EA871 – LAB. DE PROGRAMAÇÃO BÁSICA DE SISTEMAS DIGITAIS

EXPERIMENTO 5 – 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 paralela 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 microcontrolador e procuramos relacioná-los com a arquitetura do microcontrolador MKL25 que utilizamos 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 programas com uso da linguagem C, e aprofundamos no experimento 3 a arquitetura do nosso MCU através do seu repertório de instruções *Thumb* e o entendimento dos ciclos de instrução das instruções *Thumb* nos permitiu elaborar um programa que tem uma precisão maior no controle de tempo nos nossos programas. E, no experimento 4 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.

Nessa introdução 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 o nosso microcontrolador 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 ainda os periféricos disponíveis no *shield* EA871 [2] encaixado sobre o *kit* de desenvolvimento FRDM-KL25Z [3]. Tais periféricos, como as botoeiras que vimos no experimento 4, já se encontram devidamente conectados aos pinos do microcontrolador. Assim, para desenvolver um programa que os controle, basta: (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* do *shield* EA871 [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 implementa um temporizador e parcialmente um cronômetro. Como em ledRGBPB.zip, o intervalo de tempo em que a botoeira PTA5 é pressionada configura o intervalo de contagem do temporizador. A função de temporizador é iniciada quando se aperta a botoeira PTA12. E quando a botoeira PTA4 é pressionada inicia-se a função de cronômetro que pára quando a botoeira PTA4 é pressionada novamente. Quando se pressiona a botoeira PTA5, é enviada para a primeira linha do *display* do LCD a cor que está sendo exibida pelo *led* RGB. E quando se pressiona PTA4 ou PTA12, o programa acende os *leds* vermelhos do *shield* EA871 enquanto o usuário a mantém pressionada e é mostrada na segunda linha do visor do LCD a contagem do tempo precedido de um ícone “sino”. 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 compartilhando o mesmo barramento de dados? Vamos analisar mais cuidadosamente o esquemático em [1] e o diagrama de tempo do LCD na Seção 8. Observe no esquemático que o conjunto de 8 *leds* vermelhos 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 em [5] que o LCD só captura os

dados na borda de descida do sinal E que está conectado no pino 9 da porta C. Com isso, podemos controlar acessos aos 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). Estes *leds* são usados no programa [6] para sinalizar se as ações do usuário sobre as botoeiras PTA4 e PTA12 foram efetivamente reconhecidas pelo microcontrolador.

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]. Vale observar 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**.

E como controlamos a posição de um caractere no *display*? Veja ainda 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 [5]. Por exemplo, se quisermos exibir "AB" no meio da segunda linha do LCD, precisamos setar o endereço DDRAM enviando 0x86 (=0x80|0x060) com RS=0 e depois enviar o valor 0x41 (código ASCII de 'A') seguido de 0x42 (código ASCII de 'B') com RS=1. Vale observar que o endereço DDRAM é incrementado automaticamente após um acesso.

E para quê serve 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 da 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]. Eles variam de 39us a 1.53ms. Estes tempos são muito maiores do que o tempo de processamento de uma instrução do nosso microcontrolador cujo ciclo de relógio é da ordem de 50ns! Portanto, quando enviamos um comando/dado para o LCD, devemos aguardar um intervalo de tempo correspondente ao tempo de processamento daquele comando/dado antes de enviar o próximo comando para que eles não se "embaralhem".

