Trusted ePlatform Services

ADVANTECH

SUSI® Library

Software API

Version 3.02

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

This document is copyrighted, 20011, by Advantech Co., Ltd. All rights reserved. Advantech Co., Ltd. Reserves the right to make improvements to the products described in this manual at any time. Specifications are thus subject to change without notice.

No part of this manual may be reproduced, copied, translated, or transmitted in any form or by any means without prior written permission of Advantech Co., Ltd. Information provided in this manual is intended to be accurate and reliable. However, Advantech Co., Ltd., assumes no responsibility for its use, or for any infringements upon the rights of third parties which may result from its use.

All the trade marks of products and companies mentioned in this data sheet belong to their respective owners.

Copyright © 1983-20011 Advantech Co., Ltd. All Rights Reserved

Part No.
Version: 3.02

Printed in Taiwan 2011-10-31~~2011-08-29~~

Version History

Date	Version	Part no	Remark
2006-7-27	1.0		New release
2006-9-29	1.1		Added hardware monitoring support for SOM-4472/SOM-4475/SOM-4481/SOM-4486
2007-6-27	1.2		Added 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		Added Embedded BIOS interface Added 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		Added Embedded BIOS interface for Linux Added SUSI Manager for central control of SUSI Added utilities for Monitoring, PowerSaving, HotKey manager, Brightness Control, Security ID, ePlatformFlash
2009-05-01	3.02		Added API for PowerSaving,, Added New Platform of AIMB, ESBC and COM
2011-04-07	3.02		Remove VGA Control Hotkey description Modified UI figures to SUSIDemo V2 Modified OS support list Added Power Saving program tab page Modified definition of HWM Modified throttling API, removed 4 old API and added 9 new APIs. Fixed some API remarks.
2011-07-29	3.02		Modified SUSI AP icons Added new APIs description Removed Appendix A – GPIO was to old

Table of Contents

INTRODUCTION **7**
SUSI Functions 7
Benefits 10
ENVIRONMENTS **10**
PACKAGE CONTENTS **12**
ADDITIONAL PROGRAMS **13**
DEMO PROGRAM 13
SusiDemo.exe 13
i. Boot Logger 14
ii. Watchdog 15
iii. GPIO 16
iv. Programmable GPIO 18
v. SMBus 20
vi. Multibyte IIC 21
vii. VGA Control 22
viii. Hardware Monitor 23
ix. Power Saving 24
x. About 25
PROGRAMMING OVERVIEW **26**
Driver independent functions 27
Core functions 27
Watchdog (WD) functions 27
GPIO (IO) functions 28
SMBus functions 28
IIC functions 29
VGA Control (VC) functions 29
Hardware Monitoring (HWM) functions 30
SUSI API PROGRAMMER'S DOCUMENTATION **31**
SusiDllInit 31
SusiDllUnInit 32
SusiDllGetVersion 33
SusiDllGetLastError 34
SusiCoreAvailable 35
SusiCoreGetBIOSVersion 36
SusiCoreGetPlatformName 37
SusiCoreAccessBootCounter 38
SusiCoreAccessRunTimer 39
SSCORE_RUNTIMER 40
SUSIPlusCpuSetThrottling 41
SUSIPlusCpuGetThrottling 42
SUSIPlusCpuSetOnDemandThrottling 43
SUSIPlusCpuGetOnDemandThrottling 44
SUSIPlusSpeedIsActive 45
SUSIPlusSpeedSetActive 46
SUSIPlusSpeedSetInactive 47
SUSIPlusSpeedWrite 48

SUSIPlusSpeedRead 49
SusiCoreGetMaxCpuSpeed....50
SusiCoreGetCpuVendor....51
SusiWDAvailable 52
SusiWDGetRange....53
SusiWDSetConfig 54
SusiWDSetConfigEx 55
SusiWDTrigger....56
SusiWDTriggerEx 57
SusiWDDisable 58
SusiWDDisableEx 59
SusiIOAvailable....60
SusiIOCountEx....61
SusiIOQueryMask 62
SusiIOSetDirection....63
SusiIOSetDirectionMulti 64
SusiIOReadEx 65
SusiIOReadMultiEx....66
SusiIOWriteEx....67
SusiIOWriteMultiEx 68
SusiSMBusAvailable 69
SusiSMBusScanDevice....70
SusiSMBusReadQuick....71
SusiSMBusWriteQuick....72
SusiSMBusReceiveByte 73
SusiSMBusSendByte....74
SusiSMBusReadByte....75
SusiSMBusWriteByte 76
SusiSMBusReadWord....77
SusiSMBusWriteWord....78
SusiSMBusReadBlock....79
SusiSMBusWriteBlock 80
SusiSMBusI2CReadBlock 81
SusiSMBusI2CWriteBlock 82
SusiIICAvailable....83
SusiIICRead 84
SusiIICWrite....85
SusiIICWriteReadCombine....86
SusiVCAvailable 87
SusiVCGetBrightRange....88
SusiVCGetBright....89
SusiVCSetBright 90
SusiVCScreenOn....91
SusiVCScreenOff 92
SusiHWMAvailable....93
SusiHWMGetFanSpeed....94
SusiHWMGetTemperature....95
SusiHWMGetVoltage 96

SusiHWMSetFanSpeed....97
APPENDIX A – PROGRAMMING FLAGS OVERVIEW 98
APPENDIX B - API ERROR CODES....101
FUNCTION INDEX CODE 101
LIBRARY ERROR CODE 104
DRIVER ERROR CODE 106

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 Software APIs (Application Programming Interfaces), called **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 Functions

Control

- **GPIO**

(icon-1)

General Purpose Input/Output is a flexible parallel interface that allows a variety of custom connections. It supports various Digital I/O devices – input devices like buttons, switches; output devices such as cash drawers, LED lights…etc. And, allows users to monitor the level of signal input or set the output status to switch on/off the device. Our API also provide Programmable GPIO, allows developers to dynamically set the GPIO input or output status

- **SMBus**

(icon-2)

SMBus is the System Management Bus defined by Intel® Corporation in 1995. It is used in personal computers and servers for low-speed system management communications. Today, SMBus is used in all types of embedded systems.
The SMBus API allows a developer to interface a Windows XP or CE PC to a downstream embedded system environment and transfer serial messages using the SMBus protocols, allowing multiple simultaneous device control.

- **I²C**

(icon-2)

I²C is a bi-directional two wire bus that was developed by Philips for use in their televisions in the 1980s. Today, I²C is used in all types of embedded systems.
The I²C API allows a developer to interface a Windows XP or CE PC to a downstream embedded system environment and transfer serial messages using the I²C protocols, allowing multiple simultaneous device control.

Monitor

- **Watchdog**

(icon-3)

A watchdog timer (WDT) is a device or electronic card that performs a specific operation after a certain period of time if something goes wrong with an electronic system and the system does not recover on its own.
A watchdog timer can be programmed to perform a warm boot (restarting the system) after a certain number of seconds during which a program or computer fails to respond following the most recent mouse click or keyboard action.

- **Hardware Monitor**

(icon-4)

The Hardware Monitor (HWM) API is a system health supervision API that inspects certain condition indexes, such as fan speed, temperature and voltage.

- **Hardware Control**

(icon-5)

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

Display

- **Brightness Control**

(icon-6)

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

- **Backlight**

(icon-7)

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

Power Saving

- **CPU Speed**

(icon-8)

Makes use of Intel SpeedStep technology to save the power consumption (Windows XP only). The system will automatically adjust the CPU Speed depending on the system loading.

- **System Throttling**

(icon-9)

Refers to a series of methods for reducing power consumption in computers by lowering the clock frequency. These API allow a user to lower the clock from 87.5% to 12.5%.

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 XP Embedded
- Windows XP Pro or Home Edition 32-bit
- Windows 7 (x86 and x64)
- WES7 (x86 and x64)
- Linux (Project based, request from your local FAE)
- QNX (Project based, request from your local FAE)
- VxWorks (Project based, request from your local FAE)

Note that the list may be changed without notice. For the latest support list, please check: *http://www.advantech.com.tw/embcore/software_apis.aspx*
For any Questions feel free to contact your local Advantech representative.

