μC/OS－II在LonWorks网络节点的移植研究

邓燕妮，潘 宁
（武汉理工大学 自动化学院，湖北 武汉 430070）

摘 要：LonWorks是一具有强劲实力的现场总线技术，它以Neuron（神经元）芯片为核心构成智能节点。但是神经元芯片存在处理数据的能力不强、实时性也不强的缺点。针对这一缺点，通过外部扩展一个CPU（MCS－51），将神经元芯片的应用CPU中的应用程序移到单片机上，此外还在单片机中嵌入了μC/OS－II嵌入式操作系统，提高了数据的处理能力，也提高了应用系统的实时性。

关键词：LonWorks；神经元芯片；μC/OS－II；嵌入式操作系统
中图分类号：TP273 **文献标识码**：A **文章编号**：1673－629X(2006)10－0243－03

Research to μC/OS－II Transplanting into LonWorks Network Node

DENG Yan-ni, PAN Ning
(Automation College, Wuhan University of Technology, Wuhan 430070, China)

Abstract: LonWorks is a powerful Fieldbus technology, Neuron chip as its center to form intelligent node. However, it has its shortcoming that it doesn't have powerful ability in processing data, and can't respond quickly to event. Aimed at its shortcoming, the paper overcomes its shortcoming by adding a CPU(MCS－51), embedding μC/OS－II into the extended CPU and moving the application originally stored in Neuron chip to the extended CPU.

Key works: LonWorks; neuron chip; μC/OS－II; embedded operating system

1 LonWorks网络节点的构成及功能

一个典型的LonWorks网络节点主要是由神经元芯片、I/O接口单元、收发器等构成[1,2]。每个节点依照固化在Neuron芯片中的LonTalk协议与网络上其它节点通信。节点可以是直接采用Neuron芯片作为通讯处理器和测控处理器，也可以是基于Neuron芯片的HOST BASE节点（如图1所示）、通信介质和通信协议。

由于Neuron芯片是8位总线，目前只支持的最高主频是10 MHz，因此它能完成的功能也有限。对于一些复杂的控制，如带有PID算法的单回路、多回路的控制就显得力不从心。采用MIP（微处理器接口程序）结构是解决这一矛盾的好办法，微处理器接口程序是将神经元芯片作为其它微处理器的通信协处理器的转换固件。将Neuron芯片作为通信协议处理器，用高级主处理器的资源来完成复杂的测控功能。原先运行在Neuron芯片上的应用程序，此时成为了主处理器应用程序。此外，为了提高主处理器的利用率，在该主处理器上嵌入了μC/OS－II嵌入式操作系统，它主要起调度任务的作用，而主处理器应用程序将作为μC/OS－II嵌入式操作系统的任务来运行。在μC/OS－II上还可以运行别的任务，满足实际应用的需要。

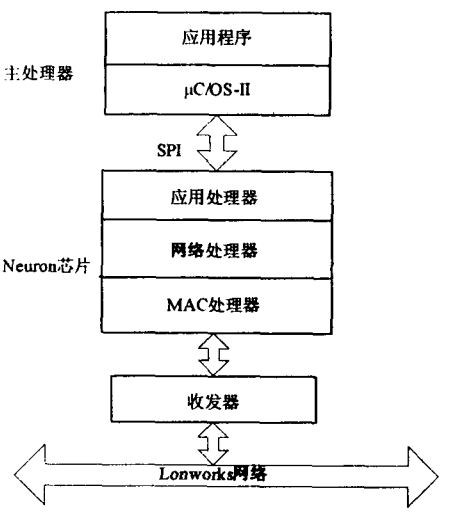

图1 Host Base结构的节点框图

2 硬件接口设计

硬件接口是主处理器（MCS－51）和Neuron3150之间的物理接口。通过编程，MCS－51单片机既支持并行I/O也支持SPI，由于Neuron3150只有11个I/O引脚，SPI系统总线一共只需3～4位数据线和控制线即可实现与具有SPI总线接口功能的各种I/O器件进行接口，而扩展并行总线则需要8根数据线、8～16位地址线、2～3位控制

收稿日期：2006－02－20
作者简介：邓燕妮（1962－），女，湖南人，副教授，研究方向为鲁棒优化控制、计算机控制应用等。

线,因此,采用 SPI 总线接口可以简化电路设计,节省很多常规电路中的接口器件和 I/O 口线,提高设计的可靠性。因此系统采取 SPI 接口。

2.1 SPI 通讯接口信号

将 Neuron3150 的 I/O 3 输入引脚置 1 时,便可选择 SPI 接口通信方式。SPI 方式通信的位速率取决于微处理器服务器的时钟速率和 IO6, IO5 这两个引脚。

SPI 通讯接口通过以下输入输出来实现[3]:

HRDY:主机就绪位(Host Ready)。主机可以使用该位来通知微处理器服务器其是否已经准备好接收数据。

MISO:主机输入从机输出信号(Master Input Slave Output)。用来从主机向微处理器发送控制或数据信号。

MOSI:主机输出从机输入信号(Master Output Slave Input)。用来从微处理器向主机发送控制或数据信号。

SCLK:串行时钟(Serial Clock)。由微处理器使用,用来为所有的数据传送提供时钟同步信号。

TREQ:发送请求标志(Transmit Request)。该位由主机处理器设置,表明其有数据要发送。主机处理器发出该信号后将等待微处理器服务器发出读写信号 R/W。

R/W:读写信号(Read/Write)。由微处理服务器设置该位,来进行读写选择。该信号置低时表示数据由微处理器传向主机,相反则表示数据由主机传向微处理器服务器。

SPI 通信接口如图 2 所示,为了防止在起动和复位的时候不正确的信号被传送,SPI 的输入输出引脚都必须有 $10^4\Omega$ 的上拉电阻。

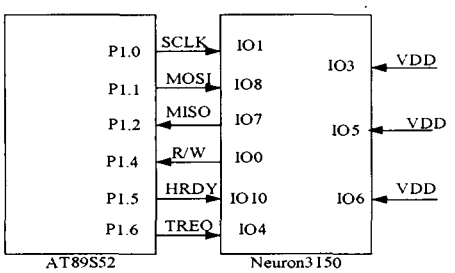

图 2 SPI 通信接口

2.2 SPI 方式通信数据交换时序图

主机处理器必须将主机就绪位 HRDY 置 0,表明其已经准备好接收数据。在 HRDY 置 0 后,微处理器服务器将 R/W 置 0,表明数据由微处理器服务器发往主机。之后微处理器服务器在每个串行时钟信号的下降沿提供数据,主机每个串行时钟信号的上升沿接收数据。在 MOSI 传输过程中,任何发往微处理器服务器的数据将被丢弃。其时序图如图 3 所示。

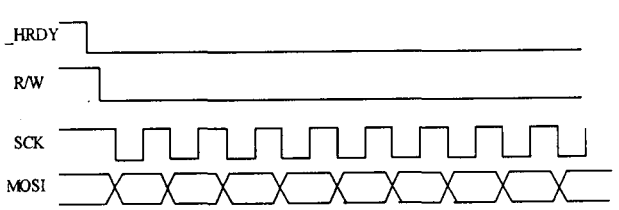

图 3 SPI 方式通信数据交换时序图

3 μC/OS-II 在 LonWorks 网络节点的移植

μC/OS-II 是一个免费的源代码公开的实时嵌入式内核,它提供了实时系统所需的基本功能,包括任务调度、任务管理、时间管理、任务间的通信与同步、内存管理等。其包含全部功能的核心部分代码只占用 8.3kb,而且由于 μC/OS-II 是可裁剪的,所以用户系统中实际的代码最少可达 2.7kb。μC/OS-II 是按占先式多任务系统设计的,总是执行处于就绪条件下优先级最高的任务,最多可以管理 64 个任务,它把连续的大块内存按分区来进行动态管理,可以有效地解决内存碎片的问题。

3.1 μC/OS-II 文件体系

μC/OS-II 源文件的可以分为 3 个部分[4]:

核心代码部分:这部分代码完成操作系统的基本功能,包括 7 个 C 源代码文件和一个头文件。它们是:OS_CORE.C, OS_MBOX.C, OS_MEM.C, OS_Q.C, OS_SEM.C, OS_TIME.C, OS_TASK.C 和 UCOS_II.H。

处理器相关的移植代码部分:这部分代码包括一个头文件,一个汇编文件和一个 C 代码文件:OS_CPU.H, OS_CPU_A.ASM, OS_CPU_C.C,它们是移植中要修改的。

设置代码部分:包括两个头文件 OS_CFG.H, INCLUDE.H,用来进行操作系统配置。

3.2 μC/OS-II 的移植步骤

(1) 移植 OS_CPU.H,进行基本的配置和定义[5]。

OS CUP.H 包括了定义与编译器相关的数据类型、定义使能和禁止中断宏、定义栈的增长方向。用户规划好栈的增长方向后;定义符号 OS_STK_GROWTH 的值。主要内容如下:

```
typedef unsigned char BOOLEAN;        //不要使用 bit 定义,因为在结构体里无法使用
typedef unsigned char INT8U;          //无符号 8 位数
typedef unsigned char OS_ STK;        //栈单元宽度为 8 比特
# define OS ENTER CRITICAL() EA=0     //关中断
# define OS EXIT CRITICAL() EA=1      //开中断
# define OS STK GROWTH 0              //MCU-51 堆栈从下往上增长 1=向下,0=向上
# define OS TASK SW() OSCtxSw()
```

(2) 移植 OS_CPU_A.ASM 汇编代码文件[5]。

OS_CPU_A.ASM 是一个汇编语言源程序,其中的 4 个函数需要根据处理器来修改,汇编是相当底层的编程,需要参考处理器手册。

OSStartHighRdy :这个函数供负责使就绪的任务开始运行的 OSStart 函数调用,这个函数功能是获取新任务的堆栈指针,并从堆栈指针中恢复新任务的所有寄存器。

