EA871 – LAB. DE PROGRAMAÇÃO BÁSICA DE SISTEMAS DIGITAIS

EXPERIMENTO 7 – 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) de dados como uma forma de ajuste dos tempos de processamento de um transmissor e de um receptor.

INTRODUÇÃO

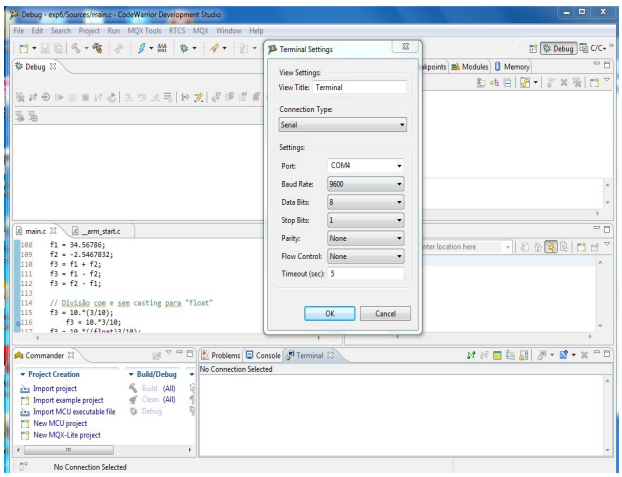
Um dos padrões mais usados para a comunicação entre sistemas ou partes de um sistema é o padrão **serial**. Na interface serial, os *bits* são enviados em sequência, um de cada vez, ao invés de em grupos de 8 ou mais simultaneamente. Existem duas variantes deste padrão: *síncrona* e *assíncrona*. A diferença 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. 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 (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.

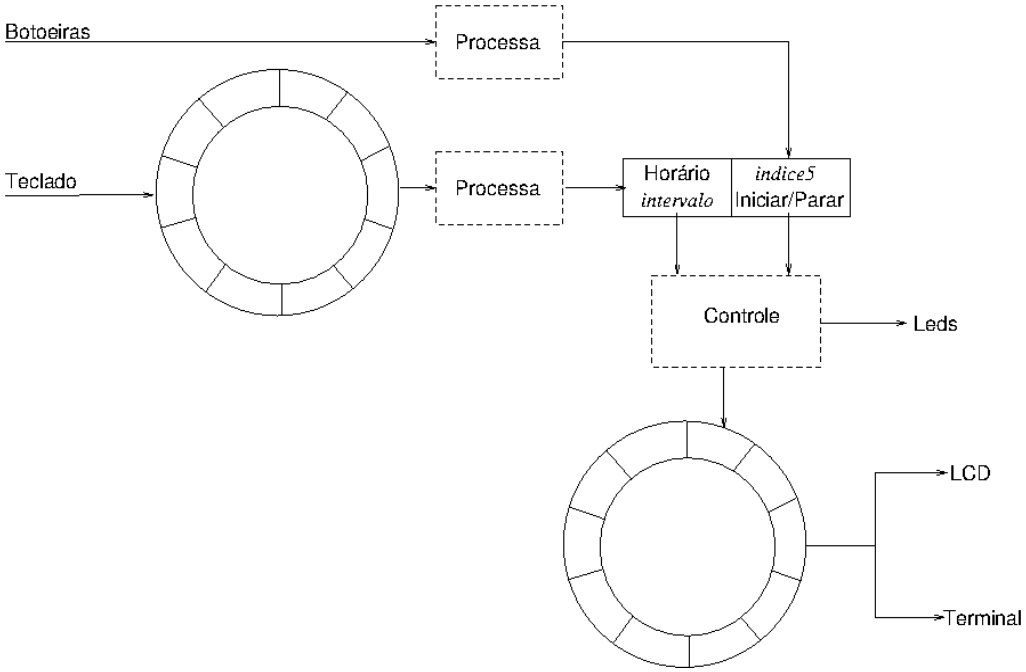
Na Seção 2.8.1 em [5] encontram-se dicas para instalar um *plugin* "Terminal" no ambiente integrado de desenvolvimento *CodeWarrior*. A comunicação do "Terminal" com o MCU se dá através do módulo UART0. Para você abri-lo no ambiente IDE, basta percorrer o caminho **Windows > Show View > Other … > Terminal**. 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. Para se comunicar, por exemplo, com os projetos uart_polling.zip [7] e uart_interrupcao.zip [8] a configuração do "Terminal" deve ser: *baud rate 4800*, caracter de 8 *bits*, 1 *stop bit*, sem *bit* de paridade 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]).



