基于 uC/OS – II 的 CANopen 从节点的实现①

A Method of Realizing CANopen Slave Node Based on uC/OS – II

徐 喆 张 卓 闫士珍 （北京工业大学 电子信息与控制工程学院 北京 100124）

摘 要： 文章简单介绍了现有的 CANopen 协议栈，提出了一种利用开源代码 CANfestival[1] 在 MC9S12XDP512 平台上实现 CANopen 网络从节点的新方法，并将 CANfestival 移植到实时操作系统 uC/OS – II 上，简化了原有代码的编译过程，大大缩短了开发周期，提高了代码移植的灵活性，对发展自主版权的现场总线协议标准和协议栈具有重要的现实意义。

关键词： CANopen CANfestival 从节点 协议栈 实时操作系统

1 CANopen 协议应用现状

在汽车电子领域，CAN 总线凭借其可靠性、实时性和灵活性等优势，已经成为车用总线的主流。众多国际组织在 CAN 总线的基础上已经推出了 40 余种应用层协议。比较著名的应用层协议包括 SAE? J1939、CAL、CANopen 等。CANopen 协议最初由 Bosch 公司倡导，1992 年，在德国成立了“自动化 CAN 用户和制造商协会（CiA，CAN In Automation）”，开始着手制定一个标准的 CAN 的应用层协议规范——CANopen 协议。

CANopen 协议着重定义了应用层以及相关的通讯架构，详细包括对象字典、网络管理、启动配置、各种传输对象的定义等等。在 CAN 物理层的基础上，建立了基本的数据对象传输机制，并详细的规定了每个节点的配置模式、启动流程、网络管理以及异常处理等等，详细定义并完善了 CAN 网络系统。它采用面向对象的思想设计，具有很好的模块化特性和很高的适应性，通过扩展可以适用于大量的应用领域。

2 利用开源代码开发协议栈

商用 CANopen 协议栈价格昂贵，若购买现成的协议栈，不但会增加开发成本，还会增加开发难度。因此进行 CANopen 协议的开发，首先想到的是如何利用现有的开源代码。在开源代码方面，有两个组织的开源 CANopen 协议栈可供使用：

其中一个是由美国的嵌入式系统研究会（Embedded Systems Academy）支持的微型 CANopen 项目（Micro Canopen）[2]。该项目提供的开源 CANopen 协议栈并不支持所有的 CANopen 功能，仅作为学习目的公开。它的体积很小，可以在 8051 位控制器上实现，并只需要 4K 字节的代码空间和 170 字节的数据空间。

另一个是由法国的 Loli Tech 资助的 CANfestival 项目[1]。与微型 CANopen 项目不同的是，该项目是一个由开源社区 LGPL 授权的完整 CANopen 协议栈，支持在 PC 和微控制器上的仿真与实现，需要 40K 字节以上的代码空间。最新版本为 2006 年 8 月的 3.0 版本，并且新版本正在开发中。CANfestival 代码由 C 语言编写，符合 DS301. V. 4. 02 的规定，每个节点有 1 个 SDO Server，PDO 个数不限，可以使用 NMT 命令改变从节点状态，PDO 的发送模式有请求、同步、事件三种驱动方式。NMT 心跳报文机制、NMT 节点保护机制。

比较这两个协议栈，微型 CANopen 协议栈代码量相对较小，不使用操作系统，实现简单，但是它并不是一个完整的协议栈，并且从一开始就仅作为学习目的发布，如果在其基础上实现全部功能，代码空间仍需 40K 字节以上的代码空间。而 CANfestival 则不同，从项目初始就是一个完整的协议栈，并在不断的升级当中。因此，从 CANfestival 开始开发自己的协议栈比较可行。

