SUSI Library

User's Manual

Version 1.1

Advantech Co., Ltd.
No. 1, Alley 20, Lane 26, Reuiguang Road,
Neihu District, Taipei, Taiwan 114, R.O.C.

Copyright Notice

Part No.
Version: 1.1

Printed in Taiwan 2006-10-05

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

Table of Contents

Introduction

To make hardware easier and convenient to access for programmers, Advantech releases a suite of API (Application Programming Interface) in the form of a program library. The program Library is called the "Secured and Unified Smart Interface" Library and it is also referred to as the Susi Library hereafter. Customer should purchase the Advantech Software Library license stickers from Advantech prior to ship or distribute his/her developed software based on the Susi Software Library. Each CPU board or computer system should have an Advantech Embedded Software IP - Library license sticker properly affixed on it before shipping/distributing or any commercial purpose.

In modern operating systems, user space applications cannot access hardware directly. Drivers are required to access hardware. User space applications access hardware through drivers. Different operating systems usually define different interface for drivers. It means that user space applications call different functions for hardware access in different operating systems. To provide a uniform interface for accessing hardware, an abstraction layer is built on top of the drivers and the Susi is such an abstraction layer. The Susi provides a uniform API for application programmers to access the hardware functions in different operating systems and on different Advantech hardware platforms.

Application programmers should invoke the functions exported by the Susi instead of calling the drivers directly. The benefit of using the Susi is portability. The same set of API is defined for different Advantech hardware platforms. Besides, the same set of API is implemented in different operating systems including Windows XP and Windows CE. This user's manual describes some sample programs and the API in the Susi. The hardware functions currently supported by the Susi can be grouped into a few categories including Watchdog, I^2C, SMBus, GPIO, HWM and VGA control. Each category of the API in the Susi is briefly described below.

The GPIO API

General Purpose Input/Output (GPIO) is a flexible parallel interface that allows a variety of custom connections. Supports Digital I/O Devices. You can control cash drawers , LED light or buttons with GPIO.

The I²C API

I²C is a bi-directional two wire bus that was developed by Philips for use in their televisions in the 1980s and it is used in various types of embedded systems nowadays. The strict timing requirements defined in the I²C protocol has been taken care of by the Susi. Instead of asking application programmers to figure out the strict timing requirements in the I²C protocol, the I²C API in the Susi can be used to control I²C devices just like invoking other function calls. Therefore, the development process of your products can be sped up by using the Susi. Besides, the Susi provides a consistent programming interface for different Advantech boards. That means user programs using the Susi are portable among different Advantech boards as long as the boards and the Susi provide the required functionalities.

The SMBus API

The System Management Bus (SMBus) is a two-wire interface defined by Intel ® Corporation in 1995. It is based on the principles of operations of I²C and it is used in personal computers and servers for low-speed system management communications. It can be seen in many types of embedded systems. As with other API in the Susi, the SMBus API is available on many platforms including Windows XP and Windows CE.

The VGA Control API

There are two kinds of VGA control APIs, backlight on/off control and brightness control, in the Susi. Backlight on/off control can allow a developer to turn on or turn off the backlight. Our API allows a developer to turn on /off the backlight and to control brightness smoothly.

The Watchdog API

A watchdog timer (abbreviated as WDT) is a hardware device which triggers an action, e.g. rebooting the system, if the system does not reset the timer within a specific period of time. The WDT API in the Susi provides developers with functions such as starting the timer, reset the timer, and set the timeout value if the hardware supports customized timeout value.

The Hardware Monitor API

The hardware monitor (abbreviated as HWM) is a system health supervision capability achieved by placing certain I/O chip along with sensors for inspecting the target of interests for certain condition indexes, such as fan speed, temperature, and voltage...etc.

However, due to the common flaw of inaccuracy among all commercial available hardware monitoring chips, Advantech, your trusted ePlatform provider, has developed an unique scheme for hardware monitoring - achieved by using a dedicated micro-processor with algorithms specifically designed for providing a accurate, real-time and reliable data content, therefore, to help protect your system in a more reliable manner.

Environments

The Susi supports many different operating systems including:

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

Many hardware boards from Advantech are supported by the Susi. Please refer to Appendix A for the list of the hardware platforms currently supported by the Susi. Note that the list may be changed without notification. For an updated list of supported platforms, please check the web site at *www.advantech.com*. Should you have any questions about your Advantech boards, please contact us by phone call or e-mail.

Package Contents

The Susi Library supports two operating systems including Windows CE and Windows XP. After the library package is extracted, you can find the directories and files listed in the table below.

Directory	Contents
\	• `SusiAPI.pdf` Please refer to this document for installation and application development. • `Change History.html` Revision history of Susi.
\Change History.Files	Required files for `Change History.html`
\Include	• `Susi.h` A header file of Susi export functions. The header file of Susi is operating system independent, so developers can include it into their own projects on Windows CE or Windows XP.
\WindowsCE\Library	• `Susi.lib` Library for developing the applications on Windows CE. • `Susi.dll` Dynamic library for Susi on Windows CE.
\WindowsCE\Demo	• `SusiDemo.EXE` Demo program on Windows CE. • `Susi.dll` Dynamic library for Susi on Windows CE.
\WindowsCE\Demo\SRC	Source code of the demo program on Windows CE.
\WindowsXP	• `Setup.EXE` An installer, include driver installation and library, for Susi on Windows XP. For a completeness of Setup.exe, please refer to the subsection **`Installation:Windows XP`**.

Installation

The Susi supports many different operating systems. Each subsection below describes how to install the Susi and related software in a specific operating system. Please refer to the subsection matching your operating system.

Windows XP

In windows XP, you can install library, drivers and demo programs into the platform easily by using the installation tool, i.e. **Susi Library Installer**. After the installer is executed, the Susi Library and related files for Windows XP can be found in the target directory where you chose when installation. The files are listed in the following table.

Directory	Contents
\Library	• `Susi.lib` Library for developing the applications on Windows XP. • `Susi.dll` Dynamic library for Susi on Windows XP.
\Demo	• `SusiDemo.EXE` Demo program on Windows XP. • `Susi.dll` Dynamic library for Susi on Windows XP.
\Demo\SRC	Source code of the demo program on Windows XP.

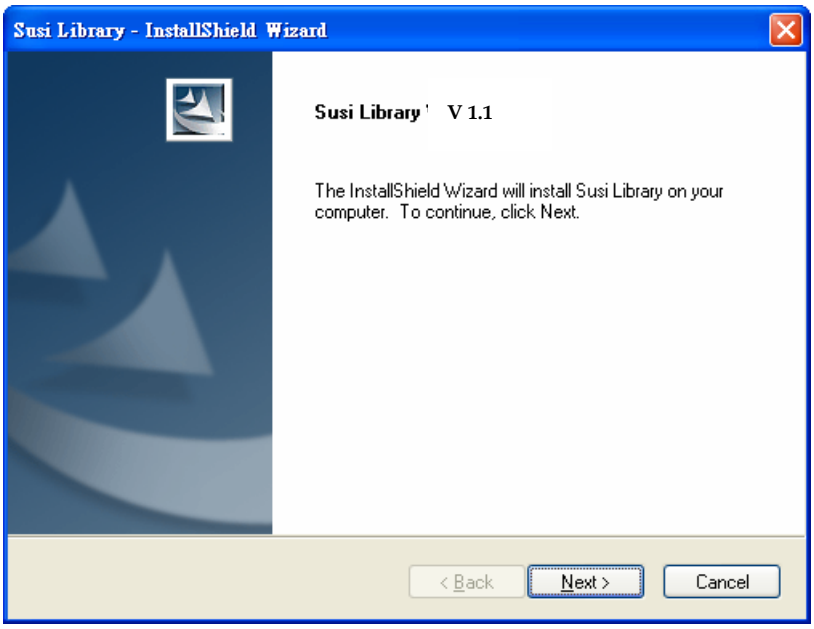
The following section of **[Installation]** illustrates all the process of installation.

***Note: The version of Susi Library Installer shows on each screen shot below should depend on your own version.**

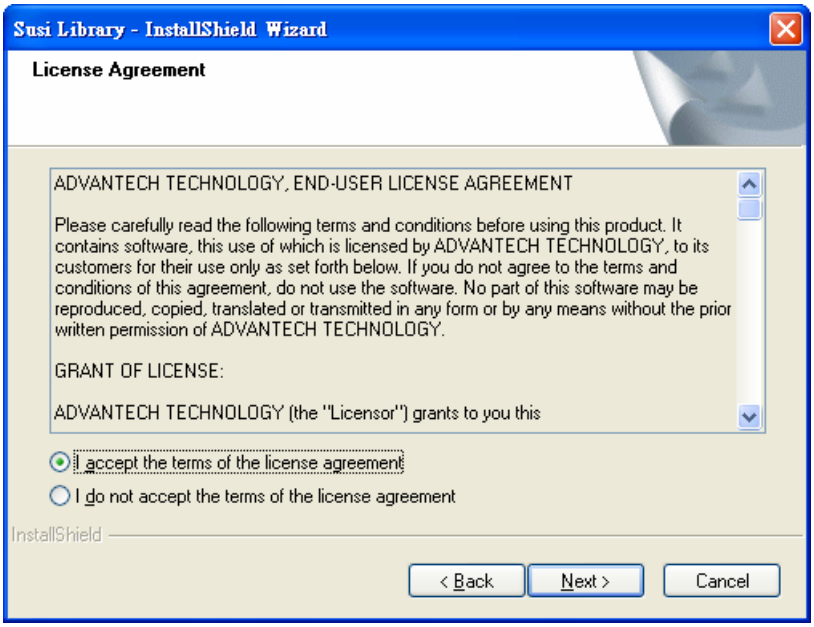
[Installation]

1. Extract **Susi.zip**.
2. Double-click the "**Setup.exe**" file.

3. The installer searches for a previous installation of **Susi Library**. If it locates one, a screen shot opens asking whether you want to **modify**, **repair** or **remove** the software. If a previous version is located, please see the section of **[Maintenance Setup]**. If it is not located, the following screen shot opens. Click **Next**.

