EA871 – LAB. DE PROGRAMAÇÃO BÁSICA DE SISTEMAS DIGITAIS

EXPERIMENTO 8 – 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?

Saber declarar as strings e usar as funções disponíveis na biblioteca de C

Saber setar simultaneamente um subconjunto de *bits* de um registrador

Saber gerar *por software* um pulso com uma largura pré-definida

Saber compatibilizar os tempos de execução de dois dispositivos de velocidade diferente

Programar LCD.

INTRODUÇÃO

Nos experimentos anteriores utilizamos somente sinais digitais e periféricos simples, os *leds* e as botoeiras. 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 funcional, elétrica, temporal e "mecânica" (pinagem) 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. Além dos módulos integrados no microcontrolador, o *shield* FEEC 871 [2] que se encontra devidamente conectado ao *kit* de desenvolvimento FRDM-KL25Z [3] dispõe de alguns periféricos bastante aplicados em projetos de sistemas embarcados. Para desenvolver um programa que controle esses periféricos, 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* FEEC 871 [2]. Nos experimentos anteriores, já vimos a programação do módulo GPIO para controlar três *leds* e três botoeiras do *shield*. Os valores dos pinos onde os *leds/botoeiras* estão

ligados eram escritos/lidos em separado. Vamos aprender agora como programar o módulo GPIO para uma transmissão paralela, ou seja, enviar vários *bits*, mais especificamente um *byte*, ao mesmo tempo para LCD ou para o conjunto de 8 *leds* vermelhos interfaceados por um *latch* 74573 [4], sob o controle de um sinal de habilitação. Este sinal de habilitação corresponde ao pino LE do *latch* e ao pino E do LCD.

Diferentemente do *latch*, o *byte* que é enviado ao LCD pode ser um comando ou um dado. A forma como o controlador do LCD decodifica o *byte* recebido depende do nível lógico da linha RS. Se RS=0, o *byte* é interpretado como um comando; caso contrário, é interpretado como um dado. Além disso, para cada *byte* recebido o LCD precisa de um intervalo de tempo, na faixa de us a ms, pra processá-lo [5] como veremos neste experimento. Os objetos renderizados no visor do LCD são *bitmaps* de 5 *pixels* x 8 *pixels*, disponíveis na memória do LCD. O endereço do *bitmap* de cada caractere alfa-numérico nessa memória é o seu próprio código ASCII. Com isso, para o LCD exibir uma *string* de caracteres, basta enviarmos para ele os elementos de uma *string* [6]. Veremos neste experimento o que são *strings* e como processá-las. Além dos *bitmaps* pré-definidos, o LCD utilizado nesta disciplina dispõe de uma memória, denominada de CGRAM, na qual se pode gravar até 8 *bitmaps* customizáveis. Em [7] dispõe-se de um editor de *bitmaps* 5x8.

EXPERIMENTO

Neste experimento vamos implementar o projeto **escolha_cor_bitmap** em que a botoeira IRQ5 tem a função de escolher uma cor dentre uma paleta de 8 cores (vermelho, verde, azul, amarelo, ciano, magenta, branco e preto) e a botoeira IRQ12 tem a função de escolher um *bitmap* dentre os 3 *bitmaps* customizados: cronômetro, relógio e termômetro. A cada acionamento da botoeira IRQ5 a cor varia sempre na mesma sequência e de forma cíclica até chegar a cor desejada. De forma análoga, escolhe-se o *bitmap*. Como realimentação visual de cada acionamento da botoeira IRQ5, além de mostrar a cor no *led* RGB, deve-se mostrar o nome da cor do *led* na segunda linha da tela do LCD (a partir do endereço 0x40 de DDRAM). Como realimentação visual de cada acionamento da botoeira IRQ12, deve-se mostrar o *bitmap* pelo qual está “passando” no canto superior direito da tela (endereço 0x0F de DDRAM).

Figura 1 apresenta uma proposta de fluxo de controle em que as mudanças de estado são detectadas na rotina de serviço PORTA_IRQHandler enquanto a efetivação destas mudanças, envolvendo processamento do LCD (dispositivo lento), no laço de espera do fluxo principal de controle.

