RunLed,uint8_t ErrorLed) 函数供协议栈调用，用于 led 展现协议栈状态
  - **ESC_CONFIG_DATA:** 参考值为：050E0344102700000000，其为 eeprom 前八个字节，在 ESC 上电或复位后自动加载到对应的寄存器
- **4.EtherCAT State Machine**
  - **BOOTSTRAPMODE_SUPPORTED:** 使能 FOE 在 BOOT 模式下更新固件的支持
  - **OP_PD_REQUIRED:** 在 Safe to Op 时，检查是否有周期数据进行传输，如果

没有接收到周期数据，则触发看门狗异常

- **PREOPTIMEOUT:** Init to PreOp/Boot 的状态转换超时时间
- **SAFEOP2OPTIMEOUT:** SafeOp to Op 的状态转换超时时间
- **CHECK_SM_PARAM_ALIGNMENT:** 检查同步管理器首地址和长度是否与 ESC 访问函数对齐

- **5.Synchronisation**
  - **AL_EVENT_ENABLED:** 事件触发是否采用中断模式，SM 模式必须
  - **DC_SUPPORTED:** 是否支持分布式时钟，DC 模式必须
  - **ECAT_TIMER_INT:** 采用定时器中断轮询 ECAT_CheckTimer()
- **6.Application** 例程相关的配置，根据需求选择

  ProcessData:
  - **MIN_PD_WRITE_ADDRESS:** 周期数据发送邮箱在 ESC 中的最小地址
  - **DEF_PD_WRITE_ADDRESS:** 周期数据发送邮箱在 ESC 中的默认地址
  - **MAX_PD_WRITE_ADDRESS:** 周期数据发送邮箱在 ESC 中的最大地址
  - **MIN_PD_READ_ADDRESS:** 周期数据接收邮箱在 ESC 中的最小地址
  - **DEF_PD_READ_ADDRESS:** 周期数据接收邮箱在 ESC 中的默认地址
  - **MAX_PD_READ_ADDRESS:** 周期数据接收邮箱在 ESC 中的最大地址
  - **MAX_PD_INPUT_SIZE:** INPUT 的最大字节数
  - **MAX_PD_OUTPUT_SIZE:** OUTPUT 的最大字节数
- **7.Mailbox**
  - **MAILBOX_QUEUE：** 是否使用队列存储邮箱服务
  - **AOE_SUPPORTED, COE_SUPPORTED, EOE_SUPPORTED, VOE_SUPPORTED, SOE_SUPPORTED, FOE_SUPPORTED:** 支持的服务
  - **COMPLETE_ACCESS_SUPPORTED:** 完全访问，使能后将会访问对应对象的所有 entries
  - **SEGMENTS_SDO_SUPPORTED:** 是否支持分段访问
  - **MAILBOX_SUPPORTED:** 邮箱支持
  - **MIN_MBX_SIZE:** 邮箱最小字节
  - **DEF_MBX_SIZE:** 邮箱默认字节
  - **MAX_MBX_SIZE:** 邮箱最大字节
  - **MIN_MBX_WRITE_ADDRESS:** 发送邮箱在 ESC 中的最小地址
  - **DEF_MBX_WRITE_ADDRESS:** 发送邮箱在 ESC 中的默认地址
  - **MAX_MBX_WRITE_ADDRESS:** 发送邮箱在 ESC 中的最大地址
  - **MIN_MBX_READ_ADDRESS:** 接收邮箱在 ESC 中的最小地址
  - **DEF_MBX_READ_ADDRESS:** 接收邮箱在 ESC 中的默认地址

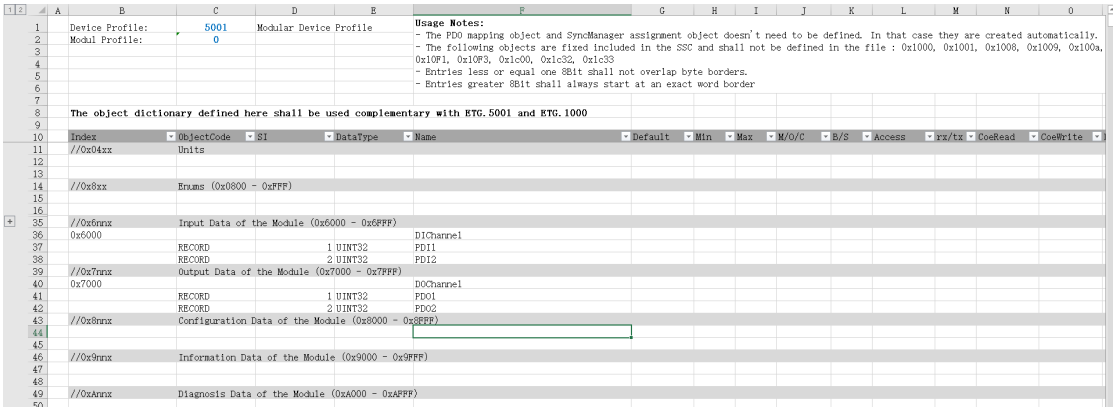
– **MAX_MBX_READ_ADDRESS:** 接收邮箱在 ESC 中的最大地址

- **8.Compiler** 编译器相关

2.2 生成源码

在配置完成后，需要添加自己的对象字典：

- 选择工具栏 tool->Application->create new 选项，创建 excel 编写对象字典，根据需求添加对象。图中为 PDI 和 PDO 映射添加了两个 UINT32 位的数据。



	A	B	C	D	E	F
1		Device Profile:	5001	Modular Device Profile		Usage Notes:
2		Modul Profile:	0			- The PDO mapping object and SyncManager assignment object doesn't need to be defined. In that case they are created automatically.
3						- The following objects are fixed included in the SSC and shall not be defined in the file : 0x1000, 0x1001, 0x1008, 0x1009, 0x100a,
4						0x10F1, 0x10F3, 0x1c00, 0x1c32, 0x1c33
5						- Entries less or equal one 8Bit shall not overlap byte borders.
6						- Entries greater 8Bit shall always start at an exact word border
7						
8		The object dictionary defined here shall be used complementary with ETG.5001 and ETG.1000				
9						
10		Index	ObjectCode	SI	DataType	Name
11		//0x04xx	Units			
12						
13						
14		//0x8xx	Enums (0x0800 - 0xFFF)			
15						
16						
35		//0x6nnx	Input Data of the Module (0x6000 - 0x6FFF)			
36		0x6000				DIChannel
37			RECORD		1 UINT32	PDI1
38			RECORD		2 UINT32	PDI2
39		//0x7nnx	Output Data of the Module (0x7000 - 0x7FFF)			
40		0x7000				DOChannel
41			RECORD		1 UINT32	PDO1
42			RECORD		2 UINT32	PDO2
43		//0x8nnx	Configuration Data of the Module (0x8000 - 0x8FFF)			
44						
45						
46		//0x9nnx	Information Data of the Module (0x9000 - 0x9FFF)			
47						
48						
49		//0xAnnx	Diagnosis Data of the Module (0xA000 - 0xAFFF)			
50						

