

异步串行口-以太网设备服务器

... 对于许多设备来讲, 串行口是它们与外部世界通信的唯一渠道。

难以计数的电子设备通过串口和其它设备通信。事实上, 很多情况下, 串行口是它们与外部世界通信的唯一渠道。例如自动调温器, 销售点系统, 远端监视器, 条码阅读器, 票据打印机, RFID收发器, 血压计, 以及许许多多千差万别的现场设备, 从传统的检测工具到最新的楼宇自动化。这些设备没有直接参与大型计算机网络的手段, 但新的应用需要TCP/IP的连通性和以太网的通信能力。然而, 昂贵的成本和耗时的设计常常又令人望而却步。

本文介绍一种简易、经济的办法, 通过在TINI平台(该平台基于DS80C390或DS80C400微控制器实现)上建立一个应用层, 可将孤立的串行设备迁移到以太网。一旦将设备连接到以太网, TINI web服务(例如HTTP服务器)的实现便水到渠成。

RS-232 串口

本文所讨论的异步串行通信基于RS-232-C标准, 可一直追溯到计算机历史的早期。RS-232-C发布于1969年¹。现在的大多数串行口不支持标准所规定的所有信号。此外, 已实现的信号也只是以一种“相当接近”于标准的方式在使用。我们将不考虑纯粹的历史定义, 而只关注当前RS-232的使用方式。

空号和传号

RS-232-C规定“空号”(二进制0)为一个+3V至+25V的电平, “传号”(二进制1)为-3V至-25V。-3V和+3V之间的区域为“切换区”。很多通用异步收发器(UART)采用更现代的(相对来说)TTL电平0V和+5V表示0和1。专用的电平转换器, 例如著名的MAX232, 可以实现TTL和RS-232电平的转换。因为DS80C390/DS80C400的串行口是TTL兼容的, 因此在和其它TTL电平的UART接口时不必采用电平转换器。

DCE 和DTE

数据通信设备(DCE)和数据终端设备(DTE)是一条通信信道的两个端点。它们之间的一个主要差别是串行连接器的插脚定义。通过一个所谓的空调制解调器可在它们两者之间进行转换。

表1列出了DTE串行连接器DB-9上的信号定义, 以及另外一个采用空调制解调器的DTE上的对应信号。

流控制

串行通信时, 你可以向一条芯线(TD)发送而从另外一条(RD)接收。然而, 如果两个采用RD/TD通信的设备任意进行发送, 其中一个有可能超速另一个而导致数据丢失。通常采用以下两种办法之一实现必要的流控制:

- XON/XOFF(常常简单地称为软件流控制)
- RTS/CTS(常常简单地称为硬件流控制)

¹这个时期NASA有了用于故障解释的计算机纸带, 因此这种比喻是有效的。

DTE 芯	信号名	空调制解调器
1	CD (载波检测)	4 (DTR)
2	RD (接收数据)	3 (TD)
3	TD (发送数据)	2 (RD)
4	DTR (数据终端就绪)	6 (DSR) 和 1 (CD)
5	公共端 (信号地)	5 (公共端)
6	DSR (数据设备就绪)	4 (DTR)
7	RTS (请求发送)	8 (CTS)
8	CTS (清除发送)	7 (RTS)
9	RI (振铃指示)	—

表 1. 空调制解调器可被用来连接两个 DTE 的 DB-9 串口的对应信号。

XON/XOFF 流控方式通过发送带内字符使另一侧暂停 (XOFF, 13h) 或继续 (XON, 11h) 发送。如果 XON 和 XOFF 字符出现在二进制数据流中, 则发送方软件忽略它而由接收方软件打开。

RTS/CTS 需要使用额外的信号线。RTS (请求发送) 由发送方发出。如果接收准备就绪, 接收方以 CTS 回应 (清除发送), 当其接收缓冲器满时就清除 CTS。

有些设备支持流控制而有些不能。于是, 默认设置常常为“没有流控制”, 如果已知某个设备执行流控制, 该设置将不起作用。

速度, 数据位, 停止位和奇偶位

为了通信成功, 还必须设置发送速度 (位速率), 数据位数和停止位, 以及奇偶校验类型 (如果有的话)。大多数新设备采用一种“8N1”设置, 代表 8 个数据位, 无奇偶位, 和一个停止位。然而, 传统系统各种可能都有, 因此, 正确地进行设置还并非微不足道。

