EA871 – LAB. DE PROGRAMAÇÃO BÁSICA DE SISTEMAS DIGITAIS

EXPERIMENTO 2 – Ferramentas de Desenvolvimento de *Software*

Profa. Wu Shin-Ting

OBJETIVO: Apresentação do ambiente de desenvolvimento de programas para MKL25Z128

ASSUNTOS: Construção de um projeto no ambiente *IDE CodeWarrior 10*: linguagem de programação C, pseudo-código, documentação e depuração de programas

O que você deve ser capaz ao final deste experimento?

Especificar a solução de um **problema bem definido** por um pseudocódigo.

Transcrever um pseudocódigo para um programa em C.

Depurar um código executando no nosso microcontrolador.

Documentar o seu código com uso de *Doxygen*.

Modularizar um projeto.

INTRODUÇÃO

Neste experimento vamos aprofundar o nosso entendimento do ambiente de desenvolvimento dos programas para controlar os periféricos integrados à placa FRDM-KL25 [1] que introduzimos no experimento [2]. Antes de escrever um programa, é recomendável organizar, no formato de um pseudocódigo [3], as ideias num fluxo de controle processável por uma máquina digital. Para torná-lo executável, é necessário utilizar um **editor** para escrever a sequência de ações definidas no pseudocódigo em instruções de uma linguagem de programação, usar um **compilador** para gerar a partir dos códigos-fonte códigos-objeto (linguagem de máquina), e um **ligador** para construir a partir dos códigos-objeto um código executável. E para executá-lo no microcontrolador, é necessário transferir via OpenSDA o código executável no formato elf para a memória FLASH do microcontrolador. Finalmente, para depurar e testar o programa no microcontrolador, um **depurador** serviria para controlar o fluxo de execução do programa, com paradas e passos de execução configuráveis, e monitorar/modificar as variáveis envolvidas.

Todas estas funções podem estar integradas num único ambiente de desenvolvimento, denominado IDE (*Integrated Development Environment*). Conforme vimos no primeiro experimento [2], usaremos nesta disciplina o *CodeWarrior* 10, um *IDE* baseado na plataforma *Eclipse*. No *IDE CodeWarrior* são disponíveis compiladores para duas linguagens de alto nível, C e C++. Optamos por C para desenvolvimento dos programas nesta disciplina [4].

EXPERIMENTO

1. *Vamos entender o ambiente de programação a ser utilizado nos experimentos?* **Leia** completamente o documento [5] para se familiarizar com os recursos disponíveis no *CodeWarrior*. Estes recursos irão assistí-lo a desenvolver os seus códigos ao longo desta disciplina. Por exemplo, todo o procedimento em *software* recomendado pelo fabricante para inicializar o micro-controlador antes de executar um código de aplicação propriamente dito

[10] é automaticamente gerado durante a criação de um projeto. Tais instruções se encontram no arquivo `Project_Settings/Startup_Code/_arm_start.c` e são ligados ao seu arquivo `main.c` para gerar um executável de extensão `.elf`.

2. *Vamos ver como os conceitos são traduzidos na prática através de um projeto constituído por um arquivo de programa?* Abra o projeto gerado com o programa apostila.c [6] na perspectiva de depuração (*Debug*) [5] [7].

a) Na perspectiva C/C++, você edita o seu código numa linguagem de alto nível. Nesta disciplina usamos a linguagem C. No experimento 1 foi dado o programa do projeto apostila.c [6] que é fazer o *led* vermelho do *kit* FRDM KL25Z piscar numa frequência em torno de 1Hz. Para se chegar ao programa foi necessário ponderar a solução almejada, os recursos disponíveis (como o *led* já está fisicamente conectado, avaliar por exemplo as funcionalidades e a programabilidade dos módulos que o pino em que o *led* vermelho está conectado serve) e os conhecimentos que dominamos (por exemplo, a arquitetura e a programabilidade dos micro-controladores). Nesta fase de concepção, deve-se focar na lógica, na estruturação ou no fluxo de controle da solução e não nos detalhes de sintaxe de uma linguagem de programação. Portanto, é comum fazer um esboço das nossas ideias e refiná-lo com uso de uma "linguagem" mais neutra possível – um **algoritmo**. Há várias formas de representar um algoritmo [9]. Nesta disciplina usaremos o **pseudo-código** [3]. Um pseudo-código re-estruturado do programa apostila.c [6] poderia ser:

```
subrotina delay
INICIO
Entrada: valor
Saída: nenhuma
Enquanto valor  é diferente de zero
        decrementa valor de 1
Fim enquanto
  FIM

programa main
  INÍCIO
  Entrada: nenhuma
  Saída: sinais digitais alternados nos pinos 18, 19 do GPIOB e o pino 1 do GPIOD
  Inicailização:
  Inicializa os registradores conforme os valores espacificados nas tabelas
  seta em 1 (3.3V) o bit 18 do registrador GPIOB
    Enquanto (verdadeiro)
        seta em 0 (0V) o bit 18 do registrador GPIOB
        chama delay com 500000
        seta em 1 (3.3V) o bit 18 do registrador GPIOB
        chama delay com 500000
      Fim enquanto
  FIM
```

Uma vez estruturada ou organizada as nossas ideias numa sequência de "regras", ou num procedimento, fica mais fácil traduzí-las para a linguagem de uma máquina digital e editar a nossa solução nesta linguagem.