Package Contents

SUSI currently supports Windows XP and Windows 7. Contents listed below:

<table>
<tr><th>Operating System</th><th>Location</th><th>Installation</th></tr>
<tr><td>Windows XP(e),
Windows 7, WES7</td><td>C:\ProgramFiles\Advantech\SUSIV30</td><td>Setup.exe</td></tr>
<tr><th>Directory</th><th colspan="2">Contents</th></tr>
<tr><td>User Manual</td><td colspan="2">SUSI.pdf</td></tr>
<tr><td>Library Files</td><td colspan="2">• Susi.lib
Function export
• Susi.dll
Dynamic link library</td></tr>
<tr><td>Include Files</td><td colspan="2">• REL_Susi.h
• REL_Debug.h / REL_Errdrv.h / REL_Errlib.h</td></tr>
<tr><td>SusiDemo</td><td colspan="2">• SusiDemo.exe
Demo program execution file
• Susi.dll
Dynamic link library</td></tr>
<tr><td>Driver Installation</td><td colspan="2">• *.sys and *.inf
Driver files
• devcon.exe
For Remove.bat
• SUSInst.exe
For Install.bat
• Install.bat
Batch for install drivers
• Remove.bat
Batch for remove drivers</td></tr>
<tr><td>SusiDemo\Source Code\</td><td colspan="2">Source code of SusiDemo program in C#, VS2005</td></tr>
<tr><td>%System32%</td><td colspan="2">• Susi.dll
Dynamic link library</td></tr>
</table>

Additional Programs

Demo Program

The SUSI demo program demonstrates how to incorporate SUSI library into user's own applications. The program is written in C# programming language and based upon .NET Compact Framework 2.0, Visual Studio 2005. If you plan to write your own application you can refer to the source code of the Demo program. If you want to write a application for Windows 7 x64 but use our SUSI standard you need to set your application to 'Platform Target = x86' at build options. If you have received a custom x64 SUSI version this is not necessary. Ask your local FAE if you are not sure about this.

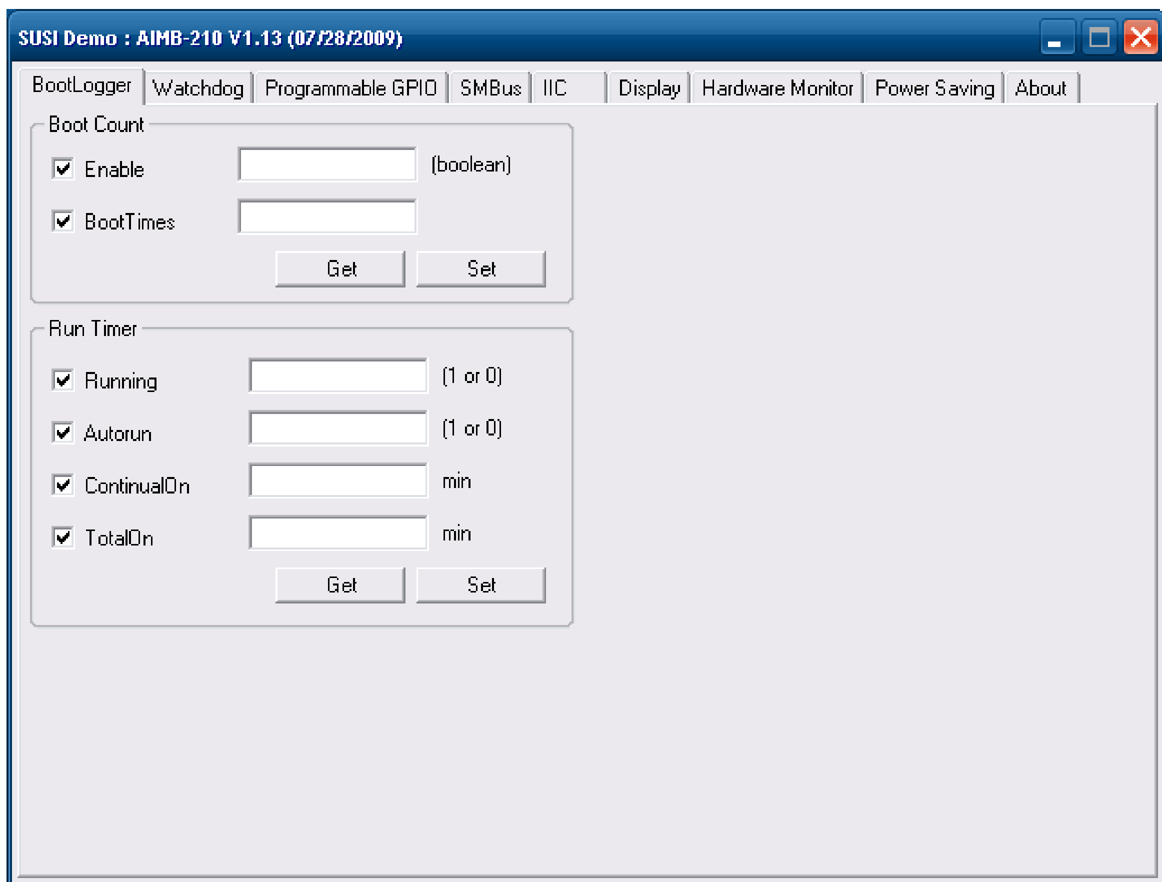
SusiDemo.exe

The execution file, **SusiDemo.exe**, released with source code can be run on both Windows XP and Windows CE. It is written to demonstrate how to access all the functions provided by Advantech SUSI. It also allows you a first test after installing if the functions you want to use are working. Advantech SusiDemo.exe is made for demonstration and testing. Engineers can use it for evaluation too. Keep in mind: SusiDemo.exe is not made as a Consumer product and it's not made for production.

The following pages are a detailed introduction to the SusiDemo.exe program. It will explain how to use all the functions with Advantech SusiDemo.exe program.

Note: The following sections explain all possible settings for SUSI. Depending on your Hardware you may have not have all these options available.

i. Boot Logger



(Figure-1)

This part belongs to the feature Core in SUSI APIs.

- Select or clear the check box to select the information to get or set in its text box.

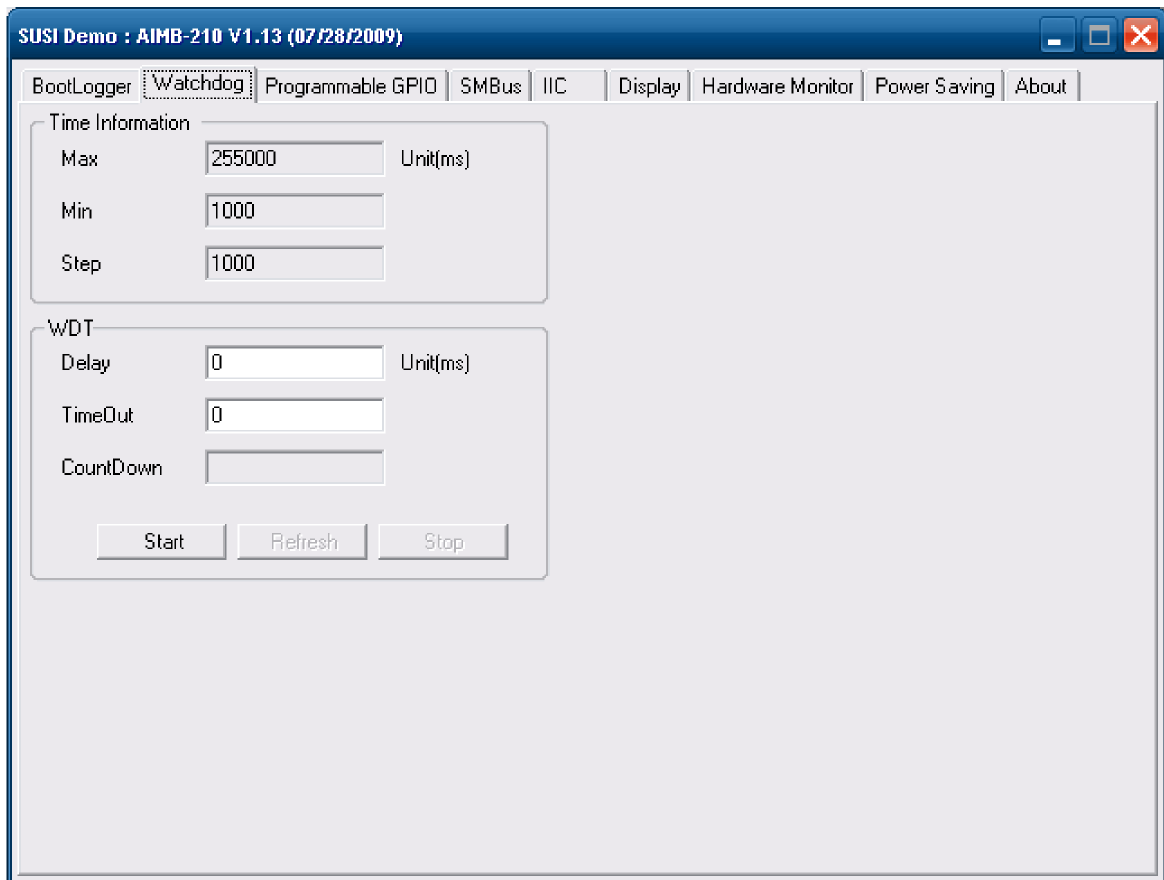
In Boot Counter

- To enable the Bootcounter write '**true**' and click set
  To disable the Bootcounter write '**false**' and click set
- To reset the BootTimes parameter to 0, just type 0 in the **BootTimes** text box with its check box selected, and then click the "Set" button.

