UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E COMPUTAÇÃO

17/03/2016

EA871 - LAB. DE PROGRAMAÇÃO BÁSICA DE SISTEMAS DIGITAIS

EXPERIMENTO 2 - Ferramentas de Desenvolvimento

Profª Wu Shin-Ting
Aluno: Tiago Eidi Hatta RA: 160388

Proposta do Experimento

Escrever um pseudo-código e depois um código-fonte de uma rotina que configura o Microcontrolador para controlar os leds verde e azul, na mesma frequência do vermelho, resultando em luz branca. Depois, modificar o programa para que os 3 leds pisquem em uma mesma frequência, mas que as cores se alterem a cada 5 piscadas na sequência: vermelho, branco, amarelo, ciano, magenta e preto.

Metodologia:

Para resolver os problemas, sabe-se que há a disponibilidade de três cores puras, de acordo com os leds vermelho, verde e azul. Por isso, deve-se acender somente as cores necessárias para a formação do resultado desejado. Por exemplo, a cor branca é a junção das três cores disponíveis.
Além disso, o módulo GPIO deve ser configurado corretamente de acordo com seus endereços [2]. A mudança dos níveis lógicos, que fazem os leds (1) acenderem ou (0) apagarem, podem ser feitos a partir de operações lógicas (AND e OR).
Ademais, é preciso utilizar uma função delay, com um laço for, entre as instruções, para que os fenômenos de acender ou apagar sejam visíveis a olho nú.

Proposta de Solução

Abaixo estão os pseudo-códigos das duas partes do experimento, bem como a explicação da solução baseada nos códigos fontes escritos em C.

Parte 1 – Controle de Leds resultando em luz branca

Pseudo-Código

```
INÍCIO: delay
        ENTRADA: número total de iterações
        ENQUANTO número total de iterações é maior que 0
                DECREMENTA número total de iterações
        FIM ENQUANTO
FIM: delay

INÍCIO
        CONFIGURE registrador de controle SIM_SCGC5[10] de forma a habilitar clock do módulo GPIO
        CONFIGURE registrador de controle PORTB_PCR18[8:10] de forma que o pino 18 da porta B
        sirva GPIO
        CONFIGURE registrador de controle PORTB_PCR19[8:10] de forma que o pino 19 da porta B
        sirva GPIO
        CONFIGURE registrador de controle PORTD_PCR1[8:10] de forma que o pino 1 da porta D sirva
        GPIO
```

CONFIGURE registrador de controle GPIOB_PDDR[18, 19] de forma que os pinos 18 e 19 da porta B sejam de saída
CONFIGURE registrador de controle GPIOD_PDDR[1] de forma que o pino 1 da porta D seja de saída

ENQUANTO VERDADEIRO
    Apague led vermelho
    Apague led verde
    Apague led azul
    Espera (CHAME delay com número total de iterações)
    Acende led vermelho
    Acende led verde
    Acende led azul
    Espera (CHAME delay com número total de iterações)
FIM ENQUANTO
FIM

Código Fonte

```
/*!
@file exp2.c
@authors Tiago Hatta
@date 17/03/2016
*/

#define SIM_SCGC5    (*(unsigned int volatile *) 0x40048038) /*! Habilita as Portas do GPIO (Reg. SIM_SCGC5) */
#define PORTB_PCR18 (*(unsigned int volatile *) 0x4004A048) /*!  MUX de PTB18 (Reg. PORTB_PCR18) */
#define PORTB_PCR19 (*(unsigned int volatile *) 0x4004A04C) /*!  MUX de PTB19 (Reg. PORTB_PCR19) */
#define PORTD_PCR1   (*(unsigned int volatile *) 0x4004C004) /*!  MUX de PTB19 (Reg. PORTB_PCR) */
#define GPIOB_PDDR   (*(unsigned int volatile *) 0x400FF054) /*! Data direction do PORTB (Reg. GPIOB_PDDR) */
#define GPIOB_PSOR   (*(unsigned int volatile *) 0x400FF044) /*! Set bit register do PORTB (Reg. GPIOB_PSOR) */
#define GPIOB_PCOR   (*(unsigned int volatile *) 0x400FF048) /*! Clear bit register do PORTB (Reg. GPIOB_PCOR) */
#define GPIOD_PDDR   (*(unsigned int volatile *) 0x400FF0D4) /*! Data direction do PORTD (Reg. GPIOD_PDDR) */
#define GPIOD_PSOR   (*(unsigned int volatile *) 0x400FF0C4) /*! Set bit register do PORTD (Reg. GPIOD_PSOR) */
#define GPIOD_PCOR   (*(unsigned int volatile *) 0x400FF0C8) /*! Clear bit register do PORTD (Reg. GPIOD_PCOR) */

/*
* @brief gera um atraso correspondente a I iteracoes
* @param [in] i numero de iteracoes
*/

void delay( unsigned int i)
{
   while (i) i--;
}
```

