Implementação de uma interface UART para barramento SPI empregando sistemas microcontrolados de baixo custo

André Melo Carvalhais Dutra, Wagner Chiepa Cunha
Instituto Tecnológico de Aeronáutica - Praça Marechal-do-Ar Eduardo Gomes, 50, São José dos Campos – SP

***Resumo* — Este artigo aborda a implementação de um conversor de barramentos RS-232/SPI empregando microcontroladores da família MCS-51. O mesmo se inicia com uma breve introdução que contextualiza o trabalho, e aborda a motivação para a realização do mesmo. Em seguida é feito uma breve revisão sobre o padrão RS-232, o barramento SPI, e da família MCS-51. O artigo passa, então, a abordar a implementação da interface, explicando as diversas funções necessárias à sua realização, tais como as interfaces de comunicação com os barramentos, como foram implementados os *buffers* dos mesmos, e as rotinas de controle deste, e como tudo isso é integrado no laço principal do *firmware*. Após uma breve explanação sobre os resultados alcançados, e identificados pontos da implementação que apresentam oportunidades de melhorias, o artigo é finalizado apresentando algumas sugestões para implementação em futuros trabalhos.**

***Palavras-chaves* — Padrão RS-232, barramento SPI, UART, 8051, MCS-51.**

I. INTRODUÇÃO

A cada instante que se passa, mais e mais sistemas embarcados, utilizados para as mais diversas finalidades, surgem à nossa volta. Desde práticos e baratos relógios digitais, a complexas transações financeiras realizadas enquanto nos sentamos numa confortável cadeira à beira-mar, o nosso cotidiano passa pela eletrônica embarcada.

A grande maioria dos dispositivos embarcados, porém, é desenvolvida com base num conceito fundamental: um pequeno computador (normalmente um microcontrolador) que realiza uma tarefa específica e que muitas vezes se comunica com pelo menos um, ou alguns poucos periféricos. E é justamente neste contexto, o da comunicação entre estes pequenos computadores e seus periféricos, que este trabalho encontra sua contribuição.

O emprego de barramentos seriais é muito comum para que haja o estabelecimento deste tipo de comunicação, uma vez que comumente exigem um número menor de sinais e complexidade reduzida para a sua implementação.

Entretanto, é comum que algumas vezes o barramento serial escolhido para a realização do enlace de dados não esteja disponível em determinado dispositivo.

Embora a escolha de outro dispositivo que implemente o barramento com o qual se deseja trabalhar seja a opção mais natural, muitas vezes ela não está disponível devido a uma imposição do projeto, ou indisponibilidade de outro dispositivo com o barramento escolhido, por exemplo.

A grande motivação para realização do presente estudo surgiu de uma situação semelhante às acima descritas: desejava-se conectar um dispositivo que emprega o *"Serial Peripheral Interface Bus"* (Barramento Serial para Interface de Periféricos, ou simplesmente barramento SPI), mais especificamente o módulo transceptor que consta da Fig. 1 abaixo, com uma porta serial padrão RS-232 de um computador do tipo PC.

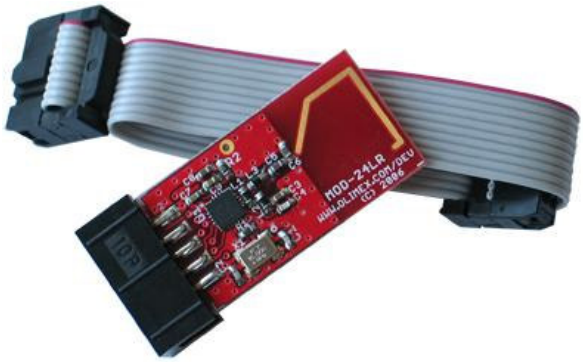


Fig. 1: Módulo transceptor que emprega o barramento SPI

II. O PADRÃO RS-232

O padrão RS-232, em uso desde 1969, é o padrão de comunicação serial mais amplamente utilizado e o mais bem conhecido [1]. A sua designação formal é EIA/TIA 232-E.

Este padrão especifica características mecânicas, elétricas e funcionais da interface [2], sendo que estas últimas são as que mais interessam ao presente estudo. A interface RS-232 é uma interface serial assíncrona, que pode ser utilizada tanto para receber, quanto para transmitir dados.

