UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Universidade Federal do Ceará - Campus Quixadá

Prática 3 - TIMERS
QXD0143 - Microcontroladores

Prof. Thiago Werlley
3 de junho de 2022

Aluno: Samuel Henrique - **Matrícula:** 473360
Aluno: Antônio César - **Matrícula:** 473444

Quixadá-CE

Sumário

1 Introdução

Timers são de vital importância para microcontroladores desempenhando diversas funções como PWM, interrupção, tempo-real, entre outras. Isso poderá ser notado nessa e em futuras práticas que usam algumas dessas aplicações.

No Cortex-M0 exitem diversos módulos de *timer*, mas essa prática focou na paredizagem de apenas 3, *Low-Power Timer* (LPTMR), *Periodic Interrupt Timer* (PIT) e *Timer/PWM Module* (TPM).

2 *Low-Power Timer* (LPTMR)

O *timer* LPTMR trata-se de um módulo de baixo potência em comparação a outros presentes no microcontrolador.

Os recursos do módulo LPTMR incluem:

- Contador de tempo ou contador de pulsos com comparador de 16 bits
- Clock configurável a partir do *prescaler*
- Fonte de entrada configurável para contador de pulsos
  - Borda de subida ou borda de subida

O registrador *prescaler* (LPTMRx_PSR) apresenta no campo dos bits 0-1 quatro opções de *clocks* que podem ser utilizadas. No bit 2 o *bypass* pode ser ativado ou não. Quando esse bit é setado, então o *prescaler* será ignorado (bypassed), não haverá divisão na onda de clock de entrada. No campo dos bits 3-6 o valor de *presclae* é determinado, começando por 2 e indo até o valor de 8192 (valor mais alto), esse valor escolhido pelo programado irá dividir a fonte de clock selecionada nos bits 0-1.

O registrador *compare* (LPTMRx_CMR) apresenta 16 bits disponíveis para a armazenar o valor de comparação, ou seja, valor final que o *timer* irá contar.

O registrador de *status* de controle (LPTMRx_CSR) apresenta alguns bits importantes que devem ser modificados ou não de acordo com sua aplicação. Nessa aplicação é preciso habilitar o bit 0 para o *timer* LPTMR seja ativado. O bit 1 determina qual modo será selecionado. O bit 2 como o registrador CNR será resetado ao ser detectada uma interrupção na comparação do timer (TCF). Os bits 3-5 trata-se do contador de pulso que não será utlizado nessa aplicação. O bit 6 habilita a interrupção ou não de *timer*. O bit 7 trata-se da *Timer Compare Flag* (TCF) esse bit é setado quando os registradores CNR e CMR são iguais.

O registrador contador (LPTMRx_CNR) é usado para fazer a contagem de 16 bits, ou seja, começa em 0 e vai incrementando até chegar no valor guardado em *CMR* e a interrupção seja disparada

Existem algumas fontes de clocks dentro do microcontrolador, MCGIRCLK, LPO, ERCLK32K, OSCERCLK são as fontes que o registrador PSR permite selecionar. No caso dessa prática o clock LPO será utilizado. LPO (*Low Power Output Oscillator*) trata-se de um clock de baixa potência gerado pelo PMC (*Power Management Controller*), sua frequência é de 1kHz.

2.1 Item 1

2.1.1 Explicação e registradores

No item 1 da prática é pedido para que acenda e a apague um LED a cada 1 segundo sem a utilização de *prescale*. Para fazer o que se pede é necessário ativar o bit de *bypass* para que o p*prescale* seja ignorado e depois realizar um cálculo simples para descobrir o valor de CMR para 1s.

Quando o *bypass* é ativado, admitimos que o valor de *prescale* é 1. Para isso, montamos uma equação que nos ajude a encontrar o valor a ser escrito no registrador CMR. Vale lembrar que o registrador CNR começa a contagem a partir do valor 0.

$$CMR = \frac{T \cdot f}{prescale} - 1 \tag{1}$$

Onde, f: frequência de clock escolhida, T: tempo desejado, *prescale*: valor escolhido para divisão do clock

É pedido no item para que o tempo de *blink* do LED seja de 1s, ou seja, $T = 1$, a frequência de clock escolhida foi o LPO de $1kHz$, então $f = 1kHz$ e não foi pedido *prescale*, então admitimos $prescale = 1$.

$$CMR = \frac{1 \cdot 1000}{1} - 1 \Rightarrow CMR = 999$$

Então, para que a contagem seja feita dentro de 1 segundo, é preciso escrever o valor 999 no registrador CMR.

Após descobrir o valor de comparação é necessário ativar os outros bits em outros registradores. Começando pelo registrador SCGC5, é preciso habilitar o bit 0 que corresponde ao acesso do módulo LPTMR, ao colocar 1 nesse bit significa que podemos fazer uso do seu módulo, além disso, é preciso ativar o bit 12 para habilitar o clock do PORTD que é o periférico do LED que será aceso (Linha 86). O próximo passo é definir multiplexação do pino correspondente ao LED como GPIO, através do PCR no bit 8 (Linha 95). Definida a multiplexação, é necessário dar direção ao pino, ele será um LED então é saída, basta ativar o bit do pino no registrador PDDR do módulo GPIOD (Linha 98).

Agora acessando os registradores do LPTMR, no registrador *prescale* (PSR) é necessário ativar o bit 0 para selecionar a fonte de clock 1, que corresponde ao LPO de 1kHz. Além disso, o bit 2 também precisa ser setado para ativar o *bypass*, ou seja, fazer com que o *prescaler* seja ignorado (Linha 107). Após a configuração de PSR, é necessário alterar o valor de comparação, para isso basta atribuir o valor de 999 encontrado na equação acima ao registrador *compare* (CMR) do código (Linha 111). Agora alte-

rando o registrador de *status* de controle (CSR), é necessário setar o bit 0 em nível lógico alto para habilitar o LPTMR, além disso, também é necessário setar o bit 6 para habilitar a interrupção de *Timer* no módulo (Linha 122). Está feita nossa configuração de LPTMR.

O próximo passo é acessar o registrador do NVIC, o ISER (*Interrupt Set Enable Registers*) esse registrador fará com que seja a habilitada a interrupção de LPTMR0_IRQn, ou seja, habilita a rotina de interrupção (Linha 125). Na rotina de interrupção apenas duas ações são realizadas, o *toggle* do LED que foi configurado (Linha 62) e a flag de interrupção de timer que é setada no bit 7 (w1c) do registrador de status de controle (CSR) após o estado do LED ser alterado, chamando a interrupção de timer (Linha 73).

2.1.2 Código

```
1  /**
2   * @file    LPTMR_FRDM-KL43Z.c
3   * @brief   Application entry point.
4   */
5  #include <stdio.h>
6  #include "board.h"
7  #include "peripherals.h"
8  #include "pin_mux.h"
9  #include "clock_config.h"
10 #include "MKL43Z4.h"
11 #include "fsl_debug_console.h"
12
13 /* TODO: insert other include files here. */
14
15 /* TODO: insert other definitions and declarations here. */
16
17 typedef struct{
18   uint32_t PCR[32];
19 }PORTRegs_t;
20
```

```
21 #define PORT_B ((PORTRegs_t *)0x4004A000)
22 #define PORT_D ((PORTRegs_t *)0x4004C000)
23
24 typedef struct{
25   uint32_t PDOR;
26   uint32_t PSOR;
27   uint32_t PCOR;
28   uint32_t PTOR;
29   uint32_t PDIR;
30   uint32_t PDDR;
31 }GPIORegs_t;
32
33 #define GPIO_B ((GPIORegs_t *)0x400FF040)
34 #define GPIO_D ((GPIORegs_t *)0x400FF0C0)
35
36 typedef struct {
37   uint32_t iser[1];
38   uint32_t rsvd[31];
39   uint32_t icer[1];
40   uint32_t rsvd1[31];
41   uint32_t ispr[1];
42   uint32_t rsvd2[31];
43   uint32_t icpr[1];
44   uint32_t rsvd3[31];
45   uint32_t rsvd4[64];
46   uint32_t ipr[8];
47 }NVIC_Regs_t;
48
49 #define NVIC_REG ((NVIC_Regs_t *)0xE000E100)
50
51 typedef struct{
52   uint32_t tcsr;
53   uint32_t prescale;
54   uint32_t compare;
55   uint32_t count;
56 }LPTMR_Regs_t;
57
```

```
58 #define LPTMR_REG ((LPTMR_Regs_t *) 0x40040000)
59
60 void LPTMR0_IRQHandler(void){
61   //Alternar pino 5 do led red para acender e apagar
62   GPIO_D->PTOR = (1 << 5);
63
64   // Timer Compare Flag
65   //0 The value of CNR is not equal to CMR and increments --
66   //1 The value of CNR is equal to CMR and increments.
67
68 //    *  Low Power Timer Control Status Register (LPTMRx_CSR)
69 //    *| 31  --  8 |  7  |  6  | 5  4 |  3  |  2  |  1  |  0
      |
70 //    *|     0    | ISF | TIE |  TPS | TPP | TFC | TMS | TEN |
71 //    *|            | w1c |
72
73   LPTMR_REG->tcsr |= (1 << 7);
74 }
75
76
77 // * @brief   Application entry point.
78
79 int main(void) {
80   //1 - Ativar o clock para LPTMR e para porta B
81
82 //    *  System Clock Gating Control Register 5 (SIM_SCGC5)
83 //    *| 31  --  20| 19 | 18 -- 14 | 13 | 12 | 11 | 10 |  9 |
      8  7 | 6 |   5 | 4  3  2 | 1 |   0   |
84 //    *|     0    |  0 |     0    | PE | PD | PC | PB | PA |
      1  | 0 | TSI |    0    | 0 | LPTMR |
85
86   SIM->SCGC5 = (1 << 0) | (1 <<12);
87
88   //2 - Port_D_5 como GPIO
89
90 //    *  Pin Control Register n (PORTx_PCRn)
91 //    *| 31  --  25|  24 | 23 -- 20 | 19 -- 16 | 15 -- 11 | 10
```

```
      9   8 | 7 |  6   | 5 |  4   | 3 |  2   | 1  |  0 |
//    *|     0    | ISF |     0     |    IRQC    |     0     |
    MUX    | 0 | DSE | 0 | PFE | 0 | SRE | PE | PS |      | 0 |
     LPTMR |
//    *|           | w1c |      |

  PORT_D->PCR[5] = (1 << 8);

  //GPIO_D_5 como sa da
  GPIO_D->PDDR = (1 << 5);


  //Bypass    ativado e clock do LPTMR    LPO

//    *  Low Power Timer Prescale Register (LPTMRx_PSR)
//    *| 31  --  7 |  6  5  4  3  |   2  | 1   0 |
//    *|     0    |   PRESCALE    | PBYP |  PCS  |

  LPTMR_REG->prescale = (1 << 2) | (1 << 0);

  //LPO    1kHz -> T = 1/f = 1/1000 = 1ms. Queremos o tempo
    de 1s -> compare register = 1/1ms = 1000s
  //CMR = 1/(1/(1000/1)) = 1000
  LPTMR_REG->compare = 999;


  //Ativar Timer Enable -- 0 LPTMR is disabled and internal
    logic is reset -- 1 LPTMR is enabled
  //Timer Interrupt Enable -- 0 Timer interrupt disabled -- 1
    Timer interrupt enabled

//    *  Low Power Timer Control Status Register (LPTMRx_CSR)
//    *| 31  --  8 |  7  |  6  | 5  4 |  3  |  2  |  1  |  0
    |
//    *|     0    | ISF | TIE |  TPS | TPP | TFC | TMS | TEN |
//    *|           | w1c |
```

```
122   LPTMR_REG->tcsr = (1 << 0) | (1 << 6);
123
124   //NVIC_EnableIRQ(LPTMR0_IRQn);
125   NVIC_REG->iser[0] = (1 << 28);
126
127   while(1){
128   }
129
130     return 0 ;
131 }
```

2.2 Item 2

2.2.1 Explicação e registradores

O segundo item pede que o o mesmo LED acenda e apague por 8 segundos, mas agora uma coisa muda, é necessário usar o valor de *prescale* de 64, ou seja, é preciso dividir a onda da fonte de clock selecionada por 64.