In Run Timer

- Set the **Running** text box to 1 to start the timer, or 0 to stop the timer.
- Set the **Autorun** text box to 1 to start the timer when the system restarts.

ii. Watchdog



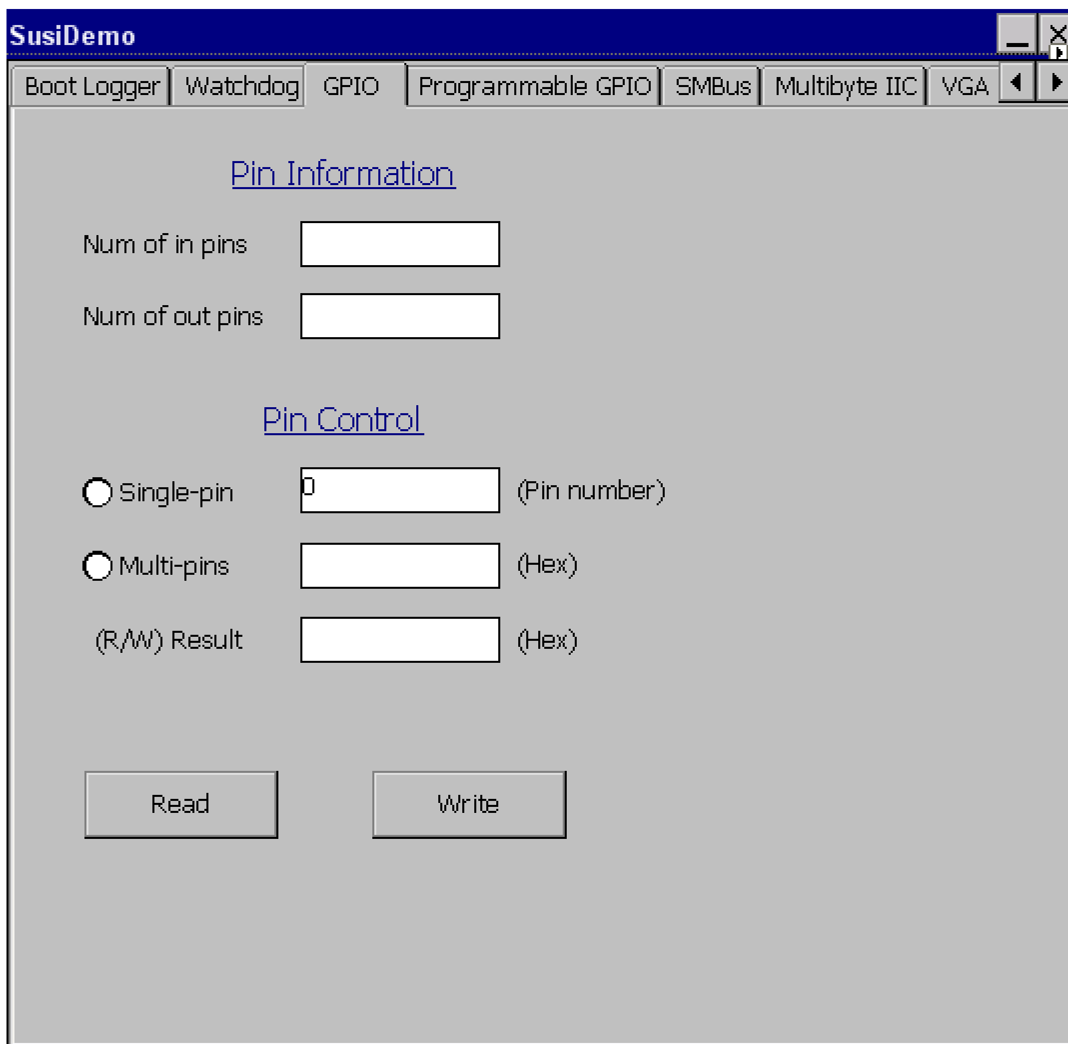
(Figure-2)

When the SusiDemo program executes, it shows watchdog information in the "Timeout Information" fields - "Min", "Max", and "Step" in milliseconds. For example, for a range of 1 ~ 255 seconds, 1000 appears in the "Min" text box, 255000 appears in the "Max" text box, and 1000 appears in the "Step" text box.

Here is an example of how to use the watchdog timer:

- Type 3000 (3 sec.) in the "Timeout" text box and optionally type 2000 (2 sec.) in the "Delay" text box. Click the "Start" button. The "Left" text box will show the approximate countdown value the watchdog timer. (This is a software timer in the demo program, not the actual watchdog hardware timer so it is not very accurate.)
- Before the timer counts down to zero, you may reset the timer by clicking the "Refresh" button, stop it by clicking the "Stop" button.

iii. GPIO



(Figure-3)

This page is only for backward compatibility with previous APIs that are bidirectional. In new GPIO supported platforms, this page will not be shown. **We highly recommend you use the new Programmable GPIO.**

When the SusiDemo program executes, it displays the fixed numbers of input pins and output pins in “Pin Information” field. You can click the “Single-pin” or “Multi-pins” radio button to choose single or multiple pins. For GPIO pinout information for each platform, please refer to the Appendix .

Read Single Input Pin

- Click “Single-Pin” radio button.
- Type the input pin number to read the status from. Pins are numbered from 0 to the total number of input pins minus 1.
- Click “Read” button and the status of the GPIO pin appears in “(R/W) Result”.

Read Multiple Input Pins

- Click “Multiple-Pins” radio button.
- Type a pin number from ‘0x01’ to ‘0x0F’ to read the status of the input pins. The pin numbers are bitwise-ORed, i.e. bit 0 stands for input pin 0, bit 1 stands for input pin 1, etc. For example, to read input pins 0, 1, and 3, type ‘0x0B’ into the “Multi-Pins” text box.
- Click the “Read” button and the status of the GPIO pins appears in the “(R/W) Result” text box.

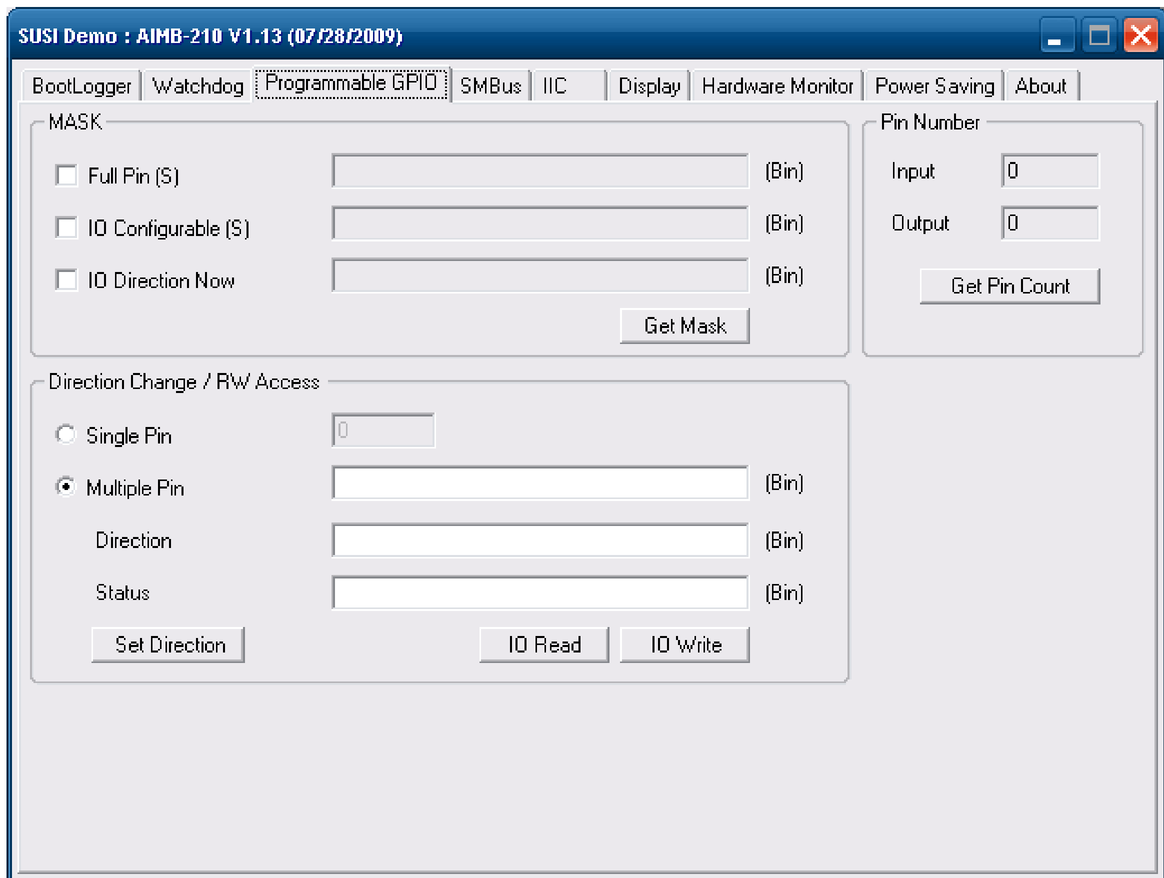
Write Single Output Pin

- Click the “Single-Pin” radio button.
- Type the output pin number to write the status to. Pins are numbered from 0 to the total number of input pins minus 1.
- Type either '0' or '1' in “(R/W) Result” to set the output status as low or high.
- Click “Write” button to perform the operation.