Em função disso, e das características apresentadas anteriormente, será utilizado um sub-conjunto das especificações deste padrão para a implementação da UART (termo consagrado na literatura, que significa Transmissor / Receptor Assíncrono Universal) proposta no presente trabalho.

O padrão especifica a existência de dois tipos de dispositivos: *"Data Terminal Equipment"* (Equipamento Terminal de Dados, ou DTE) e *"Data Comunication Equipment"* (Equipamento de Comunicação de Dados, ou DCE). Podemos entender o DTE como sendo o destino final dos dados, e o DCE como sendo um tradutor de sinais, convertendo os sinais deste padrão para algum outro qualquer, de interesse para a ocasião [1].

A. M. C. Dutra, carvalhais@ita.br Tel +55-12-3947-3000, ramal 6885, W. C. Cunha, chiepa@ita.cta.br.

O RS-232 especifica os seguintes sinais lógicos e de controle (utilizados os nomes consagrados em inglês por razões de clareza) [2]:

TABELA 1: SINAIS DEFINIDOS PELO PADRÃO RS-232

Sinal	Direção
TD – Transmitted Data	DTE → DCE
RD – Received Data	DTE ← DCE
RTS – Request to Send	DTE → DCE
CTS – Clear to Send	DTE ← DCE
DSR – Data Set Ready	DTE ← DCE
DTR – Data Terminal Ready	DTE → DCE
CD – Carrier Detect	DTE ← DCE
RI – Ring Indicator	DTE ← DCE
GND – Ground	–

Os sinais TD e RD se destinam à transmissão e recepção de dados, os sinais RTS e CTS são de finalidade geral, os sinais DSR e DTR sinalizam que o equipamento que origina cada um dos mesmos foi ativado e o sinal GND é a referência para as tensões [2]; os sinais CD e RI, mais empregados quando o DCE trata-se de um *modem*, servem para, respectivamente, indicar a presença de uma portadora na linha, e que outro assinante do sistema telefônico está chamando.

Outro aspecto de grande importância quando tratamos do comunicação assíncrona diz respeito à taxa de bauds (ou *"baud rate"*, abreviada por BR), que no caso do padrão RS-232, se iguala à taxa de bits. Como neste tipo de comunicação não há um sinal de relógio para validar o bit, o sucesso da mesma é completamente dependente da temporização adequada entre transmissor e receptor.

Para marcar o início da transmissão, existe um bit especial, denominado bit de partida; de forma semelhante, o final da palavra de dados é marcada com outro bit especial, denominado bit de parada [3]. Entre o bit de partida e o de parada estão os bits de dados, e cada um desses bits deve ocupar uma janela de tempo precisa; essa janela é justamente o tempo que o bit demora para ser transmitido, e é numericamente igual ao inverso da taxa de bits; é importante que esse intervalo seja judiciosamente respeitado, uma vez que o dispositivo com o qual se deseja comunicar pode amostrar o bit em qualquer instante desta janela. A figura abaixo ilustra melhor o esquema da comunicação assíncrona:

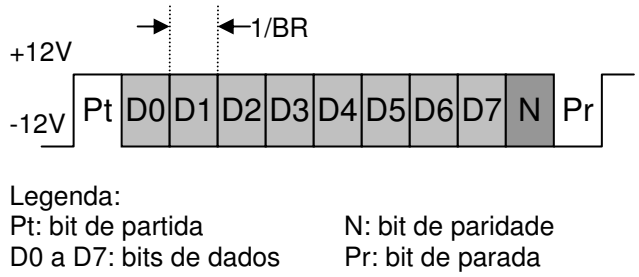


Fig. 2: Esquema da comunicação serial assíncrona

Na figura é possível observar que além dos bits de partida, de dados, e de parada, pode existir mais um bit (que é opcional), denominado bit de paridade.

Dessa forma, para que uma comunicação deste tipo seja estabelecida com sucesso, é preciso que as partes envolvidas na comunicação saibam, *a priori*, a taxa de bauds, bem como o formato da palavra de dados (número de bits de dados, tipo de paridade, e número de bits de parada). Exemplos de taxas de bits típicas são 9600 bps, 19200 bps, 38400 bps e 57600 bps, bem como um formato bastante comum da palavra de dados é aquele no qual são transmitidos 8 bits de dados, nenhum de paridade e 1 bit de parada.