Para realizar com excelência o que é pedido, poucas coisas precisam ser alteradas. A únicas alterações são nos registradores PSR e CMR.

As alterações são as seguintes, agora será necessário o uso de *prescale*, para isso é preciso colocar o bit 2 em nível lógico baixo, isso fará com que o *bypass* seja desativado e nos bit 3-6 é preciso passar o valor 5 em binário (0b0101), ou seja, bit 3 e bit 5 ativados para selecionar o valor de *prescale* de 64 (Linha 109). Agora, o valor que o registrador comparador recebe muda um pouco devido a presença do valor de *prescale*.

De acordo com a Equação 1:

$$CMR = \frac{8 \cdot 1000}{64} - 1 \Rightarrow CMR = 124$$

Agora basta fazer a atribuição ao registrador *compare* (Linha 115). Após a realização dessas mudanças, o LED ficará aceso por 8s e apagado por 8s, todo o resto permanece o mesmo.

2.2.2 Código

```
/**
 * @file    LPTMR_2_FRDM-KL43Z.c
 * @brief   Application entry point.
 */
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL43Z4.h"
#include "fsl_debug_console.h"
/* TODO: insert other include files here. */

/* TODO: insert other definitions and declarations here. */

typedef struct{
  uint32_t PCR[32];
}PORTRegs_t;

#define PORT_B ((PORTRegs_t *)0x4004A000)
#define PORT_D ((PORTRegs_t *)0x4004C000)

typedef struct{
  uint32_t PDOR;
  uint32_t PSOR;
  uint32_t PCOR;
  uint32_t PTOR;
  uint32_t PDIR;
  uint32_t PDDR;
}GPIORegs_t;

#define GPIO_B ((GPIORegs_t *)0x400FF040)
#define GPIO_D ((GPIORegs_t *)0x400FF0C0)

typedef struct {
```

```
36   uint32_t iser[1];
37   uint32_t rsvd[31];
38   uint32_t icer[1];
39   uint32_t rsvd1[31];
40   uint32_t ispr[1];
41   uint32_t rsvd2[31];
42   uint32_t icpr[1];
43   uint32_t rsvd3[31];
44   uint32_t rsvd4[64];
45   uint32_t ipr[8];
46 }NVIC_Regs_t;
47
48 #define NVIC_REG ((NVIC_Regs_t *)0xE000E100)
49
50 typedef struct{
51   uint32_t tcsr;
52   uint32_t prescale;
53   uint32_t compare;
54   uint32_t count;
55 }LPTMR_Regs_t;
56
57 #define LPTMR_REG ((LPTMR_Regs_t *) 0x40040000)
58
59 void LPTMR0_IRQHandler(void){
60   //Alternar pino 18 do led red para acender e apagar
61   GPIO_D->PTOR = (1 << 5);
62
63   // Timer Compare Flag
64   //0 The value of CNR is not equal to CMR and increments --
65   //1 The value of CNR is equal to CMR and increments.
66   /*
67    *  Low Power Timer Control Status Register (LPTMRx_CSR)
68    *| 31  --  8 |  7  |  6  | 5  4 |  3  |  2  |  1  |  0  |
69    *|     0    | ISF | TIE |  TPS | TPP | TFC | TMS | TEN |
70    *|           | w1c |
71    */
72   LPTMR_REG->tcsr |= (1 << 7);
```

```
73 }
74
75 /*
76  * @brief   Application entry point.
77  */
78
79 //1A - Acender led sem preescale (1s)
80 int main(void) {
81   //1 - Ativar o clock para LPTMR e para porta B
82   /*
83    *  System Clock Gating Control Register 5 (SIM_SCGC5)
84    *| 31  --  20| 19 | 18 -- 14 | 13 | 12 | 11 | 10 |  9 | 8
       7 | 6 |   5 | 4  3  2 | 1 |   0   |
85    *|     0    |  0 |     0    | PE | PD | PC | PB | PA |   1
       | 0 | TSI |    0    | 0 | LPTMR |
86    */
87   SIM->SCGC5 = (1 << 0) | (1 << 12);
88
89   //2 - Port_D_5 como GPIO
90   /*
91    *  Pin Control Register n (PORTx_PCRn)
92    *| 31  --  25|  24 | 23 -- 20 | 19 -- 16 | 15 -- 11 | 10
       9  8 | 7 |  6  | 5 |  4  | 3 |  2  | 1  |  0 |
93    *|     0    | ISF |     0    |   IRQC   |     0    |    MUX
          | 0 | DSE | 0 | PFE | 0 | SRE | PE | PS |      | 0 |
       LPTMR |
94    *|           | w1c |      |
95    */
96   PORT_D->PCR[5] = (1 << 8);
97
98   //GPIO_D_5 como sa da
99   GPIO_D->PDDR = (1 << 5);
100
101
102   //Bypass    ativado e clock do LPTMR    LPO
103   /*
104    *  Low Power Timer Prescale Register (LPTMRx_PSR)
```

```
105     *| 31  --  7 |  6  5  4  3  |   2  | 1   0 |
106     *|     0    |   PRESCALE    | PBYP |  PCS  |
107     */
108    //Habilitando o prescaler --> colocando o bit em nivel
        logico baixo
109    LPTMR_REG->prescale &= ~(1 << 2);
110    //Selecionando a fonte | Selecionando o valor de prescale
111    LPTMR_REG->prescale = (1 << 0) | (0b0101 << 3);
112
113    //LPO    1kHz -> T = 1/f = 1/1000 = 1ms. Queremos o tempo
        de 1s -> compare register = 1/1ms = 1000s
114    //CMR = 8/(1/(1000/64)) = 125
115    LPTMR_REG->compare = 124;
116
117
118    //Ativar Timer Enable -- 0 LPTMR is disabled and internal
        logic is reset -- 1 LPTMR is enabled
119    //Timer Interrupt Enable -- 0 Timer interrupt disabled -- 1
         Timer interrupt enabled
120    /*
121     *  Low Power Timer Control Status Register (LPTMRx_CSR)
122     *| 31  --  8 |  7  |  6  | 5  4 |  3  |  2  |  1  |  0  |
123     *|     0    | ISF | TIE |  TPS | TPP | TFC | TMS | TEN |
124     *|           | w1c |
125     */
126    LPTMR_REG->tcsr = (1 << 0) | (1 << 6);
127
128    //NVIC_EnableIRQ(LPTMR0_IRQn);
129    NVIC_REG->iser[0] = (1 << 28);
130
131    while(1){
132    }
133
134      return 0 ;
135 }
```

2.3 Item 3

2.3.1 Explicação e registradores

No item 3, é solicitado a utilização de dois LEDs, um para ficar aceso durante 2s e apagado durante 2s e outro para ficar aceso durante 6s e apagado durante 6s. Para fazer isso acontecer, algumas informações devem ser alteradas. As alterações que serão feitas na rotina de interrupção LPTMR0_IRQn e no valor do registrador de comparação. Também foram adicionadas duas funções: toggleLedGreen() e toggleLedRed(), que configuram o LEDs e fazem eles trocarem de estado.

Nesse item, temos dois tempos para temporizar, um de 2s e outro de 6s, então iremos utilizar o valor de 2s como valor base, por esse motivo o registrador CMR receberá um valor correspondendo ao tempo de 2s. Para isso, usa-se a Equação 1.

$$CMR = \frac{2 \cdot 1000}{64} - 1 \Rightarrow CMR = 30,25 = 30$$

Na rotina de interrupção há uma pequena alteração na lógica, dessa vez é utilizado um contador global fazendo uma lógica com o LED verde que piscará por 2s. Basicamente, o LED verde começa apagado por 2s e ao entrar na rotina troca de estado e o contador incrementa, é avisada a interrupção, passam-se 2s, o estado do LED é trocado novamente, o contador incrementa, passam-se 2s e o contador já é igual a 2, então o código entra no *else* o qual altera o sinal do LED vermelho que permanecerá o mesmo até entrar no *else* novamente, ou seja, após 6s.

2.3.2 Código

```
1 /**
2  * @file    LPTimer.c
3  * @brief   Application entry point.
4  */
```

```
5  #include <stdio.h>
6  #include "board.h"
7  #include "peripherals.h"
8  #include "pin_mux.h"
9  #include "clock_config.h"
10 #include "MKL43Z4.h"
11 #include "fsl_debug_console.h"
12 /* TODO: insert other include files here. */
13
14 /* TODO: insert other definitions and declarations here. */
15 #define PIN_LED_GREEN 5
16
17 typedef struct{
18   uint32_t PCR[32];
19 }PORTRegs_t;
20
21 #define PORT_D ((PORTRegs_t *)0x4004C000)
22 #define PORT_E ((PORTRegs_t *)0x4004D000)
23
24 typedef struct{
25   uint32_t PDOR;
26   uint32_t PSOR;
27   uint32_t PCOR;
28   uint32_t PTOR;
29   uint32_t PDIR;
30   uint32_t PDDR;
31 }GPIORegs_t;
32
33 #define GPIO_D ((GPIORegs_t *)0x400FF0C0)
34 #define GPIO_E ((GPIORegs_t *)0x400FF100)
35
36 typedef struct {
37   uint32_t iser[1];
38   uint32_t rsvd[31];
39   uint32_t icer[1];
40   uint32_t rsvd1[31];
41   uint32_t ispr[1];
```

```
42   uint32_t rsvd2[31];
43   uint32_t icpr[1];
44   uint32_t rsvd3[31];
45   uint32_t rsvd4[64];
46   uint32_t ipr[8];
47 }NVIC_Regs_t;
48
49 #define NVIC_REG ((NVIC_Regs_t *)0xE000E100)
50
51 typedef struct{
52   uint32_t CSR;
53   uint32_t PSR;
54   uint32_t CMP;
55   uint32_t CNR;
56 }LPTMR_Regs_t;
57
58 #define LPTMR_REG ((LPTMR_Regs_t *) 0x40040000)
59
60 int counter = 1;
61
62 void toggleLedGreen(){
63   PORT_D->PCR[5] |= (1 << 8);
64   GPIO_D->PDDR |= (1 << 5);
65   GPIO_D->PTOR |= (1 << 5);
66 }
67
68 void toggleLedRed(){
69   PORT_E->PCR[31] |= (1 << 8);
70   GPIO_E->PDDR |= (1 << 31);
71   GPIO_E->PTOR |= (1 << 31);
72 }
73
74 void LPTMR0_IRQHandler(void){
75
76   // Deve ligar a cada 2s
77   toggleLedGreen();
78   if(counter < 2){
```

```
79      counter++;
80    }else {
81    // Deve ligar a cada 6s
82      toggleLedRed();
83      counter = 0;
84    }
85
86    // Timer Compare Flag
87    // 0 The value of CNR is not equal to CMR and increments.
88    // 1 The value of CNR is equal to CMR and increments.
89
90    LPTMR_REG->CSR = LPTMR_REG->CSR | (1 << 7);
91  }
92
93  /*
94   * @brief   Application entry point.
95   */
96  int main(void) {
97    //1 - Ativar o clock para LPTMR e para porta D e E/
98    SIM->SCGC5 = (1 << 0) | (1 << 13) | (1 << 12);
99
100   //pescaler    ativado e clock do LPTMR    LPO
101   LPTMR_REG->PSR &= ~(1 << 2);
102   LPTMR_REG->PSR |= (1 << 0);
103
104   //TFC -- Timer Free-Running Counter
105   //0 -- CNR is reset whenever TCF is set.
106   //1 -- CNR is reset on overflow.
107   LPTMR_REG->CSR = (1 << 2);
108
109   //PRESCALER -- LPO/64
110   //PCS -- LPO
111   LPTMR_REG->PSR = (1 << 0) | (0b0101 << 3);
112
113
114   // Com prescaler
115   //LPO    1kHz -> T = 1/f = 1/(1000/64) = 64ms.
```

```
116   ///Queremos o tempo de 2s -> CMP = 2s/64ms = 31,25 --> 30
117   LPTMR_REG->CMP = 30;
118
119   //TEN -- 0 LPTMR is disabled and internal logic is reset --
        1 LPTMR is enabled
120   //TIE -- Timer Interrupt Enable -- 0 Timer interrupt
       disabled -- 1 Timer interrupt enabled
121   LPTMR_REG->CSR = (1 << 0) | (1 << 6);
122
123   //NVIC_EnableIRQ(LPTMR0_IRQn);
124   NVIC_REG->iser[0] = (1 << 28);
125
126   while(1){
127   }
128
129     return 0 ;
130 }
```

