Introdução ao padrão físico RS-485 para comunicação serial

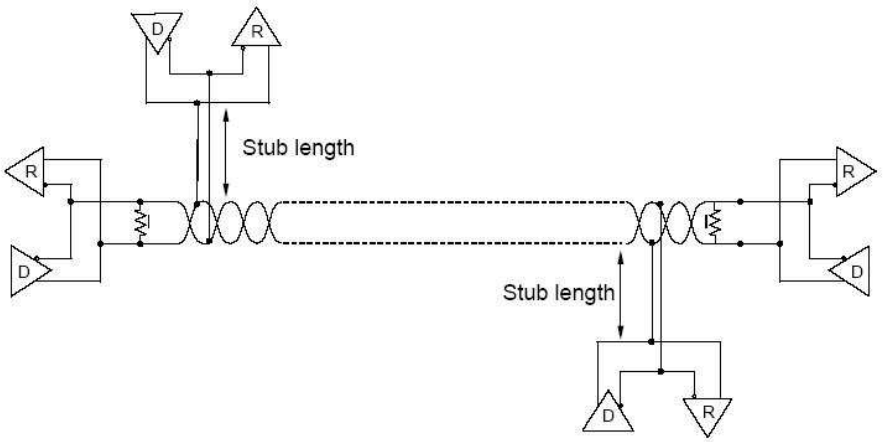
Alexandre S. Martins, Geovany A. Borges

Laboratório de Controle e Visão por Computador (LCVC)
Departamento de Engenharia Elétrica - ENE
Faculdade de Tecnologia - FT
Universidade de Brasília - UnB
Brasília - DF - Brasil

5 de junho de 2006

Resumo

Esta nota técnica apresenta de uma forma geral as características e o princípio de funcionamento do padrão físico RS-485 para comunicação serial. Um enfoque maior é dado na aplicação do mesmo em transferência de dados utlizando microcontroladores e PCs, tanto no interfaceamento físico (circuitos, conversores, cabos, etc.), como também no software (protocolo e configurações).



Revisões:

03/06/2006 Primeira versão da nota técnica sobre RS-485

1 Introdução

Muitos projetos de sistemas baseados em microprocessadores requerem comunicação entre dispositivos. Esta comunicação pode ser efetuada de forma paralela ou serial, desde que os dispositivos possuam meios para isto. No caso específico de comunicação serial, o dispositivo deve ser dotado internamente de uma UART (*Universal Asynchronous Receiver Transmiter*), USART (*Universal Synchronous-Asynchronous Receiver Transmiter*) ou qualquer controlador de comunicação serial. Estes controladores possuem pinos de entrada/saída pelos quais ocorre o fluxo e controle de informações. Em geral, os níveis de tensão nos pinos são baixos, de acordo com a alimentação. Se a distância entre dispositivos for pequena, da ordem de algumas dezenas de centímetros, e o nível de interferência eletromagnética for baixo, pode-se fazer a conecção direta entre os pinos dos dispositivos (em geral ponto-a-ponto). Entretanto, quando se deseja realizar comunicação a mais longas distâncias, faz-se necessário empregar um padrão físico de comunicação serial. Um destes padrões é o RS-232, que é mais comumente utilizada para comunicação serial para curtas distâncias, a baixas velocidades de comunicação. Para maiores informações sobre RS-232 e porta serial de microcomputadores PC, sugere-se o site `http://www.beyondlogic.org/serial/serial.htm`. Por outro lado, o padrão RS-485 é capaz de prover uma forma bastante robusta comunicação multiponto, bastante comum na indústria, em controle de sistemas com taxas podendo ultrapassar 10 Mbps, e distâncias podendo alcançar centenas de metros.

Este documento é direcionado para quem deseja entender como o RS-485 funciona, bem como para aqueles que pretendem desenvolver qualquer aplicação com computadores e/ou microcontroladores utilizando comunicação RS-485. A seção seguinte descreve o RS-485, comparando-o com o padrão RS-232. A Seção 3 traz alguns transceptores comerciais para comunicação em RS-485, seguida de uma proposta de conversor simples entre RS-232 e RS-485 na Seção 4. Aspectos de comunicação multiponto e protocolo são discutidos na Seção 5. Por fim, no Apêndice A é mostrado o código finte de uma biblioteca com funções em linguagem C para comunicação serial com RS-485 em Linux.

2 O padrão RS-485

A principal diferença entre RS-485 e RS-232 está que o RS-485 é um padrão diferencial, enquanto que o RS-232 é referenciado ao comum (0V). No RS-232, o nível lógico 1 é interpretado como sendo qualquer tensão no intervalo [–15V;3V], enquanto que tensões no intervalo [3V;15V] correspondem ao nível lógico 0. Tensões entre -3V e 3V não possuem nível lógico definido. Este tipo de interface é útil em comunicações ponto-a-ponto a baixas velocidades de transmissão. Entretanto, devido à grande faixa de variação dos sinais, faz-se necessário dispor de *drivers* de alto *slewrate* para se alcançar altas velocidades de comunicação. No mais, com o aumento do comprimento do cabo de comunicação, o padrão RS-232 se torna altamente susceptível a inteferência eletromagnética e a retorno de sinal.