OSCTxSw:这个函数在任务切换中用到,它负责保存当前任务的寄存器到堆栈中,并将要切换的任务的寄存器从堆栈中恢复出来。

OSIntCTxSw:这个函数在定时中断任务切换中用到,

它负责保存当前任务的堆栈指针,并将要切换的任务的寄存器从堆栈中恢复出来。

OSTickISR:这是时间节拍函数,它通过调用 OSIntExit 函数间接调用 OSIntCTxSw 函数。

它们的主要代码如下:

```
void OSStartHighRdy(void)
{
    调用用户可定义的 OSTaskSwHook();
    获取任务的堆栈指针用于恢复:
    堆栈指针 = OSTCBHighRdy - >OSTCBStkPtr;
    OSRuning = TRUE;
    从新任务的堆栈恢复所有处理器寄存器;
    从中断指令返回;
}
void OSCtxSw(void)
{
    保存处理器寄存器:
    将当前任务的堆栈指针保存到当前任务的 OSJCB 中:
    OSTCBCur - >OSTCBStkPtr = 堆栈指针;
    调用用户定义的 OSTaskSwHooK();
    OSTCBCur = OSTCBHighRdy;
    OSPrioCur = OSPrioHighRdy;
    得到需要恢复的任务的堆栈指针:
    堆栈指针 = OSTCBHighRdy - > OSTCBStkPtr;
    将所有处理器寄存器从新任务的堆栈中恢复出来;
    执行中断返回命令;
}
void OSIntCtxSw(void)
{
    调整堆栈指针来去掉在调用 OSIntExit(),OSIntCtxSw()过程中压入堆栈的内容;
    将当前任务的堆栈指针保存到当前任务的 OSTCB 中;
    OSTCBCur - >OSTCBStkPtr = 堆栈指针;
    调用用户定义的 OSTaskSwHooK();
    OSTCBCur = OSTCBHighRdy;
    OSPrioCur = OSPrioHighRdy;
    得到需要恢复的任务的堆栈指针:
    堆栈指针 = OSTCBHighRdy - >OSTCBStkPtr;
    将所有处理器寄存器从新任务的堆栈中恢复出来;
    执行中断返回命令;
}
void OSTickISR(void)
{
    保存处理器寄存器;
    调用 OSIntEnter()或者直接将 OSIntNesting 加 1;
    调用 OSTimeTick();
    调用 OSIntExit();
    恢复处理器寄存器;
    执行中断返回指令;
}
```

(3) 裁减不必要的内容(OS_CFG.H)[5]。

例如要去掉消息队列的相关功能,则

```
#define OS_MAX_QS     0   //使消息队列的最大值为 0
#define OS_Q_EN       0   //关闭使能
```

(4) 移植 OS_CPU_C.C 标准 C 代码文件[5]。

这个源文件中有 6 个函数要移植:

```
OSTaskStkInit()
OSTaskCreateHook()
OSTaskDelHook()
OSTaskSwHook()
OSTaskStatHook()
OSTimeTickHook()
```

5 个带"Hook"的是 Hook 函数,即所谓钩子函数,用来扩展 μC/OS-II 的功能,如果不进行扩展,可以不包含任何代码,但必须声明。OSTaskStkInit()是必须根据处理器来移植的,它在任务创建时被调用,负责初始化任务的堆栈结构。

```
void * OSTaskStkInit (void ( * task)(void * pd), void * ppdata. void * ptos, INT16U opt) reentrant     //初始化任务堆栈
{
    根据设置的任务栈进行编程
}
```

由于 KEIL 编译器的特殊性,有些代码要多处改动。因为 KEIL 默认情况下编译的代码不可重入,而多任务系统要求并发操作导致重入,所以要在每个 C 函数及其声明后标注 reentrant 关键字。

4 总 结

LonWorks 技术已经在工业、楼宇、家庭、能源等领域得到一定程度的应用,而且有其它控制网络技术不可比拟的优越性,但是由于 Neuron 芯片的应用处理器的数据处理能力有限,通用 I/O 相对较少,在一定程度上限制了 LonWorks 技术的广泛应用。文中通过扩展 Neuron 芯片的处理能力,移植 μC/OS-II 嵌入式操作系统,可以在扩展的 CPU 上运行多个任务,各个任务并行操作,充分利用了 CPU 的资源,从而大大减少系统的出错率,提高系统的实时性,并且大大地简化了编程工作。

参考文献:

[1] 凌志浩.从神经元芯片到控制网络[M].北京:北京航空航天大学出版社,2002.148-149.

[2] 马 莉.智能控制与 Lon 网络开发技术[M].北京:北京航空航天大学出版社,2003.84-95.

[3] 杨金岩,郑应强,张振仁.8051 单片机数据传输接口扩展技术与应用实例[M].北京:人民邮电出版社,2005:155-158.

[4] LABROSSE J.嵌入式实时操作系统 μC/OS-II[M].邵贝贝译.北京:北京航空航天大学出版社,2003:283-289.

[5] 叶丰桥,黄 海.μC/OS-II 在 51XA 上的移植应用[J].工业控制计算机,2002(10):55-56.