摘 要

CANopen 是一种架构在 CAN（控制局域网路, Controller Area Network）上的高层通讯协定，包括通讯子协定及设备子协定，常在嵌入式系统中使用，也是工业控制常用到的一种现场总线。CanFestival 提供了将 CANOpen 运行起来的源码以及字典编辑器。

本文档以 CS32F103 为例，详细介绍将 CanFestival 协议栈移植到 CS32F103 中，并实现基本的心跳、主从 SDO 读写、主从 PDO 通信等功能。同时给出了基于裸机和基于 RTOS(FreeRTOS)的示例代码，用户可以参考并轻松移植到自己的芯片里。

本文档介绍的移植方法和例程，同时适用于所有芯海的 32 位带有 CAN 外设的 MCU。

适用范围

类型	适用产品型号或系列	说明
MCU	CS32F103 以及其他芯海带有 CAN 外设的 MCU	

本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

版 本

历史版本	修改内容	日期
V1.0	初版生成	

芯海科技CHIPSEA

本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

目 录

摘 要 1
1. CANOpen 基本概述 4
2. Canfestival 基本介绍 6
3. Canfestival 裸机移植到芯海 32 位 MCU 7
3.1 Canfestival 源码的下载 7
3.2 对象字典编辑器的使用 7
3.3 基于芯海 32 位 MCU 移植 Canfestival 9
3.3.1 文件添加和对象字典编辑 9
3.3.1 特定接口的移植与实现 14
4. Canfestival 移植到芯海 32 位 MCU-基于 FreeRTOS 18
5. 基于 Canfestival 的 demo 解析 23
5.1 对象字典文件的解析 23
5.2 主程序的解析 25

本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

1. CANOpen 基本概述

CAN 全称为 Controller Area Network，即控制器局域网，由德国 Bosch 公司最先提出，是国际上应用最广泛的现场总线之一。

CAN 组网就必须得要应用层协议，原因就在于：

* 便于网络管理与控制；
* 确认数据的收发；
* 发送大于 8 个字节的数据块（CAN 每帧数据传输大小为 8 字节）；
* 为不同节点分配不同的报文标识符；
* 定义帧报文的内容及含义（最主要的原因）；
* 网络的监控，节点故障的诊断与标识；

CAN 上层协议有许多，用大家都公认的，便于产品的兼容，因此， CANopen 成为备选项。

CANopen 是一种架构在控制局域网路（Controller Area Network, CAN）上的高层通讯协定，包括通讯子协定及设备子协定常在嵌入式系统中使用，也是工业控制常用到的一种现场总线。由非营利组织 CiA（CAN in Automation）进行标准的起草及审核工作，基本的 CANopen 设备及通讯子协定定义在 CAN in Automation (CiA) draft standard 301 中。针对个别设备的子协定以 CiA 301 为基础再进行扩充。如针对 I/O 模组的 CiA401 及针对运动控制的 CiA402。

CANopen 实现了 OSI 模型中的网络层以上（包括网络层）的协定。CANopen 标准包括寻址方案、数个小的通讯子协定及由设备子协定所定义的应用层。 CANopen 支持网络管理、设备监控及节点间的通讯，其中包括一个简易的传输层，可处理资料的分段传送及其组合。一般而言数据链结层及物理层会用 CAN 来实作。除了 CANopen 外，也有其他的通讯协定（如 EtherCAT）实作 CANopen 的设备子协定。

本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

图 1 通信协议分层模型

简言之，CAN 是物理层和数据链路层，CANOpen 是调用 CAN 进行通行的应用层。

CANopen 主要的通信方式和报文包括 NMT(网络管理)、同步帧、时间戳、紧急报文、PDO(过程数据对象)、SDO(同步数据对象)、错误控制。其中，最主要的主从机间的数据交互方式为 SDO 和 PDO 通信。

关于 CANopen 协议的具体标准，协议规则与协议内容，较为复杂，本应用笔记侧重于基于芯海 32 位 MCU 对 CANopen 协议栈的移植与使用，在此不对 CANopen 协议作具体说明，用户可参考相关 CANopen 官方文档。

表 1　使用环境说明

类别	名称	下载路径
MCU	CS32F103	/
开发板	CS32F103 LQFP48 开发板 V1.0	/
下载工具	JLINK	/
SDK	ChipSea.CS32F1xx_DFP.2.0.5	/
编译工具	Keil uVision V5.27.1.0	/

本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

2. Canfestival 基本介绍

CanFestival 提供了将 CANOpen 运行起来的源码以及字典编辑器。可以理解为 canopen 是一个协议，而 canfestival 是将 canopen 运行起来的协议栈，该协议栈开源，并且提供一个简单的字典编辑器的脚本。

Canfestival 协议栈具有很多版本，不同版本之间源码可能会有区别，字典编辑器的版本需要与字典文件和源码对应。提供最新版，版本号为 CanFestival-3-de1fc3261f21。该文件中包含了 canfestival 的字典编辑器、RTT 的 CANOpen 修复版本以及编辑器运行需要的软件。

本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA
芯海科技CHIPSEA

3. Canfestival 裸机移植到芯海 32 位 MCU

3.1 Canfestival 源码的下载

Canfestival 源码地址：https://hg.beremiz.org/canfestival。

最新版本号为 CanFestival-3-de1fc3261f21。

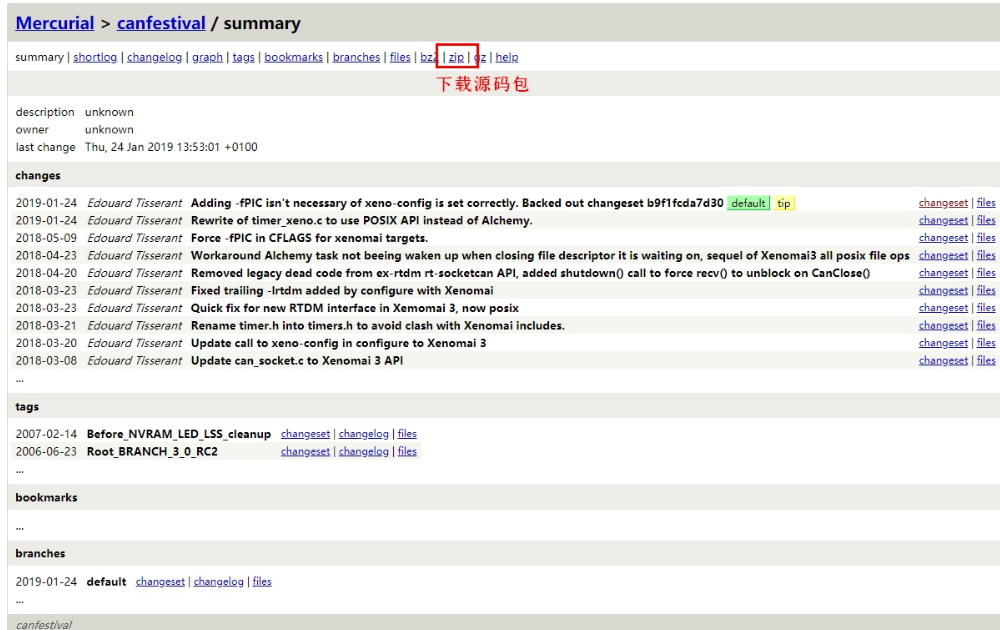

图 2 Canfestival 源码下载

3.2 对象字典编辑器的使用

对象字典本质是一个结构体，亦可以理解为一个数据库，其中存放了所有通过 CANOpen 协议的数据。内部根据 CANOpen 协议预设了许多参数变量，例如节点 ID、节点状态、波特率，这些变量可在对象字典编辑器中修改、添加或者删除。对每一个 CANOpen 设备来说，对象字典是其数据储存的容器，正如名字一样，该结构体储存的数据可以通过 SDO 读写索引位置数据。主索引为两个字节，0x0000~0xffff。部分索引如果需要存放多个数据，便有了子索引，子索引为一个字节。