O RS-485 utiliza um princípio diferente. Um transceptor RS-485 traduz um sinal lógico TTL em dois sinais, denominados de A e B (ver Figura 1). O sinal A possui a mesma lógica do sinal TTL, enquanto que o sinal B é complementar. A informação do sinal de entrada está condificada na forma do sinal A-B, ou seja, da diferença entre os sinais A e B. Se esta diferença for superior a 200mV, então tem-se nível lógico 1. Caso a diferença seja inferior a -200mV, então considera-se nível lógico 0. No intervalo de -200mV a 200mV o nível lógico é indefinido, servindo também como meio de detecção de cabo solto, para alguns transceptores comerciais. Entretanto, deve-se ter algum cuidado com relação ao compartilhamento de referência de 0V entre dispositivos [1].

Uma das vantagens da transmissão diferencial é sua robustez a interferência eletromagnética. A conexão entre dispositivos RS-485 é feita por cabos de par trançado [3], com resistores de terminação para balanceamento. Dessa forma se um ruído é introduzido na linha, ele é induzido nos dois fios de modo que a diferença entre A e B é quase nula. Outra vantagem da transmissão diferencial é que diferentes potenciais de terra são, até certo ponto, ignorados pelos transmissores e receptores. Isso se torna importante quando tem-se que percorrer grandes distâncias ou mesmo em sistemas com

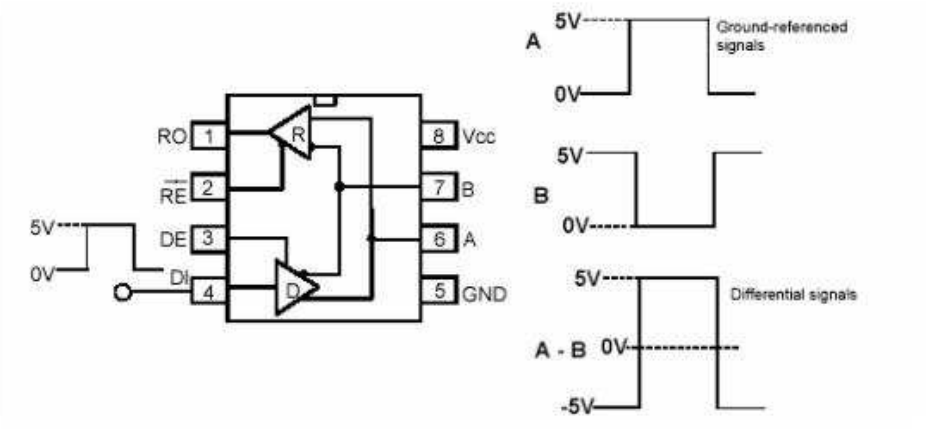


Figura 1: Diagrama dos transceptores DS485 e ST485, e sinais diferenciais [2].

diferentes fontes de alimentação. Cabos trançados com terminações corretas que minimizam reflexão do sinal permitem taxas de transferência de 10Mbps a distâncias de até 1 km.

O RS-485 é um padrão de comunicação multiponto, permindo-se a conexão de até 32 dispositivos num simples cabo de par trançado. Dependendo do transceptor, pode-se conectar mais de 200 dispositivos, seguindo a topologia mostrada na Figura 2. Na verdade, a Figura 2 mostra a arquitetura mais comum de utilização do RS-485. A letras D e R indicam os drivers de transmissão e recepção de cada dispositivo, respectivamente. Uma observação a ser considerada é a impedância característica do cabo. Se o cabo for utilizado para transmissões com componentes espectrais a altas freqüências, podem ocorrer reflexões do sinal em sua extremidade provocando inconsistência nos dados transmitidos. Para minimizar este efeito, deve-se adicionar resistores de terminação de valores iguais à impedância característica do cabo para que ele se comporte como um cabo infinito. Os valores típicos para essa resistência é de cerca de 120Ω para cabos trançados, e de cerca de 54Ω para cabos blindados. Apesar de típicos, estes valores podem ser diferentes pois dependem também dos requisitos de carga mínima dos transmissores. É importante também que hajam apenas dois resistores, um em cada extremidade, que podem estar também dentro do último dispositivo conectado. Na prática, porém, para pequenas distâncias e baixas velocidades, a terminação não chega a ser algo crucial e a maioria dos circuitos funciona bem. Um outro artifício para minimizar a influência da reflexão dos sinais é por meio da rdução forçada da banda passante. Isto já é feito em vários transceptores, por meio de uma limitação do *slew-rate*.