É possível ainda configurar um módulo UARTx para operar por interrupções durante uma comunicação. Ele 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 exceção **a cada um dos três módulos** UARTx disponíveis no nosso microcontrolador.

Neste experimento vamos praticar a programação do módulo UART0 agregando mais valores ao seu projeto de temporizador e de cronômetro do experimento 6 [12]. Vamos adicionar a funcionalidade de relógio digital ao projeto, em que o ajuste do horário no formato HH:MM:SS.s seja feito pelo teclado. O programa converte esta sequência de caracteres em horas, minutos, segundos com uma casa decimal que serão atualizados automaticamente pela rotina de serviço `IRQHandler_PIT` em cada 100ms. O horário atual é exibido centrado na primeira linha do visor do LCD, substituindo a cor do *led* RGB.

Configuração do intervalo de tempo do temporizador, na unidade de segundos com uma casa decimal, também será feito pelo teclado além da botoeira PTA5. Além disso, é possível consultar o intervalo de tempo programado para temporizador via "Terminal". **Novos estados são adicionados ao nosso sistema: 5, modo de configuração do relógio; 6, modo de consulta do relógio; 7, modo de consultado intervalo do temporizador programado; 8, modo de consulta do último tempo cronometrado**. Para entrar no modo de configurar o intervalo do temporizador, o usuário deve digitar "0"; para entrar no modo de configurar o horário, "1"; para entrar no modo de consulta do horário, "2"; para entrar no modo de consulta do intervalo programado, "3"; e para entrar no modo de consulta do último valor registrado pelo cronômetro, "4". E, a fim de melhorar a interação com usuários, **os caracteres digitados pelo usuário devem ser sempre ecoados no "Terminal"**. Usamos ainda dois *buffers* circulares [9] para armazenar os dados de entrada e de saída via "Terminal", de forma que não seja necessário introduzir *delays* no fluxo de controle do nosso programa para compatibilizar a velocidade de processamento do processador do nosso microcontrolador e a velocidade de processamento do "Terminal" e do LCD, que são significativamente mais lentos. O módulo UART0 deve ter a seguinte configuração: *baud rate 19200*, caracter de 8 *bits*, 1 *stop bit*, sem *bit* de paridade e taxa de amostragem 9x.



EXPERIMENTO

1. *Vamos entender como é o protocolo de comunicação serial assíncrona, como opera um módulo UARTx no nosso microcontrolador para fazer conversão entre o formato paralelo e o formato serial de um dado, e como faz acesso direto aos dados da porta serial do nosso PC através de um aplicativo emulador de terminal?* Leia atentamente o tutorial sobre a comunicação serial [1] para se ter uma noção sobre o protocolo de comunicação serial, a Seção 8.3 em [3] para entender como um módulo UARTx opera, e o tutorial sobre um terminal serial [11] para conhecer os termos comumente utilizados.

2. *Vamos ver como os conceitos são traduzidos na prática através de dois projetos: uart_polling.zip [7]* e *uart_interrupcao.zip [8]*. Em amboso os projeto é mostrada no "Terminal" a cor "VERMELHO", "BRANCO", "AMARELO", "CIANO", "MAGENTA" e "PRETO", quando se digita no "Terminal" V/v, B/b, A/a. C/c, M/m e outros caracteres, respectivamente. As palavras são agrupadas em linhas com 10 (caractere de controle "CR

(C*arriage Return*)” = '\r' = 0xD e “mudar linha” = '\n' = 0xA) com um espaço branco (' ') entre elas. No primeiro projeto a comunicação entre o “Terminal” e o micro-controlador é por *polling*, enquanto no segundo é por interrupção. Além disso, no primeiro projeto a digitação de um caractere é suficiente para atualizar o nome da cor, enquanto no segundo projeto é necessário entrar o caractere de controle '\r' (<ENTER>) para que o caractere digitado seja processado. Este segundo exemplo tem o objetivo de ilustrar como se processa uma unidade de informação constituída por vários caracteres ASCII. Primeiro, é necessário definir um caractere de controle que identifica o término de uma unidade de informação. No caso, é o caractere de controle '\r'.