Finalmente, observe que, em relação ao experimento 4 [1], adicionamos dois novos arquivos, `lcdled.c` e `lcdled.h`, para gerenciar a interface entre o nosso microcontrolador 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?* 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 4 [11]. Em seguida, adeque os códigos para que o nosso microcontrolador opere com o LCD conforme a funcionalidade planejada. Para isso, é necessário
   - *a)* substituir "xxxx" no arquivo `lcdled.c` para que (1) os registradores de controle da PORTC, (2) os registradores de controle do LCD sejam configurados apropriadamente e (3) os tempos de execução do nosso microcontrolador e do LCD fiquem compatíveis.
   - *b)* Observe que a rotina responsável pelos atrasos é de fato `delay20us` que é chamada dentro da rotina `pulso`. Por quê? O que os argumentos de entrada `p` e `t` da rotina `pulso` controlam?
   - *c)* completar a rotina `ConvF2String` no arquivo `util.c`, que faz a conversão de um valor em ponto flutuante j para uma sequência de caracteres ASCII correspondentes aos dígitos do número com uma casa decimal, a fim de que um valor em ponto flutuante seja exibível no visor do LCD. Tente elaborar o seu algoritmo antes de olhar para um possível procedimento de conversão:
     - i) se j for negativo, inserir o caractere ' -' no vetor `s`;
     - ii) extrair a parte inteira do valor absoluto de `j`;
     - iii) converter a parte inteira para ASCII;
     - iv) montar a sequencia de caracteres em ASC da parte inteira no vetor `s`;
     - v) extrair a parte fracionária do valor absoluto de `j`;
     - vi) se a parte fracionária for diferente de zeo, então
       - (1) deslocar `casas` dígitos da parte fracionária para casas inteira;
       - (2) converter o novo valor inteiro para ASCII;
       - (3) adicionar '.' no vetor `s`;
       - (4) adicionar a sequência de caracteres em ASCII do novo valor inteiro no vetor `s`.
   - *d)* completar a rotina `CriaBitmap(uint8_t end, uint8_t *ícone)` capaz de criar um novo *bitmap* no endereço `end=0bxxxxxx` da memória CGRAM do LCD, de forma que seja possível exibir *bitmaps* personalizados no visor do LCD. Observe em [5] que o formato de codificação do endereço de CGRAM é 0b01xxxxxx. Isso não equivaleria (0x40|0bxxxxxx)?
   - *e)* substituir o comando de comparação teoricamente válido, "`if (multiplo == 0.0)`" no arquivo `main.c`, por uma outra forma de comparação válida na prática. Qual seria uma alternativa prática correta? Justifique.

f) completar a funcionalidade de cronômetro (modo =1) em que a contagem progressiva em cada 0.1s seja visualizada na segunda linha do visor do LCD, substituindo "Cron. Ligado/Desligado". Tente elaborar o seu algoritmo antes de olhar para um possível procedimento de cronometragem:

se a botoeira PTA4 estiver pressionada, então
- (1) inicializa `contagem` em zero;
- (2) enquanto a botoeira PTA4 estiver pressionada
  - (a) após 0.1s de espera, incrementa `contagem` ;
  - (b) converter `contagem` em segundos, minutos e horas;
  - (c) chamar a rotina `ConvHHMMSS2String` para montar com segundos, minutos e horas uma sequência HH:MM:SS em ASCII.

g) completar a rotina `ConvHHMMSS2String (unsigned short hh, unsigned short mm, unsigned short ss, char *s)`, que gerar uma sequência de caracteres no formato HH:MM:SS onde HH, MM e SS correspondem a horas, minutos e segundos de um valor de tempo. Tente elaborar o seu algoritmo antes de olhar para um possível procedimento de conversão
- i) converter `hh` para ASCII: HH;
- ii) converter `mm` para ASCII: MM;
- iii) converter `ss` para ASCII: SS;
- iv) armazenar no vetor `s` a sequência de caracteres no formato HH:MM:SS

4. *Vamos ver se você entendeu?* Como no programa [6], o seu projeto do roteiro 5 deve agregar ao seu projeto do roteiro 4 [11] um visor do LCD onde é mostrado a cor selecionada na primeira linha do visor LCD (quando PTA5 é pressionada), a contagem regressiva, por décimo de segundo, do temporizador na segunda linha (quando a botoeira PTA12 é pressionada), e a contagem progressiva, por décimo de segundo, de um cronômetro (iniciar/parar com a botoeira PTA4). Observe que os dois modos temporizador e cronômetro são mutuamente exclusivos.

a) (50%) Relatório 5a: Submeta no sistema *Moodle* uma versão funcional do programa [6] (a-e do item 3) com o *bitmap* de sino substituído por um *bitmap* de temporizador e por um de cronômetro, respectivamente, no modo de temporizador e no modo de cronômetro. Limpe as pastas do seu projeto (*Project* > *Clean* …), exporte-o para um arquivo de extensão **zip** e suba o arquivo no sistema *Moodle*.

b) (50%) Relatório 5b*:* Estenda a funcionalidade do seu programa do roteiro 4 [11] conforme a especificação do projeto deste roteiro (temporizador+cronômetro) (f-g do item 3). Elabore o relatório sobre este projeto segundo o modelo, justificando as suas decisões de projeto que atendam 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 seu projeto exportado para um arquivo de extensão **zip** *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] *Rickey's World. LCD Tutorial for Interfacing with Microcontrollers.*
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/EA871/1s2018/T/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
[11] Wu Shin-Ting. EA871 – Roteiro 4 – 2s2017
http://www.dca.fee.unicamp.br/cursos/EA871/1s2018/T/roteiros/exp4.pdf

Agosto de 2016
Revisado em Fevereiro de 2017
Revisado em Julho de 2017
Revisado em Fevereiro de 2018