A maioria dos sistemas com RS-485 utiliza uma arquitetura mestre/escravo para comunicação, onde apenas um único dispositivo (geralmente o PC), chamado mestre, envia periodicamente mensagens endereçadas aos escravos. Cada escravo por sua vez tem um único endereço e responde apenas a pacotes endereçados a ele. Além do mais, pode-se usar USARTS half-duplex ou full-duplex, como ilustrado pela Figura 3. Para tanto, fisicamente as seguintex conexões devem ser feitas:

- Half-duplex: um único par trançado, em que todos os dipositivos estão conectados no memo cabo trançado. Dessa forma, todos eles devem possuir transceptores com saídas *tri-state*, incluindo o mestre. A comunicação se dá em ambas direções e é importante evitar por software que mais de um dispositivo tenha o seu driver da transmissão habilitado ao mesmo tempo.

- Full-duplex: usa-se dois pares trançados, em que os escravos transmitem para o mestre através do segundo par trançado. Essa solução geralmente permite comunicação multiponto em sistemas que foram originalmente projetados para RS-232 com pequenas modificações no software mestre. Outro fato é que o mestre também não precisa colocar a saída do seu transceptor em estado de alta impedância.

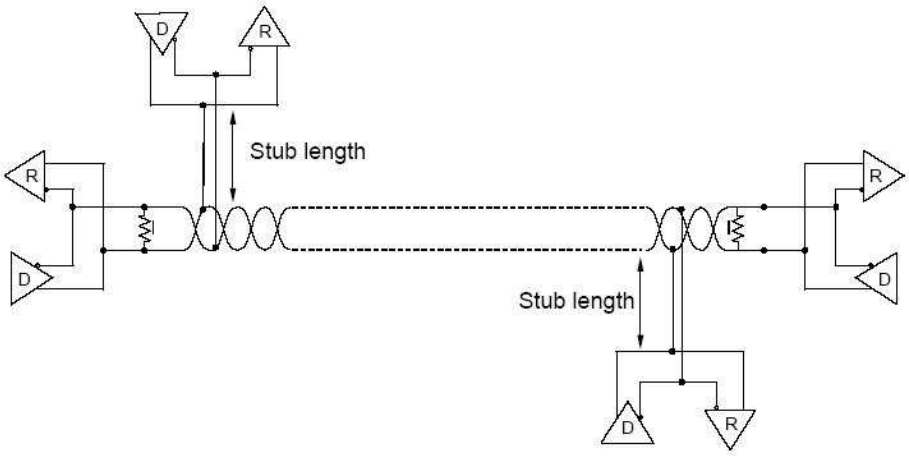
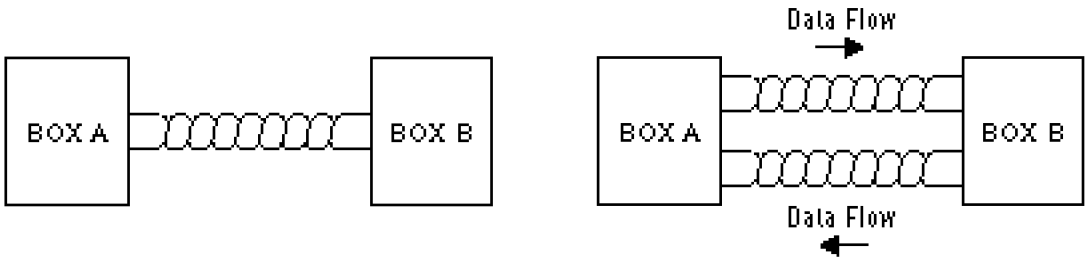


Figura 2: Barramento RS-485 típico [2].



(a) half-duplex (b) full-duplex

Figura 3: Conexões físicas RS-485 para half-duplex e full-duplex [4].

3 Transceptores RS-485

É possível encontrar no mercado vários transceptores de RS-485 e até circuitos prontos para determidas aplicações. No LCVC tem-se utilizado os integrados DS485 e ST485, da National Semiconductor e STMicroeletronics, respectivamente. Estes dispositivos são bastante populares, devido principalmente ao baixo custo. Entretanto, dispositivos mais caros como o LTC485 (Linear Technology) e o MAX485 (Maxim), possuem outras características, tais como maiores taxas de comunicação e número de dispositivos. Estes mesmos fabricantes possuem uma linha mais extensa de dispositivos, com características de full-duplex, opto-isolamento, conversão RS232/485, etc.

A Figura 1 mostra o diagrama dos circuitos integrados DS485 e ST485. A diferença básica entre eles está somente nas suas características elétricas, bem como de taxas de velocidade máxima: até 2,5Mbps para o DS485 e até 30 Mbps para o ST485. O pino **RE** habilita o *driver* de recepção **R**, sendo ativo no nível 0. O pino **DE** habilita o driver de transmissão **D** (ativo em 1). Normalmente esses dois pinos são conectados juntos de forma que o tranceptor esteja apenas recebendo ou transmitindo. Os pinos **RO** e **DI** representam saída da recepção e driver de entrada respectivamente, e trabalham com níveis lógicos TTL (0 a 5V). Já as saídas **A** e **B** para o barramento operam com tensão diferencial entre seus terminais. O transceptor normalmente é alimentado com 5V CC.