本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

:::::	:::::	:::::	:::::	:::::	:::::
11FFh		reserved			
Server SDO parameter					
1200h	RECORD	1st SDO server parameter	SDO PARAM (22h)	ro	0
1201h	RECORD	2nd SDO server parameter	SDO PARAM (22h)	rw	M/O**
:::::	:::::	:::::	:::::	:::::	:::::
127Fh	RECORD	128th SDO server parameter	SDO PARAM (22h)	rw	M/O**
Client SDO parameter					
1280h	RECORD	1st SDO client parameter	SDO PARAM (22h)	rw	M/O**
1281h	RECORD	2nd SDO client parameter	SDO PARAM (22h)	rw	M/O**
:::::	:::::	:::::	:::::	:::::	:::::
12FFh	RECORD	128th SDO client parameter	SDO PARAM (22h)	rw	M/O**
1300h		reserved			
:::::	:::::	:::::	:::::	:::::	:::::

图 3 CANOpen 主索引

0x00	Number of Entries	UNSIGNED8
0x01	Vendor ID	UNSIGNED32
0x02	Product Code	UNSIGNED32
0x03	Revision Number	UNSIGNED32
0x04	Serial Number	UNSIGNED32

图 4 CANOpen 子索引

每个设备中都有一个对象字典，对象字典可以用一个文件描述，它叫做 EDS（Electronic data sheet），它采用了 INI 文件格式。在 canopen 协议栈里实际起作用的是一对 c/h 文件。c/h 文件内容，一般基于 EDS 的内容通过对象字典编辑器来生成。CanFestival 提供了一个基于 python wxPython 图形库的脚本来编辑对象字典。当然，也有很多功能更为强大的 CANopen 对象字典编辑器，如 Vector 的 CANeds，在此仅以简单的字典编辑器 objdictedit.py 来例来作说明。

首先，

1. 需要下载安装 Python2.7；
2. 下载安装 python 图形库， wxPython2.8-win32-unicode-2.8.12.1-py27.exe。下载地址：http://www.wxpython.org/download.php，或使用 pip 命令行安装；
3. 下载安装 Gnosis_Utils-1.2.0.win32-py24.exe。下载地址：http://gnosis.cx/download/Gnosis_Utils.More/；
4. 从 .\CanFestival3\objectgen 目录下打开脚本 objectedit.py；

最新的版本号为 CanFestival-3-de1fc3261f21 的代码包，解压后，Gnosis 已在目录下，只需要另外安装 Python2.7 和 wxPython2.8 库即可。路径为：CanFestival-3-de1fc3261f21\objdictgen。

本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

名称	修改日期	类型
gnosis	2023/2/8 16:18	文件夹
Gnosis_Utils-1.2.2	2023/2/8 16:18	文件夹
i18n	2023/2/8 16:18	文件夹
locale	2023/2/8 16:18	文件夹
bug_report_Tue_Feb__2_23-56-42_202...	2021/2/2 23:56	TXT 文件
canfestival_config.py.in	2019/1/24 20:53	IN 文件
commondialogs.py	2019/1/24 20:53	Python File
commondialogs.pyc	2021/1/6 16:32	Compiled Pytho...
eds_utils.py	2019/1/24 20:53	Python File
eds_utils.pyc	2021/1/6 16:32	Compiled Pytho...
gen_cfile.py	2019/1/24 20:53	Python File
gen_cfile.pyc	2021/1/6 16:32	Compiled Pytho...
Gnosis_Utils-current.tar.gz	2019/1/24 20:53	360压缩
Makefile.in	2019/1/24 20:53	IN 文件
networkedit.ico	2019/1/24 20:53	图标
networkedit.png	2019/1/24 20:53	PNG 文件
networkedit.py	2019/1/24 20:53	Python File
networkeditortemplate.py	2019/1/24 20:53	Python File
node.py	2019/1/24 20:53	Python File
node.pyc	2021/1/6 16:32	Compiled Pytho...
nodeeditortemplate.py	2019/1/24 20:53	Python File
nodeeditortemplate.pyc	2021/1/6 16:32	Compiled Pytho...
nodelist.py	2019/1/24 20:53	Python File
nodemanager.py	2019/1/24 20:53	Python File
nodemanager.pyc	2021/1/6 16:26	Compiled Pytho...
objdictedit.py	2019/1/24 20:53	Python File
objdictgen.py	2019/1/24 20:53	Python File
subindextable.py	2019/1/24 20:53	Python File
subindextable.pyc	2021/1/6 16:32	Compiled Pytho...

图 5 字典编辑器脚本

3.3 基于芯海 32 位 MCU 移植 Canfestival

3.3.1 文件添加和对象字典编辑

我们移植需要用到的源文件在 CanFestival-3-de1fc3261f21\src 目录下，头文件在 CanFestival-3-de1fc3261f21\include 目录下。

将 CanFestival-3-de1fc3261f21\src 目录下的 dcf.c、emcy.c、lifegrd.c、lss.c、nmtMaster.c、nmtSlave.c、objacces.c、pdo.c、sdo.c、states.c、sync.c、timer.c 共 12 个文件加入到工程中，将 CanFestival-3-de1fc3261f21\include 目录下的所有.h 文件共 19 个文件全部加入到工程里（或将头文件路径加到工程中），另外，还需要一些与协议栈功能和配置相关的文件，即 applicfg.h、canfestival.h、config.h、timerscfg.h。这些文件可以从 CanFestival-3-de1fc3261f21\include\AVR 目录下拷贝，后续再根据自己的芯片类型和移植情况进行修改。

本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

名称	修改日期	类型
applicfg.h	2019/1/24 20:53	H 文件
can_AVR.h	2019/1/24 20:53	H 文件
can_drv.h	2019/1/24 20:53	H 文件
canfestival.h	2019/1/24 20:53	H 文件
config.h	2019/1/24 20:53	H 文件
iar.h	2019/1/24 20:53	H 文件
timerscfg.h	2019/1/24 20:53	H 文件

图 6 移植所需头文件

最后，需要 CANopen 的设备字典文件。在 canopen 协议栈里实际上是一对 c/h 文件。文件内容，一般基于 EDS 的内容通过对象字典编辑器来生成。CanFestival 提供了一个基于 python wxPython 图形库的脚本来编辑对象字典。当然，也有很多功能更为强大的 CAN 对象字典编辑器，在此以 objdictedit.py 来例来作说明。此处，将 CS32F103 作从机，设备中，拥有一个 SDO client 索引，接收主机发来的数据请求和其他指令，同时拥有一个 TPDO，在特定条件下，向 CAnopen 主机发送 Temperture 的值。

首先，新建一个从节点，名称为“Slave_f103”，网络管理模式，可选节点护卫和心跳，在心跳协议中，CANopen 节点定期发送心跳消息，该消息使 CANopen 主设备或心跳使用者知道该节点仍处于活动状态。如果心跳消息在一定时间内未到达，则主机可以采取特定措施。此类操作可能是重置节点或向操作员报告错误。心跳消息由 CAN-ID 0x700 +节点 ID 标识，其中第一个数据字节等于 1110。

