```c
#include "interrupt.h"

#define PA15_FREQ 1000000
#define PB4_FREQ 1000000

struct keys key[4] = {0};

char rx_arry[50];
char rx data;
char rx_pointer;

uint PA15_rise,PA15_fall;//TIM8
uint PA15_freq,PA15_duty;

uint PB4_rise,PB4_fall;//TIM3
uint PB4_freq,PB4_duty;

void HAL TIM IC CaptureCallback(TIM HandleTypeDef *htim)
{
    if(htim->Instance == TIM8)
    {
        if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)
        {
            PA15_rise = __HAL_TIM_GetCounter(htim);
            __HAL_TIM_SetCounter(htim,0);
            PA15_freq = PA15_FREQ/PA15_rise;
            PA15_duty = PA15_fall*100/PA15_rise;
        }
        if(htim->Channel == HAL TIM ACTIVE CHANNEL 2)
        {
            PA15_fall = __HAL_TIM_GetCounter(htim);
        }
    }
    if(htim->Instance == TIM3)
    {
        if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)
        {
            PB4_rise = __HAL_TIM_GetCounter(htim);
            __HAL_TIM_SetCounter(htim,0);
            PB4 freq = PB4 FREQ/PB4 rise;
            PB4_duty = PB4_fall*100/PB4_rise;
        }
        if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2)
        {
            PB4_fall = __HAL_TIM_GetCounter(htim);
        }
    }
}
void HAL UART RxCpltCallback(UART HandleTypeDef *huart)
{
    if(huart->Instance == USART1)
    {
        rx_arry[rx_pointer++] = rx_data;
        HAL_UART_Receive_IT(huart, (uint8_t *)&rx_data, 1);
    }
}
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;
```

```
72                      break;
73                  case 1:
74                      if(key[i].value == 0)
75                      {
76                          key[i].state = 2;
77                          key[i].click_time = 0;
78                      }
79                      else key[i].state = 0;
80                      break;
81                  case 2:
82  //                      if(key[i].value == 0)
83  //                          key[i].click_time++;
84  //                      else if(key[i].click_time>70)
85  //                      {
86  //                          key[i].long_flag = 1;
87  //                          key[i].state = 0;
88  //                      }
89  //                      else
90  //                      {
91  //                          switch(key[i].double_state)
92  //                          {
93  //                              case 0:
94  //                                  key[i].double_state = 1;
95  //                                  break;
96  //                              case 1:
97  //                                  key[i].double_state = 0;
98  //                                  key[i].double_flag = 1;
99  //                                  break;
100 //                          }
101 //
102 //                          key[i].state = 0;
103 //                          key[i].double_time = 0;
104 //                      }
105
106                     if(key[i].value == 1)
107                     {
108                         key[i].state = 0;
109                         key[i].short_flag = 1;
110                     }
111                     break;
112             }
113
114 //          if(key[i].double_state == 1)
115 //          {
116 //              key[i].double_time++;
117 //              if(key[i].double_time>30)
118 //              {
119 //                  key[i].short_flag = 1;
120 //                  key[i].double_state = 0;
121 //              }
122 //          }
123         }
124     }
125 }
```