Write Multiple Output Pins

- Click the “Multi-Pins” radio button.
- Type a pin number from ‘0x01’ to ‘0x0F’ to choose the output pins to write. The pin numbers are bitwise-ORed, i.e. bit 0 stands for output pin 0, bit 1 stands for output pin 1, etc. For example, to write input pins 0, 1, and 3, type ‘0x0B’ into the “Multi-Pins” text box.
- Type a value from ‘0x01’ to ‘0x0F’ into the “(R/W) Result” text box to set the status of the output pins. Again, the pin statuses are bitwise-ordered, i.e. bit 0 stands for the desired status of output pin 0, bit 1 for output pin 1, etc. For example, if you want to set pin 0 and 1 high, 3 to low, the value given in text box of “(R/W) Result” should be ‘0x0A’.
- Click “Write” button to perform the operation.

iv. Programmable GPIO



(Figure-4)

Pin Number

- Get the numbers of input pins and output pins respectively. Each number may vary with the direction of current pins, but the sum remains the same.

MASK

- Choose the mask of interest by selecting or clearing its check box, then clicking "Get Mask".

Direction Change / RW Access

- Choose either "Single Pin" or "Multiple Pin".
- The possible values that the "Single Pin" text box can be set to ranges from 0 to the total number of GPIO pins minus 1.

Single Pin Operation – "IO Write" / "Set Direction"

- Give a value of '1' (output status high / input direction) or '0' (output status low / output direction) to set the pin then click the "IO Write" or "Set Direction" button.

Single Pin Operation – "IO Read"

- Click "IO Read" to get the pin input status.

Multiple Pin Operation – "IO Write" / "Set Direction"

If there are 8 GPIO pins:

- To write the status of GPIO output pins 0, 1, 6 and 7, give the "Multiple Pin" text box the value 11000011. Bit 0 stand for GPIO 0, bit 1 stand for GPIO 1, and so on.
  To set pin 0 as high, pin 1 as low, pin 6 as high and pin 7 as low, give the "Value" text box the value 01XXXX01, where X stands for a don't care pin.
  Please simply assign a 0 for don't care pins, e.g. 10000001.
- To set the direction of GPIO pins 0, 1, 6 and 7, give the "Multiple Pin" text box the value 11000011. Again bit 0 stands for GPIO 0, bit 1 stands for GPIO 1, and so on. To set pin 0 as an input, pin 1 as an output, pin 6 as an input and pin 7 as an output, give the "Value" text box with 01XXXX01, where X is for don't care
  Please simply assign a 0 for don't care pins, e.g. 10000001.

Multiple Pin Operation – "IO Read"

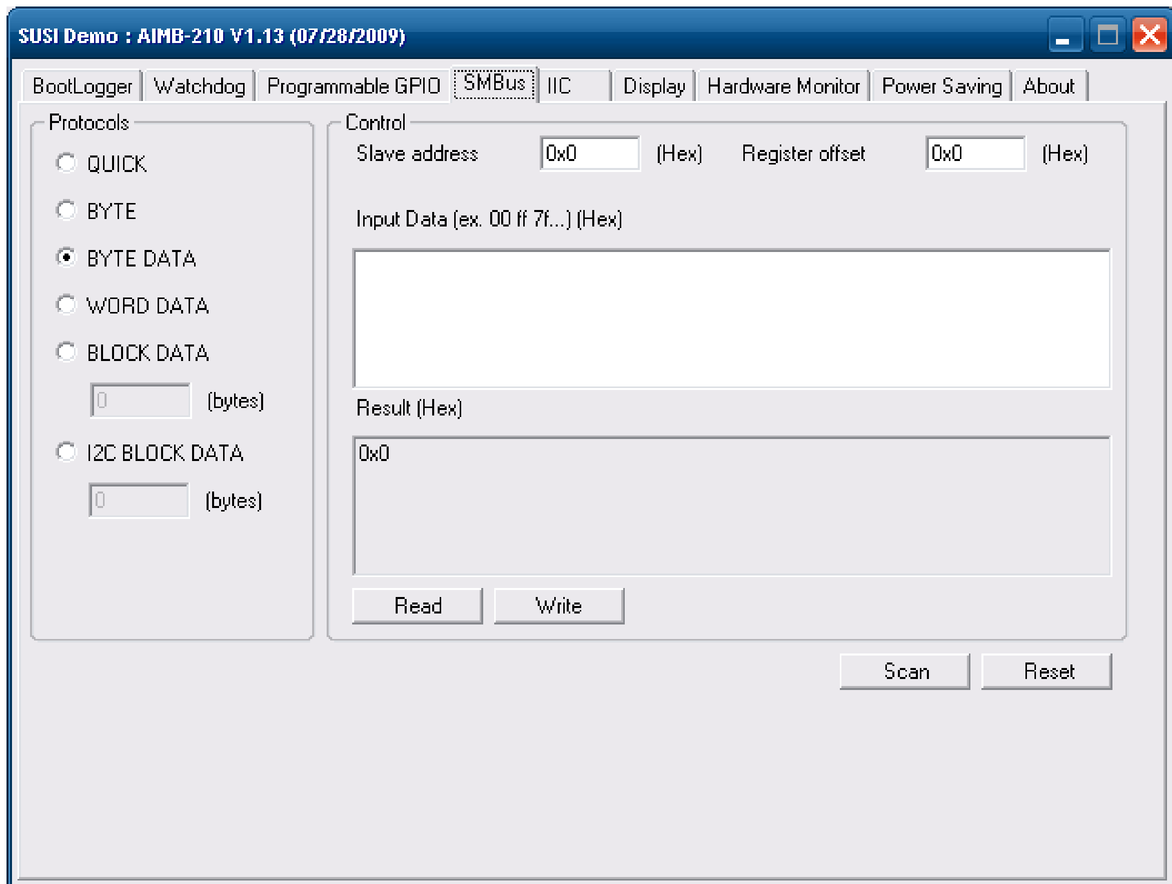
- For example, if you want to read the status of GPIO pins 0, 1, 6 and 7, give the "Multiple Pin" text box the value 11000011. Bit 0 stands for GPIO 0, bit 1 stands for GPIO 1, and so on. Again, if the pin is in status high, the value in the relevant bit of the "Value" text box will be 1. If the pin status is low, the "Value" text box will be 0.

[Note]

1. "IO Write" can only be performed on pins in the output direction.
2. "Set Direction" can only be performed on bidirectional pins.
3. "IO Read" can get the status of both input and output pins.

