```
/**
 * app_main.c
 *
 *  @date Created at:    13/07/2021 14:09:30
 *    @author:    Pedro Igor B. S. / modificado por Giovani Fonseca
 *    @email:        pibscontato@gmail.com
 *     GitHub:        https://github.com/pedro-ibs
 *     tabSize:    8
 *
 * ########################################################
 *   Copyright (C) Pedro Igor B. S 2021
 * ---------------------------------------------------------------------
 *
 *   Licença: GNU GPL 2
 * ---------------------------------------------------------------------
 *   This program is free software; you can redistribute it and/or
 *   modify it under the terms of the GNU General Public License as
 *   published by the Free Software Foundation; version 2 of the
 *   License.
 *
 *   This program is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *   GNU General Public License for more details.
 * ---------------------------------------------------------------------
 * ########################################################
 *
 * Os pinos e serial usados estão definidos em hardware.h
 * Demais configurações estão em consfig.h
 *
 * funcionameno:
 *
 * * Inicia o freeRTOS.
 *     Inicia os drives e configura o MCU isso ocorre em main.c
 *     Configurações avançadas do sietema encontram se em:
 *         FreeRTOS/include/FreeRTOSConfig.h
 *
 *
 * * Inicia o setup da aplicação
 *     Antes do loop é enviado ascolunas do csv.
 *
 * * Loop:
 *     O Semaforo é utilizado para controle de acesso do buffer entre o
 *     loop app_main e o conjundo de aquisição (sendo eles o ADC1, DMA e
 *     TIM3) alem disso detemina o fluxo de operação, como se fosse uma
 *     flag.
 *
 *     Quando o semaforo é devolvido pelo conjundo de aquisição o loop
 *     do app_main pega o acesso e trasmite os dados, e imediatamente após
 *     o logo apos devolver o semaforo o conjunto de aquisição pega o semaforo
 *     e começa a coleta de dados e devolve o semaforo e o loop se inicia.
 *
 *     NOTA: Enquanto o conjunto de aquisição estiver om o semaforo o loop
 *     ficará travado.
 *
 */


/* Includes ------------------------------------------------------------------------------------------------------------------------------------------------------*/
#include <config.h>
#include <core.h>
#include <FreeRTOS/include/FreeRTOSConfig.h>
#include <FreeRTOS/includes.h>

/* Private macro -------------------------------------------------------------------------------------------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------------------------------------------------------------------------------------------*/
static const AdcChannel psxAdc[]     = {CH1, CH2, CH3};    // Canais que serão lidos
static u16 spuBuffer[SAMPLE_SIZE]    = { 0 };        // Buffer de aquisição
static SemaphoreHandle_t spxSemaphore    = NULL;            // Samaforo usado como MUTEX e flag
static u8 suIdx             = 0;            // Indice do canal adc


/* Private Functions ---------------------------------------------------------------------------------------------------------------------------------------------*/
void app_vSatartGetSample(const AdcChannel cxChannel);
void app_vStopGetSample(const AdcChannel cxChannel, BaseType_t *const pxHigherPriorityTaskWoken);
void app_vWaitComand(const char * cpcCommand);
/* Functions -----------------------------------------------------------------------------------------------------------------------------------------------------*/


void main_vApp(void * pvParameters){
    (void) pvParameters;

    if ( spxSemaphore == NULL){
        spxSemaphore = xSemaphoreCreateMutex();
    }

    // usart_vSetup(STDIO,    usart_115k2bps);
    usart_vSetup(STDIO,    usart_500k0bps);

    gpio_vMode(LED, GPIO_MODE_OUTPUT_OD, GPIO_NOPULL);

    for (u16 uIdx = 0; uIdx < SAMPLE_SIZE; uIdx++) {
        spuBuffer[uIdx] = 0;
    }


    vTaskDelay(_3S);

    u8 uSample = 0;

    while (TRUE) {

        /**
         * Iniciar a o programa pele primeira vez
         *
         * envie o comando _START_ para inicar
```

```
108          */
109         if(uSample == 0){
110             uSample = 1;
111 
112             usart_vSendStr(STDIO, "Wait command to satart aquisition\n");
113 
114             app_vWaitComand(CMD_START);
115 
116             usart_vSendStr(STDIO, SIGINAL_COLUMNS);
117         }
118 
119 
120         /**
121          * trocar o canal ADC apos atingir  SAMPLES_MAX. Ao completar
122          * o ciclo é preciso enviar o comando _START_ novamente
123          */
124         if( uSample > SAMPLES_MAX) {
125             uSample = 1;
126             suIdx++;
127             if(suIdx >= 3) suIdx = 0;
128 
129             usart_vSendStr(STDIO, SIGINAL_FINISH);
130 
131             app_vWaitComand(CMD_START);
132 
133             usart_vSendStr(STDIO, SIGINAL_COLUMNS);
134         }
135 
136         app_vSatartGetSample(psxAdc[suIdx]);
137 
138         /**
139          * Para que a trasmissão prossiga o DMA tem que devolver
140          * o semaforo
141          */
142         if(xSemaphoreTake(spxSemaphore, portMAX_DELAY) == pdPASS){
143             char puSwap[20];
144 
145 
146             /**
147              * mostrar valores lidos no formato CSV
148              */
149             for (u16 uIdx = 0; uIdx < SAMPLE_SIZE; uIdx++) {
150                 usart_vSendStr(STDIO, itoa(suIdx, puSwap, HEX));
151                 usart_vSendChr(STDIO, ',');
152                 usart_vSendStr(STDIO, itoa(uSample, puSwap, HEX));
153                 usart_vSendChr(STDIO, ',');
154                 usart_vSendStr(STDIO, itoa(spuBuffer[uIdx], puSwap, HEX));
155                 usart_vSendChr(STDIO, '\n');
156                 spuBuffer[uIdx] = 0;
157             }
158 
159             uSample++;
160 
161             xSemaphoreGive(spxSemaphore);
162         }
163     }
164 }
165 
166 
167 
168 /*##########################################################################################################################################################################*/
169 /*##########################################################################################################################################################################*/
170 /*##########################################################################################################################################################################*/
171 /*------------------------------------------------------------------ Private Functions -----------------------------------------------------------------------------*/
172 /*##########################################################################################################################################################################*/
173 
174 
175 void app_vWaitComand(const char * cpcCommand){
176     while (TRUE) {
177         if (usart_iSizeBuffer(STDIO) > 1){
178             /**
179              *  esperar receber todos bytes
180              */
181             vTaskDelay(_200MS);
182 
183             if ( textp_bFindString(usart_pcGetBuffer(STDIO), cpcCommand) ){
184                 /**
185                  * apenas inicia a aquisição por dma caso não
186                  * esteja travada
187                  */
188                 break;
189             }
190 
191         }
192         vTaskDelay(_200MS);
193     }
194 
195     usart_vCleanBuffer(STDIO);
196 }
197 
198 void app_vSatartGetSample(const AdcChannel cxChannel){
199     if(xSemaphoreTake(spxSemaphore, portMAX_DELAY) == pdPASS){
200         adc1_vSetGetSampleMode(1);
201         adc1_vSetChannel(cxChannel, ADC_SAMPLETIME_1CYCLE_5,  ADC_REGULAR_RANK_1);
202         adc1_vStartGetSampleMode( spuBuffer, SAMPLE_SIZE, 1, FREG_TO_COUNTER(FREQUENCY_HZ, 1) );
203         gpio_vWrite(LED, TRUE);
204     }
205 }
206 
207 void app_vStopGetSample(const AdcChannel cxChannel, BaseType_t *const pxHigherPriorityTaskWoken){
208     adc1_vDeInitChannel(cxChannel);
209     adc1_vDeInitFromISR(pxHigherPriorityTaskWoken);
210     xSemaphoreGiveFromISR(spxSemaphore, pxHigherPriorityTaskWoken);
211     gpio_vWrite(LED, FALSE);
212 }
213 
214 
215 
```

```
216 void tim3_vHandler(BaseType_t *const pxHigherPriorityTaskWoken){
217 (void) pxHigherPriorityTaskWoken;
218 }
219
220 void adc1_vDMA1Ch1Handler(BaseType_t *const pxHigherPriorityTaskWoken){
221     (void) pxHigherPriorityTaskWoken;
222 }
223
224 void acd1_vBufferDoneHandler(BaseType_t *const pxHigherPriorityTaskWoken){
225     app_vStopGetSample(psxAdc[suIdx], pxHigherPriorityTaskWoken);
226 }
```