# EA871 – LAB. DE PROGRAMAÇÃO BÁSICA DE SISTEMAS DIGITAIS

## EXPERIMENTO 6 – Interface Paralela

Profa. Wu Shin-Ting

**OBJETIVO:** Apresentação de interfaces de periféricos paralelos.

**ASSUNTOS:** Configuração e programação do LCD, programação do MKL25Z128 para processamento de interface peralela com sinais GPIO.

### O que você deve ser capaz ao final deste experimento?

Programar LCD.

Programar MKL25Z128 para processamento de sinais GPIO.

Utilizar uma técnica para uso multiplexado de um conjunto de pinos de um microprocessador.

Usar diferentes formas para declarar uma *string* de caracteres.

Saber converter valores numéricos para códigos ASCII.

Organizar arquivos de rotinas de uso frequente em bibliotecas no ambiente CodeWarrior.

## INTRODUÇÃO

No experimento 1 fizemos uma breve revisão dos principais conceitos relacionados com a arquitetura de um microcomputador/microcontrolador e procuramos relacioná-los com a arquitetura do microcontrolador MKL25 que utilizaremos nesta disciplina. A principal característica destes *hardwares* é a programabilidade, ou seja, é possível customizar a operação dos seus circuitos através de um programa dedicado. Conhecemos no experimento 2 o ambiente *CodeWarrior* para desenvolvimento de tais programa programas com uso da linguagem C, e aprofundamos no experimento 3 a arquitetura do nosso MCU através do seu repertório de instruções *Thumb*. O entendimento dos ciclos de instrução das instruções *Thumb* nos permitiu elaborar um programa que apresenta maior precisão no controle de tempo no experimento 4. E, no experimento 5 vimos como as declarações dos tipos de dados em linguagem C estão relacionadas com a ocupação dos dados na memória e aprendemos que a escolha apropriada dos tipos de dados pode impactar diretamente no uso da memória.

Nesta introdução inicial utilizamos somente sinais digitais e periféricos simples, os *leds* e as botoeiras, para ilustrar os conceitos. Vimos que somente para controlar a entrada e a saída de sinais digitais nos pinos do nosso MCU precisamos configurar os registradores dos módulos SIM (*System Integration Module*) (veja Capítulo 12 de [1]), PORT (*Port Control and Interrupts*) (veja Capítulo 11 de [1]) e GPIO (*General-Purpose Input/Output*) (veja Capítulo 41 de [1]). A partir deste experimento vamos aplicar o nosso MCU no controle de alguns periféricos com um circuito de interface mais elaborado. No contexto desta disciplina, uma interface é um conjunto de conexões lógicas (protocolos de comunicação) e físicas (durações dos níveis lógicos, valores de tensão) utilizadas entre ele e os periféricos controlados por ele, para assegurar a compatibilidade elétrica, funcional e temporal entre eles. Felizmente, uma grande parte dos circuitos de interface já se

encontra integrada como módulos no nosso MCU. Com isso, dependendo do tipo de periférico, o projeto de uma interface pode se reduzir à escolha de módulos que tenham características elétricas, funcionais e temporais mais próximas possíveis do periférico de interesse e à programação do nosso MCU. Nesta disciplina utilizaremos os periféricos disponíveis na placa auxiliar [2] encaixada diretamente sobre o *kit* de desenvolvimento FRDM-KL25Z [3]. Tais periféricos, como as botoeiras que vimos no experimento 5, já se encontram devidamente conectados aos pinos do MCU. Assim, para desenvolver um programa que os controle, é somente necessário: (1) entender as características funcionais e temporais dos periféricos; (2) identificar os pinos e as portas em que cada um está conectado; e (3) identificar os módulos mais apropriados do MCU para controlá-los.

Neste experimento vamos tratar de acessos diretos de leitura e escrita dos pinos do MCU via o módulo GPIO (*General Purpose Input /Output*) para controlar o *display* LCD e o conjunto de 8 *leds* da placa auxiliar [2]. Nos experimentos anteriores, já vimos a programação do módulo GPIO para controlar três *leds* do *kit* de desenvolvimento FRDM-KL25Z [3] e dois *push-buttons* da placa auxiliar. Os valores dos pinos onde os *leds/push-buttons* estão ligados eram escritos/lidos em separado. E, agora, vamos entender melhor como configurar o módulo GPIO para uma transmissão paralela, ou seja, enviar vários *bits*, mais especificamente um *byte*, ao mesmo tempo para LCD e para o conjunto de 8 *leds* vermelhos de forma multiplexada.