	G	H	I	J	K	L	M	N	O
1									
2									
3									
4									
5									
6									
7									
8									
9									
10	Default	Min	Max	M/O/C	B/S	Access	rx/tx	CoeRead	CoeWrite
11									
12									
13									
14									
15									
16									
35									
36									
37									
38									
39									
40									
41									
42									
43									
44									
45									
46									
47									
48									
49									
50									

- 最后点击 project->create new Slave Files 生成代码及 ESI 文件。

2.3 完整版源码移植

1. 下载倍福 SSC 工具，同时会附带一份带宏定义的完整源码

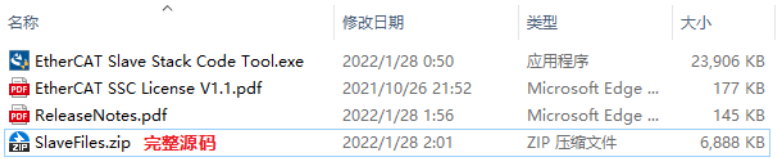


图 2.1 SlaveFiles

2. 将源码压缩包解压后添加到 ssc_files 目录下

3.ecat_def.h 配置文件，我们使用此文件进行全局的配置，此文件与 SSC 的配置作用是相同的。接下来我们对 ssc_files 里的文件进行一些修改：

- 1. 在 ecat_def.h 中全局搜索 USER_HW_ACCESS_FILE 和 USER_HW_ACCESS_FILE 两个宏，若不存在则添加 (用户可添加自己的接口文件)。

```
/*指定适配文件,
↪此文件的作用是为协议栈提供几个接口函数，需要用户实现(针对不同的硬件)*/
#define USER_HW_ACCESS_FILE        "hw_access.h"

/*此为用户自定义应用文件，会为协议栈提供应用函数的接口*/
#define USER_HW_APLICATION_FILE    "ecat_application.h"        //指定应用程序文件
```

注意

在不同版本的协议栈 USER_HW_ACCESS_FILE 和 USER_HW_APLICATION_FILE 可能不会存在，此时我们需要手动添加。

- 2. 在以下文件中添加包含文件：

ecatslv.h 中添加

```
------  Includes
------
ecatslv.h文件

#include "ecat_def.h"

#if EL9800_HW
#include "el9800hw.h"
#elif FC1100_HW
#include "fc1100hw.h"
#elif MCI_HW
#include "mcihw.h"
#else
#include  "esc.h"          添加如下
/*CODE_INSERT_START (HW_ACCESS_FILE)*/
#include USER_HW_ACCESS_FILE
/*CODE_INSERT_END*/
#endif

#ifndef _ECATSLV_H_
#define _ECATSLV_H_
```

图 2.2　ecatslv.h

在 coeappl.c，ecatappl.c，ecatslv.c 中添加

```
   coeappl.c
/*Add Application specific Objects*/
#if CiA402_DEVICE
#include "cia402appl.h"
#elif EL9800_APPLICATION
#include "el9800appl.h"
#elif SAMPLE_APPLICATION
#include "sampleappl.h"
#elif TEST_APPLICATION
#include "testappl.h"
#elif SAMPLE_APPLICATION_INTERFACE
#include "SampleApplicationInterface.h"
#else          添加如下
/*CODE_INSERT_START (APPLICATION_FILE)*/
#include USER_HW_APLICATION_FILE
/*CODE_INSERT_END*/
#endif
```

图 2.3　coeapp.c

ecatappl.c

```
#if CiA402_DEVICE
#include "cia402appl.h"
#elif EL9800_APPLICATION
#include "el9800appl.h"
#elif SAMPLE_APPLICATION
#include "sampleappl.h"
#elif TEST_APPLICATION
#include "testappl.h"
#elif SAMPLE_APPLICATION_INTERFACE
#include "SampleApplicationInterface.h"
#elif BOOTLOADER_SAMPLE
/*ECATCHANGE_START(V5.12) BOOT1*/
#include "bootloaderappl.h"
/*ECATCHANGE_END(V5.12) BOOT1*/
#else    添加如下
/*CODE_INSERT_START (APPLICATION_FILE)*/
#include USER_HW_APLICATION_FILE
/*CODE_INSERT_END*/
#endif
```

图 2.4　ecatappl.c

ecatslv.c

```
#if CiA402_DEVICE
#include "cia402appl.h"
#elif TEST_APPLICATION
#include "testappl.h"
#elif EL9800_APPLICATION
#include "el9800appl.h"
#elif SAMPLE_APPLICATION_INTERFACE
#include "SampleApplicationInterface.h"
#elif SAMPLE_APPLICATION
#include "sampleappl.h"
#elif BOOTLOADER_SAMPLE
/*ECATCHANGE_START(V5.12) BOOT1*/
#include "bootloaderappl.h"
/*ECATCHANGE_END(V5.12) BOOT1*/
#else    添加如下
/*CODE_INSERT_START (APPLICATION_FILE)*/
#include USER_HW_APLICATION_FILE
/*CODE_INSERT_END*/
#endif
```

图 2.5　ecatslv.c

- 3. 配置 ecat_def.h 文件：

  配置项的含义参照 SSC 中宏定义即可，ecat_def.h 文件的配置参考例程即可。

```
/*定义为1则支持同步模式，定义为0则仅支持freerun*/
#define AL_EVENT_ENABLED
/*从站是否支持分布时钟*/
#define DC_SUPPORTED
/*是否使用定时器中断，通常为1*/
#define ECAT_TIMER_INT
```

