```c
/* ************************************************************** */
/* File name:       adc.c                                         */
/* File description: This file has a couple of useful functions to   */
/*                  control the ADC from the peripheral board.      */
/*                  The converter is connected to the Temperature   */
/*                  sensor.                                       */
/* Author name:      dloubach, julioalvesMS, lagoAF e rbacurau       */
/* Creation date:    07jun2018                                     */
/* Revision date:    20mai2020                                     */
/* ************************************************************** */

#include "board.h"
#include "adc.h"

#define ADC0_SC1A_COCO (ADC0_SC1A >> 7)
#define ADC0_SC2_ADACT (ADC0_SC2 >> 7)

#define ADC_CFG1_BUS_CLK_2   01U
#define ADC_CFG1_CONVERSION  00U
#define ADC_CFG1_SAMPLE_TIME  0U
#define ADC_CFG1_CLK_DIVIDER 00U
#define ADC_CFG1_LOW_POWER    0U

#define ADC_SC2_VOLT_REF    00U
#define ADC_SC2_DMA          0U
#define ADC_SC2_COMPARE      0U
#define ADC_SC2_TRIGGER_CONV 0U

#define ADC_CFG2_LONG_SAMPLE 00U
#define ADC_CFG2_HIGH_SPEED   0U
#define ADC_CFG2_ASYNC_CLK    0U
#define ADC_CFG2_MUX_SELECT   0U

#define ADC_SC1A_COMPLETE     4U
#define ADC_SC1A_INTERRUPT    0U
#define ADC_SC1A_DIFFERENTIAL 0U

#define CGC_CLOCK_ENABLED 1 //ASSUMINDO QUE SEJA 1 OU 0 (NAO TINHA NOS DEFINES).
      //pra arrumar (se n funcionar) pagina 206 do KL25 Sub-Family Reference Manual

/* ************************************************** */
/* Method name:       adc_initADCModule               */
/* Method description: Init a the ADC converter device */
/* Input params:      n/a                             */
/* Output params:     n/a                             */
/* ************************************************** */
void adc_initADCModule(void)
{
  /* un-gate port clock*/
  SIM_SCGC6 |= SIM_SCGC6_ADC0(CGC_CLOCK_ENABLED);    //Enable clock for ADC

  /* un-gate port clock*/
  SIM_SCGC5 |= SIM_SCGC5_PORTE(CGC_CLOCK_ENABLED);

  /* set pin as ADC In */
  PORTE_PCR21 |= PORT_PCR_MUX(THERMOMETER_ALT);   //Temperature Sensor

  /*
  ADC_CFG1_ADICLK(x)// bus/2 clock selection
  ADC_CFG1_MODE(x)  // 8-bit Conversion mode selection
  ADC_CFG1_ADLSMP(x)// Short sample time configuration
  ADC_CFG1_ADIV(x)  // Clock Divide Select (Divide by 1)
  ADC_CFG1_ADLPC(x) // Normal power Configuration
  */
  ADC0_CFG1 |= (ADC_CFG1_ADICLK(ADC_CFG1_BUS_CLK_2) | ADC_CFG1_MODE(ADC_CFG1_CONVERSION) | ADC_CFG1_ADLSMP(ADC_CFG1_SAMPLE_TIME) | ADC_CFG1_ADIV(ADC_C

  /*
  ADC_SC2_REFSEL(x)// reference voltage selection - external pins
  ADC_SC2_DMAEN(x) // dma disabled
  ADC_SC2_ACREN(x) // dont care - range function
  ADC_SC2_ACFGT(x) // dont care - 0 -> Less than, 1 -> Greater Than
  ADC_SC2_ACFE(x)  // compare function disabled
  ADC_SC2_ADTRG(x) // When software trigger is selected, a conversion is initiated following a write to SC1A
  ADC_SC2_ADACT(x) // HW-set indicates if a conversion is being held, is cleared when conversion is done
  */
  ADC0_SC2 |= (ADC_SC2_REFSEL(ADC_SC2_VOLT_REF) | ADC_SC2_DMAEN(ADC_SC2_DMA) | ADC_SC2_ACFE(ADC_SC2_COMPARE) | ADC_SC2_ADTRG(ADC_SC2_TRIGGER_CONV));

  /*
  ADC_CFG2_ADLSTS(x)  // default time
  ADC_CFG2_ADHSC(x)   // normal conversion sequence
  DC_CFG2_ADACKEN(x) // disable adack clock
  ADC_CFG2_MUXSEL(x)  // select 'a' channels
  */
  ADC0_CFG2 |= (ADC_CFG2_ADLSTS(ADC_CFG2_LONG_SAMPLE) | ADC_CFG2_ADHSC(ADC_CFG2_HIGH_SPEED) | ADC_CFG2_ADACKEN(ADC_CFG2_ASYNC_CLK) | ADC_CFG2_MUXSEL
}


/* ************************************************** */
/* Method name:       adc_initConvertion              */
/* Method description: init a conversion from A to D  */
/* Input params:      n/a                             */
/* Output params:     n/a                             */
/* ************************************************** */
void adc_initConvertion(void)
{
   /*
   ADC_SC1_COCO(x) // conversion complete flag HW-set
   ADC_SC1_AIEN(x) // conversion complete interrupt disables
   ADC_SC1_DIFF(x) // selects single-ended convertion
   ADC_SC1_ADCH(x) // selects channel, view 3.7.1.3.1 ADC0 Channel Assignment ADC0_SE4a from datasheet
   */
 ADC0_SC1A &= (ADC_SC1_ADCH(ADC_SC1A_COMPLETE) | ADC_SC1_DIFF(ADC_SC1A_DIFFERENTIAL) | ADC_SC1_AIEN(ADC_SC1A_INTERRUPT));
}

/* ************************************************** */
/* Method name:       adc_isAdcDone                   */
/* Method description: check if conversion is done    */
```

```
/* Method description: check if conversion is done     */
/* Input params:       n/a                             */
/* Output params:      char: 1 if Done, else 0         */
/* ************************************************** */
char adc_isAdcDone(void)
{
   if(ADC0_SC1A_COCO) // watch complete convertion flag
      return 1; // if the conversion is complete, return 1
   else
      return 0; // if the conversion is still taking place, return 0
}

/* ************************************************** */
/* Method name:        adc_getConvertionValue         */
/* Method description: Retrieve converted value       */
/* Input params:       n/a                            */
/* Output params:      int: Result from convertion    */
/* ************************************************** */
int adc_getConvertionValue(void)
{
   return ADC0_RA; // return the register value that keeps the result of convertion
}
```