PSI3441 - Arquitetura de Sistemas Embarcados

Relatório Exercício Prático 04

Nome: Kevin Kirsten Lucas nºUSP: 10853306

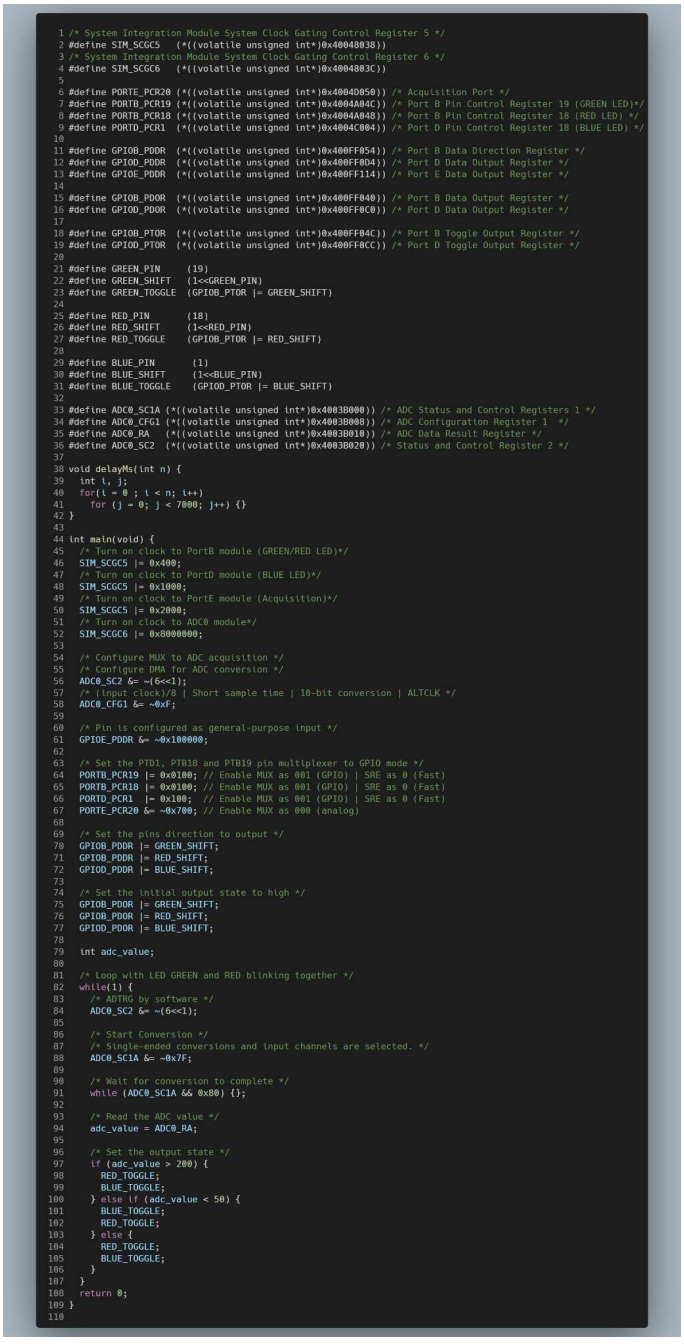
Escrever o código, sem utilização do Processo Expert, para:

1) Fazer uma aquisição analógica;

2) Acender o LED azul quando o valor for próximo de 3.3 V e o LED verde quando o valor for próximo de 0 V.

Para os dois primeiros itens, utilizei como base o código do exercício prático da aula 2, onde já eram realizadas as operações necessárias para ligar os LEDs Verde e Vermelho.

Para trabalhar com o LED Azul e utilizar uma das portas do kit KL25Z , foi necessário realizar algumas modificações. O código segue abaixo:



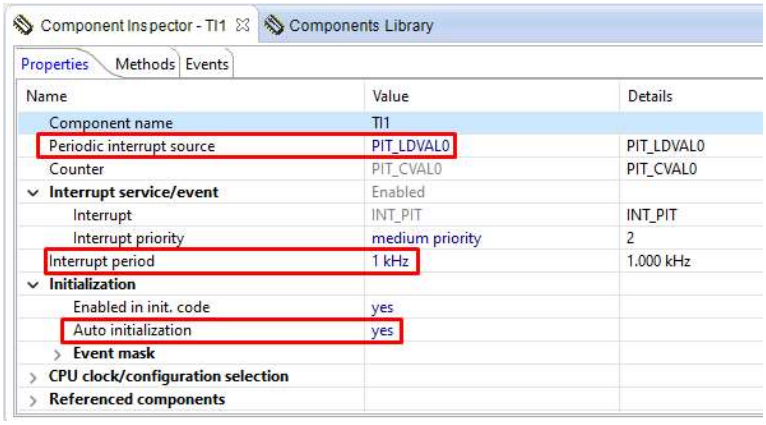
```c
/* System Integration Module System Clock Gating Control Register 5 */
#define SIM_SCGC5   (*((volatile unsigned int*)0x40048038))
/* System Integration Module System Clock Gating Control Register 6 */
#define SIM_SCGC6   (*((volatile unsigned int*)0x4004803C))

#define PORTE_PCR20 (*((volatile unsigned int*)0x4004D058)) /* Acquisition Port */
#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 PORTD_PCR1  (*((volatile unsigned int*)0x4004C004)) /* Port D Pin Control Register 18 (BLUE LED) */

#define GPIOB_PDDR  (*((volatile unsigned int*)0x400FF054)) /* Port B Data Direction Register */
#define GPIOD_PDDR  (*((volatile unsigned int*)0x400FF0D4)) /* Port D Data Output Register */
#define GPIOE_PDDR  (*((volatile unsigned int*)0x400FF114)) /* Port E Data Output Register */

#define GPIOB_PDOR  (*((volatile unsigned int*)0x400FF040)) /* Port B Data Output Register */
#define GPIOD_PDOR  (*((volatile unsigned int*)0x400FF0C0)) /* Port D Data Output Register */

#define GPIOB_PTOR  (*((volatile unsigned int*)0x400FF04C)) /* Port B Toggle Output Register */
#define GPIOD_PTOR  (*((volatile unsigned int*)0x400FF0CC)) /* Port D 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)

#define BLUE_PIN      (1)
#define BLUE_SHIFT    (1<<BLUE_PIN)
#define BLUE_TOGGLE   (GPIOD_PTOR |= BLUE_SHIFT)

#define ADC0_SC1A (*((volatile unsigned int*)0x4003B000)) /* ADC Status and Control Registers 1 */
#define ADC0_CFG1 (*((volatile unsigned int*)0x4003B008)) /* ADC Configuration Register 1  */
#define ADC0_RA   (*((volatile unsigned int*)0x4003B010)) /* ADC Data Result Register */
#define ADC0_SC2  (*((volatile unsigned int*)0x4003B020)) /* Status and Control Register 2 */

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

int main(void) {
 /* Turn on clock to PortB module (GREEN/RED LED)*/
 SIM_SCGC5 |= 0x400;
 /* Turn on clock to PortD module (BLUE LED)*/
 SIM_SCGC5 |= 0x1000;
 /* Turn on clock to PortE module (Acquisition)*/
 SIM_SCGC5 |= 0x2000;
 /* Turn on clock to ADC0 module*/
 SIM_SCGC6 |= 0x8000000;

 /* Configure MUX to ADC acquisition */
 /* Configure DMA for ADC conversion */
 ADC0_SC2 &= ~(6<<1);
 /* (Input clock)/8 | Short sample time | 10-bit conversion | ALTCLK */
 ADC0_CFG1 &= ~0xF;

 /* Pin is configured as general-purpose input */
 GPIOE_PDDR &= ~0x100000;

 /* Set the PTD1, 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)
 PORTD_PCR1  |= 0x100;  // Enable MUX as 001 (GPIO) | SRE as 0 (Fast)
 PORTE_PCR20 &= ~0x700; // Enable MUX as 000 (analog)

 /* Set the pins direction to output */
 GPIOB_PDDR |= GREEN_SHIFT;
 GPIOB_PDDR |= RED_SHIFT;
 GPIOD_PDDR |= BLUE_SHIFT;

 /* Set the inittial output state to high */
 GPIOB_PDOR |= GREEN_SHIFT;
 GPIOB_PDOR |= RED_SHIFT;
 GPIOD_PDOR |= BLUE_SHIFT;

 int adc_value;

 /* Loop with LED GREEN and RED blinking together */
 while(1) {
   /* ADTRG by software */
   ADC0_SC2 &= ~(6<<1);

   /* Start Conversion */
   /* Single-ended conversions and input channels are selected. */
   ADC0_SC1A &= ~0x7F;

   /* Wait for conversion to complete */
   while (ADC0_SC1A && 0x80) {};

   /* Read the ADC value */
   adc_value = ADC0_RA;

   /* Set the output state */
   if (adc_value > 200) {
     RED_TOGGLE;
     BLUE_TOGGLE;
   } else if (adc_value < 50) {
     BLUE_TOGGLE;
     RED_TOGGLE;
   } else {
     RED_TOGGLE;
     BLUE_TOGGLE;
   }
 }
 return 0;
}
```