O fluxo de dados no programa uart_polling.zip [7] é simples. Depois de configurado apropriadamente o módulo UART0 através da rotina `initUART`, o programa fica monitorando num laço, através do registrador de estado UART0_S1, se há alguma entrada no canal de receptor. Caso sim, é alterada a cor que é enviada ao “Terminal”. Como a serialização e o envio envolvem vários ciclos de relógio, é aguardada a conclusão do envio de um caractere antes de mandar um outro. Observe que, além de habilitar o sinal de relógio do módulo, precisamos configurar os parâmetros de comunicação serial assíncrona através de alguns registradores de controle (Seção 39.2 em [2]): quantidade de *bits* por caractere, quantidade de *stop bits*, *bit* de paridade, e *baud rate*. No nosso micro-controlador setamos, de fato, o divisor do relógio-base no campo SBR de 13 *bits* separados em dois registradores, `UARTx_BDH` (Seção 39.2.1 em 2]) e `UARTx_BDL` (Seção 39.2.2, p.726, em [2]), que resulta no valor de *baud rate* desejado (Seção 8.3.2, p. 78, em [3]). Observe ainda que a frequência do relógio-base é a frequência do relógio do módulo configurada (20971520Hz/1), dividida pela quantidade de amostras por *bit*, pois no nosso micro-controlador é feita uma superamostragem nos sinais recebidos e a taxa de superamostragem, configurável pelo campo `UARTx_C4_OSR` (Seção 39.2.11 em [2]), varia de 4x até 32x. Em termos de códigos, adicionamos dois novos arquivos `uart.*` na biblioteca.

Ao invés de monitorar o registrador de estado, podemos habilitar o mecanismo de interrupção do módulo UARTx que gera automaticamente interrupções quando há um novo dado no receptor ou quando o transmissor fica pronto para uma nova transmissão. O projeto art_interrupcao.zip [8] é uma versão com interrupção. Note que é usada a mesma rotina de configuração do módulo UART0 do projeto anterior. Só precisamos (1) habilitar a interrupção com `enableRIEInterrup`, (2) configurar através da rotina `enableNVICUART` o módulo NVIC para processá-las com prioridade configurada em 1, uma vez que estas interrupções são consideradas externas ao núcleo (Seção 3.3.2.3 em [2]), e (3) programar a rotina de serviço `UART0_IRQHandler`, que organizamos no arquivo `handler.c`, para tratar os dois eventos de interrupção: recepção e transmissão. Observe que usamos uma variável de estado `flag_recepcao` para coordenar o fluxo de controle da rotina principal com o da rotina de serviço uma vez que a entrada do caractere de controle '\r' acontece de forma assíncrona. E, para evitar que novos dados sejam inseridos enquanto se extrai o conteúdo do `buffer_receptor`, uma estratégia simples é desabilitar momentaneamente a interrupção do canal de recepção enquanto se faz uma cópia do conteúdo do `buffer_receptor` na variável local `buffer`. Este trecho do código de cópia que faz acesso concorrente a um recurso compartilhado é conhecido como **região crítica**. No projeto uart_interrupcao.zip [8], como a entrada do usuário é bem “comportada”, bem pausada, não foi aplicada esta última estratégia na execução de uma região crítica. No entanto, para ecoar os caracteres digitados pelo usuário neste mesmo projeto, o trecho de código de eco compete com o trecho do código que escreve nomes de cores o acesso ao

“Terminal”. Para coordenar estes acessos sem desabilitar o mecanismo de interrupção, adotou-se no projeto o sequenciamento dos acessos, forçando que o canal de receptor aguarde o envio de todos os nomes armazenados no `buffer_transmissor` antes de efetuar ecos.

A espera entre envios de caractere numa comunicação serial assíncrona é um grande gargalo no desemepnho do sistema. Como melhorar a fluidez dos dados numa transmissão? Estrategicamente, usamos a estrutura de dados ***buffer* circular** [9] para “estocar” os caracteres produzidos que um consumidor não consegue consumir em tempo no projeto uart_interrupcao.zip [8]. Implementamos um *buffer* circular com uso de um arranjo e dois ponteiros, `head` e `tail`. Estes dois ponteiros nos permitem controlar o fluxo dos dados num arranjo sem deslocar os dados armazenados na memória. Os códigos desta estrutura de dados estão organizados nos arquivos `estrutura.*`.

