V512 触控软件包使用手册

版本：V1.00　日期：2022-11-04

www.bestsolution.com

目录

1 触控参数说明

1.1 GLOBE_VARIES.INC 参数介绍

1.1.1 参数列表

参数	功能	数值范围 (十进制)	建议值
DebounceTimes	按键去抖次数设定	0~15	2~5
AutoCalibrationPeriod	自动校准时间设定	0~15	4~8
HighSensitive	感度设定	0 = 低感度 1 = 高感度	0
MaximumKeyHoldTime	按键最长动作时间	0~15	1~3
AutoFrequencyHopping	自动跳频设定	0 = 关闭；1 = 开启	1
OneKeyActive	单键输出设定	0 = 关闭；1 = 开启	视应用功能设定
PowerSave	省电功能	0 = 关闭；1 = 开启	
MovingCalibration	动态校准设定	0 = 正常校准 1 = 动态校准	1
MainFreqSelect	触控振荡频率选择	0=3MHz 和 7MHz 1=3MHz 和 11MHz 2=1MHz 和 3MHz	默认选择 3MHz 和 11MHz，不建议串电阻
KeyNThreshold (N 为按键编号)	触发阈值	10~64	
IO_TOUCH_ATTR	触控按键或 I/O 功能切换	0=I/O 1=KEY	
WAKEUP_KEY	按键唤醒使能	0 = 休眠时对应按键无法唤醒 1 = 休眠时对应按键能够唤醒	

* 以上参数均放在 GLOBE_VARIES.INC 内，可手动修改参数值。

1.1.2 参数说明

DebounceTimes

设定去抖次数 (值越大，按键反应越慢)，以 10ms 为单位 (判断 _SCAN_CYCLEF 标志位) 计 1 数一次，每阶增加 10ms。

当 DebounceTimes=0 时，时间最短，为 60ms~80ms。

AutoCalibrationPeriod

设定校准时间，以 80ms 为单位，0=80ms；1=160ms……15=1280ms。

当设定的校准时间一到，且无按键时，即校准一次环境 (更新参考值)。

HighSensitive

设定感度高低。

0 = 低感度

1 = 高感度

MaximumKeyHoldTime

设定最长动作时间，以 4s 为单位，计算最长动作时间限制。

0 = 关闭此功能；1=4s……15=60s。

AutoFrequencyHopping

设定硬件自动跳频开启或关闭。

0 = 关闭

1 = 开启

OneKeyActive

设定单键输出功能 (不支持 4KEY 及以下的 IC)

设为 0 时，按多个触控键都有对应的按键标志输出。

设为 1 时，按多个触控键只有一个键成立，变量最大的键成立；若超过 3 个 (含) 以上的键同时被按下，则视为所有触控键都无效。

PowerSave

设定省电模式功能。

设为 0 时，关闭省电模式。

设为 1 时，无按键动作发生，8s 之后自动进入休眠模式。

MovingCalibration

设定动态更新环境功能。

设为 0 时：触控键按下后不更新“参考值”

设为 1 时：动态更新“参考值”，不论是否有触控键成立，都依当前环境进行校准。

MainFreqSelect

设定触控按键振荡频率。

设为 0 时，是选择 3MHz 和 7MHz 的触控按键振荡频率 (可以串小于 1kΩ 的电阻)。

设为 1 时，是选择 3MHz 和 11MHz 的触控按键振荡频率 (不建议串电阻)。

设为 2 时，是选择 1MHz 和 3MHz 的触控按键振荡频率。

KeyNThreshold

设定触发阈值；值越大感度越低，值越小感度越高。

建议取值为 10~64。

IO_TOUCH_ATTR

0 = 将对应的按键引脚设置成 I/O 口模式

1 = 将对应的按键引脚设置成 KEY 模式

WAKEUP_KEY

0 = 对应的按键不可以在触控的时候唤醒休眠模式

1 = 对应的按键可以在触控的时候唤醒休眠模式

1.2 函数介绍

请参考触控包 .EXT 档的内容。

1.2.1 休眠周期 RAM

_STANDBY_TIME (可读写)

进入休眠前的周期计数器，减为 0 会进入休眠；

开启休眠模式且无按键时会从 127 减至 0 (127×63ms=8s)；

如果要短时间内进入休眠，可直接将该寄存器设置为 0。

_STANDBY_TIME_CTRL (可读写)