Para que um dispositivo transmita um dado pelo barramento, é necessário elevar para 5V o pino **DE**, fazendo com que **RE** seja desabilitado, para então transmitir a informação necessária pelo pino **DI**. Com o final da transmissão, deve-se desabilitar **DE** e reabilitar **RE**, de forma que o transceptor volte ao modo de recepção. Esta habilitação pode ser feita via software, controlando pinos de um microcontrolador ou de uma porta de comunicação de um microcomputador, ou mesmo através de um hardware construído especialmente para detectar início de transmissão para o formato de dados utilizado (e.g., UART).

Um problema que pode ocorrer com um barramento RS-485 que dipõe apenas de resistores de terminação é que, quando todos os dispositivos estão em modo de recepção, o nível lógico do barramento pode ficar indefinido [2]. Para garantir que o barramento fique sempre em nível lógico 1, que corresponde ao estado default NRZ das USARTS conectadas ao barramento, deve-se adicionar um resistor *pull-up* ao pino A e um de *pull-down* no pino B, conforme ilustra a Figura 4. Esse resistores devem ter valores iguais para não alterar o balancemento da linha de transmissão.

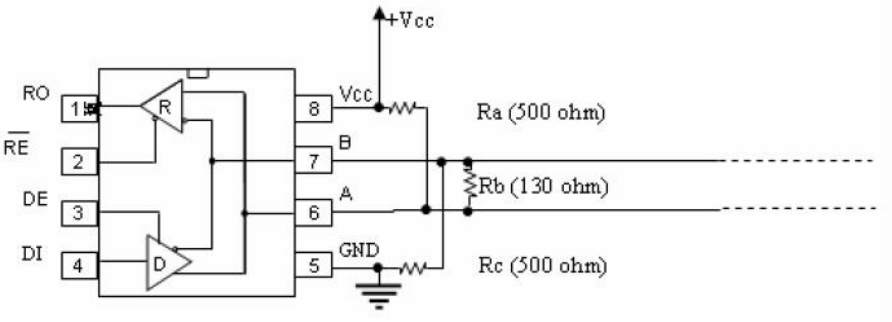


Figura 4: Pull-up e pull-down para evitar sinais indefinidos no barramento [2].

4 Um simples conversor RS-232/RS-485 para PC

Em muitos casos, em uma rede RS-485, um dos dispositivos da rede pode ser um microcomputador PC. Vários projetos desenvolvidos no LCVC fizeram uso de microcomputador PC como mestre do barramento RS-485. As razões para isto vão desde a necessidade de coleta de dados para análise no MATLAB [5] a até o controle avançado em tempo real [6]. Para tanto, faz-se necessário dispor de

uma transceptor RS-232/RS-485, uma vez que as portas seriais de microcomputadores PC usam o padrão físico RS-232, inclusive para controle de fluxo. Este transceptor, cujo esquemático é mostrado na Figura 5, utiliza um circuito integrado DS485 (conversor RS-485/TTL) e um circuito integrado MAX232 (conversor RS-232/TTL). A habilitação de transmissão pelo barramento RS-485 seria feito por meio do sinal **RTS** da porta serial do PC, também utilizando o MAX232 como interface. Se o microcomputador ficar numas das extremidades do barramento, pode-se incluir, por meio de *jumpers*, o resistor de terminação e também, se for o caso, os resistores de pull-up e pull-down. A seção seguite discutirá um software para realizar esse controle de escrita e leitura no PC em Linux.

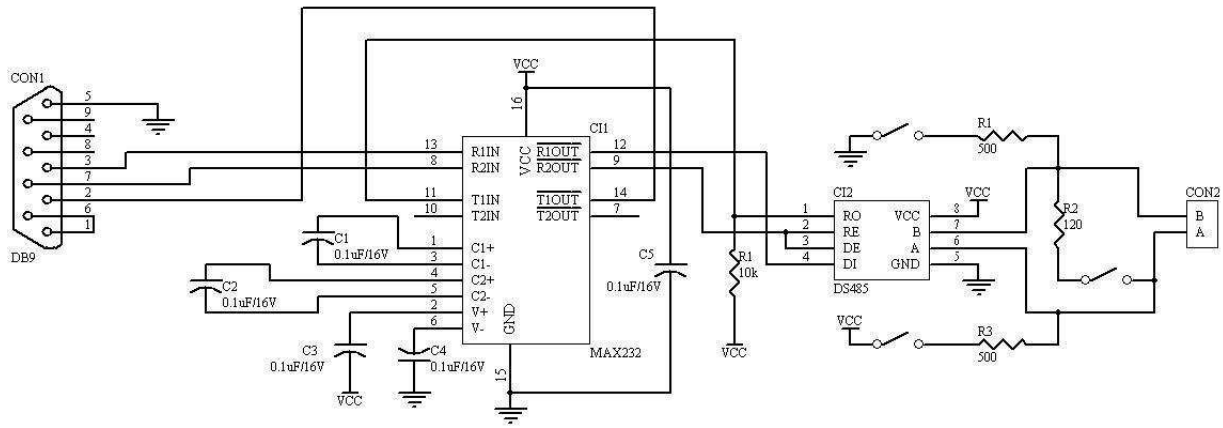


Figura 5: Conversor RS-232/RS-485 com controle pelo pino RTS.