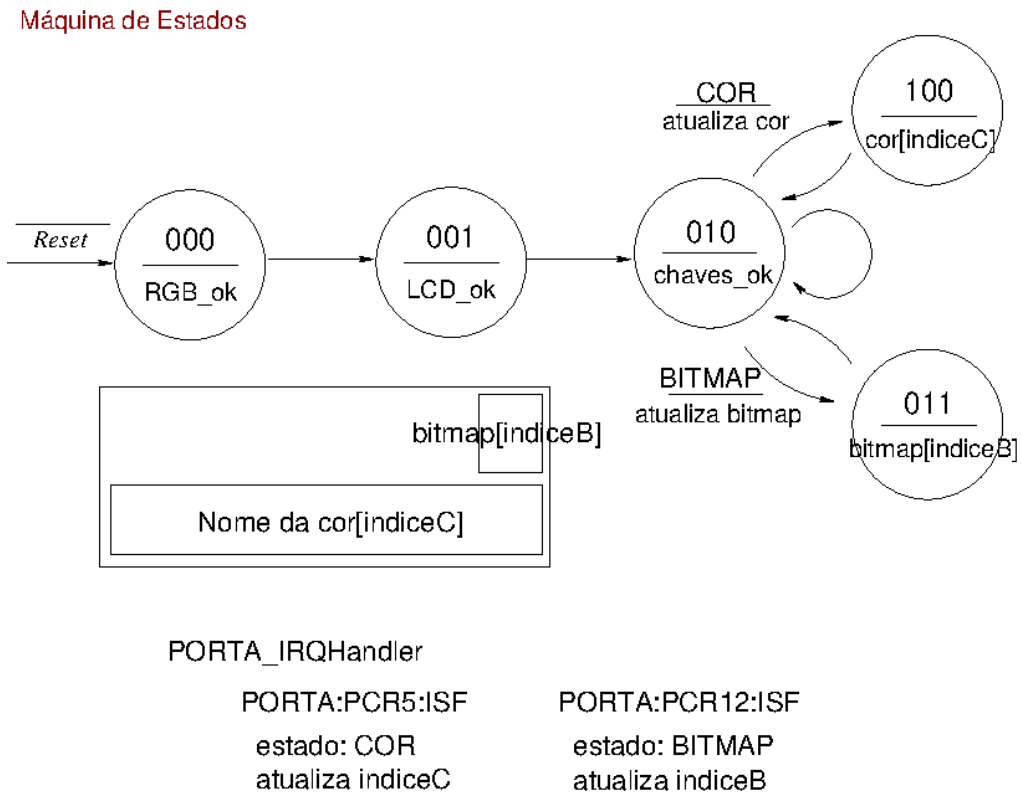

Figura. 1: Fluxo de controle proposto para o projeto **escolha_cor_bitmap**.

1. Qual técnica de programação você aplicaria para acessar sequencial e ciclicamente as cores de uma paleta de 8 cores e os *bitmaps* de um vetor de 3 *bitmaps*?
2. Sob o ponto de vista de programação das atualizações do conteúdo da tela do LCD como resposta aos acionamentos das 2 botoeiras, como você modelaria os possíveis estados de atualização da tela do LCD na rotina de serviço para que a rotina main faça as atualizações corretamente? Justifique.
3. Escreva um pseudocódigo da função main em que descreve o tratamento de cada estado de operação sobre o LCD para que o sistema tenha o comportamento especificado. Reuse as funções implementadas nos projetos-exemplo.
4. Para a sua solução ao projeto, quantas variáveis precisam ser compartilhadas entre as funções PORTA_IRQHandlere e main, ou seja, quantas variáveis globais precisam ser declaradas? Justifique.
5. Defina os 3 *bitmaps*, cronômetro, relógio e termômetro, que você projetou em valores hexadecimais por *byte*. Em qual endereço da memória CGRAM você gravaria cada *bitmap*? Por qual endereço você acessaria cada *bitmap*?
6. Implemente o aplicativo **escolha_cor_bitmap** em C por técnica de interrupção.
7. 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 **escolha_cor_bitmap.** 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] *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] *Datasheet* 74573
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/datasheet/74HC_HC573.pdf
[5] *Datasheet* do *display* LCD.
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea079/datasheet/AC162A.pdf
[6] *Rickey's World. LCD Tutorial for Interfacing with Microcontrollers.*
http://www.8051projects.net/lcd-interfacing/index.php
[7] *LCD Interfacing Tutorial: CGRAM Creating custom character*
https://www.8051projects.net/lcd-interfacing/lcd-custom-character.php

Outubro de 2020