进入休眠前的周期设定，取值范围 N (8~127)，时间为 N×63ms；

对该 RAM 写值后，进入休眠前的周期计数器的最大值会设为该数值；

用户可根据实际需求设定进入休眠前的周期时间。

1.2.2 标志位

_SCAN_CYCLEF (只能读)

所有触控键扫描完成一个周期标志位，每周期约 10ms。

_ANY_KEY_PRESSF (只能读)

任一触控键按下标志位。

_TKS_ACTIVEF (只能读)

触控软件包完成初始化、并开始运行标志位

_TKS_63MSF (只能读)

63ms 定时器标志位。

_TKS_250MSF (只能读)

250ms 定时器标志位。

_TKS_500MSF (只能读)

500ms 定时器标志位。

_FORCE_CALIBRATEF (可读写)

强制所有触控键对环境做重校准标志位。

设定该标志位为 1 时，触摸底层会执行重新校正平衡并将其清 0，当校准完成后软件包初始化完成标志位 TKS_ACTIVEF 会重新置一。

_HALT_STATE (可读写)

休眠判断标志位，根据该标志位判断能否进入休眠模式。

1.2.3 触控函数

1. “_GET_KEY_BITMAP”

功能：读取“触控键”状态

以位映射的方式输出，相对应的位为 0 = 放开；1 = 压下。

输入：无

输出：

_DATA_BUF[0]；_DATA_BUF[1]……_DATA_BUF[N]

_DATA_BUF[0]=KEY8(MSB)~KEY1(LSB)

_DATA_BUF[1]=KEY16(MSB)~KEY9(LSB)

……

_DATA_BUF[N]=KEY[8×(N+1)](MSB)~KEY(8×N+1)(LSB)

当 IC 的触控按键数 ≤ 8 个时只有 _DATA_BUF[0] 是有效数据，IC 的触控按键数 ≤ 16 个时，_DATA_BUF[0] 和 _DATA_BUF[1] 是有效数据，以 8 为一阶进行类推。

范例程序：将按键 1~8 的按键状态保存在变量 TEMP1 中，按键 9~16 的按键状态存在变量 TEMP2 中

C 语言范例：

```
GET_KEY_BITMAP();
TEMP1=DATA_BUF[0];
TEMP2=DATA_BUF[1];
```

汇编范例：

```
CALL    _GET_KEY_BITMAP
MOV     A, _DATA_BUF[0]
MOV     TEMP1, A
MOV     A, _DATA_BUF[1]
MOV     TEMP2, A
```

2. “_GET_ENV_VALUE”

功能：读取触控键“信号值”

输入：ACC=0 (第 1 KEY)；1 (第 2 KEY)……N (第 N+1 KEY)

输出：_DATA_BUF[0]

Stack：1

说明：无

范例程序：将按键 1 的信号值保存在变量 TEMP 中

C 语言范例：

```
_ACC=0;
GET_ENV_VALUE();
TEMP=DATA_BUF[0];
```

汇编范例：

```
MOV     A, 0
CALL    _GET_ENV_VALUE
MOV     A, _DATA_BUF[0]
MOV     TEMP, A
```

3. “_GET_REF_VALUE”

功能：读取触控键“参考值”

输入：ACC=0 (第 1 KEY)；1 (第 2 KEY)……N (第 N+1 KEY)

输出：_DATA_BUF[0]

Stack：1

说明：无

范例程序：将按键 1 的参考值保存在变量 TEMP 中

C 语言范例：

```
_ACC=0;
GET_REF_VALUE();
TEMP=DATA_BUF[0];
```

汇编范例：

```
MOV     A, 0
CALL    _GET_REF_VALUE
MOV     A, _DATA_BUF[0]
MOV     TEMP, A
```

4. “_GET_RCC_VALUE”

功能：读取内部的平衡电容值

输入：ACC=0 (第 1 KEY)；1 (第 2 KEY)……N (第 N+1 KEY)

输出：_DATA_BUF[0]

Stack：1

说明：无

范例程序：将按键 1 的内建电容值保存在变量 TEMP 中

C 语言范例：

```
_ACC=0;
GET_RCC_VALUE();
TEMP=DATA_BUF[0];
```

汇编范例：

```
MOV     A, 0
CALL    _GET_RCC_VALUE
MOV     A, _DATA_BUF[0]
MOV     TEMP, A
```

5. “_GET_LIB_VER”