5 Protocolos e configurações de software

Pelo fato da grande maioria dos dipositivos possuírem UARTs e quando taxas de até 115200 bps são suficientes para a aplicação em questão, o formato de dados NRZ comum em UARTs é uma boa opção para sincronização de bits, quando da utilização de transmissão assíncrona (ver Figura 6). De acordo com o modelo OSI, este é ainda um aspecto da parte física de uma rede de comunicação, e que ainda não garante a idoneidade dos dados transmitidos. Para tanto, faz-se necessário ainda empregar um protocolo de comunicação. Quando se refere a sistemas embarcados microcontrolados, estes protocolos implementam em geral a camada de enlace. Um protocolo bastante empregado na indústria é o MODBUS. No LCVC, muitos trabalhos esão sendo realizados a partir do protocolo descrito em [5].

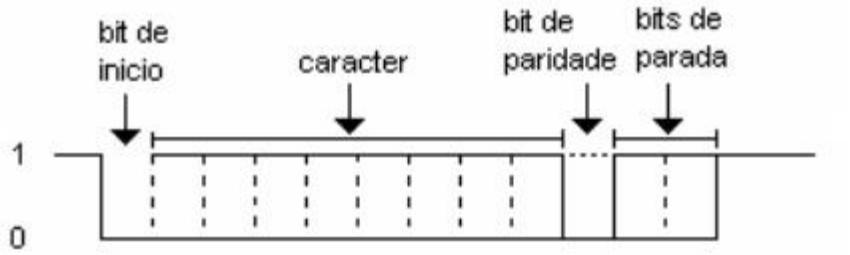


Figura 6: Formato de dados UART.

5.1 Considerações para o projeto de protocolos

Quando deseja implementar seu próprio protocolo, um projetista deve levar em conta alguns aspectos. Na verdade, a definição do protocolo depende fortemente da aplicação. Se for utilizada a arquitetura mestre/escravo, em que cada escravo tem um endereço único e responde apenas a pacotes endereçados a ele, o protocolo torna-se simplificado, visto que elimina a necessidade de algoritmos de detecção de colisão, re-transmissão e algoritmos complicados de controle de acesso ao meio presentes em alguns sistemas distribuídos. Mesmo assim, o formato das mensagens deve levar em conta a aplicação. Por exemplo, se as informações de interesse trafegam no sentido dos escravos para o mestre, o mestre pode apenas enviar um byte contendo o endereço e um comando para o escravo, e este poderá responder com um ou mais bytes. Uma sugestão quando se tem mensagens que variam de tamanho é a utilização de um cabeçalho de um ou dois bytes, por exemplo, contendo, além do endereço e um comando, o tamanho da mensagem que está sendo transmitida. Um byte de CRC também pode ser anexado ao final da mensagem como medida da integridade da mensagem. Este é o formato usado no MODBUS.

Algumas precauções devem ser tomadas quando utliza-se a arquitetura mestre/escravo. Se for desejada a propriedade *plug-and-play* ao sistema, pode ser interessante que o mestre, quando não tiver solicitando informações dos escravos já detectados, possa varrer o barramento enviando mensagens de checagem para detectar (i) a retirada de um dispositivo da rede e (ii) a entrada de um novo dispositivo. Também pode-se definir mensagens de *broadcast*, para as quais todos os escravos recebem a mesma informação do mestre, que pode ser uma configuração geral ou sincronização de relógios. E também, o mestre deve sempre esperar a resposta do escravo antes de enviar outro pacote para evitar colisões. O projetista deve também considerar que alguma falha que possa ocorrer no sistema, que seja na transmissão como também nos dispositivos da rede. Assim, o mestre não pode ficar indefinidamente esperando uma resposta de um escravo, nem o escravo do mestre. Se isto ocorrer, tem-se uma situação de bloqueio do sistema. Ao invés disso, deve-se estipular um tempo de espera máximo para chegada de uma mensagem de retorno (se o mestre solicitou uma do escravo) ou de conclusão da mensagem (mestre ou escravo). Este cuidado deve ser levado em conta também caso uma mensagem chegue incompleta, devendo-se aguardar um tempo limitado por cada byte esperado. Em situações de falha na comunicação, sugere-se que o dispositivo descarte a mensagem parcialmente recebida ou com erro de CRC, sinalizando ao mestre esta situação. Em geral isto é feito por meio de uma mensagem de reconhecimento (ACK). Se tal mensagem não for enviada por quem recebeu a mensagem, então o dispositivo que a enviou pode concluir que houve falha de comunicação. Nestes casos, pode-se tentar retransmitir a mensagem por até um certo número de vezes. Se a falha se mantiver, então conclui-se que o dispositivo está em falha permanente.