在节点保护协议中，CANopen 主站轮询从站节点的当前状态信息。如果节点在特定时间段内未响应，则主服务器将认为该节点已死，并将采取措施。

心跳协议是首选方法，因为它的开销比节点保护少。

本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

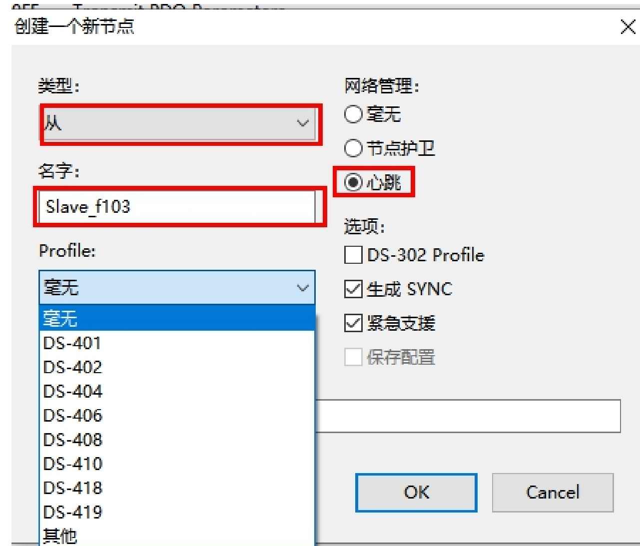

图 7 新建节点

图中 DS-4xx 文件协议属于行业特定协议，如 401 为伺服电机控制协议，若节点需要控制伺服电机，即可选择该协议。字典中有厂商自定义区域，不同行业在该区域订立标准，形成上述的多种协议。

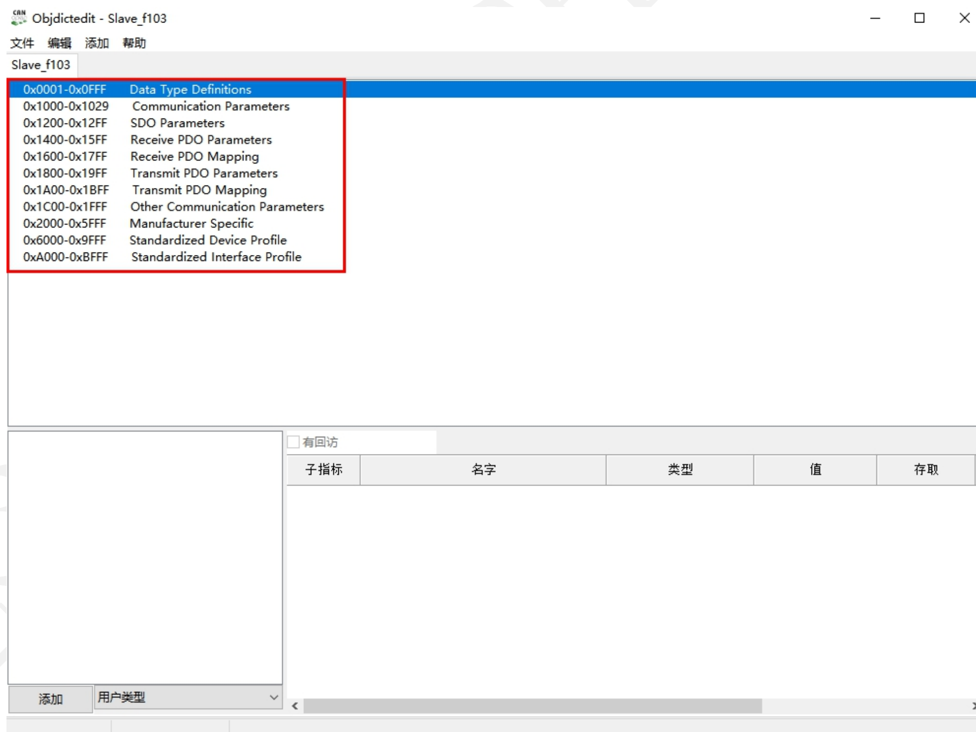

图 8 主索引

上图所列，为可编辑的所有对象字典索引。

0x0001-0x0fff 区域在该字典编辑器中，为保留区域，无法编辑。实际上 0x1-0x25F 为数据类型定义索引，在一些商业的字典编辑器中是可以编辑的，这里不作过多介绍。

本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

0x1000-0x1029 区域为从站节点的一些基本属性，例如心跳时间，硬件/软件版本号等等。有些对象字典索引（例如设备类型（1000h））是强制性的，而另一些则是可选的，例如制造商软件版本（100Ah）。强制性索引的集合表示最小对象字典，这是标记符合 CANopen 的设备所必需的。此应用笔记，仅为介绍 CANopen 协议栈移植示意，因此只编辑配置一些索引项目，比如心跳时间，其余值，不予配置或保持默认。这里，配置心跳时间为 0x1F4，单位 ms。即心跳时间 500ms。此处的心跳值，一般在后续，会通过主机发来的 SDO 进行修改。

0x1000 Device Type
0x1001 Error Register
0x1017 Producer Heartbeat Time
0x1018 Identity

☑有回访

子指标	名字	类型	值	存取	保存
0x00	Producer Heartbeat Time	UNSIGNED16	0x01F4	读 / 写	否

图 9 心跳值

0x1200-0x12ff 区域为 SDO 数据块，存放 SDO 主从的参数。在此添加或者删除 SDO 通道。SDO 类型为服务 SDO，其中客户端到服务端的 COB-ID 为节点号+0x600, 服务端到客户端的 COB-ID 为节点号+0x580。

0x1200 Server SDO Parameter

☐有回访

子指标	名字	类型	值	存取	保存
0x00	Number of Entries	UNSIGNED8	2	只读	否
0x01	COB ID Client to Server (Receive SDO)	UNSIGNED32	$NODEID+0x601	只读	否
0x02	COB ID Server to Client (Transmit SDO)	UNSIGNED32	$NODEID+0x581	只读	否

图 10　服务 SDO

0x1400-0x15ff 区域为 RPDO 基础参数区域，即特质化某个 RPDO 的属性。其中包括了传输类型、生产禁止时间、开始传输的 SYNC 值等等。每个 RPDO 根据实际需要设置参数。在这里只是开启了 RPDO 的传输通道，并且配置了某个 RPDO 是如何工作的，但是没有涉及到接收到的数据应该存放在何处或者变量名称。RPDO 的配置，需要配合 CANopen 来配置。其中，子索引 0x01 的 COB-ID 与主机的对应的 TPDO COB-ID 应保持一致。

0x1400 Receive PDO 1 Parameter

☐有回访

子指标	名字	类型	值	存取	保存
0x00	Highest SubIndex Supported	UNSIGNED8	6	只读	否
0x01	COB ID used by PDO	UNSIGNED32	$NODEID+0x181	读 / 写	否
0x02	Transmission Type	UNSIGNED8	0x00	读 / 写	否
0x03	Inhibit Time	UNSIGNED16	0x0000	读 / 写	否
0x04	Compatibility Entry	UNSIGNED8	0x00	读 / 写	否
0x05	Event Timer	UNSIGNED16	0x0000	读 / 写	否
0x06	SYNC start value	UNSIGNED8	0x00	读 / 写	否

图 11　接收 PDO 通信参数

0x1600-0x17ff 区域即为 RPDO 的映射区域。上面提到 RPDO 的数据接收存放问题即在

本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

此编辑。每添加或者删除一个 RPDO，字典编辑器会在此自动添加或者删除一个 PDO 映射。简言之，当我们在上一步骤中添加或者删除一个 RPDO，字典编辑器会在此区域自动添加或者删除一个 RPDO 映射。每个映射中数据的个数、值需要我们自己编辑。subindex 即为子索引，箭头所指的数字 8 是该 RPDO 接收的数据个数，最下方箭头所指内容即为需要接收的数据。此应用笔记 Demo 中，CS32F103 实现了一个 TPDO，此处 RPDO 没有编辑。

