# V512 Touch Library User Guide

Revision: V1.00  Date: November 04, 2022

www.bestsolution.com

# Table of Contents

# 1 Touch Parameter Description

## 1.1 GLOBE_VARIES.INC Parameter Introduction

### 1.1.1 Parameter List

| Parameter | Function | Range (Decimal Value) | Suggested Value |
|---|---|---|---|
| DebounceTimes | Key debounce times | 0~15 | 2~5 |
| AutoCalibrationPeriod | Auto calibration period | 0~15 | 4~8 |
| HighSensitive | Sensitivity setup | 0=Low sensitivity<br>1=High sensitivity | 0 |
| MaximumKeyHoldTime | Maximum key holding time | 0~15 | 1~3 |
| AutoFrequencyHopping | Auto frequency hopping | 0=Off; 1=On | 1 |
| OneKeyActive | Single key output | 0=Off; 1=On | Depends on application function |
| PowerSave | Power-saving | 0=Off; 1=On | |
| MovingCalibration | Dynamic calibration | 0=Normal calibration<br>1=Dynamic calibration | 1 |
| MainFreqSelect | Touch key oscillator frequency selection | 0=3MHz and 7MHz<br>1=3MHz and 11MHz<br>2=1MHz and 3MHz | Defaults to 3MHz and 11MHz, no resistor recommended |
| KeyNThreshold (N is the key serial number) | Key trigger threshold | 10~64 | |
| IO_TOUCH_ATTR | Touch key or I/O functional switching | 0=I/O<br>1=KEY | |
| WAKEUP_KEY | Key wake-up control | 0=cannot be woken up by key during sleep<br>1=can be woken up by key during sleep | |

* These parameters are located in the GLOBE_VARIES.INC file, their setting values can be modified.

### 1.1.2 Parameter Description

#### DebounceTimes

This parameter sets the debounce times. The larger the value, the slower the key response. The counter counts once every 10ms by polling the _SCAN_CYCLEF flag. The debounce time increases by 10ms for each count.

DebounceTimes=0 sets the shortest debounce time, which is 60ms~80ms.

#### AutoCalibrationPeriod

This parameter sets the calibration period.

The units here are 80ms, 0=80ms, 1=160ms……15=1280ms.

When the preset calibration period is reached and there is no key pressed, an ambient calibration is executed once, which will update the reference values.

#### HighSensitive

This parameter sets the sensitivity.

0=low sensitivity

1=high sensitivity

#### MaximumKeyHoldTime

This parameter sets the maximum key holding time limit in units of 4s.

0=disable; 1=4s……15=60s

#### AutoFrequencyHopping

This parameter controls the hardware auto frequency hopping function.

0=off

1=on

#### OneKeyActive

This parameter sets the single key output function. Note that a device with 4 or less touch keys does not support this function.

0=off – when multiple touch keys are pressed, there are corresponding flag outputs.

1=on – when multiple touch keys are pressed, only the key with the largest variable value is considered valid.

If this parameter is set to 1 and three or more keys are simultaneously pressed, all touch keys are considered invalid.

#### PowerSave

This parameter sets the power-saving function.

0=turn off power-saving mode.

1=if no key is pressed, enter the sleep mode automatically after 8 seconds.

#### MovingCalibration

This parameter sets the dynamic updating ambient function.

0=do not update the reference value when a touch key is pressed.

1=update the reference value dynamically regardless of whether there is a touch key action or not.

#### MainFreqSelect

This parameter sets the touch key oscillator frequency.

0=3MHz and 7MHz, a resistor of less than 1kΩ is recommended

1=3MHz and 11MHz, no resistor recommended

2=1MHz and 3MHz

#### KeyNThreshold

This parameter sets the key trigger threshold. The larger the value, the lower the sensitivity. The smaller the value, the higher the sensitivity.

The recommended value has a range of 10~64.

#### IO_TOUCH_ATTR

0=set the pin to the I/O mode

1=set the pin to the KEY mode

#### WAKEUP_KEY

0=the key will not wake up the device from the sleep mode when pressed

1=the key will wake up the device from the sleep mode when pressed

## 1.2 Function Introduction

Refer to the contents of the .EXT file in the touch library.

