```
#include "STM32F4xx.h"
#include <rtl.h>
#include <string.h>
//#include "pdm_filter.h"


uint16_t out1 ;

/* Exported macros -----------------------------------------------------------*/
#define HTONS(A)   ((((uint16_t)(A) & 0xff00) >> 8) | \
                    (((uint16_t)(A) & 0x00ff) << 8))

void gpio_config(void){
RCC->AHB1ENR  |= ((1UL <<  4) );


GPIOE->MODER    |=  ((1UL << 2*12) |  // 4x LEDY out
                     (1UL << 2*13) |
                     (1UL << 2*14) |
                     (1UL << 2*15)  );

GPIOE->OTYPER   &= ~((1UL <<   12) |
                     (1UL <<   13) |
                     (1UL <<   14) |
                     (1UL <<   15)  );   // Push - Pull
GPIOE->OSPEEDR  &= ~((3UL << 2*12) |
                     (3UL << 2*13) |
                     (3UL << 2*14) |
                     (3UL << 2*15)  );
GPIOE->OSPEEDR  |=  ((2UL << 2*12) |  // 50 Mhz out
                     (2UL << 2*13) |
                     (2UL << 2*14) |
                     (2UL << 2*15)  );
GPIOE->PUPDR    &= ~((3UL << 2*12) |
                     (3UL << 2*13) |
                     (3UL << 2*14) |
                     (3UL << 2*15)  );
GPIOE->PUPDR    |=  ((1UL << 2*12) |  // Pull up
                     (1UL << 2*13) |
                     (1UL << 2*14) |
                     (1UL << 2*15)  );
}

void klavesnice_config(void){
RCC->AHB1ENR  |= ((1UL <<  3) );


GPIOD->MODER    |=  ((1UL << 2*6) | // vystup PD6..9
                     (1UL << 2*7) |
                     (1UL << 2*8) |
                     (1UL << 2*9)  );
GPIOD->OTYPER   &= ~((1UL <<   6) |
                     (1UL <<   7) |
                     (1UL <<   8) |
                     (1UL <<   9)  );   // Push - Pull
GPIOD->OSPEEDR    &= ~((3UL << 2*6) |
                     (3UL <<2*7) |
                     (3UL << 2*8) |
                     (3UL << 2*9)  );
GPIOD->OSPEEDR  |=  ((2UL << 2*6) | // 50 Mhz out
                     (2UL << 2*7) |
                     (2UL << 2*8) |
```

```
63                              (2UL << 2*9)  );
64    GPIOD->PUPDR     &= ~((3UL << 2*6) |
65                            (3UL << 2*7) |
66                            (3UL << 2*8) |
67                            (3UL << 2*9)  );
68
69    GPIOD->MODER     &=  ~((3UL << 2*0) |  // vstup PD0..2
70                            (3UL << 2*1) |
71                            (3UL << 2*2))  ;
72
73    GPIOD->PUPDR     &= ~((3UL << 2*0) |
74                            (3UL << 2*1) |
75                            (3UL << 2*2) );
76    GPIOD->PUPDR     |= ((1UL << 2*0) |
77                            (1UL << 2*1) |
78                            (1UL << 2*2) );
79    }
80    void LCD_config(void){
81    RCC->AHB1ENR  |= ((1UL <<  4) );  // PE povolit clock
82    GPIOE->MODER     |=  ( (1UL << 2*3) |  //  RS
83                             (1UL << 2*4) |  //  R/W
84                             (1UL << 2*5) |  //  E
85                            (1UL << 2*6) | // vystup DB4..DB7 data
86                            (1UL << 2*7) |
87                            (1UL << 2*8) |
88                            (1UL << 2*9) |
89                             (1UL << 2*10))  ;
90    GPIOE->OTYPER    &= ~( (1UL << 3) |
91                             (1UL << 4) |
92                             (1UL << 5) |
93                            (1UL <<   6) |
94                            (1UL <<   7) |
95                            (1UL <<   8) |
96                            (1UL <<   9) |
97                             (1UL <<  10)  );   // Push - Pull
98    GPIOE->OSPEEDR     &= ~( (3UL << 2*3) |
99                             (3UL << 2*4) |
100                            (3UL << 2*5 )|
101                            (3UL << 2*6) |
102                           (3UL << 2*7) |
103                           (3UL << 2*8) |
104                           (3UL << 2*9) |
105                           (3UL << 2*10) ) ;
106   GPIOE->OSPEEDR  |=  ( (2UL << 2*3) |  // 50 Mhz out
107                           (2UL << 2*4) |
108                           (2UL << 2*5) |
109                            (2UL << 2*6) |
110                           (2UL << 2*7) |
111                           (2UL << 2*8) |
112                           (2UL << 2*9) |
113                          (2UL << 2*10) );
114   GPIOE->PUPDR     &= ~((3UL << 2*3) |
115                           (3UL << 2*4) |
116                           (3UL << 2*5) |
117                            (3UL << 2*6) |
118                           (3UL << 2*7) |
119                           (3UL << 2*8) |
120                           (3UL << 2*9) |
121                          (3UL << 2*10) )
122   ;}
123
124   void ADC_config(void){
```

```
125         RCC->AHB1ENR  |= ((1UL ) );  // PA povolit clock
126         RCC->APB2ENR |= 0x00000100;    // povolení hodinového signálu k AD
    pøevodníku
127         GPIOA->MODER    |=    (3UL << 2*1)  ; // PA1    *** Vstup ch1 A/D prevod
    ***
128
129         ADC1->SMPR2 |= (7UL << 3*1); // nastaveni casu samplovani pro kanal 1: 480
    cycles = 111
130         ADC1->CR1 = 0x00000800; // nastaveni DISCEN do 1  Discontinuous mode
131         ADC1->SQR1 = 0x00000000; // nastaveni poctu konvertovanych kanalu na 1
132         ADC1->SQR3 = 0x00000001; // konvertovat se bude kanal cislo 1
133         ADC1->CR2  |= 1UL  ;  // PA povolit ADC1 => zapnuti AD prevodniku
134
135   // ADC1->CR2 |= 0x08; // resetovani kalibracnich registru
136   // while ((ADC1->CR2 & 0x08) != 0) {} ;// cekani na nacteni hodnot po resetu
137   // ADC1->CR2 |= 0x04; // zapnuti autokalibrace
138   // while ((ADC1->CR2 & 0x04) != 0) {} ;// cekani na konec autokalibrace
139   }
140   void Buzz_config(void){
141         RCC->AHB1ENR  |= ((1UL ) );  // PA povolit clock
142           GPIOA->MODER    |=    (1UL << 2*8) ;  // PA8  out
143           GPIOA->OTYPER   &= ~ (1UL << 8) ;       // Output push-pull
144           GPIOA->OSPEEDR    &= ~ (3UL << 2*8)  ;
145           GPIOA->OSPEEDR  |=    (2UL << 2*8) ;  // 50 Mhz out
146           GPIOA->PUPDR    &= ~(3UL << 2*8) ; // No pull-up, pull-down
147   }
148   void Mic_config(void){
149             RCC->AHB1ENR  |= ((1UL << 1 ) );  // PB povolit clock
150           GPIOB->MODER    |=    (2UL << 2*10) ;  // PB pin 10
151           GPIOB->AFR[1] |=  (5UL << 4*2) ;   // AF 5 .... SPI2_SCK alter.f.
152           GPIOB->OTYPER   &= ~ (1UL << 10) ;       // Output push-pull
153           GPIOB->OSPEEDR    &= ~ (3UL << 2*10)  ;
154           GPIOB->OSPEEDR  |=    (2UL << 2*10) ; // 50 Mhz out
155           GPIOB->PUPDR    &= ~(3UL << 2*10) ; // No pull-up, pull-down
156             RCC->AHB1ENR  |= ((1UL << 2 ) );  // PC povolit clock
157           GPIOC->MODER    |=    (2UL << 2*3) ;  // PC pin 3
158           GPIOC->AFR[0] |=  (5UL << 4*3) ;   // AF 5 .... SPI2_MOSI alter.f.
159           GPIOC->OTYPER   |=  (1UL << 3) ;       // Output OPEN drain
160           GPIOC->OSPEEDR    &= ~ (3UL << 2*3)  ;
161           GPIOC->OSPEEDR  |=    (2UL << 2*3) ;  // 50 Mhz out
162           GPIOC->PUPDR    &= ~(3UL << 2*3) ; // No pull-up, pull-down
163
164           RCC->APB1ENR |= (1UL << 14 ); //  SPI2 clock enabled
165   //        SPI2->CR1 |= ((1L << 1*1)|    //  CK to 1 when idle
166   //                      (1L << 1*0 )|   //  PHA = 1
167   //                      (1L << 1*10)|   //  Bit 10 RXONLY: Receive only
168   //                      (1L << 1*6)|    //  Bit 6 SPE: SPI enable
169   //                      (5L << 3)|    //  BR[2:0]: Baud rate control 101: fPCLK/64
170   //                      (1L << 1*2));   //  Bit 2 MSTR: Master selection
171
172       SPI2->I2SCFGR  |= (     (1L << 1*3)|    // CKPOL: Steady state clock polarity
173                       (2L << 1*4 )|     //  10: LSB justified standard
174                       (3L << 1*8)|    //  11: Master - receive
175                 //    (1L << 1*10)|   //  Bit 10 I2SE: I2S Enable
176                       (1L << 11*1)) ; //  Bit 11 I2SMOD: I2S mode selection
177         SPI2->I2SPR  |= (16L )  ; //  Bit 7:0 I2SDIV: I2S Linear prescaler
178         SPI2->I2SCFGR  |=(1L << 1*10) ; //  Bit 10 I2SE: I2S
    Enable
179   }
180   void DA_config(void){    // DA ... PA4
181     RCC->AHB1ENR  |= ((1UL ) );  // PA povolit clock
182         RCC->APB1ENR |= (1UL << 1*29 );     //DACEN: DAC interface clock enable
```

```
183         GPIOA->MODER    |=    (3UL << 2*4)  ; // PA4    *** Vystup  D/A prevod ***
184         DAC->CR |= (1UL << 1*0) ;     // EN1: DAC channel 1 enable
185                   //  (7UL << 1*3 ));     // TSEL1[2:0]: DAC channel 1 sw
    trigger
186   }
187
188
189
190   void Delay (uint32_t ms){
191   os_itv_set (ms);
192      {
193       os_itv_wait ();
194     }
195   }
196
197   uint8_t getkey(){
198       uint8_t key= 0;
199       uint16_t  temp;
200       while (!key){
201           Delay(1000);       // delay for key
202           GPIOD->ODR &= ~((15UL << 6));
203           GPIOD->ODR |= ((7UL << 6));
204             temp = GPIOD->IDR ;
205             temp &= 7 ;
206             switch (temp){
207               case 6 : key ='.'; break ;  // exchange '.'
208               case 5 : key ='0'; break ;
209               case 3 : key ='#'; break ;}
210
211           GPIOD->ODR &= ~((15UL << 6));
212           GPIOD->ODR |= ((11UL << 6));
213             temp = GPIOD->IDR ;
214             temp &= 7 ;
215             switch (temp){
216               case 6 : key ='7'; break ;
217               case 5 : key ='8'; break ;
218               case 3 : key ='9'; break ;}
219
220           GPIOD->ODR &= ~((15UL << 6));
221           GPIOD->ODR |= ((13UL << 6));
222             temp = GPIOD->IDR ;
223             temp &= 7 ;
224             switch (temp){
225               case 6 : key ='4'; break ;
226               case 5 : key ='5'; break ;
227               case 3 : key ='6'; break ;}
228
229           GPIOD->ODR &= ~((15UL << 6));
230           GPIOD->ODR |= ((14UL << 6));
231             temp = GPIOD->IDR ;
232             temp &= 7 ;
233             switch (temp){
234               case 6 : key ='1'; break ;
235               case 5 : key ='2'; break ;
236               case 3 : key ='3'; break ;}
237             }
238     return key ;
239   }
240
241   void write_nibble_res(uint16_t nibble){
242     uint16_t x;
243         GPIOE->BSRR  = (    (1UL <<   (3+16)) |      //     RS=0
```

```
244                             (1UL << ( 4+16) ) |      //  R/W=0
245                             (1UL << (5+16)) );     //  E=0
246         Delay(1); // 100 us
247         GPIOE->BSRR = (1UL << 5) ;     //  E=1  (clk)
248         nibble &= 0x0F;
249         nibble = nibble << 6 ;
250         x = GPIOE->ODR & 0xFC3F ; // nuluj data
251         GPIOE->ODR = nibble | x ; // pridej data
252         Delay(1); // 100 us
253         GPIOE->BSRR = (1UL << (5+16)) ;      //  E=0
254         Delay(1); // 100 us
255   }
256
257   void write_nibble(uint16_t nibble){
258     uint16_t x;
259         GPIOE->BSRR  = (  (1UL << ( 4+16) ) |      //  R/W=0
260                             (1UL << (5+16)) );     //  E=0
261         GPIOE->BSRR = (1UL << 5) ;     //  E=1  (clk)
262         nibble &= 0x0F;
263         nibble = nibble << 6 ;
264         x = GPIOE->ODR & 0xFC3F ; // nuluj data
265         GPIOE->ODR = nibble | x ; // pridej data
266         Delay(1); // 100 us
267         GPIOE->BSRR = (1UL << (5+16)) ;      //  E=0
268         Delay(1); // 100 us
269   }
270   void LCD_ctrlWR(uint16_t cmd){
271         GPIOE->BSRR  =  (1UL <<  (3+16)); //    RS=0
272         write_nibble ( cmd >> 4);
273         write_nibble (cmd);
274           Delay(1); // 100 us
275   }
276   void LCD_dataWR(uint16_t cmd){
277         GPIOE->BSRR  =  (1UL <<  3); //   RS=1
278         write_nibble ( cmd >> 4);
279         write_nibble (cmd);
280           Delay(1); // 100 us
281   }
282   void LCD_ini (void){
283   GPIOE->BSRR  =  (1UL << (10+16) ) ;      //  DIR=0
284         Delay(800); // 80 ms
285         write_nibble_res(0x3);  // 3
286         Delay(50);  // 5 ms
287         write_nibble_res(0x3);
288         Delay(10);  // 1 ms
289         write_nibble_res(0x3);  //
290         Delay(10);  // 1 ms
291         write_nibble_res(0x2);
292         LCD_ctrlWR(0x28);
293         LCD_ctrlWR(0x28);
294         LCD_ctrlWR(0x0C);
295         LCD_ctrlWR(0x06);
296         LCD_ctrlWR(0x01);
297         Delay(20);  // 2 ms
298   }
299   void puts_LCD(int radek, char* ukaz){