## EXPERIMENTO

1. *Vamos entender como se programa um microcontrolador para controlar um LCD com interface paralela?* Leia atentamente o tutorial sobre o LCD [4] e o *datasheet* do LCD [5] para entender como um *display* LCD funciona.
2. *Vamos ver como os conceitos são traduzidos na prática através de um* programa [6]? O programa envia para a segunda linha do *display* do LCD a cor que está sendo exibida pelo *led* RGB precedido de um ícone “sino” e, enquanto o usuário mantém uma botoeira pressionada, o programa acende os *leds* vermelhos. Tanto o LCD quanto os *leds* vermelhos são dispositivos digitais e compartilham o uso dos pinos 0 a 7 da porta C, configurados como pinos digitais, conforme mostra o esquemático em [1]. É possível controlar separadamente os dois periféricos?

   Vamos analisar mais cuidadosamente o esquemático em [1] e o diagrama de tempo do LCD na Seção 8 em [5]. Observe no esquemático que o conjunto de 8 *leds* é conectado com 8 pinos do MCU através de MC74HC573N. Este CI é um *latch* de 8 *bits* provido de dois pinos de controle: /OE (*output enable*) e LE (*latch enable*) [7]. O primeiro pino está aterrado, portanto a saída do *latch* é sempre liberada e o segundo pino está ligado no pino 10 da porta C, ou seja, o estado do *latch* só será alterado se o pino estiver no nível lógico 1. Em relação ao LCD, podemos verificar pelo diagrama de tempo que o LCD só captura os dados na sua entrada na borda de descida do sinal E que está conectado no pino 9 da porta C. Com isso, podemos controlar os dois periféricos separadamente desde que manipulemos adequadamente os sinais de controle no pino 9 e no pino 10 da porta C.

   Habilitando o pino 10, os estados dos *leds* vermelhos são atualizados conforme os valores lógicos nos pinos (1, aceso e 0, apagado). E quando ocorrer uma borda de descida no sinal do pino 9, o estado do LCD poderá variar se

a) ele for inicializado seguindo a sequência dada na Seção 10 em [5], e
b) o sinal R/\W estiver em 0.

Veja na tabela da Seção 9 em [5] que quando o sinal RS, conectado no pino 8, for zero, os valores binários nos pinos 0-7 são interpretados como códigos de controle, tais como limpar o *display* e alterar o modo do *cursor*. E quando RS estiver em 1, os sinais D0-D7 do LCD são interpretados como endereços dos *bitmaps* pré-armazenados na memória do LCD. Estes bitmaps são acessados e carregados na memória DDRAM do *display* do LCD para então serem mostrados no visor. Os endereços dos *bitmaps* disponíveis na memória são dados na Seção 12 em [5]. Note que os endereços dos caracteres alfa-numéricos correspondem exatamente aos códigos ASCII. Portanto, se quisermos exibir um caractere no *display*, só precisamos passar para LCD o seu código em ASCII. Como controlamos a posição de um caractere no *display*?

Veja na tabela da Seção 9 em [5] que há dois códigos de controle para setar o endereço corrente da memória DDRAM e da memória CGRAM. Na Seção 11 temos os endereços de cada elemento das duas linhas do *display* do LCD. Por exemplo, se quisermos exibir "AB" no meio da segunda linha do LCD, precisamos setar o endereço DDRAM enviando 0x46 com RS=0 e depois enviar o valor 0x41 seguido de 0x42 com RS=1. Vale observar que o endereço DDRAM é incrementado automaticamente após um acesso. E onde está a memória CGRAM? Ela é uma memória de 16x8 *bytes* mapeada nos 16 primeiros endereços da matriz de *bitmaps*, reservada para adicionar *bitmaps* customizados. Como cada *bitmap* é composto de 8 linhas (5x8) e cada linha é representada por 1 *byte*, usamos 8 *bytes* para representar um *bitmap* na memória CGRAM. Com a instrução que seta o endereço da memória CGRAM no *address counter* e uma instrução que transfere 8 *bits* de dados para a memória do LCD, podemos gravar um novo bitmap com 8 acessos. Veja na Seção *CGRAM Creating custom character* em [4] como se grava um *bitmap* na memória CGRAM através de uma interface de 8 *bits* de dados do LCD.