2.4 Item 4

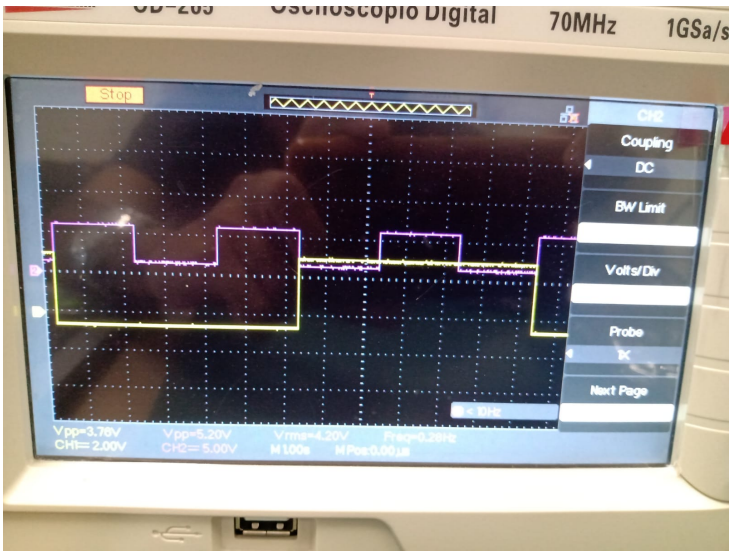

Figura 1: Resultado do código do item 3

No canal de cor rosa, é possível ver o comportamento do LED que tem *timer* de 2 segundos e no canal amarelo é possível ver o comportamento do LED de 6 segundos. Cada quadrado do osciloscópio corresponde a 1 segundo.

2.5 Item 5.a)

2.5.1 Funções

O item 4 da primeira questão pede que os itens 1 e 2 sejam implementados através do SDK, uma abstração feita para o desenvolvimento do software. Não é necessário mexer diretamente no registrador, com SDK, funções prontas são utilizadas.

Funções utilizadas:

- CLOCK_EnableClock() - habilita o clock no perisférico passado no argumento.
- PORT_SetPinMux() - realiza a multiplexação do pino do periférico como a função desejada.
- GPIO_PinInit() - realiza a inicialização do módulo GPIO. Recebe o GPIO, o pino e o ponteiro da configuração do LED.
- LPTMR_GetDefaultConfig() - retorna a configuração padrão do LPTMR para o ponteiro da estrutura de configuração do LPTMR.
- LPTMR_Init() - realiza a inicialização do LPTMR passado como argumento com o ponteiro da estrutura configuração passada.
- LPTMR_SetTimerPeriod() - faz a atribuição do valor a ser comparado pelo registrador.
- LPTMR_EnableInterrupts() - habilita a interrupção no módulo e o tipo de interrupção passada.

- NVIC_EnableIRQ() - habilita a rotina de interrupção.
- GPIO_PortToggle() - faz um *toggle* no pino do módulo GPIO que é passado.
- LPTMR_ClearStatusFlags() - escreve no bit 7 w1c limpando o *statos* de interrupção.

Estruturas utilizadas:

- lptmr_config_t - apresenta configurações de polaridade, se o contador vai resetar na interrupção ou *overflow*, o valor do *prescale* etc.
- gpio_pin_config_t - configurações de GPIO, a direção do pino e a saída lógica

Na linha 48, a variável de estrutura **lptmrCfg** recebe os valores *default* de configuração, na linha 50 o campo de *bypass* é acessado e modificado para 1, fazendo com que o *prescaler* seja ignorado, na linha 51 é configurado o valor de *prescale* que é 1, nesse caso.

Na linha 33 a estrutura **gpioLed** é modificada, selecionando o pino como saída e o sinal de saída 1.

2.5.2 Códigos

```
1  /**
2   * @file    LPTMR_SDK.c
3   * @brief   Application entry point.
4   */
5  #include <stdio.h>
6  #include "board.h"
7  #include "peripherals.h"
8  #include "pin_mux.h"
9  #include "clock_config.h"
10 #include "MKL43Z4.h"
11 #include "fsl_debug_console.h"
```

```
12 #include "fsl_gpio.h"
13 #include "fsl_port.h"
14 #include "fsl_lptmr.h"
15
16 /* TODO: insert other include files here. */
17
18 /* TODO: insert other definitions and declarations here. */
19
20 void LPTMR0_IRQHandler(void) {
21
22   GPIO_PortToggle(GPIOE, (1 << 31));
23   LPTMR_ClearStatusFlags(LPTMR0, kLPTMR_TimerCompareFlag);
24 }
25
26
27 /*
28  * @brief   Application entry point.
29  */
30 int main(void) {
31   lptmr_config_t lptmrCfg = {0};
32
33   gpio_pin_config_t gpioLed = {
34         kGPIO_DigitalOutput,
35         1
36   };
37
38   // Ativando o clock da PORTA_E
39   CLOCK_EnableClock(kCLOCK_PortE);
40
41   // Ativando PORTA_E_31 como GPIO
42   PORT_SetPinMux(PORTE, 31, kPORT_MuxAsGpio);
43
44   //Configurando GPIO_E_31 como Output
45   GPIO_PinInit(GPIOE, 31, &gpioLed);
46
47   //Obtendo a configura  o padr o do LPTMR
48   LPTMR_GetDefaultConfig(&lptmrCfg);
```

```
49
50    lptmrCfg.bypassPrescaler = true;
51    lptmrCfg.value = 1;
52
53    //Inicializando o LPTMR
54    LPTMR_Init(LPTMR0, &lptmrCfg);
55
56    //Registrador CMR LPTMR para 1000 em 1s Timer
57    //CMR =1/(1/(1000/1)) = 1000
58    LPTMR_SetTimerPeriod(LPTMR0, 1000);
59
60    // Interru  o da LPTMR
61    LPTMR_EnableInterrupts(LPTMR0, kLPTMR_TimerInterruptEnable)
       ;
62
63    //ativando a interru  o para o LPTMR
64    NVIC_EnableIRQ(LPTMR0_IRQn);
65
66    //iniciando o Timer
67    LPTMR_StartTimer(LPTMR0);
68
69    while(1){}
70
71      return 0 ;
72 }
```

2.6 Item 5.b)

2.6.1 Funções

O segundo subitem pede que seja utilizado o SDK para fazer o LED acender e apagar por 8s e com *prescaler* ativado. Em comparação com o código de subitem acima, poucas coisas mudarão. As funções do SDK utilizadas anteriormente serão novamente utilizadas agora.

As mudanças estão nas linhas 50, 51, 52 e 55. O parâmetro **bypassPrescaler** recebe valor *false*, o parârametro **prescalerClockSource** recebe uma

constante referente a fonte de clock 1 (a mesma escolhida no registrador) e por último, o campo *value* recebe o valor 5, que corresponde ao valor de *prescale* de 64. Na linha 55 o valor de CMR é alterado pois agora há a presença do *prescale*, então o valor recebido agora é 124, valor explorado em seções acima.

2.6.2 Códigos

```
1  /**
2   * @file    LPTMR_SDK.c
3   * @brief   Application entry point.
4   */
5  #include <stdio.h>
6  #include "board.h"
7  #include "peripherals.h"
8  #include "pin_mux.h"
9  #include "clock_config.h"
10 #include "MKL43Z4.h"
11 #include "fsl_debug_console.h"
12 #include "fsl_gpio.h"
13 #include "fsl_port.h"
14 #include "fsl_lptmr.h"
15
16 /* TODO: insert other include files here. */
17
18 /* TODO: insert other definitions and declarations here. */
19
20 void LPTMR0_IRQHandler(void) {
21
22   GPIO_PortToggle(GPIOE, (1 << 31));
23   LPTMR_ClearStatusFlags(LPTMR0, kLPTMR_TimerCompareFlag);
24 }
25
26
27 /*
28  * @brief   Application entry point.
```

```
29  */
30 int main(void) {
31   lptmr_config_t lptmrCfg = {0};
32 
33   gpio_pin_config_t gpioLed = {
34         kGPIO_DigitalOutput,
35         1
36   };
37 
38   // Ativando o clock da PORTA_E
39   CLOCK_EnableClock(kCLOCK_PortE);
40 
41   // Ativando PORTA_E_31 como GPIO
42   PORT_SetPinMux(PORTE, 31, kPORT_MuxAsGpio);
43 
44   //Configurando GPIO_E_31 como Output
45   GPIO_PinInit(GPIOE, 31, &gpioLed);
46 
47   //Obtendo a configura  o padr o do LPTMR
48   LPTMR_GetDefaultConfig(&lptmrCfg);
49 
50   lptmrCfg.bypassPrescaler = false;
51   lptmrCfg.prescalerClockSource = kLPTMR_PrescalerClock_1;
52   lptmrCfg.value = 5;
53 
54   //Inicializando o LPTMR
55   LPTMR_Init(LPTMR0, &lptmrCfg);
56 
57   //CMR = 8/(1/(1000/64)) = 125
58   LPTMR_SetTimerPeriod(LPTMR0, 124);
59 
60   // Interru  o da LPTMR
61   LPTMR_EnableInterrupts(LPTMR0, kLPTMR_TimerInterruptEnable)
      ;
62 
63   //ativando a interru  o para o LPTMR
64   NVIC_EnableIRQ(LPTMR0_IRQn);
```

```
65
66    //iniciando o Timer
67    LPTMR_StartTimer(LPTMR0);
68
69    while(1){
70
71    }
72
73      return 0 ;
74 }
```

3 Periodic Interrupt Timer (PIT)

O *timer* PIT é usado para gerar interrupções em intervalos periódicos. Os recursos do módulo incluem:

- Dois canais (0 e 1)
- Contador de 32 bits
- Capacidade de temporizar e gerar disparo para o DMA
- Capacidade de temporizar e gerar interrupções
- Temporizador independente para cada canal

Esse tipo de *timer* não possui a opção de prescaler.

