Trusted ePlatform Services

ADVANTECH

SUSI® Library

Version 3.0

User's Manual

Advantech Co. Ltd.

No. 1, Alley 20, Lane 26,

Rueiguang Road, Neihu District,

Taipei 114, Taiwan, R. O. C.

www.advantech.com

Copyright Notice

Part No.
Version: 3.0

Printed in Taiwan 2008-11-26

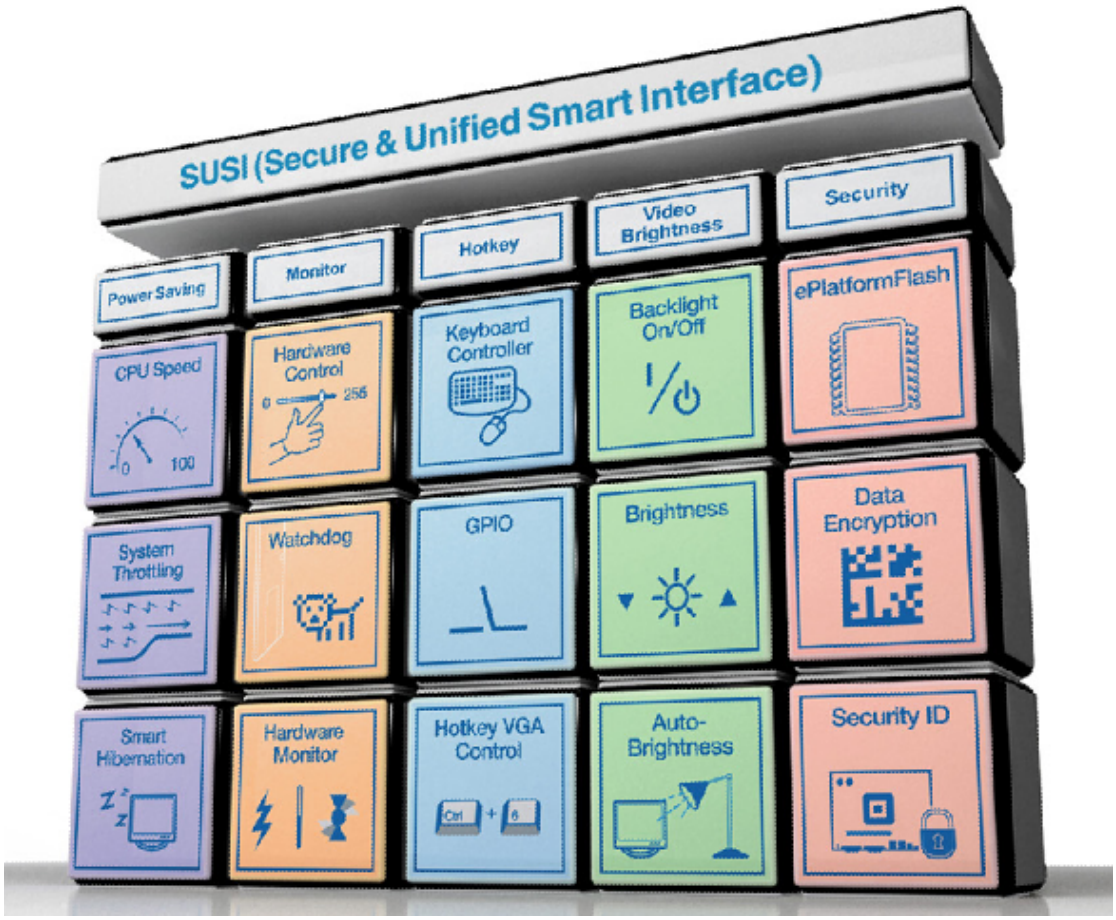
Version History

Date	Version	Part no	Remark
2006-7-27	1.0		New release
2006-9-29	1.1		Add hardware monitoring support for SOM-4472/SOM-4475/SOM-4481/SOM-4486
2007-6-27	1.2		Add many new functions over Control APIs Programmable GPIO, SMBus Enhanced Protocols Monitoring APIs Boot Counter and Running Timer, H/W Control Display APIs Auto-Brightness, Hotkey VGA Control Debug API Get last error code About new SUSI-enabled platforms, please refer to Appendix A
2007-10-01	2.0		Add Embedded BIOS interface Add Power Saving API: CPU Speed, System Throttling & Smart Hibernation Add Security API for AIMB-440 onboard FPGA: SRAM, AES, RNG, 72 bit GPIO
2008-05-01	3.0		Add Embedded BIOS interface for Linux Add SUSI Manager for central control of SUSI Add utilities for Monitoring, PowerSaving, HotKey manager, Brightness Control, Security ID, ePlatformFlash
2008-11-26	3.0		Add SMBus Programming Note

Table of Contents

Introduction

SUSI – A Bridge to Simplify & Enhance H/W & Application Implementation Efficiency



When developers want to write an application that involves hardware access, they have to study the specifications to write the drivers. This is a time-consuming job and requires lots of expertise.

Advantech has done all the hard work for our customers with the release of a suite of APIs (Application Programming Interfaces), called the **Secured & Unified Smart Interface** (SUSI).

SUSI provides not only the underlying drivers required but also a rich set of user-friendly, intelligent and integrated interfaces, which speeds development, enhances security and offers add-on value for Advantech platforms. SUSI plays the role of catalyst between developer and solution, and makes Advantech embedded platforms easier and simpler to adopt and operate with customer applications.

SUSI v3.0 contains 5 main off-the-shelf utilities:

- **Power Saving**
  Provides green power settings for max. energy savings and performance
- **HotKey**
  Sets keyboard short-cuts for GPIO and control
- **Video Brightness**
  Manual and auto modes for easy control of panels
- **Monitoring**
  Real-time temperature, fan and voltage monitoring, and alarm settings
- **Security**
  For multi-level protection with hash encryption

SUSI Functions

Hotkey

General Purpose Input Output function is a flexible parallel interface that allows a variety of custom connections and supports Digital I/O devices.

The Keyboard Controller API allows developers to dynamically set short-cuts for different combinations of keyboard events. Customers can also use it to pre-define function keys for a keyboard-less system.

This function provides a Hotkey for VGA Control; users can press CTRL plus “+” or “-” to increase or decrease brightness. Pressing Ctrl + 6 for example will give 60% brightness.

Monitoring

A watchdog timer (WDT) is a function that performs a specific operation after a certain period of time if something goes wrong with the system. A watchdog timer can be programmed to restart the system after a certain time period when a program or computer fails to respond or hangs.

The Hardware Monitor (HWM) API is a system health supervision API that inspects certain condition indexes, such as Fan Speed, Temperature and Voltage.

The Hardware Control API allows developers to set the PWM (Pulse Width Modulation) value to adjust Fan Speed or other devices and can also be used to adjust the LCD brightness.

Video Brightness

The Brightness Control API allows a developer to interface with Windows XP and Windows CE PCs to easily control brightness.

The Auto-Brightness function contains a new API and a Light Sensor IC, so systems can have a built-in Auto-Brightness adjustment utility.

The Backlight API allows a developer to control the backlight (screen) on/off in Windows XP and Windows CE.

Power Saving

Makes use of Intel SpeedStep technology to save power consumption. The system will automatically adjust the CPU Speed depending on the system load.

Methods for reducing power consumption in computers by lowering the clock frequency. These API allow users to lower the clock from 87.5% to 12.5%.

API for customers to easily enable Hibernation. The Windows hibernation feature conforms to the S4 Sleep State in the ACPI standard.

Security

ePlatformFlash is used to update the Advantech ePlatform BIOS and read/write a security ID into BIOS

We use SHA1-160, SHA1-256, SHA1-384, SHA1-512 array values to adopt different algorithms and lengths to encrypt each column to protect the keys.

Security consists of 3 different IDs: 1. Board ID: a unique string for each board, ready in the factory - Read Only. 2. Vendor ID: unique string for each customer or project, set in the factory - Read Only. 3. Customer ID: unique string defined by the customer, and input by the customer. Together they provide triple protection.

Benefits

- **Faster Time to Market**
  SUSI's unified API helps developers write applications to control the hardware without knowing the hardware specs of the chipsets and driver architecture.
- **Reduced Project Effort**
  When customers have their own devices connected to the onboard bus, they can either: study the data sheet and write the driver & API from scratch, or they can use SUSI to start the integration with a 50% head start. Developers can reference the sample program on the CD to see and learn more about the software development environment.
- **Enhances Hardware Platform Reliability**
  SUSI provides a trusted custom ready solution which combines chipset and library function support, controlling application development through SUSI enhances reliability and brings peace of mind.
- **Flexible Upgrade Possibilities**
  SUSI supports an easy upgrade solution for customers. Customers just need to install the new version SUSI that supports the new functions.

Environments

Operating Systems that SUSI supports include:

- Windows CE
- Windows XP Embedded
- Windows XP Pro or Home Edition

For the complete list of SUSI-enabled platforms, please refer to Appendix A. Note that the list may be changed without notice. For the latest support list, please check: http://www.advantech.com.tw/ess/SUSI.asp
Should you have any questions about your Advantech boards, please contact us by telephone or E-mail.

Package Contents

SUSI currently supports two OS - Windows CE and Windows XP. Contents listed below:

Operating System	Location	Installation
Windows XP(e)	C:\Program Files\SUSI\V30	Setup.exe
Windows CE	\Program Files\SUSI\V30	Image Built-in *
Directory	**Contents**	
User Manual	SUSI.pdf	
Library Files	• Susi.lib Function export • Susi.dll Dynamic link library	
Include Files	• Susi.h • Debug.h / Errdrv.h / Errlib.h	
SusiDemo	• SusiDemo.exe Demo program execution file • Susi.dll Dynamic link library	
SusiDemo\SRC\ C#	Source code of SusiDemo program in C#, VS2005	
SusiDemo\SRC \VB.NET	Source code of Watchdog of SusiDemo program in VB.NET, VS2005	

* Windows CE manual installation:

You can add the SUSI Library into the image by editing any bib file.

- First you open project.bib in the platform builder.
- Add this line to the MODULES section of project.bib
  Susi.dll $(_FLATRELEASEDIR)\Susi.dll NK SH
- If you want to run the window-based demo, add the following line:
  SusiTest.exe $(_FLATRELEASEDIR)\SusiTest.exe
- If you want to run the console-based demo, add following lines:
  Watchdog.exe $(_FLATRELEASEDIR)\Watchdog.exe NK S
  GPIO.exe $(_FLATRELEASEDIR)\GPIO.exe NK S
  SMBUS.exe $(_FLATRELEASEDIR)\SMBUS.exe NK S
- Place the three files into any *files* directory.
- Build your new Windows CE operating system.