Inicialmente, usa-se #define para definir os mnemônicos para cada endereço dos registradores, sejam para as portas físicas ou para o módulo de GPIO. Isso facilita no manuseio do código ao distinguir os diferentes números hexadecimais de forma mais clara.
Para o led vermelho, a porta indicada por [1] é a PTB18, para a verde a PTB19 e para a azul é PTD1.

```
/*
* @brief Led piscante
*/

int main( void)
{
   SIM_SCGC5 = SIM_SCGC5 | (101<<10);    /*!  Habilita clock GPIO do PORTB e PORTD */
   PORTB_PCR18 = PORTB_PCR18 & 0xFFFFF8FF; /*! Zera bits 10, 9 e 8 (MUX) de PTB18 */
   PORTB_PCR18 = PORTB_PCR18 | 0x00000100; /*! Seta bit 8 do MUX de PTB18, assim os 3 bits de MUX serao 001 */

   PORTB_PCR19 = PORTB_PCR19 & 0xFFFFF8FF; /*! Zera bits 10, 9 e 8 (MUX) de PTB19 */
   PORTB_PCR19 = PORTB_PCR19 | 0x00000100; /*! Seta bit 8 do MUX de PTB19, assim os 3 bits de MUX serao 001 */

   PORTD_PCR1 = PORTD_PCR1 & 0xFFFFF8FF; /*! Zera bits 10, 9 e 8 (MUX) de PTD1 */
   PORTD_PCR1 = PORTD_PCR1 | 0x00000100; /*! Seta bit 8 do MUX de PTD1, assim os 3 bits de MUX serao 001 */

   GPIOB_PDDR  = GPIOB_PDDR  | (11<<18);    /*! Seta pinos 18 e 19 do PORTB como saida */
   GPIOD_PDDR  = GPIOD_PDDR  | 10;    /*! Seta pino 1 do PORTD como saida */

   for(;;)
   {


      GPIOB_PSOR = (11<<18); /*! Set bits 18 e 19, LEDs vermelho em PTB18 e verde em PTB19 (apagam) */
      GPIOD_PSOR = 10;       /*! Set bit 2, LED azul em PTD1 (apaga) */
      delay(500000);         /*! Espera um tempo */
      GPIOB_PCOR = (11<<18); /*! Clear bits 18 e 19, LED vermelho em PTB18 e verde em PTB19 (acendem) */
      GPIOD_PCOR = 10;       /*! Clear bit 2, LED vermelho em PTD1 (acende) */
      delay(500000);         /*! Espera um tempo */

   }
}
```

Em seguida, tem-se a função main. Nela, há a inicialização dos registradores de controle do microcontrolador. Primeiro, o clock pelo GPIO é habilitado nas portas B (led vermelho e verde) e porta D (led azul), através do módulo SIM (System Integration Module). Através do OU Lógico, os bits 12 e 10 foram setados, seguindo o Capítulo 12 de [2].
Depois, os registradores de cada porta são zerados (para evitar qualquer problema, com bits setados sem necessidade) e ocorre a multiplexação dos sinais nos pinos, para funcionamento correto do GPIO (valor 001 nos bits 8 a 10 – Cap.11 de [2]).
Além disso, para que o MCU possa trabalhar com o GPIO, é preciso definir quais as portas de saídas em GPIOx_PDDR (x no caso é B ou D), setando os bits desejados. Por causa dos leds, foram configurados bits 18 e 19 de B e 2 de D.

Após a inicialização dos registradores, dentro de uma rotina do tipo for (inifinito), primeiro são apagados os leds vermelho, verde e azul, cada um de acordo com o GPIO configurado para suas portas e em PSOR (Port Set Output Register). PSOR tem a função de setar determinados bits de PDOR (Port Data Output Register), responsável por configurar os níveis lógicos dos pinos de uso geral do MCU.
Após um tempo de delay, os leds são apagados, agora utilizando PCOR, que tem a função de zerar os bits de PDOR.