TINI 和网络

TINI 是 Dallas Semiconductor 开发的一种技术平台, 目的是协助用户快速整合 DS80C390 和 DS80C400 网络微控制器到其目标应用中。TINI 定义了一个芯片组, 并包含一个嵌入式操作系统, 其中整合了经过高度优化的 Java 运行环境。Java 编程者可从其中获得一般的嵌入式开发中不多见的强大功能: 多线程, 无用单元收集, 继承性, 虚拟化, 跨平台能力, 强大的网络支持, 以及——最后但很重要——大量免费的开发工具。TINI 使用者通常不直接面对汇编代码。不过, 为了优化严格要求速度的通道或者访问底层硬件, 同时也支持并鼓励本地语言子程序 (TINI 操作系统用本地代码写成, 因此串行 I/O 的吞吐率和现代 PC 没有明显差异)。

TINI 定义了一个芯片组, 并包含一个嵌入式操作系统, 其中整合了经过高度优化的 Java 运行环境。

除了完全支持 *java.net* 包之外, TINI Java 运行环境还包含一个完整实现的 *javax.comm* 子系统。通过 Java 可毫不费力地访问 TCP/IP 和串行口, 因此 TINI 系统可非常容易地用来实现串行口-以太网桥。

下面的例子所用到的 TINI_m390 验证模块 (可插入 E10 插座) 是 DS80C390 TINI 开发平台的硬件部分 (TINI_m400 采用 DS80C400)。除了 SRAM, 闪存, 以太网, CAN 总线, 1-Wire 等, 系统还具有四个串行口。两个 UART 位于 DS80C390 内部 (称为 *serial0* 和 *serial1*); 两个端口在外面 (采用一片 16550 选配件)。需要注意的是, E10 插座上的两个串行连接器都被接到了 *serial0*, 它们只是在 DTE/DCE 引脚安排上有所差异。

有关 TINI 环境的详细文档参见 *The TINI Specification and Developer's Guide* (Addison-Wesley, 2001)。可从 www.maxim-ic.com/TINIguide 下载免费的 PDF 拷贝。

举例

下面我们开始介绍两个具体应用, 并从一个普通的串行口-以太网程序中摘录片段出来, 经过修改, 它可适应于几乎各种应用。这些范例利用 TINI_m390/400 验证模块搭建而成。

TINI 验证模块作为一个“黑匣子”被用来连接多个串行设备到以太网。根据最终设备的需要，TINI 可以执行简单的数据传递，或者也可以对数据流进行解析、翻译或调整(图1)。

尽管你可以在 TINI_m390/400 的开发者外壳上运行这些范例，更合理的做法是将其驻留于闪存之中，掉电之后还能够自启动，并利用其他的一些 TINI 构造技术使最终产品更加牢靠。

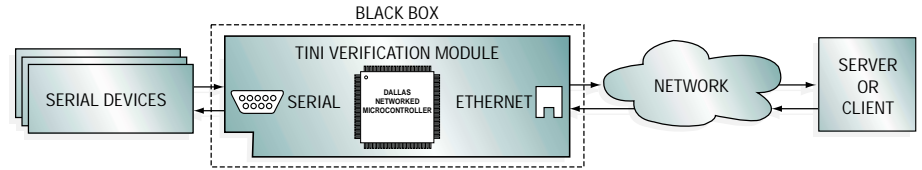


图1. TINI 设备服务器被用作串行设备和以太网之间的桥梁。

如果想修改这些范例，需要具备一些基本的网络知识和编程经验。样例工作代码也可从 Dallas 的 ftp 站点(<ftp://dalsemi.com>) 下载。

虚拟调制解调器

第一个范例——“虚拟调制解调器”³，利用 TINI_m390/400 和 TCP/IP 连接，替代物理调制解调器和电话线。假定有一个旧设备，比如某工厂的“机器状态监视器”，它利用一个调制解调器，一天之内数次拨号到一个中央服务器，报告机器的状态、负载和效率数据。为了削减服务器侧日益增长的调制解调器库，并利用现有的 LAN 取代连接到设备的电话线，我们可以：