① 基金项目：北京市科技新星基金（2003A005）；北京市教委重点项目和北京市自然科学基金（KZ20041000501）

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

3 研究意义

本文从修改 CANfestival 代码的角度出发，提出了一种新的构建 CANopen 从节点平台的方法。CANfestival 采用 C 语言开发，源代码并不是基于操作系统的，这增加了其应用开发的难度。针对 S12 平台，代码的编译可以采用两种方法：一是利用 GCC，可以使用 GNU Development Chain for 68HC11&68HC12[3] 进行编译，该软件支持 Freescale HC/S12 系列单片机的开发。但由于 GCC 是 Linux 下的编译器和交叉编译器，沿用的是 Linux 命令行方式，命令参数选项特别多，对于不熟悉 Linux 的用户不太容易学习，对于初学者来说要花费相当多的时间。而 CodeWarrior for S12 是专门面向以 HC12 或 S12 为 CPU 的单片机嵌入式应用开发的软件包[5]。许多操作只需点击鼠标即可完成，可视化强，能够大大提高开发人员的效率，缩短开发周期。因此有着 GCC 所不能比拟的优势。而使用实时操作系统不但可以使系统的实时性得以保证，还可以提高代码扩展应用的灵活性。

4 在 CodeWarrior 编译环境下修改代码

4.1 代码结构

首先下载 CANfestival2. 03 版本，表 1 对各文件夹中的代码做出说明。

表 1　协议栈结构说明

文件路径	文件说明
CanOpen/CanOpenMain	包括所有与处理器相关的. c 文件
CanOpen/include/hc12	applicfg. h 处理器硬件配置相关的. h 文件
CanOpen/include/hc12	timerhw. h、interrupt. h 定时器、中断相关的. h 文件
CanOpen/CanOpenDriver-HC12	针对 HC12 平台的 CANopen 驱动
CanOpen/AppliSlave_HC12	appli. c、objdict. c 应用程序和描述对象字典的代码

注：如果只是构建从节点，则不必使用 AppliMaster_HC12 文件夹。

工程的建立以 AppliSlave_HC12 中的 appli. c 文件为基础，把相关的. c 文件添加至工程里。

（1）appli. c 里面包括以下. c 文件，在这里选择把相关的. c 文件用#include 包含到主文件里，这样不但使工程简单，也不用考虑限制版的 CodeWarrior 的添加文件个数的限制。. c 文件包括以下几个文件：canOpenDriver. c、init. c、interrupt. h、objacces. c、sdo. c、pdo. c、objdict. c、lifegrd. c、timer. c、timerhw. c、nmtSlave. c、nmtMaster. c、sync. c、variahw. c。

（2）以下. h 文件是针对处理器的，这些不必添加到工程里，也是使用 include 的方法加进分别的文件。regs. h、exit. h、interrupt. h、param. h、portsaccess. h、ports_def. h、ports. h、applicfg. h、interrupt. h、timerhw. h、CanOpenMain. h。

（3）以下文件是 HC12 的相关驱动。在这些文件中定义了所有硬件相关的文件，这些文件与 CANopen 无关，只是与不同的硬件平台有关。这部分也是改动最大的部分。candriver. h、canOpenDriver. h、error. h、interrupt. c、timerhw. c、variahw. c、regbase. h。

4.2 建立工程

从站系统构成包括：MC9S12XDP512 开发板、S12 的集成开发环境 CodeWarrior 4. 6、USB Multilink 仿真器。首先在 CodeWarrior 下建立工程。通常使用的 CodeWarrior 都是限制版的，对编译生成的代码量会有要求，所以建议使用 CodeWarrior for HC12 Professional Edition，这个版本需要申请无限制的 lisece，有效期 30 天。

4.3 代码修改

从站的主程序是一个状态机的轮换，按照 DS301 的规定，从节点启动运行后会有四个状态，分别是初始化、预运行、运行、停止。代码中分别用 Initialisation、Pre_operational、Operational、Stopped 四种状态来表示。用 switch 语句实现状态的切换。

```
switch( getState( ) ) {
    case Initialisation:
    ……
    case Pre_operational:
    ……
    case Operational:
    ……
    case Stopped:
    ……}
```


©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

四个状态的功能函数分别是 initialisation( )、preOperational( )、operational( )、stopped( )。初始化函数 initialisation( )中包含的程序流程图如图 1 所示。这部分代码需要修改的地方比较多，包括中断函数和 CAN 驱动初始化函数 initCanHCS12( ) 里的 CAN 波特率结构体。