Código das Tarefas 1 e 2

3) Utilizar um timer periódico para, por interrupção, disparar a conversão AD. Usar a interrupção de fim de conversão para acender os LEDs. Permitido o uso do Processor Expert para este item Obs. Utilizar o KL25 Sub-Family Reference Manual para detalhes dos registradores (capítulo 28).

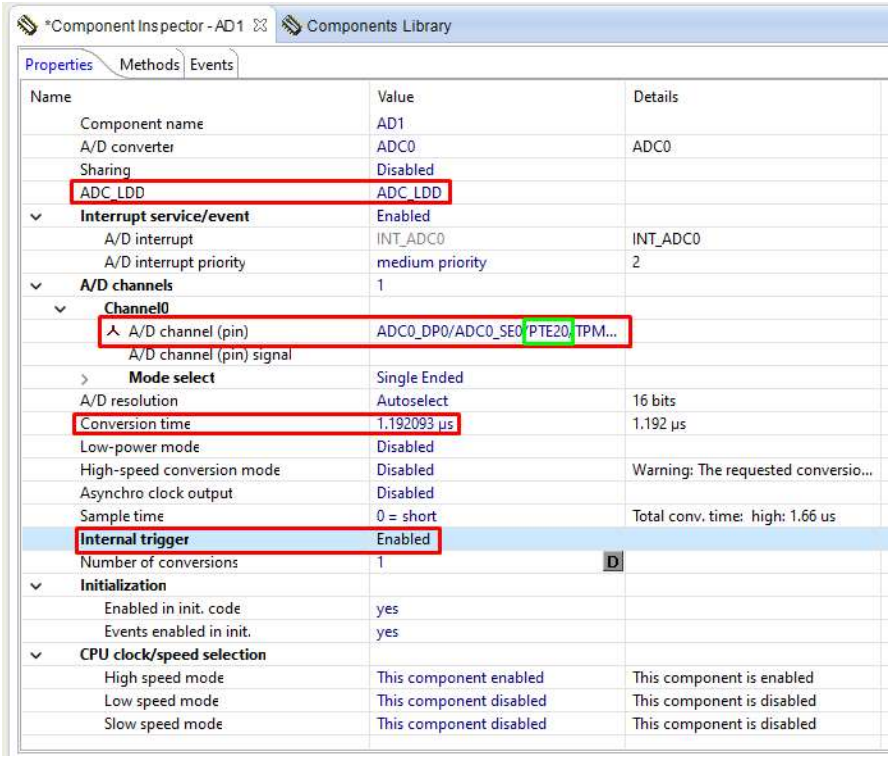
Para configurar o KL25Z com um timer periódico, podemos utilizar o componente `TimerInt_LDD`, capaz de gerar uma interrupção periódica com qualquer tipo de timer.

Estando com a opção advanced habilitada, podemos configurá-lo com o timer do tipo PIT, e com período de interrupção de 1 kHz, como mostrado na figura abaixo:



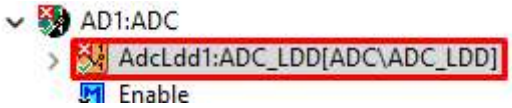
Configuração do Timer PIT com Processor Expert

Em seguida, vamos configurar o componente ADC como mostrado abaixo:

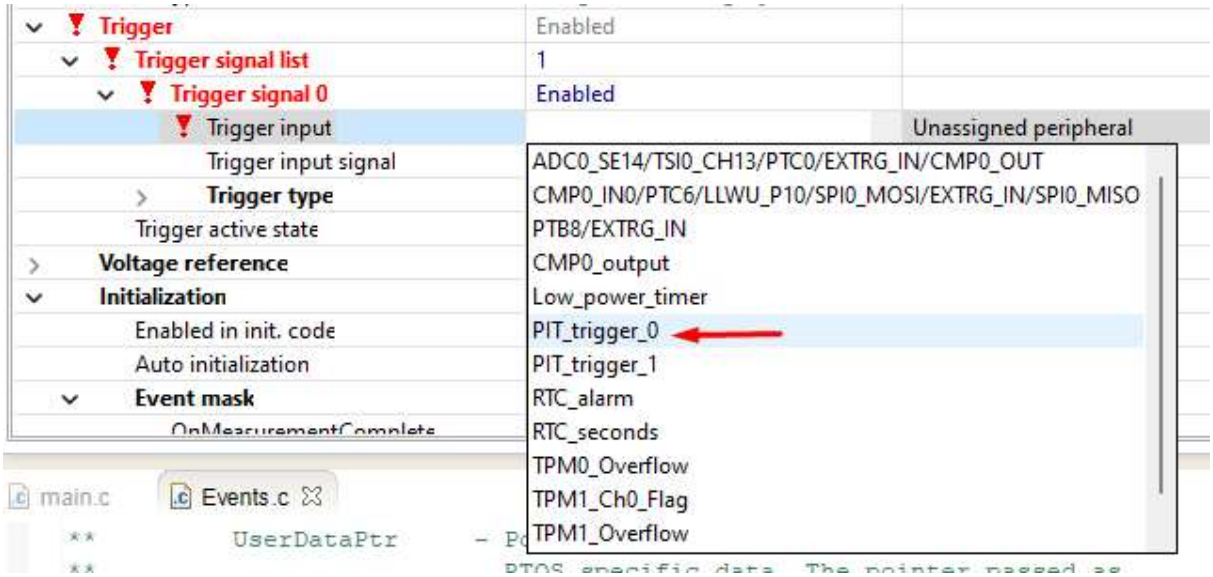


Configuração inicial do ADC

No componente indicado abaixo, temos mais um nível de vonfiguração, onde vamos definir qual será o input do trigger do nosso ADC. Neste caso, vamos utilizar o componente criado anteriormente:

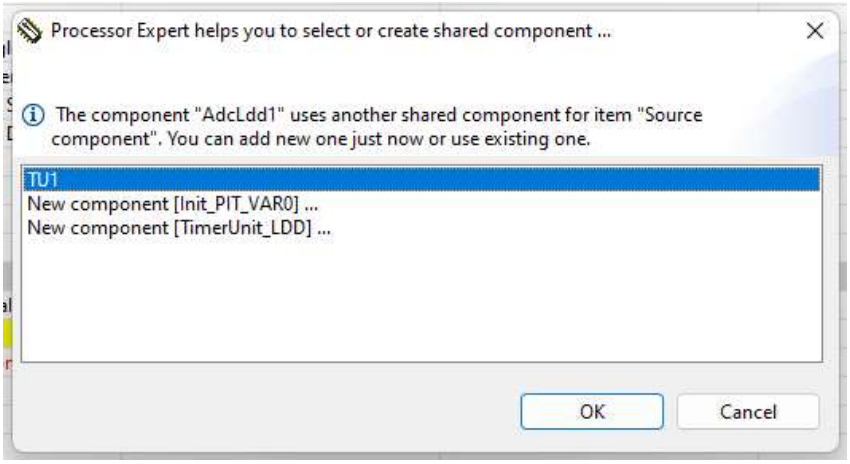


Componente com configurações extras do trigger do ADC



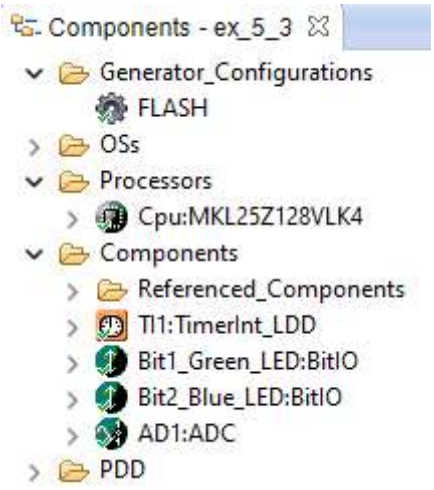
Seleção do trigger PIT

Podemos escolher o componente criado anteriormente:



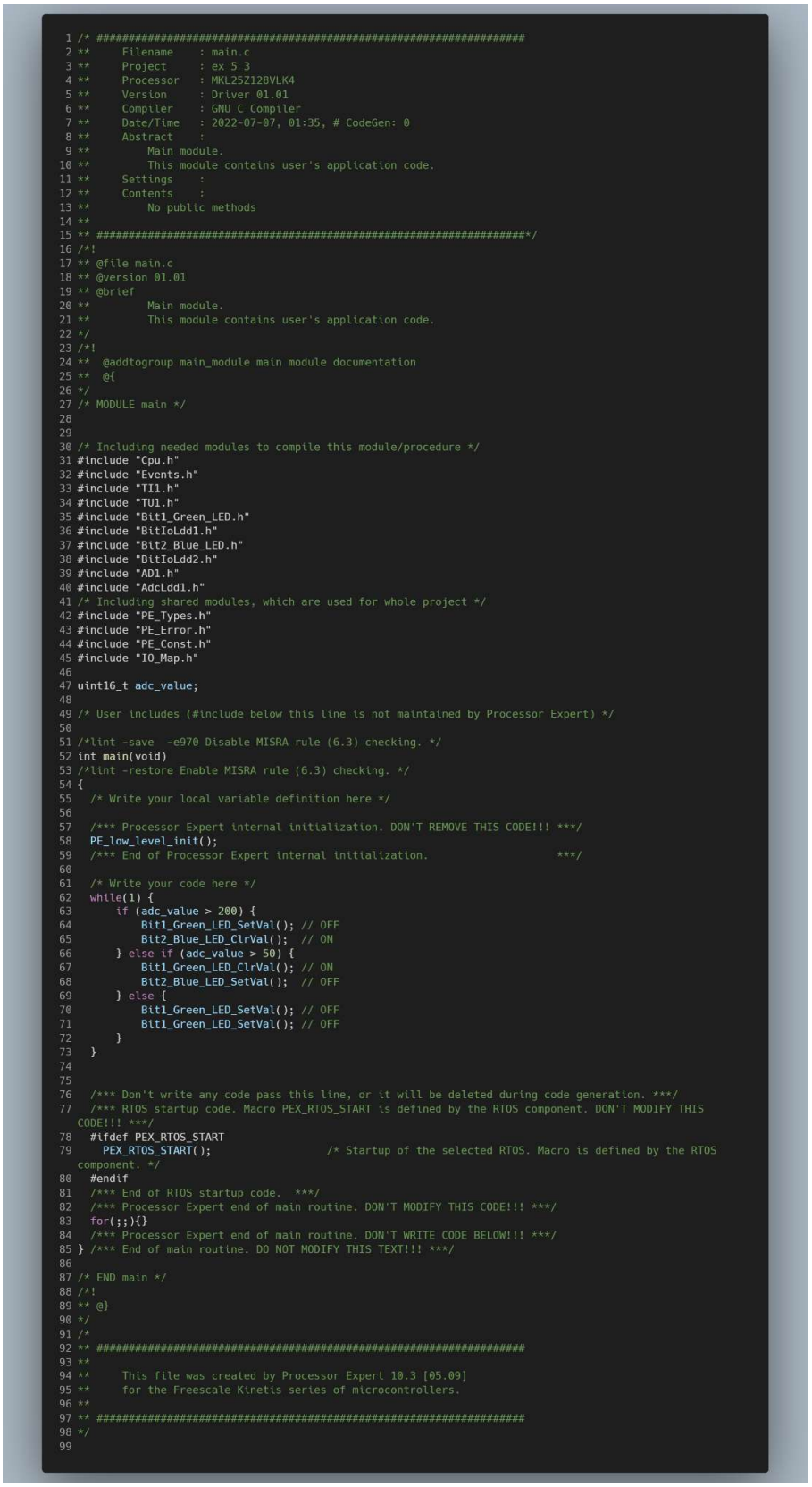
Seleção do componente TU1 do trigger criado anteriormente

Finalmente, configuramos os LEDs Verde e Azul com o componente BitIO.
A árvore final de componentes, está mostrada a seguir:



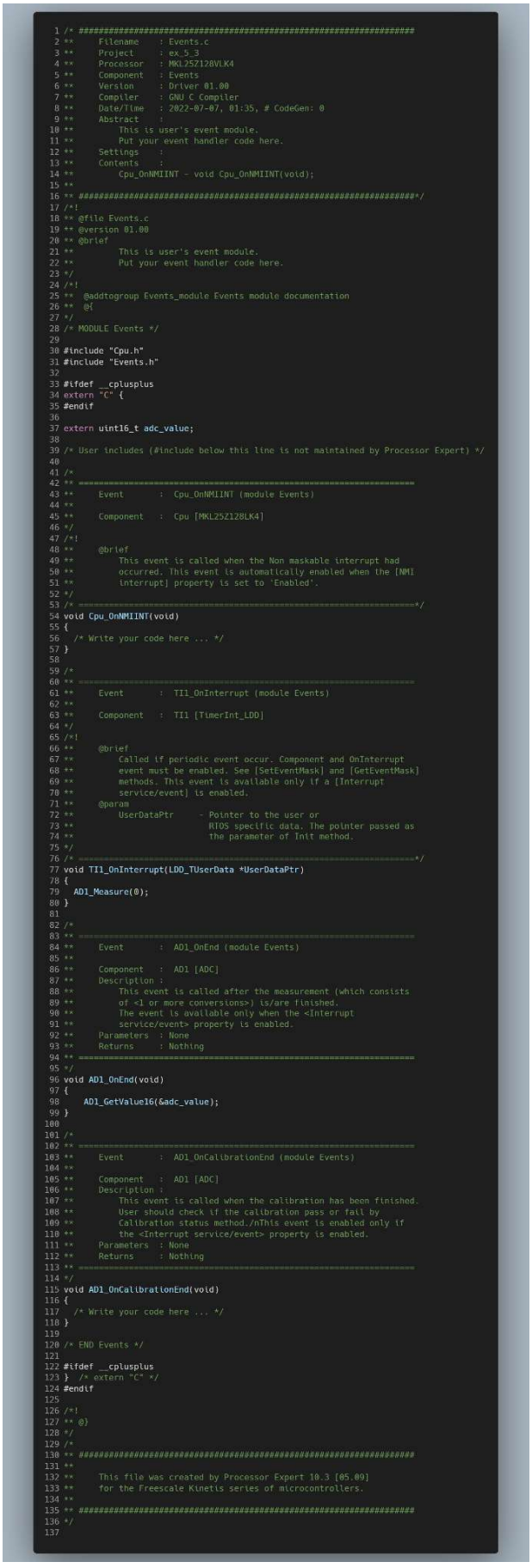
Árvore de componentes

Agora basta configurar os arquivos `main.c` e `events.h` para realizar a conversão e mudar os LEDs. Os códigos ficaram como mostrado a seguir:



```c
/* ###################################################################
**     Filename    : main.c
**     Project     : ex_5_3
**     Processor   : MKL25Z128VLK4
**     Version     : Driver 01.01
**     Compiler    : GNU C Compiler
**     Date/Time   : 2022-07-07, 01:35, # CodeGen: 0
**     Abstract    :
**         Main module.
**         This module contains user's application code.
**     Settings    :
**     Contents    :
**         No public methods
**
** ###################################################################*/
/*!
** @file main.c
** @version 01.01
** @brief
**         Main module.
**         This module contains user's application code.
*/
/*!
**  @addtogroup main_module main module documentation
**  @{
*/
/* MODULE main */


/* Including needed modules to compile this module/procedure */
#include "Cpu.h"
#include "Events.h"
#include "TI1.h"
#include "TU1.h"
#include "Bit1_Green_LED.h"
#include "BitIoLdd1.h"
#include "Bit2_Blue_LED.h"
#include "BitIoLdd2.h"
#include "AD1.h"
#include "AdcLdd1.h"
/* Including shared modules, which are used for whole project */
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"

uint16_t adc_value;

/* User includes (#include below this line is not maintained by Processor Expert) */

/*lint -save  -e970 Disable MISRA rule (6.3) checking. */
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */
{
  /* Write your local variable definition here */

  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/

  /* Write your code here */
  while(1) {
      if (adc_value > 200) {
          Bit1_Green_LED_SetVal(); // OFF
          Bit2_Blue_LED_ClrVal();  // ON
      } else if (adc_value > 50) {
          Bit1_Green_LED_ClrVal(); // ON
          Bit2_Blue_LED_SetVal();  // OFF
      } else {
          Bit1_Green_LED_SetVal(); // OFF
          Bit1_Green_LED_SetVal(); // OFF
      }
  }


  /*** Don't write any code pass this line, or it will be deleted during code generation. ***/
  /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component. DON'T MODIFY THIS CODE!!! ***/
  #ifdef PEX_RTOS_START
    PEX_RTOS_START();                  /* Startup of the selected RTOS. Macro is defined by the RTOS component. */
  #endif
  /*** End of RTOS startup code.  ***/
  /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
  for(;;){}
  /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
} /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/

/* END main */
/*!
** @}
*/
/*
** ###################################################################
**
**     This file was created by Processor Expert 10.3 [05.09]
**     for the Freescale Kinetis series of microcontrollers.
**
** ###################################################################
*/
```

Código do arquivo main.c



```c
/* ###################################################################
**     Filename    : Events.c
**     Project     : ex_5_3
**     Processor   : MKL25Z128VLK4
**     Component   : Events
**     Version     : Driver 01.00
**     Compiler    : GNU C Compiler
**     Date/Time   : 2022-07-07, 01:35, # CodeGen: 0
**     Abstract    :
**         This is user's event module.
**         Put your event handler code here.
**     Settings    :
**     Contents    :
**         Cpu_OnNMIINT - void Cpu_OnNMIINT(void);
**
** ###################################################################*/
/*!
** @file Events.c
** @version 01.00
** @brief
**         This is user's event module.
**         Put your event handler code here.
*/
/*!
**  @addtogroup Events_module Events module documentation
**  @{
*/
/* MODULE Events */

#include "Cpu.h"
#include "Events.h"

#ifdef __cplusplus
extern "C" {
#endif

extern uint16_t adc_value;

/* User includes (#include below this line is not maintained by Processor Expert) */

/*
** ===================================================================
**     Event       :  Cpu_OnNMIINT (module Events)
**
**     Component   :  Cpu [MKL25Z128LK4]
*/
/*!
**     @brief
**         This event is called when the Non maskable interrupt had
**         occurred. This event is automatically enabled when the [NMI
**         interrupt] property is set to 'Enabled'.
*/
/* ===================================================================*/
void Cpu_OnNMIINT(void)
{
  /* Write your code here ... */
}

/*
** ===================================================================
**     Event       :  TI1_OnInterrupt (module Events)
**
**     Component   :  TI1 [TimerInt_LDD]
*/
/*!
**     @brief
**         Called if periodic event occur. Component and OnInterrupt
**         event must be enabled. See [SetEventMask] and [GetEventMask]
**         methods. This event is available only if a [Interrupt
**         service/event] is enabled.
**     @param
**         UserDataPtr     - Pointer to the user or
**                           RTOS specific data. The pointer passed as
**                           the parameter of Init method.
*/
/* ===================================================================*/
void TI1_OnInterrupt(LDD_TUserData *UserDataPtr)
{
  AD1_Measure(0);
}

/*
** ===================================================================
**     Event       :  AD1_OnEnd (module Events)
**
**     Component   :  AD1 [ADC]
**     Description :
**         This event is called after the measurement (which consists
**         of <1 or more conversions>) is/are finished.
**         The event is available only when the <Interrupt
**         service/event> property is enabled.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void AD1_OnEnd(void)
{
    AD1_GetValue16(&adc_value);
}

/*
** ===================================================================
**     Event       :  AD1_OnCalibrationEnd (module Events)
**
**     Component   :  AD1 [ADC]
**     Description :
**         This event is called when the calibration has been finished.
**         User should check if the calibration pass or fail by
**         Calibration status method./nThis event is enabled only if
**         the <Interrupt service/event> property is enabled.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void AD1_OnCalibrationEnd(void)
{
  /* Write your code here ... */
}

/* END Events */

#ifdef __cplusplus
}  /* extern "C" */
#endif

/*!
** @}
*/
/*
** ###################################################################
**
**     This file was created by Processor Expert 10.3 [05.09]
**     for the Freescale Kinetis series of microcontrollers.
**
** ###################################################################
*/
```

Código do arquivo events.h

Link com o código dos arquivos do projeto:
https://github.com/kevinkirsten/psi3441-arquitetura-de-sistemas-embarcados/tree/main/exercicio_pratico_05