O registrador *Module Control Register* (MCR) diz se será utilizado o modo de depuração de PIT. Para ativar a depuração o bit 31 é setado como 0, caso seja setado como 1 esse modo é desativado.

O registrador *Lifetime Timer Register* (LTM) guarda o valor dos canais do PIT, por esse motivo é dividido em 2 LTM64H para o canal 1 e LTM64L para o canal 0.

O registrador *Timer Load Value Register* (LDVAL) guarda o valor inicial do PIT, no caso, o tempo desejado.

O registrador *Current Timer Value Register* (CVAL) guarda o valor atual do PIT.

O registrador *Timer Control Register* (TCTRL) é usado para habilitar o timer e habilitar interrupção. Quando o bit 0 estiver setado com 1 o timer estará habilitado e quando o bit 30 estiver setado com 1 interrupções estarão habilitadas.

O registrador *Timer Flag Register* (TFLAG) notifica a interrupção do PIT. Quando o bit 0 estiver setado como 1, então ocorreu interrupção.

Pra cada um dos canais existe um conjunto dos 4 últimos registradores citados.

3.1 Item 1

3.1.1 Explicação e Registradores

No item 1 da prática é pedido para que se acenda e apague um LED a cada 1 segundo sem a utilização de prescale. Como o PIT não possui a a opção de prescale, então não será precisa fazer alguma configuração.

Foi utilizado o Bus Clock que possui 24MHz e para o cálculo de tempo foi feita uma macro que faz o produto entre a frequência do clock e o valor passado como parâmetro na macro (tempo esperado em segundos).

Para que o PIT seja a opção escolhida com timer, o bit 23 de SCGC6 foi setado em 1. Para ativar o modo de depuração, o MCR é setado como 0. O LDVAL recebe o retorno da macro citada anteriormente e os bits 0 e 1 de TCTRL são setados em 1 para que tanto o timer quanto interrupção sejam habilitados.

No controlador de interrupção, para que o LED fique alternando entre HIGH e LOW foi necessário setar seu registrador de toggle em 1 e seta o bit 0 de TFLAG em 1 para que ocorra interrupção.

3.1.2 Código

```
1
2  /**
3   * @file    PIT_FRDM-KL43Z.c
4   * @brief   Application entry point.
5   */
6  #include <stdio.h>
7  #include "board.h"
8  #include "peripherals.h"
9  #include "pin_mux.h"
10 #include "clock_config.h"
11 #include "MKL43Z4.h"
12 #include "fsl_debug_console.h"
13 /* TODO: insert other include files here. */
14
15 /* TODO: insert other definitions and declarations here. */
16 typedef struct {
17   uint32_t MCR;
18   uint32_t LTM64H;
19   uint32_t LTM64L;
20 }PIT_Regs_t;
21
22 #define PIT_REG ((PIT_Regs_t*) 0x40037000)
23
24 typedef struct {
25   uint32_t LDVAL;
26   uint32_t CVAL;
27   uint32_t TCTRL;
28   uint32_t TFLAG;
29 }PIT_Chnl_Regs_t;
30
31 #define PIT_CH_REG ((PIT_Chnl_Regs_t*) 0x40037100)
32
33 typedef struct {
34   uint32_t PCR[32];
35 }PORTRegs_t;
36
37 #define PORT_B ((PORTRegs_t*) 0x4004A000)
```

```
38 #define PORT_D ((PORTRegs_t*) 0x4004C000)
39 #define PORT_E ((PORTRegs_t*) 0x4004D000)
40
41 typedef struct {
42   uint32_t PDOR;
43   uint32_t PSOR;
44   uint32_t PCOR;
45   uint32_t PTOR;
46   uint32_t PDIR;
47   uint32_t PDDR;
48 }GPIORegs_t;
49
50 #define GPIO_B ((GPIORegs_t*) 0x400FF040)
51 #define GPIO_D ((GPIORegs_t*) 0x400FF0C0)
52 #define GPIO_E ((GPIORegs_t*) 0x400FF100)
53
54
55 typedef struct {
56   uint32_t ISER[1];
57   uint32_t RSVD[31];
58   uint32_t ICER[1];
59   uint32_t RSVD1[31];
60   uint32_t ISPR[1];
61   uint32_t RSVD2[31];
62   uint32_t ICPR[1];
63   uint32_t RSVD3[31];
64   uint32_t RSVD4[64];
65   uint32_t IPR[1];
66 }NVIC_Regs_t;
67
68 #define NVIC_REG ((NVIC_Regs_t*) 0xE000E100)
69
70 #define GET_SEC_COUNT(x) (CLOCK_GetBusClkFreq() * x)
71
72 void PIT_IRQHandler(void) {
73
74   GPIO_E->PTOR |= (1 << 31);
```

```
75   PIT_CH_REG->TFLAG |= (1 << 0);
76 }
77 /*
78  * @brief   Application entry point.
79  */
80 int main(void) {
81   uint32_t clock = CLOCK_GetBusClkFreq();
82 
83   SIM->SCGC5 |= (1 << 13);
84 
85   PORT_E->PCR[31] |= (1 << 8);
86   GPIO_E->PDDR |= (1 << 31);
87 
88   SIM->SCGC6 |= (1 << 23);
89   PIT_REG->MCR = 0;
90   PIT_CH_REG->LDVAL = GET_SEC_COUNT(1);
91 
92   PIT_CH_REG->TCTRL = (1 << 1) | (1 << 0);
93 
94   NVIC_REG->ISER[0] |= (1 << 22);
95 
96   while(1){
97     printf("Clock: %d\n", clock);
98   }
99 
100    return 0 ;
101 }
```

3.2 Item 2

3.2.1 Explicação e Registradores

Como o timer PIT não possui prescale, então o código do item 1 ainda é válido, será necessário mudar apenas o valor passado como parâmetro na macro (8 para 8 segundos).

3.2.2 Código

```
1
2  /**
3   * @file    PIT_FRDM-KL43Z.c
4   * @brief   Application entry point.
5   */
6  #include <stdio.h>
7  #include "board.h"
8  #include "peripherals.h"
9  #include "pin_mux.h"
10 #include "clock_config.h"
11 #include "MKL43Z4.h"
12 #include "fsl_debug_console.h"
13 /* TODO: insert other include files here. */
14
15 /* TODO: insert other definitions and declarations here. */
16 typedef struct {
17   uint32_t MCR;
18   uint32_t LTM64H;
19   uint32_t LTM64L;
20 }PIT_Regs_t;
21
22 #define PIT_REG ((PIT_Regs_t*) 0x40037000)
23
24 typedef struct {
25   uint32_t LDVAL;
26   uint32_t CVAL;
27   uint32_t TCTRL;
28   uint32_t TFLAG;
29 }PIT_Chnl_Regs_t;
30
31 #define PIT_CH_REG ((PIT_Chnl_Regs_t*) 0x40037100)
32
33 typedef struct {
34   uint32_t PCR[32];
35 }PORTRegs_t;
```

```
36
37 #define PORT_B ((PORTRegs_t*) 0x4004A000)
38 #define PORT_D ((PORTRegs_t*) 0x4004C000)
39 #define PORT_E ((PORTRegs_t*) 0x4004D000)
40
41 typedef struct {
42   uint32_t PDOR;
43   uint32_t PSOR;
44   uint32_t PCOR;
45   uint32_t PTOR;
46   uint32_t PDIR;
47   uint32_t PDDR;
48 }GPIORegs_t;
49
50 #define GPIO_B ((GPIORegs_t*) 0x400FF040)
51 #define GPIO_D ((GPIORegs_t*) 0x400FF0C0)
52 #define GPIO_E ((GPIORegs_t*) 0x400FF100)
53
54
55 typedef struct {
56   uint32_t ISER[1];
57   uint32_t RSVD[31];
58   uint32_t ICER[1];
59   uint32_t RSVD1[31];
60   uint32_t ISPR[1];
61   uint32_t RSVD2[31];
62   uint32_t ICPR[1];
63   uint32_t RSVD3[31];
64   uint32_t RSVD4[64];
65   uint32_t IPR[1];
66 }NVIC_Regs_t;
67
68 #define NVIC_REG ((NVIC_Regs_t*) 0xE000E100)
69
70 #define GET_SEC_COUNT(x) (CLOCK_GetBusClkFreq() * x)
71
72 void PIT_IRQHandler(void) {
```

```
73
74    GPIO_E->PTOR |= (1 << 31);
75    PIT_CH_REG->TFLAG |= (1 << 0);
76  }
77  /*
78   * @brief   Application entry point.
79   */
80  int main(void) {
81    uint32_t clock = CLOCK_GetBusClkFreq();
82
83    SIM->SCGC5 |= (1 << 13);
84
85    PORT_E->PCR[31] |= (1 << 8);
86    GPIO_E->PDDR |= (1 << 31);
87
88    SIM->SCGC6 |= (1 << 23);
89    PIT_REG->MCR = 0;
90    PIT_CH_REG->LDVAL = GET_SEC_COUNT(8);
91
92    PIT_CH_REG->TCTRL = (1 << 1) | (1 << 0);
93
94    NVIC_REG->ISER[0] |= (1 << 22);
95
96    while(1){
97      printf("Clock: %d\n", clock);
98    }
99
100     return 0 ;
101 }
```

3.3 Item 3

3.3.1 Explicação e Registradores

O item 3 para que um LED acenda a cada 2 segundos e apague no intervalo de mesmo tempo e que aconteça o mesmo com outro LED no tempo de 6 segundos.

Foi utilizado o Bus Clock que possui 24MHz e para o cálculo de tempo foi feita uma macro que faz o produto entre a frequência do clock e o valor passado como parâmetro na macro (tempo esperado em segundos).

Como nesse item é pedido o controle de 2 LEDs, então é preciso utilizar os 2 canais do PIT, para isso foi preciso definir o endereço de cada canal, *0x40037100* para o canal 0 e *0x40037110* para o canal 1. Cada canal tem seus próprios registradores de configuração (LDVAL, TCTRL E TFLAG), então a configuração deles é igual a feita nos itens anteriores especificando cada canal.

No controlador de interrupção é testado o TFLAG de qual canal está setado e depois esse registrador é limpo.

