```c
#include "interrupt.h"
#include  "led.h"

struct keys key[4]={0};
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{

    if(htim->Instance == TIM6)
    {
        key[0].value = HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_0);
        key[1].value = HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_1);
        key[2].value = HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_2);
        key[3].value = HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0);

        for(int i=0;i<4;i++)
        {
            switch(key[i].state)
            {
                case 0:
                    if(key[i].value == 0)
                        key[i].state = 1;
                    break;
                case 1:
                    if(key[i].value == 0)
                    {
                        key[i].state = 2;
                        key[i].click_time = 0;
                    }
                    else
                        key[i].state = 0;
                    break;
                case 2:
                    if(key[i].value == 0)
                        key[i].click_time ++;
                    else if(key[i].click_time >100)
                    {
                        key[i].long_flag = 1;
                        key[i].state = 0;
                    }
                    else
                    {
                        key[i].short_flag = 1;
                        key[i].state = 0;
                    }

//                    if(key[i].value == 0)
//                        key[i].click_time ++;
//                    else if(key[i].click_time >100)
//                    {
//                        key[i].long_flag = 1;
//                        key[i].state = 0;
//                    }
//                    else
//                    {
//                        switch(key[i].double_state)
//                        {
//                            case 0:
//                                key[i].double_state = 1;
//                                key[i].double_time = 0;
//                                break;
//                            case 1:
//                                key[i].double_state = 0;
//                                key[i].double_flag = 1;
//                                break;
//                        }
//
//                        key[i].state = 0;
//
//                    }
                    break;
            }
```

```
72  //          if(key[i].double_state == 1)
73  //          {
74  //              key[i].double_time++;
75  //              if(key[i].double_time>30)
76  //              {
77  //                  key[i].short_flag = 1;
78  //                  key[i].double_state = 0;
79  //              }
80  //          }
81
82          }
83
84
85      }
86
87
88  }
89
90
91  char rx_data;
92  char rx_arry[50];
93  char rx_pointer;
94
95  void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
96  {
97      rx_arry[rx_pointer++] = rx_data;
98      HAL_UART_Receive_IT(huart,(uint8_t *)&rx_data,1);
99  }
100
101
102 #define PB4_FERQ 1000000
103 #define PA15_FERQ 1000000
104 #define PA1_FERQ 1000000
105 #define PA2_FERQ 1000000
106 #define PA6_FERQ 1000000
107 #define PA7_FERQ 1000000
108
109 char flag_PB4;
110 char flag_PA1;
111 char flag_PA2;
112 char flag_PA6;
113 char flag_PA7;
114 uint PB4_freq,PB4_duty;
115 uint PB4_rise,PB4_fall;
116 uint PA15_freq,PA15_duty;
117 uint PA15_rise,PA15_fall;
118 uint PA1_freq,PA1_duty;
119 uint PA1_high,PA1_low,PA1_value1,PA1_value2,PA1_value3;
120 uint PA2_freq,PA2_duty;
121 uint PA2_high,PA2_low,PA2_value1,PA2_value2,PA2_value3;
122 uint PA6_freq,PA6_duty;
123 uint PA6_high,PA6_low,PA6_value1,PA6_value2,PA6_value3;
124 //uint PA7_freq,PA7_duty;
125 //uint PA7_high,PA7_low,PA7_value1,PA7_value2,PA7_value3;
126
127 void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
128 {
129     if(htim->Instance == TIM16)
130     {
131
132         if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)
133         {
134
135             if(flag_PB4 == 0)//rise
136             {
137                 PB4_rise = __HAL_TIM_GetCounter(htim);
138                 __HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_1,TIM_INPUTCHANNELPOLARITY_FALLING);
139                 __HAL_TIM_SetCounter(htim,0);
140             }
141             else
142             {
```

```
143                 PB4_fall = __HAL_TIM_GetCounter(htim);
144 
    __HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_1,TIM_INPUTCHANNELPOLARITY_RISING);
145             }
146 
147             flag_PB4 = !flag_PB4;
148             PB4_freq = PB4_FERQ/PB4_rise;
149             PB4_duty = PB4_fall*100/PB4_rise;
150         }
151     }
152 
153     if(htim->Instance == TIM2)
154     {
155 
156         if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2)
157         {
158 
159             if(flag_PA1 == 0)//rise
160             {
161                 PA1_value1 = __HAL_TIM_GetCounter(htim);
162                 __HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_2,TIM_INPUTCHANNELPOLARITY_FALLING);
163                 flag_PA1 = 1;
164             }
165             else if(flag_PA1 == 1)
166             {
167                 PA1_value2 = __HAL_TIM_GetCounter(htim);
168                 __HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_2,TIM_INPUTCHANNELPOLARITY_RISING);
169                 flag_PA1 = 2;
170                 if(PA1_value2 >= PA1_value1)
171                 {
172                     PA1_high = PA1_value2 - PA1_value1;
173                 }
174                 else
175                 {
176                     PA1_high = 0xffffffff - PA1_value1 + PA1_value2;
177                 }
178             }
179             else if(flag_PA1 == 2)
180             {
181                 PA1_value3 =  HAL_TIM_GetCounter(htim);
182                 __HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_2,TIM_INPUTCHANNELPOLARITY_RISING);
183                 flag_PA1 = 0;
184                 if(PA1_value3 >= PA1_value2)
185                 {
186                     PA1_low = PA1_value3 - PA1_value2;