b) Na perspectiva de depuração, confere os endereços que você atribuiu aos registradores dos módulos SIM, PORT e GPIO que você utilizou com os endereços em que estes registradores estão mapeados através da aba *Registers*.

c) Compare os valores nos registradores utilizados (aba *Registers*) com os valores setados na memória (aba *Memory*)

d) Coloque um *breakpoint* na linha "while (i) i -- ;" monitore pala aba *Variables* como o conteúdo da variável "i" é modificada cada vez que você clica em "*Resume*". O que acontece se você editar o valor "i" no campo "Value" antes de clicar em "*Resume*"?

e) Veja o conteúdo dos 4 *bytes* a partir do endereço 0x00000004 pela aba *Memory*. O que são os dados destes 4 *bytes*?

f) Como você reinicia, com uso de "*Reset*" a execução do seu código a partir de *__thumb_startup*? E a partir da primeira instrução da sua rotina *main*? Veja na aba "*Disassembly*" os endereços onde as instruções são armazenadas.

g) Coloque um *breakpoint* na linha "`delay (50000);` ". Certifique na aba *Breakpoints* a quantidade de pontos de parada já setados. Ao párar nesta linha, avance para a próxima instrução com "*Step into*". E na iteração seguinte, avance com "*Step over*". Qual é a diferença que você observou?

3. *Vamos ver se você consegue gerar uma documentação do seu projeto com uso de Doxygen? Siga os passos mostrados em* [5] e gere uma documentação em *html* do projeto gerado com o programa apostila.c [6].

4. *Vamos ver se você consegue criar um projeto importando-o de um arquivo existente?* O arquivo hello_world.zip [8] é um arquivo criado através da exportação de um projeto existente (Seção 2.6 em [5]). Importa-o segundo as instruções dadas na mesma Seção. Observe que hello_world.zip [8] é apenas uma reestruturação do programa apostila.c [4] em arquivos menores `atrasos.c`, `ledRGB.c` e `main.c`. O primeiro arquivo contém a função de "espera", o segundo as funções relacionadas com o periférico *led* RGB e o terceiro, o programa principal. Veremos nos próximos experimentos que isso não só reduz o tempo de desenvolvimento como facilita a manutenção dos códigos.

5. *Vamos ver se você entendeu?* Reestruture o programa que você elaborou no experimento 1 em arquivos menores conforme o projeto-modelo hello_world.zip [8]. **Complete as rotinas de manipulação dos três *leds* no arquivo `ledRGB.c`. e use-as no seu programa reestrutuado**. Veja na Seção 2.1.4 em [5] como se insere novos arquivos numa pasta pré-criada. Gere um projeto com os arquivos re-estruturados. Documente os seus códigos seguindo a sintaxe de *Doxygen* e gere uma documentação em *html*. Execute o projeto e responda: (1) Quais são os endereços e os conteúdos das variáveis de cada rotina (`atrasos.c`, `ledRGB.c`) antes de retornar ao programa main.c? (2) Mostre que é possível ver informações equivalente de conteúdos das variáveis pela aba *Memory*. (3) Para o estado "aceso" do *led* verde, qual é o conteúdo dos registradores do módulo GPIOD antes de retornar ao programa main.c? (4) Mostre que os dois pontos de parada, que você setou nas questões c e f do item 2, são registrados na aba

Breakpoints. (5) Qual é o espaço de memória em que a rotina `main.c` é carregado? (6) Qual é a função do botão com ícone "óculos" na Aba *Variables*?

RELATÓRIO

O relatório deve ser devidamente identificado e conter as respostas do item 5 através das capturas de tela do *IDE CodeWarrior*. Limpe a pasta do projeto do item 5 (*Project* > *Clean* …) (Seção 2.2.2 em [5]), exporta o projeto num arquivo comprimido no formato **zip** e suba-o no sistema *Moodle*.

REFERÊNCIAS

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

[1] *FRDM-KL25Z User's Manual – Freescale Semiconductors (doc. Number KL25P80M48SF0RM*, Setembro 2012*)*
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/FRDMKL25Z.pdf
[2] Wu, Shin-Ting. EA871 - Roteiro 1 – 2s2017.
http://www.dca.fee.unicamp.br/cursos/EA871/2s2017/EM/roteiros/exp1.pdf
[3] Expert.net. Fluxograma e Pseudocódigo.
http://www.tiexpert.net/programacao/algoritmo/fluxogramas-e-pseudocodigo.php
[4] Freescale CodeWarriorU. *Learn Programming with C*.
http://cache.freescale.com/files/training_presentation/TP_C_PROGRAMMING.pdf?lang_cd=en
[5] 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
[6] apostila.c
http://www.dca.fee.unicamp.br/cursos/EA871/1s2017/ST/codes/apostila.c
[7] Freescale. *CodeWarrior Development Suite: Eclipse Quick Reference Windows*.
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea076/complementos/CWMCUQRCARD.pdf
[8] hello_world.zip
http://www.dca.fee.unicamp.br/cursos/EA871/2s2017/EM/codes/hello_world.zip
[9] Seiji Isotani. Algoritmo e Pseudo-código.
https://edisciplinas.usp.br/pluginfile.php/118843/mod_resource/content/1/aula2.pdf
[10] Kinetis L Peripheral Module Quick Reference (Rev. 0.09/2012)
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KLQRUG.pdf

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