3.3.2 Código

```
1
2  /**
3   * @file    PIT_FRDM-KL43Z.c
4   * @brief   Application entry point.
5   */
6  #include <stdio.h>
7  #include "board.h"
8  #include "peripherals.h"
9  #include "pin_mux.h"
10 #include "clock_config.h"
11 #include "MKL43Z4.h"
12 #include "fsl_debug_console.h"
13
14
15
16 /* TODO: insert other include files here. */
17
18 /* TODO: insert other definitions and declarations here. */
19 typedef struct {
20   uint32_t MCR;
```

```
21   uint32_t LTM64H;
22   uint32_t LTM64L;
23 }PIT_Regs_t;
24 
25 #define PIT_REG ((PIT_Regs_t*) 0x40037000)
26 
27 typedef struct {
28   uint32_t LDVAL;
29   uint32_t CVAL;
30   uint32_t TCTRL;
31   uint32_t TFLAG;
32 }PIT_Chnl_Regs_t;
33 
34 #define PIT_CH_0_REG ((PIT_Chnl_Regs_t*) 0x40037100)
35 #define PIT_CH_1_REG ((PIT_Chnl_Regs_t*) 0x40037110)
36 
37 typedef struct {
38   uint32_t PCR[32];
39 }PORTRegs_t;
40 
41 #define PORT_B ((PORTRegs_t*) 0x4004A000)
42 #define PORT_D ((PORTRegs_t*) 0x4004C000)
43 #define PORT_E ((PORTRegs_t*) 0x4004D000)
44 
45 typedef struct {
46   uint32_t PDOR;
47   uint32_t PSOR;
48   uint32_t PCOR;
49   uint32_t PTOR;
50   uint32_t PDIR;
51   uint32_t PDDR;
52 }GPIORegs_t;
53 
54 #define GPIO_B ((GPIORegs_t*) 0x400FF040)
55 #define GPIO_D ((GPIORegs_t*) 0x400FF0C0)
56 #define GPIO_E ((GPIORegs_t*) 0x400FF100)
57 
```

```
58
59 typedef struct {
60   uint32_t ISER[1];
61   uint32_t RSVD[31];
62   uint32_t ICER[1];
63   uint32_t RSVD1[31];
64   uint32_t ISPR[1];
65   uint32_t RSVD2[31];
66   uint32_t ICPR[1];
67   uint32_t RSVD3[31];
68   uint32_t RSVD4[64];
69   uint32_t IPR[1];
70 }NVIC_Regs_t;
71
72 #define NVIC_REG ((NVIC_Regs_t*) 0xE000E100)
73
74 #define GET_SEC_COUNT(x) (CLOCK_GetBusClkFreq() * x)
75
76 void PIT_IRQHandler(void) {
77   //Checa e a interrup  o no canal foi gerada
78   if(PIT_CH_0_REG->TFLAG & (1 << 0)) {
79     GPIO_E->PTOR |= (1 << 31);
80     PIT_CH_0_REG->TFLAG |= (1 << 0);
81   } else {
82     GPIO_D->PTOR |= (1 << 5);
83     PIT_CH_1_REG->TFLAG |= (1 << 0);
84   }
85 }
86 /*
87  * @brief   Application entry point.
88  */
89 int main(void) {
90   uint32_t clock = CLOCK_GetBusClkFreq();
91
92   SIM->SCGC5 |= (1 << 13) | (1 << 12);
93
94   PORT_E->PCR[31] |= (1 << 8);
```

```
95    GPIO_E->PDDR |= (1 << 31);
96    GPIO_E->PSOR |= (1 << 31);
97
98    PORT_D->PCR[5] |= (1 << 8);
99    GPIO_D->PDDR |= (1 << 5);
100   GPIO_D->PSOR |= (1 << 5);
101
102   SIM->SCGC6 |= (1 << 23);
103   PIT_REG->MCR = 0;
104   PIT_CH_0_REG->LDVAL = GET_SEC_COUNT(2);
105   PIT_CH_1_REG->LDVAL = GET_SEC_COUNT(6);
106
107
108   PIT_CH_0_REG->TCTRL = (1 << 1) | (1 << 0);
109   PIT_CH_1_REG->TCTRL = (1 << 1) | (1 << 0);
110
111   NVIC_REG->ISER[0] |= (1 << 22);
112   printf("Clock: %d\n", clock);
113   while(1){
114   }
115
116     return 0 ;
117 }
```

3.4 Item 4

Na figura 2 é mostrado os pulsos decorrentes do acionamento do *timer* para 2 segundos (onda amarela) e para 6 segundos (onda roxa).

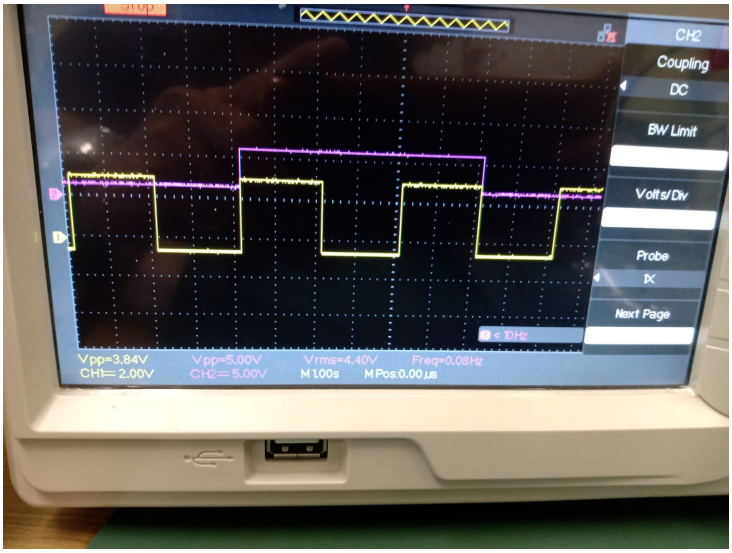

Figura 2: Resultado item 4 para PIT

3.5 Item 5.a)

3.5.1 Funções

O item 5.a) pede para que o item 1 seja refeito utilizando SDK.
Funções:

- CLOCK_EnableClock: Ativa o clock de algum periférico ou *timer* específico.
- PIT_GetDefaultConfig: Obtém a configuração padrão do PIT.
- PIT_Init: inicializa o PIT.
- PIT_SetTimerPeriod: Define o valor inicial de tempo no pit (1 segundo).
- PIT_EnableInterrupts: Ativa interrupção para um canal específico do PIT.
- NVIC_EnableIRQ: Ativa a interrupção do PIT.

- PIT_StartTimer: Inicia o PIT para um canal específico.
- PIT_StopTimer: Para o PIT para um canal específico.
- PIT_GetStatusFlags: Diz se a flag de interrupção de um canal específico está ativada.
- PIT_ClearStatusFlags: Limpa a flag de interrupção de um canal específico.

Foi utilizada a mesma macro para utilizada nos itens anteriores para o cálculo de contagem (o produto entre a frequência do clock e o tempo desejado).

3.5.2 Código

```
/**
 * @file    PIT_SDK.c
 * @brief   Application entry point.
 */
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL43Z4.h"
#include "fsl_debug_console.h"
#include "fsl_pit.h"
#include "fsl_gpio.h"
#include "fsl_port.h"

/* TODO: insert other include files here. */

/* TODO: insert other definitions and declarations here. */

/*
```

```
 * Timer in N sec cnt = (CLOCK_GetBusClkFreq) x N
 *
 * Timer in N mSec cnt = (CLOCK_GetBusClkFreq / 1000) X N
 *
 * Timer in N uSec cnt = (CLOCK_GetBusClkFreq / 1000000) x N
 */

#define GET_SEC_COUNT(x) (CLOCK_GetBusClkFreq() * x)
#define GET_USEC_COUNT(x) ((CLOCK_GetBusClkFreq() * x)
    /1000000)
uint32_t timerCnt;

void PIT_IRQHandler(void)
{
//  GPIO_PortToggle(GPIOD, (1 << 5));
  if(PIT_GetStatusFlags(PIT, 0) & kPIT_TimerFlag) {
    PIT_ClearStatusFlags(PIT, 0, kPIT_TimerFlag);
    GPIO_PortToggle(GPIOD, (1 << 5));
  }
//  PIT_ClearStatusFlags(PIT, kPIT_Chnl_0, kPIT_TimerFlag);
}

void DelaySec(uint32_t sec)
{
  timerCnt=0;
  PIT_StartTimer(PIT, kPIT_Chnl_0);
  while((sec * 100000) > timerCnt);
  PIT_StopTimer(PIT, kPIT_Chnl_0);
}

//void DelayMsec(uint32_t mSec)
//{
//  timerCnt=0;
//  PIT_StartTimer(PIT, kPIT_Chnl_0);
//  while((mSec * 100) > timerCnt);
//  PIT_StopTimer(PIT, kPIT_Chnl_0);
//
```

```
57 //}
58 //
59 //void DelayUSec(uint32_t uSec)
60 //{
61 //  timerCnt=0;
62 //  PIT_StartTimer(PIT, kPIT_Chnl_0);
63 //  while((uSec / 10) > timerCnt);
64 //  PIT_StopTimer(PIT, kPIT_Chnl_0);
65 //}
66
67 /*
68  * @brief   Application entry point.
69  */
70 int main(void) {
71   // Ativar o clock da PORTA_D e PORTA_E
72   CLOCK_EnableClock(kCLOCK_PortD);
73
74   // Ativar o GPIO da PORTA_D_5
75   PORT_SetPinMux(PORTD, 5, kPORT_MuxAsGpio);
76
77   // Configurar GPIO_D pino 5 como sa da
78   gpio_pin_config_t gpioLed = {kGPIO_DigitalOutput, 0};
79   GPIO_PinInit(GPIOD, 5, &gpioLed);
80
81   //Habilitando o clock do PIT
82   CLOCK_EnableClock(kCLOCK_Pit0);
83   pit_config_t pitCfg = {0};
84
85   // Obter configura  o padr o do PIT
86   PIT_GetDefaultConfig(&pitCfg);
87   pitCfg.enableRunInDebug = true;
88
89   // Initializar o PIT
90   PIT_Init(PIT, &pitCfg);
91
92   // Definir o per odo de tempo do PIT
93   PIT_SetTimerPeriod(PIT, kPIT_Chnl_0, GET_SEC_COUNT(2));
```

```
94
95    // Ativar o Timer de interrup  o no canal 0 do PIT
96    PIT_EnableInterrupts(PIT, kPIT_Chnl_0,
       kPIT_TimerInterruptEnable);
97
98    // Ativar interru  o do PIT no NVIC
99    NVIC_EnableIRQ(PIT_IRQn);
100
101   // Iniciar o Timer do PIT
102   PIT_StartTimer(PIT, kPIT_Chnl_0);
103
104   while(1) {
105   }
106   return 0;
107 }
```

3.6 Item 5.b)

3.6.1 Funções

O item 5.b) pede para que o item 2 seja refeito utilizando SDK.

Como o *timer* PIT não possui opção de prescale, então o mesmo código do item anterior se aplica aqui, mudando apenas o valor do parâmetro da macro *GET_SEC_COUNT* para 8.

