```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
125
126 void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
127 {
128     if(htim->Instance == TIM16)
129     {
130
131         if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)
132         {
133
134             if(flag_PB4 == 0)//rise
135             {
136                 PB4_rise = __HAL_TIM_GetCounter(htim);
137                 __HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_1,TIM_INPUTCHANNELPOLARITY_FALLING);
138                 __HAL_TIM_SetCounter(htim,0);
139             }
140             else
141             {
142                 PB4_fall = __HAL_TIM_GetCounter(htim);
```

```
__HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_1,TIM_INPUTCHANNELPOLARITY_RISING);
            }

            flag_PB4 = !flag_PB4;
            PB4_freq = PB4_FERQ/PB4_rise;
            PB4_duty = PB4_fall*100/PB4_rise;
        }
    }

    if(htim->Instance == TIM2)
    {

        if(htim->Channel1 == HAL_TIM_ACTIVE_CHANNEL_2)
        {

            if(flag_PA1 == 0)//rise
            {
                PA1_value1 = __HAL_TIM_GetCounter(htim);
                __HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_2,TIM_INPUTCHANNELPOLARITY_FALLING);
                flag_PA1 = 1;
            }
            else if(flag_PA1 == 1)
            {
                PA1_value2 = __HAL_TIM_GetCounter(htim);
                __HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_2,TIM_INPUTCHANNELPOLARITY_RISING);
                flag_PA1 = 2;
                if(PA1_value2 >= PA1_value1)
                {
                    PA1_high = PA1_value2 - PA1_value1;
                }
                else
                {
                    PA1_high = 0xffffffff - PA1_value1 + PA1_value2;
                }
            }
            else if(flag_PA1 == 2)
            {
                PA1_value3 = __HAL_TIM_GetCounter(htim);
                  HAL TIM SET CAPTUREPOLARITY(htim,TIM CHANNEL 2,TIM INPUTCHANNELPOLARITY RISING);
                flag_PA1 = 0;
                if(PA1_value3 >= PA1_value2)
                {
                    PA1_low = PA1_value3 - PA1_value2;
                }
                else
                {
                    PA1_low = 0xffffffff - PA1_value2 + PA1_value3;
                }
            }
            PA1_freq = PA1_FERQ/(PA1_high + PA1_low);
            PA1_duty = PA1_high*100/(PA1_high + PA1_low);
        }

        if(htim->Channel1 == HAL_TIM_ACTIVE_CHANNEL_3)
        {

            if(flag_PA2 == 0)//rise
            {
                PA2 value1 =   HAL TIM GetCounter(htim);
                __HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_3,TIM_INPUTCHANNELPOLARITY_FALLING);
                flag_PA2 = 1;
            }
            else if(flag_PA2 == 1)
            {
                PA2_value2 = __HAL_TIM_GetCounter(htim);
                __HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_3,TIM_INPUTCHANNELPOLARITY_RISING);
                flag_PA2 = 2;
                if(PA2_value2 >= PA2_value1)
                {
                    PA2_high = PA2_value2 - PA2_value1;
```

```
213                 }
214                 else
215                 {
216                     PA2_high = 0xffffffff - PA2_value1 + PA2_value2;
217                 }
218             }
219             else if(flag_PA2 == 2)
220             {
221                 PA2 value3 =   HAL TIM GetCounter(htim);
222                 __HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_3,TIM_INPUTCHANNELPOLARITY_RISING);
223                 flag_PA2 = 0;
224                 if(PA2_value3 >= PA2_value2)
225                 {
226                     PA2_low = PA2_value3 - PA2_value2;
227                 }
228                 else
229                 {
230                     PA2 low = 0xffffffff - PA2 value2 + PA2 value3;
231                 }
232             }
233             PA2_freq = PA2_FERQ/(PA2_high + PA2_low);
234             PA2_duty = PA2_high*100/(PA2_high + PA2_low);
235         }
236 
237     }
238 //
239     if(htim->Instance == TIM3)
240     {
241 
242         if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)
243         {
244 
245             if(flag_PA6 == 0)//rise
246             {
247                 PA6_value1 = __HAL_TIM_GetCounter(htim);
248                 __HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_1,TIM_INPUTCHANNELPOLARITY_FALLING);
249                 flag_PA6 = 1;
250             }
251             else if(flag_PA6 == 1)
252             {
253                 PA6_value2 = __HAL_TIM_GetCounter(htim);
254                 __HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_1,TIM_INPUTCHANNELPOLARITY_RISING);
255                 flag_PA6 = 2;
256                 if(PA6_value2 >= PA6_value1)
257                 {
258                     PA6_high = PA6_value2 - PA6_value1;
259                 }
260                 else
261                 {
262                     PA6_high = 0xffff - PA6_value1 + PA6_value2;
263                 }
264             }
265             else if(flag_PA6 == 2)
266             {
267                 PA6_value3 = __HAL_TIM_GetCounter(htim);
268                 __HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_1,TIM_INPUTCHANNELPOLARITY_RISING);
269                 flag_PA6 = 0;
270                 if(PA6_value3 >= PA6_value2)
271                 {
272                     PA6 low = PA6 value3 - PA6 value2;
273                 }
274                 else
275                 {
276                     PA6_low = 0xffff - PA6_value2 + PA6_value3;
277                 }
278             PA6_freq = PA6_FERQ/(PA6_high + PA6_low);
279             PA6_duty = PA6_high*100/(PA6_high + PA6_low);
280             }
281 //          PA6_freq = PA6_FERQ/(PA6_high + PA6_low);
282 //          PA6_duty = PA6_high*100/(PA6_high + PA6_low);
283         }
```

```
284
285
286         }
287     //
288
289         if(htim->Instance == TIM8)
290         {
291             if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)//rise
292             {
293                 PA15_rise = __HAL_TIM_GetCounter(htim);
294                 __HAL_TIM_SET_COUNTER(htim,0);
295             }
296             else if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2)
297             {
298                 PA15_fall = __HAL_TIM_GetCounter(htim);
299             }
300
301             PA15 freq = PA15 FERQ/PA15 rise;
302             PA15_duty = PA15_fall*100/PA15_rise;
303
304         }
305
306
307     }
308
309
310
```