187                 }
188                 else
189                 {
190                     PA1_low = 0xffffffff - PA1_value2 + PA1_value3;
191                 }
192             }
193             PA1_freq = PA1_FERQ/(PA1_high + PA1_low);
194             PA1_duty = PA1_high*100/(PA1_high + PA1_low);
195         }
196 
197         if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_3)
198         {
199 
200             if(flag_PA2 == 0)//rise
201             {
202                 PA2_value1 = __HAL_TIM_GetCounter(htim);
203                 __HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_3,TIM_INPUTCHANNELPOLARITY_FALLING);
204                 flag_PA2 = 1;
205             }
206             else if(flag_PA2 == 1)
207             {
208                 PA2_value2 = __HAL_TIM_GetCounter(htim);
209                 __HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_3,TIM_INPUTCHANNELPOLARITY_RISING);
210                 flag_PA2 = 2;
211                 if(PA2_value2 >= PA2_value1)
212                 {
```

```
213                     PA2_high = PA2_value2 - PA2_value1;
214                 }
215                 else
216                 {
217                     PA2_high = 0xffffffff - PA2_value1 + PA2_value2;
218                 }
219             }
220             else if(flag_PA2 == 2)
221             {
222                 PA2_value3 = __HAL_TIM_GetCounter(htim);
223                 __HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_3,TIM_INPUTCHANNELPOLARITY_RISING);
224                 flag_PA2 = 0;
225                 if(PA2_value3 >= PA2_value2)
226                 {
227                     PA2_low = PA2_value3 - PA2_value2;
228                 }
229                 else
230                 {
231                     PA2_low = 0xffffffff - PA2_value2 + PA2_value3;
232                 }
233             }
234             PA2_freq = PA2_FERQ/(PA2_high + PA2_low);
235             PA2_duty = PA2_high*100/(PA2_high + PA2_low);
236         }
237 
238     }
239 //
240     if(htim->Instance == TIM3)
241     {
242 
243         if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)
244         {
245 
246             if(flag_PA6 == 0)//rise
247             {
248                 PA6_value1 = __HAL_TIM_GetCounter(htim);
249                 __HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_1,TIM_INPUTCHANNELPOLARITY_FALLING);
250                 flag_PA6 = 1;
251             }
252             else if(flag PA6 == 1)
253             {
254                 PA6_value2 = __HAL_TIM_GetCounter(htim);
255                 __HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_1,TIM_INPUTCHANNELPOLARITY_RISING);
256                 flag_PA6 = 2;
257                 if(PA6_value2 >= PA6_value1)
258                 {
259                     PA6_high = PA6_value2 - PA6_value1;
260                 }
261                 else
262                 {
263                     PA6_high = 0xffff - PA6_value1 + PA6_value2;
264                 }
265             }
266             else if(flag_PA6 == 2)
267             {
268                 PA6_value3 = __HAL_TIM_GetCounter(htim);
269                 __HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_1,TIM_INPUTCHANNELPOLARITY_RISING);
270                 flag_PA6 = 0;
271                 if(PA6_value3 >= PA6_value2)
272                 {
273                     PA6_low = PA6_value3 - PA6_value2;
274                 }
275                 else
276                 {
277                     PA6_low = 0xffff - PA6_value2 + PA6_value3;
278                 }
279             PA6_freq = PA6_FERQ/(PA6_high + PA6_low);
280             PA6_duty = PA6_high*100/(PA6_high + PA6_low);
281             }
282 //          PA6_freq = PA6_FERQ/(PA6_high + PA6_low);
283 //          PA6_duty = PA6_high*100/(PA6_high + PA6_low);
```

```
284         }
285 
286 //      if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2)
287 //      {
288 
289 //          if(flag_PA7 == 0)//rise
290 //          {
291 //              PA7_value1 = __HAL_TIM_GetCounter(htim);
292 //                HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_2,TIM_INPUTCHANNELPOLARITY_FALLING);
293 //              flag_PA7 = 1;
294 //          }
295 //          else if(flag_PA7 == 1)
296 //          {
297 //              PA7_value2 = __HAL_TIM_GetCounter(htim);
298 //              __HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_2,TIM_INPUTCHANNELPOLARITY_RISING);
299 //              flag_PA7 = 2;
300 //              if(PA7_value2 >= PA7_value1)
301 //              {
302 //                  PA7_high = PA7_value2 - PA7_value1;
303 //              }
304 //              else
305 //              {
306 //                  PA7_high = 0xffff - PA7_value1 + PA7_value2;
307 //              }
308 //          }
309 //          else if(flag_PA7 == 2)
310 //          {
311 //              PA7_value3 = __HAL_TIM_GetCounter(htim);
312 //                HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_2,TIM_INPUTCHANNELPOLARITY_RISING);
313 //              flag_PA7 = 0;
314 //              if(PA7_value3 >= PA7_value2)
315 //              {
316 //                  PA7_low = PA7_value3 - PA7_value2;
317 //              }
318 //              else
319 //              {
320 //                  PA7_low = 0xffff - PA7_value2 + PA7_value3;
321 //              }
322 //          }
323 //          PA7_freq = PA7_FERQ/(PA7_high + PA7_low);
324 //          PA7_duty = PA7_high*100/(PA7_high + PA7_low);
325 //      }
326 
327     }
328 //
329 
330     if(htim->Instance == TIM8)
331     {
332         if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)//rise
333         {
334             PA15_rise = __HAL_TIM_GetCounter(htim);
335             __HAL_TIM_SET_COUNTER(htim,0);
336         }
337         else if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2)
338         {
339             PA15_fall = __HAL_TIM_GetCounter(htim);
340         }
341 
342         PA15_freq = PA15_FERQ/PA15_rise;
343         PA15_duty = PA15_fall*100/PA15_rise;
344 
345     }
346 
347 
348 }
349 
350 
351 
```