功能：读取触控软件包版本信息

输入：无

输出：_DATA_BUF[0]；_DATA_BUF[1]

Stack：1

说明：无

范例程序：获取软件包的版本信息并存在变量 TEMP1 与 TEMP2 中

C 语言范例：

```
GET_LIB_VER();
TEMP1=DATA_BUF[0];
TEMP2=DATA_BUF[1];
```

汇编范例：

```
CALL    _GET_LIB_VER
MOV     A, _DATA_BUF[0]
MOV     TEMP1, A
MOV     A, _DATA_BUF[1]
MOV     TEMP2, A
```

6. “_GET_KEY_AMOUNT”

功能：读取触控键的总数

输入：无

输出：ACC

Stack：1

说明：无

范例程序：获取当前 IC 所有的触控按键总数并保存在变量 TEMP 中

C 语言范例：

```
GET_KEY_AMOUNT();
TEMP=_ACC;
```

汇编范例：

```
CALL    _GET_KEY_AMOUNT
MOV     TEMP1, ACC
```

7. “_SET_KEY_THR”

功能：设定指定按键的阈值

输入：将阈值赋给 DATA_BUF[0]；指定按键赋值给 ACC，0 对应 KEY1，1 对应 KEY2

输出：无

Stack：1

说明：无

范例程序：将按键 1 的阈值设为变量 TEMP 中的值

C 语言范例：

```
DATA_BUF[0]=TEMP;
_ACC=0;
SET_KEY_THR();
```

汇编范例：

```
MOV     A, TEMP
MOV     _DATA_BUF[0], A
MOV     A, 0
CALL    _SET_KEY_THR
```

8. “_LIBRARY_RESET”