运行模式参考：

AL_EVENT_ENABLED	1	1	0	0
DC_SUPPORTED	1	0	1	0
模式	sm+dc	sm	dc	freerun

3. 软件适配

以下汇总了协议栈需要实现的函数接口，接下来我们会逐步解析实现：

原型	UINT8 HW_Init(void)
参数	void
返回	0：初始化成功 other: 初始化失败
说明	硬件的初始化
原型	UINT16 HW_GetALEventRegister(void)
参数	void
返回	返回 ALEvent register 的值 (0x220)
说明	获取事件寄存器 ALEvent register 的值
原型	UINT16 HW_GetALEventRegister_Isr(void)
参数	void
返回	返回 ALEvent register 的值 (0x220)
说明	在中断中访问 ESC，与 HW_GetALEventRegister 功能相同
原型	void HW_EscRead(MEM_ADDR *pData，UINT16 Address，UINT16 Len)
参数	MEM_ADDR *pData：读取出的数据
	UINT16 Address：ESC 寄存器地址
	UINT16 Len：读取长度
返回	void
说明	读取 ESC 寄存器和 DPRAM 区域
原型	void HW_EscReadIsr(MEM_ADDR *pData，UINT16 Address，UINT16 Len)
参数	MEM_ADDR *pData：读取出的数据
	UINT16 Address：ESC 寄存器地址
	UINT16 Len：读取长度
返回	void
说明	与 HW_EscRead 函数作用相同，用于中断中的访问
原型	void HW_EscWrite(MEM_ADDR *pData，UINT16 Address，UINT16 Len)
参数	MEM_ADDR *pData：写入的数据
	UINT16 Address：ESC 寄存器地址
	UINT16 Len：写入长度
返回	void
说明	将数据写入 ESC 寄存器和 DPRAM 区域

续上表

原型	void HW_EscWriteIsr(MEM_ADDR *pData，UINT16 Address，UINT16 Len)
参数	MEM_ADDR *pData：写入的数据
	UINT16 Address：ESC 寄存器地址
	UINT16 Len：写入长度
返回	void
说明	用于在中断中，将数据写入 ESC 寄存器和 DPRAM 区域
原型	UINT32 HW_GetTimer(void)
参数	void
返回	定时器计数值
说明	获取定时器当前计数值
原型	void HW_ClearTimer(void)
参数	void
返回	void
说明	清除定时器计数值
原型	void ENABLE_ESC_INT(void)
参数	void
返回	void
说明	使能 ESC 外部中断
原型	void DISABLE_ESC_INT(void)
参数	void
返回	void
说明	失能 ESC 外部中断

原型	void APPL_Application(void)
参数	void
返回	void
说明	在中断或是循环中，该函数会被周期调用
原型	void APPL_AckErrorInd(void)
参数	void
返回	void
说明	当从站错误应答时，调用此函数
原型	void APPL_StartMailboxHandler(void)

续上表

参数	void
返回	void
说明	在状态从 INIT 切换到 PREOP 或从 INIT 切换到 BOOT 期间调用该函数
原型	void APPL_StopMailboxHandler(void)
参数	void
返回	void
说明	在状态从 PREOP 切换到 INIT 或从 BOOT 切换到 INIT 期间调用该函数
原型	void APPL_StartInputHandler(void)
参数	void
返回	void
说明	在状态从 PREOP 切换到 SAFEOP 期间调用该函数
原型	void APPL_StopInputHandler(void)
参数	void
返回	void
说明	在状态从 SAFEOP 切换到 PREOP 期间调用该函数
原型	void APPL_StartOutputHandler(void)
参数	void
返回	void
说明	在状态从 SAFEOP 切换到 OP 期间调用该函数
原型	void APPL_StopOutputHandler(void)
参数	void
返回	void
说明	在状态从 OP 切换到 SAFEOP 期间调用该函数
原型	void APPL_GenerateMapping(void)
参数	void
返回	void
说明	在状态 PREOP 切换到 SAFEOP 时，调用此函数，用于计算对象字典映射的大小
原型	void APPL_InputMapping(UINT16 *pData)
参数	UINT16 *pData：输入数据的指针
返回	void
说明	在 input 动作时调用，将对象字典数据映射到 SM 缓存区中

续上表

原型	void APPL_OutputMapping(UINT16 *pData)
参数	UINT16 *pData：输出数据的指针
返回	void
说明	在 output 动作时调用，将 SM 缓存区中的数据映射到对象字典中

3.1 ESC 访问接口函数

3.1.1 实现原理

依据官方手册：

Table 58: 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 Extension (3 address/command bytes)
1	1	1	reserved

图 3.1 SPI command

根据手册，常用的命令大致可分为三类:

command	number	meaning
ESC_RD	0x02	read without wait state
ESC_RDWS	0x03	read with wait state
ESC_WR	0x04	write

每个命令对应着不同的通信规则，如图所示 spi 读通信分为 2 字节和 3 字节模式：

注:

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

ESC_RD:

Table 59: Address modes without (Read access without Wait state byte)

Byte	2 Byte address mode		3 Byte address mode	
0	A[12:5]	address bits [12:5]	A[12:5]	address bits [12:5]
1	A[4:0] CMD0[2:0]	address bits [4:0] read/write command	A[4:0] CMD0[2:0]	address bits [4:0] 3 byte addressing: 110b
2	D0[7:0]	data byte 0	A[15:13] CMD1[2:0] res[1:0]	address bits [15:13] read/write command two reserved bits, set to 00b
3	D1[7:0]	data byte 1	D0[7:0]	data byte 0
4 ff.	D2[7:0]	data byte 2	D1[7:0]	data byte 1

图 3.2　esc read without wait state byte

根据手册所示，整个数据帧可分为两个部分：头帧和数据帧。

头帧就是由地址和命令组成的 2byte 或 3byte

byte0	byte1
addr[12:5]	addr[4:0]cmd[2:0]

数据帧则是紧接着头帧所要发送的数据，对于 ESC 读来说，数据帧的作用是连续的读取：发送 0x00 的同时接收来自 ESC 回馈的数据，最后一个数据的读取以 0xff 结尾。

假设读取首地址为 addr 的连续的 n-2 个数据。

byte0	byte1	byte3	···	byteN
addr[12:5]	addr[4:0]cmd[2:0]	0x00	···	0xff

ESC_RWS:

Table 60: Address modes for Read access with Wait state byte

Byte	2 Byte address mode		3 Byte address mode	
0	A[12:5]	address bits [12:5]	A[12:5]	address bits [12:5]
1	A[4:0] CMD0[2:0]	address bits [4:0] read command: 011b	A[4:0] CMD0[2:0]	address bits [4:0] 3 byte addressing: 110b
2	0xFF	wait state byte	A[15:13] CMD1[2:0] res[1:0]	address bits [15:13] read command: 011b 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

图 3.3　esc read with wait state byte

如图所示，ESC_RWS 和 ESC_RD 大致相同，不同的是，在 ESC_RWS 头帧结尾需要多附带一个字节的数据 (0xff)。

byte0	byte1	byte3
addr[12:5]	addr[4:0]cmd[2:0]	0xff

数据帧则是与 ESC_RD 相同。

ESC_WR:

Table 62: Write access for 2 and 4 Byte SPI Masters

Byte	2 Byte SPI master		4 Byte SPI master	
0	A[12:5]	address bits [12:5]	A[12:5]	address bits [12:5]
1	A[4:0] CMD0[2:0]	address bits [4:0] write command: 100b	A[4:0] CMD0[2:0]	address bits [4:0] 3 byte addressing: 110b
2	D0[7:0]	data byte 0	A[15:13] CMD1[2:0] res[1:0]	address bits [15:13] 3 byte addressing: 110b two reserved bits, set to 00b
3	D1[7:0]	data byte 1	A[15:13] CMD2[2:0] res[1:0]	address bits [15:13] write command: 100b 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

图 3.4　esc write

ESC_WR 的头帧格式与 ESC_RD 相同，对于 ESC 写来说，数据帧就是实际要写入的数据：紧接着头帧，将数帧据按字节发送。

假设在地址为 addr 连续写入 n-2 个数据：

byte0	byte1	byte2	···	byteN
addr[12:5]	addr[4:0]cmd[2:0]	data[0]	···	data[n]

3.1.2　代码实现

以 EPC103-DP 为例：

配置 spi 为：

- Data Size:8 bit
- First Bit:MSB Fisrt
- Clock Polarity:Low
- Clock Phase:1 Edge

注意

- spi 的模式取决于 configData 字段的对应寄存器值，前文推荐的 configData 值 (050E0344102700000000) 就对应以上配置。
- DPort 对应的 spi 标准最大速率为 20Mhz，配置时需要注意。

```
/*spi读写函数*/
#define SPI_WRITE_AND_READ(TX_BUFFER,RX_BUFFER,Size)        HAL_SPI_
↪TransmitReceive(&hspi1, TX_BUFFER, RX_BUFFER,Size, HAL_MAX_DELAY)
/*cs片选引脚拉低*/
#define SELECT_SPI                                          HAL_GPIO_WritePin(CS_
↪GPIO_Port, CS_Pin, GPIO_PIN_RESET)
```

```
/*cs片选引脚拉高*/
#define DESELECT_SPI                                    HAL_GPIO_WritePin(CS_
↪GPIO_Port, CS_Pin, GPIO_PIN_SET)
/*全局中断使能与失能*/
#define ENABLE_GLOBAL_INT                               __enable_irq()
#define DISABLE_GLOBAL_INT                              __disable_irq()
```

```
static uint8_t ESC_ADDR (uint16_t addr, uint8_t cmd, uint8_t* buffer)
{
    uint8_t head_len = 0;

    buffer[0] = addr >> 5;
    head_len += 1;

    if ((addr + 8) & ~0x1fff) {
    /*ESC访问时，最小单元是8bit，设想此时有个场景address首地址为0x1fff - 1时，
    此时地址范围在0~8k可以使用2byte寻址，但因为最小操作单元为8bit，
    此时0x1fff之后的地址范围则是在8k到64k之间只能使用3byte的方式，
    为了解决这个过渡的问题，所以在此时会选择3byte的寻址方式*/
        buffer[1] = (addr << 3) + ESC_ADR;
        head_len += 1;

        buffer[2] = ((addr >> 8) & 0xe0) + (cmd << 2);
        head_len += 1;
    } else {
        buffer[1] = (addr << 3) | cmd;
        head_len += 1;
    }

    if (cmd == ESC_WAIT_RD) {
        buffer[head_len] = 0xff;
        head_len +=1;
    }

    return head_len;
}

void HW_EscRead (MEM_ADDR *pData, uint16_t Address, uint16_t Len)
{
    uint8_t  head_len;
    uint16_t i = Len;
    uint8_t* temp_pData = (uint8_t *)pData;
    uint8_t TX_BUFFER[5];/*Head_len:max is 4,and a topbyte */
    uint8_t RX_BUFFER[5];

    while (i-->0){
        head_len = ESC_ADDR(Address,ESC_RWS,TX_BUFFER);
        TX_BUFFER[head_len] = 0xff;
        DISABLE_GLOBAL_INT;
```

```
        SELECT_SPI;
        SPI_WRITE_AND_READ(TX_BUFFER,RX_BUFFER,head_len+1);
        DESELECT_SPI;
        ENABLE_GLOBAL_INT;
        *temp_pData = RX_BUFFER[head_len];
        Address++;
        temp_pData++;
    }
}

void HW_EscReadIsr (MEM_ADDR *pData, uint16_t Address, uint16_t Len)
{
    uint8_t head_len;
    uint8_t i=0;
    uint8_t TX_BUFFER[Len + 4];
    uint8_t RX_BUFFER[Len + 4];
    memset(TX_BUFFER, 0, Len + 4);
    head_len = ESC_ADDR(Address, ESC_RWS, TX_BUFFER);
    SELECT_SPI;
    SPI_WRITE_AND_READ(TX_BUFFER,RX_BUFFER, head_len + Len);
    DESELECT_SPI;
    memcpy(pData,&RX_BUFFER[head_len],Len);
}

void HW_EscWrite (MEM_ADDR* pData, uint16_t Address, uint16_t Len)
{
    uint8_t head_len;
    uint16_t i = Len;
    uint8_t* pTmpData = (uint8_t *)pData;
    uint8_t TX_BUFFER[4];
    uint8_t RX_BUFFER[4];
    while(i-- > 0){
        head_len = ESC_ADDR(Address,ESC_WR,TX_BUFFER);
        TX_BUFFER[head_len] = *pTmpData;
        DISABLE_GLOBAL_INT;
        SELECT_SPI;
        SPI_WRITE_AND_READ(TX_BUFFER,RX_BUFFER,head_len + 1);
        DESELECT_SPI;
        ENABLE_GLOBAL_INT;
        Address++;
        pTmpData++;
    }
}

void HW_EscWriteIsr (MEM_ADDR *pData, uint16_t Address, uint16_t Len)
{
    uint8_t head_len;
    uint8_t i=0;
    uint8_t TX_BUFFER[Len + 3];
```

```
    uint8_t RX_BUFFER[Len + 3];
    head_len = ESC_ADDR(Address,ESC_WR,TX_BUFFER);
    memcpy(&TX_BUFFER[head_len],pData,Len);
    SELECT_SPI;
    SPI_WRITE_AND_READ(TX_BUFFER,RX_BUFFER, head_len + Len);
    DESELECT_SPI;
}
/*读ESC：0x220事件寄存器*/
uint16_t HW_GetALEventRegister (void)
{
    HW_EscRead((MEM_ADDR *)&EscALEvent.Word, 0x220, 2);
    return EscALEvent.Word;
}

uint16_t HW_GetALEventRegister_Isr (void)
{
    HW_EscReadIsr((MEM_ADDR *)&EscALEvent.Word, 0x220, 2);
    return EscALEvent.Word;
}
```