### 1.2.1 Sleep Period RAM

#### _STANDBY_TIME (read/write)

This is the period counter before entering the sleep mode. The system will enter the sleep mode when this counter decreases to zero.

When the sleep mode has been enabled and no key has been pressed, the counter will count down from a preset value (max. 127) to zero, 127×63ms=8s.

To quickly enter the sleep mode, set this register to 0 directly.

#### _STANDBY_TIME_CTRL (read/write)

This is the period setting before entering the sleep mode. N=8~127, period=N×63ms.

After a value has been written here, the maximum value of the period counter will be equal to this value.

Users can set this period time according to their requirements.

### 1.2.2 Flags

#### _SCAN_CYCLEF (read only)

This flag, if set high, indicates that all touch keys have been scanned for one cycle. 1 cycle=10ms.

#### _ANY_KEY_PRESSF (read only)

This flag, if set high, indicates that any key has been pressed.

#### _TKS_ACTIVEF (read only)

This flag, if set high, indicates that the touch library has completed it's initialisation and has started running.

#### _TKS_63MSF (read only)

63ms timer flag.

#### _TKS_250MSF (read only)

250ms timer flag.

#### _TKS_500MSF (read only)

500ms timer flag.

#### _FORCE_CALIBRATEF (read/write)

When this flag is set high, the touch layer will calibrate the ambient values for all the touch keys and clear this flag to zero. When the calibration has been completed the touch library initialisation completed flag TKS_ACTIVEF will be set high again.

#### _HALT_STATE (read/write)

Sleep flag, which indicates whether to enter the sleep mode.

### 1.2.3 Touch Functions

#### 1. "_GET_KEY_BITMAP"

Function: read the touch key state

Output is in a bitmap type, bit=0 – key not pressed, bit=1 – key pressed

Input: none

Output:

_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)

If the number of touch keys is less than or equal to 8, only _DATA_BUF[0] is valid data. If the number of touch keys is less than or equal to 16, _DATA_BUF[0] and _DATA_BUF[1] are valid data and so on.

Program example: store the states of keys 1~8 in the variable TEMP1 and the states of keys 9~16 in the variable TEMP2.

C code:

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

ASM code:

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

### 2. "_GET_ENV_VALUE"

Function: read the touch key signal value

Input: ACC=0 (KEY1); 1 (KEY2)……N (KEY(N+1))

Output: _DATA_BUF[0]

Stack: 1

Description: none

Program example: store the signal value of KEY1 in the variable TEMP.

C code:

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

ASM code:

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

### 3. "_GET_REF_VALUE"

Function: read the touch key reference value

Input: ACC=0 (KEY1); 1 (KEY2)……N (KEY(N+1))

Output: _DATA_BUF[0]

Stack: 1

Description: none

Program example: store the reference value of KEY1 in the variable TEMP.

C code:

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

ASM code:

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

**4. "_GET_RCC_VALUE"**

Function: read the key internal balance capacitance value

Input: ACC=0 (KEY1); 1 (KEY2)……N (KEY(N+1))

Output: _DATA_BUF[0]

Stack: 1

Description: none

Program example: store the internal capacitance value of KEY1 in the variable TEMP.

C code:

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

ASM code:

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

**5. "_GET_LIB_VER"**

Function: read the touch library version information

Input: none

Output: _DATA_BUF[0]; _DATA_BUF[1]

Stack: 1

Description: none

Program example: obtain the library version information and store it into variables TEMP1 and TEMP2.

C code:

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

ASM code:

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

**6. "_GET_KEY_AMOUNT"**

Function: read the total number of touch keys

Input: none

Output: ACC

Stack: 1

Description: none

Program example: obtain the total number of touch keys of the current device and store it in the variable TEMP.

C code:

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

ASM code:

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

### 7. "_SET_KEY_THR"

Function: set the threshold value of a specific key

Input: assign the threshold value to DATA_BUF[0]; assign the key number to ACC, for example, 0=KEY1, 1=KEY2……

Output: none

Stack: 1

Description: none

Program example: set the threshold for KEY1 to the value in the variable TEMP.