3.6.2 Código

```
1  /**
2   * @file    PIT_SDK.c
3   * @brief   Application entry point.
4   */
5  #include <stdio.h>
6  #include "board.h"
7  #include "peripherals.h"
8  #include "pin_mux.h"
9  #include "clock_config.h"
10 #include "MKL43Z4.h"
```

```
11 #include "fsl_debug_console.h"
12 #include "fsl_pit.h"
13 #include "fsl_gpio.h"
14 #include "fsl_port.h"
15
16 /* TODO: insert other include files here. */
17
18 /* TODO: insert other definitions and declarations here. */
19
20 /*
21  * Timer in N sec cnt = (CLOCK_GetBusClkFreq) x N
22  *
23  * Timer in N mSec cnt = (CLOCK_GetBusClkFreq / 1000) X N
24  *
25  * Timer in N uSec cnt = (CLOCK_GetBusClkFreq / 1000000) x N
26  */
27
28 #define GET_SEC_COUNT(x) (CLOCK_GetBusClkFreq() * x)
29 #define GET_USEC_COUNT(x) ((CLOCK_GetBusClkFreq() * x)
      /1000000)
30 uint32_t timerCnt;
31
32 void PIT_IRQHandler(void)
33 {
34 //  GPIO_PortToggle(GPIOD, (1 << 5));
35   if(PIT_GetStatusFlags(PIT, 0) & kPIT_TimerFlag) {
36     PIT_ClearStatusFlags(PIT, 0, kPIT_TimerFlag);
37     GPIO_PortToggle(GPIOD, (1 << 5));
38   }
39 //  PIT_ClearStatusFlags(PIT, kPIT_Chnl_0, kPIT_TimerFlag);
40 }
41
42 void DelaySec(uint32_t sec)
43 {
44   timerCnt=0;
45   PIT_StartTimer(PIT, kPIT_Chnl_0);
46   while((sec * 100000) > timerCnt);
```

```
47   PIT_StopTimer(PIT, kPIT_Chnl_0);
48 }
49
50 //void DelayMsec(uint32_t mSec)
51 //{
52 //   timerCnt=0;
53 //   PIT_StartTimer(PIT, kPIT_Chnl_0);
54 //   while((mSec * 100) > timerCnt);
55 //   PIT_StopTimer(PIT, kPIT_Chnl_0);
56 //
57 //}
58 //
59 //void DelayUSec(uint32_t uSec)
60 //{
61 //   timerCnt=0;
62 //   PIT_StartTimer(PIT, kPIT_Chnl_0);
63 //   while((uSec / 10) > timerCnt);
64 //   PIT_StopTimer(PIT, kPIT_Chnl_0);
65 //}
66
67 /*
68  * @brief   Application entry point.
69  */
70 int main(void) {
71   // Ativar o clock da PORTA_D e PORTA_E
72   CLOCK_EnableClock(kCLOCK_PortD);
73
74   // Ativar o GPIO da PORTA_D_5 e PORTA_E_31
75   PORT_SetPinMux(PORTD, 5, kPORT_MuxAsGpio);
76
77   //Configurando os leds como sa da
78   gpio_pin_config_t gpioLed = {kGPIO_DigitalOutput, 0};
79
80   // Configurar GPIO_D pino 5 e GPIO_E pino 31 como sa da
81   GPIO_PinInit(GPIOD, 5, &gpioLed);
82
83   //Habilitando clock do PIT
```

```
84    CLOCK_EnableClock(kCLOCK_Pit0);
85    pit_config_t pitCfg0 = {0}; //Canal 0
86
87    // Obter configura  o padr o do PIT
88    PIT_GetDefaultConfig(&pitCfg0);
89    pitCfg0.enableRunInDebug = true;
90
91    // Initializar o PIT
92    PIT_Init(PIT, &pitCfg0);
93
94    // Definir o per odo de tempo do PIT
95    PIT_SetTimerPeriod(PIT, kPIT_Chnl_0, GET_SEC_COUNT(8));
96
97    // Ativar o Timer de interrup  o
98    PIT_EnableInterrupts(PIT, kPIT_Chnl_0,
       kPIT_TimerInterruptEnable);
99
100   // Ativar interru  o do PIT no NVIC
101   NVIC_EnableIRQ(PIT_IRQn);
102
103   // Iniciar o Timer do PIT
104   PIT_StartTimer(PIT, kPIT_Chnl_0);
105
106   while(1) {
107   }
108   return 0;
109 }
```

4 TPM

O TPM(Timer/PWM Module) é um *timer* é um temporizador de dois a oito canais que suporta captura de entrada, comparação de saída e geração de sinais PWM para controlar o motor elétrico e aplicações de gerenciamento de energia. Os registradores de contagem, comparação e captura são cronometrados por um relógio assíncrono que pode permanecer habilitado em modos

de baixa potência.

O recursos do TPM incluem:

- O modo de clock é configurável
- Apresenta *prescaler*, então o clock pode ser dividido por 1, 2, 4, 8... 128.
- Contém um contador de 16 bits.
- Contém 6 canais que podem ser configurados.

Para a prática de TPM, outra placa foi utilizada, a FRDM-KL46Z, a qual é muito parecida com a FRDM-KL43Z utilizada nas práticas anteriores.

4.1 Item 1

O item 1 da prática de TPM pede que um LED pisque a cada 1s sem a utlização do *prescaler*.

4.1.1 Registradores

O módulo TPM apresenta 3 registradores principais. O primeiro deles é *Status and Control* (TPM0_SC), o segundo é *Counter* (TPM0_CNT) e por último temos *Modulo* (TPM0_MOD).

TPM0_SC contém 32 bits. Os bits 0-2 determinam a seleção do valor de *prescale*, os bits 3-4 determinando o clock do contador e como ele irá incrementar, o bit 6 habilita a interrupção de *overflow* e o bit 7 trata-se da flag de *overflow*.

TPM0_CNT é um registrador de 16 bits, ele será o contador do temporizador.

TPM0_MOD é o registrador de módulo, ele receberá o valor final da contagem do temporizador. Esse também é um registrador de 16 bits.

No TPM existem canais e cada canal tem registradores específicos. O primeiro deles é o *Status and Control* (TPMx_CnSC) e Value (TPMx_CnV). O

TPMx_CnSC serve para configurar o canal que você está utilizando, pode ser configurado o modo, a borda e o nível de seleção. Já o TPMx_CnV serve para receber o valor de captura de acordo com os modos que foram configurados. Entretando, nesta práticas esses registradores não serão explorados.

4.1.2 Explicação código

Na função principal temos a ativação do bit 1 no registrador C1 do MGC (*Multipurpose Clock Generator*), selecionando o clock interno do microcontrolador que é de aproximadamente 32kHz (Linha 90). Em sequência, no registrador SOPT2 os bit 24-25 são alterados para que o clock de 32kHz seja selecionado. Na linha 101 o registrador SCGC6 é setado no bit 24 que corresponde a ativação do clock no módulo TPM0 que é o utlizado.

Na linha 110, o bit 6 do registrador TPM0_SC é alterado para que a interrupção por estouro seja habilitada, em seguida o registrador de contador é resetado.

Na linha 119 o valor de *prescale* é definido em 1, ou seja, não será utilizado o *prescaler* nessa ocasião. Em seguida, o valor de MOD é atribuído, nesse caso o valor de frequência do clock selecionado foi atribuído, por conta que a onda não é dividida, então baseando-se pela Equação 1, temos que MOD = 32000.

De volta ao registrador TPMx_CnSC, agora é hora configurar os bits 3-4, setando apenas o bit 3 que significa que o contador vai incrementar a cada pulso de clock do TPM. Está feita nossa configuração de TPM.

Próximo passo agora é configurar e habilitar o LED a ser usado, que já foi visto como no itens anteriores. A rotina de interrupção também é habilitada.

Na rotina de interrupção **TPM0_IRQHandler** temos uma verificação do bit 7 do registrador TPM0_SC, para saber se a flag de interrupção foi gerada, se ocorreu o estouro ou não, caso tenha acontecido, então o LED tem seu estado alterado e é escrito 1 no bit 7 novamente para limpar a interrupção. E assim o LED fica piscando em 1s.

```c
/**
 * @file    TPM.c
 * @brief   Application entry point.
 */
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL46Z4.h"
#include "fsl_debug_console.h"

/* TODO: insert other include files here. */

/* TODO: insert other definitions and declarations here. */

typedef struct{
  uint32_t PCR[32];
}PORTRegs_t;

#define PORT_B ((PORTRegs_t *)0x4004A000)
#define PORT_E ((PORTRegs_t*) 0x4004D000)

typedef struct{
  uint32_t PDOR;
  uint32_t PSOR;
  uint32_t PCOR;
  uint32_t PTOR;
  uint32_t PDIR;
  uint32_t PDDR;
}GPIORegs_t;

#define GPIO_B ((GPIORegs_t*) 0x400FF040)
#define GPIO_E ((GPIORegs_t*) 0x400FF100)

typedef struct {
  uint32_t ISER[1];
```

```
38   uint32_t RSVD[31];
39   uint32_t ICER[1];
40   uint32_t RSVD1[31];
41   uint32_t ISPR[1];
42   uint32_t RSVD2[31];
43   uint32_t ICPR[1];
44   uint32_t RSVD3[31];
45   uint32_t RSVD4[64];
46   uint32_t IPR[1];
47 }NVIC_Regs_t;
48
49 #define NVIC_REG ((NVIC_Regs_t *)0xE000E100)
50
51 typedef struct{
52   uint32_t SC;
53   uint32_t CNT;
54   uint32_t MOD;
55 }TPMR_Regs_t;
56
57 #define TPM_REG ((TPMR_Regs_t *) 0x40038000)
58
59 typedef struct{
60   uint32_t STATUS;
61   uint32_t CONF;
62 }TPM_CONF_Regs_t;
63
64 #define TPM_CONF_REG ((TPM_CONF_Regs_t *) 0x40038050)
65
66 typedef struct{
67   uint32_t SOPT2;
68 }SOPT2R_Regs_t;
69
70 #define SOPT2_REG ((SOPT2R_Regs_t *) 0x40048004)
71
72 void TPM0_IRQHandler(void){
73   if((TPM_REG->SC & (1 << 7))){
74     GPIO_E->PTOR = (1 << 29);
```

```
75
76      //(TOF) Timer Overflow Flag -- 1 : LPTPM counter has
        overflowed.
77      TPM_REG->SC |= (1 << 7);
78
79      //TPM_REG->CNT = 0x0000;
80      TPM_REG->CNT = 0x0000; // resetar
81    }
82 }
83
84 /*
85  * @brief   Application entry point.
86  */
87 int main(void) {
88
89   // Habilitando o clock de 32kHz
90   MCG->C1 |= (1 << 1);
91
92   /*
      ********************************************************************************
      */
93   //pagina 195
94   //(TPMSRC) clock source select
95   //Selects the clock source for the TPM counter clock
96   // 01  MCGFLLCLK clock or MCGPLLCLK/2
97   SOPT2_REG->SOPT2 |= (0b11 << 24);//clock MCGFLLCLK    de
       20,97
98   //PLL/FLL clock select
99   //Selects the MCGPLLCLK or MCGFLLCLK clock for various
       peripheral clocking options.
100  // 0  MCGFLLCLK clock
101  SIM->SCGC6 |= (1 << 24);
102
103  /*
104   * Configura  o TPM
105   *
106   */
```

```
107 //(TOIE) Timer Overflow Interrupt Enable -- Enables LPTPM
     overflow interrupts.
108 // 0  Disable TOF interrupts. Use software polling or DMA
     request.
109 // 1  Enable TOF interrupts. An interrupt is generated when
       TOF equals one.
110 TPM_REG->SC |= (1 << 6);//ativando a interrup  o para TOF
111
112 TPM_REG->CNT = 0x0000; // resetar
113
114 //TPM_REG->MOD = TPM_CLOCK / (1 << (TPM_clk_PRESCALE + 1))/
     TPM_OVerflow_frequency
115 // MOD = (32768Hz/32)*1 = 1024
116
117
118 //(PS)  Prescaler
119 TPM_REG->SC |= (0b000 << 0); // desativando o preescale
120 TPM_REG->MOD = 32000; //coloca a frequencia do pino, 32kHz,
       (MCG->C1)
121
122 //(CMOD) Clock Mode Selection -- 01 : LPTPM counter
     increments on every LPTPM counter clock
123 TPM_REG->SC |= (0b01 << 3);//incrementa a cada pulso do TPM
124
125 /*
     ******************************************************************************
     */
126
127 SIM->SCGC5 |= (1 << 13); //ativar clock porta E
128
129 /*
     ******************************************************************************
     */
130
131 //Port_E_29 como GPIO
132 PORT_E->PCR[29] |= (1 << 8);
133
```

```
134   //GPIO_E_29 como output
135   GPIO_E->PDDR |= (1 << 29);
136
137   /*
       ********************************************************************************
       */
138
139   //NVIC_EnableIRQ(TPM0_IRQn);
140   NVIC_REG->ISER[0] = (1 << 17);
141
142   while(1){
143   }
144
145
146   return 0 ;
147 }
```

4.2 Item 2

4.2.1 Explixação e Registradores

O item 2 pede que um LED seja acionado a cada 8 segundo e apagado a cada 8 segundos utilizando prescale.