3.2 ecat_def.h 配置相关

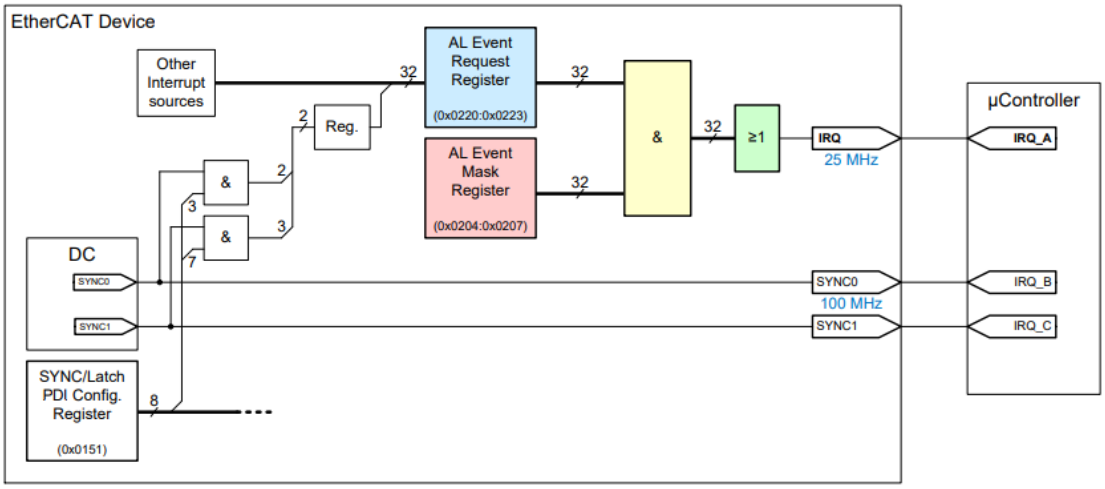
3.2.1 PDI 中断

当事件发生时，事件寄存器 AL EVENT Request Registers 的对应位会被置 1，SSC 协议栈中更新事件的实质就是读取此寄存器的值。可编程的系统中断由 ESC 内部生成，可配置为通过 IRQ 中断输出到主机的外部中断。

注意

注意 PDI 中断的事件有多种，包括数据同步，状态转换等。

根据手册所示：



当事件寄存器 (0x220) 和中断屏蔽器 (0x204) 同时活跃时，就会产生一次电平信号。中

断屏蔽器是由协议栈控制，在对应的阶段开启。

3.2.2 DC-SYNC

DC-SYNC 信号是周期性触发，由主站配置其触发周期:

9.2.3.6 SyncSignal Initialization Example

The SyncSignal generation is initialized with the following procedure:

1. Enable DC SYNC Out Unit in ESC Configuration register (0x0141[2]=1; specific ESCs only)
2. Set SYNC/Latch PDI Configuration register (0x0151, initialized by SII EEPROM) to SYNC0/1 output with appropriate driver settings.
3. Set Pulse Length register (0x0982:0x0983, initialized by EEPROM) to pulse length of SYNC signals. Select a value > 0 ns for cyclic repetition of the SyncSignals

XML

4. Assign Sync Unit to ECAT or PDI (0x0980, part of ESI)
5. Set cycle time of SYNC0 signal (0x09A0:0x09A3) and for SYNC1 signal (0x09A4:0x09A7)
6. Set Start Time of Cyclic Operation (0x0990:0x0997) to a time later than the time the cyclic generation will be activated (end of activation frame; e.g., read the System Time and add the time for writing Start Time and Activation). For 32 bit DCs, the SyncSignal generation will start at worst after a turn-over of the System Time (~ 4 s), but with 64 bit DCs, SyncSignal generation may start in hundreds of years.
7. Activate Cyclic Operation (0x0981[0]=1) to start cyclic generation of SyncSignals and activate SYNC0/1 generation (0x0981[2:1]=0x3). The Sync Unit waits until the Start Time of Cyclic Operation is reached for the generation of the first SYNC0 pulse.

主站配置

Register Start Time of Cyclic Operation and register Next SYNC1 pulse can be read to get the time of the next output event. In the acknowledged modes, the Sync0/1 Status registers (0x098E:0x098F) give the status of the SyncSignals. The SyncSignals are acknowledged by reading the SYNC0/1 Status registers.