C code:

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

ASM code:

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

### 8. "_LIBRARY_RESET"

Function: rebalance the ambient and reference values and clear the key state bits

Stack: 1

Description: none

C code:

```
LIBRARY_RESET();
```

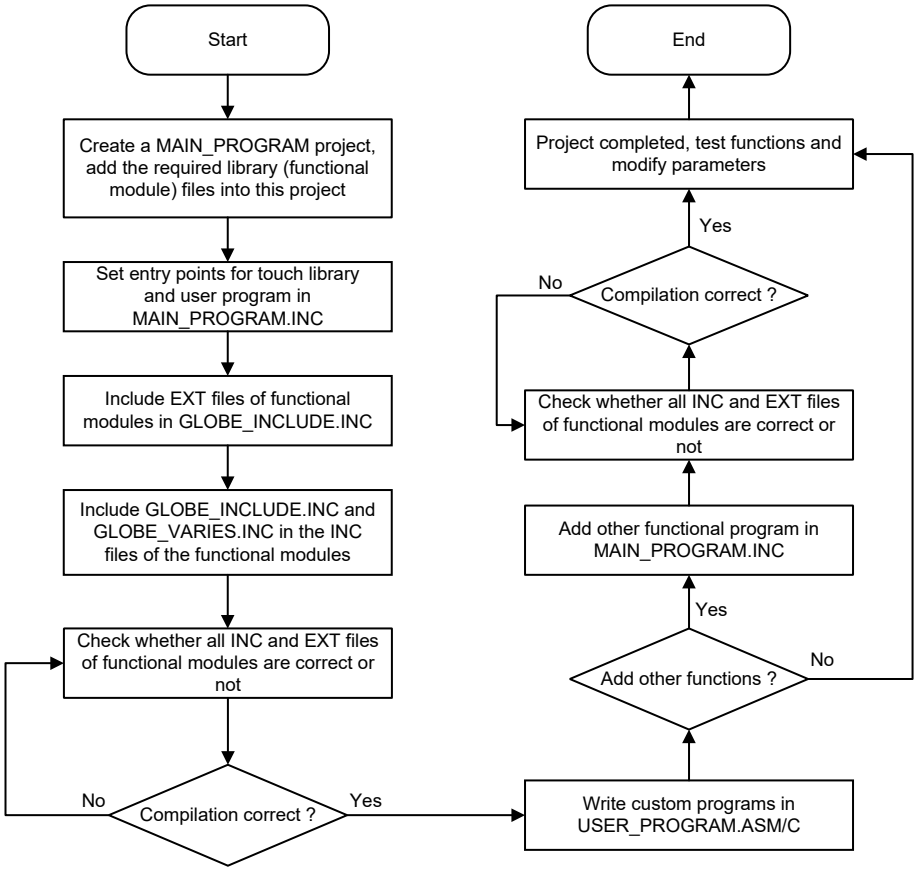
ASM code:

```
CALL        _LIBRARY_RESET
```

* The above functions are viewable in the BSXXXXXXC_CTOUCH.EXT file. If there is no external declaration of the function in the EXT file, this means that the library does not have this function.

# 2 Library Usage Flow

## 2.1 Usage Flowchart




The V512 library has the touch function separated. Users can choose whether to add touch or other functions into the library. The figure above shows the flowchart after adding the touch function and other user programs to the library. The library downloaded from the official website contains the touch library and the user program folders. Users can write their own programs directly in USER_PROGRAM.

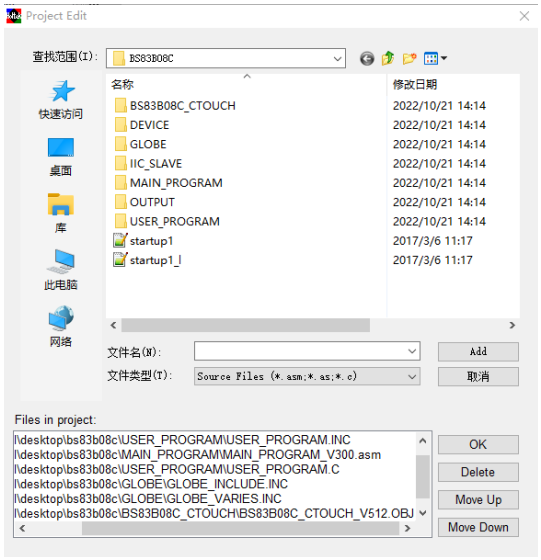
## 2.2 Library Usage Example

Taking the BS83B08C as an example, the following will describe how to design a simple application function that outputs a high level on PA1 when any key is pressed. The V512 library on the official website has completed Steps 1 and 2.

### Step 1: Create a touch project

Create a MAIN_PROGRAM project. Add the BS83B08C_CTOUCH and USER_PROGRAM folders to the library. Then through “Project→Edit→Project Edit” add the BS83B08C_CTOUCH OBJ file, user program (.asm or .c file selectable) in the USER_PROGRAM folder and relevant header files into the project.

Note: any program that is added will consume a certain amount of RAM and ROM space. For example, even if the sensitivity test program IIC_SLAVE is not called, it will also consume memory resources.



### Step 2: Add the touch library and user programs to the main program

Add the EXT files of the touch library and user programs, which include the external function declarations, into the project. Then include these EXT files in the GLOBE_INCLUDE.INC head file. Include the GLOBE_INCLUDE.INC and GLOBE_VARIES.INC header files in MAIN_PROGRAM.INC and then set entry points for the touch library and user programs. The library will execute each type of external functions sequentially according to the order, 1A~2F. As shown in the figure below, when entering the initialisation process, the program will first execute the touch functions of 1A, and then execute the user program initialisation of 1D.

```
#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
```

### Step 3: Set touch parameters

Set the functions and parameters in GLOBE_VARIES.INC and refer to the 1.1 section for specific parameter meanings.

```
;;;;;//-------------------------
;;;;;//-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           00000000000000000000000011111111B;;;;;//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              00000000000000000000000011111111B;//0=DISABLE ; 1=ENABLE
                      ;//KEY 3   ~   2 ~ 2 ~ 1 ~ 1 ~ 0 ~ 0 ~0 ;KEY32~KEY1
                      ;//    2       4   0   6   2   8   4  1
```

### Step 4: Build\Rebuild All

If the compiler has generated any errors, check whether the INC and EXT files for all functional programs are correct or not.

### Step 5: Write the key-controlled lighting function

Create the custom functional programs in USER_PROGRAM.ASM/C, where USER_PROGRAM_INITIAL() is the user program initialisation function. This is executed only once after power-on.

USER_PROGRAM() is a user work function. If the power-saving mode has not been turned on, the program will loop through the key scanning and USER_PROGRAM(). In the program area, checking whether TKS_ACTIVEF is 1 will ensure that the touch library initialisation has been completed. If SCAN_CYCLEF is 1, this means that a key scanning cycle has been completed. If ANY_KEY_PRESSF is 1, this means that a key has been pressed.

```
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 Example**

```
                    ;;*************************
_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 Example**

### Step 6: Functional test

Compile the file and program it into the device, press any key to observe if PA1 outputs a high level. If the function is abnormal, detect whether there is a problem with the configuration of the above steps and if the key sensitivity is insufficient. If so then reduce the key threshold.

# 3 User Function Description

## 3.1 Initialisation Functions

After the device has been powered on, the touch library initialisation and user initialisation functions will be executed in turn. The USER_PROGRAM_INITIAL() located in the user program area is the user initialisation function, in which the user can write the initialisation program. These initialisation functions are executed only once after power-on. The library will automatically configure the system clock and watchdog, and turn on the global interrupt EMI, so no additional relevant user configurations are required.

## 3.2 Work Functions

After all initialisation functions are completed, the program will loop through all external work functions. This will loop through the underlying touch functions and the user functions written in USER_PROGRAM(). If the power-saving mode has not been turned on by the user, the library will always loop through the various work functions. The library will also automatically clear the watchdog and reload the clock, eliminating the need for the user to manually clear the watchdog.

If the power-saving mode has been turned on by the user, the STANDBY_TIME counter will count down from 7FH. When the counter value reaches 00H, the library will enter the sleep mode. If a key is pressed before entering the sleep mode, the counter value will remain at 7FH until no key is pressed and then count down again. Users can directly modify STANDBY_TIME to 00H to implement a faster sleep mode entry. For details, refer to the Sleep Period RAM section. After entering the sleep mode, the library will start to execute the sleep functions.