0x1600 Receive PDO 1 Mapping

☐有回访

子指标	名字	类型	值	存取	保存
0x00	Number of Entries	UNSIGNED8	8	读/写	否
0x01	PDO 1 Mapping for an application object 1	UNSIGNED32	毫无	读/写	否
0x02	PDO 1 Mapping for an application object 2	UNSIGNED32	毫无	读/写	否
0x03	PDO 1 Mapping for an application object 3	UNSIGNED32	毫无	读/写	否
0x04	PDO 1 Mapping for an application object 4	UNSIGNED32	毫无	读/写	否
0x05	PDO 1 Mapping for an application object 5	UNSIGNED32	毫无	读/写	否
0x06	PDO 1 Mapping for an application object 6	UNSIGNED32	毫无	读/写	否
0x07	PDO 1 Mapping for an application object 7	UNSIGNED32	毫无	读/写	否
0x08	PDO 1 Mapping for an application object 8	UNSIGNED32	毫无	读/写	否

图 12　接收 PDO 映射参数

0x1800-0x19ff 区域结构与 0x1400-0x15ff 相似，只是该区域设置的是 TPDO 的参数。此时基本保持默认值，后续再具体修改该对象字典生成的 c 文件。

0x1800 Transmit PDO 1 Parameter

后续修改字典c文件

☑有回访

子指标	名字	类型	值	存取	保存
0x00	Highest SubIndex Supported	UNSIGNED8	6	只读	否
0x01	COB ID used by PDO	UNSIGNED32	$NODEID+0x281	读/写	否
0x02	Transmission Type	UNSIGNED8	0x00	读/写	否
0x03	Inhibit Time	UNSIGNED16	0x0000	读/写	否
0x04	Compatibility Entry	UNSIGNED8	0x00	读/写	否
0x05	Event Timer	UNSIGNED16	0x0000	读/写	否
0x06	SYNC start value	UNSIGNED8	0x00	读/写	否

图 13　发送 PDO 通信参数

0x1a00-0x1bff 结构与 0x1600-0x17ff 相似，该区域设置的是 TPDO 的映射。这里编辑了一个温度值，通过索引 0x1800 所编辑的特定发送方式和触发条件，将温度值，通过 TPDO 方式发给 CANopen 主机。

0x1A00 Transmit PDO 1 Mapping

☐有回访

子指标	名字	类型	值	存取
0x00	Number of Entries	UNSIGNED8	1	读/写
0x01	PDO 1 Mapping for a process data variable 1	UNSIGNED32	Temperture 1 (0x2000)	读/写

图 14　发送 PDO 映射参数

0x2000-0x5fff 区域即为自定义传输的地图变量。该编辑器提供了三种数据类型，一是单一变量，二是数组，三是多变量集合。变量在字典文件中定义为全局变量。添加单一变量或数组变量。数组成员的 value 值可以预设，也可以在程序中赋值。添加数据之后，在 RPDO

本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

或者 TPDO 的 mapping 中即可选择需要传输的数据。注意：如果想要通过 PDO 传输数据，那两个节点的 RPDO 与 TPDO 应该对应，即接收方的 RPDO 接收的数据与发送方 TPDO 地图（Mapping）中映射的的数据以及数据类型都应一一对应，协议栈会自动将接收到的数据一一对应并赋值，无需接收后手动分解报文解码。

0x6000-0xbfff 为行业自定义区域，例如伺服电机的 DS401 在该区域定义了许多变量用以存放数据，例如线圈数，最大转速，当前位置等等。这些都是由组织统一规范制定的。该应用笔记这里没有涉及。

最后，选择文件，导出 Slave_f103.c/Slave_f103.h 文件。并将 c/h 文件加入到工程中。

至此，协议栈移植所需要的所有 c/h 文件，均已获得。

3.3.1 特定接口的移植与实现

一些底层的接口函数需要根据芯片来实现。例如，硬件定时器的初始化，CAN 的底层收发驱动等。

函数 void setTimer(TIMEVAL value)主要被源码用来定时的，时间到了就需要调用一下函数 TimeDispatch()，函数 TIMEVAL getElapsedTime(void)主要被源码用来查询距离下一个定时触发还有多少时间，函数 unsigned char canSend(CAN_PORT notused, Message *m)主要被源码用来发一个 CAN 包的，需要调用外设 CAN 驱动来将一个 CAN 包发出去。

这里，以 CS32F103 为例，给出这些特定接口的实现代码：

代码 1 用户需实现的接口示例代码

```
// 被协议栈调用，底层 CAN 发送接口
unsigned char canSend(CAN_PORT notused, Message *m)
{
   uint8_t mailbox;
   uint16_t i = 0;
   can_tx_msg_t tx_msg;

   tx_msg.std_id = m->cob_id;
   tx_msg.ext_id = m->cob_id;
   tx_msg.id_select = CAN_ID_STANDARD;
   tx_msg.remote_txreq = (m->rtr == CAN_RMOTE_TX_REQ_DATA) ? 0:2;
   tx_msg.data_length = m->len;

   memcpy(tx_msg.data, m->data, m->len);

   mailbox = can_message_send(CAN1, &tx_msg);
   i = 0;  //i used for timeout
   while(can_tx_status_get(CAN1,mailbox) == 0x01)
   {
      i++;
      if(i == 0XFFF)
      {
```


本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

```
        }
    }
    return 0;
}

// 被协议栈调用，初始化一个硬件定时器，协议栈将以此为时基，并生成多个软件定时器
void setTimer(TIMEVAL value)
{
    uint32_t timer = __TIM_COUNTER_GET(TIM4);       // Copy the value of the running timer
    elapsed_time += timer - last_counter_val;
    last_counter_val = 65535-value;
    __TIM_COUNTER_SET(TIM4, 65535-value);
    __TIM_ENABLE(TIM4);
}


// 被协议栈用来查询距离下一个定时触发还有多少时间
TIMEVAL getElapsedTime(void)
{
    uint32_t timer = __TIM_COUNTER_GET(TIM4);       // Copy the value of the running timer
    if(timer < last_counter_val)
        timer += 65535;
    TIMEVAL elapsed = timer - last_counter_val + elapsed_time;
    //printf("elapsed %lu - %lu %lu %lu\r\n", elapsed, timer, last_counter_val, elapsed_time);
    return elapsed;
}

// 返回 0 即可，指定 CAN 主从通信的速率
UNS8 canChangeBaudRate(CAN_PORT port, char* baud)
{
    return 0;
}
```

另外，需要用户提供一个和初始化一个硬件定时器，协议栈将以此为时基，并生成多个软件定时器。并在定时中断中调用 TimeDispatch() 。下面给出示例，以 TIM4 为例：

代码 2 用户需实现的定时器相关示例代码

```
void tim4_config(uint16_t arr, uint16_t psc)
{
    tim_base_t ptr_time = {0};
    nvic_init_t ptr_nvic;

    __RCU_APB1_CLK_ENABLE(RCU_APB1_PERI_TIM4);

    /* Time base configuration */
    ptr_time.clk_div = TIM_CLK_DIV1;
    ptr_time.cnt_mode = TIM_CNT_MODE_UP;
    ptr_time.period = arr;
    ptr_time.pre_div = psc;
    tim_base_init(TIM4, &ptr_time);

    __TIM_INTR_ENABLE(TIM4, TIM_INTR_UPDATE);

    /* Configure and enable TIM4 interrupt. */
    ptr_nvic.nvic_irqchannel = IRQn_TIM4;
```