如图 DC-SYNC 的配置需要 ESI 文件和主站共同完成：

ESI 文件中 ConfigureData 代表着 eeprom 中的前八个字，在上电时，会自动将值加载到指定的寄存器中。其中包括 sync 的使能，输出信号配置等。

Table 40: ESC Configuration Area

Word Address	Parameter	Description	Register Address
0x0	PDI Control/ ESC Configuration	Initialization value for PDI Control register (EEPROM ADR 0x0000[9] is also mapped to register 0x0110[2])	0x0140:0x0141
0x1	PDI Configuration	Initialization value for PDI Configuration register	0x0150:0x0151
0x2	Pulse Length of SYNC Signals	Initialization value for Pulse Length of SYNC Signals register	0x0982:0x0983
0x3	Extended PDI Configuration	Initialization value for extended PDI Configuration register	0x0152:0x0153
0x4	Configured Station Alias	Initialization value for Configured Station Alias Address register	0x0012:0x0013
0x5	Reserved	Reserved, shall be zero	-
0x6	Reserved	Reserved, shall be zero	-
0x7	Checksum	Low byte contains remainder of division of word 0 to word 6 as unsigned number divided by the polynomial x^8+x^2+x+1(initial value 0xFF). NOTE: For debugging purposes it is possible to disable the checksum validation with a checksum value of 0x88A4. Never use this for production!	-

接下来我们拆解一下，configData 字段的作用：

PDI Control

2.25 PDI Control (0x0140)

Table 27: Register PDI Control (0x0140)

		ESC20	ET1100	ET1200	IP Core
Bit	**Description**	**ECAT**	**PDI**	**Reset Value**	
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 (port 3) 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	r/-	r/-	IP Core: Depends on configuration Others: 0, later EEPROM word 0	

配置 PDI 接口 (Process Data Interface),ESC 提供两种类型的接口：- 直接 IO 信号接口，无需应用层微处理器；- DPRAM 数据接口，使用外部微处理器访问，支持并行和串行两种方式。

当程序复杂时，我们就需要微处理器来进行应用层处理，DPort 引出了 spi 通信引脚，所以我们使用 spi 通信时，此寄存器的值应当填充为 0x05(SPI Slave)

ESC Configuration

2.26 ESC Configuration (0x0141)

Table 28: Register ESC Configuration (0x0141)

		ESC20 ~~[7:1]~~	ET1100 ~~[7:4]~~	ET1200 ~~[7:2]~~	IP Core [7:4] V2.2.0/ V2.02a
Bit	**Description**	**ECAT**	**PDI**	**Reset Value**	
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	r/-	r/-	IP Core: 1 with Digital I/O PDI, PDI EMULATION pin with µC/On-chip bus Others: 0, later EEPROM word 0	
1	Enhanced Link detection all ports: 0: disabled (if bits [7:4]=0) 1: enabled at all ports (overrides bits [7:4])	r/-	r/-	1, later EEPROM word 0	
2	Distributed Clocks SYNC Out Unit: 0: disabled (power saving) 1: enabled	r/-	r/-	IP Core: Depends on configuration Others: 0, later EEPROM word 0	
3	Distributed Clocks Latch In Unit: 0: disabled (power saving) 1: enabled	r/-	r/-		
4	Enhanced Link port 0: 0: disabled (if bit 1=0) 1: enabled	r/-	r/-	1, later EEPROM word 0	
5	Enhanced Link port 1: 0: disabled (if bit 1=0) 1: enabled	r/-	r/-		
6	Enhanced Link port 2: 0: disabled (if bit 1=0) 1: enabled	r/-	r/-		
7	Enhanced Link port 3: 0: disabled (if bit 1=0) 1: enabled	r/-	r/-		

NOTE: EEPROM values of bits 1, 4, 5, 6, and 7 are only transferred into this register at first EEPROM load after power-on or reset.

- **bit[0] 是选择 AL 状态寄存器的更新方式，从站有两种方式完成状态的转换：**
  - 主站请求状态转换 (写状态到 AL 控制寄存器 0x120)，从站检测到有状态转换请求，遂开启对应状态机的检查，检查通过后表明此时从站配置无误，将对应状态码写入 AL 状态寄存器中完成转换；
  - AL 状态寄存器的值会自动随 AL 控制寄存器变化，不必再通过从站 PDI 写入。
- bit[1] 当端口没有被正确连接的时候，端口内部会自动将 TX 的数据转发至 RX，此举会将 EtherCAT 帧发送回主站；
- bit[2] 是使能 DC-SYNC 信号输出；
- bit[3] 输入信号锁存。

寄存器填充为：0x0E

PDI Configuration

2.28.2 PDI SPI Slave Configuration

Table 33: Register PDI SPI Slave Configuration (0x0150)

		ESC20 [3:2], [7:6]	ET1100 [7:6]	ET1200 [7:6]	IP Core [7:6]
Bit	**Description**	**ECAT**	**PDI**	**Reset Value**	
1:0	SPI mode: 00: SPI mode 0 01: SPI mode 1 10: SPI mode 2 11: SPI mode 3 NOTE: SPI mode 3 is recommended for Slave Sample Code NOTE: SPI status flag is not available in SPI modes 0 and 2 with normal data out sample.	r/-	r/-	IP Core: Depends on configuration Others: 0, later EEPROM word 1	
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)	r/-	r/-		
4	SPI_SEL polarity: 0: Active low 1: Active high	r/-	r/-		
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)	r/-	r/-		
7:6	Reserved, set EEPROM value to 0	r/-	r/-		

根据 0x140 寄存器的不同模式，此寄存器的配置含义也有所不同，在 SPI Slave 模式下，此寄存器就是配置 spi 的通信模式，以及 SPI 中断的信号模式，根据我们前文所配置的中断以及 spi 的通信很容易就可以推断出此寄存器的值。

寄存器填充为：0x03

PDI Configuration

2.28.7 Sync/Latch Configuration

Table 43: Register Sync/Latch Configuration (0x0151)

		ESC20	ET1100	ET1200	IP Core
Bit	**Description**	**ECAT**	**PDI**	**Reset Value**	
1:0	SYNC0 output driver/polarity: 00: Push-Pull active low 01: Open Drain (active low) 10: Push-Pull active high 11: Open Source (active high)	r/-	r/-	IP Core: 10 Others: 00, later EEPROM word 1	
2	SYNC0/LATCH0 configuration*: 0: LATCH0 Input 1: SYNC0 Output	r/-	r/-	IP Core: 1 Others: 0, later EEPROM word 1	
3	SYNC0 mapped to AL Event Request register 0x0220[2]: 0: Disabled 1: Enabled	r/-	r/-	IP Core: Depends on configuration Others: 0, later EEPROM word 1	
5:4	SYNC1 output driver/polarity: 00: Push-Pull active low 01: Open Drain (active low) 10: Push-Pull active high 11: Open Source (active high)	r/-	r/-	IP Core: 10 Others: 00, later EEPROM word 1	
6	SYNC1/LATCH1 configuration*: 0: LATCH1 input 1: SYNC1 output	r/-	r/-	IP Core: 1 Others: 0, later EEPROM word 1	
7	SYNC1 mapped to AL Event Request register 0x0220[3]: 0: Disabled 1: Enabled	r/-	r/-	IP Core: Depends on configuration Others: 0, later EEPROM word 1	

* The IP Core has concurrent SYNC[1:0] outputs and LATCH[1:0] inputs, independent of this configuration.