功能：重新平衡环境值与参考值，清除按键状态位。

Stack：1

说明：无

C 语言范例：

```
LIBRARY_RESET();
```

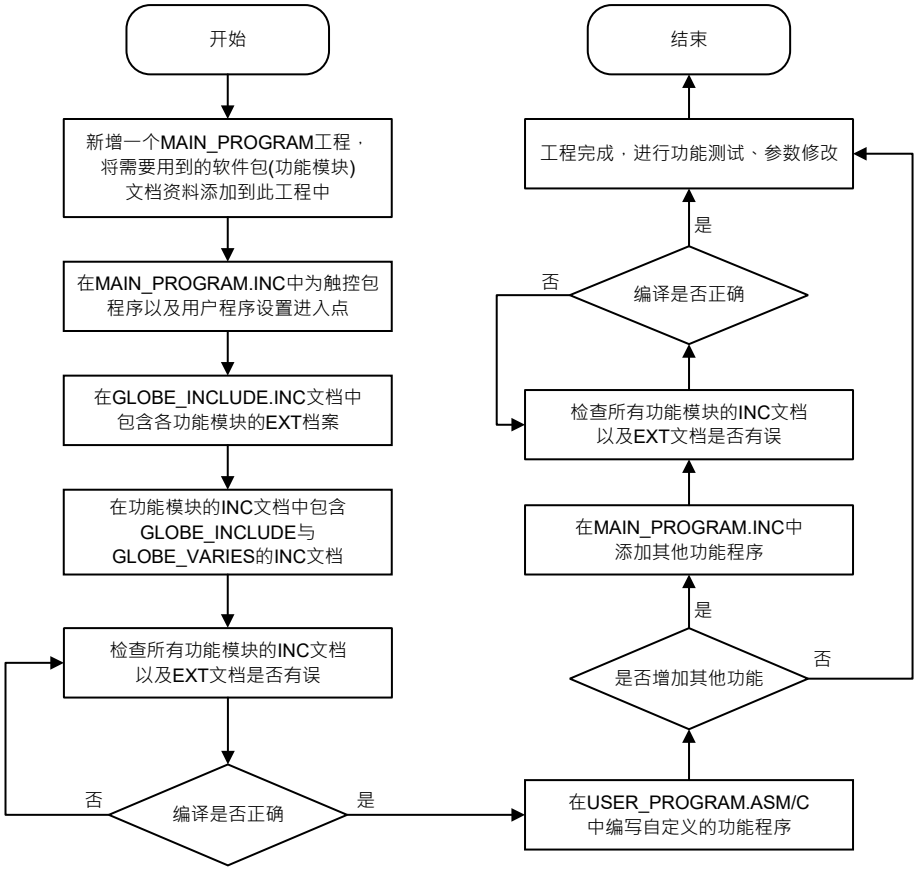
汇编范例：

```
CALL    _LIBRARY_RESET
```

* 以上函数可在触控的 EXT 档案 BSXXXXXXC_CTOUCH.EXT 中查看，若 EXT 档案中无该函数的外部声明，即表示该软件包没有此函数功能。

2 软件包使用流程

2.1 使用流程图



V512 软件包将触控功能独立出来，用户可选择是否将触控功能或者其他功能加入到软件包中。上图即为将触控功能与用户程序区功能加入到软件包后的流程图。官网下载的软件包已将触控包与用户程序功能加入，用户可直接在 USER_PROGRAM 中编写自己的程序。

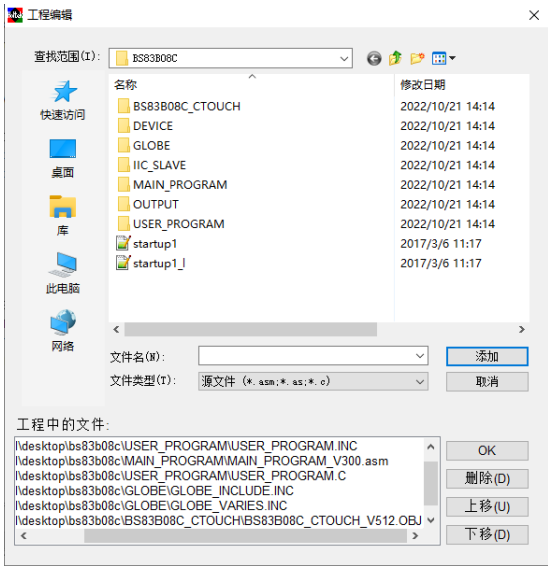
2.2 软件包使用范例

下面以 BS83B08C 为例，一步步讲述如何设计当 IC 有任何键按下时，在 PA1 输出高电平的简单应用功能。官网的 V512 软件包已完成步骤一、步骤二。

步骤一：创建触控工程

新增一个 MAIN_PROGRAM 工程，将 BS83B08C_CTOUCH 触控包、USER_PROGRAM 用户程序包的文件夹加入到软件包后，在“工程 → 编辑 → 工程编辑”选项中将 BS83B08C_CTOUCH 的 OBJ 档案、USER_PROGRAM 中的用户程序 (可选择使用汇编还是 C 语言)、以及相关的头文件加入到工程之中。

注：任何被加进来的程序都会占用 RAM/ROM，如感度测试程序 IIC_SLAVE，即使程序不调用也会占用资源。



步骤二：为主函数添加触控包与用户程序

将触控包与用户程序的 EXT 档案加入到工程中 (EXT 档案包含其外部函数声明)，在头文件 GLOBE_INCLUDE.INC 中包含 EXT 档案。在 MAIN_PROGRAM.INC 包含 GLOBE_INCLUDE.INC 与 GLOBE_VARIES.INC 两个头文件后，为触控包与用户程序设置主程序进入点，软件包会根据 1A~2F 的顺序依次执行各个类型的外部函数功能。如下图所示，当软件包进入初始化流程时，程序会先执行 1A 的触控包功能后会再执行 1D 的用户程序初始化。

```
#define MAIN_PROGRAM_DEF
#include "..\GLOBE\GLOBE_VARIES.INC"
#include "..\GLOBE\GLOBE_INCLUDE.INC"

;;;;;//-FUNCTION DEFINE                ;-FUNCTION NAME
#include "..\BS83B08C_CTOUCH\BS83B08C_CTOUCH.INI"
#define EXTEND_FUNCTION_1A_INITIAL          _BS83B08C_CTOUCH_INITIAL
#define EXTEND_FUNCTION_1A                  _BS83B08C_CTOUCH
#define EXTEND_FUNCTION_1A_HALT_PREPARE     _BS83B08C_CTOUCH_HALT_PREPARE
#define EXTEND_FUNCTION_1A_RETURN_MAIN      _BS83B08C_CTOUCH_RETURN_MAIN

;//;;;;;//--
;//#define EXTEND_FUNCTION_1C_INITIAL          _IIC_SLAVE_INITIAL
;//#define EXTEND_FUNCTION_1C                  _IIC_SLAVE
;//#define EXTEND_FUNCTION_1C_HALT_PREPARE     _IIC_SLAVE_HALT_PREPARE
;//#define EXTEND_FUNCTION_1C_HALT_WAKEUP      _IIC_SLAVE_HALT_WAKEUP
;//#define EXTEND_FUNCTION_1C_RETURN_MAIN      _IIC_SLAVE_RETURN_MAIN

;;;;;//--
#define EXTEND_FUNCTION_1D_INITIAL          _USER_PROGRAM_INITIAL
#define EXTEND_FUNCTION_1D                  _USER_PROGRAM
#define EXTEND_FUNCTION_1D_HALT_PREPARE     _USER_PROGRAM_HALT_PREPARE
#define EXTEND_FUNCTION_1D_HALT_WAKEUP      _USER_PROGRAM_HALT_WAKEUP
#define EXTEND_FUNCTION_1D_RETURN_MAIN      _USER_PROGRAM_RETURN_MAIN
```