本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

```
    ptr_nvic.nvic_irq_enable = ENABLE;
    ptr_nvic.nvic_irq_pre_priority = 0;
    ptr_nvic.nvic_irq_sub_priority = 0;
    nvic_init(&ptr_nvic);

    __TIM_COUNTER_SET(TIM4, 0);
    __TIM_ENABLE(TIM4);
}

// 被协议栈调用，初始化一个硬件定时器，协议栈将以此为时基，并生成多个软件定时器
void initTimer(void)
{
    /* TIM4 Init 1ms */
    tim4_config(10, 7200-1); //72000000hz/7200=1 000 0hz  心跳 max 可为 6553.5ms

    // this is needed for correct canfestival virtual timer management start
    SetAlarm(NULL, 0, start_callback, 0, 0);
}

extern TIMEVAL last_counter_val;
extern TIMEVAL elapsed_time;

// 协议栈将 TIM4 中断为时基，定期调用 TimeDispatch();
void TIM4_IRQHandler(void)
{
#if OS_ENABLE
    uint32_t status_value = 0;
    status_value = taskENTER_CRITICAL_FROM_ISR();
#endif
    if(__TIM_INTR_STATUS_GET(TIM4, UPDATE))
    {
        last_counter_val = 0;
        elapsed_time = 0;
        __TIM_FLAG_CLEAR(TIM4, TIM_FLAG_UPDATE);

        /* canfestival dispatch */
        TimeDispatch();
    }
#if OS_ENABLE
    taskEXIT_CRITICAL_FROM_ISR(status_value);
#endif
}
```

同时，还需要根据硬件定时器的配置，相应修改 timecfg.h 文件。CS32F103 的 TIME4 是计数寄存器为 16 位，因此修改 TIMEVAL_MAX 宏为 0xFFFF。之前的代码 tim4_config(10, 7200-1)中，将计数值配置为 10，分频值为 7200，即 TIM4 的计数频率为 1000Hz，因此，修改 MS_TO_TIMEVAL(ms) 为((ms) * 10)，US_TO_TIMEVAL(us) 为((us) / 100)。此时心跳值最大可达 65535ms。

本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

代码 3 timecfg.h 文件示例代码

```
#ifndef __TIMERSCFG_H__
#define __TIMERSCFG_H__

// Whatever your microcontroller, the timer wont work if
// TIMEVAL is not at least on 32 bits
#define TIMEVAL UNS32

// using 16 bits timer
#define TIMEVAL_MAX 0xFFFF

// The timer is incrementing every 10 us.
//#define MS_TO_TIMEVAL(ms) ((ms) * 100)
//#define US_TO_TIMEVAL(us) ((us) / 10)

#define MS_TO_TIMEVAL(ms) ((ms) * 10)   //1ms 中断，1 000 0hz，即 1ms 计数 10000/1000 = 10 次
#define US_TO_TIMEVAL(us) ((us) / 100)  // 1us 10/1000 =0.01

#endif
```

当然，还需要对 CAN 底层驱动进行初始化，并且还需要将 CAN 底层驱动接收到的 CAN 帧（一般通过 CAN 接收中断来接收数据）转换成 CANopen 协议栈所需要的格式，进而作为参数传进 canDispatch 进行协议处理。下面为示例代码：

代码 4 CAN 接收中断示例代码

```
void USB_LP_CAN1_RX0_IRQHandler(void)
{
   can_rx_msg_t rx_msg;
   can_message_receive(CAN1, CAN_RECEIVE_FIFO_0,&rx_msg);

   /* Assembly canopen data packet */
   canopen_msg.cob_id = rx_msg.std_id;
   canopen_msg.rtr    = rx_msg.remote_txreq;
   canopen_msg.len    = rx_msg.data_length;
   memcpy(canopen_msg.data, rx_msg.data, rx_msg.data_length);

   can_rec_flag = TRUE;
}
```


本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

4. Canfestival 移植到芯海 32 位 MCU-基于 FreeRTOS

前述基于芯海 32 位 MCU 移植 Canfestival 是在裸机环境下，若带 FreeRTOS 移植，基本步骤没有任何区别。只需另外将 FreeRTOS 所需要的内核文件全部加入到工程，另外选择 CS32F103 CM3 内核需要的 port.c 文件、头文件路径和任一内存管理文件（一般选择 heap_4.c）加入工程即可。

应用上，一般可以将 App 任务和 canfestival 协议栈处理任务，单独建立 Task 线程。通过在 CAN 接收中断发送信号量或消息队列的方式与 canfestival 协议栈处理线程同步。

在本应用笔记中，为简单起见，canopen 协议栈的接收处理单独创建线程，can 发送只调用底层驱动接口。Can 中断所收到的数据发送到接收队列里，canfestival 接收 task 里接收这个消息队列。以下为一些接口的示例代码，均有详细注释，用户可参考。

代码 5 FreeRTOS 下代码示例

```
xQueueHandle xCANRcvQueue = NULL;       //CAN 接收队列

/**@brief      主任务 app_task 创建
 *
 * @param[in]
 *
 * @param[in]
 *
 * @return     None.
 */
void app_task_create(void)
{
 BaseType_t xReturn;

 xReturn = xTaskCreate(app_task, "app_task", APP_STACK_SIZE, NULL, APP_TASK_PRIORITY, NULL);
 if(pdPASS != xReturn)
 {
  printf("app_task create failed");
  return;
 }
}

/**@brief      创建接收队列和 canfestival 接收任务，发送直接通过 canSend()直接发送
 *
 * @param[in]
 *
 * @param[in]
 *
 * @return     None.
 */
void canopen_driver_init(void)
{
 BaseType_t xReturn;
```


本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

```
 if(xCANRcvQueue == NULL)
 {
  xCANRcvQueue = xQueueCreate(CANRX_QUEUE_LEN, sizeof(can_rx_msg_t));
  if(xCANRcvQueue == NULL)
  {
   printf("CANRcvQueue create failed");
   return;
  }
 }

 xReturn = xTaskCreate(can_rcv_task, "can_rcv_task", CANRX_STACK_SIZE, NULL, CANRX_TASK_PRIORITY,
NULL);
 if(pdPASS != xReturn)
 {
  printf("CANRcv_Task create failed");
  return;
 }
}

/**@brief    canopen 协议处理
 *
 * @param[in]
 *
 * @param[in]
 *
 * @return    None.
 */
static void can_rcv_task(void *pvParameters)
{
 static can_rx_msg_t rx_msg;
 static Message canopen_msg;

 for(;;)
 {
  if(xQueueReceive(xCANRcvQueue, &rx_msg, 100) == pdTRUE)
  {

    printf("接收到 CANopen 数据...\r\n");

    /* Assembly canopen data packet */
    canopen_msg.cob_id = rx_msg.std_id;
    canopen_msg.rtr    = rx_msg.remote_txreq;
    canopen_msg.len    = rx_msg.data_length;
    memcpy(canopen_msg.data, rx_msg.data, rx_msg.data_length);

    __TIM_INTR_DISABLE(TIM4, TIM_INTR_UPDATE);
    canDispatch(&Slave_Data, &canopen_msg);
    __TIM_INTR_ENABLE(TIM4, TIM_INTR_UPDATE);
  }
 }
}

/**@brief    这个接口主要是 CANopen 所需的一些定时器和 CAN 的配置，以及创建消息队列和 canopen 任务
 *
 * @param[in]
 *
 * @param[in]
 *
```