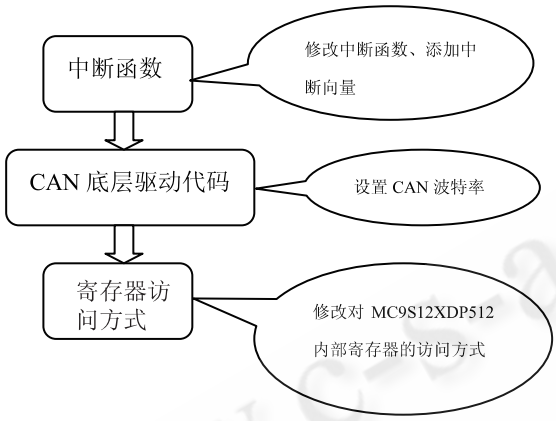

图 1　代码修改步骤

1、中断函数的修改

(1) GCC 编译方式下中断响应的关键字是“__attribute__”。

源代码中对中断处理函数的声明和定义如下：

```
void __attribute__((interrupt)) timer3Hdl(void);
```

修改为 CodeWarrior 下的声明和定义：

```
void interrupt timer3Hdl(void);
pragma CODE_SEG__NEAR_SEG_NON_BANKED
void interrupt timer3Hdl(void)
{...}
pragma CODE_SEG_DEFAULT
```

此处的

```
#pragma CODE_SEG __NEAR_SEG NON_BANKED
#pragma CODE_SEG_DEFAULT
```

要成对使用，目的是加上声明，将中断函数放入非分页地址。因为中断矢量只有 16 位，无法在分页地址中寻址。因此，中断函数必须放入非分页地址。

(2) 中断向量的添加

在 CodeWarrior 中建立工程时，生成了几个默认的文件夹，其中. prm 文件夹中定义的是连接使用到的参数，这些是与硬件相关的，可以根据使用的单片机的型号来修改这些参数。该文件夹下的 P&E_Multilink_CyclonePro_linker. prm 和 Full_Chip_Simulation_linker. prm 这两个文件用于定义目标代码的装载地址，可以根据需要修改这个文件。其中，MY_RAM 是程序的数据区，MY_ROM 是程序的代码区。

VECTOR 0 _Startup 表示把单片机的 0FFFF 处的复位向量设为这个程序的入口地址。具体中断向量可以查看数据手册[6]。

全部中断向量定义如下：

```
VECTOR 0 _Startup
VECTOR ADDRESS 0xFFE8 timer3Hdl
VECTOR ADDRESS 0xFFE6 timer4Hdl
VECTOR ADDRESS 0xFF90 can4HdlTra
VECTOR ADDRESS 0xFF92 can4HdlRcv
VECTOR ADDRESS 0xFF96 can4HdlWup
VECTOR ADDRESS 0xFF94 can4HdlErr
```

(3) GCC 编译环境下的代码中，开中断和关中断的语句写法如下：

```
void unlock (void)
{
__asm__ __volatile__("cli");
}
void lock (void)
{
unsigned short mask;
__asm__ __volatile__("tpa\n\tsei":"=d"(mask));
}
```

但是 CodeWarrior 中不支持这样的写法，修改如下：

```
void unlock (void)
{
__asm("cli");
}
void lock (void)
{
unsigned short mask;
__asm
{
tpa:tsei:"=d"(mask);
}
}
```


©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

2、CAN 底层驱动代码的修改

CAN 驱动程序在从站初始化中作用很重要，它定义了与 CAN 相关的数据结构和功能函数，依照 CAN2.0A 协议中定义的 CAN 报文的格式编写程序，设置了 CAN 总线通信速率、CAN 寄存器等。主要修改的地方是 CAN 总线初始化中的数据结构定义。该函数位于从节点初始化过程中的 initCanHCS12( )函数里，该函数主要是初始化与 CANopen 相关的硬件，本文中主要是指 MC9S12XDP512 的硬件。例如函数里的 canBusInit 是个如下结构的结构体：

```
typedef struct {
UNS8 cswai; /* 等待模式为低电压或正常 */
UNS8 time; /* 时间戳定时器使能（1/0） */
UNS8 cane; /* CAN 使能（yes =1） */
UNS8 clksrc; /* 时钟源选择 */
UNS8 loopb; /* 测试模式（1/0）*/
UNS8 listen;/* CAN 监听 */
UNS8 wupm; /* 低通滤波唤醒(yes =1/no =0) */
canBusTime clk; /* 时钟系统初始化的值 */
canBusFilterInit fi; /* 报文接收寄存器的初始化值 */
} canBusInit;
```

