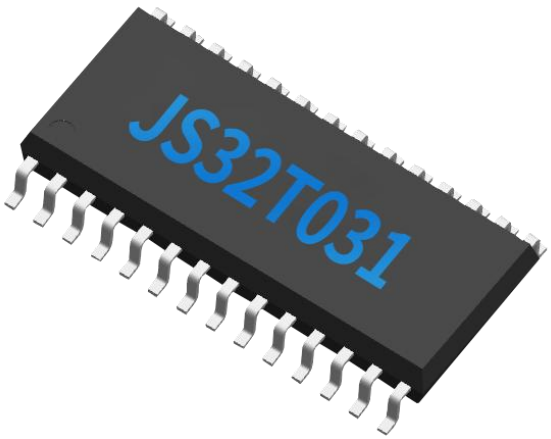
巨晟科技
Jusheng Technology

支持 LED/LCD 驱动+触控的 32 位微控制器

JS32T031 触控系列

软件设计注意事项

V1.2



珠海巨晟科技股份有限公司
地　址：广东省珠海市高新区金唐路 1 号港湾 1 号湾 8 栋 4 楼
电　话：0756-3335384　　传　真：0756-3335384
网　站：www.honor-ic.com　　邮　编：519080

版本历史

变更类型：A - 增加　M - 修订　D - 删除

变更版本号	日期	变更类型	修改人	审核	摘要
V1.0	2022.3.15	A			正式版本
V1.1	2023.6.28	M			
V1.2	2024.6.25	M			

版权声明

目录

1. 用户配置文件说明

应用工程编译成功后，在 Project->KEIL-ARM->Eflash_Proj 中生成烧录文件。
同目录有以下几个文件：
app_project.bin ： 应用工程最终生成的烧录文件。
makecode.ini : 用户配置文件。
makecode.exe ： 用户配置文件转换工具，应用工程默认每次编译后自动调用此程序，用户注意不要随意更改该配置。
js_link_burn.exe ：用户烧录文件生成工具，应用工程默认每次编译后自动调用此程序，用户注意不要随意更改该配置。

1.1. 用户配置文件的配置说明

用户可通过修改 makecode.ini，配置芯片的相关功能，功能如下表所示
注：所有配置为 16 进制，前面不加 0x，配置内容中‘=’前后不要加空格等额外符号。

Name	Defualt	Description
CODE_PROTECT_DIS	1	代码是否需要保护 0: 保护 1: 不保护（默认）
NVR_WRITE_PER	FF	Nvr0～7 使能擦写标识，总共 8bit，分别对应 Nvr0～7，每一 bit 中 0: 禁止擦写 1: 使能擦写（默认）
MAIN_WRITE_PER0	FFFFFFFF	用户程序 main 区使能擦写标识,总共 32bit,每一 bit 对应 1k(2 个 sector)内容，32bit 分别对应 32k 内容，每一 bit 中 0: 禁止擦写 1: 使能擦写（默认）
MAIN_WRITE_PER1	FFFFFFFF	用户程序 main 区使能擦写标识,总共 32bit,每一 bit 对应 1k(2 个 sector)内容，32bit 分别对应 32k 内容，每一 bit 中 0: 禁止擦写 1: 使能擦写（默认）
MAIN_CODE_CHECK_EN	0	用户 main 程序区代码校验是否使能 0: 不使能（默认） 1: 使能；若设置为校验使能且校验不通过，用户程序不能启动
MAIN_CODE_LENGTH	8000	用户程序 main 区代码大小,代码大小的值为一个 sector(512byte)的倍数。用户 main 区程序的 4 字节校验放在最大代码长度处。
SWD_EN	0	SWD 是否使能 0: SWD 失能，需要和 SYS_CON1 的【SWD_EN】同时作用，才能关闭 SWD 接口。 1: SWD 使能
MCLR_EN	0	MCLR(PA12)复位引脚使能控制 0: MCLR 功能无效 1: MCLR 功能有效
SWD_CLK_PULLUP	1	SWD_CLK 引脚上下拉配置 0: 下拉 1: 上拉（默认）