300     uint32_t i, adresa, n = 9;
301     if (radek==1)  adresa = 0x80 ; else adresa = 0xC0 ;
302     LCD_ctrlWR(adresa);
303     for (i=0 ; i < n ; ++i ){
304           LCD_dataWR(*ukaz);
305           ++ukaz ;
```

```
306           };
307   }
308   unsigned char hextoascii(unsigned char cislo ){
309     unsigned char asci ;
310     asci = cislo&0x0F ;
311     if (asci > 9)  asci += 7 ;
312     return (asci +='0') ;
313   }
314   void putasci(uint16_t cislo1 ){
315       LCD_dataWR(hextoascii(cislo1 >> 8));
316       LCD_dataWR(hextoascii(cislo1 >> 4));
317       LCD_dataWR(hextoascii(cislo1));
318     }
319   uint16_t sample_1( void ){
320           ADC1->CR2 |= (0x40000000); // start mereni AD => SWSTART
321           while ((ADC1->SR & 0x02) == 0){} ; // cekani na konec mereni AD
322           return (uint16_t) ADC1->DR;
323     }
324   void bzz(uint16_t ms){
325   uint16_t i; ms *= 10;
326      for (i=0;i< ms; ++i ){
327          GPIOA->ODR &= ~(1UL << 8) ;  // bzz off
328       Delay(5);
329         GPIOA->ODR |= (1UL << 8) ;    // bzz on
330       Delay(5);
331       }};
332   void SERIAL_ini(){
333     RCC->APB1ENR |= (1UL << 17);  // USART2 CLK ENABLE
334     RCC->AHB1ENR |= (1UL << 0);   // PORT A CLK ENABLE
335     GPIOA->AFR[0] |= (7U << 4*2) | (7U << 4*3);
336     /* Tx Configuration */
337     GPIOA->MODER &= ~(3UL << 2*2);
338     GPIOA->MODER |= (2UL << 2*2);   // PA2 Alternate Function
339     GPIOA->OTYPER &= ~(1UL << 2); // Pull
340     GPIOA->OSPEEDR &= ~(3UL << 2*2);
341     GPIOA->OSPEEDR |= (2UL << 2*2);   // 50Mhz
342     GPIOA->PUPDR &= ~(3UL << 2*2);
343     GPIOA->PUPDR |= (1UL << 2*2);   // PullUp
344     /* Rx Configuration */
345     GPIOA->MODER &= ~(3UL << 2*3);
346     GPIOA->MODER |= (2UL << 2*3);   // PA2 Alternate Function
347     USART2->BRR = 0x16D; // 115200 Bd
348     USART2->CR1 |= (1U << 3) | (1U << 2); //Rxen Txen
349     USART2->CR1 |= (1U << 13); // usart enable
350   }
351   void putchar1(char znak ){
352     { USART2->DR = (znak);
353       while((!(USART2->SR & 0x40))); // Transmision Complete
354     } }
355
356   void putserial(char* ukaz1 ){
357       unsigned char n=strlen(ukaz1)  ;
358     unsigned char i ;
359       for(i=0; i<n ;++i)
360     { USART2->DR = (*ukaz1);
361       while((!(USART2->SR & 0x40))); // Transmision Complete
362       ++ukaz1 ;
363     } }
364   uint16_t  sample1( ){
365       uint16_t znak;
366      while(!(SPI2->SR & 0x1)){}; // Bit 0 RXNE: Receive buffer not empty
367       znak = SPI2->DR ;
```

```
368       return znak ;
369     }
370   void read_data_mic(uint16_t* ukaz1, uint16_t delka){
371       uint16_t ind2 ;
372       for (  ind2=0;ind2< delka; ++ind2 ){
373           while(!(SPI2->SR & 0x1)){}; // Bit 0 RXNE: Receive buffer not empty
374             (*ukaz1) = HTONS(SPI2->DR); // microphone
375               ++ukaz1 ;
376             }
377   }
378    void write_da (uint16_t* ukaz1,uint16_t delka, uint16_t cas_us){
379       uint16_t ind2, k ;
380         DAC->DHR12R1= 1500 ;
381
382       for (ind2=0; ind2< delka; ++ind2 ){
383         out1 = (*ukaz1)+1500;
384   //      if (out1 > 3000) {out1 = 1500;}
385         DAC->DHR12R1 = out1 ;
386
387           ++ukaz1 ;
388         for(k=0 ; k < cas_us; ++k ){};
389   //        Delay(1);
390           } }
391
392
```