Please get the information first in the "MASK" field.

v. SMBus



(Figure-5)

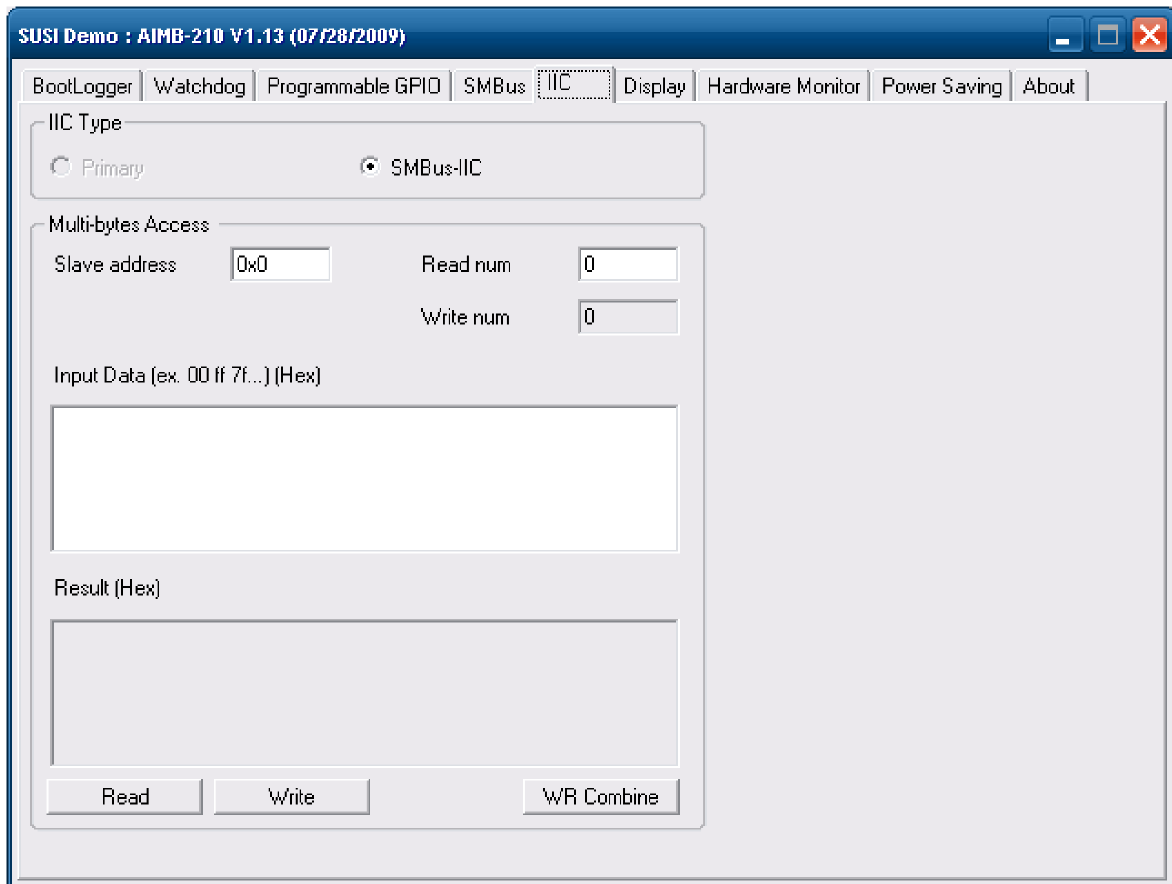
Protocols

- Choose one of the protocol operations by selecting a radio button.
- Give the proper value to the "Slave address" and "Register offset" text boxes. Some protocol operations don't have register offsets. Slave addresses must be converted from 7-bit to 8-bit (e.g. Datasheet say device has 7-bit address 0x20, then you have to type in 0x40)
- Click the "Read" button for read/receive operations, and the "Write" button for write/send operations. Slave addresses must be converted from 7-bit to 8-bit (e.g. Datasheet say device has 7-bit address 0x20, then you have to type in 0x40)
- The values read or to be written are in the "Result (Hex)" text box.

"Scan" Button (Scan Address Occupancy)

- Click this button to get the addresses currently used by slave devices connected to the SMBus.
- The occupied addresses will be shown in the "Result (Hex)" text box. The addresses are already in an 8-bit format (that means if your device has the address 0x20 it will show 0x40).

vi. Multi-byte IIC



(Figure-6)

- Select the "Primary" or "SMBus-IIC" radio button. If one of them is not supported, its radio button will be unavailable.

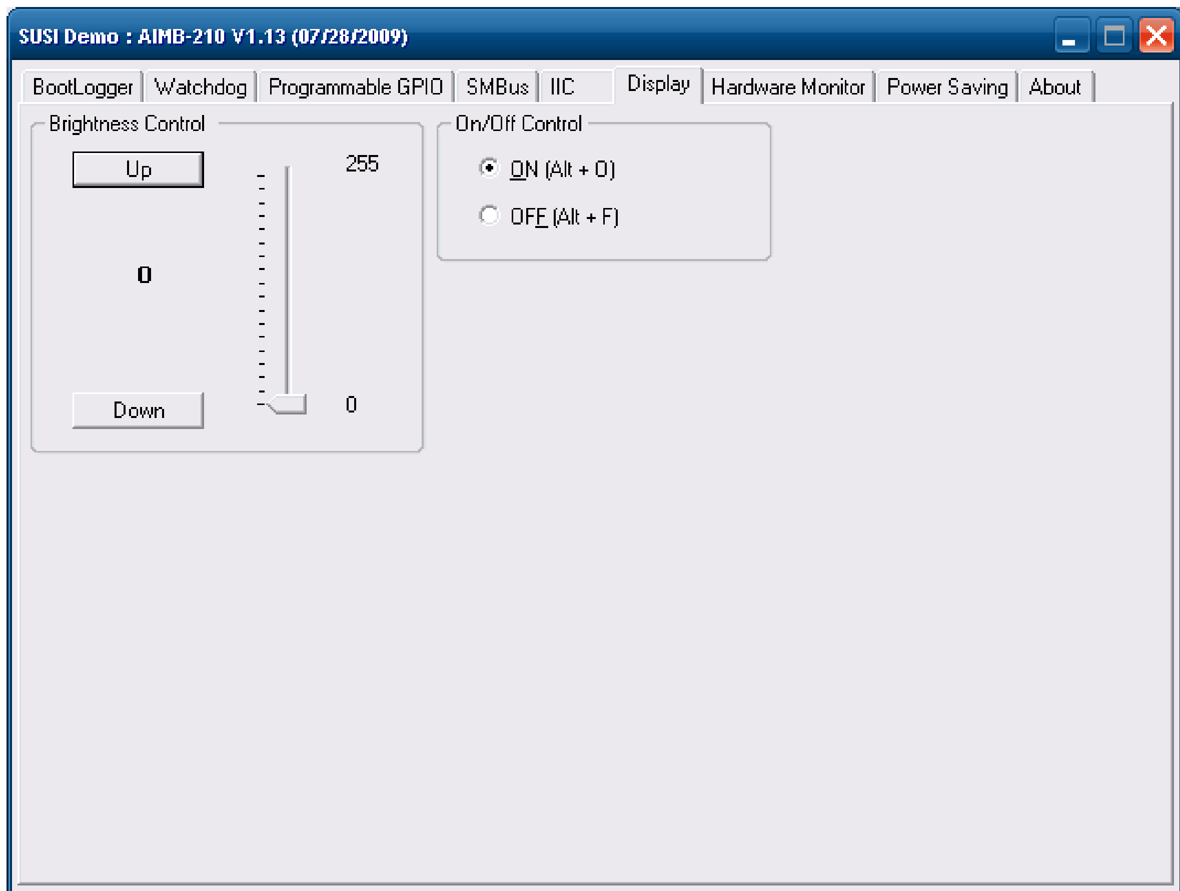
Primary

- Connect the IIC devices to the IIC connector.
- Type in the data bytes to be written in the "Input Data" text box.
- The bytes read will be shown in the "Result" text box.

SMBus-IIC

- Connect the IIC devices to the SMBus connector.
- In AMD platforms, all the IIC functions are fully supported.
- In Intel or VIA platforms, only Read and Write with "Read num" = 1 or "Write num" = 1 are supported. "WR Combine" is not supported.