1.2. 用户配置文件的功能说明

1.2.1. 关于校验功能

使能用户 main 区代码校验功能，需要配置 makecode.ini 的【MAIN_CODE_CHECK_EN】和【MAIN_CODE_LENGTH】。

若设置【MAIN_CODE_CHECK_EN】=1，则开启 main 区代码校验功能，校验代码区的大小由【MAIN_CODE_LENGTH】决定，校验码存放在校验代码区的最后 4 个字节，注意要正确配置，否则校验不通过，用户程序无法启动。

1.2.2. 关于 nvr 区的使用说明

NVR0~7 区域可以用于保存用户的数据信息。

出于对 NVR 的保护，请不要频繁保存参数。如果需要频繁地保存参数，建议避免重复操作某个 NVR 区域。

2. Keil 工程内存空间的配置说明

JS32T031 系列芯片有两个版本的内存配置

- Flash 的大小 32KB，RAM 的大小为 2KB
- Flash 的大小 64KB，RAM 的大小为 4KB

用户需要根据实际芯片的内存大小，修改 keil 工程的内存空间配置，才能正确使用芯片。有两处地方需要修改

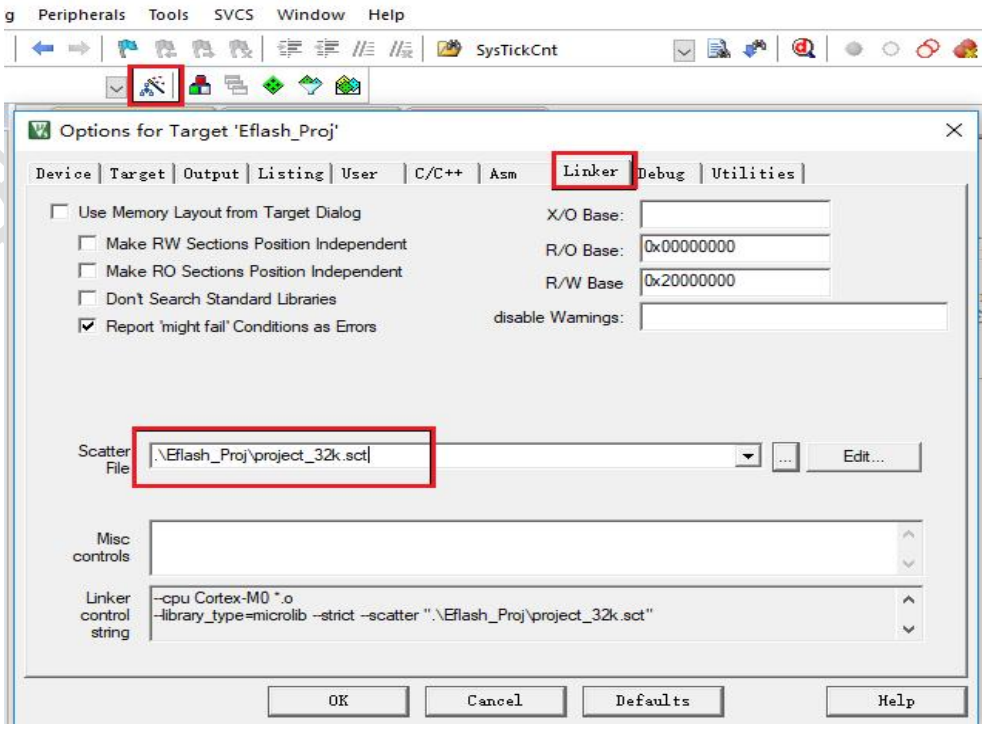
- 不同版本的芯片需要使用不同的 Scatter File 配置。
- 用户配置文件 makecode.ini 的【MAIN_CODE_LENGTH】配置。