III. O BARRAMENTO SPI

O SPI é um padrão de barramento serial desenvolvido pela Motorola, e está amplamente disponível em dispositivos produzidos por vários fabricantes diferentes [4]. Ele opera no modo síncrono e *"full-duplex"* (os dados trafegam nas duas direções simultaneamente).

No barramento SPI existe um dispositivo mestre com capacidade para controlar múltiplos dispositivos escravos conectados a um mesmo barramento. Os sinais empregados no barramento SPI estão descritos na tabela 2.

Os sinais MOSI e MISO servem para a transmissão dos dados entre o mestre e o escravo. O sinal SCLK é o relógio empregado na comunicação síncrona, e serve para marcar o instante de validade dos sinais MOSI e MISO. O sinal #SS, ativo em nível baixo, serve para selecionar qual escravo será o destino da transmissão; como o barramento SPI admite múltiplos escravos, podem existir múltiplas linha #SS no barramento.

TABELA 2: SINAIS EMPREGADOS NO BARRAMENTO SPI

Sinal	Direção
MISO – Master In Slave Out	Mestre ← Escravo
MOSI – Master Out Slave In	Mestre → Escravo
SCLK – SPI Clock	Mestre → Escravo
#SS – Slave Select	Mestre → Escravo

Podemos observar na figura abaixo como vários dispositivos escravos se conectam a um mesmo mestre num barramento SPI:

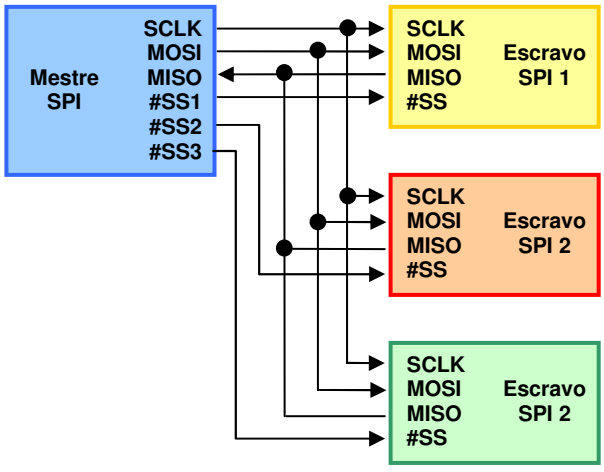


Fig. 3: Esquema de conexão dos dispositivos ao barramento SPI

Como pode ser observado o barramento SPI possui três sinais comuns a todos os dispositivos (MOSI, MISO e SCLK), e um quarto sinal que serve para habilitar, individualmente, cada dispositivo escravo presente no barramento (#SS).

Um aspecto interessante do barramento SPI, é que considera-se que o mesmo está sempre transmitindo dados em ambas as direções. Dessa forma, cabe aos dispositivos mestre

e escravo determinar se o dado recebido possui algum significado, ou não [4].

A figura 4 ilustra as formas de ondas para os diversos sinais durante uma comunicação SPI, e serve para ilustrar alguns dos conceitos abordados, bem como alguns outros a serem introduzidos logo a seguir:

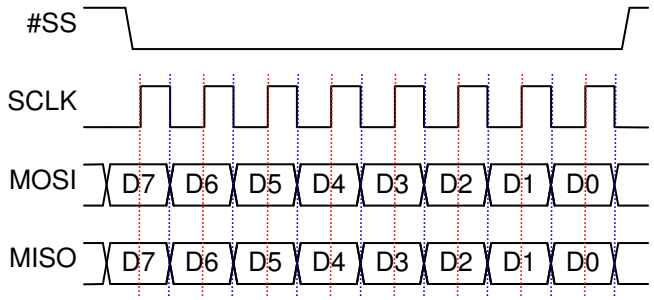


Fig. 4: Formas de ondas para a comunicação SPI

Para que a comunicação através do barramento SPI seja realizada com sucesso, inicialmente é necessário que o dispositivo mestre imponha um nível baixo na linha #SS do dispositivo escravo com o qual se deseja comunicar.