Programming Overview

Header Files

- SUSI.H includes API declaration, constants and flags that are required for programming.
- DEBUG.H / ERRDRV.H / ERRLIB.H are for debug code definitions.
  DEBUG.H – Function index codes
  ERRLIB.H – Library error codes
  ERRDRV.H – Driver error codes

Library Files

- Susi.lib is for library import and Susi.dll is a dynamic link library that exports all the API functions.

Demo Program

- The `SusiDemo` program, released with source code, demonstrates how to fully use SUSI APIs. The program is written in the latest programming language C#.

Drivers

There are seven drivers for SUSI: CORE, WDT, GPIO, SMBus, IIC, VC and HWM.
E.g. Driver CORE is for `SusiCore-` prefixed APIs, and so on.
A driver will be loaded only if its corresponding function set is supported by a platform.

Installation File

In Windows CE, the files and drivers mentioned above are already built-in to the image. In Windows XP, you have to run Setup.exe for installation. To avoid double installation, please make sure you have removed any existing SUSI drivers, either by using Setup.exe or by manually removing them in Device Manger.

Dll functions

`SusiDll-` APIs are driver-independent, i.e. they can be called without any drivers. In Windows XP, after drivers having been installed, users have to call `SusiDllInit` for initialization before using any other APIs that are not `SusiDll-` prefixed. Before the application terminates, call `SusiDllUnInit` to free allocated system resources.
When an API call fails, use `SusGetLastError` to get an error report. An error value will be either

Function Index Code + Library Error Code, or
Function Index Code + Driver Error Code

The Function Index Code indicates which API the error came from and the library / Driver Error Code indicates the actual error type, i.e. whether it was an error in a library or driver. For a complete list of error codes, please refer to the Appendix

- SusiDllInit
- SusiDllUnInit
- SusiDllGetLastError
- SusiDllGetVersion

Core functions

SusiCore- APIs are available for all Advantech SUSI-enabled platforms to provide board information such as the platform name and BIOS version. New SusiCoreAccessBootCounter and SusiCoreAccessBootCounter APIs are **Boot Logger** features that enable monitoring of system reboot times, total OS run time and continual run time.

- SusiCoreGetPlatformName
- SusiCoreGetBIOSVersion
- SusiCoreAccessBootCounter
- SusiCoreAccessRunTimer

Watchdog (WD) functions

The hardware watchdog timer is a common feature among all Advantech platforms. In user applications, call SusiWDSetConfig with specific timeout values to start the watchdog timer countdown, meanwhile create a thread or timer to periodically refresh the timer with SusiWDTrigger before it expires. If the application ever hangs, it will fail to refresh the timer and the watchdog reset will cause a system reboot.

- SusiWDGetRange
- SusiWDSetConfig
- SusiWDTrigger
- SusiWDDisable

GPIO (IO) functions

There are two sets of GPIO functions. It is highly recommended to use the new one. With pin read and write, more flexibility has been added to allow easy pin direction change as needed, as well as the capability of reading output pin status.

New programmable GPIO function set:

- SusiIOCountEx
- SusiIOQueryMask
- SusiIOSetDirection
- SusiIOSetDirectionMulti
- SusiIOReadEx
- SusiIOReadMultiEx
- SusiIOWriteEx
- SusiIOWriteMultiEx

Previous function set:

- SusiIOCount
- SusiIOInitial
- SusiIORead
- SusiIOReadMulti;

- SusiIOWrite
- SusiIOWriteMulti

Refer to Appendix for pin allocation and their default direction.

SMBus functions

We support the SMBus 2.0 compliant protocols in SusiSMBus- APIs :

- Quick Command – SusiSMBusReadQuick/SusiSMBusWriteQuick
- Byte Receive/Send – SusiSMBusReceiveByte/SusiSMBusSendByte
- Byte Data Read/Write – SusiSMBusReadByte/SusiSMBusWriteByte
- Word Data Read/Write – SusiSMBusReadWord/SusiSMBusWriteWord

We also support an additional API for probing:

- SusiSMBusScanDevice

The slave address is expressed as a 7-bit hex number between 0x00 to 0x7F, however the actual addresses used for R/W are

8-bit write address = 7-bit address <<1 (left shift one) with LSB 0 (for write)

8-bit read address = 7-bit address <<1 (left shift one) with LSB 1 (for read)

E.g. Given a 7-bit slave address 0x20, the write address is 0x40 and the read address is 0x41.

Here in all APIs (except for SusiSMBusScanDevice), parameter SlaveAddress is the 8-bit address and users don't need to care about giving it as a read or write address, since the actual R/W is taken care by the API itself, i.e. you could even use a write address, say 0x41 for APIs with write operation and get the right result, and vice versa.

SusiSMBusScanDevice is used to probe whether an address is currently used by certain devices on a platform. You can find out which addresses are occupied by scanning from 0x00 to 0x7f. For example, you could scan for occupied addresses and avoid them when connecting a new device; or by probing before and after connecting the new device, you could quickly know its address. The SlaveAddress_7 parameter given in this API is a 7-bit address.

Notice that we don't recommend user to access the address A0~AF due to it might hurt DRAM SPD value and cause RAM failed.

IIC functions

The APIs here cover IIC standard mode operations with a 7-bit device address:

- SusiIICRead
- SusiIICWrite
- SusiIICWriteReadCombine

IIC versus SMBus - compatibility

On platforms that do not have IIC but do have SMBus, a call to SusiIICAvailable returns SUSI_IIC_TYPE_SMBUS (2). Users might be able to use SMBus as a substitute; however, whether it's with fully or partially supported depends on the SMBus controller type.

In AMD platforms, we have implemented the SMBus driver to be totally IIC standard mode compatible; users could use the IIC APIs implemented by the SMBus controller with `IICType` = `SUSI_IIC_TYPE_SMBUS` to communicate with all kinds of IIC devices.
In Intel and VIA's platforms, the currently compatible protocols are

- `SusiIICRead` with `ReadLen` = 1
- `SusiIICWrite` with `WriteLen` = 1

IIC devices with 7-bit slave addresses can also be scanned by `SusiSMBusScanDevice` on all platforms that have SMBus support.
We are now working on more IIC compatible APIs for Intel and VIA controllers. These APIs will be supported soon.
For more details on platform IIC/SMBus support, please refer to Appendix A.

VGA Control (VC) functions

`SusiVC-` functions support VGA signal ON/OFF on all SUSI-enabled platforms and also LCD brightness adjustment.

- `SusiVCScreenOn`
- `SusiVCScreenOff`
- `SusiVCGetBrightRange`
- `SusiVCGetBright`
- `SusiVCSetBright`

One application of `SusiVCScreenOn` and `SusiVCScreenOff` is to have the display signal disabled when system idles after certain period of time to expand the panel life span.

Hardware Monitoring (HWM) functions

`SusiHWM-` functions support system health supervision by retrieving the values of voltage, temperature and fan sensors. In some platforms, it is possible to control the CPU/System fan speed. Use these functions cautiously.

- `SusiHWMAvailable`
- `SusiHWMGetFanSpeed`
- `SusiHWMGetTemperature`
- `SusiHWMGetVoltage`
- `SusiHWMSetFanSpeed`

FPGA SRAM functions

- SUSIFPGAStorageAreaGetType
- SUSIFPGAStorageAreaGetSize
- SUSIFPGAStorageAreaRead
- SUSIFPGAStorageAreaWrite
- SUSIFPGAStorageAreaErase
- SUSIFPGAStorageAreaFPGAConfig

FPGA 72 Bit GPIO functions

- SUSIFPGAStorageAreaFPGAConfig
- SUSIFPGAIOFPGACountEx

- SUSIFPGAIOFPGAReadEx
- SUSIFPGAIOFPGAReadMultiEx
- SUSIFPGAIOFPGAWriteEx
- SUSIFPGAIOFPGAWriteMultiEx

FPGA AES functions

- SUSIFPGASecurityFPGASetAESKey
- SUSIFPGASecurityFPGAGetAESKey
- SUSIFPGASecurityFPGAGenerateAESData

FPGA RNG functions

- SUSIFPGASecurityFPGAGenerateRandomNum

SUSI API Programmer’s Documentation

All APIs return the `BOOL` data type except `Susi*Available` and some special cases that are of type `int`. If any function call fails, i.e. a `BOOL` value of FALSE, or an `int value` of -1, the error code can always be retrieved by an immediate call to `SusiGetLastError.`

SusiDllInit

Initialize the Susi Library.

```
BOOL SusiDllInit(void);
```

Parameters
None.

Return Value
TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks
An application must call `SusiDllInit` before calling any other non SusiDll-functions.

SusiDllUnInit

Uninitialize the Susi Library.

```
BOOL SusiDllUnInit(void);
```

Parameters
None.

Return Value
TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks
Before an application terminates, it must call `SusiDllUnInit` if it has successfully called `SusiDllInit`. Calls to `SusiDllInit` and `SusiDllUnInit` can be nested but must be paired.

.

SusiDllGetVersion

Retrieve the version numbers of SUSI Library.

```
void SusiDllGetVersion(WORD *major, WORD *minor);
```

Parameters

`major`

`[out]` Pointer to a variable containing the major version number.

`minor`

`[out]` Pointer to a variable containing the minor version number.

Return Value

None.

Remarks

This function returns the version numbers of SUSI. It's suggested to call this function first and compare the numbers with the constants `SUSI_LIB_VER_MJ` and `SUSI_LIB_VER_MR` in header file SUSI.H to insure the library compatibility.

SusiDllGetLastError

This function returns the last error code value.

```
int SusiDllGetLastError(void);
```

Parameters

None

Return Value

The code of error reason for the last function call with failure.

Remarks

You should call the `SusiDllGetLastError` immediately when a function's return value indicates failure.

The return error code will be either

Function Index Code + Library Error Code, or

Function Index Code + Driver Error Code

The Function Index Code distinguishes which API the error resulted from and the library / Driver Error Code indicates the actual error type, i.e. if it is an error in a library or driver. For a complete list of error codes, please refer to the Appendix.

SusiCoreAvailable

Check if Core driver is available.

```
int SusiCoreAvailable (void);
```

Parameters

None.

Return Value

Value	Meaning
-1	The function fails.
0	The function succeeds; the platform does not support SusiCore- APIs.
1	The function succeeds; the platform supports Core.

Remarks

After calling `SusiDllInit` successfully, all `Susi*Available` functions are used to check if the corresponding features are supported by the platform or not. So it is suggested to call `Susi*Available` before using any `Susi*-` functions.

SusiCoreGetBIOSVersion

Get the current BIOS version.

```
BOOL SusiCoreGetBIOSVersion(TCHAR *BIOSVersion, DWORD
*size);
```

Parameters

BIOSVersion

[out] Pointer to an array in which the BIOS version string is returned.

size

[in/out]

Pointer to a variable that specifies the size, in TCHAR, of the array pointed to by the BIOSVersion parameter.