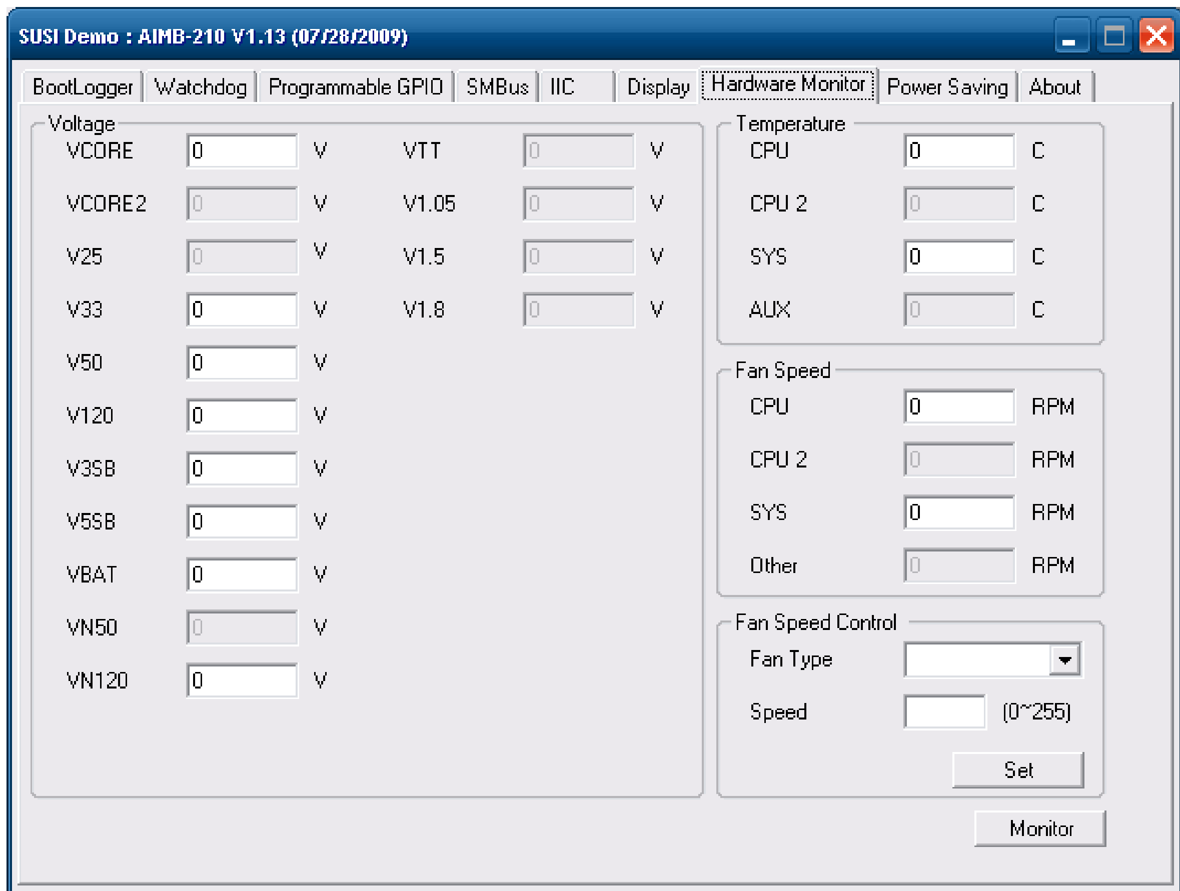
vii. VGA Control



(Figure-7)

You may control VGA functions from the “Display” tab or directly by hotkey.
If the brightness control is not supported, the control parts are unavailable (grayed-out).

viii. Hardware Monitor



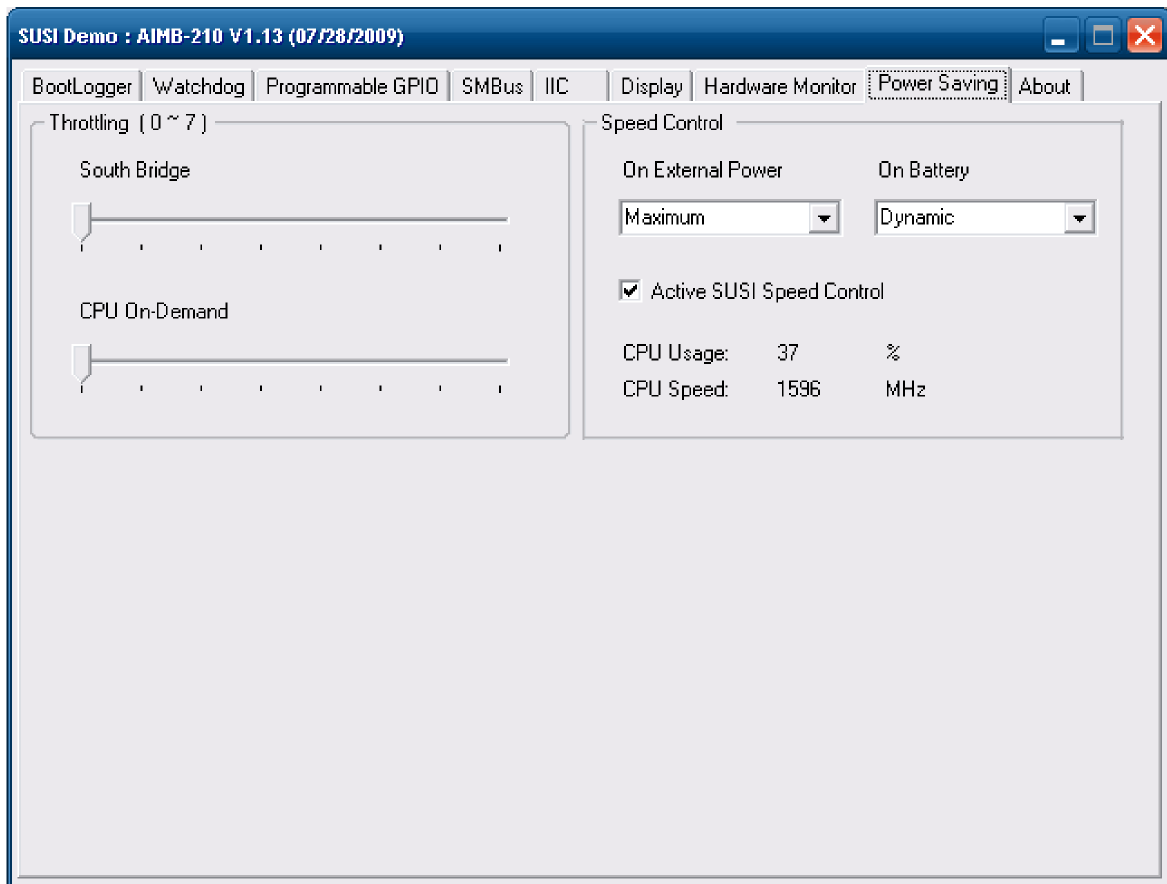
(Figure-8)

Click "Monitor" to get and display the hardware monitor values. If a data value is not supported on the platform, its text box will be unavailable (grayed-out).

The Fan Speed Control function includes Pulse Width Modulation (PWM) control. With Speed you determinate the duty cycle. Higher value means longer duty cycle and therefore higher speed.

Note: Some FAN's are going to operate at full speed if the input signal is too low. This is a security feature of the FAN's. You can slowly decrease FAN speed to find out what the minimum FAN speed for your system is.

ix. Power Saving

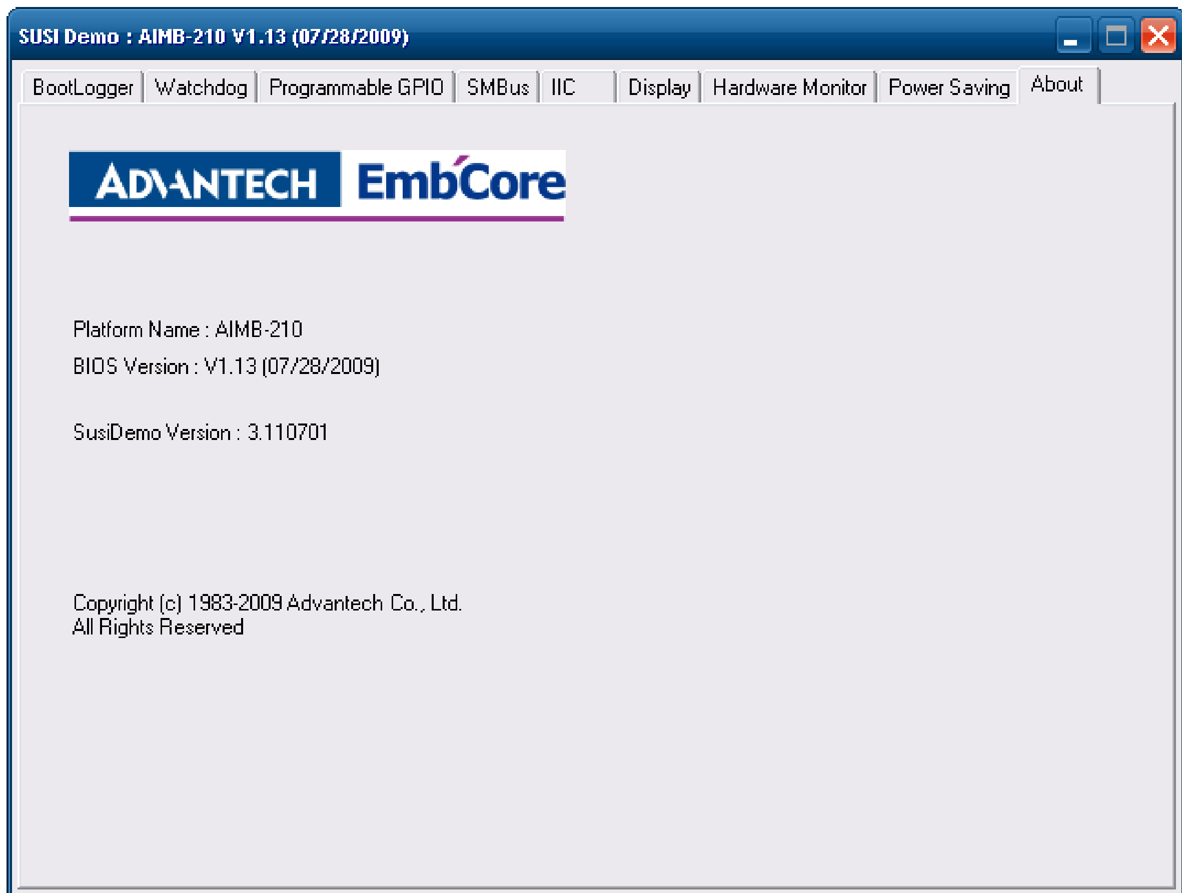


(Figure-9)

There are two methods to control the throttling configuration: South bridge and CPU on-demand.