2.1. Scatter File 配置

在 Project->KEIL-ARM->Eflash_Proj 目录下，原厂提供两个配置文件 project_32k.sct 和 project_64k.sct，两个版本的.sct 文件对应关系如下：

- **project_32k.sct**：Flash 的大小 32KB，RAM 的大小为 2KB
- **project_64k.sct**：Flash 的大小 64KB，RAM 的大小为 4KB

用户根据实际芯片的大小，在 Options for Target -> Linker->Scatter File 中加载对应的.sct 配置，如下图所示。



2.2. 【MAIN_CODE_LENGTH】配置

根据用户配置文件的配置说明，正确配置 makecode.ini【MAIN_CODE_LENGTH】。

- MAIN_CODE_LENGTH=**8000** ： Flash 的大小 32KB，RAM 的大小为 2KB
- MAIN_CODE_LENGTH=**10000** ： Flash 的大小 64KB，RAM 的大小为 4KB

3. SWD 的使用说明

3.1. SWD 作为通用 IO 的配置

SWD 的 IO 口默认是特殊功能，要使得 SWD_CLK 和 SWD_DATA 两个 IO 作为通用 IO，需要关闭 SWD，具体做法如下，

- 配置 makecode.ini 的【SWD_EN】，即 SWD__EN=0（默认配置）。
- 用户程序中配置寄存器 SYS_CON1 的 SWD__EN 为 0。

为了方便可以在 SWD 失能前，使用 SWD 接口烧录/升级程序。建议在 SWD 失能前，**延时 150ms 以上。**一旦 SWD 失能，需要进入超级模式，才能恢复 SWD 接口的默认功能。

3.2. SWD_CLK 引脚电平的使用说明

用户可通过配置 makecode.ini 的【SWD_CLK_PULLUP】决定 SWD_CLK 引脚开机后是否有上下拉功能。

SWD_CLK, SWD_DAT, PA13 三个引脚需要在芯片开机时满足特殊电平才能保证芯片正常开机。为了方便用户设计硬件电路，若 SWD_CLK 配置是上拉/下拉功能，开机时 SWD_CLK 电平要保证一直是高/低电平（SWD_CLK 配置上拉，开机时此引脚电平要为高；SWD_CLK 配置下拉，开机时此引脚电平要为低），这样，芯片便会正常开机。其次引脚不能接开机时影响电平的外设，例如：ADC，LED 等。

3.3. SWD 作为 LCD 功能

当 SWD 的 IO 被设置为 LCD 功能时，SWD 功能自动关闭，建议不要用 SWD 的 IO 口做 LCD 功能，方便用户使用 SWD 功能。

4. TK 使用说明

4.1. 移植说明

4.1.1. 文件移植

将原厂提供的触摸按键相关文件移植到目标工程，文件包括 tk.lib，tk_cfg.c，tk_cfg.h，tk_define.h，tk_uart.c

4.1.2. 文件修改

在目标工程中，根据开发板实际环境的触摸按键总个数及触摸按键 IO 口，按照格式修改 tk_cfg.h 文件中的宏定义，如下图，**黑色框标注的是需要修改的内容，红色框标注的是注释；**

注：按键总个数要和按键 IO 口个数一致，否则程序会返回错误

函数 u32 tk_process();的返回值表示触摸通道是否被检测到，每一 bit 代表一个通道，从低位到高位，分别对应 TK_0、TK_1、...TK_25 所配置的按键通道，每一 bit 的 1 表示按键被按下，0 表示没有按键被按键下。

tk_swd_buf[]数组存放触摸按键的变化值，数组序号表示的通道从低到高分别与 TK_0、TK_1、...TK_25 的相对应，即数组序号 0 存放的是 TK_0 对应通道的变化量。

```
#define TK_NUM          6                   //按键总个数
/*格式
#define TK_0            TK_PA5  | 50 | TK_WKP_EN
                        IO口    |阈值| 是否是tk唤醒IO
*/
#define TK_0            TK_PA5  | 50 | TK_WKP_EN
#define TK_1            TK_PA6  | 60
#define TK_2            TK_PA7  | 60
#define TK_3            TK_PA8  | 50
#define TK_4            TK_PA9  | 50
#define TK_5            TK_PA10 | 80
#define TK_6            TK_NONE | 0
#define TK_7            TK_NONE | 0
#define TK_8            TK_NONE | 0
#define TK_9            TK_NONE | 0
#define TK_10           TK_NONE | 0
#define TK_11           TK_NONE | 0
#define TK_12           TK_NONE | 0
#define TK_13           TK_NONE | 0
#define TK_14           TK_NONE | 0
#define TK_15           TK_NONE | 0
#define TK_16           TK_NONE | 0
#define TK_17           TK_NONE | 0
#define TK_18           TK_NONE | 0
#define TK_19           TK_NONE | 0
#define TK_20           TK_NONE | 0
#define TK_21           TK_NONE | 0
#define TK_22           TK_NONE | 0
#define TK_23           TK_NONE | 0
#define TK_24           TK_NONE | 0
#define TK_25           TK_NONE | 0
```

如上图可以得出，

按键总个数是 TK_NUM 的定义值 6。

TK_PA5、TK_PA6、TK_PA7、TK_PA8、TK_PA9、TK_PA10 是触摸按键的 IO 口，分别对应的阈值是 50、60、60、50、50、80，其中 TK_PA5 被设置为触摸唤醒 IO。

TK_PA5、TK_PA6、TK_PA7、TK_PA8、TK_PA9、TK_PA10 分别对应 TK_0、TK_1、TK_2、TK_3、TK_4、TK_5，即函数 u32 tk_process();的返回值的第 0bit 到第 5bit 的检测值，分别表示 TK_PA5、TK_PA6、TK_PA7、TK_PA8、TK_PA9、TK_PA10 这些通道是否被按下。

支持间隔配置，如下图，但需要注意 tk_process()的返回值和 tk_swd_buf[]数组中按键的对应顺序。

```
#define TK_NUM          4//按键总个数
/*
#define TK_0            TK_PA5 | 50 | TK_WKP_EN
                        IO口   |阈值| 是否为tk唤醒IO
*/
#define TK_0            TK_NONE
#define TK_1            TK_PA4 | 80
#define TK_2            TK_PA5 | 50
#define TK_3            TK_NONE
#define TK_4            TK_NONE
#define TK_5            TK_NONE
#define TK_6            TK_NONE
#define TK_7            TK_NONE
#define TK_8            TK_PA3 | 80
#define TK_9            TK_NONE
#define TK_10           TK_NONE
#define TK_11           TK_NONE
#define TK_12           TK_NONE
#define TK_13           TK_PA6 | 80
#define TK_14           TK_NONE
#define TK_15           TK_NONE
#define TK_16           TK_NONE
#define TK_17           TK_NONE
#define TK_18           TK_NONE
#define TK_19           TK_NONE
#define TK_20           TK_NONE
#define TK_21           TK_NONE
#define TK_22           TK_NONE
#define TK_23           TK_NONE
#define TK_24           TK_NONE
#define TK_25           TK_NONE
```