步骤三：设置触控参数

在 GLOBE_VARIES.INC 设置软件包的各项功能与参数，具体参数含义参考 1.1 参数介绍。

```
;;;;;//-------------------------
;;;;;//-DEFINE SYSTEM CLOCK   -
;;;;;//-------------------------
#define SystemClock              0        ;;;;;//0=8MHZ
                                          ;;;;;//1=12MHZ
                                          ;;;;;//2=16MHZ
;;;;;//---------------------------------
;;;;;//-TOUCH KEY LIBRARY VARIES DEFINE
;;;;;//---------------------------------
#define CTOUCH_V512_DEF
;;;;;//-numeric operate  range 0 ~ 15;//
#define DebounceTimes            0

;;;;;//-numeric operate : range 0 ~ 15;// function: Time period to calibrat
#define AutoCalibrationPeriod    7    ;;;;;//auto calibration period select 0

;;;;;//-bit operate                   : function: Sensitive double up
#define HighSensitive            1    ;;;;;//0=Normal ;// 1=High Sensitive

;;;;;//-numeric operate ;// range 0 ~ 15: function: key holding time ,if ti
#define MaximumKeyHoldTime       2    ;;;;;//0=disable ;// 1=4 secend ......

;;;;;//-bit operate ;// range 0/1    : function: enalbe hardware hopping fun
#define AutoFrequencyHopping     1    ;;;;;//0=disable ;// 1=enable

;;;;;//-bit operate ;// range 0/1    : function: only one or all key active
#define OneKeyActive             0    ;;;;;//0=all key active ;// 1=one key a

;;;;;//-bit operate ;// range 0/1    : function: Low power consumption
#define PowerSave                1    ;;;;;//0=disable ;// 1=power save mode

;;;;;//-bit operate ;// range 0/1    : function: moving calibration signal w
#define MovingCalibration        1    ;;;;;//0=calibrate when key non press

;;;;;//-numeric operate ;// range 0 ~ 2 : function: Main frequency selectio
#define MainFreqSelect           0    ;;;;;//0:3M/7M  1:3M/11M 2:3M/1M
```

```
;;;;;//-Key threshold define
#define Key1Threshold           16     ;;;;;//suggestion range 10 ~ 64
#define Key2Threshold           16
#define Key3Threshold           16
#define Key4Threshold           16
;;;;;//--
#define Key5Threshold           16
#define Key6Threshold           16
#define Key7Threshold           16
#define Key8Threshold           16
;;;;;//--

;;;;;//---------------------------------
;;;;;//-DEFINE PIN AS I/O OR TOUCH INPUT
;;;;;//---------------------------------
#define IO_TOUCH_ATTR           00000000000000000000000111111111B;;;;;//0=IO ;// 1=TOUCH INPUT
                    ;;;;;//KEY 3   ~   2 ~ 2 ~ 1 ~ 1 ~ 0 ~ 0 ~0 ;//KEY32~KEY1
                    ;;;;;//    2       4   0   6   2   8   4  1


;//---------------------------------
;//-DEFINE WAKEUP KEY
;//---------------------------------
#define WAKEUP_KEY              00000000000000000000000111111111B;//0=DISABLE ; 1=ENABLE
                      ;//KEY 3   ~   2 ~ 2 ~ 1 ~ 1 ~ 0 ~ 0 ~0 ;KEY32~KEY1
                      ;//    2       4   0   6   2   8   4  1
```

步骤四：编译 \ 重建全部

若编译提示错误，检查所有功能程序的 INC 文档以及 EXT 是否正确。

步骤五：编写按键点灯功能

在用户程序区 USER_PROGRAM.ASM/C 内编写自定义的功能程序，其中 USER_PROGRAM_INITIAL() 为用户程序初始化函数，只在上电后执行一次。

USER_PROGRAM() 为用户工作函数，若未开启省电模式，程序会循环执行按键扫描与 USER_PROGRAM()。在程序区判断 TKS_ACTIVEF 为 1 的作用为确保软件包初始化完成，SCAN_CYCLEF 为 1 表示完成一轮按键扫描，ANY_KEY_PRESSF 为 1 表示有任意按键按下。

```
void USER_PROGRAM_INITIAL()
{
    _pac1 = 0;
}

//================================================
//************************************************
//================================================

void USER_PROGRAM()
{
    if(TKS_ACTIVEF && SCAN_CYCLEF)
    {
        if(ANY_KEY_PRESSF)
        _pa1 = 1;
        else
        _pa1 = 0;
    }
}
```

USER_PROGRAM.C 范例

```
                    ;;************************
_USER_PROGRAM_INITIAL:;;* USER_PROGRAM_INITIAL *
                    ;;************************
                CLR     PAC1
                RET

;;*************************************************************
;;*SUB. NAME:USER_PROGRAM                                     *
;;*INPUT    :                                                 *
;;*OUTPUT   :                                                 *
;;*USED REG.:WORKING MODE DO THIS                             *
;;*FUNCTION :                                                 *
;;*************************************************************
                ;;*********************
_USER_PROGRAM:  ;;USER PROGRAM ENTRY *
                ;;*********************
                SNZ   _TKS_ACTIVEF
                RET
                SNZ   _SCAN_CYCLEF
                RET

                ;;在這下面編寫用戶的功能代碼
                SZ    _ANY_KEY_PRESSF
                SET   PA1
                SNZ   _ANY_KEY_PRESSF
                CLR   PA1

                RET
```

USER_PROGRAM.ASM 范例

步骤六：功能测试

将档案编译后烧录到 IC 中，按下任意按键观察 PA1 是否输出高电平。若功能异常可检测上述步骤的配置是否有问题，若按键感度不足可以减小按键阈值。

3 用户函数说明

3.1 初始化函数

当 IC 上电后，软件包会依次执行触控包初始化和用户初始化函数。位于用户程序区内的 USER_PROGRAM_INITIAL() 为用户初始化函数，用户可在该函数内编写初始化程序。初始化函数只会在上电后执行一次。软件包会自动配置系统时钟与看门狗、并开启总中断 EMI，因此用户不需要额外配置。

3.2 工作函数

当 IC 执行完初始化后，程序会循环执行所有的外部工作函数，即循环执行底层触控函数与用户写在 USER_PROGRAM() 中的函数。若用户没有开启省电模式，软件包会一直在各个工作函数间循环执行。软件包会自动清看门狗并重新加载时钟，无需用户手动清看门狗。

若用户开启了省电模式，软件包的 STANDBY_TIME 会从 7FH 不断向下计数。当计数值为 00H 后软件包将会进入休眠流程。进入休眠前若有按键按下，计数值将维持在 7FH 直到无按键按下后重新向下计数。用户可通过直接修改 STANDBY_TIME 为 00H 达到快速休眠的目的。若用户希望修改休眠周期，可修改 STANDBY_TIME_CTRL 的值来实现，详细可参考 1.2.1 休眠周期 RAM 章节。当进入休眠流程后，软件包便会开始执行休眠函数。

3.3 休眠函数

3.3.1 休眠函数标志位说明

V512 软件包若开启省电模式，在 MAIN_PROGRAM.ASM 中会根据休眠标志位 HALT_STATE 来判断软件包能够进入休眠从而省电。在执行休眠函数前会 SET HALT_STATE 让该标志位的最高位与最低位变为 1，用户可在休眠函数中修改 HALT_STATE 标志位来控制软件包能否执行 HALT 休眠。以下为休眠标志位最高位与最低位的解释说明。

1. HALT_STATE.7 (可读写)

该位为休眠标志位 RAM 操作有效位。若 HALT_STATE.7 为 0 则视为休眠标志位有效，软件包会根据 HALT_STATE.0 决定软件包能否进入休眠。执行完休眠函数后 HALT_STATE.7 为 1 则操作无效。因此若要改变休眠状态必须将 HALT_STATE.7 清零。