Após isso, o dispositivo mestre disponibiliza na linha MOSI um bit do dado que deseja transmitir, enquanto o escravo disponibiliza na linha MISO um bit do dado a ser enviado para o mestre. O instante de validade dos bits (instante no qual os mesmos devem ser amostrados) é marcado por um dos flancos do sinal de relógio (no caso do exemplo da Fig. 4, o flanco de subida), enquanto o outro flanco do relógio é a referência para a troca dos dados existentes nas linhas de dados (flanco de descida na Fig. 4). Após cada ciclo de relógio um novo bit é transmitido em ambas as direções, e, assim, a comunicação segue o seu curso até o momento em que o mestre deixa de gerar os pulsos de relógio, e desativa a linha #SS.

O padrão não define qual flanco é utilizado para a amostragem dos dados, nem qual flanco é o sinal para a troca dos dados, ficando esta escolha a cargo de quem implementa o sistema (o padrão apenas define como as combinações de polaridade e de fase do sinal de relógio determinam o tipo de operação) [5].

Outro ponto que merece ser ressaltado é o fato dos bits da palavra de dados poderem ser transmitidos começando tanto pelo bit menos significativo, quanto pelo mais significativo. Na Fig. 4 intencionalmente colocou-se a transmissão sendo iniciada pelo bit mais significativo, para que a mesma pudesse ser contrastada com a Fig. 2.

Por ser um barramento síncrono, o SPI não exige que uma taxa de bits seja estabelecida para a realização da comunicação, uma vez que o flanco do sinal de relógio fornece um instante de tempo bem definido para a amostragem do sinal. Dessa forma é possível encontrarmos dispositivos SPI operando desde alguns Kbps até dezenas de Mbps.

IV. Microcontroladores da família MCS-51

A família MCS-51 surgiu em 1980 quando a Intel© lançou o primeiro membro de sua família, o 8051. Essa família de microcontroladores teve uma grande aceitação, e hoje, quase 30 anos após o seu lançamento, ainda encontra presença no mercado [3].

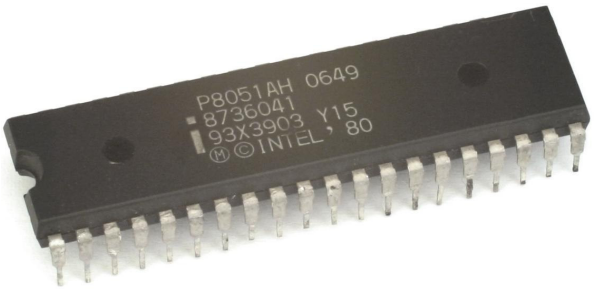


Fig. 5: Um dos primeiros membros da família MCS-51

Os motivos que levaram à escolha desta família para implementação deste projeto são: grande disponibilidade de ferramentas e programas que dão suporte ao uso do microcontrolador, baixo custo do componente e ampla literatura existente sobre o componente.

O 8051, assim como a grande maioria dos outros microcontroladores, disponibiliza uma série de facilidades ao seu utilizador. Uma explanação mais detalhada de suas possibilidades foge ao escopo do presente estudo.

Entretanto, podemos citar algumas das características principais do núcleo básico do 8051: arquitetura Harvard (memória de programa separada da memória de dados), CPU de 8 bits, uma certa quantidade da memória de programa já embutida no encapsulamento, memória RAM interna que permite armazenamento de pequena quantidade de dados e endereçamento dos registradores de funções especiais, temporizadores / contadores, uma UART, portas de I/O de finalidade geral, e um controlador de interrupções [3].

Assim, embora a família MCS-51 já seja bastante antiga, a mesma reúne num só dispositivo todas as funcionalidades necessárias à implementação da interface, e considerando as qualidades, já apresentadas, que levaram à adoção desta família, pode-se dizer, de uma maneira geral, que a escolha é adequada.

Para a realização da interface proposta no presente artigo, foi empregado um microcontrolador com nível de sofisticação bastante semelhante ao encontrado no núcleo básico da família, o AT89S52, fabricado pela Atmel©.

V. Implementação da interface

A Fig. 6 exemplifica de forma sucinta a abordagem utilizada na implementação dispositivo ora proposto. A figura busca representar de forma semelhante blocos funcionais de características semelhantes: os blocos esmaecidos indicam funções implementadas em *hardware* que se destinam a condicionar os sinais elétricos para os níveis válidos de cada circuito; os blocos amarelos representam os pontos de entrada e saída de dados do microcontrolador e são implementados tanto em *hardware* como em *firmware*; o laço principal do programa funciona como uma camada que abstrai o *hardware* do microcontrolador e suas rotinas associadas das rotinas e funções de acesso aos *buffers*; que dessa forma se preocupam somente em manipulá-los de forma adequada.