## 3.3 Sleep Functions

### 3.3.1 Sleep Function Flag

If the V512 library has the power-saving mode enabled, the MAIN_PROGRAM.ASM will determine whether to enter the sleep mode to save power according to the HALT_STATE flag. Before executing the sleep function, the highest and lowest bits of the HALT_STATE register will be set high. In the sleep function users can modify the HALT_STATE register to control the program whether to execute the HALT instruction to enter the sleep mode. The following explains these highest and lowest bits.

#### 1. HALT_STATE.7 (read/write)

This flag indicates whether it is valid to operate the sleep flag RAM. If HALT_STATE.7 is 0, this means the sleep flag is valid. Then the software will determine whether to enter the sleep mode according to the HALT_STATE.0 bit. If HALT_STATE.7 is 1 after executing the sleep function, it is invalid to operate the sleep flag. To change the sleep state, HALT_STATE.7 must be cleared to zero.

#### 2. HALT_STATE.0 (read/write)

This flag indicates whether the program can enter the sleep mode or not. If this bit is 1, this means the program meets the sleep conditions and is allowed to enter the sleep mode. If this bit is 0, the program does not meet the sleep conditions and the program will return to the main program. Therefore, the program is not allowed to enter the sleep mode if HALT_STATE is cleared.

### 3.3.2 Sleep Function Definition

The V512 library has three additional user sleep functions, which are placed under the work functions.

#### 1. USER_PROGRAM_HALT_PREPARE()

Sleep mode preparation function: executed before the device enters the sleep mode.

#### 2. USER_PROGRAM_HALT_WAKEUP()

Sleep mode wake-up function: executed after the device has been woken up by an interrupt or an I/O port.

#### 3. USER_PROGRAM_RETURN_MAIN()

Sleep-return-to-work function: executed when the device returns to the working mode from the sleep mode.

### 3.3.3 Sleep Function Usage

#### 1. USER_PROGRAM_HALT_PREPARE()

This function is executed when the period time before entering sleep is reduced to 0 or when the watchdog overflow reset occurs. This function is designed to turn off some functions to reduce power consumption, such as A/D sampling, I/O port and other configurations.

When the program enters the sleep mode from the working mode or when the watchdog overflows during the sleep mode, the library will execute the sleep mode preparation function. At the same time, a key scanning cycle will first be executed. This is to check whether any key has been pressed before executing the sleep mode preparation function.

If a key has been pressed, the underlying functions in the touch library will clear the HALT_STATE register to zero allowing the device to enter the wake-up process. If no key has been pressed, HALT_STATE will not be cleared. In the USER_PROGRAM_HALT_PREPARE function, if the user wants the program to continue working under certain conditions, clear the HALT_STATE register to zero when certain function conditions have been met, so that the program will not enter the sleep mode.

#### 2. USER_PROGRAM_HALT_WAKEUP()

This function is executed after the device has been woken up from the sleep mode by an interrupt or an I/O port. After executing the function, the program will re-enter the sleep mode and will not actively return to the working mode.

If the user wants the program to return to the working mode after being woken up by an interrupt or an I/O port, clear the HALT_STATE register in the USER_PROGRAM_HALT_WAKEUP function, as shown below.

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

#### 3. USER_PROGRAM_RETURN_MAIN()

After the sleep mode preparation function or the sleep mode wake-up function has been executed, if HALT_STATE does not meet the sleep conditions, the program will execute this return function. Here the library will restore all power-saving settings to their original settings to restore all touch functions. Other functions that have been turned off before entering the sleep mode should be turned on manually again in this function.

# 4 Additional Notes

## 4.1 Library Usage Considerations

### 1. The program cannot execute a function for a long time.

If the library executes a function for a long time, it may be unable to execute other functions, including touch functions. The main loop of the library is MAIN_PROGRAM, here users must not insert a while(1) loop in USER_PROGRAM. All touch functions must be executed at the lower layer to operate properly.

### 2. The user-written programs should not contains too long delays.

If a user program delay exceeds 10ms, the key performance may deteriorate or trigger incorrectly. If the user wants to delay for a longer time, it is recommended to turn off the global interrupt or the TB0 interrupt to pause the touch function and then turn it back on after the delay code has been executed.