注：不要多次调用 tk_init()；如果 tk_init()函数内的 tk_struct()返回成功会修改掉 tk 缓冲区的值，造成异常。

4.1.3. 阈值修改

经过调试之后获得合适的阈值，需要将阈值回填到 tk_cfg.h 文件中，如下图，注意按键的阈值与 IO 口是一一对应的关系，如图 TK_0 的 IO 口是 TK_PA5，对应的阈值是 50。

```
#define TK_NUM          6           //按键总个数
/*格式
#define TK_0            TK_PA5  | 50 | TK_WKP_EN
                        IO口    |阈值| 是否是tk唤醒IO
*/
#define TK_0            TK_PA5  | 50 | TK_WKP_EN
#define TK_1            TK_PA6  | 60
#define TK_2            TK_PA7  | 60
#define TK_3            TK_PA8  | 50
#define TK_4            TK_PA9  | 50
#define TK_5            TK_PA10 | 80
#define TK_6            TK_NONE | 0
#define TK_7            TK_NONE | 0
#define TK_8            TK_NONE | 0
#define TK_9            TK_NONE | 0
#define TK_10           TK_NONE | 0
#define TK_11           TK_NONE | 0
#define TK_12           TK_NONE | 0
#define TK_13           TK_NONE | 0
#define TK_14           TK_NONE | 0
#define TK_15           TK_NONE | 0
#define TK_16           TK_NONE | 0
#define TK_17           TK_NONE | 0
#define TK_18           TK_NONE | 0
#define TK_19           TK_NONE | 0
#define TK_20           TK_NONE | 0
#define TK_21           TK_NONE | 0
#define TK_22           TK_NONE | 0
#define TK_23           TK_NONE | 0
#define TK_24           TK_NONE | 0
#define TK_25           TK_NONE | 0
```

4.2. 低功耗唤醒说明

4.2.1. 唤醒 IO 设置

tk_cfg.h 文件中，在触摸按键 IO 宏定义后加上 TK_WKP_EN，即将该 IO 设置为触摸唤醒 IO，如图中，TK_0 的 IO 口是 TK_PA5，对应的阈值 50，并且设置为唤醒 IO。

用户可以根据实际需求设置唤醒 IO 个数，可以单选也可以多选，任意触摸 IO 都可以被设置为触摸唤醒 IO。

```
#define TK_NUM          6           //按键总个数
/*格式
#define TK_0            TK_PA5  | 50 | TK_WKP_EN
                        IO口    |阈值| 是否是tk唤醒IO
*/
#define TK_0            TK_PA5  | 50 | TK_WKP_EN
#define TK_1            TK_PA6  | 60
#define TK_2            TK_PA7  | 60
#define TK_3            TK_PA8  | 50
#define TK_4            TK_PA9  | 50
#define TK_5            TK_PA10 | 80
#define TK_6            TK_NONE | 0
#define TK_7            TK_NONE | 0
#define TK_8            TK_NONE | 0
#define TK_9            TK_NONE | 0
#define TK_10           TK_NONE | 0
#define TK_11           TK_NONE | 0
#define TK_12           TK_NONE | 0
#define TK_13           TK_NONE | 0
#define TK_14           TK_NONE | 0
#define TK_15           TK_NONE | 0
#define TK_16           TK_NONE | 0
#define TK_17           TK_NONE | 0
#define TK_18           TK_NONE | 0
#define TK_19           TK_NONE | 0
#define TK_20           TK_NONE | 0
#define TK_21           TK_NONE | 0
#define TK_22           TK_NONE | 0
#define TK_23           TK_NONE | 0
#define TK_24           TK_NONE | 0
#define TK_25           TK_NONE | 0
```

如果同时需要支持普通 IO 和触摸 IO 唤醒，则在上述设置之外，增加两点设置：

- 设置指定的普通 IO 为唤醒 IO，
- 调用函数接口 void tk_wakeup(u32 pin, u32 default_value)；这里两个参数的每一 bit 对应一个 IO，0 到 15 分别对应 GPIOA 的 PIN 0 到 PIN15，16 到 32 分别对应 GPIOB 的 PIN 0 到 PIN15，第一个参数是指向哪个普通 IO，第二参数是唤醒前普通 IO 的状态。

4.2.2. 关闭其他模块

关闭其他低功耗模式下不需要运行的模块。

注：timer4 如果被正常程序使用到，低功耗唤醒后需要重新初始化；

4.2.3. 调用低功耗唤醒函数

调用 tk_cfg.c 文件中的函数 tk_lp_init();

