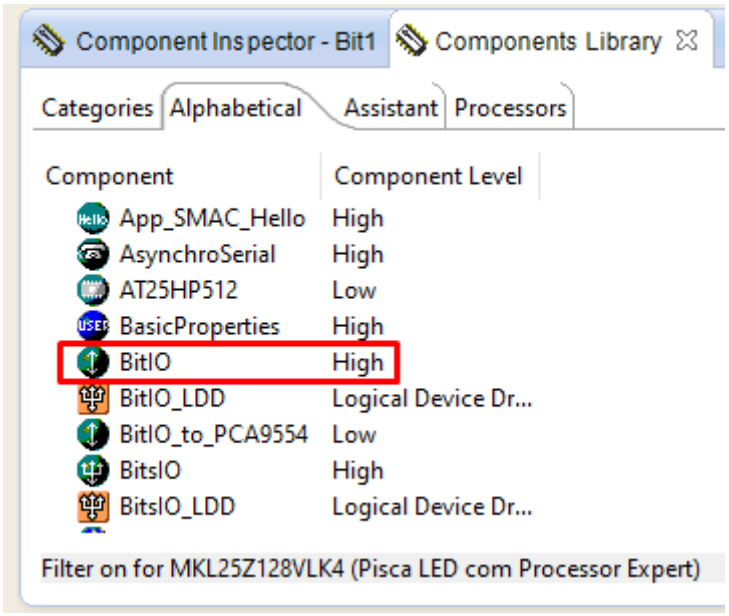
PSI3441 - Arquitetura de Sistemas Embarcados

Relatório Exercício Prático 02

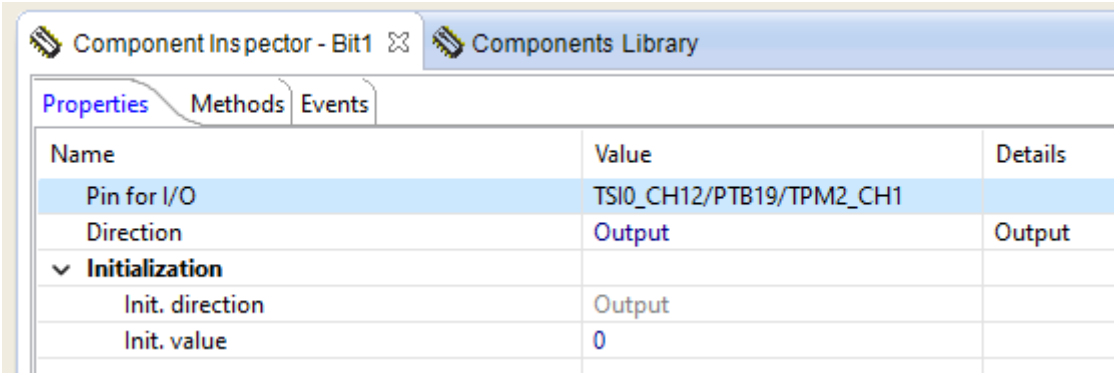
Nome: Kevin Kirsten Lucas nºUSP: 10853306

1) Faça um LED piscar utilizando o Processor Expert do CodeWarrior.

Utilizando as ferramentas do CodeWarrior, é possível criar facilmente um componente do tipo BitIO da biblioteca de componentes e atribuir a ele a configuração de saída do Led verde PTB19, como mostrado abaixo:

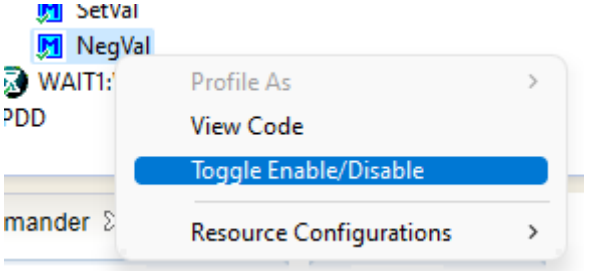


Criação do Componente Bit IO



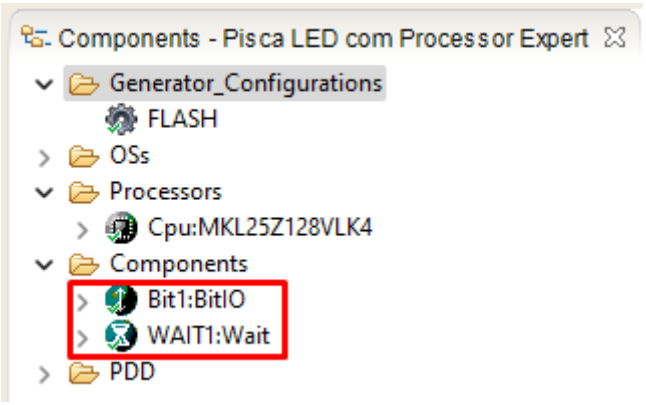
Atribuição PTB19 como Output

É importante também lembrar de ativar o método NegVal do nosso BitIO para que ele funcione no modo toggle:



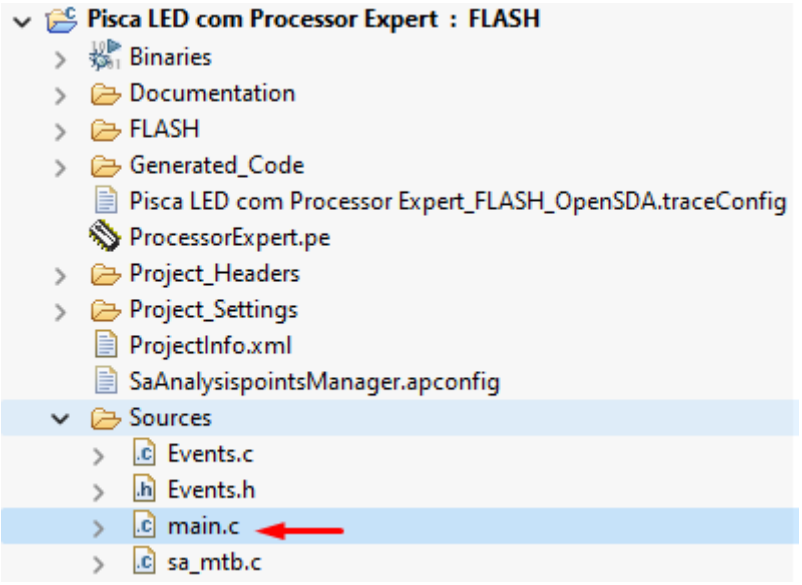
Habilitando Negval

Em seguida, basta importar um componente pronto que realiza a função de timer para adicionar um delay durante o código, gastando alguns ciclos de clock.
Nossa árvore de componentes ficou da seguinte forma:



Árvore de componentes

Finalmente o código final teve linhas e deve ser inserido dentro do arquivo main dentro do diretório sources, como mostrado a seguir:

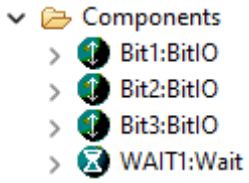


Arquivo main.c no diretório Sources

Código da Tarefa 01

2) Agora faça os 3 LEDs da placa piscarem sequencialmente.

Para fazer os 3 LEDs piscarem sequencialmente, foi necessário atribuir os pinos PTB19 (LED Verde), PTB18 (LED Vermelho) e PTD1 (LED Azul), como pinos de saída sendo componentes BitIO.



Componentes da Tarefa 02

Com o loop abaixo foi possível fazer os LEDs piscarem sequencialmente.

```
while(1)
{
    Bit1_ClrVal(); // Green LED On
    Bit2_SetVal(); // Red   LED Off
    Bit3_SetVal(); // Blue  LED Off
    WAIT1_Waitms(200);

    Bit1_SetVal(); // Green LED Off
    Bit2_ClrVal(); // Red   LED On
    Bit3_SetVal(); // Blue  LED Off
    WAIT1_Waitms(200);

    Bit1_SetVal(); // Green LED Off
    Bit2_SetVal(); // Red   LED Off
    Bit3_ClrVal(); // Blue  LED On
    WAIT1_Waitms(200);
}
```

Código da Tarefa 02

3) Faça um LED piscar sem utilizar o Processor Expert do CodeWarrior.

Para fazer um LED piscar (vamos utilizar o LED Verde) sem o Processor Expert, é necessário atribuir os endereços dos registradores responsáveis por configurar a saída do LED (PTB19) corretamente.