5.2 Controle de acesso por software ao barramento RS-485

No caso de microcontroladores ligados diretamente ao transceptor de 485 ou no caso de utilização do conversor RS-232/RS-485 com controle pelo **RTS**, o procedimento de acesso ao barramento para escrita é feito por software. Em ambos os casos é preciso setar o pino **DE** do transceptor antes da transmissão e mantê-lo em nível alto até o fim da mesma.

Em microcontroladores esse procedimento torna-se bem simples, dada a facilidade em se ativar e desativar uma das saídas digitais, e também porque a maioria dos microcontroladores com UART possuem interrupções para indicar o fim de uma transmissão, que ocorre quando o *buffer* de transmissão fica vazio.

Já em microcomputadores PC o procedimento é o mesmo, sendo que o controle é feito pelo pino **RTS** da porta serial. Se a aplicação for desenvolvida em linguagem C, por exemplo, o acesso a saída **RTS** pode ser feito utilizando a API do sistema operacional, inclusive para espera do fim da transmissão. Ao passar por APIs, o programa pode ter comportamento pouco determinístico, principalmente em se tratando de sistemas operacionais que não sejam tempo real. Isto ainda fica pior com o Windows, que com as versões 2000/XP impossibilitaram o acesso ao hardware por programas que se executam no modo usuário. O mesmo somente pode ser feito no modelo kernel via *device*

drivers. Para tanto, existem alguns *device drivers* para acesso ao hardware tais como GiveIO[1] ou PortTalk[2]. Entretanto, o tempo necessário para troca de informações do software do usuário com o *device driver* é ainda não determinístico, dificultando, por exemplo, o tempo máximo de espera por uma mensagem.

No caso do Linux, ainda que seja um sistema operacional de propósito geral, o acesso direto a portas de entrada/saída é ainda possível no espaço do usuário sem que seja necessário um *device driver*. O Apêndice A mostra o código fonte de uma biblioteca em linguagem C para Linux para envio e recepção de bytes utilizando o controle pelo **RTS**. Esta biblioteca pode ser usada com o circuito da Figura 5.

Agradecimentos

Alexandre S. Martins: ao prof. Geovany pela idéia e iniciativa de criar e disponibilizar recursos técnicos. Que este trabalho sirva também como incentivo a outros estudantes a dar continuidade a essa idéia.

Geovany A. Borges: ao Alexandre S. Martins pela primeira versão deste documento e suas contribuições de hardware e de software que vêm sendo usadas por novos trabalhos no LCVC.

Referências

[1] Control Network. *Understanding EIA-485 Networks*. 1999. The EXTENSION - Technical Supplement to Control Network. Disponível em `http://www.ccontrols.com/pdf/ExtV1N1.pdf`.

[2] RANDAZZO, A. *ST485: AN RS-485 BASED INTERFACE WITH LOWER DATA BIT ERRORS*. 2004. Application Note 1348. Disponível em `http://www.st.com/stonline/products/literature/an/7628.pdf`.

[3] MAXIM. *Guidelines for Proper Wiring of an RS-485 (TIA/EIA-485-A) Network*. July 2001. Application Note 763. Disponível em `http://www.maxim-ic.com/appnotes.cfm/appnote_number/763/`.

[4] MAXIM. *Explanation of Maxim RS-485 Features*. December 2000. Application Note 367. Disponível em `http://www.maxim-ic.com/AN367`.

[5] MARTINS, A. S.; ANDRADE, D. *Cinturão de ultra-som para detecção de obstáculos para o robô Omni*. [S.l.], Janeiro 2005. Disponível em `http://www.ene.unb.br/~gaborges/pesquisa/movel/projrob/pf.alexandre.martins.diogo.andrade.2004.2.pdf`.

[6] LIMA, L. da S.; INUZUKA, H. *Desenvolvimento de um robô móvel Omnidirecional: o robô ROHL*. [S.l.], Junho 2005. Disponível em `http://www.ene.unb.br/~gaborges/pesquisa/movel/projrob/pf.leandro.lima.heiji.inuzuka.2005.1.pdf`.

A Biblioteca C para Linux para comunicação serial

Embora esta biblioteca contenha o básico para comunicação serial no Linux, ela também permite ser utilizada diretamente para RS-485com hardware de interface dado pelo circuito da Figura 5. Para tanto, na biblioteca `seriallib.c` faz-se necessário definir `USE_RS-485` como sendo `1` para que o sinal RTS seja ativado apropriadamente quando da transmissão pela UART do PC.