Foi utilizado o clock MCGIRCLK, de 32kHz de frequência. Para isso foi preciso setar os bits 24 e 25 de SOPT2 em 1. O pino escolhido pertence ao TPM0, para ativar o clock dele foi setado o bit 24 de SCGC6. O prescale escolhido foi o 32, então os bits 0 e 2 de SC foram setados em 1. Para o cálculo do MOD foi utilizada a seguinte fórmula:

$$MOD = \frac{frequência}{prescale} \cdot (tempo\_desejado)$$

Para 8s, temos um valor de MOD = 8192, considerando a frequência de 32768Hz. Para habilitar interrupção e o incremento do contador, os bits 6 e 3 de SC foram setados em 1.

No tratador de interrupção é feito um teste no bit 7 de SC, se o mesmo estiver setado então ocorreu uma interrupção e o LED sofre um toggle, a flag de interrupção é limpa e o contador é resetado.

4.2.2 Código

```
/**
 * @file    TPM.c
 * @brief   Application entry point.
 */
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL46Z4.h"
#include "fsl_debug_console.h"

/* TODO: insert other include files here. */

/* TODO: insert other definitions and declarations here. */

typedef struct{
  uint32_t PCR[32];
}PORTRegs_t;

#define PORT_B ((PORTRegs_t *)0x4004A000)
#define PORT_E ((PORTRegs_t*) 0x4004D000)

typedef struct{
  uint32_t PDOR;
  uint32_t PSOR;
  uint32_t PCOR;
  uint32_t PTOR;
  uint32_t PDIR;
  uint32_t PDDR;
```

```
31 }GPIORegs_t;
32
33 #define GPIO_B ((GPIORegs_t*) 0x400FF040)
34 #define GPIO_E ((GPIORegs_t*) 0x400FF100)
35
36 typedef struct {
37   uint32_t ISER[1];
38   uint32_t RSVD[31];
39   uint32_t ICER[1];
40   uint32_t RSVD1[31];
41   uint32_t ISPR[1];
42   uint32_t RSVD2[31];
43   uint32_t ICPR[1];
44   uint32_t RSVD3[31];
45   uint32_t RSVD4[64];
46   uint32_t IPR[1];
47 }NVIC_Regs_t;
48
49 #define NVIC_REG ((NVIC_Regs_t *)0xE000E100)
50
51 typedef struct{
52   uint32_t SC;
53   uint32_t CNT;
54   uint32_t MOD;
55 }TPMR_Regs_t;
56
57 #define TPM_REG ((TPMR_Regs_t *) 0x40038000)
58
59 typedef struct{
60   uint32_t STATUS;
61   uint32_t CONF;
62 }TPM_CONF_Regs_t;
63
64 #define TPM_CONF_REG ((TPM_CONF_Regs_t *) 0x40038050)
65
66 typedef struct{
67   uint32_t SOPT2;
```

```
68 }SOPT2R_Regs_t;
69
70 #define SOPT2_REG ((SOPT2R_Regs_t *) 0x40048004)
71
72 void TPM0_IRQHandler(void){
73   if((TPM_REG->SC & (1 << 7))){
74     GPIO_E->PTOR = (1 << 29);
75
76     //(TOF) Timer Overflow Flag -- 1 : LPTPM counter has
     overflowed.
77     TPM_REG->SC |= (1 << 7);
78
79     //TPM_REG->CNT = 0x0000;
80     TPM_REG->CNT = 0x0000; // resetar
81   }
82 }
83
84 /*
85  * @brief   Application entry point.
86  */
87 int main(void) {
88
89   // Habilitando o clock de 32kHz
90   MCG->C1 |= (1 << 1);
91   /*
      ********************************************************************
     */
92   //pagina 195
93   //(TPMSRC) clock source select
94   //Selects the clock source for the TPM counter clock
95   // 01  MCGFLLCLK clock or MCGPLLCLK/2
96   SOPT2_REG->SOPT2 |= (0b11 << 24);//clock MCGFLLCLK    de
     20,97
97   //PLL/FLL clock select
98   //Selects the MCGPLLCLK or MCGFLLCLK clock for various
     peripheral clocking options.
99   // 0  MCGFLLCLK clock
```

```
100  SIM->SCGC6 |= (1 << 24);
101
102  /*
103   * Configura  o TPM
104   *
105   */
106  //(TOIE) Timer Overflow Interrupt Enable -- Enables LPTPM
      overflow interrupts.
107  // 0  Disable TOF interrupts. Use software polling or DMA
      request.
108  // 1  Enable TOF interrupts. An interrupt is generated when
       TOF equals one.
109  TPM_REG->SC |= (1 << 6);//ativando a interrup  o para TOF
110
111  TPM_REG->CNT = 0x0000; // resetar
112
113  //(PS)  Prescaler
114  TPM_REG->SC |= (0b101 << 0);//ativado o prescaler para 32
115
116  //TPM_REG->MOD = TPM_CLOCK / (1 << (TPM_clk_PRESCALE + 1))/
      TPM_OVerflow_frequency
117  TPM_REG->MOD = 8191; //T = 1/(32kHz/32) --> T = 0.001s -->
      MOD = 8/0.001 = 8000 aprox 8192 (1024*8)
118
119  //(CMOD) Clock Mode Selection -- 01 : LPTPM counter
      increments on every LPTPM counter clock
120  TPM_REG->SC |= (0b01 << 3);//incrementa a cada pulso do
      LPTPM
121
122  /*
      ******************************************************************
      */
123
124  SIM->SCGC5 |= (1 << 13); //ativar clock porta E
125
126  /*
      ******************************************************************
```

```
     */
127
128   //Port_E_31 como GPIO
129   PORT_E->PCR[29] |= (1 << 8);
130
131   //GPIO_E_31 como sa da
132   GPIO_E->PDDR |= (1 << 29);
133
134   /*
       ********************************************************************************
       */
135
136   //NVIC_EnableIRQ(TPM0_IRQn);
137   NVIC_REG->ISER[0] = (1 << 17);
138
139   while(1){
140   }
141
142
143   return 0 ;
144 }
```

4.3 Item 3

4.3.1 Explicação e Registradores

O item 3 pede que um LED acenda a cada 2 segundos e apague no intervalo de mesmo tempo e que aconteça o mesmo com outro LED no tempo de 6 segundos, utilizando prescale.

Foi utilizado o clock MCGIRCLK, de 32kHz de frequência. Para isso foi preciso setar os bits 24 e 25 de SOPT2 em 1. Os pinos escolhidos pertencem aos TPM0 e TPM1, para ativar o clock dos 2 foram setados os bits 24 e 25 de SCGC6. O prescale escolhido foi o 32, então os bits 0 e 2 de SC foram setados em 1. Para o cálculo do MOD foi utilizada a mesma fórmula do item anterior.

Para habilitar interrupção e o incremento do contador, os bits 6 e 3 de SC foram setados em 1.

A escolha de *prescale*, a ativação de interrupção, o incremento do contador e o cálculo do MOD precisam ser feitos para cada TPM.

Cada TPM possui um tratador de interrupção, então o mesmo teste feito na flag de interrupção do item passado é feito em cada tratador aqui.