Abaixo fazemos a configuração de acordo com os comentários e configuramos o GPIO da porta B funcionando como Toggle. Desta forma fica mais fácil ligá-lo e desligá-lo.

```
#define SIM_SCGC5   (*((volatile unsigned int*)0x40048038)) /* System Integration Module System Clock Gating Control Register 5 */
#define PORTB_PCR19 (*((volatile unsigned int*)0x4004A04C)) /* Port B Pin Control Register 19 */
#define GPIO_PDDR   (*((volatile unsigned int*)0x400FF054)) /* Port B Data Direction Register */
#define GPIO_PDOR   (*((volatile unsigned int*)0x400FF040)) /* Port B Data Output Register */
#define GPIOB_PTOR  (*((volatile unsigned int*)0x400FF04C)) /* Port B Toggle Output Register */

#define GREEN_PIN     (19)
#define GREEN_SHIFT   (1<<GREEN_PIN)
#define GREEN_TOGGLE  (GPIOB_PTOR |= GREEN_SHIFT)
```

Definições dos registradores da Tarefa 03

Em seguida, vamos utilizar a função delayMs para adicionar um tempo de intervalo entre o LED ligando e desligando.

```
void delayMs(int n) {
    int i, j;
    for(i = 0 ; i < n; i++)
        for (j = 0; j < 7000; j++) {}
}
```

Definições dos registradores da Tarefa 03

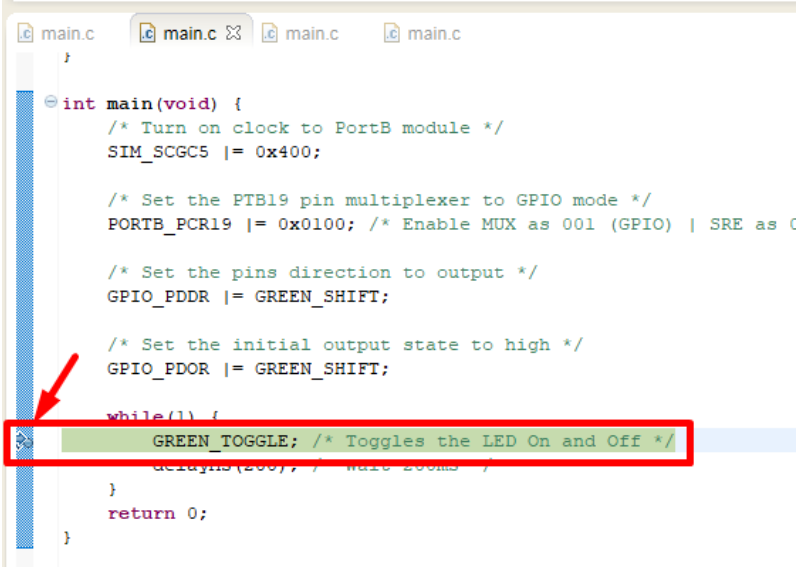
Após atribuir os endereços dos registradores, basta realizar as configurações de funcionamento de cada um deles, na ordem em que aparecem os comentários. A parte responsável por fazer o LED piscar, encontra-se no loop while entre as linhas 14 e 17.

```
1  int main(void) {
2      /* Turn on clock to PortB module */
3      SIM_SCGC5 |= 0x400;
4
5      /* Set the PTB19 pin multiplexer to GPIO mode */
6      PORTB_PCR19 |= 0x0100; /* Enable MUX as 001 (GPIO) | SRE as 0 (Fast) */
7
8      /* Set the pins direction to output */
9      GPIO_PDDR |= GREEN_SHIFT;
10
11     /* Set the initial output state to high */
12     GPIO_PDOR |= GREEN_SHIFT;
13
14     while(1) {
15         GREEN_TOGGLE; /* Toggles the LED On and Off */
16         delayMs(200); /* Wait 200ms */
17     }
18     return 0;
19 }
```