Se os textos que vemos num visor de um LCD são, de fato, uma "montagem" de bitmaps, como podemos exibir um valor numérico no seu visor? A ideia é simples: basta convertermos estes valores numa sequência de caracteres ASCII em que cada caractere ASCII corresponde a um dígito do número. Chamo também atenção aos tempos de processamento do LCD apresentados na tabela da Seção 9 em [5], variando de 39us a 1.53ms. Estes tempos são muito maiores do que o tempo de processamento de uma instrução do nosso micro-controlador! Portanto, quando enviamos um comando para o LCD, devemos aguardar um intervalo de tempo correspondente ao tempo de processamento daquele comando antes de enviar o próximo comando para que os comandos não se "embaralhem". [9]

Finalmente, observe que, em relação ao experimento 5, adicionamos somente dois novos arquivos, `lcdled.c` e `lcdled.h`, para gerenciar a interface entre o nosso micro-controlador e o *display* LCD e os 8 *leds* vermelhos. As rotinas restantes são as que desenvolvemos nos experimentos anteriores. Para facilitar este reuso, podemos agrupar as rotinas numa biblioteca conforme a receita dada na Seção 2.9 em [10].

Vale ainda acrescentar que o formato de documentação dos códigos no programa [6] é ligeiramente diferente em relação aos códigos dos experimentos anteriores. O objetivo é mostrar que Doxygen suporta diferentes estilos de documentação [9].

3. *Vamos praticar o que aprendemos?*
   a) Importe o programa [6] na sua área de trabalho do CodeWarrior. Substitua os arquivos `atrasos.c`, `ledRGB.c` e `pushbutton.c` pelos seus arquivos do experimento 5. Em seguida, substitua "xxxx" no arquivo `lcdled.c` pelos valores adequados e `main.c`. Gere um executável e veja se são mostrados no *display* LCD a cor do *led* que você selecionou precedido de um sino.
   b) Complete a rotina ConvF2String no arquivo util.c de forma que um valor em ponto flutuante j (com no máximo uma casa decimal) seja convertido numa sequência de caracteres ASCII correspondentes aos dígitos do número.

4. *Vamos ver se você entendeu? O relatório deste roteiro é dividido em duas partes*
   a) (30%) Relatório 6a: Submeta no sistema Moodle o arquivo `main.c` com a definição de um bitmap "ê" e o projeto exportado em zip da biblioteca de todas as rotinas *.c, exceto `main.c` e `sa_mtb.c`.
   b) (60%) Relatório 6b: Estenda a funcionalidade do seu programa do roteiro 5 integrando a visualização no visor do LCD a cor dos *leds* no formato "Cor: xxx"(segunda linha) e a frequência de piscadas dos *leds* em ponto flutuante no formato "Frequência: xxx" (primeira linha). O primeiro *led* (o que fica no canto) deve ficar aceso quando a cor não for preta, enquanto todos os outros *leds* devem manter apagados. Elabore o relatório sobre este projeto segundo o modelo, justificando as suas decisões de projeto que atenda as especificações. O relatório deve conter a identificação completa, uma breve descrição das funções dos *bits* dos registradores utilizados, o algoritmo implementado em pseudo-códgio, os testes de validação, conclusões acerca os resultados obtidos e o código-fonte documentado de acordo com a sintaxe de Doxygen. Suba-o junto com o código-fonte main.c no Moodle.

## REFERÊNCIAS

Todas as referências podem ser encontradas na página do curso.


[1] KL25 Sub-Family Reference Manual
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KL25P80M48SF0RM.pdf
[2] Wu Shin-Ting e A.A.F. Quevedo. Ambiente de Desenvolvimento – *Hardware*
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/apostila_C/AmbienteDesenvolvimentoHardware.pdf
[3] FRDM-KL25Z User's Manual
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/FRDMKL25Z.pdf
[4] LCD Tutorial for Interfacing with Microcontrollers. Rickey's World
http://www.8051projects.net/lcd-interfacing/index.php
[5] *Datasheet* do *display* LCD.
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea079/datasheet/AC162A.pdf
[6] lcdled.zip
http://www.dca.fee.unicamp.br/cursos/EA871/1s2017/ST/codes/lcdled.zip
[7] *Datasheet* 74573
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/datasheet/74HC_HC573.pdf
[8] D. S. Oliveira e Wu Shin-Ting. Linguagem C: Dados
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/apostila_C/RepresentacaoDados.pdf
[9] Documenting the code. Doxygen
https://www.stack.nl/~dimitri/doxygen/manual/docblocks.html
[10] Wu Shin-Ting e A.A.F. Quevedo. Ambiente de Desenvolvimento – *Software*

ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/apostila_C/AmbienteDesenvolvimentoSoftware.pdf

Agosto de 2016
Revisado em Fevereiro de 2017