- bit[1:0]/bit[5:4]: 表示 sync0/sync1 输出信号的有效电平；
- bit[2]/bit[6]: 选择 sync0/sync1 信号输出；
- bit[3]/bit[7]：将中断映射到 PDI 上，此时 sync0/sync1 信号将通过 PDI 引脚输出而不是 sync0/sync1 引脚；

寄存器填充为：0x44

Pulse Length of SYNC Signals

Table 111: Register Pulse Length of SyncSignals (0x0982:0x983)

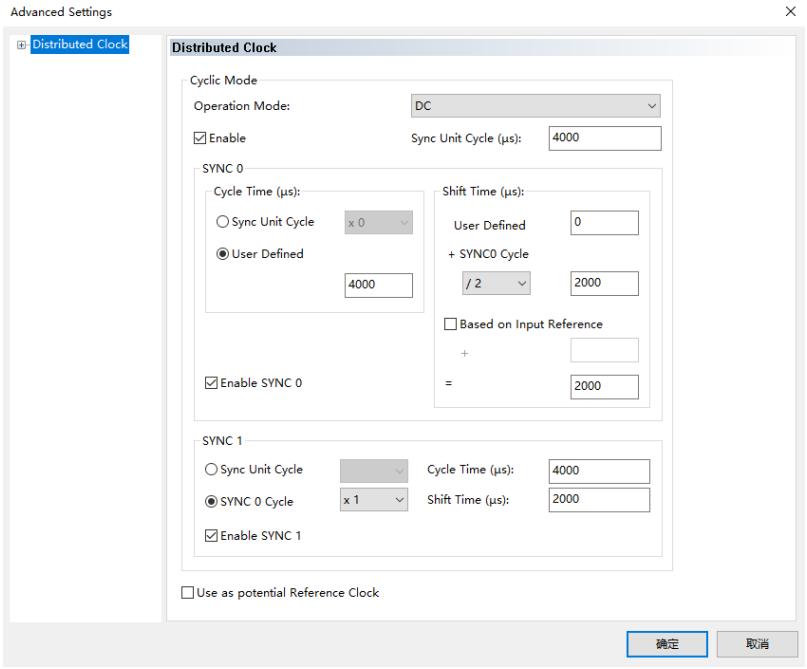
		ESC20	ET1100	ET1200	IP Core
Bit	**Description**	**ECAT**	**PDI**	**Reset Value**	
15:0	Pulse length of SyncSignals (in Units of 10ns) 0: Acknowledge mode: SyncSignal will be cleared by reading SYNC[1:0] Status register	r/-	r/-	IP Core: Depends on configuration Others: 0, later EEPROM word 2	

此寄存器位 sync 外部信号的脉冲宽度，太短会使得微控制器无法捕捉。

寄存器填充为：0x1027

注: 参考 ESI ConfigureData:050E03441027000000000000000

- 主站对 DC-SYNC 的配置通常不需要关注寄存器，以 TwinCAT 为例，我们只需填入所需的周期和偏移时间即可。



DC-SYNC 信号的长度与周期会组合形成不同的模式。

Table 35: SyncSignal Generation Mode Selection

Pulse Length of SYNC Signals (0x0982:0x0983)	SYNC0 Cycle Time (0x09A0:0x09A3)	
	> 0	= 0
> 0	Cyclic Generation	Single Shot
= 0	Cyclic Acknowledge	Single Shot Acknowledge

具体模式请参考手册，通常我们只使用 Cyclic Generation 模式

3.2.3 AL_EVENT_ENABLED/DC_SUPPORTED

AL_EVENT_ENABLED：使能后需要准备一个外部中断引脚对接 ESC 的 PDI 引脚，中断配置为：默认高电平，下降沿触发。

DC_SUPPORTED：使能后需要准备两个个外部中断引脚，分别对接 ESC 的 sync0 和 sync1 引脚，中断配置为：默认高电平，下降沿触发。

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    #if AL_EVENT_ENABLED
    /*PDI_Isr: AL_EVENT_ENABLED使能后，协议栈提供*/
        if(GPIO_Pin == PDI_Pin){
            extern void PDI_Isr(void);

            PDI_Isr();
        }
    #endif
    #if DC_SUPPORTED
    /*Sync0_Isr,Sync1_Isr:DC_SUPPORTED使能后，协议栈提供*/
        else if(GPIO_Pin == SYNC0_Pin){
            extern void Sync0_Isr(void);
```

```
            Sync0_Isr();
        }else if(GPIO_Pin == SYNC1_Pin){
            extern void Sync1_Isr(void);
            Sync1_Isr();
        }
    #endif
}
```

3.2.4 ECAT_TIMER_INT

ECAT_TIMER_INT：使能后需要准备一个 1ms 的定时器中断周期运行 ECAT_CheckTimer() 函数。

```
/*ECAT_CheckTimer()由协议栈提供*/
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    if(htim == &htim3){
        DISABLE_ESC_INT();
        ECAT_CheckTimer();
        ENABLE_ESC_INT();
    }
}
```

ECAT_TIMER_INT：不使能此宏，则需要实现以下函数接口。

```
/*获取定时器当前计数值*/
UINT32 HW_GetTimer(void)
/*清除定时器计数值*/
void HW_ClearTimer(void)
```

3.2.5 UC_SET_ECAT_LED

使能此宏后，需要向协议栈提供一个 run 和 error 的状态指示函数。