本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

```
 * @return      None.
 */
void canopen_app_init(void)
{
   BaseType_t xReturn;

   /* CAN init PA11 RX / PA12 TX  500Kbps*/
   can_mode_config(CAN_TIME_CHANGE_UNIT_1,CAN_TS1_UNIT_9,CAN_TS2_UNIT_8,4,CAN_MODE_NOR
   MAL);

   //需要改 timerscfg.h 中 MS_TO_TIMEVAL  US_TO_TIMEVAL   从机心跳最大可设置为 6553.5ms
   //1ms 中断， 10000hz  1s -> 10000 次 -> 1ms,计数 10000/1000 = 10 次 ->  MS_TO_TIMEVAL(ms) ((ms) * 10)
   initTimer();

   canopen_driver_init();

   xReturn = xTaskCreate(canopen_app_task, "canopen_app_task", CANOPEN_STACK_SIZE, NULL,
                                                        CANOPEN_TASK_PRIORITY, NULL);

   if(pdPASS != xReturn)
   {
      return;
   }
}


/**@brief      canopen 处理的主任务
 *
 * @param[in]
 *
 * @param[in]
 *
 * @return      None.
 */
static void canopen_app_task(void *pvParameters)
{
   printf("CanOpen Slave Node Test...\r\n\n");

   unsigned char nodeID = SLAVE_NODE_ID;

   setNodeId(&Slave_Data, nodeID);

   //从 Initialisation 到 Pre-operational 的跳转从节点自动完成
   setState(&Slave_Data, Initialisation);

   //主机需要发送 NMT 命令 启动从节点，如果不发启动命令，从机可发心跳， 可 SDO,不可 PDO
//    setState(&Slave_Data, Operational);

   while(1)
   {
      {
         Temperture[0] +=1; //字典值 更新  通过 PDO 发给主机;

         if(getState(&Slave_Data) == 0x5)
         {
            Message temperture_value;
            temperture_value.cob_id = 0x281;
            temperture_value.rtr = 0;
            temperture_value.len = 4;
```


本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

```
            memcpy(temperture_value.data, Temperture, temperture_value.len);
            canSend((&Slave_Data)->canHandle,&temperture_value);
          }
        else
        {
            UNS32 size = sizeof(UNS32);
            UNS16 index = 0x2000;
            UNS32 value = Temperture[0];
            UNS8 subindex=1;
                        writeLocalDict( &Slave_Data, /*CO_Data* d*/
                                    index,      /*UNS16 index*/
                                    subindex,       /*UNS8 subind*/
                                    &value,  /*void * pSourceData,*/
                                    &size,       /* UNS8 * pExpectedSize*/
                                    RW);        /* UNS8 checkAccess */
        }


      }
      sdo_test[0] = 0xAA; //主机 SDO 读 Mapped at index 0x2002, subindex 0x01 - 0x01 -> 4f 02 20 01 00 00 00 00

      printf("Temperture 值为%x...\r\n",Temperture[0]);

      vTaskDelay(3000); //任务切换
    }
}


/**@brief      主任务：底层、canopen 协议栈初始化，开始主任务
 *
 * @param[in]
 *
 * @param[in]
 *
 * @return      None.
 */
static void app_task(void *pvParameters)
{
    //串口 PB10(TX) and PB11(RX);  LED1 15  LED2 14 ; 按键 PC13
    bsp_init();

    canopen_app_init();

    while(1) //用户应用层 app 任务
    {
      led1_toggle();
      led2_toggle();

      vTaskDelay(1000);
    }
}
```

同时，CAN 接收中断的处理相比与裸机的中断处理，也有所不同，下面是基于 RTOS 时的 can 接收中断示例代码：

本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

```
/**@brief     This function handles CAN1 interrupt request
 *
 * @param[in]   None.
 *
 * @return      None.
 */
void USB_LP_CAN1_RX0_IRQHandler(void)
{
   can_rx_msg_t rx_msg;

   BaseType_t err;
   BaseType_t xHigherPriorityTaskWoken;

   can_message_receive(CAN1, CAN_RECEIVE_FIFO_0,&rx_msg);

   err = xQueueSendFromISR(xCANRcvQueue,&rx_msg, &xHigherPriorityTaskWoken);
   if(err == errQUEUE_FULL)
   {
      printf("rec_Queue send fail because of full\r\n");
   }

   // Now the buffer is empty we can switch context if necessary.
   if( xHigherPriorityTaskWoken )
   {
      // Actual macro used here is port specific.
      portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
   }
}
```


本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

5. 基于 canfestival 的 demo 解析

5.1 对象字典文件的解析

以下为一些重点关注的对象字典索引的 c 文件摘录：

代码 6 对象字典示例代码摘录

```
#include "Slave_f103.h"
#include "node_config.h"

/**************************************************************************/
/* Declaration of mapped variables                                        */
/**************************************************************************/
UNS32 Temperture[] =              /* Mapped at index 0x2000, subindex 0x01 - 0x01 */
 {
   0
 };
UNS8 ctr_para[] =                 /* Mapped at index 0x2001, subindex 0x01 - 0x01 */
 {
   0x0    /* 0 */
 };
UNS8 sdo_test[] =                 /* Mapped at index 0x2002, subindex 0x01 - 0x01 */
 {
   0x0    /* 0 */
 };

/**************************************************************************/
/* The node id                                                            */
/**************************************************************************/
/* node_id default value.*/
UNS8 Slave_bDeviceNodeId = 0x00;

/**************************************************************************/
/* Array of message processing information */

const UNS8 Slave_iam_a_slave = 1;

TIMER_HANDLE Slave_heartBeatTimers[1];

/*
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

                        OBJECT DICTIONARY

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
*/

/* index 0x1005 :   SYNC COB ID */
                UNS32 Slave_obj1005 = 0x0;   /* 0 */   //表明本节点不产生同步帧

/* index 0x1006 :   Communication / Cycle Period */
                UNS32 Slave_obj1006 = 0x0;   /* 0 */   //对从机，无意义

/* index 0x100C :   Guard Time */
                UNS16 Slave_obj100C = 0x0;   /* 0 */
```