Parte 2 – Controle de Leds resultando em uma sequência vermelho, branco, amarelo, ciano, magenta e preto.

Pseudo-Código

INÍCIO: delay
    ENTRADA: número total de iterações
    ENQUANTO número total de iterações é maior que 0
        DECREMENTA número total de iterações
    FIM ENQUANTO
FIM: delay

INÍCIO

CONFIGURE registrador de controle SIM_SCGC5[10] de forma a habilitar clock do módulo GPIO
CONFIGURE registrador de controle PORTB_PCR18[8:10] de forma que o pino 18 da porta B sirva GPIO
CONFIGURE registrador de controle PORTB_PCR19[8:10] de forma que o pino 19 da porta B sirva GPIO
CONFIGURE registrador de controle PORTD_PCR1[8:10] de forma que o pino 1 da porta D sirva GPIO

CONFIGURE registrador de controle GPIOB_PDDR[18, 19] de forma que os pinos 18 e 19 da porta B sejam de saída
CONFIGURE registrador de controle GPIOD_PDDR[1] de forma que o pino 1 da porta D seja de saída

ENQUANTO VERDADEIRO
Acende led vermelho
Espera (CHAME delay com número total de iterações)
Apague led vermelho
Espera (CHAME delay com número total de iterações)

Acende led vermelho
Acende led verde
Acende led azul
Espera (CHAME delay com número total de iterações)
Apague led vermelho
Apague led verde
Apague led azul
Espera (CHAME delay com número total de iterações)

Acende led vermelho
Acende led verde
Espera (CHAME delay com número total de iterações)
Apague led vermelho
Apague led verde
Espera (CHAME delay com número total de iterações)

Acende led verde
Acende led azul
Espera (CHAME delay com número total de iterações)
Apague led verde
Apague led azul
Espera (CHAME delay com número total de iterações)

Acende led vermelho
Acende led azul
Espera (CHAME delay com número total de iterações)
Apague led vermelho
Apague led azul
Espera (CHAME delay com número total de iterações)

Espera (CHAME delay com número total de iterações)
FIM ENQUANTO

FIM

Código Fonte

```c
/*!
@file sequenciadecores.c
@authors Tiago Hatta
@date 17/03/2016
*/

#define SIM_SCGC5   (*(unsigned int volatile *) 0x40048038) /*! Habilita as Portas do GPIO (Reg. SIM_SCGC5) */
#define PORTB_PCR18 (*(unsigned int volatile *) 0x4004A048) /*!  MUX de PTB18 (Reg. PORTB_PCR18) */
#define PORTB_PCR19 (*(unsigned int volatile *) 0x4004A04C) /*!  MUX de PTB19 (Reg. PORTB_PCR19) */
#define PORTD_PCR1  (*(unsigned int volatile *) 0x4004C004) /*!  MUX de PTB19 (Reg. PORTB_PCR) */
#define GPIOB_PDDR  (*(unsigned int volatile *) 0x400FF054) /*! Data direction do PORTB (Reg. GPIOB_PDDR) */
#define GPIOB_PSOR  (*(unsigned int volatile *) 0x400FF044) /*! Set bit register do PORTB (Reg. GPIOB_PSOR) */
#define GPIOB_PCOR  (*(unsigned int volatile *) 0x400FF048) /*! Clear bit register do PORTB (Reg. GPIOB_PCOR) */
#define GPIOD_PDDR  (*(unsigned int volatile *) 0x400FF0D4) /*! Data direction do PORTD (Reg. GPIOD_PDDR) */
#define GPIOD_PSOR  (*(unsigned int volatile *) 0x400FF0C4) /*! Set bit register do PORTD (Reg. GPIOD_PSOR) */
#define GPIOD_PCOR  (*(unsigned int volatile *) 0x400FF0C8) /*! Clear bit register do PORTD (Reg. GPIOD_PCOR) */

/*
* @brief gera um atraso correspondente a I iteracoes
* @param [in] i numero de iteracoes
*/

void delay( unsigned int i)
{
    while (i) i--;
}
```