Configuração dos registradores e loop principal da Tarefa 03

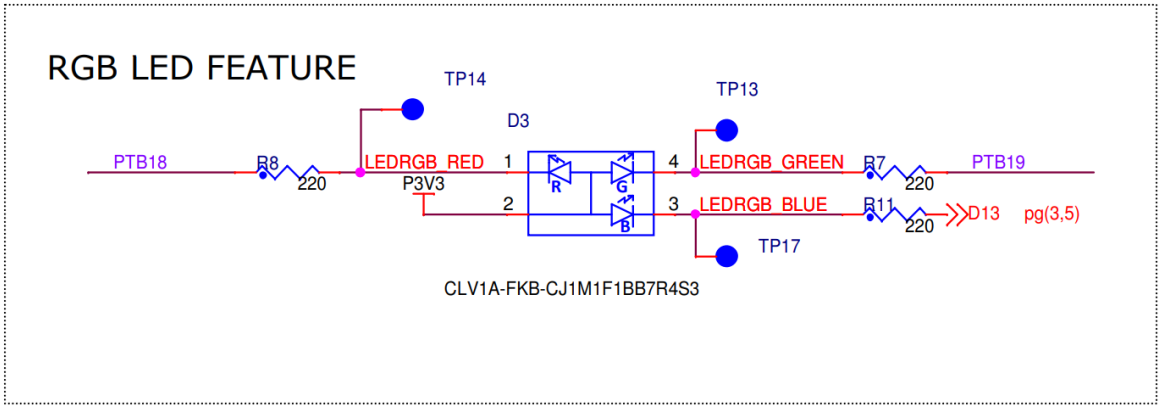
4) Analise a mudança dos registradores no debugger

Para ver a mudança nos registradores que faz o LED acender, precisamos adicionar um breakpoint no código dando dois clicks da linha desejada, como mostrado abaixo:

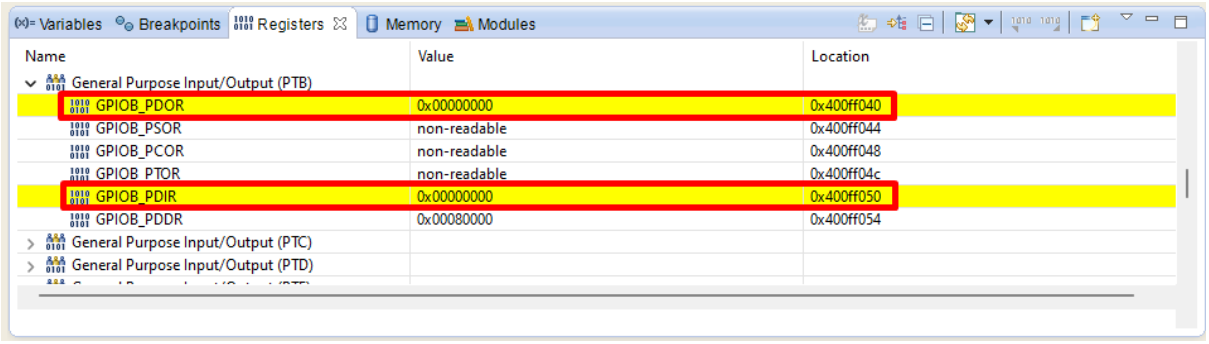


Breakpoint no debugger da Tarefa 04

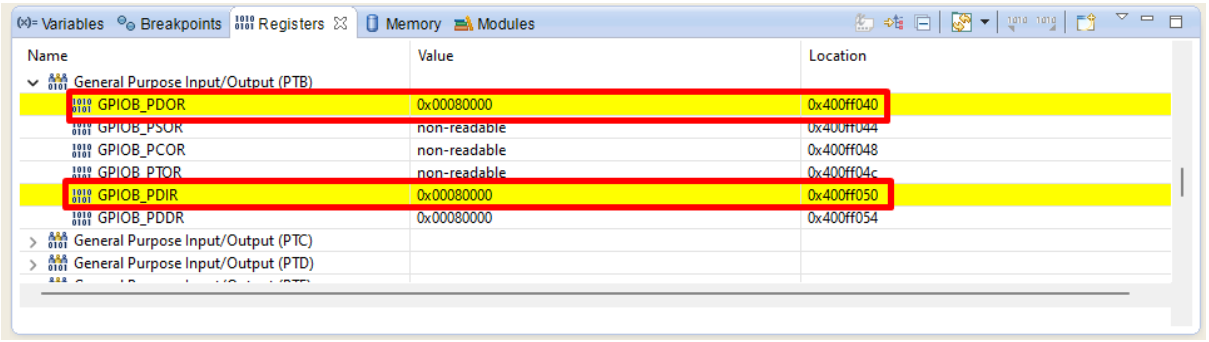
Quando o registrador PDOR está zerado, o LED acende, e quando o registrador PDOR está com o valor 0x00080000 o LED desliga. Este é o comportamento esperado, uma vez que pela configuração da placa, o LED acende em LOW e não em HIGH.