- 重写服务器软件以支持 TCP/IP，并且
- 用 TINI 虚拟调制解调器取代每个机器上原有的调制解调器。

这样，机器状态监视器就不必再作任何修改，对于最终设备而言，虚拟调制解调器的使用和一个真正的调制解调器完全一样！

除了上述配置，虚拟调制解调器当然也可以成对使用。如果双方都使用虚拟调制解调器，就无须对服务器软件作任何改动，TINI 模块可直接替换现有的调制解调器。

在此表象之下，虚拟调制解调器每次接到“ATD”调制解调器拨号命令时，实际建立的是 TCP 连接。“ATH”断开命令关闭 TCP 连接。软件也可实现一系列其他的经典 AT 调制解调器命令，并被诸如 Microsoft® Windows® 之类的网络系统认作真调制解调器。此外，虚拟调制解调器还可以侦听 TCP 端口，当收到“呼叫”信号时，向终端设备发出“振铃”。

下面的代码片段显示了如何对 TINI_m390 上的串行口进行初始化：

```
public static void main(String args[])
{
    TINIOS.setSerialBootMessagesState(false);
    TINIOS.setDebugMessagesState(false);
    TINIOS.setConsoleOutputEnabled(false);
    System.out.println("Connecting to serial0 at 9600bps, "
        "listening on TCP port 8001");
    try {
        CommPortIdentifier portId =
            CommPortIdentifier.getPortIdentifier("serial0");
        SerialPort port = (SerialPort) portId.open("VModemTINI",
            10000);

        TINIOS.setRTSCTSFlowControlEnable(1, false);
        TINIOS.setRTSCTSFlowControlEnable(0, true);
        TCPSerialVirtualModem modem = new
            TCPSerialVirtualModem(port,
```

除了完全支持 *java.net* 包之外，TINI Java 运行环境还包含一个完整实现的 *javax.comm* 子系统。通过 Java 可毫不费力地访问 TCP/IP 和串行口，因此 TINI 系统可非常容易地用来实现串行口-以太网桥。

³ 参见 *Application Note 196: Designing a Virtual Modem Using TINI* (在 www.maxim-ic.com 上)。

```

        /* Comm speed */ 9600, /*TCP Port */ 8001);
    modem.processInput();
}
catch (Exception e) {
    System.out.println("Exception: "+e.toString());
}
}

```

这段代码首先禁止掉所有来自 TINI 操作系统的调试输出——TINI 上的标准惯例。获得一个端口号后，打开这个端口(如果端口正在被另一个应用使用，第二个参数指明等待多久)。接下来，设置硬件流控制状态。由于 TINI390 仅有一套 RTS/CTS 线用于串口 0 和 1，在目标端口使用它们之前，程序首先应该关掉其它端口上的流控制。下面，就是一个 Java 虚拟调制解调器范例。

虚拟调制解调器类包含一个 AT 命令解释器(未在此示出，尽管到目前为止，它是本范例中最大的一部分)和网络代码。下面的代码用来设置串口位速率，数据和停止位，以及奇偶位，从中可以看出，处理入站连接是何等简单：

```

/** Creates a new VirtualModem connected to a serial port on
 * one end and a TCP port on the data side.
 * serial -- the serial port this VirtualModem talks to.
 * speed -- the speed the serial port should be set to.
 * tcpport -- the TCP port this VirtualModem listens on.
 * throws IOException when there's a problem with the serial
or TCP port.
 */
public TCPSerialVirtualModem(SerialPort serial, int speed, int
tcpport)
    throws IOException
{
    super(serial);

    try {
        serial.setSerialPortParams(speed, SerialPort.DATABITS_8,
SerialPort.STOPBITS_1,
SerialPort.PARITY_NONE);
    }
    catch (UnsupportedCommOperationException e) {
        throw new IOException();
    }
}
...

serverSock = new ServerSocket(tcpport, 1); // backlog of one
listenThread = new listenInbound();
listenThread.start();
}

```

最后，以下 listenThread() 片段可接受一个到来的连接请求：

```

public void run()
{
    int rc;
    Socket s;
    while (running) {
        s = null; // No incoming connection request
        try {

```

```

answered = false;
s = serverSock.accept();

// Discard incoming connection if already connected
if (connected)
    throw new IOException();

sock = s; // for answer()
...