O início do código é igual ao da parte 1, com utilização dos mnemônicos dos endereços e a função delay.

```c
/*
* @brief acende leds
* @param [in] vermelho char tal que v acende, f nao acende
* @param [in] verde char tal que v acende, f nao acende
* @param [in] azul char tal que v acende, f nao acende
*/
void acendeLeds(char vermelho, char verde, char azul){
    if(vermelho == 'v' && verde == 'v'){
        GPIOB_PCOR = (11<<18); /*! Clear bit 18 e 19, LED vermelho em PTB18 e verde em PTB19 (acende) */
    }else{
        if(vermelho == 'v')
            GPIOB_PCOR = (1<<18); /*! Clear bit 18, LED vermelho em PTB18 (acende) */
        if(verde == 'v')
            GPIOB_PCOR = (1<<19); /*! Clear bit 19, LED vermelho em PTB19 (acende) */

    }

    if(azul == 'v')
                GPIOD_PCOR = 10; /*! Clear bit 2, LED azul em PTD1 (acende) */
    delay(500000);        /*! Espera um tempo */

    return;
}
```

Como a sequência de cores exige que várias vezes os leds sejam acesos, é mais prático e claro utilizar a função acendeLeds. Ela recebe como parâmetro um caractere para cada um dos leds, que determina qual ou quais devem ser acesos. Utilizou-se como padronização, que um char ‘v’ (de verdadeiro), faça com que o determinado led possa gerar luz. Se deseja-se ligar os leds vermelho e verde, pode-se fazer isso diretamente nos bits 18 e 19 de uma vez. Além disso, a função também já chama a função delay, já que é necessária sempre que modifica-se o estado dos leds.

```
/*
* @brief apaga leds
* @param [in] vermelho char tal que v apaga
* @param [in] verde char tal que v apaga
* @param [in] azul char tal que v apaga
*/
void apagaLeds(char vermelho, char verde, char azul){
    if(vermelho == 'v' && verde == 'v'){
            GPIOB_PSOR = (11<<18); /*! Clear bit 18 e 19, LED vermelho em PTB18 e verde em PTB19 (apaga) */
    }else{
        if(vermelho == 'v')
            GPIOB_PSOR = (1<<18); /*! Clear bit 18, LED vermelho em PTB18 (apaga) */
        if(verde == 'v')
            GPIOB_PSOR = (1<<19); /*! Clear bit 19, LED vermelho em PTB19 (apaga) */

    }
    if(azul == 'v')
                GPIOD_PSOR = 10; /*! Clear bit 2, LED azul em PTD1 (apaga) */

    delay(500000);        /*! Espera um tempo */
}
```

Assim como a função acendeLeds, também tem-se a apagaLeds, que se comporta de forma semelhante, recebendo três parâmetros que indicam quais leds devem ser apagados.

```
int main( void)
{
    SIM_SCGC5 = SIM_SCGC5 | (101<<10);    /*!  Habilita clock GPIO do PORTB e PORTD */

    PORTB_PCR18 = PORTB_PCR18 & 0xFFFFF8FF; /*! Zera bits 10, 9 e 8 (MUX) de PTB18 */
    PORTB_PCR18 = PORTB_PCR18 | 0x00000100; /*! Seta bit 8 do MUX de PTB18, assim os 3 bits de MUX serao 001 */

    PORTB_PCR19 = PORTB_PCR19 & 0xFFFFF8FF; /*! Zera bits 10, 9 e 8 (MUX) de PTB19 */
    PORTB_PCR19 = PORTB_PCR19 | 0x00000100; /*! Seta bit 8 do MUX de PTB19, assim os 3 bits de MUX serao 001 */

    PORTD_PCR1 = PORTD_PCR1 & 0xFFFFF8FF; /*! Zera bits 10, 9 e 8 (MUX) de PTD1 */
    PORTD_PCR1 = PORTD_PCR1 | 0x00000100; /*! Seta bit 8 do MUX de PTD1, assim os 3 bits de MUX serao 001 */

    GPIOB_PDDR  = GPIOB_PDDR  | (11<<18);    /*! Seta pinos 18 e 19 do PORTB como saida */
    GPIOD_PDDR  = GPIOD_PDDR  | 10;    /*! Seta pino 1 do PORTD como saida */

    GPIOB_PSOR = (11<<18); /*! Set bit 18, LED vermelho em PTB18 (apaga) */
    GPIOD_PSOR = (10); /*! Set bit 18, LED vermelho em PTB18 (apaga) */
```