Esquemático LED na FRDM-KL25Z



Registrador GPIOB_PDOR com LED Ligado



Registrador GPIOB_PDOR com LED Desligado

5) Modifique o código para piscar dois LEDs ao mesmo tempo.

Para piscar dois LEDs ao mesmo tempo, precisamos adicionar o endereço do pino PTB19 ao código e configurá-lo de maneira semelhante ao procedimento feito anteriormente, como pode ser visto abaixo:

```
#define SIM_SCGC5   (*((volatile unsigned int*)0x40048038)) /* System Integration Module System Clock Gating Control Register 5 */
#define PORTB_PCR19 (*((volatile unsigned int*)0x4004A04C)) /* Port B Pin Control Register 19 (GREEN LED)*/
#define PORTB_PCR18 (*((volatile unsigned int*)0x4004A048)) /* Port B Pin Control Register 18 (RED LED) */
#define GPIO_PDDR   (*((volatile unsigned int*)0x400FF054)) /* Port B Data Direction Register */
#define GPIO_PDOR   (*((volatile unsigned int*)0x400FF040)) /* Port B Data Output Register */
#define GPIOB_PTOR  (*((volatile unsigned int*)0x400FF04C)) /* Port B Toggle Output Register */

#define GREEN_PIN     (19)
#define GREEN_SHIFT   (1<<GREEN_PIN)
#define GREEN_TOGGLE  (GPIOB_PTOR |= GREEN_SHIFT)

#define RED_PIN       (18)
#define RED_SHIFT     (1<<RED_PIN)
#define RED_TOGGLE    (GPIOB_PTOR |= RED_SHIFT)
```

Definições dos registradores da Tarefa 05

Criei dois loops, um deles (o que não está comentado) faz os dois LEDs piscarem alternados. nas linhas de 31 a 35 existe outro loop onde os dois LEDs piscam simultaneamente.

```
int main(void) {
    /* Turn on clock to PortB module */
    SIM_SCGC5 |= 0x400;

    /* Set the PTB17, PTB18 and PTB19 pin multiplexer to GPIO mode */
    PORTB_PCR19 |= 0x0100; // Enable MUX as 001 (GPIO) | SRE as 0 (Fast)
    PORTB_PCR18 |= 0x0100; // Enable MUX as 001 (GPIO) | SRE as 0 (Fast)

    /* Set the pins direction to output */
    GPIO_PDDR |= GREEN_SHIFT;
    GPIO_PDDR |= RED_SHIFT;

    /* Set the initial output state to high */
    GPIO_PDOR |= GREEN_SHIFT;
    GPIO_PDOR |= RED_SHIFT;

    /* Loop with alternate LEDs */
    GREEN_TOGGLE;
    while(1) {
        GREEN_TOGGLE;
        RED_TOGGLE;
        delayMs(200);

        GREEN_TOGGLE;
        RED_TOGGLE;
        delayMs(200);
    }

    /* Loop with LED GREEN and RED blinking together */
    /*
    while(1) {
        GREEN_TOGGLE;
        RED_TOGGLE;
        delayMs(200);
    }*/
    return 0;
}
```

Configuração dos registradores e loop principal da Tarefa 05

6) Compile o código abaixo e escreva os valores de next em cada iteração. Qual o nome desta série de números?

Antes de atualizar o valor de next na primeira iteração, seu valor apresenta algum “lixo” aleatório.

Print do debugger com as variáveis da Tarefa 06

Em seguida, os valores de next são os mostrados abaixo. Podemos ver que são os valores da sequência de Fibonacci.

c	next
0	**0**
1	**1**
2	**1**
3	**2**
4	**3**
5	**5**
6	**8**
7	**13**
8	**21**
9	**34**

Compare as vantagens e desvantagens de se utilizar um IDE profissional (comparando com o MBED).

Utilizar uma IDE profissional ao invés do MBED é bastante vantajoso, uma vez que temos mais recursos como por exemplo: um debugger onde é possível ver os valores de cada um dos registradores do nosso microcontrolador individualmente. A vantagem de utilizar bibliotecas que facilitam na criação de códigos. A facilidade para lidar com vários projetos ao mesmo tempo. Além de outras funcionalidades que ainda vamos aprender ao longo do curso!

Segue o link com os códigos criados:
https://github.com/kevinkirsten/psi3441-arquitetura-de-sistemas-embarcados/tree/main/exercicio_pratico_02