### 3. The desired functional program may not be executed immediately when a timer interrupt flag is set high.

When a function is performed by defining a time flag via a timer, the desired time period is not necessarily 100% accurate because the library may not be running in the current functional area. If the program requires very accurate timing, the desired part of program should be arranged to be in the timer interrupt subroutine.

### 4. System frequency and watchdog setting

After the library has executed all the underlying key scanning and USER_PROGRAM programs, it will automatically clear the watchdog and reload the SystemClock frequency defined in GLOBE_VARIES.INC. Therefore, the user does not have to actively clear the watchdog and configure the system clock.

The library initialisation function will automatically set the watchdog time-out period, which can also be configured by the user in the user initialisation function. After entering the sleep mode, the library executes key scanning when encountering watchdog overflow events. For example, if the watchdog time-out period is set to 128ms, the library will wake up every 128ms to execute key scanning. The shorter the watchdog time-out period, the more sensitive the sleep mode wake-up will be, however will result in higher power consumption.

## 4.2 Tips for Use of Touch Library Flags

### 1. Touch function initialisation completed flag – TKS_ACTIVEF

After power-on, it takes a certain amount of time for the touch function to execute its initialisation. Therefore, the user needs to check the TKS_ACTIVEF flag before using the touch function. The ASM example code is shown below.

```
                ;;**********************
_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
```

Code function: if the touch library initialisation has not completed, the TKS_ACTIVEF flag is equal to 0 and the program will directly execute the RET instruction to finish the user program to avoid functional exceptions caused by the library not being initialised. When the touch function has been initialised and TKS_ACTIVEF is equal to 1, the program will execute the follow-up user function code.

### 2. Touch key periodic scan flag – SCAN_CYCLEF

After scanning all the keys, the touch layer will set the SCAN_CYCLEF flag high and clear the flag to zero at the beginning of the next key scanning cycle. Users can check this flag to ensure that all keys have been scanned and then execute the key function, as shown in the example below.

```
                ;;**********************
_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
```

Code function: when the touch library has been initialised, the program will not execute the follow-up user function code until all keys have been scanned. This avoids abnormal problems caused by the touch layer judging the key state without scanning all the keys. The C example code for the above two flag judgements is as follows.

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

### 3. Touch library timer flags

The touch library has defined the _TKS_63MSF, _TKS_250MSF, _TKS_500MSF timer flags. When the timer counting value reaches the defined value, the corresponding flag will be set high and then cleared automatically the next time the user program is executed. Users can implement a time calculation in USER_PROGRAM by polling whether the used timer flag is 1.

## 4.3 Library Configuration Considerations

### 1. Specifying a correct path

An include path declaration is required when including relevant INI and EXT files from other folders in a .INC or .H file. For example, a declaration such as #include "..\USER_PROGRAM\USER_PROGRAM.EXT" in GLOBE_INCLUE.INC means to include the USER_PROGRAM.EXT file located in the USER_PROGRAM folder. Here ".." represents the parent path.

### 2. Program entry point declaration

In the MAIN_PROGRAM.INC file, users can declare entry points for programs. For example, the touch library underlying functions or the USER_PROGRAM functions are added to the main program through entry point declaration. If the user does not use some functions, the corresponding entry point declaration can be annotated to save RAM space.

### 3. Program address declaration

The program addresses, including interrupt addresses, should be declared using an absolute address, and cannot be declared by a relative address (ORG).

Absolute address:

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

√

Relative address:

```
        ORG      000H
        JMP      PROGRAM_RESET
```

×

For example, the figure below shows the correct address definition for an INT interrupt subroutine, so that when an external interrupt occurs, the program will execute the code within the user-defined INT subroutine.

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

### 4.4 Capacitance Measurement Using Library

The V512 library supports measuring capacitance in a range of 0~50pF. Connect the capacitor between the key pin and ground, and calculate the capacitance of the external capacitor through the internal capacitance data which is composed of two high bits and eight low bits. The calculation formula is as follows.

$$C = \frac{\text{Internal Capacitance} \times 50pF}{1024}$$