EA871 – LAB. DE PROGRAMAÇÃO BÁSICA DE SISTEMAS DIGITAIS

EXPERIMENTO 1 – Introdução ao *Hardware* (*FRDM KL25* e *shield* EA871)

Profa. Wu Shin-Ting

OBJETIVO: Apresentação do *kit* de desenvolvimento FRDM-KL25 e do *shield* EA871

ASSUNTOS: Microcontrolador e periféricos integrados no kit FRDM-KL25. Revisão dos conceitos relacionados com o microcontrolador. Arquitetura ARM.

O que você deve ser capaz ao final deste experimento?

Recordar os conceitos vistos na disciplina EA869 ou equivalente.

Ter uma noção do *kit* FRDM-KL25 e dos principais manuais a serem utilizados em EA871.

Ter uma noção do *shield* EA871.

INTRODUÇÃO

Os conceitos da disciplina EA869 serão revistos e ampliados numa perspectiva prática. Para tanto, será utilizada uma plataforma de desenvolvimento denominada FRDM-KL25. Esta placa de desenvolvimento é fabricada pela *Freescale Semiconductors*, e permite desenvolver e depurar programas em várias linguagens, possuindo diversas conexões para circuitos digitais, em vários padrões de interface.

O "comandante" da placa é o microcontrolador MKL25Z128VLK4 de 32 *bits*, da série Kinetis L da *Freescale* [1]. Este dispositivo é um *System-on-a-Chip* (SoC), isto é, num *chip* são agregados um núcleo de processamento Cortex-M0+ de arquitetura *ARM*® e vários módulos, como temporizadores, interfaces de comunicação, conversores DA/AD, interrupções e unidades de armazenamento. Aliados baixo custo e baixo consumo de energia ao tamanho reduzido e à flexibilidade no desenho de um novo projeto, ele constitui uma alternativa para desenvolver aplicativos portáteis e de alto desempenho.

São integrados na placa FRDM-KL25 um circuito de alimentação e de *clock*, um *touchpad* capacitivo, um acelerômetro, *leds* RGB [2]. Além disso, há na placa um adaptador serial e de depuração, *OpenSDA*, que permite que sejam transferidos programas de um computador-hospedeiro para a memória interna do microcontrolador-alvo e que os mesmos sejam depurados. Isso é feito com um *software* específico rodando no computador-hospedeiro, conectado à placa através de uma interface USB. É interessante observar que o circuito do *OpenSDA* é baseado num outro microcontrolador da família Kinetis K20 da *Freescale*, K20DX128VFM5.

EXPERIMENTO

1. *Vamos entender o hardware a ser utilizado nos experimentos?* **Leia** atentamente o documento [3] para conhecer alguns detalhes, sob o ponto de vista de programação, da placa de desenvolvimento FRDM-KL25 e do *shield* EA871.

2. Podemos configurar, via *software*, o nosso microcontrolador para realizar uma tarefa específica. Vimos que, além de módulos dedicados, como temporizadores e interfaces de comunicação, ele possui um sistema complexo e configurável de geração de sinais de relógio no módulo MCG (*Multipurpose Clock Generator*) (veja Capítulo 24 de [1]), controlado pelo módulo SIM (*System Integration Module*) (veja Capítulo 12 de [1]), e de multiplexação dos sinais nos seus pinos físicos, PORT (*Port Control and Interrupts*) (veja Capítulo 11 de [1]). E todos os elementos de *hardware* são mapeados num mesmo espaço de endereços de memória de forma que possamos manipulá-los através dos seus endereços dentro de um programa (*software*). Estes endereços são usualmente conhecidos por endereços dos **registradores** de controle, de dados ou de estado. Ao inicializar/resetar o microcontrolador, todos os registradores são inicializados com valores especificados pelo fabricante. Tanto os endereços quanto os valores "Reset" acompanham a descrição de cada registrador em [1].