O começo da função main é igual ao explicado na parte 1, com a inicialização dos registradores utilizados por cada módulo do MCU.

```
    int i = 0;
    for(;;)
    {
        for(i = 0; i < 5; i++){
            /*! cor vermelha */
            acendeLeds('v', 'f', 'f');      /*! Liga apenas led vermelho */
            apagaLeds('v', 'f', 'f');       /*! Apaga o led vermelho */
        }

        for(i = 0; i < 5; i++){
            /*! branco */
            acendeLeds('v', 'v', 'v');      /*! Acende todos os leds para cor branca */
            apagaLeds('v', 'v', 'v');       /*! Apaga todos os leds */
        }

        for(i = 0; i < 5; i++){
            /*! amarelo */
            acendeLeds('v', 'v', 'f');      /*! Acende vermelho e verde que formam cor amarela */
            apagaLeds('v', 'v', 'f');       /*! Apaga os leds ligados */
        }
```

```
        for(i = 0; i < 5; i++){
            /*! ciano */
            acendeLeds('f', 'v', 'v');      /*! Acende leds verde e azul para formar a cor ciano */
            apagaLeds('f', 'v', 'v');       /*! Apaga os leds */
        }

        for(i = 0; i < 5; i++){
            /*! magenta */
            acendeLeds('v', 'f', 'v');      /*! Acende os leds vermelho e azul para o magenta */
            apagaLeds('v', 'f', 'v');       /*! Apaga os leds vermelho e azul */
        }

        for(i = 0; i < 5; i++){
            /*! preto */
            delay(500000);        /*! Como o preto sao todos leds desligados, chama a funcao delay */
        }

    }
}
```

Em seguida, entra-se em uma rotina for, e dentro dela, há a sequência de cores, cada uma com 5 piscadas para cada cor. Os leds são acesos e depois apagados segundo o resultado desejado.

Testes

Parte 1

Para a primeira parte do experimento, foram feitas as seguintes etapas:

1) Teste do programa repassado no site da disciplina, para acender o led vermelho;
2) Após ler os manuais do MCU e configurar as portas corretas, foi realizado o teste dos leds vermelho e verde ao mesmo tempo. Verifica-se que o resultado é uma cor amarela, comprovando o resultado buscado;
3) De forma análoga, porém considerando a porta D, testou-se acender os três leds, que resultaram em uma cor branca, ou seja, obtendo sucesso ao alcançar o primeiro objetivo do experimento.

Parte 2

1) Utilizando como base o código utilizado na parte 1, são chamadas as funções de acender leds de acordo com as cores pretendidas. Inicialmente, testa-se as cores vermelhas, amarela e branca, alternada por 1 piscada;
2) Após pesquisar a formação das cores ciano e magenta, testa-se as combinações de cores, seguindo a sequência desejada com apenas 1 piscada de cada cor, mas na mesma frequência dada pelo código do roteiro;
3) Com o sucesso das duas primeiras etapas, muda-se o código, colocando as funções de acender e apagar leds para cada cor, dentro de um *for* com inteiro i variando de 0 a 5, para que haja 5 piscadas antes de serem alternadas as cores.

Conclusão

O experimento é simples do ponto de vista que é apenas para introduzir o uso do MCU para acender os leds da placa. No entanto, foi um desafio saber identificar os módulos e como usá-los corretamente, pois foi necessário relembrar os conceitos da disciplina MC404 (Organização Básica de Computadores e Linguagem de Montagem) ministrada pelo Instituto de Computação.

Além disso, o uso dos manuais também não foi intuitivo, mas após entender o código obtido pelo site da disciplina, e obtendo êxito ao acender a cor branca, as etapas seguintes foram apenas de ajustar o código. Entre os testes, também foi feito o debug na primeira parte, através do code warrior, necessário para analisar o comportamento do circuito não apenas na linguagem C, como nos registradores (por assembly).

Possíveis melhorias poderiam ser feitas no código, ao deixá-lo mais enxuto com uma função de inicialização dos registradores de controle do MCU. Por não terem sido feitos testes com isso, o código foi mantido como o explicado anteriormente.

Referências

[1] FRDM-KL25Z User's Manual.

ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/FRDMKL25Z.pdf

[2] KL25 Sub-Family Reference Manual

ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KL25P80M48SF0RM.pdf