Abaixo tem um pequeno programa exemplo ilustrando o uso da biblioteca.

[1] `http://www.ddj.com/184409876`

[2] `http://www.beyondlogic.org/porttalk/porttalk.htm`

```
#include <stdio.h>
#include <unistd.h> /* for libc5 */
#include <sys/io.h> /* for glibc */
#include "seriallib.h"

void main(void)
{
  unsigned char dado_enviado, dado_recebido;

  if(!seriallib_iniciar(SERIALLIB_COMPORT_1, SERIALLIB_BAUDRATE_9600)){
     return;
  }

  dado_enviado = 0xAA;
  printf("\nEnviando %X pela serial.",dado_enviado);
  if(!seriallib_enviarbyte(&dado_enviado)){
     printf("   Erro de transmissao!");
     return;
  }
  printf("\nAguardando resposta...");
  if(!seriallib_receberbyte(&dado_recebido, 10000)){
     printf("   Resposta nao chegou em 10 ms!");
  }
  printf("\nResposta: %X.",dado_recebido);
}
```

A.1 seriallib.c

```
//*************************************************************************
// UNIVERSIDADE DE BRASILIA - UnB                                         *
// BIBLIOTECA DE FUNCOES PARA COMUNICACAO SERIAL                          *
//                                                                        *
// Alterado a partir de "SerialLibRTX.c", desenvolvida por                *
// Geovany Araujo Borges. Adaptada por Alexandre Martins                  *
//                                                                        *
//*************************************************************************

#include <stdio.h>
#include <unistd.h> /* for libc5 */
#include <sys/io.h> /* for glibc */
#include "seriallib.h"
#include "rdtsctime.h"

// Se usar para RS-485, defina USE_RS485 como sendo 1. Caso contrario, defina 0.
#define USE_RS485 1

// porta de comunicacao default
static int SerialPortAddress = SERIALLIB_COMPORT_1;

// taxa de comunicação default.
static unsigned char SerialPortBaudrate = SERIALLIB_BAUDRATE_115200;
```

```
/************* Rotinas Genéricas de Manipulação da porta serial **************/
// abertura da porta serial
int seriallib_iniciar(int Comport, unsigned char bps)
{
        SerialPortAddress = Comport;
        SerialPortBaudrate = bps;

        if(iopl(3)<0){ // solicita privilegio de esrita em IO
              printf("seriallib_iniciar: iopl error");
              return(0); // se deu errado, eh porque nao esta sendo executado pelo root.
        }

        outb(0, SerialPortAddress + 1);     // Desativar interrupcoes
        outb(0x80, SerialPortAddress + 3);  // DLAB ON
        outb(bps, SerialPortAddress + 0);   // Baud Rate - DL Byte
        outb(0x00, SerialPortAddress + 1);  // Baud Rate - DH Byte
        outb(0x03, SerialPortAddress + 3);  // 8 Bits, No Parity, 1 Stop Bit
        outb(0xC7, SerialPortAddress + 2);  // FIFO: 14 bytes, limpa TX e RX fifos
        outb(0x0B, SerialPortAddress + 4);  // Ativa DTR, RTS = 0, e OUT2

        return(1);
}


// rotina de transmissao de um byte com controle do pino RTS
int seriallib_enviarbyte(unsigned char *pData)
{
        int contador = 0;
        unsigned char dado;

        while(!(inb(SerialPortAddress + 5) & 0x40)){
                seriallib_delay(100);
                if(++contador>10) return(0);
        } // Espera fim da ultima transmissao

#if USE_RS485 == 1
       dado = inb(SerialPortAddress + 4);
        dado = dado & (~0x02);
        outb(dado, SerialPortAddress + 4);  //setar RTS = 1 mantendo OUT2
#endif
        outb(*pData, SerialPortAddress + 0); // Envia.

#if USE_RS485 == 1
        contador  = 0;
        while(!(inb(SerialPortAddress + 5) & 0x40)); // aguarda final da transmissao

        dado = inb(SerialPortAddress + 4);
        dado = dado | (0x02);
        outb(dado, SerialPortAddress + 4);  //setar RTS = 0 mantendo OUT2
#endif
```

```
        return(1);
}

// Rotina que recebe um byte pela porta serial, com um timeout customizavel
// A resolucao da espera eh de 100us.
int seriallib_receberbyte(unsigned char *pData, double MaximaEsperaUS)
{
        double  ElapsedTime;

        if(MaximaEsperaUS<0) MaximaEsperaUS = 0; // Espera minima de 0 us.

        ElapsedTime = 0.0;
        while(1){
                if(inb(SerialPortAddress + 5) & 1){
                        *pData = inb(SerialPortAddress + 0);
                        return(1);
                }
                seriallib_delay(100.0); //
                ElapsedTime += 100.0;
                if(ElapsedTime >= MaximaEsperaUS){
                        return(0);  // Dado nao chegou no tempo estipulado.
                }
        }
}

// Rotina que provoca uma espera de tempo em microsegundos. Usa as funcoes de rdtsctime.h
void seriallib_delay(double MaximaEsperaUS)
{
        CLOCK_COUNTER ckstart, ckend;
        double  ElapsedTime;
        RDTSCCFG Rdtsccfg;

        RDTSC_SetCPUClock(Rdtsccfg,RDTSC_DEFAULTFREQ);

        if(MaximaEsperaUS<0) MaximaEsperaUS = 0; // Espera minima de 0 us.

        ElapsedTime = 0.0;
        RDTSC_GetClock(ckstart);
        while(1){
                RDTSC_GetClock(ckend);
                ElapsedTime = 1e6*RDTSC_ComputeTimeInterval(&Rdtsccfg,&ckstart, &ckend);
                if(ElapsedTime >= MaximaEsperaUS){
                        return;  // Dado nao chegou no tempo estipulado.
                }
        }
}
```