```

UPS 监视器

第二个实例是将 TINI⁴ 连接到一个不中断电源的串口。软件采用网络 UPS 工具协议⁴，允许多个客户端在多种平台上检测 UPS 的状态。该项目源于需要由一台没有串口的新 Macintosh 计算机监视现有的 UPS 电源的需求。

目前存在两种基本的 UPS 设备：即所谓的“智能型”和简单型(或“哑巴型”)。简单的 UPS 在多个串行引脚上指示其工作状态，它实际上不输出任何 ASCII 数据。由于不存在太多的串行引脚，因而它仅能够指示几组有限的信息。例如：

信号	意义
RTS (来自 UPS)	电池低
TD (来自 UPS)	电池通
CTS (到 UPS)	切断 UPS 电源

TINI 验证模块可作为一个“黑匣子”，连接多个串行设备到以太网。根据最终设备的需要，TINI 可以执行简单的数据传递，或者也可以对数据流进行解释、翻译或调整。

javax.comm.notifyOn...() 方法可在 Java 中简单实现响应状态改变的代码，例如：

```

...
// Listen for DTR changes
try {
    port.addEventListener(this);
} catch (TooManyListenersException e) {
    ...
}
port.notifyOnDSR(true);
...

public void serialEvent(SerialPortEvent ev)
{
    try {
        if (ev.getEventType() == SerialPortEvent.DSR)
            ...
    } catch ...
    ...
}

```

智能型 UPS 更有趣一点，因为它执行一个串行协议，能够返回电池充电的百分比或温度等参数。在不同的生产商之间，通信协议相差悬殊，且通常没有文档记录。Dallas ftp 站点上的 UPSMonitor 范例支持 APC SmartUPS，但经过简单调整后即可适用于其它品牌。

⁴ 见 www.exploits.org/nut/。

下列代码展示了如何接收UDP请求和如何通过UDP发送UPS的状态信息。

```
// Listen to incoming UDP requests
private class listenUDPThread extends Thread
{
    private DatagramSocket sock;
    private byte[] buffer;
    private DatagramPacket dp;

    public listenUDPThread(DatagramSocket s)
    {
        sock = s;
        buffer = new byte[BUF_SIZE];
        dp = new DatagramPacket(buffer, buffer.length);
    }

    public void run()
    {
        while (running) {
            try {
                sock.receive(dp);
                byte[] data = parseCommand(buffer, dp.getLength());
                sock.send(new DatagramPacket(data, data.length,
                    dp.getAddress(), dp.getPort()));
            }
            catch (Exception e) {
            }
        }
        try {
            sock.close();
        }
        catch (Exception e) {
        }
    }
}
```

... 串口和 TCP 端口被抽象为 `Input/OutputStreams dataIn` 和 `dataOut` ...
在 CAN 和 1-Wire 间桥接数据...

归功于Java内置的强大网络支持，本实例几乎不需要其它解释。在while()循环体内的代码一直等待，直至接收到一个UDP请求，然后解析这个请求，利用getAddress()从输入数据包中获取请求者，向请求者发送一个回应。

通用串口-以太网应用

关于完整的串口-以太网实例的讨论超出了本文范畴(完整的实例示范和解释参阅*The TINI Specification and Developer's Guide*)。不过，下列部分代码还是展示了在串口-以太网桥的串口和网络之间，如何高效地利用多线程传递数据。串口和TCP端口被抽象为Input/OutputStreams dataIn和dataOut，因而，这一层的代码实际上不必关心任何有关网络的情况，而且，它还可以在CAN和1-Wire间桥接数据。

```
public GenericBridge()
{
    ...
    running = true;
    dcThread = new dataCopy();
    dcThread.start();
}
```

```

// Thread that copies everything from dataIn to dataOut
private class dataCopy extends Thread
{
    public void run()
    {
        int r = 0;
        while (running && r >= 0) {
            try {
                synchronized (threadLock) {
                    r = dataIn.read(dataBuffer);
                    if (r > 0)
                        dataOut.write(dataBuffer, 0, r);
                }
            }
            catch (Exception e) {
                r = -1;
                ... // Handle error
            }
        }
    }
}
}

```

结论

许多旧设备仅支持异步串行通信，然而，当前的许多应用要求以太网和TCP/IP组网能力。利用强大的Java运行环境和基于DS80C390和DS80C400微控制器的TINI技术，很容易在数小时之内开发出一个串行口-以太网转换器。

Microsoft和Windows是Microsoft Corp.的注册商标。