Speed control uses windows XP internal scheme for power management configuration.

x. About



(Figure-10)

This page contains the platform name, the BIOS version etc., i.e. the information retrieved by the SUSI APIs. You can use this page to check if your installation is okay. If there is not a valid product name, contact your local FAE.

SUSI demo versions show you the major SUSI version (here 3.0) and the minor revision. The minor revision (here 110701) is also the compiling date of your SUSI.DLL in the format YY/MM/DD.

If you have any problems, it is recommended to send your local FAE a screenshot of this site or at least the data which are shown here.

Programming Overview

Header Files

- REL_SUSI.H includes API declaration, constants and flags that are required for programming.
- REL_DEBUG.H / REL_ERRDRV.H / REL_ERRLIB.H are for debug code definitions.
  REL_DEBUG.H – Function index codes
  REL_ERRLIB.H – Library error codes
  REL_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 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

There are 4 functions which are driver-independent. These 4 functions have the prefix `SusiDLL`. All other functions depending on the correlating driver. 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 `SusiDLLGetLastError` 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

Driver independent functions

- `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 `SusiCoreAccessRunTimer` APIs are **Boot Logger** features that enable monitoring of system reboot times, total OS run time and continual run time. `SUSIPlus` APIs are **CPU** features.

- `SusiCoreGetPlatformName`
- `SusiCoreGetBIOSVersion`
- `SusiCoreAccessBootCounter`
- `SusiCoreAccessRunTimer`
- `SusiCoreGetCpuVendor`
- `SusiCoreGetCpuMaxSpeed`
- `SUSIPlusCpuSetThrottling`
- `SUSIPlusCpuGetThrottling`
- `SUSIPlusCpuSetOnDemandThrottling`
- `SUSIPlusCpuGetOnDemandThrottling`
- `SusiPlusSpeedIsActive`
- `SusiPlusSpeedSetActive`
- `SusiPlusSpeedSetInactive`
- `SusiPlusSpeedWrite`
- `SusiPlusSpeedRead`

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.

- `SusiWDAvailable`
- `SusiWDGetRange`
- `SusiWDSetConfig`
- `SusiWDTrigger`
- `SusiWDDisable`
- `SusiWDSetConfigEx`
- `SusiWDTriggerEx`

- `SusiWDDisableEx`

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:

- `SusiIOAvailable`
- `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 :

- `SusiSMBusAvailable`
- Quick Command – `SusiSMBusReadQuick /SusiSMBusWriteQuick`
- Byte Receive/Send – `SusiSMBusReceiveByte /SusiSMBusSendByte`
- Byte Data Read/Write – `SusiSMBusReadByte /SusiSMBusWriteByte`
- Word Data Read/Write – `SusiSMBusReadWord /SusiSMBusWriteWord`
- Block Read/Write – `SusiSMBusReadBlock /SusiSMBusWriteBlock`
- I^2C Block Read/Write – `SusiSMBusI2CReadBlock / SusiSMBusI2CWriteBlock`

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 and uses SMBus Quick Command to do so. 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 a new device to quickly know their addresses. The `SlaveAddress_7` parameter given in this API is a 7-bit address.

IIC functions

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

- `SusiIICAvailable`
- `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.
On 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.

VGA Control (VC) functions

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

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

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

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`

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(DWORD *major, DWORD *minor)
```

Parameters

`major`

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

`minor`

`[out]` Pointer to a variable containing the minor version number. Minor version is the compiling date of Library in format YYMMDD

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 BIOS string back in size. 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\ Advantech\SUSI \BootCounter\Enable
HKEY_LOCAL_MACHINE \SYSTEM\ Advantech\SUSI \BootCounter\BootTimes

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\Advantech\SUSI\RunTimer\Running
HKEY_LOCAL_MACHINE\SYSTEM\ Advantech\SUSI\RunTimer\Autorun
HKEY_LOCAL_MACHINE\SYSTEM\ Advantech\SUSI\RunTimer\ContinualOnTime
HKEY_LOCAL_MACHINE\SYSTEM\ Advantech\SUSI\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.

SUSIPlusCpuSetThrottling

Set the CPU throttling by South Bridge

```
BOOL SUSIPlusCpuSetThrottling(unsigned char step)
```

Parameters

value

[in] CPU Throttling value, range is 0~7.

Return Value

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

Remarks

Step value 0 means FULL speed, pre increase one reduce 12.5%, maximum value is 7.

SUSIPlusCpuGetThrottling

Get the CPU throttling from South Bridge

```
BOOL SUSIPlusCpuGetThrottling(unsigned char *step)
```

Parameters

value

[out] Get the CPU Throttling value, range is 0~7.

Return Value

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

Remarks

Step value 0 means FULL speed, pre increase one reduce 12.5%, maximum value is 7.

SUSIPlusCpuSetOnDemandThrottling

Set the CPU throttling by CPU On-Demand

```
BOOL SUSIPlusCpuSetOnDemandThrottling(HANDLE proc_handler,
unsigned char cpu_index, unsigned char step)
```

Parameters

`proc_handler`

`[in]` Processor's handle

`cpu_index`

`[in]` Select core which want to control

`step`

`[in]` CPU throttling value, range is 0~7.

Return Value

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

Remarks

Step value 0 means FULL speed, pre increase one reduce 12.5%, maximum value is 7.

SUSIPlusCpuGetOnDemandThrottling

Get the CPU throttling from CPU On-Demand

```
BOOL SUSIPlusCpuGetOnDemandThrottling(HANDLE proc_handler,
unsigned char cpu_index, unsigned char *step)
```

Parameters

`proc_handler`
  `[in]` Processor's handle

`cpu_index`
  `[in]` Select core which want to control

`step`
  `[out]` CPU throttling value, range is 0~7.

Return Value

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

Remarks

Step value 0 means FULL speed, pre increase one reduce 12.5%, maximum value is 7.

SUSIPlusSpeedIsActive

Check if power scheme of windows XP is active

```
BOOL SUSIPlusSpeedIsActive(void)
```

Parameters

None.

Return Value

TRUE (1) indicates active; FALSE (0) indicates inactive.

Remarks

This power scheme is a customized scheme, named `Susi Speed Control`.

SUSIPlusSpeedSetActive

Create a customized scheme named `Susi Speed Control`.

```
int SUSIPlusSpeedSetActive(void)
```

Parameters

None.

Return Value

value	Meaning
-1	SUSI doesn't initial.
-2	Cannot use power scheme.
-4	Create power scheme failed.
-5	Delete power scheme failed.
-6	Read power scheme failed.
-7	Set power scheme failed.
0	Succeed

Remarks

Support Windows XP Series only.

SUSIPlusSpeedSetInactive

Delete power scheme that named `Susi Speed Control`.

```
int SUSIPlusSpeedSetInactive(void)
```

Parameters

None.

Return Value

value	Meaning
-1	SUSI doesn't initial.
-2	Cannot use power scheme.
-3	Set power scheme failed.
-4	Delete power scheme failed.
0	Succeed

Remarks

Support Windows XP Series only.

SUSIPlusSpeedWrite

It can change settings of power scheme.

```
int SUSIPlusSpeedWrite(BYTE ACPolicy, BYTE DCPolicy)
```

Parameters

ACPolicy
  [in] Specifies processor power policy on AC mode

DCPolicy
  [in] Specifies processor power policy on DC mode

Return Value

value	Meaning
-1	SUSI doesn't initial.
-2	SUSI speed control is invalid.
-3	Write power scheme failed.
-4	Set power scheme failed.
0	Succeed

Remarks

Processor power policy value as following:

Policy	Value
Maximum	0
Minimum	1
Dynamic	3

SUSIPlusSpeedRead

It can read settings of power scheme.

