

RS485 com microcontrolador PIC

Introdução

Este artigo apresenta uma forma simples de implementação de uma interface RS485 com microcontroladores PIC da Microchip. O uso destes recursos permite a implementação de soluções interessantes de controle via computador, através da porta COM RS232. A interface RS485 permite a transmissão de dados a uma distância relativamente grande acima de 1Km, sendo uma boa escolha para controle de equipamentos a distância numa linha de fábrica, por exemplo. Outra grande vantagem é a possibilidade de endereçamento dos módulos controlados.

A arquitetura mais comum é a mestre – escravo, onde um módulo mestre (normalmente o PC) envia comandos para os módulos escravo, cada um com endereço próprio. A forma de comunicação é normalmente do tipo pergunta – resposta. O módulo mestre envia um comando solicitando uma informação ou uma ação de um módulo escravo (pergunta) e este responde com a informação solicitada ou com a confirmação que executou a operação requisitada (resposta). A figura 1 mostra a forma de conexão entre mestre e escravos.

Como a RS485 é baseada na RS232, as informações que trafegam pelo barramento são formatadas em bytes (8bits) e transmitidas em série. A seqüência mais comum é apresentada na figura 2. Inicialmente é enviado o endereço do módulo a ser controlado, em seguida é enviado um byte correspondente à função desejada, seguido dos bytes de dados (nem sempre presentes, como veremos). Após os bytes de dados, a “pergunta” enviada pelo mestre é encerrada com dois bytes de

CRC, conhecidos como bytes de checkSum. Estes são utilizados para garantir que a informação transmitida será recebida corretamente e são calculados em função dos bytes transmitidos na comunicação. No nosso exemplo da figura 2 o CRC (byte high e byte low) são calculados de acordo com o endereço, função e dados de 1 a n.

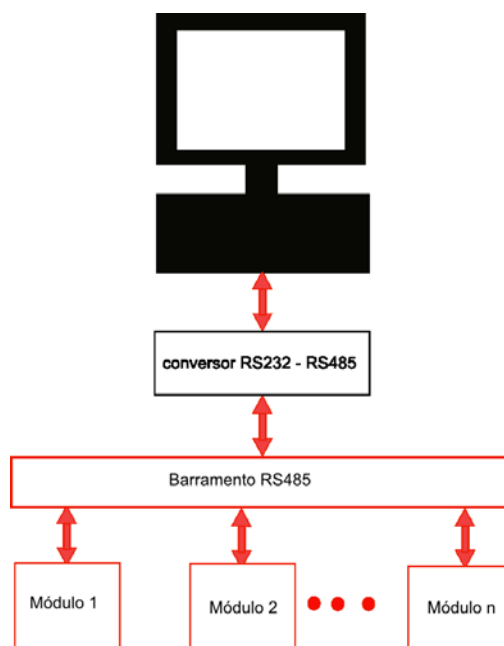


Fig 1

Posição	Tipo	Valor
01	ENDEREÇO	XXh
02	FUNÇÃO	XXh
03	DADOS 1	XXh
.	DADOS 2	XXh
.	.	XXh
.	DADOS n	XXh
.	CRC high	CALC
n + 2	CRC low	CALC

Fig 2

Por exemplo, suponhamos que queremos controlar um módulo de reles e a operação seja o acionamento de um dos reles. Vamos supor ainda que o endereço do módulo escravo seja 23H(hexadecimal)) e a função “fechar rele” seja representada pelo byte 02H. Neste caso, a seqüência transmitida via RS485 poderia ser:

Posição	Tipo	Valor
01	ENDEREÇO	23H
02	FUNÇÃO	02H
03	DADOS 1	04H
04	CRC high	CALC
05	CRC low	CALC

Onde a posição 03 (04H) é o endereço do rele que queremos acionar. Os bytes CRC high e CRC Low, serão calculados em função dos bytes das posições 01, 02 e 03. Nós não entraremos no mérito deste cálculo pois ele pode variar de acordo com a implementação escolhida.

O módulo escravo, ao receber uma seqüência deste tipo, inicialmente verifica se o endereço corresponde ao seu. Se o endereço for diferente, o módulo escravo finaliza a comunicação, assumindo o status de “listen” (que pode ser traduzido como “ouvindo” o barramento) desconsiderando os bytes que vem em seguida.. Se por outro lado, o endereço corresponde ao seu, o módulo escravo decodifica o byte de função e executa a tarefa correspondente a esta função. No nosso caso, aciona o rele identificado pelo byte 04H, recebido na posição Dados1.

Esta formatação apresentada é conhecida como ModBus e tem a vantagem de ser relativamente simples e bastante flexível. Outra vantagem é a

facilidade de isolar eletricamente o barramento de comunicação do computador mestre, o que pode ser muito útil em instalações industriais, preservando o computador de surtos de tensão normalmente presentes nos barramentos. Uma desvantagem é a velocidade de comunicação que é limitada se comparada a outros protocolos, como o USB, por exemplo.

Apesar de antigo o protocolo ModBus ainda é bastante utilizado com lançamentos de novos módulos a todo instante. É importante notar que o ModBus não está restrito a nenhuma interface física de comunicação. Ele é apenas uma formatação de dados seriais que pode ser utilizado em qualquer interface física, inclusive USB.

Neste trabalho utilizaremos o microcontrolador PIC16F883 da Microchip e o integrado MAX485 da Maxim, bastante popular e fácil de ser encontrado.