If BIOSVersion is given as NULL, when the function returns, the variable will contain the array size required for the BIOS version.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

Call the function twice, first by giving BIOSVersion as NULL to get the array size required for the string. Then allocate a TCHAR array with the size required and give the array with its size as parameters to get the BIOS version. Note that the BIOS version cannot be correctly retrieved if it's a release version.

SusiCoreGetPlatformName

Get the current platform name.

```
BOOL SusiCoreGetPlatformName(TCHAR *PlatformName, DWORD
*size);
```

Parameters

PlatformName

[out] Pointer to an array in which the platform name string is returned.

size

[in/out]

Pointer to a variable that specifies the size, in TCHAR, of the array pointed to by the PlatformName parameter.

If PlatformName is given as NULL, when the function returns, the variable will contain the array size required for the platform name.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

Call the function twice, first by giving PlatformName as NULL to get the array size required for the string. Then allocate a TCHAR array with the size required and give the array with its size as parameters to get the platform name. Note that the platform name cannot be correctly retrieved if the BIOS is a release version.

SusiCoreAccessBootCounter

Access the boot counter. A boot counter is used to count the number of boot times.

```
BOOL SusiCoreAccessBootCounter(DWORD mode, DWORD OPFlag,
BOOL *enable, DWORD *value);
```

Parameters

mode

[in] The value can be either
ESCORE_BOOTCOUNTER_MODE_GET (0)
- To get information from counter.
ESCORE_BOOTCOUNTER_MODE_SET (1)
- To set information to counter.

OPFlag

[in] The operation flag can be the combination of
ESCORE_BOOTCOUNTER_STATUS (1)
- The operation is on the parameter `enable`
ESCORE_BOOTCOUNTER_VALUE (2)
- The operation is on the parameter `value`

enable

[in/out]
If `OPFlag` contains ESCORE_BOOTCOUNTER_STATUS (1):
When `mode` equals ESCORE_BOOTCOUNTER_MODE_GET(0), after the function returns, `enable` will contain the status of the counter: TRUE (enabled) or FALSE (disabled).
When `mode` equals ESCORE_BOOTCOUNTER_MODE_SET(1), `enable` is a pointer to a variable that contains the status to set. Use TRUE to start the counter or FALSE to stop.

value

[in/out]
If `OPFlag` contains ESCORE_BOOTCOUNTER_VALUE (2):
When `mode` equals ESCORE_BOOTCOUNTER_MODE_GET(0), after the function returns, `value` will contain the reboot count.
When `mode` equals ESCORE_BOOTCOUNTER_MODE_SET(1), `value` is a pointer to a variable that contains the reboot count to set. Give a value 0 to clear the count or any other value to start from.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

In windows XP, the boot counter information is stored in the following registry values:
HKEY_LOCAL_MACHINE \SYSTEM\SusiBootCounter\Enable
HKEY_LOCAL_MACHINE \SYSTEM\SusiBootCounter\BootTimes
In windows CE:
HKEY_CURRENT_USER\Software\Advantech\Susi\Core\BootCounter\Enable
HKEY_CURRENT_USER\Software\Advantech\Susi\Core\BootCounter\BootTimes

The information will be lost only if the registry values have been wiped out.
For a definition of boot counter flags, please refer to the Appendix.

SusiCoreAccessRunTimer

Access the run timer. A run timer is used to count the system running time.

```
BOOL SusiCoreAccessRunTimer(DWORD mode, PSSCORE_RUNTIMER pRunTimer);
```

Parameters

mode

[in] The value can be either

ESCORE_BOOTCOUNTER_MODE_GET (0)
- Get the counter.

ESCORE_BOOTCOUNTER_MODE_SET (1)
- Set the counter.

pRunTimer

[in/out]

Pointer to a SSCORE_RUNTIMER structure to set or get the timer. Please see next page for details of this structure.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

In windows XP, the information is stored in the following registry values:

HKEY_LOCAL_MACHINE\SYSTEM\SusiRunTimer\Running
HKEY_LOCAL_MACHINE\SYSTEM\SusiRunTimer\Autorun
HKEY_LOCAL_MACHINE\SYSTEM\SusiRunTimer\ContinualOnTime
HKEY_LOCAL_MACHINE\SYSTEM\SusiRunTimer\TotalOnTime

In windows CE, they are in:

HKEY_CURRENT_USER\Software\Advantech\Susi\Core\RunTimer\Running
HKEY_CURRENT_USER\Software\Advantech\Susi\Core\RunTimer\Autorun
HKEY_CURRENT_USER\Software\Advantech\Susi\Core\RunTimer\ContinualOnTime
HKEY_CURRENT_USER\Software\Advantech\Susi\Core\RunTimer\TotalOnTime

The information will be lost only if the registry values have been wiped out.

For a detailed definition of the SSCORE_RUNTIMER structure, please refer to next page.

SSCORE_RUNTIMER

This structure represents the run timer information.

```
typedef struct {
  DWORD dwOPFlag;
  BOOL  isRunning;
  BOOL  isAutorun;
  DWORD dwTimeContinual;
  DWORD dwTimeTotal;
} SSCORE_RUNTIMER, *PSSCORE_RUNTIMER;
```

Members

dwOPFlag

The operation flag can be a combination of:

`ESCORE_RUNTIMER_STATUS_RUNNING` (1)

- The operation is on the member `isRunning`

`ESCORE_RUNTIMER_STATUS_AUTORUN` (2)

- The operation is on the member `isAutorun`

`ESCORE_RUNTIMER_VALUE_CONTINUALON(4)`

- The operation is on the member `dwTimeContinual`

`ESCORE_RUNTIMER_VALUE_TOTALON(8)`

- The operation is on the member `dwTimeTotal`

isRunning

TURE indicates the timer is running now, FALSE indicates not.

isAutorun

TRUE states the timer will start automatically upon startup, i.e. it will be running each time when the system reboots.

dwTimeContinual

Specify the system continual-on time in minutes, i.e. the OS running time without a system reboot. At reboot, it will be reset to 0.

dwTimeTotal

Specify the system total-on time in minutes, i.e. the total time accumulated while the OS has been running.

SusiWDAvailable

Check if the watchdog driver is available.

```
BOOL SusiWDAvailable(void);
```

Parameters

None.

Return Value

value	Meaning
-1	The function fails.
0	The function succeeds; the platform does not support SusiWD- APIs.
1	The function succeeds; the platform supports Watchdog.

Remarks

After calling `SusiDllInit` successfully, all `Susi*Available` functions are used to check if the corresponding features are supported by the platform or not. We suggest `Susi*Available` is called before using any `Susi*-` functions.

SusiWDGetRange

Get the step, minimum and maximum values of the watchdog timer.

```
BOOL SusiWDGetRange(DWORD *minimum, DWORD *maximum,
DWORD *stepping);
```

Parameters

minimum

[out] Pointer to a variable containing the minimum timeout value in milliseconds.

maximum

[out] Pointer to a variable containing the maximum timeout value in milliseconds.

stepping

[out] Pointer to a variable containing the resolution of the timer in milliseconds.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

The values may vary from platform to platform; depending on the hardware implementation of the watchdog timer. For example, if the minimum timeout is 1000, the maximum timeout is 63000, and the step is 1000, it means the watchdog timeout will count 1, 2, 3 … 63 seconds.

SusiWDSetConfig

Start watchdog timer with specified timeout value.

```
BOOL SusiWDSetConfig(DWORD delay, DWORD timeout);
```

Parameters

delay

[in] Specifies a value in milliseconds which will be added to "the first" timeout period. This allows the application to have sufficient time to do initialization before the first call to `SusiWDTrigger` and still be protected by the watchdog.

timeout

[in] Specifies a value in milliseconds for the watchdog timeout.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure

Remarks

Once the watchdog has been activated, its timer begins to count down. The application has to periodically call `SusiWDTrigger` to refresh the timer before it expires, i.e. reload the watchdog timer within the specified timeout or the system will reboot when it counts down to 0.

Actually a subsequent call to `SusiWDTrigger` equals a call to `SusiWDSetConfig` with `delay` 0 and the original `timeout` value, so if you want to change the timeout value, call `SusiWDSetConfig` with new `timeout` value instead of `SusiWDTrigger.`

Use `SusiWDGetRange` to get the acceptable timeout values.

SusiWDTrigger

Reload the watchdog timer to the timeout value given in SusiWDSetConfig to prevent the system from rebooting.

```
BOOL SusiWDTrigger(void);
```

Parameters

None

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

A watchdog protected application has to call `SusiWDTrigger` continuously to indicate that it is still working properly and prevent a system restart. The first call to `SusiWDTrigger` in the middle of a delay resulting from a previous call to `SusiWDSetConfig` causes the delay timer to be canceled immediately and starts the watchdog timer countdown from the timeout value. It is always a good choice for users to have a longer delay time in `SusiWDSetConfig`.

SusiWDDisable

Disable the watchdog and stop its timer countdown.

```
BOOL SusiWDDisable(void);
```

Parameters

None

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

If watchdog protection is no longer required by an application, it can call `SusiWDDisable` to disable the watchdog. A call to `SusiWDDisable` in the middle of a delay resulting from a previous call to `SusiWDSetConfig` causes the delay timer to be canceled immediately and stops watchdog timer countdown. Only a few hardware implementations in which the watchdog timer cannot be stopped once it has been activated, will return with FALSE.

SusiIOAvailable

Check if GPIO driver is available.

```
int SusiCoreAvailable (void);
```

Parameters

None.

Return Value

value	Meaning
-1	The function fails.
0	The function succeeds; the platform does not support SusiIO- APIs.
1	The function succeeds; the platform supports GPIO.

Remarks

After calling `SusiDllInit` successfully, all `Susi*Available` functions are used to check if the corresponding features are supported by the platform or not. It is suggested to call `Susi*Available` before using any `Susi*-` functions.

SusiIOCountEx

Query the current number of input and output pins.

```
BOOL SusiIOCountEx(DWORD *inCount, DWORD *outCount)
```

Parameters

inCount

[out] Pointer to a variable in which this function returns the count of input pins.

outCount

[out] Pointer to a variable in which this function returns the count of output pins.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

The number of GPIO pins equals the number of input pins plus the number of output pins. The number of input and output pins may vary in accordance with the current pin direction.

SusiIOQueryMask

Query the GPIO mask information.

```
BOOL SusiIOQueryMask(DWORD flag, DWORD *Mask)
```

Parameters

flag

[in] The value given to indicate the type of mask to retrieve can be one of the following values:

Static masks

ESIO_SMASK_PIN_FULL (1)

ESIO_SMASK_CONFIGURABLE (2)

Dynamic masks

ESIO_DMASK_DIRECTION (0x20)

Mask

[out] Pointer to a variable in which this function returns the queried mask.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

A mask is expressed as a series of binary digits. Each bit corresponds to a pin (bit 0 for pin 0, bit 1 for pin 1, bit 2 for pin 2 …), depending on the mask type:

A bit value **1** stands for a pin with

1. **I**nput direction
2. Status HIGH
3. Direction changeable.