2. HALT_STATE.0 (可读写)

该位为程序能否休眠判断位。若该位为 1 表示程序满足休眠条件，允许进入休眠。若该标志位为 0 则表示程序不满足休眠条件，程序会返回工作流程。因此，将 HALT_STATE 清零的时候软件包将无法进入休眠。

3.3.2 休眠函数定义说明

V512 软件包新添加了三个休眠函数供用户使用，这三个休眠函数放在工作函数下方，这三个函数的功能分别为：

1. USER_PROGRAM_HALT_PREPARE()

休眠准备函数：IC 将进入休眠前执行

2. USER_PROGRAM_HALT_WAKEUP()

休眠唤醒函数：IC 从休眠模式被中断、I/O 唤醒之后执行

3. USER_PROGRAM_RETURN_MAIN()

休眠返回工作函数：IC 从休眠模式返回工作模式时执行

3.3.3 休眠函数使用说明

1. USER_PROGRAM_HALT_PREPARE()

休眠前准备函数：当进入休眠前的周期时间减为 0 或看门狗溢出复位后会执行该函数。该函数的作用为关闭部分耗电功能降低功耗，如 A/D 采集、I/O 口配置等其他耗电配置。

当程序从工作模式进入休眠时或者休眠时看门狗溢出时，软件包会执行该休眠前准备函数，此时触控包底层会先进行一次按键扫描，即在执行该函数前会判断有无按键成立。

若有按键按下，软件包触控底层函数会将休眠标志位 HALT_STATE 清零，从而让 IC 进入唤醒流程，若无按键按下则不会对 HALT_STATE 进行操作。在 USER_PROGRAM_HALT_PREPARE，若用户希望特定条件下程序持续工作，可在该函数内满足某条件时将 HALT_STATE 清零，如此软件包将不会进入休眠。

2. USER_PROGRAM_HALT_WAKEUP()

休眠唤醒函数：当 IC 进入休眠模式后，被中断或 I/O 口唤醒后会执行该函数。在执行完该函数后程序会重新进入休眠，不会主动返回工作模式。

若用户希望软件包在被中断或 I/O 口唤醒后返回工作模式，可在 USER_PROGRAM_HALT_WAKEUP 函数中将 HALT_STATE 清零，代码如下：

```
void USER_PROGRAM_HALT_WAKEUP()
{
  HALT_STATE=0;
}
```

3. USER_PROGRAM_RETURN_MAIN()

休眠返回工作函数：当程序在执行完休眠前准备函数或休眠唤醒函数后，若 HALT_STATE 不满足休眠条件，软件包会执行返回工作函数，软件包底层在此会将所有的省电配置复原从而恢复触控的所有功能。用户需在此将之前进入休眠前省电关闭的功能重新开启。

4 补充说明

4.1 软件包使用注意事项

1. 程序不能长时间执行一个功能

如果软件包长时间执行一个功能，软件包将无法执行其他功能函数 (包括触控函数)。软件包的主循环函数为 MAIN_PROGRAM，因此不能在 USER_PROGRAM 中写 while(1) 循环。所有的触控函数必须跑底层才可以正常执行。

2. 用户编写的程序不可以延时太长时间

如果用户程序延时超过 10ms 以上可能会导致按键性能变差或误触发。若客户要长时间延时的话，建议关闭总中断或是 TB0 中断暂停触控功能，在执行完耗时代码后再重新开启。

3. 定时器中断标志发生，程序不一定立即执行

若通过定时器定义时间标志执行功能操作时，由于软件包不一定在执行该功能区域，因此时间不一定百分百精准，若程序要求十分精准的计时，可将这部分程序放在定时器中断中运行。

4. 系统频率和看门狗设置

软件包在执行完所有的底层按键扫描和 USER_PROGRAM 程序后，软件包会自动清看门狗以及重新加载 GLOBE_VARIES.INC 的 SystemClock 系统频率。因此用户不用主动清看门狗和配置系统时钟。

软件包初始化会自动设置一次看门狗时间，用户也可在用户初始化函数中自己配置。软件包休眠后是通过看门狗溢出来做按键扫描的，例如看门狗时间若设置为 128ms，软件包会间隔 128ms 起来做按键扫描。看门狗溢出时间越短休眠唤醒越灵敏，但是休眠功耗会增大。