本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

```
/* index 0x1016 :   Consumer Heartbeat Time */
                    UNS8 Slave_highestSubIndex_obj1016 = 0; //主机根据这个时间检测从机，如果从机丢失，进入回
调函数_heartbeatError
                    UNS32 Slave_obj1016[]={0};

/* index 0x1017 :   Producer Heartbeat Time. */      //从机的心跳, 单位 ms
                    UNS16 Slave_obj1017 = Producer_Heartbeat_Time;         /* 50 */
                    subindex Slave_Index1017[] =
                     {
                       { RW, uint16, sizeof (UNS16), (void*)&Slave_obj1017, NULL }
                     };
/* index 0x1200 :   Server SDO Parameter. */
                    UNS8 Slave_highestSubIndex_obj1200 = 2; /* number of subindex - 1*/
                    UNS32 Slave_obj1200_COB_ID_Client_to_Server_Receive_SDO = 0x600|SLAVE_NODE_ID;
              /* 1537 */
                    UNS32 Slave_obj1200_COB_ID_Server_to_Client_Transmit_SDO = 0x580|SLAVE_NODE_ID;
              /* 1409 */
                    subindex Slave_Index1200[] =
                     {
                       { RO, uint8, sizeof (UNS8), (void*)&Slave_highestSubIndex_obj1200, NULL },
                       { RO, uint32, sizeof (UNS32), (void*)&Slave_obj1200_COB_ID_Client_to_Server_Receive_SDO,
NULL },
                       { RO, uint32, sizeof (UNS32), (void*)&Slave_obj1200_COB_ID_Server_to_Client_Transmit_SDO,
NULL }
                     };
/* index 0x1800 :   Transmit PDO 1 Parameter. */
                    UNS8 Slave_highestSubIndex_obj1800 = 6; /* number of subindex - 1*/
                    UNS32 Slave_obj1800_COB_ID_used_by_PDO = 0x280|SLAVE_NODE_ID;        /* 641 */
//                    UNS8 Slave_obj1800_Transmission_Type = 0x0;          /* 0 */  //非循环同步模式
//                    UNS8 Slave_obj1800_Transmission_Type = 0x3;          /* 0 */  //循环同步模式,3 次同步
                    UNS8 Slave_obj1800_Transmission_Type = 0xFF;         /* 0 */  //设备子协议特定事件
//                    UNS8 Slave_obj1800_Transmission_Type = 0xFE;         /* 0 */  //设备制造商特定事件

                    UNS16 Slave_obj1800_Inhibit_Time = 0x0;        /* 0 */   //单位 0.1ms
                    UNS8 Slave_obj1800_Compatibility_Entry = 0x0;        /* 0 */
                    UNS16 Slave_obj1800_Event_Timer = 0x0;         /* 0 */   //单位 ms
                    UNS8 Slave_obj1800_SYNC_start_value = 0x0;  /* 0 */
                    subindex Slave_Index1800[] =
                     {
                       { RO, uint8, sizeof (UNS8), (void*)&Slave_highestSubIndex_obj1800, NULL },
                       { RW, uint32, sizeof (UNS32), (void*)&Slave_obj1800_COB_ID_used_by_PDO, NULL },
                       { RW, uint8, sizeof (UNS8), (void*)&Slave_obj1800_Transmission_Type, NULL },
                       { RW, uint16, sizeof (UNS16), (void*)&Slave_obj1800_Inhibit_Time, NULL },
                       { RW, uint8, sizeof (UNS8), (void*)&Slave_obj1800_Compatibility_Entry, NULL },
                       { RW, uint16, sizeof (UNS16), (void*)&Slave_obj1800_Event_Timer, NULL },
                       { RW, uint8, sizeof (UNS8), (void*)&Slave_obj1800_SYNC_start_value, NULL }
                     };

/* index 0x1A00 :   Transmit PDO 1 Mapping. */
                    UNS8 Slave_highestSubIndex_obj1A00 = 1; /* number of subindex - 1*/
                    UNS32 Slave_obj1A00[] =
                    {
                      0x20000120 /* 536871200 */
                    };
                    subindex Slave_Index1A00[] =
                     {
                       { RW, uint8, sizeof (UNS8), (void*)&Slave_highestSubIndex_obj1A00, NULL },
                       { RW, uint32, sizeof (UNS32), (void*)&Slave_obj1A00[0], NULL }
```


本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

```
                };

/* index 0x2000 :   Mapped variable Temperture */
                UNS8 Slave_highestSubIndex_obj2000 = 1; /* number of subindex - 1*/
                subindex Slave_Index2000[] =
                {
                  { RO, uint8, sizeof (UNS8), (void*)&Slave_highestSubIndex_obj2000, NULL },
                  { RW, uint8, sizeof (UNS32), (void*)&Temperture[0], NULL }
                };
/* index 0x2002 :   Mapped variable sdo_test */
                UNS8 Slave_highestSubIndex_obj2002 = 1; /* number of subindex - 1*/
                subindex Slave_Index2002[] =
                {
                  { RO, uint8, sizeof (UNS8), (void*)&Slave_highestSubIndex_obj2002, NULL },
                  { RW, uint8, sizeof (UNS8), (void*)&sdo_test[0], NULL }
                };
CO_Data Slave_Data = CANOPEN_NODE_DATA_INITIALIZER(Slave);
```

5.2 主程序的解析

此处，给出一个 main 函数的示例，LED 灯交替亮灭，Temperture[0]值逐渐累加，并通过 TPDO 发送给主机（发送方式在 index 0x1800 处配置），sdo_test[0] 为 0xAA，并在接收到主机的 client SDO 时返回给主机。

代码 7 主函数示例代码

```
int main(void)
{
   nvic_priority_group_config(NVIC_PriorityGroup_4); //适配 RTOS

   //串口 PB10(TX) PB11(RX)，初始化 LED1/2;
   bsp_init();

   /* CAN init PA11 RX / PA12 TX  500Kbps*/
   can_mode_config(CAN_TIME_CHANGE_UNIT_1,CAN_TS1_UNIT_9,CAN_TS2_UNIT_8,4,CAN_MODE_NOR
   MAL);

   //需要对应修改 timerscfg.h 中 MS_TO_TIMEVAL 、US_TO_TIMEVAL 宏
   initTimer();

   //注册一些回调函数，可自行定义，这里均定义成打印一串信息
   slave_callback();

   printf("CanOpen Slave Node Test...\r\n\n");

   unsigned char nodeID = SLAVE_NODE_ID;

   setNodeId(&Slave_Data, nodeID);

   //从 Initialisation 到 Pre-operational 的跳转从节点自动完成
   setState(&Slave_Data, Initialisation);

   //主机需要发送 NMT 命令 启动从节点，如果不发启动命令，从机可发心跳帧，可 SDO,不可 PDO 通信
```


本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

```
//    setState(&Slave_Data, Operational);

    while(1)
    {
        if(can_rec_flag)
        {
            can_rec_flag = FALSE;

            __TIM_INTR_DISABLE(TIM4, TIM_INTR_UPDATE);

            canDispatch(&Slave_Data, &canopen_msg);

            __TIM_INTR_ENABLE(TIM4, TIM_INTR_UPDATE);
        }

        led1_toggle();
        led2_toggle();
        delay(0xFFFFFF);
        {
            Temperture[0] +=1; //通过 PDO 发给主机；

            if(getState(&Slave_Data) == 0x5)
            {
                Message temperture_value;
                temperture_value.cob_id = 0x281;
                temperture_value.rtr = 0;
                temperture_value.len = 4;
                memcpy(temperture_value.data, Temperture, temperture_value.len);
                canSend((&Slave_Data)->canHandle,&temperture_value);
            }
            else
            {
                UNS32 size = sizeof(UNS32);
                UNS16 index = 0x2000;
                UNS32 value = Temperture[0];
                UNS8 subindex=1;
                writeLocalDict( &Slave_Data, /*CO_Data* d*/
                                index,         /*UNS16 index*/
                                subindex,      /*UNS8 subind*/
                                &value,        /*void * pSourceData,*/
                                &size,         /* UNS8 * pExpectedSize*/
                                 RW);          /* UNS8 checkAccess */
            }
        }
        sdo_test[0] = 0xAA; //主机 SDO 读, Mapped at index 0x2002, subindex 0x01
        printf("Temperture 值为%x...\r\n",Temperture[0]);
    }

}
```

以下为基本通信过程说明：

1. 从机 CobID 为 0x701 ，报文数据为 0，从机在 Initialisation 后自动向主机发 bootUP 报文；

帧序号	消息名称	帧ID(Hex)	长度	数据(Hex)	时间标识	方向	帧类型	帧格式	通道号	注释
0	HBGuard_001	0x00000701	1	00	19:17.15.647	接收	标准帧	数据帧	CAN1	
信号…	信号名称	原始值(Hex)	实际值	值描述	单位	变换比例	变换偏移	起始位	位宽	注释
1	Toggle	0	0			1	0	7	1	
2	State	0	0	0 - Bootup		1	0	0	7	