Or a bit value **0** stands for a pin with

1. **O**utput direction
2. Status LOW
3. Direction unchangeable

Here are the definitions for masks:

- ESIO_SMASK_PIN_FULL
  - If there are total 8 GPIO pins (GPIO 0 ~ 7) in a platform, the full pin mask is 0xFF, or in binary 11111111, i.e. the number of 1s corresponds to the number of pins.
- ESIO_SMASK_CONFIGURABLE
  - This is the mask to indicate which pins have changeable directions. If all the 8 pins are changeable, the mask would be 0xFF.
- ESIO_DMASK_DIRECTION
  - The current direction of pins. If the mask is 0xAA, or in binary 10101010, it means the even pins are output pins and the odd pins are input pins.

SusiIOSetDirection

Set direction of one GPIO pin as input or output.

```
BOOL SusiIOSetDirection(BYTE PinNum, BYTE IO, DWORD
*PinDirMask);
```

Parameters

PinNum

[in] Specifies the GPIO pin to be changed, ranging from 0 ~ (total number of GPIO pins minus 1).

IO

[in] Specifies the pin direction to be set.

PinDirMask

[out] Pointer to a variable in which the function returns the latest direction mask after the pin direction is set.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

Use an `IO` value of 1 to set a pin as an input or 0 to set a pin as an output.

The function can only set the direction of one of the pins that are direction configurable. If the pin number specified is an invalid pin or a pin that can only be configured as an input, the function call will fail and return FALSE.

SusiIOSetDirectionMulti

Set directions of multiple pins at once.

```
BOOL SusiIOSetDirectionMulti(DWORD TargetPinMask, DWORD
*PinDirMask);
```

Parameters

TargetPinMask

[in] Specifies the mask of GPIO output pins to be written.

PinDirMask

[in/out]

Specifies the directions of pins to be set in a bitwise-ORed manner. After the function call returns TRUE, it contains the latest direction mask after set.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

For example, if you set to the directions of GPIO pin 0, 1, 6, 7. Give parameter `TargetPinMask` with a value 11000011, or 0xC3. Bit 0 stand for GPIO 0, bit 1 stand for GPIO 1, and so on.

If you want to set pin 0 as input, pin 1 as output, pin 6 as input and pin 7 as output. Give value in parameter `PinDirMask` as 01XXXX01, X is for don't care, you could simply assign a 0 for it, i.e. 0x41.

SusiIOReadEx

Read current status of one GPIO input or output pin.

```
BOOL SusiIOReadEx(BYTE PinNum, BOOL *status)
```

Parameters

PinNum

[in] Specifies the GPIO pin demanded to be read, ranging from 0 ~ (total number of GPIO pins minus 1).

status

[out] Pointer to a variable in which the pin status returns.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

If the pin is in status high, the value got in `status` will be 1. If the pin is in status low, it will be zero. The function is capable of reading the status of either an input pin or an output pin.

SusiIOReadMultiEx

Read current statuses of multiple pins at once regardless of the pin directions.

```
BOOL SusiIOReadMultiEx(DWORD TargetPinMask, DWORD
*StatusMask);
```

Parameters

TargetPinMask

[in] Specifies the mask of GPIO pins demanded to be read.

StatusMask

[out] Statuses of pins in Bitwise-ORed. For pins that are not specified in `TargetPinMask`, the related bit value is invalid.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

For example, if you want to read the statuses of GPIO pin 0, 1, 6, 7. Give parameter `TargetPinMask` with a value 11000011, or 0xC3. Bit 0 stand for GPIO 0, bit 1 stand for GPIO 1, and so on. Again, if the pin is in status high, the value got in relevant bit of `StatusMask` will be 1. If the pin is in status low, it will be zero.

SusiIOWriteEx

Set one GPIO output pin as status high or low.

```
BOOL SusiIOWriteEx(BYTE PinNum, BOOL status);
```

Parameters

PinNum

[in] Specifies the GPIO pin demanded to be written, ranging from 0 ~ (total number of GPIO pins minus 1).

status

[in] Specifies the GPIO status to be written.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

The function can only set the status of one of the output pins. If the pin number specified is an input pin or an invalid pin, the function call will fail and return with FALSE. A `status` with 1 to set the pin as output high, 0 to set the pin as output low.

SusiIOWriteMultiEx

Set statuses of multiple output pins at once.

```
BOOL SusiIOWriteMultiEx(DWORD TargetPinMask, DWORD
StatusMask);
```

Parameters

TargetPinMask

[in] Specifies the mask of GPIO output pins demanded to be written.

StatusMask

[in] Statuses of pins to be set in Bitwise-ORed. For pins that are not specified in `TargetPinMask`, the related bit value is invalid.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

For example, if you want to write the statuses of GPIO output pin 0, 1, 6, 7. Give parameter `TargetPinMask` with a value 11000011, or 0xC3. Bit 0 stand for GPIO 0, bit 1 stand for GPIO 1, and so on.

If you want to set pin 0 as high, pin 1 as low, pin 6 as high and pin 7 as low. Give parameter `StatusMask` with a value 01XXXX01, X is for don't care pin, you could simply assign a 0 for it, i.e. 0x41.

Susi64BitsIOQueryMask

Query the GPIO mask information.

```
BOOL Susi64BitsIOQueryMask(DWORD flag, UINT64 *Mask)
```

Parameters

flag

[in] The value given to indicate the type of mask to retrieve can be one of the following values:

Static masks

ESIO_SMASK_PIN_FULL (1)

ESIO_SMASK_CONFIGURABLE (2)

Dynamic masks

ESIO_DMASK_DIRECTION (0x20)

Mask

[out] Pointer to a variable in which this function returns the queried mask.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

A mask is expressed as a series of binary digits. Each bit corresponds to a pin (bit 0 for pin 0, bit 1 for pin 1, bit 2 for pin 2 …), depending on the mask type:

A bit value **1** stands for a pin with

4. **I**nput direction
5. Status HIGH
6. Direction changeable.

Or a bit value **0** stands for a pin with

4. **O**utput direction
5. Status LOW
6. Direction unchangeable

Here are the definitions for masks:

- ESIO_SMASK_PIN_FULL
  - If there are total 8 GPIO pins (GPIO 0 ~ 7) in a platform, the full pin mask is 0xFF, or in binary 11111111, i.e. the number of 1s corresponds to the number of pins.
- ESIO_SMASK_CONFIGURABLE
  - This is the mask to indicate which pins have changeable directions. If all the 8 pins are changeable, the mask would be 0xFF.
- ESIO_DMASK_DIRECTION
  - The current direction of pins. If the mask is 0xAA, or in binary 10101010, it means the even pins are output pins and the odd pins are input pins.

Susi64BitsIOSetDirection

Set direction of one GPIO pin as input or output.

```
BOOL Susi64BitsIOSetDirection(ULONG PinNum, BYTE IO,
UINT64 *PinDirMask);
```

Parameters

PinNum

[in] Specifies the GPIO pin to be changed, ranging from 0 ~ (total number of GPIO pins minus 1).

IO

[in] Specifies the pin direction to be set.

PinDirMask

[out] Pointer to a variable in which the function returns the latest direction mask after the pin direction is set.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

Use an `IO` value of 1 to set a pin as an input or 0 to set a pin as an output.

The function can only set the direction of one of the pins that are direction configurable. If the pin number specified is an invalid pin or a pin that can only be configured as an input, the function call will fail and return FALSE.

Susi64BitsIOSetDirectionMulti

Set directions of multiple pins at once.

```
BOOL Susi64BitsIOSetDirectionMulti(UINT64 TargetPinMask,
UINT64 *PinDirMask);
```

Parameters

TargetPinMask

[in] Specifies the mask of GPIO output pins to be written.

PinDirMask

[in/out]

Specifies the directions of pins to be set in a bitwise-ORed manner. After the function call returns TRUE, it contains the latest direction mask after set.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

For example, if you set to the directions of GPIO pin 0, 1, 6, 7. Give parameter `TargetPinMask` with a value 11000011, or 0xC3. Bit 0 stand for GPIO 0, bit 1 stand for GPIO 1, and so on.

If you want to set pin 0 as input, pin 1 as output, pin 6 as input and pin 7 as output. Give value in parameter `PinDirMask` as 01XXXX01, X is for don't care, you could simply assign a 0 for it, i.e. 0x41.

Susi64BitsIOReadMultiEx

Read current statuses of multiple pins at once regardless of the pin directions.

```
BOOL Susi64BitsIOReadMultiEx(DWORD TargetPinMask, DWORD
*StatusMask);
```

Parameters

TargetPinMask

[in] Specifies the mask of GPIO pins demanded to be read.

StatusMask

[out] Statuses of pins in Bitwise-ORed. For pins that are not specified in `TargetPinMask`, the related bit value is invalid.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

For example, if you want to read the statuses of GPIO pin 0, 1, 6, 7. Give parameter `TargetPinMask` with a value 11000011, or 0xC3. Bit 0 stand for GPIO 0, bit 1 stand for GPIO 1, and so on. Again, if the pin is in status high, the value got in relevant bit of `StatusMask` will be 1. If the pin is in status low, it will be zero.

Susi64BitsIOWriteMultiEx

Set statuses of multiple output pins at once.

```
BOOL Susi64BitsIOWriteMultiEx(DWORD TargetPinMask, DWORD
StatusMask);
```

Parameters

TargetPinMask

[in] Specifies the mask of GPIO output pins demanded to be written.

StatusMask

[in] Statuses of pins to be set in Bitwise-ORed. For pins that are not specified in `TargetPinMask`, the related bit value is invalid.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

For example, if you want to write the statuses of GPIO output pin 0, 1, 6, 7. Give parameter `TargetPinMask` with a value 11000011, or 0xC3. Bit 0 stand for GPIO 0, bit 1 stand for GPIO 1, and so on.

If you want to set pin 0 as high, pin 1 as low, pin 6 as high and pin 7 as low. Give parameter `StatusMask` with a value 01XXXX01, X is for don't care pin, you could simply assign a 0 for it, i.e. 0x41.

SusiSMBusAvailable

Check if SMBus driver is available.

```
int SusiSMBusAvailable(void);
```

Parameters

None.

Return Value

value	Meaning
-1	The function fails.
0	The function succeeds; the platform does not support SusiSMbus- APIs.
1	The function succeeds; the platform supports SMBus.

Remarks

After calling `SusiDllInit` successfully, all `Susi*Available` functions are use to check if the corresponding features are supported by the platform or not. So it is suggested to call `Susi*Available` before using any `Susi*-` functions.

SusiSMBusScanDevice

Scan if the address is taken by one of the slave devices currently connected to the SMBus.

```
int SusiSMBusScanDevice(BYTE SlaveAddress_7)
```

Parameters

SlaveAddress

[in] Specifies the 7-bit device address, ranging from 0x00 – 0x7F.

Return Value

value	Meaning
-1	The function fails.
0	The function succeeds; the address is not occupied.
1	The function succeeds; there is a device to this address.

Remarks

There could be as much as 128 devices connected to a single SMBus. For more information about how to use this API, please refer to "Programming Overview", part "SMBus functions".

SusiSMBusReadQuick

Turn a SMBus device function on (off) or enable (disable) a specific device mode.

```
BOOL SusiSMBusReadQuick(BYTE SlaveAddress);
```

Parameters

SlaveAddress

[in] Specifies the 8-bit device address, ranging from 0x00 – 0xFF. Whether to give a 1 (read) or 0 (write) to the LSB of `SlaveAddress` could be ignored.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

For more information about how to use this API, please refer to "Programming Overview", part "SMBus functions".

SusiSMBusWriteQuick

Turn a SMBus device function off (on) or disable (enable) a specific device mode.

```
BOOL SusiSMBusWriteQuick(BYTE SlaveAddress);
```

Parameters

SlaveAddress

[in] Specifies the 8-bit device address, ranging from 0x00 – 0xFF. Whether to give a 1 (read) or 0 (write) to the LSB of SlaveAddress could be ignored.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

For more information about how to use this API, please refer to "Programming Overview", part "SMBus functions".

SusiSMBusReceiveByte

Receive information in a byte from the target slave device in the SMBus.

```
BOOL SusiSMBusReceiveByte(BYTE SlaveAddress, BYTE
*Result);
```

Parameters

SlaveAddress

[in] Specifies the 8-bit device address, ranging from 0x00 – 0xFF.
Whether to give a 1 (read) or 0 (write) to the LSB of SlaveAddress could be ignored.

Result

[out] Pointer to a variable in which the function receives the byte information.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

A simple device may have information that the host needs to be received in the parameter Result.

For more information about how to use this API, please refer to "Programming Overview", part "SMBus functions".

SusiSMBusSendByte

Send information in a byte to the target slave device in the SMBus.

```
BOOL SusiSMBusSendByte(BYTE SlaveAddress, BYTE Result);
```

Parameters

SlaveAddress

[in] Specifies the 8-bit device address, ranging from 0x00 – 0xFF. Whether to give a 1 (read) or 0 (write) to the LSB of SlaveAddress could be ignored.

Result

[in] Specifies the byte information to be sent.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

A simple device may recognize its own slave address and accept up to 256 possible encoded commands in the form of a byte given in the parameter Result.

For more information about how to use this API, please refer to "Programming Overview", part "SMBus functions".

SusiSMBusReadByte

Read a byte of data from the target slave device in the SMBus.

```
BOOL SusiSMBusReadByte(BYTE SlaveAddress, BYTE
RegisterOffset, BYTE *Result);
```

Parameters

SlaveAddress

[in] Specifies the 8-bit device address, ranging from 0x00 – 0xFF. Whether to give a 1 (read) or 0 (write) to the LSB of `SlaveAddress` could be ignored.

RegisterOffset

[in] Specifies the offset of the device register to read data from.

Result

[out] Pointer to a variable in which the function reads the byte data.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

For more information about how to use this API, please refer to "Programming Overview", part "SMBus functions".

SusiSMBusWriteByte

Write a byte of data to the target slave device in the SMBus.

```
BOOL SusiSMBusWriteByte(BYTE SlaveAddress, BYTE
RegisterOffset, BYTE Result);
```

Parameters

SlaveAddress

[in] Specifies the 8-bit device address, ranging from 0x00 – 0xFF. Whether to give a 1 (read) or 0 (write) to the LSB of `SlaveAddress` could be ignored.

RegisterOffset

[in] Specifies the offset of the device register to write data to.

Result

[in] Specifies the byte data to be written .

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

For more information about how to use this API, please refer to "Programming Overview", part "SMBus functions".

SusiSMBusReadWord

Read a word (2 bytes) of data from the target slave device in the SMBus.

```
BOOL SusiSMBusReadWord(BYTE SlaveAddress, BYTE
RegisterOffset, WORD *Result);
```

Parameters

SlaveAddress

[in] Specifies the 8-bit device address, ranging from 0x00 – 0xFF.
Whether to give a 1 (read) or 0 (write) to the LSB of `SlaveAddress` could be ignored.

RegisterOffset

[in] Specifies the offset of the device register to read data from.

Result

[out] Pointer to a variable in which the function reads the word data.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

The first byte read from slave device will be placed in the low byte of `Result`, and the second byte read will be placed in the high byte.

For more information about how to use this API, please refer to "Programming Overview", part "SMBus functions".

SusiSMBusWriteWord

Write a word (2 bytes) of data to the target slave device in the SMBus.

```
BOOL SusiSMBusWriteWord(BYTE SlaveAddress, BYTE
RegisterOffset, WORD Result);
```

Parameters

SlaveAddress

[in] Specifies the 8-bit device address, ranging from 0x00 – 0xFF. Whether to give a 1 (read) or 0 (write) to the LSB of `SlaveAddress` could be ignored.

RegisterOffset

[in] Specifies the offset of the device register to write data to.

Result

[in] Specifies the word data to be written .

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

The low byte of `Result` will be send to the slave device first and then the high byte. For more information about how to use this API, please refer to "Programming Overview", part "SMBus functions"

SusiIICAvailable

Check if I^2C driver is available and also get the IIC type supported.

```
int SusiIICAvailable();
```

Parameters

None.

Return Value

value	Meaning
-1	The function fails.
0	The function succeeds; the platform does not support any SusiIIC - APIs.
SUSI_IIC_TYPE_PRIMARY (1)	The function succeeds; the platform supports only primary IIC.
SUSI_IIC_TYPE_SMBUS (2)	The function succeeds; the platform supports only SMBus implemented IIC.
SUSI_IIC_TYPE_BOTH (3)	The function succeeds; the platform supports both primary IIC and SMBus IIC.

Remarks

After calling `SusiDllInit` successfully, all `Susi*Available` functions are use to check if the corresponding features are supported by the platform or not. So it is suggested to call `Susi*Available` before using any `Susi*-` functions.

SusiIICRead

Read bytes of data from the target slave device in the I^2C bus.

```
SUSI_API BOOL SusiIICRead(DWORD IICType, BYTE SlaveAddress,
BYTE *ReadBuf, DWORD ReadLen);
```

Parameters

IICType

[in] Specifies the I^2C type, the value can either be
SUSI_IIC_TYPE_PRIMARY (1)
SUSI_IIC_TYPE_SMBUS (2)

SlaveAddress

[in] Specifies the 8-bit device address, ranging from 0x00 – 0xFF.
Whether to give a 1 (read) or 0 (write) to the LSB of SlaveAddress could be ignored.

ReadBuf

[out] Pointer to a variable in which the function reads the bytes of data.

ReadLen

[in] Specifies the number of bytes to be read.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

Call SusiIICAvailable first to make sure the support I^2C type. For more information about how to use this API, and the relationship between IIC and SMBus, please refer to "Programming Overview", parts "SMBus functions" to "IIC versus SMBus – compatibility"

SusiIICWrite

Write bytes of data to the target slave device in the I^2C bus.

```
BOOL SusiIICWrite(DWORD IICType, BYTE SlaveAddress, BYTE
*WriteBuf, DWORD WriteLen);
```

Parameters

IICType

[in] Specifies the I^2C type, the value can either be
SUSI_IIC_TYPE_PRIMARY (1)
SUSI_IIC_TYPE_SMBUS (2)

SlaveAddress

[in] Specifies the 8-bit device address, ranging from 0x00 – 0xFF.
Whether to give a 1 (read) or 0 (write) to the LSB of SlaveAddress could be ignored.

WriteBuf

[in] Pointer to a byte array which contains the bytes of data to be written.

WriteLen

[in] Specifies the number of bytes to be written.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

Call SusiIICAvailable first to make sure the support I^2C type. For more information about how to use this API, and the relationship between IIC and SMBus, please refer to "Programming Overview", parts "SMBus functions" to "IIC versus SMBus – compatibility".

SusiIICWriteReadCombine

A sequential operation to write bytes of data followed by bytes read from the target slave device in the I^2C bus.

```
BOOL SusiIICWriteReadCombine(DWORD IICType, BYTE
SlaveAddress, BYTE *WriteBuf, DWORD WriteLen, BYTE *ReadBuf,
DWORD ReadLen);
```

Parameters

IICType

[in] Specifies the I^2C type, the value can either be
SUSI_IIC_TYPE_PRIMARY (1)
SUSI_IIC_TYPE_SMBUS (2)

SlaveAddress

[in] Specifies the 8-bit device address, ranging from 0x00 – 0xFF.
Whether to give a 1 (read) or 0 (write) to the LSB of SlaveAddress could be ignored.

WriteBuf

[in] Pointer to a byte array which contains the bytes of data to be written.

WriteLen

[in] Specifies the number of bytes to be written.

ReadBuf

[out] Pointer to a variable in which the function reads the bytes of data.

ReadLen

[in] Specifies the number of bytes to be read.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

The function is mainly for EEPROM I^2C devices - the bytes written first are used to locate to a certain address in ROM, and the following bytes read will retrieve the data bytes starting from this address.

Call SusiIICAvailable first to make sure the support I^2C type. For more information about how to use this API, and the relationship between IIC and SMBus, please refer to "Programming Overview", parts "SMBus functions" to "IIC versus SMBus – compatibility"

SusiVCAvailable

Check if VC driver is available and also get the feature support information.

```
BOOL SusiVCAvailable(void);
```

Parameters

None.

Return Value

value	Meaning
-1	The function fails.
0	The function succeeds; the platform does not support any SusiVC- APIs.
SUSI_VC_BRIGHT_CONTROL_AVAILABLE (1)	The function succeeds; the platform supports only brightness APIs.
SUSI_VC_VGA_CONTROL_AVAILABLE (2)	The function succeeds; the platform supports only screen on/off APIs.
SUSI_VC_BOTH_AVAILABLE (3)	The function succeeds; the platform supports all SusiVC- APIs.

Remarks

After calling `SusiDllInit` successfully, all `Susi*Available` functions are use to check if the corresponding features are supported by the platform or not. So it is suggested to call `Susi*Available` before using any `Susi*-functions`.

SusiVCGetBrightRange

Get the step, minimum and maximum values in brightness adjustment.

```
BOOL SusiVCGetBrightRange(BYTE *minimum, BYTE *maximum,
BYTE *stepping);
```

Parameters

minimum

[out] Pointer to a variable to get the minimum brightness value.

maximum

[out] Pointer to a variable to get the maximum brightness value.

stepping

[out] Pointer to a variable to get the step of brightness up and down

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

Call `SusiVCAvailable` first to make sure if the brightness control is available. The values may vary from platform to platform; depend on the hardware implementations of brightness control. For example, if minimum is 0, maximum is 255, and stepping is 5, it means the brightness can be 0, 5, 10, …, 255.

SusiVCGetBright

Get the current panel brightness.

```
BOOL SusiVCGetBright(BYTE *brightness);
```

Parameters

brightness

[out] Pointer to a variable in which this function returns the brightness.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

Call `SusiVCAvailable` first to make sure if the brightness control is available.

SusiVCSetBright

Set current panel brightness.

```
BOOL SusiVCSetBright(BYTE brightness);
```

Parameters

brightness

[in] Specifies the brightness value to be set.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

Call `SusiVCAvailable` first to make sure if the brightness control is available.

In some implementations, the higher the brightness value, the higher the voltage fed to the panel. So please make sure the voltage toleration of your panel prior to the API use.

SusiVCScreenOn

Turn on VGA display signal.

```
BOOL SusiVCScreenOn(void);
```

Parameters
None.

Return Value
TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks
The function enables both the LCD and CRT display signals.

SusiVCScreenOff

Turn off VGA display signal.

```
BOOL SusiVCScreenOff(void);
```

Parameters

None.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

The function disables both the LCD and CRT display signals.

SusiHWMAvailable

Check if the hardware monitor driver is available.

```
int SusiHWMAvailable();
```

Parameters

None.

Return Value

value	Meaning
-1	The function fails.
0	The function succeeds; the platform does not support SusiHWM- APIs.
1	The function succeeds; the platform supports HWM.

Remarks

After calling `SusiDllInit` successfully, all `Susi*Available` functions are use to check if the corresponding features are supported by the platform or not. So it is suggested to call `Susi*Available` before using any `Susi*-` functions.

SusiHWMGetFanSpeed

Read the current value of one of the fan speed sensors, or get the types of available sensors.

```
BOOL SusiHWMGetFanSpeed(WORD fanType, WORD *retval, WORD
*typeSupport = NULL);
```

Parameters

fantype

[in] Specifies a fan speed sensor to get value from. It can be one of the flags

FCPU (1) – CPU Fan

FSYS (2) – System / Chassis fan

retval

[out] Point to a variable in which this function returns the fan speed in RPM

Typesupport

[out]

If the value is specified as a pointer (non-NULL) to a variable, it will return the types of available sensors in flags bitwise-ORed

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

Call the function first with a non-NULL `typesupport` to know the available fan sensors and a following call to get the fan speed required.

SusiHWMGetTemperature

Read the current value of one of the temperature sensors, or get the types of available sensors.

```
BOOL SusiHWMGetTemperature(WORD tempType, float *retval,
WORD *typeSupport = NULL);
```

Parameters

tempType

[in] Specifies a temperature sensor to get value from. It can be one of the flags

TCPU (1) – CPU temperature

TSYS (2) – System / ambient temperature

retval

[out] Point to a variable in which this function returns the temperature in Celsius.

Typesupport

[out]

If the value is specified as a pointer (non-NULL) to a variable, it will return the types of available sensors in flags bitwise-ORed

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

Call the function first with a non-NULL typesupport to know the available temperature sensors and a following call to get the temperature required.

SusiHWMGetVoltage

Read the current value of one of the voltage sensors, or get the types of available sensors.

```
BOOL SusiHWMGetVoltage(DWORD voltType, float *retval,
DWORD *typeSupport = NULL);
```

Parameters

voltType

[in] Specifies a voltage sensor to get value from. It can be one of the flags

VCORE (1<<0)
V25 (1<<1)
V33 (1<<2)
V50 (1<<3)
V120 (1<<4)
VSB (1<<5)
VBAT (1<<6)
VN50 (1<<7)
VN120 (1<<8)
VTT (1<<9)

retval

[out] Point to a variable in which this function returns the voltage in Volt.

Typesupport

[out]

If the value is specified as a pointer (non-NULL) to a variable, it will return the types of available sensors in flags bitwise-ORed

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

Call the function first with a non-NULL typesupport to know the available fan sensors and a following call to get the voltage required.

SusiHWMSetFanSpeed

Control the speed of one of the fans, or get the types of available fans.

```
BOOL SusiHWMSetFanSpeed(WORD fanType, BYTE setval, WORD
*typeSupport = NULL);
```

Parameters

fantype

[in] Specifies a fan to be controlled. It can be one of the flags

FCPU (1) – CPU Fan

FSYS (2) – System / Chassis fan

setval

[in] Specifies the value to set, ranging from 0 to 255.

Typesupport

[out]

If the value is specified as a pointer (non-NULL) to a variable, it will return the types of available fans in flags bitwise-ORed

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

The fan speed is controlled by Pulse Width Modulation (PWM):
Duty cycle (%) = (setval/ 255) * 100%
And the default duty cycle is set to 100%, i.e. the maximal fan speed.
Call the function first with a non-NULL typesupport to know the available fan sensors and a following call to set the fan speed.

SUSIFPGA API Programmer's Documentation

All APIs return the `BOOL` data type except `SUSIFPGA*Available` and some special cases that are of type `int`. If any function call fails, i.e. a `BOOL` value of FALSE, or an `int value` of -1, the error code can always be retrieved by an immediate call to SUSIFPGA`GetLastError.`

SUSIFPGADllInit

Initialize the SUSIFPGA Library.

```
BOOL SUSIFPGADllInit(void);
```

Parameters

None.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

An application must call `SUSIFPGADllInit` before calling any other non SUSIFPGADll- functions.

SUSIFPGADllUnInit

Uninitialize the SUSIFPGA Library.

```
BOOL SUSIFPGADllUnInit(void);
```

Parameters

None.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

Before an application terminates, it must call `SUSIFPGADllUnInit` if it has successfully called `SUSIFPGADllInit`. Calls to `SUSIFPGADllInit` and `SUSIFPGADllUnInit` can be nested but must be paired.

.

SUSIFPGAStorageAreaGetType

To get types of storage areas a platform supports.

```
BOOL SUSIFPGAStorageAreaGetType (DWORD *dwType);
```

Parameters

dwType

[in] Pointer to a variable to get the type of storage area supported in a platform.

Return Value

value	Meaning
0	None
1	FPGA SRAM
2	EEPROM
4	CMOS

SUSIFPGAStorageAreaGetSize

To get size of storage areas a platform supports.

```
BOOL SUSIFPGAStorageAreaGetsize (DWORD dwType, DWORD
*size);
```

Parameters

dwType

[in] Pointer to a variable to get the type of storage area supported in a platform.

Size

[out] Pointer to a variable to get the size of storage area

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

SUSIFPGAStorageAreaRead

Read storage area data

```
BOOL SUSIFPGAStorageAreaRead (DWORD dwType, DWORD dwOffset,
BYTE *pbData, DWORD dwLen);
```

Parameters

dwType

[in] Pointer to a variable to get the type of storage area supported in a platform.

dwOffset

[in] Zero based offset into the storage area

pdData

[out] Pointer to location that will receive the bytes

dwLen

[in] Number of bytes to read

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

SUSIFPGAStorageAreaWrite

.

```
BOOL SUSIFPGAStorageAreaWrite (DWORD dwType, DWORD
dwOffset, BYTE *pbData, DWORD dwLen);
```

Parameters

dwType

[in] Pointer to a variable to get the type of storage area supported in a platform.

dwOffset

[in] Zero based offset into the storage area

pdData

[out] Pointer to location that will receive the bytes

dwLen

[in] Number of bytes to write

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

SUSIFPGAStorageAreaErase

.

```
BOOL SUSIFPGAStorageAreaErase(DWORD dwType, DWORD dwOffset,
DWORD dwLen);
```

Parameters

dwType

[in] Pointer to a variable to get the type of storage area supported in a platform.

dwOffset

[in] Zero based offset into the storage area

dwLen

[in] Number of bytes to erase

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

SUSIFPGAStorageAreaFPGAConfig

A structure of FPGA configuration.

```
BOOL SUSIFPGAStorageAreaAccessFPGAConfig(DWORD
mode,PSSFPGA_SRAM pSRAM)
```

Parameters

mode

[in] The value can be either

ESCORE_BOOTCOUNTER_MODE_GET (0)

- Get information from counter.

ESCORE_BOOTCOUNTER_MODE_SET (1)

- Set information to counter.

pSRAM

[in/out]

Pointer to a SSFPGA_SRAM structure to set or get the SRAM information in FPGA, please see next page.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

SRAM

This structure represents the SRAM information.

```
Typedef struct {
BOOL isCRC;
BOOL isDisableAES;
BOOL isMirror;
} SSFPGA_SRAM, *PSSFPGA_SRAM;
```

Members

isCRC

0	Disable
1	Enable

isAES

0	False; SRAM data is Encryption.
1	True; SRAM data is Decryption.

isMirror

0	Disable
1	Enable

Remarks

The storage area size would be changed if you enable CRC or Mirror function. Please make sure to get storage size again with `SUSIFPGAStorageAreaGetsize` after configure the FPGA.

SUSIFPGAIOFPGACountEx

Query the current number of input and output pins.

```
BOOL SUSIFPGAIOFPGACountEx(DWORD *inCount, DWORD
*outCount);
```

Parameters

inCount

[out] Pointer to a variable in which this function returns the count of input pins.

outCount

[out] Pointer to a variable in which this function returns the count of output pins.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

The number of total GPIO pins equals the number of input pins plugs the number of output pins. With this constant pin number, the numbers of input and output pins may vary in accordance with current pin direction.

SUSIFPGAIOFPGAReadEx

Read current status of one FPGA GPIO input or output pin.

```
BOOL SUSIFPGAIOFPGAReadEx(BYTE PinNum, BOOL *status);
```

Parameters

PinNum

[in] Specifies the GPIO pin demanded to be read, ranging from 0 ~ (total number of GPIO pins minus 1).

status

[out] Pointer to a variable in which the pin status returns.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

If the pin is in status high, the value got in `status` will be 1. If the pin is in status low, it will be zero. The function is capable of reading the status of either an input pin or an output pin.

SUSIFPGAIOFPGAReadMultiEx

Read current statuses of multiple pins at once regardless of the pin directions.

```
BOOL SUSIFPGAIOFPGAReadMultiEx(UINT64 PinMaskHi, UINT64
PinMaskLo, UINT64 *StatusMaskHi, UINT64 *StatusMaskLo);
```

Parameters

PinMaskHi

[in] GPIO Pin64 to Pin71 demanded to be read.

PinMaskLo

[in] GPIO Pin0 to Pin63 demanded to be read.

StatusMaskHi

[out] Statuses of pins in Bitwise-ORed. For pins that are not specified in `PinMaskHi`, the related bit value is invalid.

StatusMaskLo

[out] Statuses of pins in Bitwise-ORed. For pins that are not specified in `PinMaskLo`, the related bit value is invalid.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

For example, if you want to read the statuses of GPIO pin 0, 1, 6, 7. Give parameter `PinMaskLo` with a value 11000011, or 0xC3. Bit 0 stand for GPIO 0, bit 1 stand for GPIO 1, and so on. Again, if the pin is in status high, the value got in relevant bit of `StatusMaskLo` will be 1. If the pin is in status low, it will be zero.

SUSIFPGAIOFPGAWriteEx

Set one GPIO output pin as status high or low.

```
BOOL SUSIFPGAIOFPGAWriteEx(BYTE PinNum, BOOL status);
```

Parameters

PinNum

[in] Specifies the GPIO pin demanded to be written, ranging from 0 ~ (total number of GPIO pins minus 1).

status

[in] Specifies the GPIO status to be written.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

The function can only set the status of one of the output pins. If the pin number specified is an input pin or an invalid pin, the function call will fail and return with FALSE. A `status` with 1 to set the pin as output high, 0 to set the pin as output low.

SUSIFPGAIOFPGAWriteMultiEx

Set statuses of multiple output pins at once.

```
BOOL SUSIFPGAIOFPGAWriteMultiEx(UINT64 PinMaskHi, UINT64
PinMaskLo, UINT64 StatusMaskHi, UINT64 StatusMaskLo);
```

Parameters

PinMaskHi

[in] GPIO Pin64 to Pin71 demanded to be written.

PinMaskLo

[in] GPIO Pin0 to Pin63 demanded to be written.

StatusMaskHi

[out] Statuses of pins in Bitwise-ORed. For pins that are not specified in `PinMaskHi`, the related bit value is invalid.

StatusMaskLo

[out] Statuses of pins in Bitwise-ORed. For pins that are not specified in `PinMaskLo`, the related bit value is invalid.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Remarks

For example, if you want to write the statuses of GPIO output pin 0, 1, 6, 7. Give parameter `PinMaskLo` with a value 11000011, or 0xC3. Bit 0 stand for GPIO 0, bit 1 stand for GPIO 1, and so on.

If you want to set pin 0 as high, pin 1 as low, pin 6 as high and pin 7 as low. Give parameter `StatusMaskLo` with a value 01XXXX01, X is for don't care pin, you could simply assign a 0 for it, i.e. 0x41.

SUSIFPGASecurityFPGASetAESKey

Set length 16 bytes AES Key

```
BOOL SUSIFPGASecurityFPGASetAESKey(BYTE *pbKey);
```

Parameters

pdKey

[in] Pointer to 16 bytes array which contains the bytes of Key to be written.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

SUSIFPGASecurityFPGAGetAESKey

Get length 16 bytes AES Key

```
BOOL SUSIFPGASecurityFPGAGetAESKey(BYTE *pbKey);
```

Parameters

pbKey

[out] Pointer to 16 bytes array which contains the bytes of Key to be read.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

SUSIFPGASecurityFPGAGenerateAESData

Generate 16 bytes AES Data

```
BOOL SUSIFPGASecurityFPGAGenerateAESData (BYTE *pbData);
```

Parameters

pbData

[out] Pointer to a 16 bytes array which contains the bytes of data to be read.

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

SUSIFPGASecurityFPGAGenerateRandomNum

Generate a random number

```
BOOL SUSIFPGASecurityFPGAGenerateRandomNum (DWORD *num);
```

Parameters

num

[out] Pointer to a variable in which get the random number

Return Value

TRUE (1) indicates success; FALSE (0) indicates failure.

Appendix A - GPIO Information

Look up the table for the GPIO pins assignment and the default pins direction for a platform. E.g. AIMB-330(CN19) means that the platform name is AIMB-330 and its GPIO pins are located in CN19 on the board.

AIMB-330(CN19)/ AIMB-340(CN19)/ AIMB-640(CN18)

The number of GPIO pins：4 Inputs, 4 outputs

Pin	Signal	Pin	Signal
Pin-1	IN0	Pin-2	+5V
Pin-3	IN1	Pin-4	***OUT0 (Max 1A)***
Pin-5	IN2	Pin-6	GND
Pin-7	IN3	Pin-8	***OUT1 (Max 1A)***
Pin-9	GND	Pin-10	+12V
Pin-11	Key	Pin-12	Key
Pin-13	POUT3	Pin-14	GND
Pin-15	OUT2	Pin-16	+12V

***.** It should add the pull-up resistors to *OUT0*, *OUT1* on **AIMB-330**, **AIMB-340** and **AIMB-640**.

---

*PCM-3350(CN36,CN37)/PCM-3353(CN36,CN37)/PCM-3372(CN2,CN23)/PCM-4153(CN36,CN37)

`*PCM-XXXX(IN,OUT)`

The number of GPIO pins：4 Inputs, 4 outputs

IN		OUT	
Pin	**Signal**	**Pin**	**Signal**
Pin-1	VCC	Pin-1	OUT0
Pin-2	IN0	Pin-2	OUT1
Pin-3	IN1	Pin-3	OUT2
Pin-4	IN2	Pin-4	OUT3

Pin-5	IN3	Pin-5	GND

PCM-4372(CN2)/PCM-4386(CN7)/PCM-4380(CN7)/ PCM-4390(CN6)/PCM-9374(CN4)/PCM-9375(CN9)/ PCM-9377(27)/PCM-9380(CN7)/PCM-9386(CN7)/ PCM-9577(CN25)/PCM-9584(CN16)/PCM-9586(CN9)/ PCM-9679(CN7)

The number of GPIO pins：4 Inputs, 4 outputs

Pin	Signal	Pin	Signal
Pin-1	VCC	Pin-2	OUT0
Pin-3	IN0	Pin-4	OUT1
Pin-5	IN1	Pin-6	OUT2
Pin-7	IN2	Pin-8	OUT3
Pin-9	IN3	Pin-10	GND

***.** It should add the pull-up resistors to the input pins on **PCM-9577** for logic level.

PCM-9381(CN7)/ PCM-9387(CN7)

The number of GPIO pins：4 Inputs

Pin	Signal
Pin-1	VCC
Pin-2	IN0
Pin-3	IN1
Pin-4	IN2
Pin-5	IN3

PCM-9578(CN5)

The number of GPIO pins：4 Inputs, 4 outputs

Pin	Signal	Pin	Signal
Pin-1	OUT0	Pin-2	OUT1
Pin-3	OUT2	Pin-4	OUT3
Pin-5	OUT4	Pin-6	OUT5
Pin-7	OUT6	Pin-8	OUT7

Pin-9	GND	Pin-10	GND

PCM-9580(CN16)

The number of GPIO pins：4 Inputs, 4 outputs

Pin	Signal	Pin	Signal
Pin-1	IN0	Pin-2	OUT0
Pin-3	IN1	Pin-4	OUT1
Pin-5	IN2	Pin-6	OUT2
Pin-7	IN3	Pin-8	OUT3
Pin-9	GND	Pin-10	GND

PCM-9581(CN9)/ PCM-9582(CN19)/ PCM-9586(CN9)/ PCM-9587(CN19)/PCI-6681(CN16)

The number of GPIO pins：4 Inputs, 4 outputs

Pin	Signal	Pin	Signal
Pin-1	IN0	Pin-2	OUT0
Pin-3	GND	Pin-4	GND
Pin-5	IN1	Pin-6	OUT1
Pin-7	VCC	Pin-8	NC
Pin-9	IN2	Pin-10	OUT2
Pin-11	GND	Pin-12	GND
Pin-13	IN3	Pin-14	OUT3

***.** It should add the pull-up resistors to *In2*, *In3*, *OUT0*, *OUT1* on **PCM-9581** and **PCM-9586**.

PCI-6880 (CN2)

The number of GPIO pins：4 Inputs, 4 outputs

Pin	Signal	Pin	Signal
Pin-1	IN0	Pin-2	OUT0
Pin-3	IN1	Pin-4	OUT1
Pin-5	IN2	Pin-6	OUT2
Pin-7	IN3	Pin-8	OUT3
Pin-9	VCC	Pin-10	GND

SOM-5780(U17)/SOM-5782(U14)

The number of GPIO pins：4 Inputs, 4 outputs

Pin	Signal	Pin	Signal
Pin-1	VCC 3.3V	Pin-16	GND
Pin-4	IN2	Pin-20	OUT3
Pin-5	IN3	Pin-21	OUT2
Pin-11	IN0	Pin-22	OUT1
Pin-12	IN1	Pin-23	OUT0

*. SOM-5780, SOM-5782 must combine with SOM-DB5700(carrier board).

SOM-DB5700(CN27)			
Pin-1	IN0	Pin-2	VCC
Pin-3	IN1	Pin-4	OUT0
Pin-5	IN2	Pin-6	OUT1
Pin-7	IN3	Pin-8	OUT2
Pin-9	GND	Pin-10	+12V
Pin-11	NC	Pin-12	NC
Pin-13	OUT3	Pin-14	NC
Pin-15	GND	Pin-16	+12V

PCM-3375(CN16)

The number of GPIO pins：4 Inputs, 4 outputs

Pin	Signal
Pin-1	-5V
Pin-2	GND
Pin-3	-12V
Pin-19	IN0
Pin-20	IN1
Pin-21	IN2
Pin-22	IN3
Pin-23	OUT0
Pin-24	OUT1
Pin-25	OUT2
Pin-26	OUT3

***.** There are two high drive digital outputs, *OUT0*, *OUT1* (24 VDC, 1 A max), two TTL level digital outputs, OUT2, OUT3 and four digital inputs (TTL level). You can configure the digital I/O to control the opening of the cash drawer and to sense the closing of the cash drawer. The above table explains how the digital I/O is controlled via software programming and how a 12 V solenoid or relay can be triggered. For completeness, please refer to the user manual of POS-563/POS-564/POS-761.

Appendix B – Programming Flags Overview

Hardware Monitor Flags

- **Fan**

Flag	Value	Description
FCPU	1u	CPU FAN
FSYS	2u	System FAN
F2ND	4u	3rd FAN

- **Temperature**

Flag	Value	Description
TCPU	1u	CPU Temperature
TSYS	2u	System Temperature

- **Voltage**

Flag	Value	Description
VCORE	1u	Vcore
V25	2u	2.5V
V33	4u	3.3V
V50	8u	5V
V120	16u	12V
VSB	32u	Voltage of standby
VBAT	64u	VBAT
VN50	128u	-5V
VN120	256u	-12V
VTT	512u	VTT

Boot Logger Flags

- **Bootcounter**

Mode Flag	Value	Description
ESCORE_BOOTCOUNTER_MODE_GET	1u	Read Operation
ESCORE_BOOTCOUNTER_MODE_SET	2u	Write Operation

Element Flag	Value	Description
ESCORE_BOOTCOUNTER_STATUS	1u	Current Status (Is Enabled or Disabled?)
ESCORE_BOOTCOUNTER_VALUE	2u	Number of Reboot Times

- ## Runtimer

Mode Flag	Value	Description
ESCORE_RUNTIMER_MODE_GET	1u	Read Operation
ESCORE_RUNTIMER_MODE_SET	2u	Write Operation

Element Flag	Val.	Description
ESCORE_RUNTIMER_STATUS_RUNNING	1u	Current Status (Is Enabled or Disabled?)
ESCORE_RUNTIMER_STATUS_AUTORUN	2u	Is AutoRun upon Startup?
ESCORE_RUNTIMER_VALUE_CONTINUALON	4u	OS continual run time (reset to 0 after a reboot)
ESCORE_RUNTIMER_VALUE_TOTALON	8u	Sum of OS total run time

GPIO Mask Flags

Flag	Value	Description
ESIO_SMASK_PIN_FULL	0x01	Series of binary 1s for the number of total pins
ESIO_SMASK_CONFIGURABLE	0x02	Direction Changeable Pins
ESIO_DMASK_DIRECTION	0x20	Current Direction of Pins

Appendix C - API Error Codes

An error value will be either

Function Index Code + Library Error Code, or

Function Index Code + Driver Error Code.

If you call an API and returns with fail. The `Function Index Code` in its error code combination does not necessarily equal to the index code of the API. This is because the API may make a call to another API.

Function Index Code

Index Code	Function Index
DLL	
00100000	ESusiInit
00200000	ESusiUnInit
00300000	ESusiGetVersion
00400000	ESusiDllInit
00500000	ESusiDllUnInit
00600000	ESusiDllGetVersion
00700000	ESusiDllGetLastError
Core	
10100000	ESusiCoreInit
10200000	ESusiCoreAvailable
10300000	ESusiCoreGetBIOSVersion
10400000	ESusiCoreGetPlatformName
10500000	ESusiCoreAccessBootCounter
10600000	ESusiCoreAccessRunTimer
10700000	ESusiCoreRebootSystem
10800000	ESusiReserved8000000
Watchdog	
20100000	ESusiWDInit
20200000	ESusiWDAvailable
20300000	ESusiWDDisable
20400000	ESusiWDGetRange
20500000	ESusiWDSetConfig
20600000	ESusiWDTrigger
GPIO	
30100000	ESusiIOInit
30200000	ESusiIOAvailable
30300000	ESusiIOCount
30400000	ESusiIOInitial

30500000	ESusiIORead
30600000	ESusiIOReadMulti
30700000	ESusiIOWrite
30800000	ESusiIOWriteMulti
30900000	ESusiIOCountEx
31000000	ESusiIOQueryMask
31100000	ESusiIOSetDirection
31200000	ESusiIOSetDirectionMulti
31300000	ESusiIOReadEx
31400000	ESusiIOReadMultiEx
31500000	ESusiIOWriteEx
31600000	ESusiIOWriteMultiEx
SMBus	
40100000	ESusiSMBusInit
40200000	ESusiSMBusAvailable
40300000	ESusiSMBusReadByte
40400000	ESusiSMBusReadByteMulti
40500000	ESusiSMBusReadWord
40600000	ESusiSMBusWriteByte
40700000	ESusiSMBusWriteByteMulti
40800000	ESusiSMBusWriteWord
40900000	ESusiSMBusReceiveByte
41000000	ESusiSMBusSendByte
41100000	ESusiSMBusWriteQuick
41200000	ESusiSMBusReadQuick
41300000	ESusiSMBusScanDevice
41400000	ESusiSMBusWriteBlock
41500000	ESusiSMBusReadBlock
IIC	
50100000	ESusiIICInit
50200000	ESusiIICAvailable
50300000	ESusiIICReadByte
50400000	ESusiIICWriteByte
50500000	ESusiIICWriteReadCombine
50600000	ESusiIICRead
50700000	ESusiIICWrite
50800000	ESusiIICScanDevice
50900000	ESusiIICWriteRegister
51000000	ESusiIICReadRegister
VGA Control	
60100000	ESusiVCInit
60200000	ESusiVCAvailable
60300000	ESusiVCGetBright
60400000	ESusiVCGetBrightRange
60500000	ESusiVCScreenOff
60600000	ESusiVCScreenOn

60700000	ESusiVCSetBright
Hardware Monitor	
70100000	ESusiHWMInit
70200000	ESusiHWMAvailable
70300000	ESusiHWMGetFanSpeed
70400000	ESusiHWMGetTemperature
70500000	ESusiHWMGetVoltage
70600000	ESusiHWMSetFanSpeed

Library Error Code

Error Code	Error Type
Driver Open Errors	
00000001	ERRLIB_CORE_OPEN_FAIL
00000002	ERRLIB_WDT_OPEN_FAIL
00000004	ERRLIB_GPIO_OPEN_FAIL
00000008	ERRLIB_SMB_OPEN_FAIL
00000016	ERRLIB_VC_OPEN_FAIL
00000032	ERRLIB_HWM_OPEN_FAIL
DLL Functions	
00000000	ERRLIB_SUCCESS
00000001	ERRLIB_RESERVED1
00000002	ERRLIB_RESERVED2
00000003	ERRLIB_LOGIC
00000004	ERRLIB_RESERVED4
00000005	ERRLIB_SUSIDLL_NOT_INIT
00000006	ERRLIB_PLATFORM_UNSUPPORT
00000007	ERRLIB_API_UNSUPPORT
00000008	ERRLIB_RESERVED8
00000009	ERRLIB_API_CURRENT_UNSUPPORT
00000010	ERRLIB_LIB_INIT_FAIL
00000011	ERRLIB_DRIVER_CONTROL_FAIL
00000012	ERRLIB_INVALID_PARAMETER
00000013	ERRLIB_INVALID_ID
00000014	ERRLIB_CREATEMUTEX_FAIL
00000015	ERRLIB_OUTBUF_RETURN_SIZE_INCORRECT
00000016	ERRLIB_RESERVED16
00000017	ERRLIB_ARRAY_LENGTH_INSUFFICIENT
00000032	ERRLIB_RESERVED32
00000050	ERRLIB_BRIGHT_CONTROL_FAIL
00000051	ERRLIB_BRIGHT_OUT_OF_RANGE
00000064	ERRLIB_RESERVED64
00000128	ERRLIB_RESERVED128
00000256	ERRLIB_RESERVED256
Core Functions	
00000500	ERRLIB_CORE_BIOS_STRING_NOT_FOUND
00000512	ERRLIB_RESERVED512
Watchdog Functions	
00001024	ERRLIB_RESERVED1024
GPIO Functions (N/A)	
SMBus Functions	
00001400	ERRLIB_SMB_MAX_BLOCK_SIZE_MUST_WITHIN_32
IIC Functions	
00001600	ERRLIB_IIC_GETCPUFREQ_FAIL

VGA Control Functions (N/A)	
Hardware Monitor Functions	
00002000	ERRLIB_HWM_CHECKCPUTYPE_FAIL
00002001	ERRLIB_HWM_FUNCTION_UNSUPPORT
00002002	ERRLIB_HWM_FUNCTION_CURRENT_UNSUPPORT
00002003	ERRLIB_HWM_FANDIVISOR_INVALID
00002048	ERRLIB_RESERVED2048
Reserved Functions	
00004096	ERRLIB_RESERVED4096
00008192	ERRLIB_RESERVED8192

Driver Error Code

Error Code	Error Type
00000000	ERRDRV_SUCCESS
Common to all Drivers	
00010000	ERRDRV_CTRLCODE
00010001	ERRDRV_LOGIC
00010002	ERRDRV_INBUF_INSUFFICIENT
00010003	ERRDRV_OUTBUF_INSUFFICIENT
00010004	ERRDRV_STOPTIMER_FAILED
00010005	ERRDRV_STARTTIMER_FAILED
00010006	ERRDRV_CREATEREG_FAILED
00010007	ERRDRV_OPENREG_FAILED
00010008	ERRDRV_SETREGVALUE_FAILED
00010009	ERRDRV_GETREGVALUE_FAILED
00010010	ERRDRV_FLUSHREG_FAILED
00010011	ERRDRV_MEMMAP_FAILED
Core Driver (N/A)	
Watchdog Driver (N/A)	
GPIO Driver	
00011200	ERRDRV_GPIO_PIN_DIR_CHANGED
00011201	ERRDRV_GPIO_PIN_INCONFIGURABLE
00011202	ERRDRV_GPIO_PIN_OUTPUT_UNREADABLE
00011203	ERRDRV_GPIO_PIN_INPUT_UNWRITTABLE
00011204	ERRDRV_GPIO_INITIAL_FAILED
00011205	ERRDRV_GPIO_GETINPUT_FAILED
00011206	ERRDRV_GPIO_SETOUTPUT_FAILED
00011207	ERRDRV_GPIO_GETSTATUS_IO_FAILED
00011208	ERRDRV_GPIO_SETSTATUS_OUT_FAILED
00011209	ERRDRV_GPIO_SETDIR_FAILED
SMBus Driver	
00011400	ERRDRV_SMB_RESETDEV_FAILED
00011401	ERRDRV_SMB_TIMEOUT
00011402	ERRDRV_SMB_BUSTRANSACTION_FAILED
00011403	ERRDRV_SMB_BUSCOLLISION
00011404	ERRDRV_SMB_CLIENTDEV_NORESPONSE
00011405	ERRDRV_SMB_REQUESTMASTERMODE_FAILED
00011406	ERRDRV_SMB_NOT_MASTERMODE
00011407	ERRDRV_SMB_BUS_ERROR
00011408	ERRDRV_SMB_BUS_STALLED
00011409	ERRDRV_SMB_NEGACK_DETECTED
00011410	ERRDRV_SMB_TRANSMITMODE_ACTIVE
00011411	ERRDRV_SMB_TRANSMITMODE_INACTIVE
00011412	ERRDRV_SMB_STATE_UNKNOWN
IIC Driver	

00011600	ERRDRV_IIC_RESETDEV_FAILED
00011601	ERRDRV_IIC_TIMEOUT
00011602	ERRDRV_IIC_BUSTRANSACTION_FAILED
00011603	ERRDRV_IIC_BUSCOLLISION
00011604	ERRDRV_IIC_CLIENTDEV_NORESPONSE
00011605	ERRDRV_IIC_REQUESTMASTERMODE_FAILED
00011606	ERRDRV_IIC_NOT_MASTERMODE
00011607	ERRDRV_IIC_BUS_ERROR
00011608	ERRDRV_IIC_BUS_STALLED
00011609	ERRDRV_IIC_NEGACK_DETECTED
00011610	ERRDRV_IIC_TRANSMITMODE_ACTIVE
00011611	ERRDRV_IIC_TRANSMITMODE_INACTIVE
00011612	ERRDRV_IIC_STATE_UNKNOWN
VGA Control Driver	
00011800	ERRDRV_VC_FINDVGA_FAILED
00011801	ERRDRV_VC_FINDBRIGHTDEV_FAILED
00011802	ERRDRV_VC_VGA_UNSUPPORTED
00011803	ERRDRV_VC_BRIGHTDEV_UNSUPPORTED
Hardware Monitor Driver (N/A)	