函数简单的解析如下：

```
void tk_lp_init()
{
    //IO，除触摸按键 IO 外的 IO 设置为模拟状态，注意将悬空的引脚设置为输入上拉
    …

    //clk，系统时钟切换到 lirc_256k，关闭除 lirc_256k 之外其他的时钟
    …

    tk_lp_ctrl_init();  //设置 TK 模块为低功耗模式

    //pmu，关闭 pmu 模块
    …

    tk_wakeup();  //进入睡眠，等待唤醒

    //pmu，恢复 pmu 模块
    …

    //clk，恢复时钟
    …
}
```

4.2.4. 恢复现场

当触摸按键唤醒系统之后，需要把之前关闭的其他模块（不包括触摸模块）打开，IO 恢复到初始状态。

4.3. 调试说明

4.3.1. JLINK 调试

将 tk_cfg.c 文件中宏定义 **TK_DEBUG_EN** 设置为 1，即

```
#define TK_SWD_DEBUG_EN 1
```

同时在 keil 的调试窗口中添加变量 tk_swd_buf，便可在该数组中观察到触摸按键的变化值。

4.3.2. 串口调试

基于已移植好触摸库的用户开发工程，用户需要在工程中配置三个地方

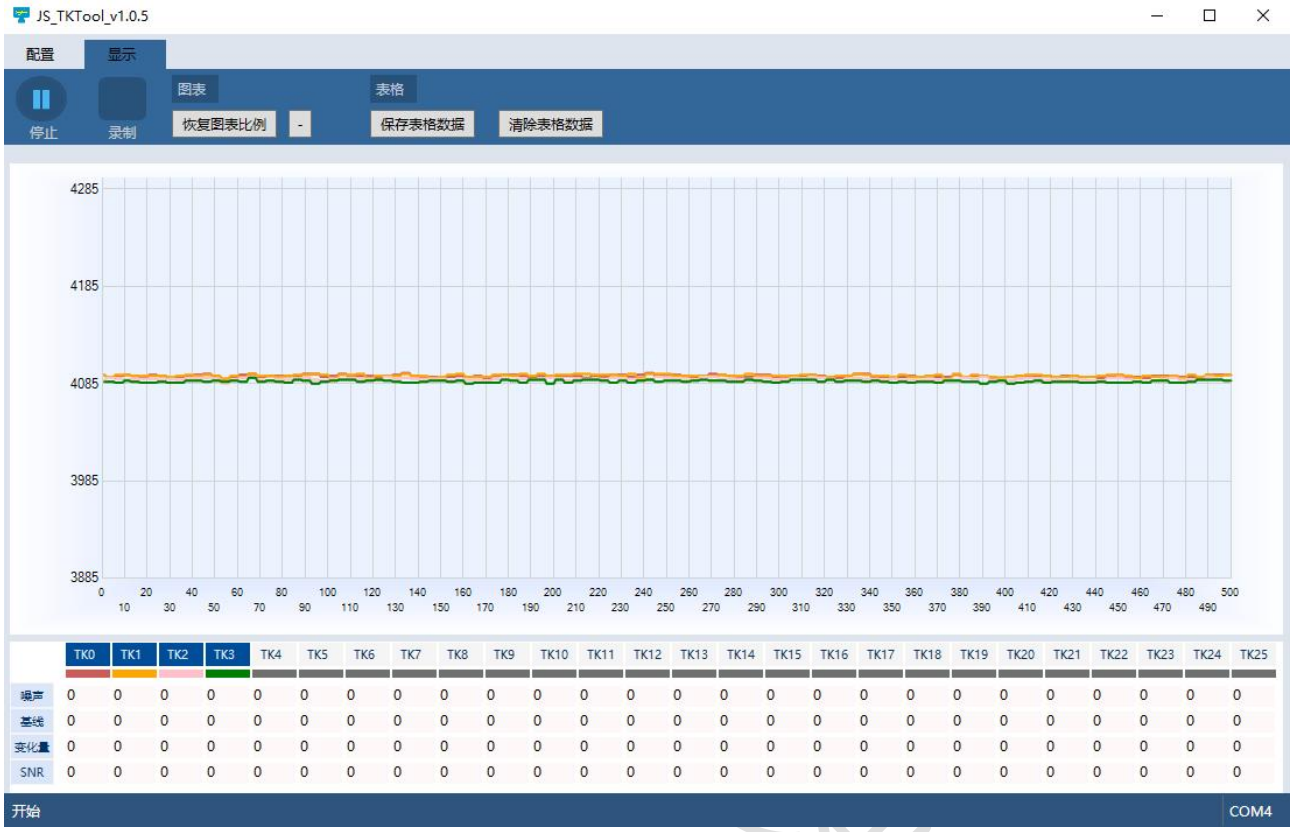
- 根据开发板实际环境，初始化 uart 接口，在 tk_uart.c 中配置宏定义 TK_UART_SEL

```
#define TK_UART_SEL    UART1_TX_PA1_RX_PA0
```