A.2 seriallib.h

```
//*************************************************************************
// UNIVERSIDADE DE BRASILIA - UnB                                         *
// BIBLIOTECA DE FUNCOES PARA COMUNICACAO SERIAL                          *
```

```
//                                                                      *
// Alterado a partir de "SerialLibRTX.c", desenvolvida por              *
// Geovany Araujo Borges. Adaptada por Alexandre Martins                *
//                                                                      *
//**********************************************************************
#define SERIALLIB_COMPORT_1 0x3F8
#define SERIALLIB_COMPORT_2 0x2F8
#define SERIALLIB_COMPORT_3 0x3E8
#define SERIALLIB_COMPORT_4 0x2E8

#define SERIALLIB_BAUDRATE_38400     0x03
#define SERIALLIB_BAUDRATE_115200    0x01
#define SERIALLIB_BAUDRATE_57600     0x02
#define SERIALLIB_BAUDRATE_19200     0x06
#define SERIALLIB_BAUDRATE_9600      0x0C
#define SERIALLIB_BAUDRATE_4800      0x18
#define SERIALLIB_BAUDRATE_2400      0x30

//Prototipos
int seriallib_iniciar(int Comport, unsigned char bps);
int seriallib_enviarbyte(unsigned char *pData);
int seriallib_receberbyte(unsigned char *pData, double MaximaEsperaUS);
void seriallib_delay(double MaximaEsperaUS);
```

A.3 rdtsctime.h

```
// Funcoes que fazem uso do contador de coclos de relogio TSC da arquitetura x86 (>=Pentium)
// para medir lapsos de tempo com alta resolucao.

// Nao esquecer de especificar aqui a frequencia da CPU.
// No Linux, ele pode ser determinada fazendo: grep MHz /proc/cpuinfo
#define RDTSC_DEFAULTFREQ 2.421607e9

// Estrutura de contador de 64 bits
typedef struct {
        unsigned int l;
        unsigned int h;
} CLOCK_COUNTER,*PCLOCK_COUNTER;

// Frequencia da CPU
typedef struct{
        double CpuFreq;
} RDTSCCFG,*PRDTSCCFG;

/*************************************************************************
**************************************************************************
** RDTSC_SetCPUClock:
**  Set CPU clock frequency. Must be used before any call to RDTSC_ComputeTimeInterval.
**************************************************************************
**************************************************************************/
#define RDTSC_SetCPUClock(Rdtsccfg,Freq)        Rdtsccfg.CpuFreq = Freq
```

```
/*****************************************************************************
******************************************************************************
** RDTSC_GetClock:
**  Recover CPU clock counter.
******************************************************************************
******************************************************************************/
#define RDTSC_GetClock(Clock) asm("rdtsc"); asm("mov %%eax, %0": "=r"(Clock.l));
                              asm("mov %%edx, %0": "=r"(Clock.h));

/*****************************************************************************
******************************************************************************
** RDTSC_ComputeTimeInterval:
**   Compute the time interval elapsed between pckStart and pckStop in seconds.
**   Returns -1.0 if pckStop < pckStart.
******************************************************************************
******************************************************************************/
__inline double RDTSC_ComputeTimeInterval(PRDTSCCFG pRdtsccfg,
                              PCLOCK_COUNTER pckStart, PCLOCK_COUNTER pckStop)
{
   unsigned long long ckInterval;
   unsigned long long ckStart,ckStop;

   ckStart  = pckStart->h;
   ckStart  = ckStart << 32;
   ckStart += pckStart->l;

   ckStop  = pckStop->h;
   ckStop  = ckStop << 32;
   ckStop += pckStop->l;

   if (ckStop < ckStart){
      return (-1.0);
   }
   ckInterval = ckStop-ckStart;

   return( ((double)((long long )(ckInterval)))/pRdtsccfg->CpuFreq );
}
```