```
/*led默认低电平点亮*/
void HW_SetLed(uint8_t RunLed,uint8_t ErrorLed){
    HAL_GPIO_WritePin(RUN_GPIO_Port,RUN_Pin,!RunLed);
    HAL_GPIO_WritePin(ERR_GPIO_Port,ERR_Pin,!ErrorLed);
}
```

3.3 应用层接口函数

在使用 SSC 生成源码时，会附带生成一份应用文件，用户可以修改部分应用文件来满足实际工程所需。

```
/*将对象字典映射到要发给主站的数据中去*/
extern TOBJ6000 PDIChannel0x6000;
void APPL_InputMapping(UINT16* pData)
{
    /*将字典中的数据映射到PDI*/
    memcpy(pData,&(PDIChannel0x6000.PIN_485A),15);
```

```
}
/*将从站接收数据映射到对象字典上*/
extern TOBJ7000 PDOChannel0x7000;
void APPL_OutputMapping(UINT16* pData)
{
    /*将PDO的数据映射到字典*/
    memcpy(&(PDOChannel0x7000.PIN_485B),pData,15);
}
/*用户自定义app函数,此函数会随周期数据周期性调用*/
void APPL_Application(void)
{
    .......
}

/*以下函数是非必要重写*/

/*错误状态被主站应答时调用*/
void APPL_AckErrorInd(unsigned short stateTrans);
/*状态从Init to PreOp时调用*/
unsigned short APPL_StartMailboxHandler(void);
/*当状态从PreOp降阶时调用*/
unsigned short APPL_StopMailboxHandler(void);
/*当状态从PreOp tro SafeOp时调用*/
unsigned short APPL_StartInputHandler(unsigned short *pIntMask);
/*当状态从SafeOp降阶时调用*/
unsigned short APPL_StopInputHandler(void);
/*状态从SafeOp to OP时调用*/
unsigned short APPL_StartOutputHandler(void);
/*当状态从OP降阶时调用*/
unsigned short APPL_StopOutputHandler(void);
/*计算周期数据大小，使用其生成默认的函数即可*/
unsigned short APPL_GenerateMapping(unsigned short *pInputSize,unsigned short␣
↪*pOutputSize);
```

4. 常见问题及解决方法

4.1 ‘init to preop’ timeout

出现这个问题时，极大可能是 spi 的读写函数：HW_EscRead HW_EscWrite HW_EscReadIsr HW_EscWriteIsr 等适配不成功，此时检查一下代码运行是否卡在了 HW_Init() 函数中的如下位置：

```
uint8_t HW_Init(void)
{
  extern void app_gpio_init(void);
  app_gpio_init();
  exit_intr_init();
  spi_init();
  uint32_t intMask;
  do
  {
    intMask = 0x93;
    HW_EscWriteDWord(intMask, ESC_AL_EVENTMASK_OFFSET);
    intMask = 0;
    HW_EscReadDWord(intMask, ESC_AL_EVENTMASK_OFFSET);
  } while (intMask != 0x93);

  intMask = 0x00;
  timer_1ms_init();
  return 0;
}
```

图 4.1 PDI 读写测试

此处是使用 PDI 接口对 ESC 进行一次读写测试，如果读写出现问题，可使用逻辑分析仪参考如下波形：

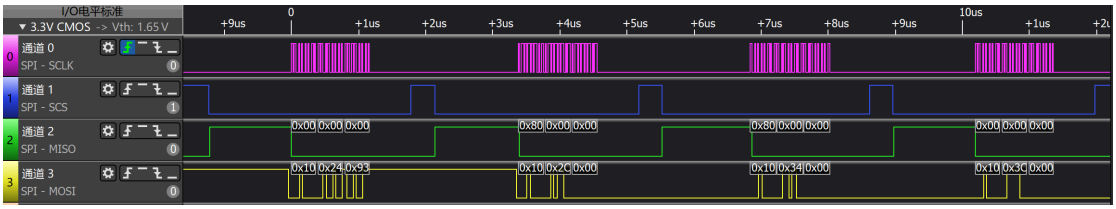


图 4.2 向 ESC 的寄存器写 0x93

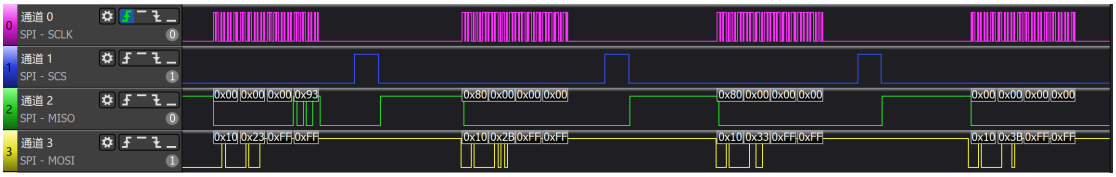


图 4.3 向 ESC 的寄存器读 0x93

4.2 ‘preop to safeop’ SM OUT(IN) cfg Invaild

1：此时是 TwinCAT 检测到 ESC 中 eeprom 中的周期数据的对象字典配置与 MCU 中的不同，此时需要重新烧录对应的 ESI 文件，并重新扫描。

2：检查对象字典的大小是否超过宏 MAX_PD_INPUT_SIZE 和 MAX_PD_OUTPUT_SIZE 定义的上限。

4.3 ERROR CODE : 0x1a

同步错误：

- 1 检查 TwinCAT 是否运行在 run 模式；
- 2 检查 dc 参数是否设置正确；
- 3 twinCAT 对网卡以及 CPU 有严格要求需要使用官方要求的主机配置；
- 4 检查 ESI 文件中的 configdata 字段是否为：<Config-Data>050E0344102700000000</ConfigData>。

可以通过观测 spi init 信号与 sync 信号的相位判断是否是主站发帧抖动的问题，在 DPort 资料汇总中提供了示例波形。

4.4 扫描多个从站出现 type 一样的情况

每个从站的 ESI 文件中的 product_code 不能相同。

4.5 从站一直处于复位状态

检查 ESC 的复位引脚是否一直处于低电平

4.6 ‘safeop to op’ timeout

safeop 到 op 的转换失败，但并未出现错误码，此时需要检查适配层的功能函数定时器中断，spi 读写等。

4.7 eeprom 的访问

主站可以通过操作 ESC 的 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 提现，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

SSC 源码中带有操作 eeprom 的函数，使能‘ESC_EEPROM_ACCESS_SUPPORT’后，便可调用。

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