3. *Vamos praticar o que aprendemos?*
   a) Em relação à execução dos projetos uart_polling.zip [7] e uart_interrupcao.zip [8]: substitua “xxxx” e complete as rotinas definidas no arquivo `uart.c`.
   b) Em relação à configurabilidade do módulo UART0: Como se determinam os valores dos campos UART0_C4_OSR, UART0_BDH_SBR, UART0_BDL_SBR a partir dos valores de taxa de superamostragem `taxa` e da taxa de transmissão `baud_rate` especificadas? Aumente a flexibilidade da rotina `initUART` de forma que seja possível configurar através dela o módulo UART0 para operar em todas as possíveis taxas de amostragem (4x a 32x) e todas as *baud rates* que o “Terminal” suporta (300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200) quando a combinação especificada for factível; caso contrário, retorna um erro (0). O protótipo da nova função seria: `uint8_t initUART (uint8_t taxa, uint32_t baud)`. Não se esqueça de setar o *bit* `UART0_C5_BOTHEDGE` quando a taxa de superamostragem for menor ou igual a 7x (Seção 39.2.12 em [2]).
   c) Em relação à implementação de um *buffer* circular: complete as rotinas no arquivo `estrutura.c` com base em [9]. Note que, ao invés de dados do tipo `unsigned int`, os dados a serem armazenados no *buffer* circular do nosso projeto são do tipo `char`. Adaptações são necessárias.
   d) Em relação ao processamento dos caracteres armazenados no *buffer* circular `buffer_receptor`: implemente uma função `ConvString2HMS(char *string, uint8_t *hora, uint8_t *minuto, float *segundo)` nos arquivos `util.*` capaz de extrair a partir de uma *string* HH:MM:SS.s as horas, os minutos e os segundos definidos pelo usuário. Esta função poderia ainda utilizar duas outras funções para converter a sub-string HH e MM para inteiros (`ConvString2UI(char *s, unsigned int *j)`) e a sub-string SS.s para um valor em ponto flutuante (`ConvString2F (char *s, float *f)`).
   e) Em relação ao processamento por interrupção dos caracteres armazenados no *buffer* circular `buffer_transmissor`: Veja na Seção 39.2.5 em [2] que é sempre gerada uma interrupção quando o canal de transmissor fique vazio, pois ele está pronto para enviar um outro caractere. O que acontece se uma interrupção no canal de transmissor é gerada com o `buffer_transmissor` vazio? E o que acontece quando há um novo caractere para ser enviado e o `buffer_transmissor` estiver cheio? Quais são as soluções adotadas no projeto [8] [8]?
   f) Em relação ao projeto, atualize o diagrama de estado do experimento 6 [12] com novos estados.

4. *Vamos ver se você entendeu? O relatório deste roteiro é dividido em duas partes*
   a) (50%) Relatório 8a: Elabore um relatório com as soluções das questões (b), (d), (e) e (f) do item 3 e implemente os códigos conforme solicitados nas questões (a)-(d) do item 3. Suba tanto o relatório quanto as rotinas `uart.*`, `util.*`, `estrutura.*` modificadas e compactadas num arquivo **zip** no sistema Moodle.
   b) (50%) Relatório 8b: Estenda a funcionalidade do seu programa do roteiro 6 [12] conforme a especificação deste roteiro: um relógio digital com as funcionalidades de temporizador e cronômetro. Suba no Moodle o seu projeto exportado (não se esqueça de limpá-lo antes) junto com os pseudo-códigos do fluxo de controle principal (`main`) e da rotina de serviço `UART0_IRQHandler`.

REFERÊNCIAS

Todas as referências podem ser encontradas nos *links* abaixo ou ainda na página do curso.

[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] uart_polling.zip
http://www.dca.fee.unicamp.br/cursos/EA871/2s2017/EM/codes/uart_polling.zip
[8] uart_interrupcao.zip
http://www.dca.fee.unicamp.br/cursos/EA871/2s2017/EM/codes/uart_interrupcao.zip
[9] Wu S.-T. Estrutura de Dados
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/apostila_C/EstruturaDados.pdf
[10] biblioteca.zip
http://www.dca.fee.unicamp.br/cursos/EA871/2s2017/EM/codes/biblioteca.zip
[11] Joel E. B e Jimbo Maettu_this. Serial Terminal Basics.
https://learn.sparkfun.com/tutorials/terminal-basics/all
[12] Wu, Shin-Ting. EA871 – Roteiro 6 – 2s2017
http://www.dca.fee.unicamp.br/cursos/EA871/2s2017/EM/roteiros/exp6.pdf

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