本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

图 15 bootUP 报文

2. 从机 CobID 0x701， 报文数据为 127，向主机报告 Pre-operational 状态，等待主机启动从节点；

图 16 Pre-operational 状态报文

3. CANopen 主机发 NMT 启动命令，启动从机 1（从机节点 ID 设置成 1）；

	选择	消息名称	ID(Hex)	DLC	原始数据(Hex)	间隔(ms)	注释
1	✓	NMTZeroMsg	0x0	2	01 01	10	

	信号名称	原始值(Hex)	实际值	值描述	单位	变换比例	变换偏移	起始位	位宽	最小值	最大值	注释
1	Node_ID	1	1	0 - All		1	0	8	8	0	255	
2	CS	1	1	1 - Start		1	0	0	8	0	255	

图 17 主机发 NMT 启动命令报文

4. 从机变更状态为 Operational,可开始发送和接收 PDO；

图 18 Operational 状态报文

5. 主机通过 SDO 读 Temperture 和 sdo_test 值；

```
[2023-02-28 20:10:00.217]# RECV ASCII>
Test 1...
..\CanFestival\src\sdo.c,773 : 0X3a60 proceedSDO  0X0
..\CanFestival\src\sdo.c,789 : 0X3a62 proceedSDO. I am server. index :  0X1200
..\CanFestival\src\sdo.c,837 : 0X3a69 I am SERVER number  0X0
..\CanFestival\src\sdo.c,1106 : 0X3a89 Received SDO Initiate upload (to send data) defined
at index 0x1200 +  0X0
..\CanFestival\src\sdo.c,1108 : 0X3a90 Reading at index :  0X2002
..\CanFestival\src\sdo.c,1109 : 0X3a91 Reading at subIndex :  0X1
..\CanFestival\src\sdo.c,476 : 0X3a25 init SDO line nb

[2023-02-28 20:10:00.266]# RECV ASCII>
 :  0X0
..\CanFestival\src\sdo.c,479 : 0X3a06 StartSDO_TIMER for line :  0X0
..\CanFestival\src\sdo.c,252 : 0X3a05 objdict->line index :  0X2002
..\CanFestival\src\sdo.c,253 : 0X3a06   subIndex :  0X1
..\CanFestival\src\sdo.c,1164 : 0X3a96 SDO. Sending expedited upload initiate response
defined at index 0x1200 +  0X0
..\CanFestival\src\sdo.c,671 : 0X3a38 sendSDO 0X0
..\CanFestival\src\sdo.c,685 : 0X3a41 I am server Tx cobId :  0X581
..\CanFestival\src\sdo.c,454 : 0X3a25 reset SDO line nb :  0X0
..\C

[2023-02-28 20:10:00.330]# RECV ASCII>
anFestival\src\sdo.c,476 : 0X3a25 init SDO line nb :  0X0
..\CanFestival\src\sdo.c,481 : 0X3a05 StopSDO_TIMER for line :  0X0
..\CanFestival\src\timer.c,106 : 0X3320 DelAlarm. handle =  0X1
```

图 19 打印信息-主机 SDO 读

本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

'A0503-[5300062C][CAN1] 清空数据 暂停显示 滚动显示 展开所有

消息名称	帧ID(Hex)	长度	数据(Hex)	时间标识
HBGuard_001	0x00000701	1	7F	20:10.10.341
NMTZeroMsg	0x00000000	2	01 01	19:57.19.121
SYNC	0x00000080	2	01 01	19:29.30.870
CSDO_001	0x00000601	8	4F 02 20 01 00 00 00 00	20:09.57.740
SSDO_001	0x00000581	8	4F 02 20 01 AA 00 00 00	20:10.00.492

图 20　主机 SDO 读且从机返回数据

6. 主机通过 SDO 写通信，修改从机心跳值，从机心跳从 3s 变为 1s

265	0x701	1	7F	02:16.29.817
266	0x601	8	2B 17 10 00 E8 03 00 00	02:16.30.268
267	0x701	1	7F	02:16.30.818
268	0x701	1	7F	02:16.31.819
269	0x581	8	60 17 10 00 00 00 00 00	02:16.32.601
270	0x701	1	7F	02:16.32.608

图 21　心跳从 3s 变为 1s

7. 修改从机对象字典 c 文件中 0x1800 索引，配置温度值为非循环同步 TPDO，主机发来同步帧后，触发 PDO；

1	0x701	1	7F	00:31.14.327
2	0x701	1	7F	00:31.17.328
3	0x0	2	01 01	00:31.18.074
4	0x701	1	05	00:31.20.330
5	0x701	1	05	00:31.23.331
6	0x701	1	05	00:31.26.332
7	0x80	8	00 00 00 00 00 00 00 00	00:31.29.170
8	0x701	1	05	00:31.29.333
9	0x281	4	01 00 00 00	00:31.30.503
10	0x701	1	05	00:31.32.335
11	0x701	1	05	00:31.35.336

图 22　从机同步触发 PDO

8. 修改从机对象字典 c 文件中 0x1800 索引，配置温度值为非循环同步 TPDO，此时温度值为常量，主机发来同步，不触发 PDO；

1	05
1	05
1	05
8	00 00 00 00 00 00 00 00
1	05
1	05
8	00 00 00 00 00 00 00 00
1	05
1	05

图 23　从机同步不触发 PDO

9. 修改从机对象字典 c 文件中 0x1800 索引，温度值为常量，配置温度值为循环同步 TPDO，并且为循环 3 次同步；

本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA

0	2	01 01
701	1	05
701	1	05
80	8	00 00 00 00 00 00 00 00
701	1	05
701	1	05
701	1	05
80	8	00 00 00 00 00 00 00 00
701	1	05
701	1	05
80	8	00 00 00 00 00 00 00 00
281	4	01 00 00 00
701	1	05
701	1	05

图 24 从机同步 3 次触发 PDO

10. 修改从机对象字典 c 文件中 0x1800 索引，温度值为变量，配置温度值为设备子协议特定事件 TPDO，同时修改定时触发时间：Slave_obj1800_Event_Timer

.7	0x701	1	05
.8	0x0	2	01 01
.9	0x281	4	0A 00 00 00
20	0x701	1	05
21	0x281	4	0B 00 00 00
22	0x701	1	05
23	0x281	4	0C 00 00 00

图 25 从机定时触发 PDO

芯海科技
CHIPSEA
股票代码:688595

免责声明和版权公告

本文档中的信息，包括供参考的 URL 地址，如有变更，恕不另行通知。

本文档可能引用了第三方的信息，所有引用的信息均为“按现状”提供，芯海科技不对信息的准确性、真实性做任何保证。

芯海科技不对本文档的内容做任何保证，包括内容的适销性、是否适用于特定用途，也不提供任何其他芯海科技提案、规格书或样品在他处提到的任何保证。

芯海科技不对本文档是否侵犯第三方权利做任何保证，也不对使用本文档内信息导致的任何侵犯知识产权的行为负责。本文档在此未以禁止反言或其他方式授予任何知识产权许可，不管是明示许可还是暗示许可。

Wi-Fi 联盟成员标志归 Wi-Fi 联盟所有。蓝牙标志是 Bluetooth SIG 的注册商标。

文档中提到的所有商标名称、商标和注册商标均属其各自所有者的财产，特此声明。

版权归 © 2023 芯海科技（深圳）股份有限公司，保留所有权利。

本资料为芯海科技专有财产，非经许可，不得复制、翻印或转变其他形式使用。
This document is exclusive property of CHIPSEA and shall not be reproduced or copied or transformed to any other format without prior permission of CHIPSEA