在 candriver.h 文件中定义变量 bi 为 canBusInit 类型的结构体：

```
extern canBusInit bi;
```

变量 bi0 就是 canBusInit 类型的结构体：

```
const canBusInit bi0 = {
0, /* 等待模式为正常 */
0, /* 时间戳定时器关闭 */
1, /* CAN 使能 */
0, /* MSCAN 时钟源为晶振时钟 */
0, /* 自收自发测试模式关闭*/
0, /* CAN 监听关闭*/
0,/* 低通滤波唤醒关闭*/
{/* 时钟系统初始化的值 */
1, /* clksrc */
3, /* brp */
0, /* sjw */
0, /* samp */
1, /* tseg2 */
12,/* tseg1 */
},
{ /* 报文接收寄存器的初始化值 */
0x01, /* 选择设置为 4 个 16 bits 缓冲寄存器*/
0x00, 0xFF, /* filter 0 hight accept all msg */
0x00, 0xFF, /* filter 0 low accept all msg */
0x00, 0xFF, /* filter 1 hight filter all of msg */
0x00, 0xFF, /* filter 1 low filter all of msg */
0x00, 0xFF, /* filter 2 hight filter most of msg */
0x00, 0xFF, /* filter 2 low filter most of msg */
0x00, 0xFF, /* filter 3 hight filter most of msg */
0x00, 0xFF, /* filter 3 low filter most of msg */
}
};
```

3、修改寄存器访问方式

使用 CodeWarrior 来建立工程，系统默认添加了 mc9s12xdp512.c 和 mc9s12xdp512.h 两个文件，这里包含了对于 CPU 寄存器的定义，地址访问方式。但是 CANfestival 中并没有使用 CodeWarrior 建立工程时自动添加的.c 和.h 文件，也就是说 CANfestival 代码中关于 CPU 寄存器的定义和地址访问方式另有文件来定义。这个文件名是 ports_def.h，这个文件相当于建立工程时系统默认添加的 mc9s12xdp512.h，这部分代码不需要改动。而是要重新修改 portsaccess.h，它定义了对寄存器的访问方式，具体修改如下：

```
extern volatile unsigned char _io_ports\[ \];
#define _io_ports ((char*)(0))
#define IO_PORTS_8(adr) _io_ports\[adr\]
#define IO_PORTS_16(adr) *((unsigned volatile short*)(_io_ports + (adr)))
```

5 协议栈向 uC/OS－Ⅱ操作系统的移植

移植到 uC/OS－Ⅱ操作系统上的协议栈运行的流程图如图 2 所示。如前文所讲，没有移植到 uC/OS－Ⅱ操作系统上的协议栈是一个状态机，要将协议栈移植到操作系统上，就需要把这个状态机封装成一个任务。

5.1 主函数的编写

将从站代码移植到操作系统上需要重新定义一个主函数，包括操作系统初始化、创建任务和启动任务。

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

```
void main (void)
{
    INT8U err;
    OSInit();
    OSTaskCreate (AppTask1, (void *) 0,
&AppTask1Stk\[TASK_STK_SIZE-1\],TASK_1_PRIO);
    OSStart();
}
```