注：配置的选项在 tk_define.c 的枚举类型 TYPE_ENUM_TK_UART_SEL 中，必须使用该类型。

- 主程序调用函数 void tk_uart_init();初始化串口
- 调用函数 void tk_debug_process(void)；替换 void tk_process(void);

根据以上步骤正确配置后，编译下载到用户的开发芯片中，连接串口工具到 PC 端，同时在 PC 端打开触摸上位机，并设置好上位机参数，便可以通过图形界面看到触摸按键的采样情况，如下图所示。



注：打印数据仅做参考，打印数据呈现的效果跟打印数据，打印函数在程序中的调用位置相关。

4.4. 其他功能说明

4.4.1. 组合按键

在 tk_cfg.c 文件中，设置 tk_struct 的 two_keys_proc 和 multi_keys_proc，可以实现简单的组合按键控制。当设置为 0 的时候，功能失效；当设置为 1 的时候，功能如下：

```
tk_proc.two_keys_proc = 1; //当两个按键同时按下取变化比例大的按键值
tk_proc.multi_keys_proc = 1; //当三个或三个以上按键同时按下时，按键值为 0
```

4.4.2. 长按失键

长按失键，即长按按键，达到设置的时间自动释放按键值。长按失键的计时方式是 Systick，所以需要先配置 Systick，配置步骤如下：

- SystemTickInit(); //初始化
- 中断设置

```
void SysTick_Handler(void)
{
    SysTickCnt++;  //在中断中，用全局变量 SysTickCnt 计数,其他变量也可以
}
```

- 在 tk_cfg.c 中,将全局变量 SysTickCnt 配置到 tk_struct.systick 中

```
...
tk_struct.systick = (u32)&SysTickCnt;
...
```

- tk_proc.hold_dowm_release_time 可以设置长按释放的时间，该值小于 8，0 等于 10s，1 等于 20s，以此类推

```
tk_proc.hold_dowm_release_time = 0; //<8  0: 10s, 1: 20s,...,7:80s
```

在原先长按失键的基础上，新增一组长按失键的配置（ JS32T031 SDK V1.0.13-2024.04.07 版本新增），分为第一组长按失键按键和第二组长按失键按键。

第一组长按失键的时间配置是 tk_cfg.c 中的 tk_proc.hold_dowm_release_time，第二组长按失键的时间配置是 tk_cfg.c 中的 tk_proc.hold_dowm_release_time1，两个配置选项的值的意义一致。

如果没有配置第二组按键，默认所有已使能的按键都属于第一组；如果配置了第二组，那剩余的已使能的按键都属于第一组；配置第二组的按键的方法如下图所示，只要在对应的按键或上宏定义 TK_LPR_LV2_EN，该按键就属于第二组按键。

```
#define TK_NUM                      4  //按键总个数

/* 格式
#define TK_0                        TK_PA3 | 80 | TK_WKP_EN     | TK_LPR_LV2_EN
                                    IO口   |阈值| 是否是tk唤醒IO | 是否为长按失效第二组IO
*/

#define TK_0                        TK_PA3 | 80
#define TK_1                        TK_PA4 | 80
#define TK_2                        TK_PA5 | 80
#define TK_3                        TK_PA6 | 80
#define TK_4                        TK_NONE
#define TK_5                        TK_NONE
#define TK_6                        TK_NONE
#define TK_7                        TK_NONE
#define TK_8                        TK_NONE
#define TK_9                        TK_NONE
#define TK_10                       TK_NONE
#define TK_11                       TK_NONE
#define TK_12                       TK_NONE
#define TK_13                       TK_NONE
#define TK_14                       TK_NONE
#define TK_15                       TK_NONE
#define TK_16                       TK_NONE
#define TK_17                       TK_NONE
#define TK_18                       TK_NONE
#define TK_19                       TK_NONE
#define TK_20                       TK_NONE
#define TK_21                       TK_NONE
#define TK_22                       TK_NONE
#define TK_23                       TK_NONE
#define TK_24                       TK_NONE
#define TK_25                       TK_NONE
```

4.4.3. 滑条

原厂提供一个滑条的参考算法，源码在 tk_slip.c 文件中，使用该文件首先需要将 TK_SWD_DEBUG_EN 宏定义设置为 1，用户可以根据实际需求对算法进行修改。

4.5. 历史版本说明

SDK 版本	调试工具版本	修改点	备注
JS32T031 SDK V1.0.5-2022.07.28	V1.0.7		
JS32T031 SDK V1.0.7-2022.10.08	V1.0.7	1.修改唤醒接口	
JS32T031 SDK V1.0.9-2023.06.25	V1.0.7	1.增加普通 IO 唤醒接口（功能修改）	
JS32T031 SDK V1.0.10-2023.08.01	V1.0.9	1.基线更新机制修改 2.采用一组新的参数	**注：** 必须打开 systick，不然基线更新会失效
JS32T031 SDK V1.0.11-2023.08.30	V1.0.9	1.修改唤醒接口	
JS32T031 SDK V1.0.12-2023.10.30	V1.0.9	1.增加唤醒时间参数，修改长按释放参数（功能修改）	
JS32T031 SDK V1.0.13-2024.04.07	V1.0.9	1.增加参数接口，客户可调 2.增加长按选项（功能修改）	

4.6. 注意事项

- 按键总个数要和按键 IO 口个数一致，否则程序会返回错误，详见 4.1.2
- 不要多次调用 tk_init()；如果 tk_init()函数内的 tk_struct()返回成功会修改掉 tk 缓冲区的值，造成异常
- Timer4 如果被正常程序使用到，低功耗唤醒后需要重新初始化；详见 4.2
- 打印数据仅做参考，打印数据呈现的效果跟打印数据，打印函数在程序中的调用位置相关；详见 4.3.2
- 客户使用触摸时，必须打开在初始化的时候打开 1ms 的 systick 模块（注：调用 SystemTickInit()；该函数默认为 1ms 定时）且 systick 中断复位函数必须保留全局变量 SysTickCnt++；可参考 4.5
- 触摸的库函数中会根据外部看门狗模块是否打开，从而决定是否在采样前执行喂狗动作
- 建议客户在应用上做 50ms~100ms 的按键应用滤波

5. 优化 EFT 效果软件说明

- 芯片未使用的引脚，建议 IO 状态设置为输出高电平且驱动能力最大
- 芯片功能为输入的引脚，打开 GPIO 消抖，消抖时钟选为 32K，该设置已包含在最新的 SDK（V1.0.9）内
- 方案上若有数码管、led 灯等，建议在可接受的范围内降低亮度
- LVD 增加滤波次数

6. HIRC 配置软件说明

- 在 HIRC 使能的情况下修改 HIRC 的频率选择（32M/36M），HIRC 输出频率异常
- 芯片默认 HIRC 使能，正确的配置顺序是先将 HIRC 关闭，选择目标频率，再重新使能 HIRC