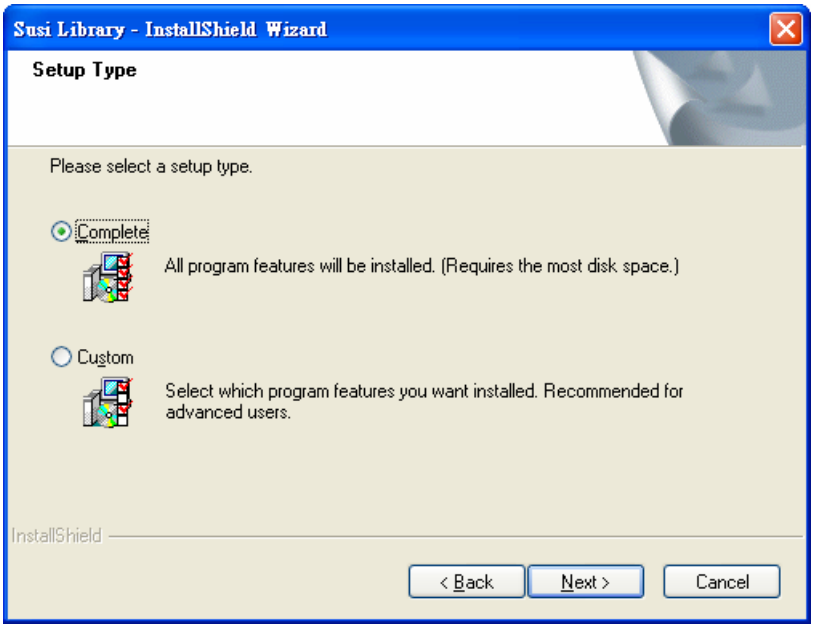


4. The **License Agreement** screen shot as the following displays.

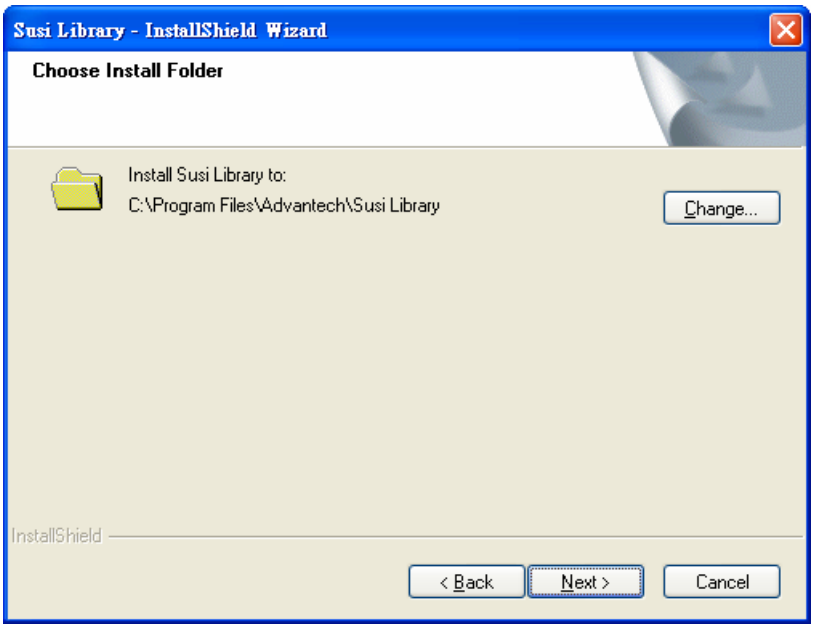


Read the license agreement carefully. If you accept the terms of the agreement, click 『**I accept the terms of the License Agreement**』. If you do not accept the terms, click 『**I do not accept the terms of the License Agreement**』. Click **Next**.

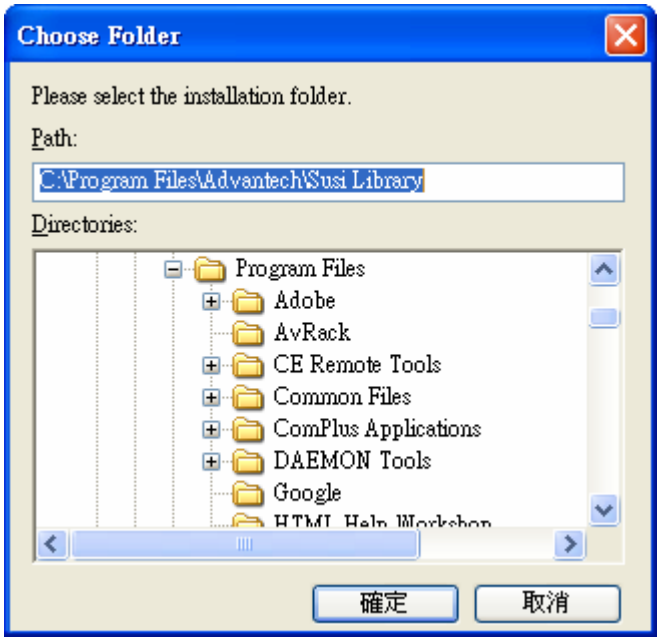
5. The **Setup Type** screen shot opens. If you choose **Complete** type, all features of Susi Library are selected and installed into the target platform. If you choose **Custom** type, you can select required features of Susi Library to install into your platform yourself. Click **Next**.



6. The **Choose Install Folder** screen shot opens. It indicates the path of the folder where Susi Library will be installed. If you want to **install** Susi Library in another location, click **Change**. If you do not want to change the default installation folder, please click **Next** directly.

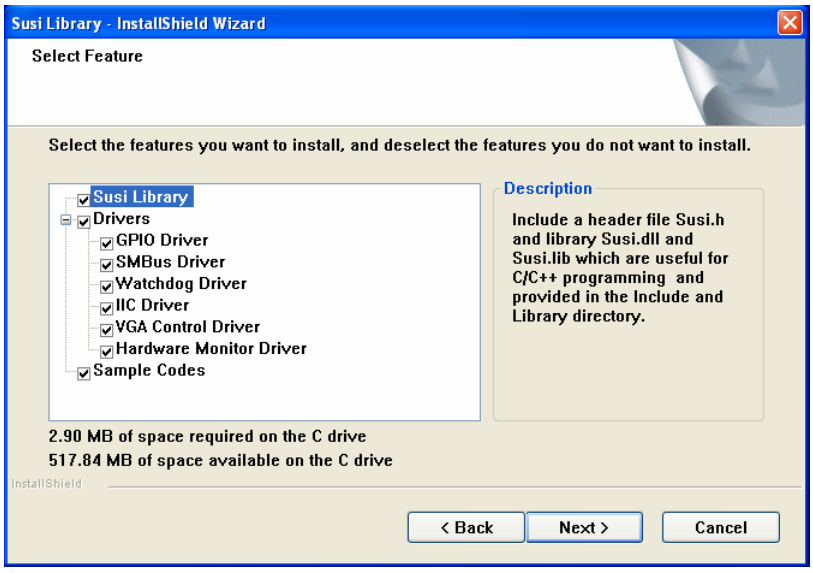


7. If you decide to change the installation folder above, the **Choose Folder** screen shot shows.

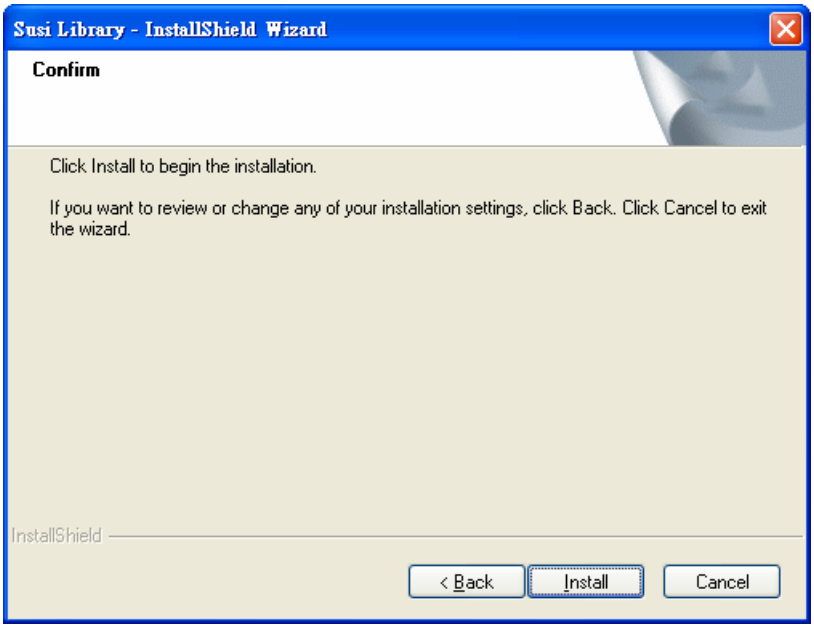


You can choose or type the path of desired installation folder and then click **OK** to return to the **Choose Install Folder** screen shot. After you have verified the location where Susi Library will be installed, click **Next**.

8. If you choose **Custom** Type on **Setup Type** screen shot, the **Select Feature** screen shot shows. Select the features you want to install. Click **Next**.

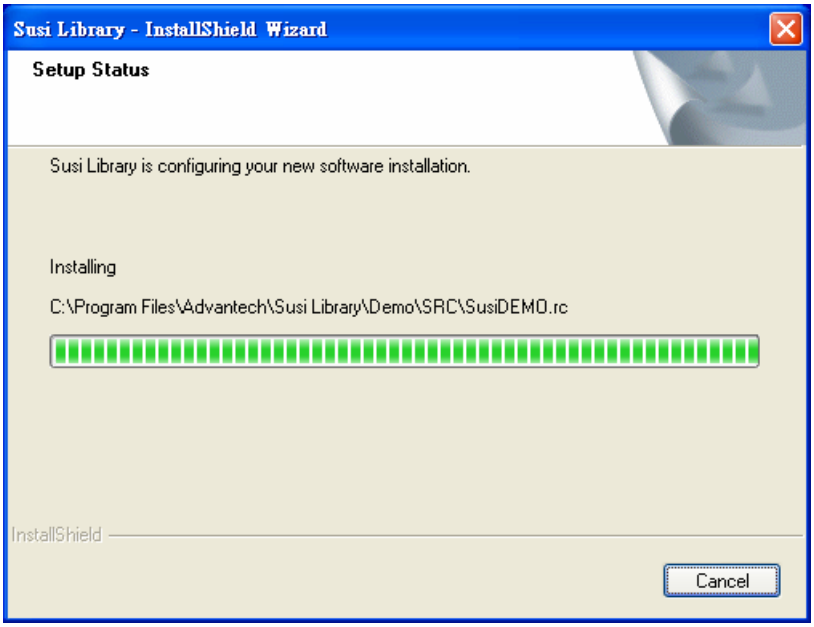


9. The **Confirm** screen shot opens.



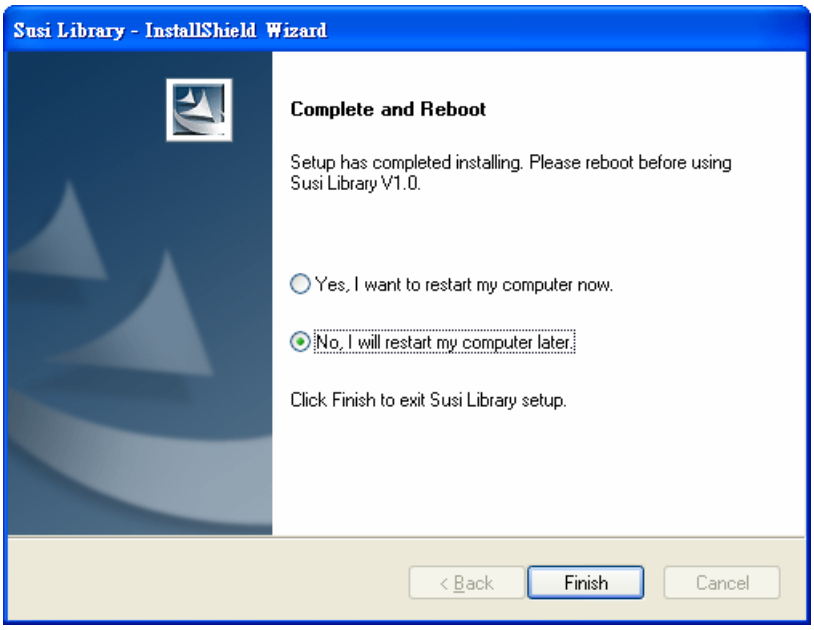
If you want to review or change any of your installation setting, please click **Back**. If all setting is correct, click **Install**.

10. The **Setup Status** screen shot opens.

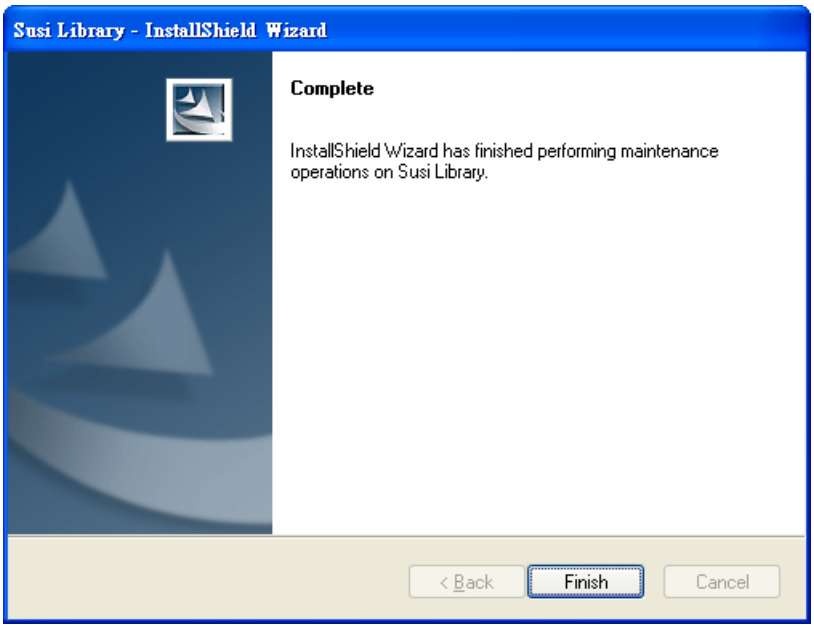


The installer starts to install all features of Susi Library you selected. When the installation is being processed, the Installing status bar progresses from empty to full. If you want to terminate the installation, click **Cancel**.

11. If you have selected **driver** features, the **Complete and Reboot** screen shot opens when all features are installed. Because some drivers are able to be active after rebooting, you can choose 『**Yes, I want to restart my computer**』 now to reboot right away. Otherwise, you can reboot later by choose 『**No, I will restart my computer later**』. Click **Finish** to reboot or exit the installer.

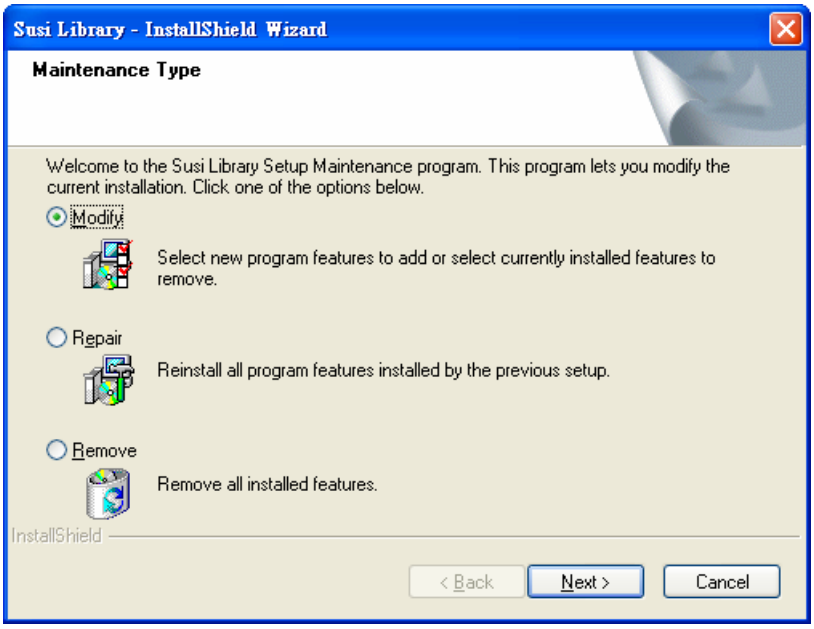


12. If you do not select any **driver** feature, the **Complete** screen shot opens when all features are installed. Click **Finish** to exit the installer.



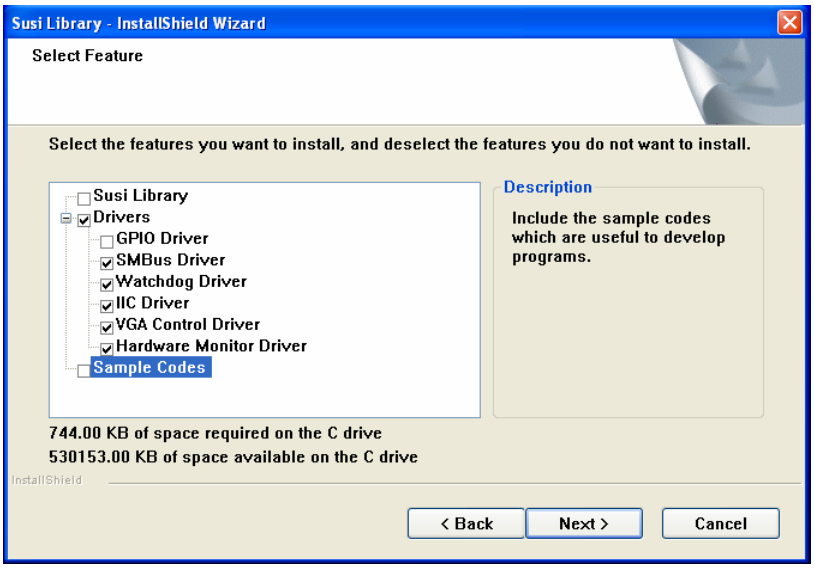
[Maintenance Setup]

1. Extract **Susi.zip**.
2. Open the 『**SusiInstaller**』 folder and then double-click the "**Setup.exe**" file.
3. The installer searches for a previous installation of **Susi Library**. If a previous version is not located, please see the section of **[Installation]**. If it is located, the **Maintenance Type** screen shot opens.



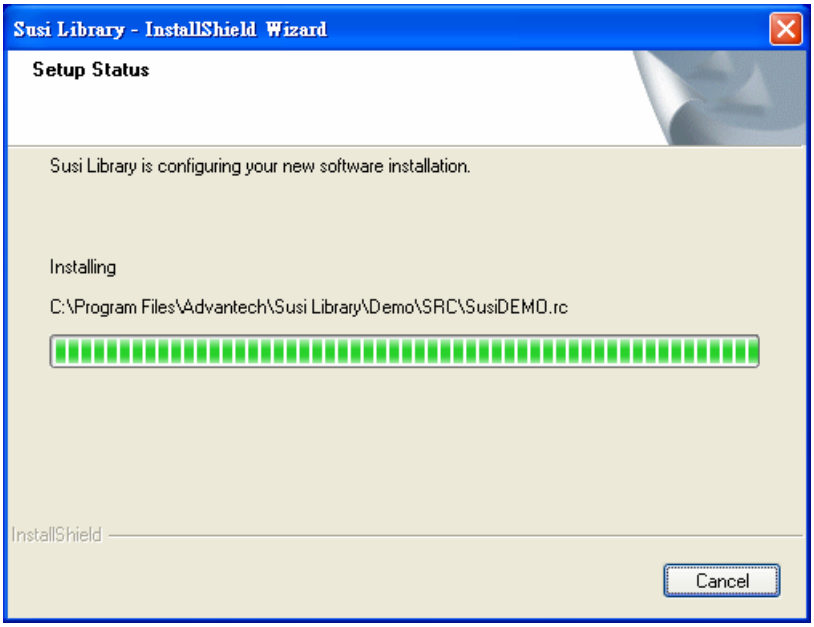
It asking whether you want to **modify**, **repair** or **remove** the software. Please choose one and then click **Next**.

4. If you choose **Modify** on **Maintenance Type** screen shot, the **Select Feature** screen shot opens. It means the installed feature if the feature is marked.



If you want to **add some features**, click them to mark. On the contrary, if you want to **remove some features**, click them to disable marks. Click **Next**.

5. The **Setup Status** screen shot opens.



The installer starts to install or remove all features of Susi Library you selected. When the installation is being processed, the Installing status bar progresses from empty to full. If you want to terminate the installation, click **Cancel**.

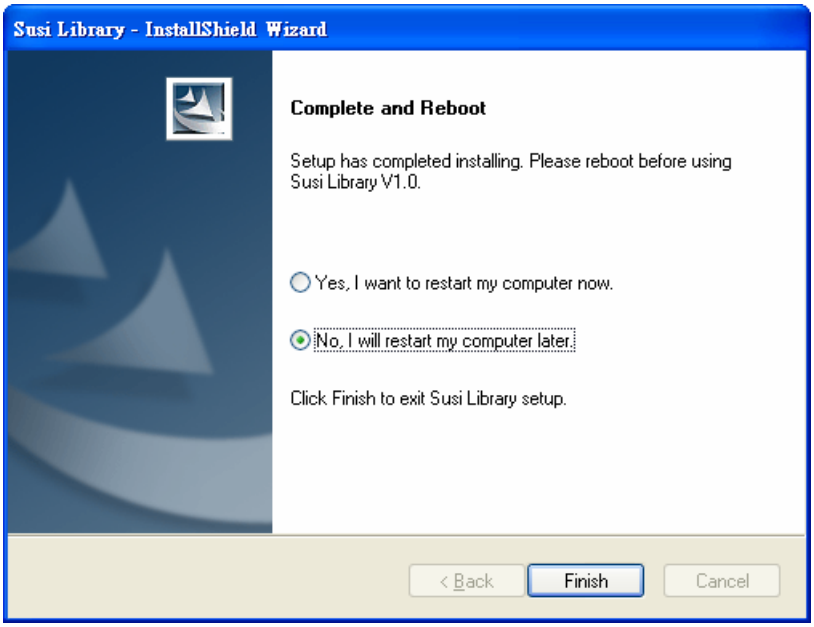
6. If you choose **Repair** on **Maintenance Type** screen shot, the **Setup Status** screen

shot opens later. It reinstall all installed again.

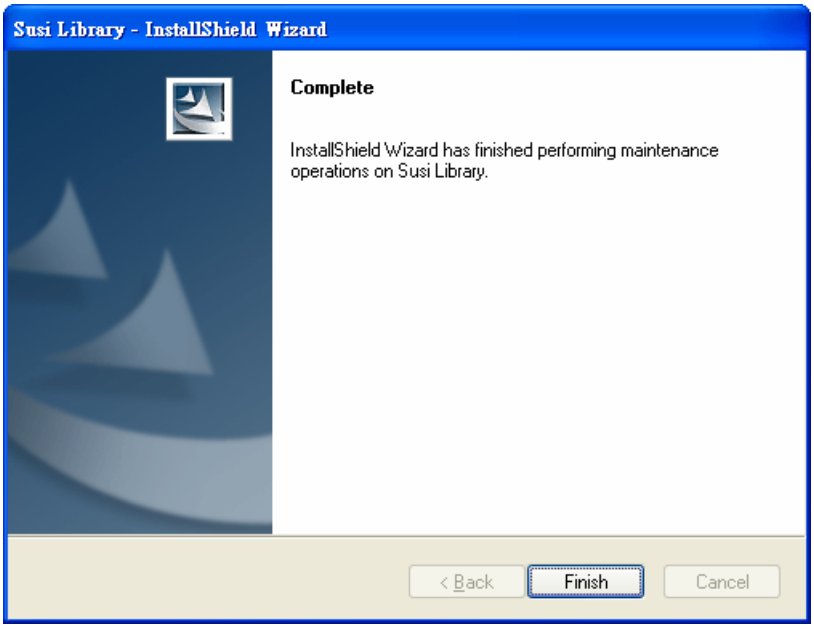
7. If you choose **Remove** on **Maintenance Type** screen shot, the dialog opens.

If you want to all installed features, click **Yes**. Otherwise, Click **No**.

8. If you have selected to install **driver** features on **Modify** or **Repair** type of **Maintenance Type**, the **Complete and Reboot** screen shot opens when all features are installed. Because some drivers are able to be active after rebooting, you can choose 『**Yes, I want to restart my computer**』 now to reboot right away. Otherwise, you can reboot later by choose 『**No, I will restart my computer later**』. Click **Finish** to reboot or exit the installer.



9. If you have not selected to install any **driver** feature, the **Complete** screen shot opens when all features are installed. Click **Finish** to exit the installer.



Windows CE

In windows CE, there are three ways to install Susi Library, you can install manually or use Advantech CE-Builder to install the library or just copy the programs and the library into the compact flash card.

Express Installation:

You can use Advantech CE-Builder to help you put the library into the image.

- First, you click the *My Component* tab.
- In this tab, you click *Add New Category* button to add a new category, ex Susi Library.
- Then you can add a new file in this category, and upload the Susi.dll for this category.
- After doing these steps, you can just select the *Susi Library* category you created for every project.

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 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.

Sample Programs

The sample programs demonstrate how to incorporate the Susi into your program. There are sample programs for two categories of operating system, i.e. Windows XP and Windows CE. The sample programs run in graphics mode in Windows XP and Windows CE. The sample programs are described in the subsections below.

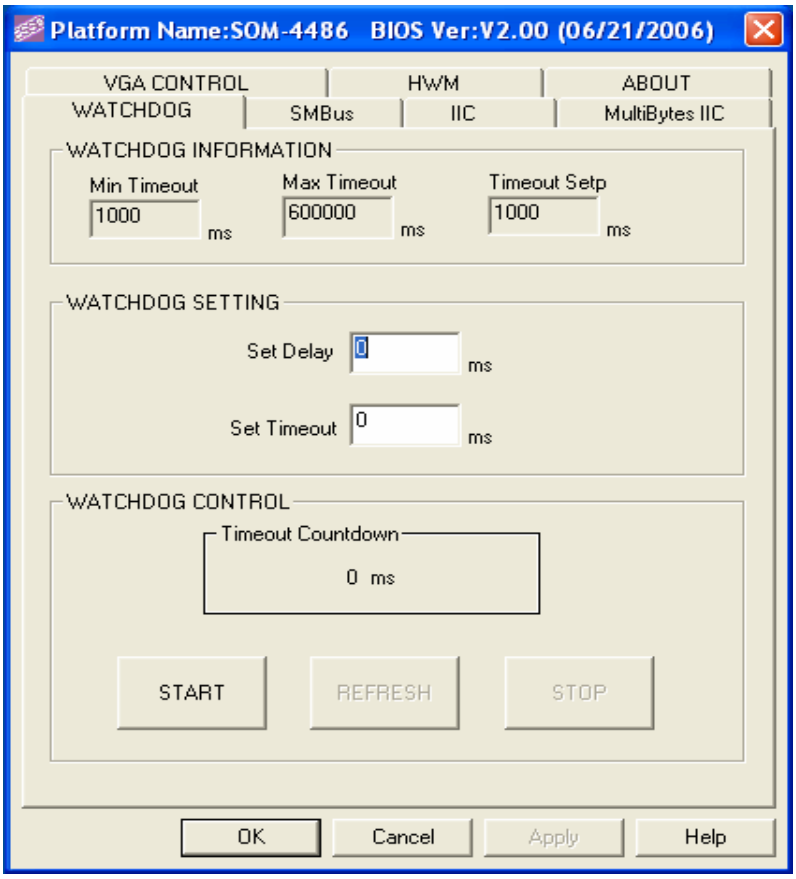
Windows Graphics Mode

There are sample programs of Windows graphics mode for two categories of operating system, i.e. Windows CE and Windows XP. Each demo application contains an executable file `SusiDemo.exe`, a shared library `Susi.dll` and `source codes` within the release package. The files of Windows CE and Windows XP are not compatible with each other.

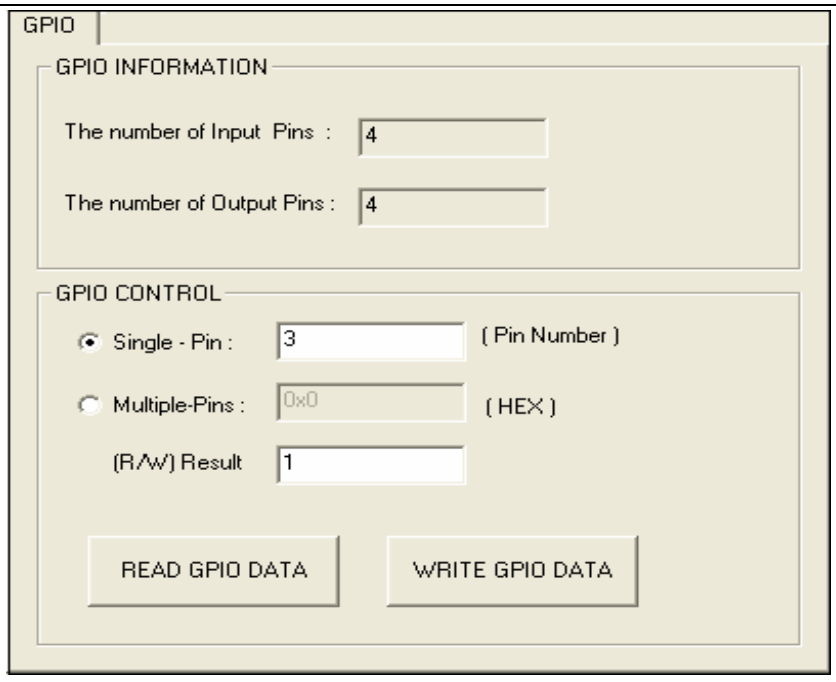
`SusiDemo.exe` is an executable file and it requires the shared library, `Susi.dll`, to demonstrate the Susi functions. The source codes of SusiDemo.exe also have two versions, i.e. Windows CE and Windows XP, and must be compiled under Microsoft Visual C++ 6.0 on Windows XP or under Microsoft Embedded Visual C++ 4.0 on Windows CE. Developers must add the header file `Susi.h` and library `Susi.lib` to their own projects when they want to develop something with Susi.

SusiDemo.exe

The `SusiDemo.exe` test application is an application which used all functions of the Susi Library. It has five major function blocks: `Watchdog`, `GPIO`, `SMBus`, `I2C` and `VGA control`. The following screen shot shows when you execute `SusiDemo.exe`. You can click function tabs to select test functions respectively. Some function tabs will not show on the test application if your platform does not support such functions. For a completeness of support list, please refer to `Appendix A`. We describe steps to test all functions of this application.


Platform Name:SOM-4486 BIOS Ver:V2.00 (06/21/2006)
VGA CONTROL
HWM
ABOUT
WATCHDOG
SMBus
IIC
MultiBytes IIC
WATCHDOG INFORMATION
Min Timeout
1000
ms
Max Timeout
600000
ms
Timeout Setp
1000
ms
WATCHDOG SETTING
Set Delay
0
ms
Set Timeout
0
ms
WATCHDOG CONTROL
Timeout Countdown
0 ms
START
REFRESH
STOP
OK
Cancel
Apply
Help

GPIO


GPIO
GPIO INFORMATION
The number of Input Pins :
4
The number of Output Pins :
4
GPIO CONTROL
Single - Pin :
3
(Pin Number)
Multiple-Pins :
0x0
(HEX)
(R/W) Result
1
READ GPIO DATA
WRITE GPIO DATA

When the application is executed, it will display GPIO information in the *GPIO INFORMATION* group box. It displays *the number of input pins and output pins*. You can click radio button to choose to test either *single pin function* or *multiple pins function*. The GPIO pins assignments of the supported platforms are located in Appendix B.

- **Test Read Single Input Pin**
  - Click the radio button of *Single-Pin*.
  - Key in the pin number to read the value of the input pin. Pin number is started from '0'.
  - Click *READ GPIO DATA* button and then the status of GPIO pin will be displayed in *(R/W)Result* field.

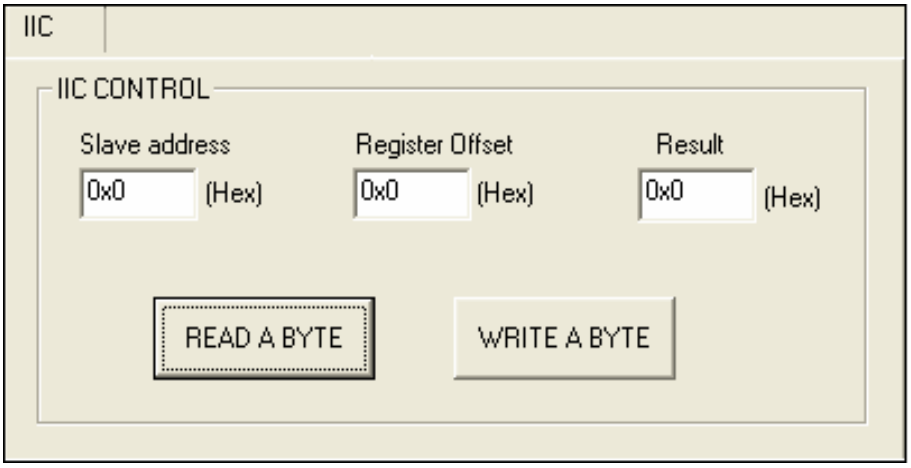
- **Test Read Multiple Input Pin**
  - Click the radio button of *Multiple-Pins.*
  - Key in the pin number from ‘0x01’ to ‘0x0F’ to read the value of the input pin. The pin numbers are bitwise-ored, i.e. bit 0 stands for GPIO 0, bit 1 stands for GPIO 1, etc. For example, if you want to read pin 0, 1, and 3, the pin numbers should be ‘0x0B’.
  - Click *READ GPIO DATA* button and then the statuses of GPIO pins will be displayed in *(R/W)Result* field.
- **Test Write Single Output Pin**
  - Click the radio button of *Single-Pin.*
  - Key in the pin number you want to write. Pin number is started from '0'.
  - Key in the value either '0' or '1' in *(R/W)Result* field to write the output pin you chose above step.
  - Click the *WRITE GPIO DATA* button to write the GPIO output pin.

- **Test Write Multiple Output Pins**
  - Click the radio button of *Multiple-Pins.*
  - Key in the pin number from ‘0x01’ to ‘0x0F’ to choose the multiple pin numbers to write the value of the output pin. The pin numbers are bitwise-ored, i.e. bit 0 stands for GPIO 0, bit 1 stands for GPIO 1, etc. For example, if you want to write pin 0, 1, and 3, the pin numbers should be ‘0x0B’.
  - Key in the value in *(R/W)Result* field from ‘0x01’ to ‘0x0F’ to write the value of the output pin. The pin numbers are bitwise-ored, i.e. bit 0

stands for GPIO 0, bit 1 stands for GPIO 1, etc. For example, if you want to set pin 0 and 1 high, 3 to low, the pin number should be '0x0B', and then you should key in the value '0x0A' to write.

- Click the *WRITE GPIO DATA* button to write the GPIO output pins.

I²C

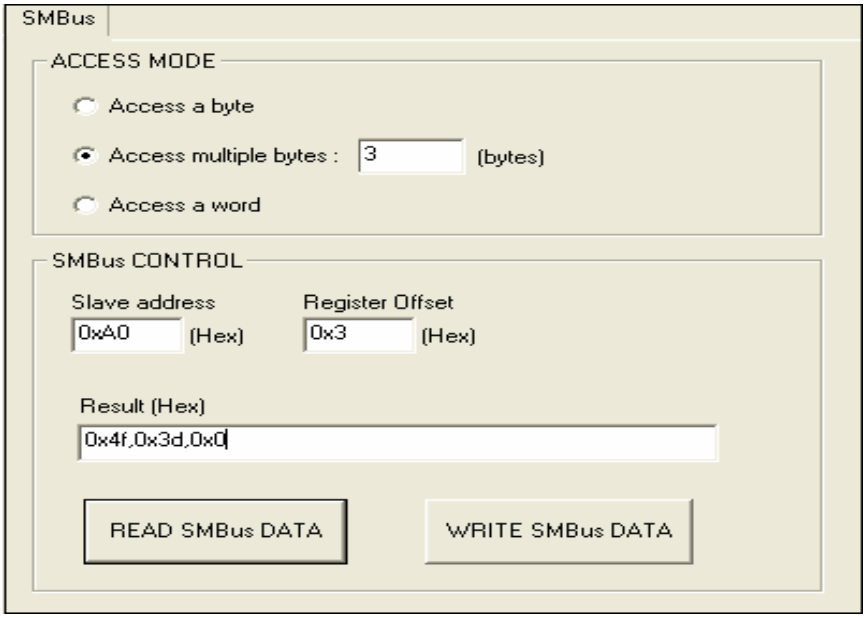


When the application is executed, you can read or write a byte of data through I²C devices. All data must be read or written in *hexadecimal system*.

- **Read a byte**
  - Key in the slave device address in *Slave address* field.
  - Key in the register offset in *Register Offset* field.
  - Click the *READ A BYTE* button and then a byte of data from the device will be shown on the *Result* field.

- **Write a byte**
  - Key in the slave device address in *Slave address* field.
  - Key in the register offset in *Register Offset* field.
  - Key in the desirous of data in *Result* field to write to the device.
  - Click the *WRITE A BYTE* button and then the data will be written to the device through I²C.

SMBus



When the application is executed, you can click radio button to choose to test each access mode, i.e. *Access a byte*, *Access multiple bytes and Access a word*. All data must be read or written in *hexadecimal system* except the number of bytes of *Access multiple bytes* mode must be written in *decimal system*. You can test the functionalities of watchdog as follows:

- **Read a byte**
  - Click the radio button of *Access a byte.*
  - Key in the slave device address in *Slave address* field.
  - Key in the register offset in *Register Offset* field.
  - Click the *READ SMBus DATA* button and then a byte of data from the device will be shown on the *Result* field.

- **Write a byte**
  - Click the radio button of *Access a byte.*
  - Key in the slave device address in *Slave address* field.
  - Key in the register offset in *Register Offset* field.
  - Key in the desirous of data in *Result* field to write to the device.

- Click the *WRITE SMBus DATA* button and then the data will be written to the device through SMBus.

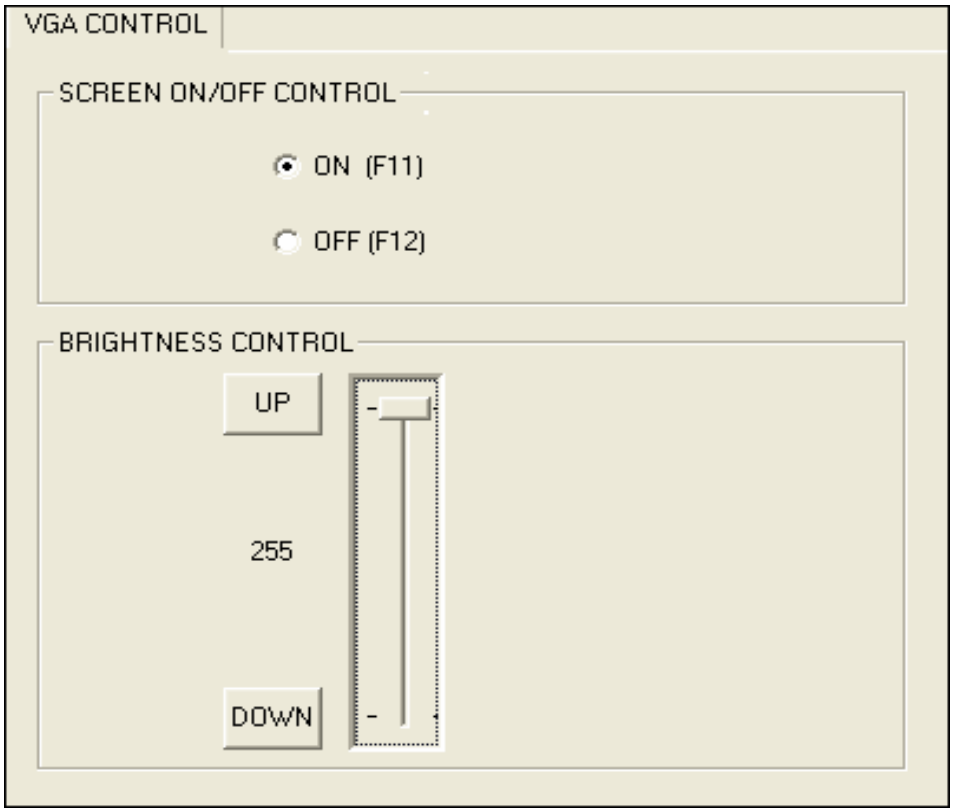
- **Read a word**
  - Click the radio button of *Access a word.*
  - Key in the slave device address in *Slave address* field.
  - Key in the register offset in *Register Offset* field.
  - Click the *READ SMBus DATA* button and then a word of data from the device will be shown on the *Result* field.

- **Write a word**
  - Click the radio button of *Access a word.*
  - Key in the slave device address in *Slave address* field.
  - Key in the register offset in *Register Offset* field.
  - Key in the desirous of data, such as 0x1234, in *Result* field to write to the device.
  - Click the *WRITE SMBus DATA* button and then the data will be written to the device through SMBus.

- **Read Multiple bytes**
  - Click the radio button of *Access multiple bytes.*
  - Key in the slave device address in *Slave address* field.
  - Key in the register offset in *Register Offset* field.
  - Key in the desirous of the number of bytes, such as 3, in the right side field of the radio button of *Access multiple bytes*. The number must be written in decimal system.
  - Click the *READ SMBus DATA* button and then all data from the device will be divided from each other by commas and be shown on the *Result* field.
- **Write Multiple bytes**
  - Click the radio button of *Access multiple bytes.*
  - Key in the slave device address in *Slave address* field.
  - Key in the register offset in *Register Offset* field.
  - Key in the desirous of the number of bytes, such as 3, in the right side field of the radio button of *Access multiple bytes*. The number must be written in decimal system.

- Key in all of the desirous of data in *Result* field in hexadecimal format and divided by commas, for example, *0x50,0x60,0x7A*.
- Click the *WRITE SMBus DATA* button and then all of the data will be written to the device through SMBus.

VGA Control

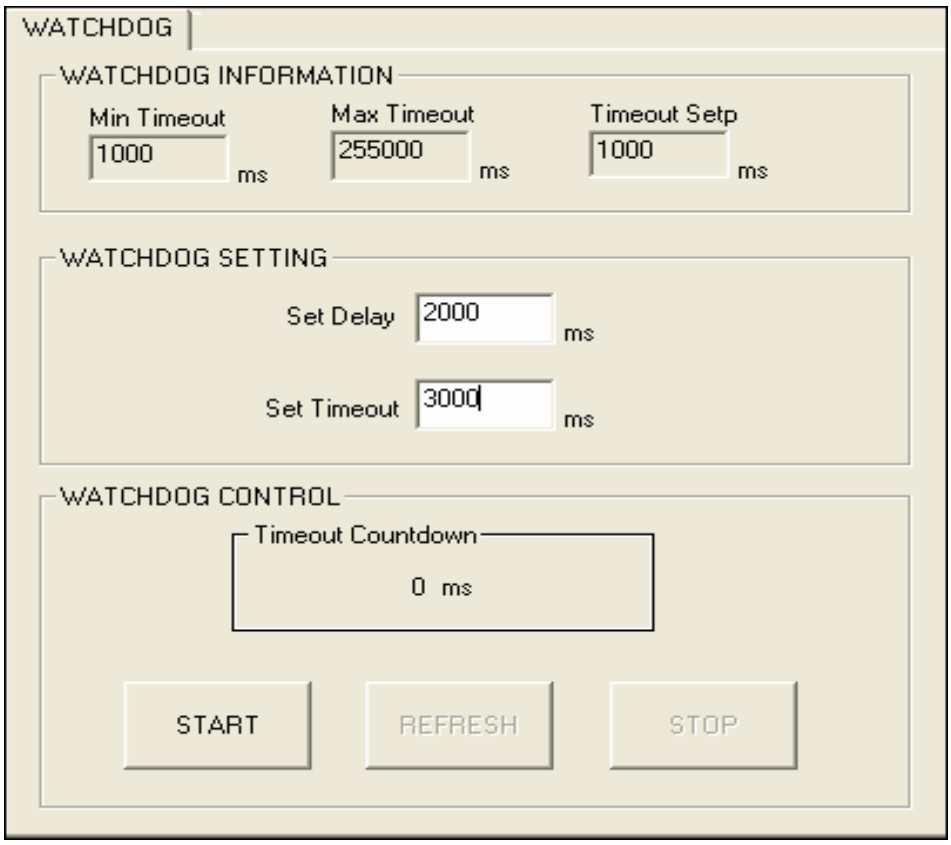


When the application is executed, it will display two blocks of VGA control functions. The application can turn on or turn off the screen shot freely, and it also can tune the brightness of the panels if your platform is being supported. You can test the functionalities of VGA control as follows:

- **Screen on/off control**
  - Click the radio button of *ON* or push the key *F11* can turn on the panel screen.
  - Click the radio button of *OFF* or push the key *F12* can turn off the panel screen.
  - The display chip of your platform must be in the support list in Appendix A, or this function can not work.

- **Brightness control**
  - Move the slider in *increments*, using either the mouse or the direction keys, or click the *UP button* to increase the brightness.
  - Move the slider in decrements, using either the mouse or the direction keys, or click the DOWN button to decrease the brightness.

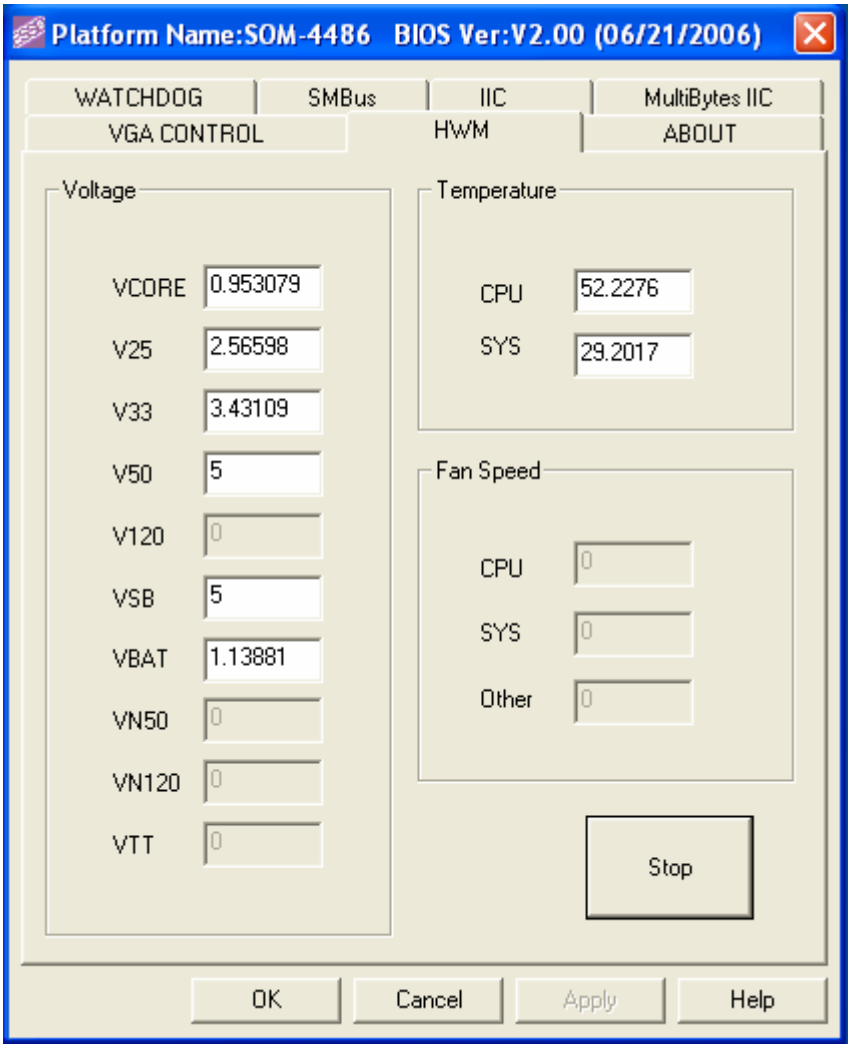
Watchdog



When the application is executed, it will display watchdog information in the *WATCHDOG INFORMATION* group box. It displays max timeout, min timeout, and timeout step in milliseconds. For example, a 1~255 seconds watchdog will has 255000 max timeout, 1000 min timeout, and 1000 timeout step. You can test the functionalities of watchdog as follows:

- Set the timeout value 3000 (3 sec.) in *SET TIMEOUT* field and set delay value 2000 (2 sec.) in *SET DELAY* field, then click the *START* button. The *Timeout Countdown* field will countdown the watchdog timer and displays 5000 (5 sec.).
- Before the timer countdown to zero, you can reset the timer by click the *REFRESH* button. After you click this button, the *Timeout Countdown* field will display the value of *SET TIMEOUT* field.
- If you want to stop the watchdog timer, you just click the *STOP* button.

HWM



When the application is executed by clicking button *Monitor*, it will display hardware monitoring data values. If certain data value is not supported in the platform, the correspondent data field will show as color gray with value 0.

Susi Library Reference

SusiInit

Initialize the Susi Library.

```
BOOL SusiInit(void);
```

Parameters

No parameter.

Return Value

TRUE (nonzero) on success.

FALSE (zero) on failure.

Remarks

Application must call `SusiInit()` before calling other functions. The function will return FALSE if the driver on Windows XP is not properly installed or not working.

SusiUnInit

Uninitialize the Susi Library.

```
BOOL SusiUnInit(void);
```

Parameters

No parameter.

Return Value

`TRUE` (nonzero) on success.

`FALSE` (zero) on failure.

Remarks

Application must call `SusiUnInit()` before it terminate. After calling `SusiUnInit()`, application cannot call other functions anymore except `SusiGetVersion()`. Calls to `SusiInit()` and `SusiUnInit()` should be paired.

SusiGetVersion

Returns the version number of Susi Library.

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

Parameters

`major`

`[out]` Point to a variable in which this function returns the major version number of Susi.

`minor`

`[out]` Point to a variable in which this function returns the minor version number of Susi.

Return Value

No return value.

Remarks

This function returns the version number of Susi. It's suggested to call this function first and compare major number with the constant `SUSI_VER_MJ` in header file.

SusiGetBIOSVersion

Get the version of the current BIOS.

```
int SusiGetBIOSVersion(TCHAR *BIOSVersion, BYTE size);
```

Parameters

BIOSVersion

[out] Point to a variable in which this function returns the version of the BIOS.

size

[in] Specifies the size in bytes allocated for the first parameter *BIOSVersion*.

Return Value

value	Meaning
< 0	The function fails
0	The function succeeds
> 0	If the parameter, *size*, is less than the real length of the BIOS version, it will return the correct value.

Remarks

The platform name can not be got correctly if:

1. The BIOS is not the released version.
2. The driver SusiCore.sys is not properly installed or not working on Windows XP.

SusiGetPlatformName

Get the name of the current platform.

```
int SusiGetPlatformName(TCHAR *Platformname, BYTE size);
```

Parameters

PlatformName

[out] Point to a variable in which this function returns the name of the platform.

size

[in] Specifies the size in bytes allocated for the first parameter *PlatformName*.

Return Value

value	Meaning
< 0	The function fails
0	The function succeeds
> 0	If the parameter, *size*, is less than the real length of the platform name, it will return the correct value.

Remarks

The platform name can not be got correctly if:

1. The BIOS is not the released version.
2. The driver `SusiCore.sys` is not properly installed or not working on Windows XP.

SusiIICAvailable

Query whether I^2C driver is available.

```
BOOL SusiIICAvailable(void);
```

Parameters

No parameter.

Return Value

`TRUE` (nonzero) if I^2C driver is present.

`FALSE` (zero) otherwise.

Remarks

Query whether I^2C driver is available in the platform. Application is suggested to call `SusiIICAvailable()` to make sure I^2C driver is present before calling I^2C functions.

SusiIICReadByte

Read a byte of data from the target slave device, which exists on I^2C.

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

Parameters

Slaveaddress

[in] Specifies the device address, in hexadecimal format, demanded to be read.

Registeroffset

[in] Specifies the offset of the address register belongs to the target slave device, which should be represented as the hexadecimal format.

Result

[out] Get a byte of data from the target slave device.

Return Value

TRUE (nonzero) on success.

FALSE (zero) on failure.

Remarks

Application should specify valid parameters, SlaveAddress and RegisterOffset, of the target slave device on I^2C to read. If the specified value is invalid, the return value is FALSE.

SusiIICWriteByte

Write a byte of data to the target slave device exists on I^2C.

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

Parameters

SlaveAddress

[in] Specifies the device address, in hexadecimal format, demanded to be written.

RegisterOffset

[in] Specifies the offset of the address register belongs to the target slave device, which should be represented as the hexadecimal format.

Result

[in] Write a byte of data to the target slave device, which should be represented as the hexadecimal format.

Return Value

TRUE (nonzero) on success.

FALSE (zero) on failure.

Remarks

Application should specify valid parameters, SlaveAddress and RegisterOffset, of the target slave device on I^2C to write. If the specified value is invalid, the return value is FALSE.

WARNING: Use this function with great care to prevent PERMANENT DAMAGE TO YOUR SYSTEM.

SusiIICWriteReadCombine

Write bytes of data to the target slave device exists on I^2C, and then read bytes from it.

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

Parameters

SlaveAddress

[in] Specifies the device address, in hexadecimal format, demanded to be written.

WriteBuf

[in] Write bytes of data from the target slave device.

WriteLen

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

ReadBuf

[out] Get bytes of data from the target slave device.

ReadLen

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

Return Value

TRUE (nonzero) on success.

FALSE (zero) on failure.

Remarks

Application should specify valid parameters, SlaveAddress, WriteBuf, WriteLen, ReadBuf and ReadLen of the target slave device on I^2C to read. If the specified value is invalid, the return value is FALSE.

WARNING: Use this function with great care to prevent PERMANENT DAMAGE TO YOUR SYSTEM.

SusiIICReadByteMulti

Read bytes of data from the target slave device exists on I^2C.

```
BOOL SusiIICReadByteMulti(BYTE SlaveAddress,
BYTE *ReadBuf, DWORD ReadLen);
```

Parameters

SlaveAddress

[in] Specifies the device address, in hexadecimal format, demanded to be written.

ReadBuf

[out] Get bytes of data from the target slave device.

ReadLen

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

Return Value

TRUE (nonzero) on success.

FALSE (zero) on failure.

Remarks

Application should specify valid parameters, SlaveAddress, ReadBuf, and ReadLen of the target slave device on I^2C to read. If the specified value is invalid, the return value is FALSE.

SusiIICWriteByteMulti

Write bytes of data to the target slave device exists on I^2C.

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

Parameters

SlaveAddress

[in] Specifies the device address, in hexadecimal format, demanded to be written.

WriteBuf

[in] Write bytes of data from the target slave device.

WriteLen

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

Return Value

TRUE (nonzero) on success.

FALSE (zero) on failure.

Remarks

Application should specify valid parameters, SlaveAddress, WriteBuf, and WriteLen of the target slave device on I^2C to read. If the specified value is invalid, the return value is FALSE.

WARNING: Use this function with great care to prevent PERMANENT DAMAGE TO YOUR SYSTEM.

SusiIOAvailable

Query whether GPIO driver is available.

```
BOOL SusiIOAvailable(void);
```

Parameters

No parameter.

Return Value

TRUE (nonzero) on success.

FALSE (zero) otherwise.

Remarks

Query whether GPIO driver is available in the platform. Application is suggested to call `SusiIOAvailable()` to make sure GPIO driver is present before calling GPIO functions.

SusiIOCount

Query how many GPIO pins are supported.

```
BOOL SusiIOCount(WORD *inCount, WORD *outCount);
```

Parameters

inCount

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

outCount

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

Return Value

TRUE (nonzero) on success.

FALSE (zero) on failure.

Remarks

Application can call this function to get the number of input and output GPIO pins supported in this platform.

SusiIOInitial

Initialize all the output pins of GPIO.

```
BOOL SusiIOInitial(DWORD statuses);
```

Parameters

statuses

[in] Bitwise-ored status of assigned pins. Set related bit of assigned pin to 1 will set the pin active (high). Otherwise, set the pin inactive (low).

Return Value

TRUE (nonzero) on success.

FALSE (zero) on failure.

Remarks

Initialize all the output pins at first. The other GPIO related functions will return FALSE if the applications have not called `SusiIOInitial()`. The value of writing to the output pins must be confirmed no influence on the system.

The parameter statuses are bitwise-ored. Bit 0 stand for GPIO 0, bit 1 stand for GPIO 1, etc. For example, if there are 8 output pins in your platform, and you want to set pin 0, 1 and 3 high, rest to low, the statuses parameter should be 0x0000000B.

SusiIORead

Read current status of one GPIO pin.

```
BOOL SusiIORead(BYTE pin, BOOL *status);
```

Parameters

pin

[in] Specifies the GPIO pin demanded to be read. Begin from 0.

status

[out] If the pin is active (high), status is nonzero. If the pin is inactive (low), status is zero.

Return Value

TRUE (nonzero) on success.

FALSE (zero) on failure.

Remarks

Application should specify valid input pin number to read. If the specified pin is invalid, the return value is FALSE.

SusiIOReadMulti

Read current statuses of several GPIO pins.

```
BOOL SusiIOReadMulti(DWORD pins, DWORD *statuses);
```

Parameters

pins

[in] Specifies the GPIO pins demanded to be read. The pins to read are bitwise-ored. Pin number begins from 0.

statuses

[out] Bitwise-ored status of assigned pins. For pins that are not specified, the related bit value is useless. For valid assigned pins, if the pin is active(high), the bit status is 1, otherwise 0.

Return Value

TRUE (nonzero) on success.

FALSE (zero) on failure.

Remarks

Read multiple input pins at the same time. The parameter pins is bitwise-ored. Bit 0 stand for GPIO 0, bit 1 stand for GPIO 1, etc. For example, if you want to read pin 0, 1, and 5, the pins parameter should be 0x00000023.

SusiIOWrite

Set high/low value to one GPIO pin.

```
BOOL SusiIOWrite(BYTE pin, BOOL status);
```

Parameters

pin

[in] Specifies the GPIO pin demanded to be written. Begin from 0.

status

[in] Set status to TRUE will set the pin active (high). Otherwise, set the pin inactive (low).

Return Value

TRUE (nonzero) on success.

FALSE (zero) on failure.

Remarks

Application should specify valid input pin number to write. If the specified pin is invalid, the return value is FALSE.

SusiIOWriteMulti

Set several GPIO pins at the same time.

```
BOOL SusiIOWriteMulti(DWORD pins, DWORD statuses);
```

Parameters

pins

[in] Specifies the GPIO pins demanded to be written. The pins to write are bitwise-ored. Pin number begin from 0.

statuses

[in] Bitwise-ored status of assigned pins. Set related bit of assigned pin to 1 will set the pin active (high). Otherwise, set the pin inactive (low).

Return Value

TRUE (nonzero) on success.

FALSE (zero) on failure.

Remarks

Write multiple output pins at the same time. The parameter pins and statuses are bitwise-ored. Bit 0 stand for GPIO 0, bit 1 stand for GPIO 1, etc. For example, if you want to set pin 0 and 1 high, 5 to low, the pin parameter should be 0x00000023, and statuses parameter can be 0x00000003.

SusiSMBusAvailable

Query whether SMBus driver is available.

```
BOOL SusiSMBusAvailable(void);
```

Parameters

No parameter.

Return Value

`TRUE` (nonzero) on success.

`FALSE` (zero) otherwise.

Remarks

Query whether SMBus driver is available in the platform. Application is suggested to call `SusiSMBusAvailable()` to make sure SMBus driver is present before calling SMBus functions.

SusiSMBusReadByte

Read a byte of data from the target slave device, which exists on SMBus.

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

Parameters

SlaveAddress

[in] Specifies the device address, in hexadecimal format, demanded to be read.

RegisterOffset

[in] Specifies the offset of the address register belongs to the target slave device, which should be represented as the hexadecimal format.

Result

[out] Get a byte of data from the target slave device.

Return Value

TRUE (nonzero) on success.

FALSE (zero) on failure.

Remarks

Application should specify valid parameters, SlaveAddress and RegisterOffset, of the target slave device on SMBus to read. If the specified value is invalid, the return value is FALSE.

SusiSMBusReadByteMulti

Read multiple bytes of data once from the target slave device, which exists on SMBus.

```
BOOL SusiSMBusReadByteMulti(BYTE SlaveAddress,
BYTE RegisterOffset, BYTE *Result, BYTE ByteCount);
```

Parameters

`SlaveAddress`

`[in]` Specifies the device address, in hexadecimal format, demanded to be read.

`RegisterOffset`

`[in]` Specifies the beginning of the offset of the address register belongs to the target slave device, which should be represented as the hexadecimal format.

`Result`

`[out]` Get the data from the target slave device.

`ByteCount`

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

Return Value

`TRUE` (nonzero) on success.

`FALSE` (zero) on failure.

Remarks

Read multiple bytes of data from the offset `RegisterOffset` of the target slave device once. Application should specify valid parameters, `SlaveAddress` and `RegisterOffset`, of the target slave device on SMBus to read. If the specified value is invalid, the return value is `FALSE`.

SusiSMBusReadWord

Read a word of data from the target slave device, which exists on SMBus.

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

Parameters

SlaveAddress

[in] Specifies the device address, in hexadecimal format, demanded to be read.

RegisterOffset

[in] Specifies the offset of the address register belongs to the target slave device, which should be represented as the hexadecimal format.

Result

[out] Get a word of data from the target slave device.

Return Value

TRUE (nonzero) on success.

FALSE (zero) on failure.

Remarks

Application should specify valid parameters, SlaveAddress and RegisterOffset, of the target slave device on SMBus to read. If the specified value is invalid, the return value is FALSE.

This function does not support PCM-9375 and SOM-4455.

SusiSMBusWriteByte

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

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

Parameters

SlaveAddress

[in] Specifies the device address, in hexadecimal format, demanded to be written.

RegisterOffset

[in] Specifies the offset of the address register belongs to the target slave device, which should be represented as the hexadecimal format.

Result

[in] Write a byte of data to the target slave device, which should be represented as the hexadecimal format.

Return Value

TRUE (nonzero) on success.

FALSE (zero) on failure.

Remarks

Application should specify valid parameters, SlaveAddress and RegisterOffset, of the target slave device on SMBus to write. If the specified value is invalid, the return value is FALSE.

SusiSMBusWriteByteMulti

Write multiple bytes of data to the target slave device, which exists on SMBus.

```
BOOL SusiSMBusWriteByteMulti(BYTE SlaveAddress,
BYTE RegisterOffset, BYTE *Result, BYTE ByteCount);
```

Parameters

SlaveAddress

[in] Specifies the device address, in hexadecimal format, demanded to be written.

RegisterOffset

[in] Specifies the beginning of the offset of the address register belongs to the target slave device, which should be represented as the hexadecimal format.

Result

[in] Write the data to the target slave device.

ByteCount

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

Return Value

TRUE (nonzero) on success.

FALSE (zero) on failure.

Remarks

Write multiple bytes of data to the target slave device from the offset `RegisterOffset` once. Application should specify valid parameters, `SlaveAddress` and `RegisterOffset`, of the target slave device on SMBus to write. If the specified value is invalid, the return value is FALSE.

SusiSMBusWriteWord

Write a word of data to the target slave device exists on SMBus.

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

Parameters

SlaveAddress

[in] Specifies the device address, in hexadecimal format, demanded to be written.

RegisterOffset

[in] Specifies the offset of the address register belongs to the target slave device, which should be represented as the hexadecimal format.

Result

[in] Write a word of data to the target slave device, which should be represented as the hexadecimal format.

Return Value

TRUE (nonzero) on success.

FALSE (zero) on failure.

Remarks

Application should specify valid parameters, SlaveAddress and RegisterOffset, of the target slave device on SMBus to write. If the specified value is invalid, the return value is FALSE.

This function does not support PCM-9375 and SOM-4455.

SusiVCAvailable

Query whether VGA control driver is available.

```
int SusiVCAvailable(void);
```

Parameters

No parameter.

Return Value

If the function succeeds, the return value is zero.

If the function fails, the return value is negative error code. For a complete list of error codes, please refer to `Appendix E` or head file `Susi.h`.

Remarks

Query whether VGA control driver is available in the platform. Application is suggested to call `SusiVCAvailable()` to make sure VGA control driver is present before calling VGA control functions. There are two kinds of the controls for VGA control, that is, `backlight on/off control` and `brightness control`. This function will return success if one of VGA controls supports your platform. For a completeness of support platform list, please refer to `Appendix A`.

SusiVCGetBright

Get the present brightness of the CRT panel.

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

Parameters

brightness

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

Return Value

If the function succeeds, the return value is zero.

If the function fails, the return value is negative error code. For a complete list of error codes, please refer to Appendix E or head file Susi.h.

Remarks

This function only supports CRT panels.

SusiVCGetBrightRange

Get acceptable minimum, maximum timeout and stepping unit of the CRT panel brightness.

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

Parameters

minimum

[out] Point to a variable in which this function returns the acceptable minimum brightness.

maximum

[out] Point to a variable in which this function returns the acceptable maximum brightness.

stepping

[out] Point to a variable in which this function returns the acceptable stepping of brightness between *minimum* and *maximum*.

Return Value

If the function succeeds, the return value is zero.

If the function fails, the return value is negative error code. For a complete list of error codes, please refer to Appendix E or head file Susi.h.

Remarks

Application can call this function to know the hardware dependent parameters of brightness. For example, if minimum is 0, maximum is 255, and stepping is 5, it means the brightness can be 0, 5, 10, …, 255 steps.

The display chip of your platform must be in the support list in Appendix C: Backlight On/Off Support Display Chipsets or this function will return errors. For a completeness of support platform list, please refer to Appendix A.

SusiVCScreenOff

Turn off the backlight.

```
int SusiVCScreenOff(void);
```

Parameters

No parameter.

Return Value

If the function succeeds, the return value is zero.

If the function fails, the return value is negative error code. For a complete list of error codes, please refer to `Appendix E` or head file `Susi.h`.

Remarks

The display chip of your platform must be in the support list in `Appendix C: Backlight On/Off Support Display Chipsets` or this function will return errors. For a completeness of support platform list, please refer to `Appendix A`.

SusiVCScreenOn

Turn on the backlight.

```
int SusiVCScreenOn(void);
```

Parameters

No parameter.

Return Value

If the function succeeds, the return value is zero.

If the function fails, the return value is negative error code. For a complete list of error codes, please refer to `Appendix E` or head file `Susi.h`.

Remarks

The display chip of your platform must be in the support list in `Appendix C: Backlight On/Off Support Display Chipsets` or this function will return errors. For a completeness of support platform list, please refer to `Appendix A`.

SusiVCSetBright

Set current brightness of the CRT panel.

```
int SusiVCSetBright(BYTE brightness);
```

Parameters

brightness

[in] Specifies the value demanded to set the brightness.

Return Value

If the function succeeds, the return value is zero.

If the function fails, the return value is negative error code. For a complete list of error codes, please refer to Appendix E or head file Susi.h.

Remarks

This function only supports CRT panels. The parameter brightness cannot exceed the range of the brightness that was specified in the function SusiVCBrightGetRange.

SusiWDAvailable

Query whether watchdog is available.

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

Parameters

No parameter.

Return Value

`TRUE` (nonzero) if watchdog is present.

`FALSE` (zero) otherwise.

Remarks

Query whether watchdog is available in the platform. Application is suggested to call `SusiWDAvailable()` to make sure watchdog is supported before trigger it.

SusiWDDisable

Disable the watchdog.

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

Parameters

No parameter.

Return Value

TRUE (nonzero) on success.

FALSE (zero) on failure.

Remarks

If watchdog is not longer required, application can call `SusiWDDisable()` to disable the watchdog. The return value may be FALSE if the watchdog hardware can not be stopped.

SusiWDGetRange

Get acceptable minimum, maximum timeout and stepping unit of watchdog.

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

Parameters

minimum

[out] Point to a variable in which this function returns the acceptable minimum timeout in milliseconds.

maximum

[out] Point to a variable in which this function returns the acceptable maximum timeout in milliseconds.

stepping

[out] Point to a variable in which this function returns the acceptable stepping of timeout in milliseconds between *minimum* and *maximum*.

Return Value

TRUE (nonzero) on success.

FALSE (zero) on failure.

Remarks

Application can call this function to know the hardware dependent parameters of watchdog. For example, if minimum is 1000, maximum is 63000, and stepping is 1000, it means the watchdog timeout can be 1, 2, 3, …, 63 seconds.

SusiWDSetConfig

Set configuration to watchdog driver.

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

Parameters

delay

[in] Specifies a timer in milliseconds allocated for the delay before first timeout period.

timeout

[in] Specifies a timer in milliseconds allocated for the watchdog timeout period.

Return Value

TRUE (nonzero) on success.

FALSE (zero) on failure.

Remarks

This function will enable and activate the watchdog with given parameters. After watchdog activated, application should periodically call `SusiWDTrigger()` within specified timeout milliseconds. Note that acceptable timeout value is depend on hardware, call `SusiWDGetRange()` to get the minimum/maximum timeout and stepping unit.

SusiWDTrigger

Tell watchdog that application is still working.

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

Parameters

No parameters

Return Value

TRUE (nonzero) on success.

FALSE (zero) on failure.

Remarks

After watchdog is activated, application should call this function continuously to indicate that it is still working properly. If watchdog is activated, and application doesn't call `SusiWDTrigger()` after timeout milliseconds, system will reboot.

SusiHWMAvailable

Query whether hardware monitor is available..

```
int SusiHWMAvailable ();
```

Parameters

No parameter.

Return Value

If the function succeeds, the return value is zero.

If the function fails, the return value is negative error code. For a complete list of error codes, please refer to `Appendix E` or head file `Susi.h`.

Remarks

Query whether hardware monitor is available in the platform. Application is suggested to call `SusiHWMAvailable()` to make sure hardware monitoring driver is present before calling its functions.

SusiHWMGetFanSpeed

This function is capable of either getting target fan speed data from hardware monitoring sensor, or querying which fans are supported in the platform.

```
int SusiHWMGetFanSpeed(WORD HWMtype, WORD *retval, WORD*
typesupport = NULL);
```

Parameters

HWMtype

[in] Specifies a fan flag which to get speed from. For a define list of possible fan flags, please refer to Appendix D or head file Susi.h

retval

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

Typesupport

[in/out] If the value is specified as a pointer (non-NULL) to a variable, it will return the supported fans in flag bitwise-ORed

Return Value

If the function succeeds, the return value is zero.

If the function fails, the return value is negative error code. For a complete list of error codes, please refer to Appendix E or head file Susi.h.

Remarks

To successfully monitor possible fan speeds in certain platform, call the function with non-NULL *typesupport* variable to know which fans are supported. According to it, call the function and specify the fan flag one by one, to get each fan speed.

SusiHWMGetTemperature

This function is capable of either getting target temperature data from hardware monitoring sensors or querying which sensor subjects are supported in the platform.

```
int SusiHWMGetTemperature(WORD HWMtype, float *retval,
WORD* typesupport);
```

Parameters

HWMtype

[in] Specifies a subject flag which to get temperature from. For a define list of possible flags, please refer to Appendix D or head file Susi.h

retval

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

Typesupport

[in/out] If the value is specified as a pointer (non-NULL) to a variable, it will return the supported sensor subjects in flag bitwise-ORed

Return Value

If the function succeeds, the return value is zero.

If the function fails, the return value is negative error code. For a complete list of error codes, please refer to Appendix E or head file Susi.h.

Remarks

To successfully monitor temperature change of interest subjects in certain platform, call the function with non-NULL *typesupport* variable to know which sensor subjects are supported. According to it, call the function and specify the temperature flag one by one, to get each temperature value in **Celsius**.

SusiHWMGetVoltage

This function is capable of either getting target voltage data from hardware monitoring sensor, or querying which voltage targets are supported in the platform.

```
int SusiHWMGetVoltage(DWORD HWMtype, float *retval, DWORD* typesupport);
```

Parameters

HWMtype

[in] Specifies a flag which to get voltage from. For a define list of possible flags, please refer to Appendix D or head file Susi.h

retval

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

Typesupport

[in/out] If the value is specified as a pointer (non-NULL) to a variable, it will return the supported voltage targets in flag bitwise-ORed

Return Value

If the function succeeds, the return value is zero.

If the function fails, the return value is negative error code. For a complete list of error codes, please refer to Appendix E or head file Susi.h.

Remarks

To successfully monitor possible supported voltage targets in certain platform, call the function with non-NULL *typesupport* variable to know which of them are supported. According to it, call the function and specify the voltage flag one by one, to get each voltage in **Volt**.

Appendix A - Support Platform List

Windows XP

Platform	WatchDog	GPIO	SMBus	I²C	Brightness	Backlight On/Off	HWM
AIMB-330	○	○	NA	NA	NA	○	NA
AIMB-340	○	○	NA	NA	NA	NA	NA
AIMB-640	NA	○	NA	NA	NA	○	NA
PCA-6751	○	NA	NA	NA	NA	○	NA
PCA-6753	○	NA	NA	NA	NA	○	NA
PCA-6770	○	NA	NA	NA	NA	○	NA
PCA-6772	○	NA	NA	NA	NA	○	NA
PCA-6773	○	NA	NA	NA	NA	○	NA
PCA-6774	○	NA	NA	NA	NA	○	NA
PCI-6870	○	NA	NA	NA	NA	NA	NA
PCI-6871	○	NA	NA	NA	NA	○	NA
PCI-6872	○	NA	NA	NA	NA	○	NA
PCI-6880	○	○	NA	NA	NA	○	NA
PCI-6881	○	○	NA	NA	NA	○	NA
PCM-3350	○	○	NA	NA	NA	○	NA
PCM-3370	○	NA	NA	NA	NA	○	NA
PCM-3380	○	NA	NA	NA	NA	○	NA
PCM-3386	○	NA	NA	NA	NA	○	NA
PCM-5820	○	NA	NA	NA	NA	○	NA
PCM-5822	NA	NA	NA	NA	NA	○	NA
PCM-5823	○	NA	NA	NA	NA	○	NA
PCM-5824	○	NA	NA	NA	NA	NA	NA
PCM-5825	○	NA	NA	NA	NA	○	NA
PCM-9371	○	NA	NA	NA	NA	○	NA
PCM-9372	○	NA	NA	NA	NA	○	NA
PCM-9373	○	NA	NA	NA	NA	○	NA

PCM-9374	○	○	NA	NA	NA	○	NA
PCM-9375	○	○	○	NA	○	○	NA
PCM-9377	○	○	○	NA	NA	○	NA
PCM-9380	○	○	NA	NA	NA	○	NA
PCM-9381	○	○	NA	NA	NA	○	NA
PCM-9386	○	○	NA	NA	NA	○	NA
PCM-9387	○	○	NA	NA	NA	○	NA
PCM-9572	○	NA	NA	NA	NA	○	NA
PCM-9575	○	NA	NA	NA	NA	○	NA
PCM-9577	○	○	NA	NA	NA	○	NA
PCM-9578	○	○	NA	NA	NA	NA	NA
PCM-9579	○	NA	NA	NA	NA	○	NA
PCM-9580	○	○	NA	NA	NA	NA	NA
PCM-9581	○	○	NA	NA	NA	○	NA
PCM-9582	○	○	NA	NA	NA	○	NA
PCM-9586	○	○	NA	NA	NA	○	NA
PCM-9587	○	○	NA	NA	NA	○	NA
POS-564	○	○	○	NA	NA	○ *3	NA
POS-761	NA	○	○	NA	NA	○	NA
SOM-4450	○	NA	NA	NA	NA	○	NA
SOM-4455	○	NA	○	○	NA	○	NA
SOM-4470	○	NA	○	NA	NA	○	NA
SOM-4472	○	NA	○	○ *2	NA	○	○
SOM-4475	○	NA	○	○ *2	NA	○	○
SOM-4481	○	NA	○	○	NA	○	○
SOM-4486	○	NA	○	○	NA	○	○
SOM-5780	○	NA	○	○	○	○	NA

Note 1. Please reference Appendix B. for more information of GPIO Pins Assignment

Note 2. This function only supports newer versions of SOM-4472 (A3) and SOM-4475 (A2).

Note 3. This function only supports POS-564 with SMI 720 chipset.

Windows CE

Platform	Watchdog	GPIO	SMBus	I²C	Brightness	Backlight On/Off	HWM
PCA-6751	○	NA	NA	NA	NA	○	NA
PCA-6772	○	NA	NA	NA	NA	○	NA
PCA-6773	○	NA	NA	NA	NA	○	NA
PCM-3341	○	NA	NA	NA	NA	NA	NA
PCM-3347	○	NA	NA	NA	NA	○	NA
PCM-3348	○	NA	NA	NA	NA	NA	NA
PCM-3350	○	○	NA	NA	NA	○	NA
PCM-3370	○	NA	NA	NA	NA	○	NA
PCM-3380	○	NA	NA	NA	NA	○	NA
PCM-3386	○	NA	NA	NA	NA	○	NA
PCM-5820	○	NA	NA	NA	NA	○	NA
PCM-5825	○	NA	NA	NA	NA	○	NA
PCM-9371	○	NA	NA	NA	NA	○	NA
PCM-9372	○	NA	NA	NA	NA	○	NA
PCM-9373	○	NA	NA	NA	NA	○	NA
PCM-9374	○	○	NA	NA	NA	○	NA
PCM-9375	○	○	○	NA	○	○	NA
PCM-9377	○	○	○	NA	NA	○	NA
PCM-9380	○	○	NA	NA	NA	○	NA
PCM-9381	○	○	NA	NA	NA	○	NA
PCM-9386	○	○	NA	NA	NA	○	NA
PCM-9387	○	○	NA	NA	NA	○	NA
PCM-9575	○	NA	NA	NA	NA	○	NA
PCM-9577	○	○	NA	NA	NA	○	NA
PCM-9579	○	NA	NA	NA	NA	○	NA
PCM-9581	○	○	NA	NA	NA	○	NA

PCM-9586	○	○	NA	NA	NA	○	NA
SOM-4450	○	NA	NA	NA	NA	○	NA
SOM-4455	○	NA	○	○	NA	○	NA
SOM-4470	○	NA	○	NA	NA	○	NA
SOM-4472	○	NA	○	○ *2	NA	○	○
SOM-4475	○	NA	○	○ *2	NA	○	○
SOM-4481	○	NA	○	○	NA	○	○
SOM-4486	○	NA	○	○	NA	○	○
SOM-5780	○	NA	○	○	○	○	NA

Note 1. Please reference Appendix B. for more information of GPIO Pins Assignment

Note 2. This function only supports newer versions of SOM-4472 (A3) and SOM-4475 (A2).

Appendix B - GPIO Pins Assignment

AIMB-330/ AIMB-340/ AIMB-640

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

The number of GPIO pins：0 Inputs, 2 outputs

Pin of CN20	Signal	Pin of CN19	Signal
Pin-1	+5V	Pin-1	+5V
Pin-2	OUT0	Pin-2	OUT1

PCM-9374/ PCM-9375/ PCM-9377/PCM-9380/ PCM-9386/ PCM-9577

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/ PCM-9387

The number of GPIO pins：4 Inputs

Pin	Signal
Pin-1	VCC
Pin-3	IN0
Pin-5	IN1
Pin-7	IN2
Pin-9	IN3

PCM-9580

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/ PCM-9582/ PCM-9586/ PCM-9587

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

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

PCI-6881

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	+5V	Pin-8	NC
Pin-9	IN2	Pin-10	OUT2
Pin-11	GND	Pin-12	GND
Pin-13	IN3	Pin-14	OUT3

POS-564/ POS-761

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****
Pin-5	IN2	Pin-6	GND
Pin-7	IN3	Pin-8	***OUT1****
Pin-9	GND	Pin-10	+12V
Pin-11	NC	Pin-12	NC
Pin-13	OUT3	Pin-14	GND
Pin-15	OUT2	Pin-16	+12V

*. 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 C - Backlight On/Off Support

Backlight On/Off Support Display Chipsets

- AMD CX5530
- Asiliant (C&T) 6900
- Intel 855GME chip
- SMI Lynx 721 chip
- VIA CLE266 chipset
- VIA VT8606 Twister chip

Appendix D – Hardware Monitoring Flags

Fan Speed

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

```
// Fan Speed
#define FCPU   (1<<0)   // CPU FAN Speed
#define FSYS   (1<<1)   // System FAN Speed
#define F2ND   (1<<2)   // Other FAN Speed
```

Temperature

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

```
// Temperature
#define TCPU (1<<0)     // CPU Temperature
#define TSYS (1<<1)     // 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

```
//Voltage
#define VCORE    (1<<0)     // Vcore
#define V25      (1<<1)     // 2.5V
#define V33      (1<<2)     // 3.3V
#define V50      (1<<3)     // 5V
#define V120     (1<<4)     // 12V
#define VSB      (1<<5)     // Voltage of standby
#define VBAT     (1<<6)     // VBAT
#define VN50     (1<<7)     // -5V
#define VN120    (1<<8)     // -12V
#define VTT      (1<<9)     // VTT
```

Appendix E - API Error Codes

The following table provides a list of Susi error codes.

Code	Name	Description
-10	ERR_SUSI_LIB_INIT_FAIL	Susi cannot be initialized.
-11	ERR_SUSI_DRIVER_CONTROL_FAIL	The driver on Windows XP does not work normally.
-50	ERR_SUSI_BRIGHTNESS_CONTROL_FAIL	The brightness inside control fails.
-51	ERR_BRIGHT_OUT_OF_RANGE	The specified brightness is out of range.