4.3.2 Código

```
/**
 * @file    TPM.c
 * @brief   Application entry point.
 */
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL46Z4.h"
#include "fsl_debug_console.h"

/* TODO: insert other include files here. */

/* TODO: insert other definitions and declarations here. */

typedef struct{
  uint32_t PCR[32];
}PORTRegs_t;

#define PORT_D ((PORTRegs_t *) 0x4004C000)
#define PORT_E ((PORTRegs_t *) 0x4004D000)

typedef struct{
  uint32_t PDOR;
  uint32_t PSOR;
```

```
27   uint32_t PCOR;
28   uint32_t PTOR;
29   uint32_t PDIR;
30   uint32_t PDDR;
31 }GPIORegs_t;
32 
33 #define GPIO_D ((GPIORegs_t *) 0x400FF0C0)
34 #define GPIO_E ((GPIORegs_t *) 0x400FF100)
35 
36 typedef struct {
37   uint32_t ISER[1];
38   uint32_t RSVD[31];
39   uint32_t ICER[1];
40   uint32_t RSVD1[31];
41   uint32_t ISPR[1];
42   uint32_t RSVD2[31];
43   uint32_t ICPR[1];
44   uint32_t RSVD3[31];
45   uint32_t RSVD4[64];
46   uint32_t IPR[1];
47 }NVIC_Regs_t;
48 
49 #define NVIC_REG ((NVIC_Regs_t *) 0xE000E100)
50 
51 typedef struct{
52   uint32_t SC;
53   uint32_t CNT;
54   uint32_t MOD;
55 }TPMR_Regs_t;
56 
57 #define TPM_0_REG ((TPMR_Regs_t *) 0x40038000)
58 #define TPM_1_REG ((TPMR_Regs_t *) 0x40039000)
59 
60 //typedef struct{
61 //   uint32_t STATUS;
62 //   uint32_t CONF;
63 //}TPM_CONF_Regs_t;
```

```
64 //
65 //#define TPM_CONF_REG ((TPM_CONF_Regs_t *) 0x40038050)
66
67 typedef struct{
68   uint32_t SOPT2;
69 }SOPT2R_Regs_t;
70
71 #define SOPT2_REG ((SOPT2R_Regs_t *) 0x40048004)
72
73 void TPM0_IRQHandler(void){
74   if((TPM_0_REG->SC & (1 << 7))){
75     GPIO_E->PTOR = (1 << 29);
76
77     //(TOF) Timer Overflow Flag -- 1 : LPTPM counter has
     overflowed.
78     TPM_0_REG->SC |= (1 << 7);
79
80     //TPM_REG->CNT = 0x0000;
81     TPM_0_REG->CNT = 0x0000; // resetar
82   }
83 }
84
85
86 void TPM1_IRQHandler(void) {
87   if(TPM_1_REG->SC & (1 << 7)) {
88     GPIO_D->PTOR = (1 << 5);
89     TPM_1_REG->SC |= (1 << 7);
90     TPM_1_REG->CNT = 0x0000;
91   }
92 }
93 /*
94  * @brief   Application entry point.
95  */
96 int main(void) {
97
98   // Habilitando o clock de 32kHz
99   MCG->C1 |= (1 << 1);
```

```
100 /*
     ********************************************************************************
     */
101 //pagina 195
102 //(TPMSRC) clock source select
103 //Selects the clock source for the TPM counter clock
104 // 01  MCGFLLCLK clock or MCGPLLCLK/2
105 SOPT2_REG->SOPT2 |= (0b11 << 24);//clock MCGFLLCLK    de
     20,97
106 //PLL/FLL clock select
107 //Selects the MCGPLLCLK or MCGFLLCLK clock for various
     peripheral clocking options.
108 // 0  MCGFLLCLK clock
109
110 //Habilitando o clock no TMP0 e TMP1
111 SIM->SCGC6 |= (1 << 24) | (1 << 25);
112
113 /*
114  * Configura  o TPM
115  *
116  */
117 //(TOIE) Timer Overflow Interrupt Enable -- Enables LPTPM
     overflow interrupts.
118 // 0  Disable TOF interrupts. Use software polling or DMA
     request.
119 // 1  Enable TOF interrupts. An interrupt is generated when
      TOF equals one.
120 TPM_0_REG->SC |= (1 << 6);//ativando a interrup  o para
     TOF
121
122 TPM_0_REG->CNT = 0x0000; // resetar
123
124 //(PS)  Prescaler
125 TPM_0_REG->SC |= (0b101 << 0);//ativado o prescaler para 32
126
127 //TPM_REG->MOD = TPM_CLOCK / (1 << (TPM_clk_PRESCALE + 1))/
     TPM_OVerflow_frequency
```

```
128 TPM_0_REG->MOD = (1024-1)*2; //T = 1/(32kHz/32) --> T =
     0.001s --> MOD = 8/0.001 = 8000 aprox 8192 (1024*8)
129
130 //(CMOD) Clock Mode Selection -- 01 : LPTPM counter
     increments on every LPTPM counter clock
131 TPM_0_REG->SC |= (0b01 << 3);//incrementa a cada pulso do
     LPTPM
132
133
134 TPM_1_REG->SC |= (1 << 6);
135
136 TPM_1_REG->CNT = 0x0000; // resetar
137
138 //(PS)  Prescaler
139 TPM_1_REG->SC |= (0b101 << 0);//ativado o prescaler para 32
140
141 //TPM_REG->MOD = TPM_CLOCK / (1 << (TPM_clk_PRESCALE + 1))/
     TPM_OVerflow_frequency
142 TPM_1_REG->MOD = (1024-1)*6; //T = 1/(32kHz/32) --> T =
     0.001s --> MOD = 8/0.001 = 8000 aprox 8192 (1024*8)
143
144 //(CMOD) Clock Mode Selection -- 01 : LPTPM counter
     increments on every LPTPM counter clock
145 TPM_1_REG->SC |= (0b01 << 3);//incrementa a cada pulso do
     LPTPM
146 /*
     ******************************************************************************
     */
147
148 SIM->SCGC5 |= (1 << 13) | (1 << 12); //ativar clock porta E
149
150 /*
     ******************************************************************************
     */
151
152 //Port_E_31 como GPIO
153 PORT_E->PCR[29] |= (1 << 8);
```

```
154   PORT_D->PCR[5] |= (1 << 8);
155   //GPIO_E_31 como sa da
156   GPIO_E->PDDR |= (1 << 29);
157   GPIO_D->PDDR |= (1 << 5);
158
159   /*
       ********************************************************************************
       */
160
161   //NVIC_EnableIRQ(TPM0_IRQn);
162   NVIC_REG->ISER[0] = (1 << 17) | (1 << 18);
163
164   while(1){
165   }
166
167
168   return 0 ;
169 }
```

4.4 Item 4

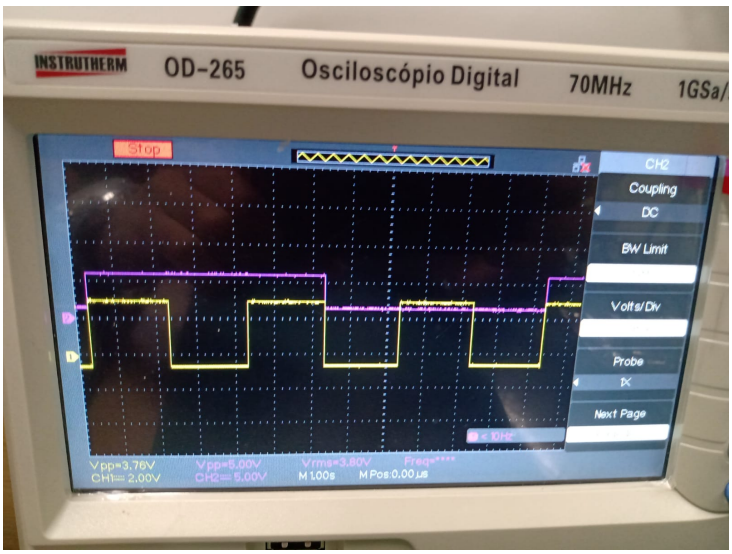

Figura 3: Resultado item 4 para TPM

Na figura 3 é mostrado os pulsos decorrentes do acionamento do *timer* para 2 segundos (onda amarela) e para 6 segundos (onda roxa).

4.5 Item 5.a)

4.5.1 Funções

O item 5.a) pede para que o item 1 seja refeito utilizando SDK.
Funções:

- CLOCK_SetInternalRefClkConfig: Configura o clock para MCGIRCLK (32kHz).
- CLOCK_EnableClock: Ativa o clock do TPM0.
- TPM_Init: Configura o TPM.
- TPM_SetTimerPeriod: Define o MOD utilizado, como é sem prescale, então foi utilizado a opção 0b000 do prescale.
- TPM_EnableInterrupts: Habilita interrupção para o TPM.
- TPM_StartTimer: Inicializa o TPM.
- TPM_ClearStatusFlags: Limpa a flag de interrupção.

Na linha 56 é definido o prescale, nesse caso 1. As demais funções são as mesmas utilizadas em itens anteriores para configuração de saída.

4.5.2 Código

```
1 /**
2  * @file    TPM.c
3  * @brief   Application entry point.
4  */
5 #include <stdio.h>
```

```
6  #include "board.h"
7  #include "peripherals.h"
8  #include "pin_mux.h"
9  #include "clock_config.h"
10 #include "MKL46Z4.h"
11 #include "fsl_debug_console.h"
12 #include "fsl_tpm.h"
13 #include "fsl_clock.h"
14 #include "fsl_port.h"
15 #include "fsl_gpio.h"
16
17 /* TODO: insert other include files here. */
18
19 /* TODO: insert other definitions and declarations here. */
20
21 void init_LEDS(void);
22 void init_TPM0(void);
23 void init_TPM1(void);
24
25 void TPM0_IRQHandler(void){
26   TPM_ClearStatusFlags(TPM0, kTPM_TimeOverflowFlag);
27   GPIO_TogglePinsOutput(GPIOE, (1 << 29));
28 }
29 /*
30  * @brief   Application entry point.
31  */
32 int main(void) {
33
34   init_LEDS();
35   init_TPM0();
36
37   while(1){}
38
39   return 0 ;
40 }
41
42 void init_LEDS(void) {
```

```
43   CLOCK_EnableClock(kCLOCK_PortE);
44
45   PORT_SetPinMux(PORTE, 29, kPORT_MuxAsGpio);
46
47   gpio_pin_config_t led2 = {kGPIO_DigitalOutput, 0};
48   GPIO_PinInit(GPIOE, 29, &led2);
49 }
50
51 void init_TPM0(void) {
52
53   CLOCK_SetInternalRefClkConfig(kMCG_IrclkEnable,
       kMCG_IrcSlow, 0x0u);
54   CLOCK_EnableClock(kCLOCK_Tpm0);
55
56   tpm_config_t tpm0_config = {};
57   TPM_GetDefaultConfig(&tpm0_config);
58
59   tpm0_config.prescale = kTPM_Prescale_Divide_1;
60   TPM_Init(TPM0, &tpm0_config);
61
62   CLOCK_SetTpmClock(3);
63   TPM_SetTimerPeriod(TPM0, 32000);
64
65   TPM_EnableInterrupts(TPM0, kTPM_TimeOverflowInterruptEnable
       );
66   NVIC_EnableIRQ(TPM0_IRQn);
67
68   TPM_StartTimer(TPM0, kTPM_SystemClock);
69 }
```

4.6 Item 5.b)

4.6.1 Explicação e Registradores

O item 5.b) pede para que o item 2 seja refeito utilizando SDK. A mesma lógica do item 5.a) se aplica aqui, mudando apenas a escolha de prescale (64) e o MOD.

4.6.2 Código

```
/**
 * @file    TPM.c
 * @brief   Application entry point.
 */
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL46Z4.h"
#include "fsl_debug_console.h"
#include "fsl_tpm.h"
#include "fsl_clock.h"
#include "fsl_port.h"
#include "fsl_gpio.h"

/* TODO: insert other include files here. */

/* TODO: insert other definitions and declarations here. */

void init_LEDS(void);
void init_TPM0(void);
void init_TPM1(void);

void TPM0_IRQHandler(void){
  TPM_ClearStatusFlags(TPM0, kTPM_TimeOverflowFlag);
  GPIO_TogglePinsOutput(GPIOE, (1 << 29));
}
/*
 * @brief   Application entry point.
 */
int main(void) {

  init_LEDS();
  init_TPM0();
```

```
36
37    while(1){}
38
39    return 0 ;
40  }
41
42  void init_LEDS(void) {
43    CLOCK_EnableClock(kCLOCK_PortD);
44    CLOCK_EnableClock(kCLOCK_PortE);
45
46    PORT_SetPinMux(PORTD, 5, kPORT_MuxAsGpio);
47    PORT_SetPinMux(PORTE, 29, kPORT_MuxAsGpio);
48
49    gpio_pin_config_t led1 = {kGPIO_DigitalOutput, 0};
50    GPIO_PinInit(GPIOD, 5, &led1);
51
52    gpio_pin_config_t led2 = {kGPIO_DigitalOutput, 0};
53    GPIO_PinInit(GPIOE, 29, &led2);
54  }
55
56  void init_TPM0(void) {
57
58    CLOCK_SetInternalRefClkConfig(kMCG_IrclkEnable,
       kMCG_IrcSlow, 0x0u);
59    CLOCK_EnableClock(kCLOCK_Tpm0);
60
61    tpm_config_t tpm0_config = {};
62    TPM_GetDefaultConfig(&tpm0_config);
63
64    tpm0_config.prescale = kTPM_Prescale_Divide_64;
65    TPM_Init(TPM0, &tpm0_config);
66
67    CLOCK_SetTpmClock(3);
68    //MOD = 8s * (32768/64) =
69    TPM_SetTimerPeriod(TPM0, 4096);
70
```

```
71    TPM_EnableInterrupts(TPM0, kTPM_TimeOverflowInterruptEnable
       );
72    NVIC_EnableIRQ(TPM0_IRQn);
73
74    TPM_StartTimer(TPM0, kTPM_SystemClock);
75 }
```