AppTask1 是操作系统创建的 CANopen 从站任务函数。

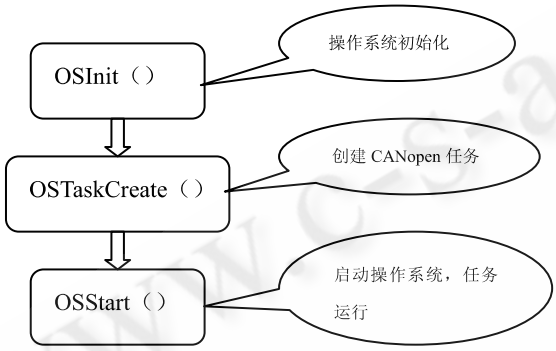

图 2　操作系统下的协议栈流程图

5.2 任务封装

CANopen 从站的主函数是一个状态调度机，将该主函数封装成一个任务，程序如下：

```
static void AppTask1 (void *p_arg)
{
INT8U err;
e_nodeState lastState = Unknown_state;
(void)p_arg;
p_arg = p_arg;
while(1) { /* 从节点状态机 */
switch( getState() ) {
case Initialisation:
……
    case Pre_operational:
……
    break;
    case Operational:
……
    case Stopped:
……
  }
}
```

5.3 中断的处理

OS_CPU_A. S 和 OS_CPU. C 是操作系统中与 CPU 有关的代码文件，分别用汇编和 C 语言编写。在 OS_CPU_A. S 中，对 CANopen 从站代码所用到的中断函数进行定义，原代码中使用了 6 个中断函数，定义格式如下：

```
……
    xdef timer3Hdl_A
    xdef timer4Hdl_A
    xdef can4HdlTra_A
    xdef can4HdlRcv_A
    xdef can4HdlWup_A
    xdef can4HdlErr_A
……
……
    xref timer3Hdl
    xref timer4Hdl
    xref can4HdlTra
    xref can4HdlRcv
    xref can4HdlWup
    xref can4HdlErr
……
```

其中 xdef 是在汇编文件里定义的中断函数，xref 是在 C 文件里定义的中断函数，是被调用的函数。作用是通知操作系统，有中断发生，需要进行处理。程序可以仿照操作系统本身自带的中断函数的写法。

然后需要对中断向量进行定义，方法与不带操作系统的移植类似，在 P&E_Multilink_CyclonePro_linker. prm 和 Full_Chip_Simulation_linker. prm 这两个文件中定义目标代码的装载地址。

6　从节点的连接调试

将移植在 uc/OS－Ⅱ实时操作系统上的从站与 PC 机主节点进行通讯，采用 CAN232 智能转换器监视报文，并将相应信息从串口打印输出，如图 3、图 4 所示。用超级终端显示输出信息，证明从节点工作正常。

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

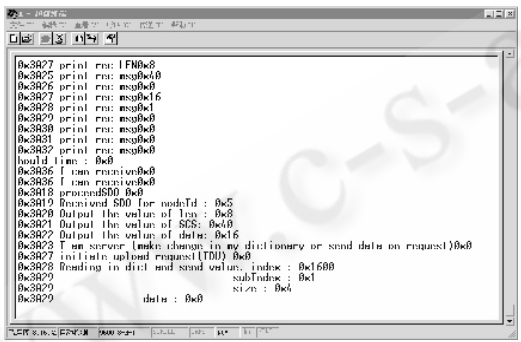

图 3　串口输出

图 4　RS232/CAN 监控结果

7　结论

文章对 CANopen 现有协议栈作出了简单介绍，提出了一种利用 CANfesitval 开源代码在 MC9S12XDP512 平台上实现 CANopen 网络从节点的新方法，并将代码移植到操作系统上，大大提高了代码的应用灵活性，对发展自主版权的现场总线协议标准和协议栈代码具有重要的现实意义。

参考文献

1　CANopen 的开源网站：http://canfestival.sourceforge.net/

2　微型 CANopen 网站：http://www.microcanopen.com/

3　GNU for 68HC11&68HC12 网站：http://stephane.carrez.free.fr/m68hc11_port.php

4　郇极，杨斌，魏继光. 一种开放式的现场总线协议 CANopen. 制造业自动化，2002，10(24)：33－34.

5　邵贝贝. 单片机嵌入式应用的在线开发方法. 北京：清华大学出版社.

6　MC9S12XDP512 Data Sheet ReV. 2.15，Freescale Semiconductor，Inc.

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn