EA871 – LAB. DE PROGRAMAÇÃO BÁSICA DE SISTEMAS DIGITAIS

EXPERIMENTO 13 – Interface Serial Assíncrona

Profa. Wu Shin-Ting

OBJETIVO: Apresentação de uma interface serial assíncrona.

ASSUNTOS: Interface serial assíncrona UART, comunicação serial do MKL25Z128 com PC via porta COM, programação do MKL25Z128 para processamento de sinais de uma comunicação UART.

O que você deve ser capaz ao final deste experimento?

Entender o princípio de comunicação via UART.

Programar MKL25Z128 para processamento de sinais de uma comunicação UART.

Utilizar *buffer* (circular) numa transferência serial.

INTRODUÇÃO

Um dos padrões mais usados para a comunicação entre sistemas ou partes de um sistema é o padrão **serial**. Na interface serial, os *bits* são enviados em sequência, um de cada vez, ao invés de em grupos de 8 ou mais simultaneamente. Existem duas variantes deste padrão: *síncrona* e *assíncrona*. A diferença está na presença ou não de um sinal de *clock* que sincroniza temporalmente a transmissão de *bits*. Neste experimento veremos o padrão assíncrono, que exige que os dois sistemas tenham seus *clocks* individuais bem ajustados, bem como o estabelecimento prévio da velocidade de transmissão. A transmissão de dados é *full duplex*, ou seja, ambos os lados podem transmitir e receber simultaneamente, através das linhas Tx e Rx respectivamente. A linha Tx de um lado deve ser conectada à linha Rx do outro, e vice-versa. O sincronismo é mantido através de *bits* de sincronismo, chamados *start bit* e *stop bit*. Para uma verificação simples dos possíveis erros na transmissão, pode ser incluído no caracter transmitido um *bit* de paridade [1]. O padrão de protocolo de comunicação serial assíncrono mais popular é o padrão RS-232, em que o nível lógico 1 está associado a uma tensão entre -3V a -15V enquanto o nível lógico 0 a uma tensão entre 3V a 15V. Vale ressaltar que os sinais de controle, como *start* e *stop bits*, tem a polaridade invertida. Este padrão é encontrado nas portas seriais dos PCs.

Nosso microcontrolador KL25 possui três módulos dedicados à comunicação serial assíncrona, porém com níveis de tensão compatíveis com a tensão de operação do KL25. Cada módulo `UARTx` atua essencialmente como uma "ponte" entre uma interface paralela e uma interface serial no molde do padrão RS-232 (caps.39 e 40 de [2], cap. 8 de [3]). Ele contém um circuito de transmissão (Fig. 39-1 de [2]) e um de recepção (Fig. 39-2 de [2]). Através destes circuitos os *bytes* a serem transmitidos são serializados e processados, e os *bits* recebidos são reagrupados em *bytes* automaticamente, a uma frequência pré-estabelecida. Portanto, para que um módulo UARTx opere corretamente, é necessário habilitar os seus sinais de relógio pelo módulo `SIM` (Seção 12.2.8 de [2]). É também necessário configurar os pinos através do módulo `PORT` para que estes servem a comunicação serial RX e TX. No caso do módulo UART0, pode-se ainda selecionar a fonte de *clock* da base de tempo (Seção 8.3.1 de [3]) através do campo `SIM_SOPT2_UART0SRC` (Seção 12.2.3 de [2]).

No *kit* de desenvolvimento FRDM-KL25Z, o módulo UART0 é conectado ao microcontrolador do OpenSDA (OpenSDA MCU) e este a uma interface serial USB (*Universal Serial Bus*), através do qual

pode-se conectar com a porta COM de um computador-hospedeiro (Seção 5.2 de [4]). No OpenSDA estão residentes o *P&E Debug Application,* com o qual podemos depurar os códigos executados no nosso MCU, e a interface CDC (USB *Communications Device Class*) que faz a "ponte" entre as linhas Tx e Rx do processador-alvo e a interface USB. Portanto, se tivermos um emulador de terminal no nosso computador-hospedeiro podemos digitar caracteres no terminal e enviá-los para serem processados no MCU, e vice-versa.

Na Seção 2.8.1 em [5] encontram-se dicas para instalar um *plugin* "Terminal" no ambiente integrado de desenvolvimento *CodeWarrior*. A comunicação do "Terminal" com o MCU se dá através do módulo UART0. Para você abri-lo no ambiente IDE, basta percorrer o caminho **Windows > Show View > Other … > Terminal**. Aparecerá uma aba na janela do canto inferior direito e uma janela "*Terminal Settings*", através da qual você pode configurar os parâmetros de comunicação serial setados no módulo UARTx, como ilustra a Fig. 1. Para se comunicar, por exemplo, com o projeto seria_uart.zip uart [7] a configuração do "Terminal" deve ser: *baud rate* 115200, caracter de 8 *bits*, sem *bit* de paridade, 1 *stop bit* e sem fluxo de controle. Vale observar aqui que o módulo UARTx superamostra o sinal no canal receptor na busca pelas bordas de descida (Seção 39.3.3.1 em [2]). Uma visão geral sobre "Terminal" é dada em [6].

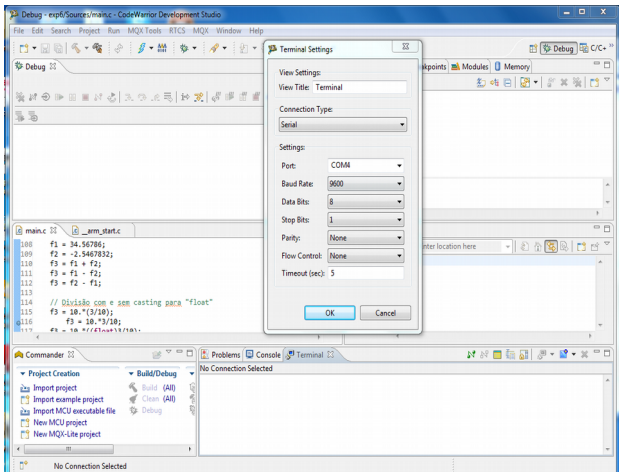

Fig. 1: *Pop-up* menu de configuração dos parâmetros seriais do lado do computador-hospedeiro.

É possível ainda configurar um módulo UARTx para operar por interrupções durante uma comunicação, como demonstra serial_uart_interrupr.zip [8]. Um módulo UARTx consegue diferenciar várias condições de interrupção que são classificadas em três classes de causas de interrupção: uma é relacionada com a recepção, outra é relacionada com o canal de transmissão e a terceira é relacionada com diversas condições de erro que possam surgir durante uma transferência (Seção 39.3.5 em [2]). Há dois registradores de estado, UARTx_S1 (Seção 39.2.5 em [2]) e UARTx_S2 (Seção 39.2.6 em [2]), em cada módulo para armazenar estas condições. Porém, para que estas interrupções externas sejam processáveis pelo núcleo do nosso microcontrolador, é necessário (1) configurar o módulo NVIC para que este arbitre o momento de atendimento pelo processador, e (2) customizar a rotina de serviço para tratamento das diversas condições de interrupção de cada módulo UARTx (Seção 8.4.1 em [3]). Vale ressaltar que, conforme a Tabela 3-7 em [2], o nosso processador associa somente **um número de IRQ a cada um dos três módulos** UARTx disponíveis no nosso microcontrolador.

A velocidade de processamento dos módulos UARTx é muito menor do que a do processador. Para compatibilizar os dois passos distintos de processamento sem sacrificar em demasiado a capacidade de processamento do processador, é bastante comum usar nos projetos de sistemas embarcados os ***buffers* circulares** [9] para armazenar os dados de entrada e de saída seriais. Um buffer circular é, de fato, uma estrutura de dados fila com as pontas conectadas sobre a qual são aplicadas as operações que

envolvem somente os ponteiros. O projeto serial_cbuffer.zip [10] demonstra o uso de um *buffer* circular numa transmissão serial assíncrona.

EXPERIMENTO

Neste exeprimento vamos desenvolver o projeto **serial** que recebe, **por interrupção,** as **linhas** de até 80 caracteres digitadas pelos usuário via Terminal e as mostre no visor do LCD por linha. Ou seja, ao digitar "Return", codificado por um caractere de controle "\r", deve-se alternar a linha do visor do LCD e iniciar o envio da linha digitada para esta nova linha do visor. E quando se aciona uma das três chaves do *shield* FEEC 871, é enviado, **também por interrupção**, ao Terminal uma **nova linha** de mensagem: "Mensagem do shield FEEC 871: chave x acionada", onde x corresponde ao nome da chave acionada (NMI, IRQ5 ou IRQ12). Fig. 2 esquematiza a arquitetura do projeto usando dois *buffers* circulares.

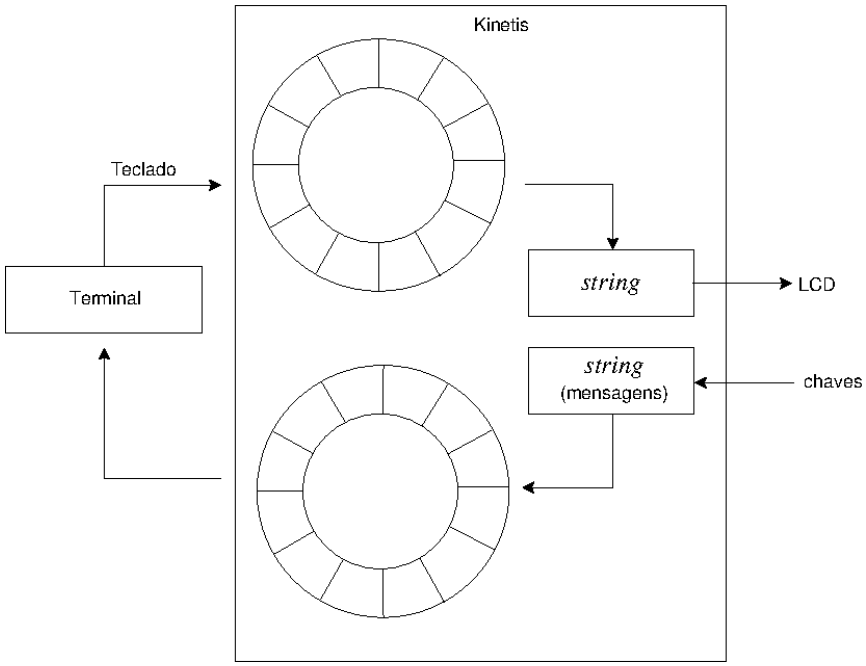

Fig. 2: Projeto **serial**.

O modo de operação do módulo UART0 especificado é *baud rate 19200*, caracter de 8 *bits*, 2 *stop bits*, *bit* de paridade ímpar e taxa de amostragem 9x.

1. Em qual espaço de memória estão mapeados os registradores dos três módulos de UARTx?
2. Ao digitar a tecla "Return" são enviados para o módulo UART0 um caractere de controle, "CR (*Carriage Return*)" = '\r' = 0xD. Como você detectaria este caractere por interrupção e comunicar o fato ao fluxo principal para que este processe a nova linha digitada, colocando os caracteres numa *string*? Dê um motivo do uso de um vetor de caracteres (*string*) para extrair os caracteres de uma linha digitada antes do seu envio para o LCD. Justifique. Dica: As *strings* processadas pelas funções de C tem como terminador '\0' e uma estrutura linear.
3. A quantidade de caracteres digitados por linha pode ser maior do que a quantidade máxima suportada por linha no LCD (16 caracteres). Usualmente é considerado 80 caracteres por linha para Terminal. Como você aplicaria a instrução "*Shift entire display to the left*" (**comando 0x18**) do LCD para escrever toda a mensagem a uma velocidade compatível com a percepção humana? Justifique. Dica: O comando 0x18 preserva o endereço do cursor do LCD e desloca

as duas linhas do seu visor de uma casa para esquerda, deixando "vagos" os endereços 0x0F e 0x4F de DDRAM.

4. Quais caracteres de controle você deve anexar à mensagem enviada ao Terminal para que ela seja mostrada numa nova linha e sem sobrepor as linhas existentes? Justifique. Dica: Os caracteres de controle '\r' e '\n' posicionam o cursor, respectivamente, para a primeira coluna do Terminal (início de uma linha) e para uma nova linha.
5. Como você enviaria, por interrupção, as mensagens inseridas no *buffer circular* ao Terminal?
6. Escreva o pseudocódigo do fluxo de controle principal do seu projeto, onde se encontram as operações sobre os dois *buffers* circulares para o envio de *strings* tanto para o LCD quanto para o Terminal.
7. Implemente o aplicativo **serial** em C.
8. Documente todas as funções que não foram geradas pelo IDE CodeWarrior.

RELATÓRIO

Para este experimento, responda as questões 1 a 6 num arquivo em **pdf**, implemente e documente o projeto ***serial*.** Exporte o projeto no ambiente IDE CodeWarrior para um arquivo em formato zip. Suba **os dois arquivos, em separado**, no sistema *Moodle*. Não se esqueça de identificar todos os seus arquivos de códigos com a palavra reservada "@author" de Doxygen.

REFERÊNCIAS

[1] Jimb0. Serial Communication.
https://learn.sparkfun.com/tutorials/serial-communication
[2] KL25 Sub-Family Reference Manual – Freescale Semiconductors (doc. Number KL25P80M48SF0RM), Setembro 2012.
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KL25P80M48SF0RM.pdf
[3] Kinetis L Peripheral Module Quick Reference – Freescale Semiconductors, Setembro 2012.
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KLQRUG.pdf
[4] FRDM-KL25Z User's Manual
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/FRDMKL25Z.pdf
[5] Wu S.-T. Ambiente de Desenvolvimento – *Software*
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/apostila_C/AmbienteDesenvolvimentoSoftware.pdf
[6] Joel_E_B e Jimb0. Serial Terminal Basics.
https://learn.sparkfun.com/tutorials/terminal-basics
[7] serial_uart.zip
http://www.dca.fee.unicamp.br/cursos/EA871/2s2020/ST/codes/serial_uart.zip
[8] serial_uart_interrupt.zip
http://www.dca.fee.unicamp.br/cursos/EA871/2s2020/ST/codes/serial_uart_interrupt.zip
[9] Wu S.-T. Estrutura de Dados
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/apostila_C/EstruturaDados.pdf
[10] serial_cbuffer.zip
http://www.dca.fee.unicamp.br/cursos/EA871/2s2020/ST/codes/serial_cbuffer.zip

Agosto de 2016
Revisado em Fevereiro de 2017
Revisado em Julho de 2017
Revisado em Novembre de 2020