4.2 触控包标志位使用技巧

1. 触控功能初始化完成标志位 TKS_ACTIVEF

软件包在上电后，触控功能需要一点时间才能完成初始化。因此需要用户在主程序中判断 TKS_ACTIVEF 标志位再使用触控功能，汇编的范例如下：

```
;;**********************
_USER_PROGRAM:  ;;USER PROGRAM ENTRY *
                ;;**********************
                SNZ   _TKS_ACTIVEF
                RET

                ;;在這下面編寫用戶的功能代碼
                SZ    _ANY_KEY_PRESSF
                SET   PA1
                SNZ   _ANY_KEY_PRESSF
                CLR   PA1

                RET
```

程序作用为：若触控包没有初始化完成，TKS_ACTIVEF 等于 0，程序会直接执行 RET 结束用户程序，从而避免因为软件包没有初始化完成导致的功能异常。当触控功能初始化完成、TKS_ACTIVEF 等于 1 后，再执行下方的用户功能代码。

2. 按键周期扫描标志位 SCAN_CYCLEF

触控底层在扫描完所有的按键后，会将 SCAN_CYCLEF 标志位置一，在下一轮按键扫描开始时再将 SCAN_CYCLEF 清零。用户可通过判断该标志位确保全部按键都扫描完成再做按键操作，范例代码如下所示：

```
_USER_PROGRAM:  ;;*********************
                ;;USER PROGRAM ENTRY *
                ;;*********************
                SNZ   _TKS_ACTIVEF
                RET
                SNZ   _SCAN_CYCLEF
                RET

                ;;在這下面編寫用戶的功能代碼
                SZ    _ANY_KEY_PRESSF
                SET   PA1
                SNZ   _ANY_KEY_PRESSF
                CLR   PA1

                RET
```

程序作用为：当触控包初始化完成后，只有扫描完所有的按键之后程序才会执行下方的用户功能代码。这样可以避免由于触控底层未扫描全部的按键就对按键做判断从而导致的异常问题。以上两个标志位判断换成 C 语言表示的范例代码如下：

```
void USER_PROGRAM()
{
    if(TKS_ACTIVEF && SCAN_CYCLEF)
    {
        if(ANY_KEY_PRESSF)
        _pa1 = 1;
        else
        _pa1 = 0;
    }
}
```

3. 触控包时间扫描标志位

触控包本身具备了 _TKS_63MSF、_TKS_250MSF、_TKS_500MSF 的时间标志位，当时间计数值达到设定值后该标志位会置一，等下次执行用户程序时会自动清零。用户可在 USER_PROGRAM 程序中通过判断该标志位是否为 1 来计算时间。

4.3 软件包配置注意事项

1. 指定正确的路径

在 .INC 或 .H 中包含其他文件夹内的相关 INI、EXT 文档时，需要有包含路径宣告。例如，在 GLOBE_INCLUE.INC 文档内，#include "..\USER_PROGRAM\USER_PROGRAM.EXT" 表示包含位于 USER_PROGRAM 文件夹内的 USER_PROGRAM.EXT 文档，其中的 .. 表示更上层的路径。

2. 程序进入点宣告

在 MAIN_PROGRAM.INC 文档中用户可为程序设置进入点宣告，例如触控包底层函数或者 USER_PROGRAM 函数就是通过宣告后加入主程序中。用户若不使用某部分函数功能，可通过注释对应的进入点宣告从而节省 RAM 空间。

3. 程序地址宣告

程序地址 (含中断地址) 采用绝对地址宣告，不可用相对地址 ORG 的方式宣告。

绝对地址：

```
;===============
PROGRAM_ENTRY   .SECTION  AT 000H 'CODE'
                JMP      PROGRAM_RESET
```

√

相对地址：

```
ORG     000H
JMP     PROGRAM_RESET
```

×

例如，在 USER_PROGRAM.ASM 文件中定义 INT 中断子程序地址要如下图这样定义，如此一来当外部中断发生时，程序会执行用户自定义的 INT 子程序内的程序。

```
INT_ENTRY   .SECTION  AT  004H
            JMP     INT
```

4.4 软件包测量电容值

V512 软件包支持测量 0~50pF 的电容，只需将电容接到按键引脚与地之间，通过高两位低八位的内建电容值可以计算外接电容的容值，计算公式为：

$$C = \frac{\text{内建电容值} \times 50pF}{1024}$$