A adaptação da solução apresentada para outras famílias é relativamente simples devido à semelhança de estrutura mantida pela Microchip. A linguagem adotada aqui será o assembly mas em breve disponibilizaremos em C

Recursos necessários

Para desenvolver este trabalho, você terá que dispor da estrutura mínima necessária. Como vamos utilizar a linguagem assembly da Microchip específica da família 16F, todos os programas necessários para a programação do PIC16F883 estão disponíveis de graça na página da Microchip (www.microchip.com) na plataforma MPLAB. O módulo de programação utilizado por nós é o

PICStartPlus, que não é mais fabricado, mas você pode adquirir os programadores da série PICKIT 3 por aproximadamente R\$ 120,00, também controlados pelo MPLAB

O que é comunicação serial?

Comunicação serial é aquela em que a informação trafega pela interface física em série, ou seja a informação é transmitida numa seqüência de bits em série. Os bits que compõem o byte a ser transmitido são transferidos um a um até completar o byte (8 bits). Na comunicação paralela, os 8 bits do byte são transferidos de uma só vez através de 8 linhas paralelas. A comunicação paralela, muito utilizada no passado (porta paralela da impressora, por exemplo) tem a vantagem de alcançar velocidades maiores, pois os bits são transferidos todos de uma vez, e de apresentar o controle de comunicação mais simples. Basta disponibilizar os bits no barramento paralelo e enviar um comando indicando que o byte esta pronto para ser lido pelo módulo receptor. Na comunicação serial, a velocidade final é mais baixa pois cada bit é transmitido individualmente. Então para um mesmo clock, a velocidade seria pelo menos 8 vezes menor que a da paralela. A outra desvantagem é que o controle da transmissão é consideravelmente mais complexo que o da comunicação paralela. Não existe a informação de que o byte está pronto para ser lido. O sincronismo é todo feito por temporizações do próprio barramento e circuitos de controle. Entretanto, a comunicação paralela caiu em desuso enquanto a serial tornou-se popular. Isto aconteceu devido a constante evolução das técnicas de comunicação serial, com velocidades cada vez mais altas e confiabilidade na transmissão. A grande

vantagem é o custo baixo de implementação, uma vez que a transmissão pode se dar por 1 fio, ao contrário de 8 da paralela. No caso dos microcontroladores com duas linhas (TX e RX) é possível implementar uma interface serial. Como o custo do microcontrolador é diretamente proporcional às portas que disponibiliza, o custo da comunicação serial traz vantagens inquestionáveis.

Conector RS232

O conector padrão para RS232 é o DB9. Como já dissemos anteriormente, a comunicação se dá entre um módulo mestre e um módulo escravo. No lado mestre o padrão de conector é macho e no lado escravo é fêmea. A tabela abaixo mostra a pinagem padrão para o Módulo Mestre (lado computador):

Pinagem DB9 – RS232 (lado Mestre - macho)	
Pino	Sinal
1	DCD – Data Carrier Detect – usado em modems (não utilizaremos este sinal)
2	RxD – Recebe dados (entrada)
3	TxD – Envia Dados (saída)
4	DTR – Data Terminal Ready - usado pelo mestre para solicitar o envio de dados. Normalmente não é utilizado .
5	GND – massa da comunicação
6	DSR – Data Set Ready - enviado pelo escravo, indica ao mestre que o escravo está pronto para receber dados. Normalmente não é utilizado
7	RTS – Request to Send – enviado pelo mestre solicitando permissão para envio de dados
8	CTS – clear to Send – trabalha em conjunto com RTS para controle de fluxo
9	RI – Ring Indicator -indicador de chamada utilizado em modems

No lado escravo (equipamento sobre controle) o conector é fêmea e os sinais são conforme a tabela abaixo.

Pinagem DB9 – RS232 (lado Escravo - fêmea)	
Pino	Sinal
1	DCD – Data Carrier Detect – usado em modems (não utilizaremos este sinal)
2	TxD – Envia Dados (saída)
3	RxD – Recebe dados (entrada)
4	DTR – Data Terminal Ready - usado pelo mestre para solicitar o envio de dados. Normalmente não é utilizado .
5	GND – massa da comunicação
6	DSR – Data Set Ready - enviado pelo escravo, indica ao mestre que o escravo está pronto para receber dados. Normalmente não é utilizado
7	RTS – Request to Send – enviado pelo mestre solicitando permissão para envio de dados
8	CTS – clear to Send – trabalha em conjunto com RTS para controle de fluxo
9	RI – Ring Indicator -indicador de chamada utilizado em modems

Note a inversão entre os pinos 2 e 3. Um cabo padrão completo para comunicação RS232 pode ser montado com um conector DB9 macho de um lado e DB9 fêmea do outro com os pinos de mesma numeração interligados, ou seja 1 com 1, 2 com 2 até o último, 9 com 9, sem inversões. Alguns equipamentos utilizam cabos com inversão dos pinos 2 e 3, ou seja, o 2 macho ligado ao 3 fêmea e 3 macho ligado ao 2 fêmea. No nosso caso utilizaremos a conexão sem inversão.

Comparando RS232 e RS485

A comunicação RS485 utiliza a mesma formatação de bits da RS232. A semelhança, entretanto, para por ai. A grande diferença está na forma de transmitir a informação pelo meio físico. No caso da RS232 a linha TX do mestre está conectada à linha RX do escravo e a

TX do escravo à RX do mestre. Nesta comunicação só é possível conectar um módulo mestre e um módulo escravo. Não é possível a conexão de um mestre a vários escravos. Isto acontece porque na RS232 não é previsto o estado “tri state” da linha TX. Então se conectarmos duas linhas TX de equipamentos escravos em paralelo ocorrerá o conflito entre estas linhas TX.

Na comunicação RS485, ao contrário, um módulo escravo pode entrar em modo “listen” (escuta), colocando suas linhas de comunicação em estado “tri state”. Assim é possível conectar em paralelo vários módulos escravos, conforme figura 1.

Continua.....

(este trabalho sofre atualizações semanais até ser concluído)