   Vamos ver como os conceitos são traduzidos na prática através de um programa [4]? O programa dado realiza a tarefa específica de piscar o *led* vermelho do *kit* FRDM-KL25Z Este *led* está ligado no pino 18 da porta B do nosso microcontrolador com o nível lógico ativo baixo (Figura 13 em [3]). Portanto, para piscá-lo, precisamos gerar alternadamente no pino 18 da porta B **sinais digitais** 0 e 1 de largura correspondente à metade do período correspondente à frequência desejada. Como configurar o micro-controlador para gerar este sinal?

   Vimos que a função de cada pino é controlado pelo módulo PORTx e o relógio de cada módulo PORTx controlado pelo módulo SIM. Os circuitos que estão por baixo são mapeados, respectivamente, nos endereços 0x4004A048 (Seção 11.5, p. 179 em [1]) e 0x40048038 (Seção 12.2.9, p. 206 em [1]). Neste caso específico, a função do pino é gerar sinal digital de propósito geral, portanto foi atribuído o valor 0b001 no campo MUX do endereço 0x4004A048 correspondente ao pino 18 da porta B (Seção 11.5.1, p. 183 em [1]). O valor 0b001 é o código binário da função GPIO (*General Purpose Input/Output*) que o pino desempenhará. Uma vez configurado como GPIO, as características específicas dos sinais digitais genéricas passam a ser controladas pelo módulo GPIO. Utilizando os circuitos deste último módulo podemos, por exemplo, setar a direção dos sinais, o nível do sinal no pino configurado como saída ou alternar o nível do sinal através dos endereços 0x400FF054, 0x400FF040 e 0x400FF04C, respectivamente (Seção 41.2, p. 774 em [1]). Observe que cada *bit* destes registradores controla um pino da porta B. Portanto, as operações de atribuição devem ser por *bit* utilizando os operadores lógicos & (*and*), | (*or*), << (deslocamento para esquerda) e >> (deslocamento para direita) da linguagem C [5]. Há, porém, três regitradores que alteram o valor binário dos *bits* somente quando o *bit* correspondente estiver em "1", GPIOx_PSOR (Seção 41.2.2, p. 776 em [1]), GPIOx_PCOR (Seção 41.2.3, p. 776 em [1]) e GPIOx_PTOR (Seção 41.2.4, p. 777 em [1]). Note ainda que no programa todos os endereços foram renomeados pelos nomes utilizados em [1] via `typedef` para tornar o programa mais inteligível. Finalmente, veja que todos os endereços são declarados com o qualificador `volatile` para evitar que o compilador otimize trechos de códigos contendo registradores cujo conteúdo seja passível de ser alterado por eventos externos ao processador.

3. *Vamos praticar mais o que acabamos de ver?* Quais alterações você faria no *programa* [4] para subistituir a cor vermelha pela cor verde do *led* RGB, para alternar o estado do *led* com uso de GPIOx_PTOR e para evitar uso desnecessário do qualificador `volatile`?

4. *Vamos ver se você entendeu?* Escreva um programa em C, bem documentado, que controla as piscadas simultâneas dos 3 *leds* R, G e B, de forma que a luz resultante seja branca, com três modificações:
   a) só usar qualificador *volatile* em casos estritamente necessários,
   b) os leds devem ser inicializados com o estado "apagado", e
   c) usar a função GPIOx_PTOR para alternar o estado dos *leds*.

RELATÓRIO

O relatório deste exeperimento deve constar a resposta do item 3 e o código em C do item 4 (não precisa carregá-lo nem executá-lo no microcontrolador). Suba o relatório no sistema Moodle.

REFERÊNCIAS

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

[1] KL25 Sub-Family Reference Manual – Freescale Semiconductors (doc. Number KL25P80M48SF0RM), Setembro 2012.
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KL25P80M48SF0RM.pdf
[2] FRDM-KL25Z User's Manual – Freescale Semiconductors, Setembro 2012.
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/FRDMKL25Z.pdf
[3] 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
[4] apostila.c
http://www.dca.fee.unicamp.br/cursos/EA871/1s2017/ST/codes/apostila.c
[5] Bitwise Operators in C and C++: A Tutorial
http://www.cprogramming.com/tutorial/bitwise_operators.html

Agosto de 2016
Revisado em Fevereiro de 2017