A Fig. 6 permite, ainda, observar claramente o fluxo de dados através do dispositivo: as linhas vermelhas indicam dados que trafegam entre o dispositivo e a UART, enquanto as linha azuis indicam dados que trafegam entre o dispositivo e a interface SPI; de forma análoga, as linhas tracejadas indicam dados que saem do dispositivo, enquanto que as linhas

cheias representam dados que entram no dispositivo. Essa abordagem modular é bastante conveniente uma vez que permite que uma rotina desenvolvida para realizar determinada tarefa atue de forma isolada das demais rotinas, precisando conhecer somente como se comunicar com as rotinas imediatamente adjacentes a ela.

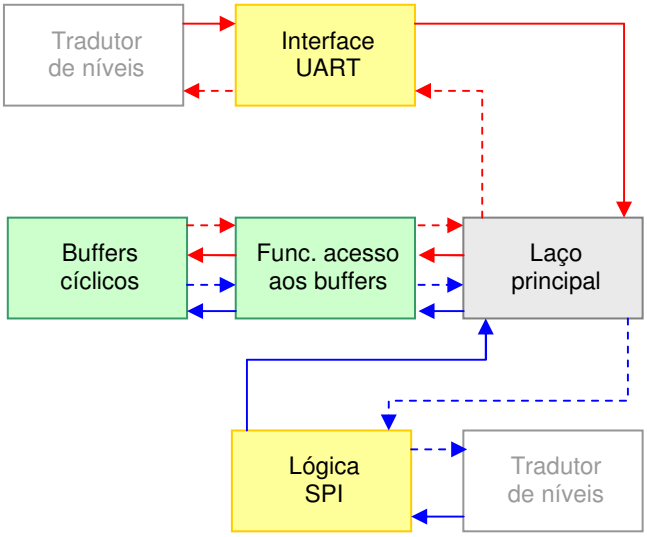


Fig. 6: Visão geral da implementação da interface

Conforme abordado anteriormente, a interface UART deste dispositivo irá empregar um subconjunto dos sinais especificados pelo barramento RS-232. Os sinais empregados constam da Tab. 3, listada a seguir (dispositivo proposto atuando como DCE).

TABELA 3: SINAIS DO PADRÃO RS-232 EMPREGADOS NO DISPOSITIVO

Sinal	Finalidade
TD – Transmitted Data	Comunicação de dados
RD – Received Data	Comunicação de dados
RTS – Request to Send	*Handshaking* e *reset* remoto
CTS – Clear to Send	*Handshaking*
GND – Referência	–

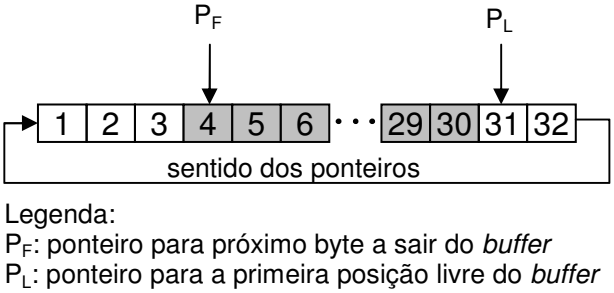
Conforme pode ser constatado, além dos sinais destinados à comunicação dos dados, fez-se uma previsão para empregar os sinais RTS e CTS para a negociação da comunicação (*"handshaking"*), além do sinal RTS poder ativar um circuito de *reset* remoto; o usuário pode selecionar a função do sinal RTS mediande um *"jumper"* existente na placa. É importante destacar que o sinal RTS, que chega ao microcontrolador, quando configurado para *handshaking*, aciona uma das fontes de interrupção externa do microcontrolador (mascarável), o que facilita bastante a implementação da lógica de *handshake*.

Como não havia um dispositivo próprio para o interfaceamento com o barramento SPI no microcontrolador utilizado, foram empregados pinos de I/O de finalidade geral, sendo a lógica da interface realizada através do *firmware*. As rotinas que implementam essa lógica, bem como todo o restante do programa que roda no microcontrolador, foram desenvolvidos em *assembly*, o que assegura um código eficiente e rápido. Como visto anteriormente, o barramento SPI, além de permitir diversas variações, impõe que tanto o mestre como o escravo decidam sobre o significado das informações recebidas, fato que exige extrema flexibilidade das interfaces que empregam este tipo de barramento. Isso seguramente é alcançado quando se realiza uma implementação através do *firmware*, pois pode-se facilmente implementar modificações e pequenos ajustes na lógica. É interessante citar que foram elaboradas três rotinas distintas para a lógica SPI: uma para comunicação *full-duplex*, e outras duas para comunicação *half-duplex* (entrada de dados e saída de dados), o que garante um máximo de otimização desta implementação.

Um aspecto bastante interessante desta implementação, são os *buffers* cíclicos empregados no armazenamento temporário dos dados. Os *buffers*, que foram incluídos após a realização de testes preliminares, conferiram grande agilidade à interface, contribuindo significativamente para uma melhoria no desempenho da mesma.

Uma vez que os dados podem seguir dois caminhos no interior do dispositivo (entrada RS-232 e saída SPI, ou saída SPI e entrada RS-232), houve a necessidade de implementação de dois *buffers*. Atualmente cada um dos *buffers* tem capacidade para 32 bytes, mas espera-se conseguir até 48 bytes com otimizações futuras.

A Fig. 7 ilustra a idéia dos *buffers* cíclicos e facilita a compreensão do modo de funcionamento dos mesmos. Enquanto um ponteiro aponta para a primeira posição de memória livre após o último byte escrito no *buffer* (aponta para a posição onde dados que serão escritos entram no *buffer*), outro ponteiro aponta para o próximo byte a ser lido (aponta para a posição a partir da qual os dados lidos saem do *buffer*). Após cada operação de escrita e leitura, o ponteiro correspondente é incrementado. Quando qualquer dos ponteiros chega ao final do buffer, sua numeração recomeça no início do mesmo. A Fig. 7 ilustra uma situação na qual as posições 4 a 30 estão ocupadas por dados, e a próxima posição livre é a 31.



Legenda:
P_F: ponteiro para próximo byte a sair do *buffer*
P_L: ponteiro para a primeira posição livre do *buffer*

Fig. 7: Esquema do *buffer* cíclico implementado

Embora o esquema adotado seja bastante conveniente em termos de praticidade de implementação, o mesmo pode se tornar perigoso quando não se toma o devido cuidado com os limites do *buffer*.

Para isso, foram desenvolvidas rotinas próprias de manipulação de cada um dos *buffers*, que detectam situações de *overflow* (tentativa de escrever num buffer já cheio), bem como de *underflow* (tentativa de ler o conteúdo de um buffer vazio). Cada buffer tem o seu próprio conjunto de rotinas, porém a estrutura de todas elas é bastante semelhante e encontra-se representado na Fig. 8.

Ao ser chamada, a rotina de escrita (leitura) testa o conteúdo do *buffer* para verificar se irá ocorrer uma situação de *overflow* (ou *underflow*). Caso positivo, a mesma não executa sua ação e sinaliza esta situação através de um *flag*. Caso a

mesma possa prosseguir na sua ação, a mesma é realizada e o controle do programa é devolvido à rotina chamadora.

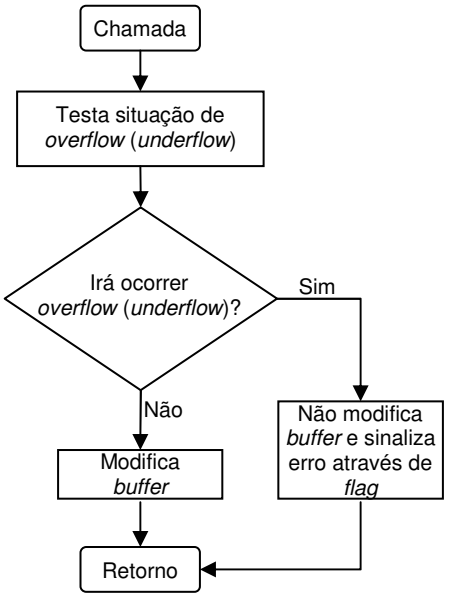


Fig. 8: Fluxograma das rotinas de manipulação dos *buffers*

O laço principal do programa, último componente da interface que falta ser abordado, ainda não se encontra completamente implementado, restando alguns ajustes finais ainda por serem realizados. Apesar disso, a sua estrutura geral é bem simples e encontra-se representada na Fig. 9.

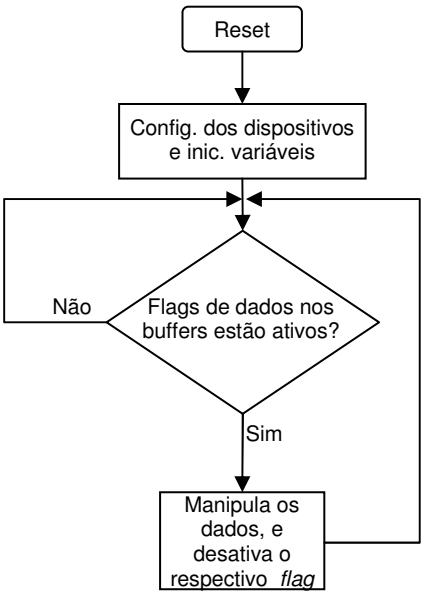


Fig. 9: Fluxograma do laço principal

Como pode ser facilmente observado a partir da Fig. 8, o laço principal do programa fica constantemente testando o conteúdo dos *buffers*. Assim que um algum deles contenha um dado, o laço principal se encarrega de transmití-los (através de chamadas às rotinas apropriadas).

Como qualquer manipulação dos *buffers* é sempre realizada através das rotinas apropriadas, nunca corre-se o risco de incorrer nas situações de *overflow*, ou *underflow*.

VI. TESTES E ENSAIOS INICIAIS

Apesar do laço principal ainda não estar totalmente implementado, já existe uma funcionalidade mínima que permitiu a realização de testes e ensaios iniciais, que já permitem a apresentação de alguns resultados.

O grande gargalo da interface é, obviamente, a comunicação com o barramento SPI, que é implementado no *firmware* da mesma.

A maneira como o programa encontra-se implementado garante uma velocidade máxima teórica da ordem de 40 Kbps. Entretanto, as velocidades médias medidas ficaram bem abaixo disso, em torno do 20 Kbps.

Após um estudo mais detalhado da estrutura do programa, verificou-se que a maior parte do retardo deve-se à rotina de comunicação SPI no modo *full-duplex*, que exige aproximadamente o dobro do tempo gasto pelas rotinas *half-duplex* para cada iteração.

Estes resultados sugerem que a rotina em questão seja otimizada para que uma melhoria nestes resultados possa ser alcançada. A interface RS-232, por ser parcialmente implementada em hardware, não apresenta degradação de performance significativa.

VII. CONCLUSÕES

Os resultados colhidos indicam que a implementação de uma interface UART do tipo RS-232 para barramento SPI, empregando membro da família 8051, foi adequada, considerando a destinação da mesma.

Embora o barramento SPI esteja longe se ser empregado no seu limite de velocidade, para muitas aplicações o resultado alcançado é mais do que o suficiente.

Esse fato sugere que o uso de microcontroladores pode se tornar uma solução viável, também, quando se deseja interfacear outros tipos barramentos seriais, principalmente barramentos pouco comuns e para os quais o equipamento especializado é escasso e muitas vezes por demasiado caro.

Por fim, deixamos a sugestão de se empregar microcontroladores mais elaborados e com mais recursos do que o ora empregado, na tentativa de se alcançar alternativas viáveis para comunicação com barramentos de grande interesse para as Forças Armadas brasileiras.

REFERÊNCIAS

[1] SINCLAIR, Ian R., DUNTON, John. Practical Electronics Handbook. 6ª ed. Oxford: Newnes, 2007, pp. 369-398.

[2] ZELENOVSKY, Ricardo, MENDONÇA, Alexandre. PC: um guia prático de hardware e interfaceamento. 4ª ed. Rio de Janeiro: MZ Editora, agosto 2006, pp. 551-588.

[3] ZELENOVSKY, Ricardo, MENDONÇA, Alexandre. Microcontroladores: programação e projeto com a família 8051. 1ª ed. Rio de Janeiro: MZ Editora, agosto 2005.

[4] KALINSKY, David, KALINSKY, Roee. Introduction to Serial Peripheral Interface. Embedded Systems Design, february 2002. Disponível em: <http://www.embedded.com/story/OEG20020124S0116>. Acesso em: 10 setembro 2008

[5] FREESCALE SEMICONDUCTOR INC. SPI Block Guide V4.01. 14 Jul 2004. 40 p.