```
int SUSIPlusSpeedRead(BYTE *ACPolicy, BYTE *DCPolicy)
```

Parameters

`ACPolicy`
`[out]` Processor power policy on AC mode

`DCPolicy`
`[out]` Processor power policy on DC mode

Return Value

value	Meaning
-1	SUSI doesn't initial.
-2	SUSI speed control is invalid.
-3	Read power scheme failed.
0	Succeed

Remarks

Processor power policy value as following:

Policy	Value
Maximum	0
Minimum	1
Dynamic	3

SusiCoreGetMaxCpuSpeed

Get max CPU speed

```
BOOL SusiCoreGetCpuMaxSpeed(DWORD &Value)
```

Parameters

value

[out] Get the CPU Max CPU Speed value

Return Value

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

Remarks

N/A

SusiCoreGetCpuVendor

Get the CPU Vendor type

```
BOOL SusiCoreGetCpuVendor(DWORD &Value)
```

Parameters

value

[out] Get the CPU vendor type

```
// Vendor
#define INTEL     1 << 0
#define VIA       1 << 1
#define SIS       1 << 2
#define NVIDIA    1 << 3
#define AMD       1 << 4
#define RDC       1 << 5
```

Return Value

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

Remarks

N/A

SusiWDAvailable

Check if the watchdog driver is available.

```
int 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.

SusiWDSetConfigEx

Extend watchdog timer set configuration function for multi-WDT. Start watchdog timer with specified timeout value.

```
BOOL SusiWDSetConfigEx(int group_number, DWORD delay,
DWORD timeout)
```

Parameters

`group_number`

`[in]` Specifies the number of watchdog timer, 0 is first WDT.

`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 `SusiWDTriggerEx` 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 `SusiWDTriggerEx` equals a call to `SusiWDSetConfigEx` with `delay` 0 and the original `timeout` value, so if you want to change the timeout value, call `SusiWDSetConfigEx` with new `timeout` value instead of `SusiWDTriggerEx`.

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`.

SusiWDTriggerEx

Extend watchdog timer trigger function for multi-WDT. Reload the watchdog timer to the timeout value given in SusiWDSetConfigEx to prevent the system from rebooting.

```
BOOL SusiWDTriggerEx(int group_number)
```

Parameters

group_number

[in] Specifies the number of watchdog timer, 0 is first WDT.

Return Value

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

Remarks

A watchdog protected application has to call `SusiWDTriggerEx` continuously to indicate that it is still working properly and prevent a system restart. The first call to `SusiWDTriggerEx` in the middle of a delay resulting from a previous call to `SusiWDSetConfigEx` 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 `SusiWDSetConfigEx`.

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.

SusiWDDisableEx

Extend watchdog timer disable function for multi-WDT.

```
BOOL SusiWDDisableEx(int group_number)
```

Parameters

group_number

`[in]` Specifies the number of watchdog timer, 0 is first WDT.

Return Value

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

Remarks

If watchdog protection is no longer required by an application, it can call `SusiWDDisableEx` to disable the watchdog. A call to `SusiWDDisableEx` in the middle of a delay resulting from a previous call to `SusiWDSetConfigEx` causes the delay timer to be canceled immediately and stops watchdog timer countdown. Only a few hardware implementation 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.

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"

SusiSMBusReadBlock

Read multi-data from the target slave device in the SMBus.

```
BOOL SusiSMBusReadBlock(BYTE SlaveAddress, BYTE
RegisterOffset, BYTE *Result, BYTE *ByteCount)
```

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 byte array in which the function reads the block data.

ByteCount

`[in]` Pointer to a byte in which specifies the number of bytes to be read and also return succeed bytes.

Return Value

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

Remarks

None.

SusiSMBusWriteBlock

Write multi-data to the target slave device in the SMBus.

```
BOOL SusiSMBusWriteBlock(BYTE SlaveAddress, BYTE
RegisterOffset, BYTE *Result, BYTE ByteCount)
```

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

`[out]` Pointer to a byte array in which the function writes the block data.

ByteCount

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

Return Value

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

Remarks

None.

SusiSMBusI2CReadBlock

Read multi-data from the target slave device by I^2C block read protocol in the SMBus.

```
BOOL SusiSMBusI2CReadBlock(BYTE SlaveAddress, BYTE
RegisterOffset, BYTE *Result, BYTE *ByteCount)
```

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 byte array in which the function reads the block data.

ByteCount

`[in]` Pointer to a byte in which specifies the number of bytes to be read and also return succeed bytes.

Return Value

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

Remarks

None.

SusiSMBusI2CWriteBlock

Write multi-data to the target slave device by I^2C block write protocol in the SMBus.

```
BOOL SusiSMBusI2CWriteBlock(BYTE SlaveAddress, BYTE
RegisterOffset, BYTE *Result, BYTE ByteCount)
```

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`

`[out]` Pointer to a byte array in which the function writes the block data.

`ByteCount`

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

Return Value

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

Remarks

None.

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 I²C bus.

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

Parameters

IICType

[in] Specifies the I²C 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 I²C 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^{2}C bus.

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

Parameters

IICType

[in] Specifies the I^{2}C 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^{2}C 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.
1	The function succeeds; the platform supports only brightness APIs.
2	The function succeeds; the platform supports only screen on/off APIs.
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
The flags refer "Appendix A - Hardware Monitor Flags - 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
The flags refer "Appendix A - Hardware Monitor Flags - 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 The flags refer “Appendix A - Hardware Monitor Flags - Voltage”

[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
The flags refer “Appendix A - Hardware Monitor Flags - 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.

Appendix A – Programming Flags Overview

Hardware Monitor Flags

- ### Fan

Flag	Value	Description
FCPU	0x0001	CPU FAN
FSYS	0x0002	System FAN
F2ND	0x0004	3rd FAN
FCPU2	0x0008	CPU 2 FAN
FAUX2	0x0010	3rd FAN 2

- ### Temperature

Flag	Value	Description
TCPU	0x0001	CPU Temperature
TSYS	0x0002	System Temperature
TAUX	0x0004	3rd Temperature
TCPU2	0x0008	CPU 2 Temperature

- ### Voltage

Flag	Value	Description
VCORE	0x0001	Vcore
V25	0x0002	2.5V
V33	0x0004	3.3V
V50	0x0008	5V
V120	0x0010	12V
V5SB	0x0020	Voltage of standby 5V
V3SB	0x0040	Voltage of standby 3V
VBAT	0x0080	VBAT
VN50	0x0100	-5V
VN120	0x0200	-12V
VTT	0x0400	VTT
VCORE2	0x0800	Vcore 2
V105	0x1000	1.05V
V15	0x2000	1.5V
V18	0x4000	1.8V

Boot Logger Flags

▪ Bootcounter

Mode Flag	Value	Description
ESCORE_BOOTCOUNTER_MODE_GET	0x0001	Read Operation
ESCORE_BOOTCOUNTER_MODE_SET	0x0002	Write Operation

Element Flag	Value	Description
ESCORE_BOOTCOUNTER_STATUS	0x0001	Current Status (Is Enabled or Disabled?)

▪ Runtimer

Mode Flag	Value	Description
ESCORE_RUNTIMER_MODE_GET	0x0001	Read Operation
ESCORE_RUNTIMER_MODE_SET	0x0002	Write Operation

Element Flag	Val.	Description
ESCORE_RUNTIMER_STATUS_RUNNING	0x01	Current Status (Is Enabled or Disabled?)
ESCORE_RUNTIMER_STATUS_AUTORUN	0x02	Is AutoRun upon Startup?
ESCORE_RUNTIMER_VALUE_CONTINUALON	0x04	OS continual run time (reset to 0 after a reboot)
ESCORE_RUNTIMER_VALUE_TOTALON	0x08	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 B - 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	ESusiCoreReadMemory
10900000	ESusiCoreWriteMemory
11000000	ESusiCoreReadIO
11100000	ESusiCoreWriteIO
11200000	ESusiCoreReadULongIO
11300000	ESusiCoreWriteULongIO
11400000	ESusiCorePciBusSetULong
11500000	ESusiCorePciBusGetULong
11600000	ESusiCoreGetCpuMaxSpeed
11700000	ESusiCoreGetCpuVendor
12000000	ESusiCoreEnableBootfail
12100000	ESusiCoreDisableBootfail
12200000	ESusiCoreRefreshBootfail

12300000	ESusiPlusSetThrottlingfail
12500000	ESusiPlusGetThrottlingfail
12700000	ESusiPlusGetOnDemandThrottlingfail
12800000	ESusiPlusSetOnDemandThrottlingfail
13000000	ESusiPlusSpeedIsActive
13100000	ESusiPlusSpeedSetActive
13200000	ESusiPlusSpeedSetInactive
13300000	ESusiPlusSpeedWrite
13400000	ESusiPlusSpeedRead
Watchdog	
20100000	ESusiWDInit
20200000	ESusiWDAvailable
20300000	ESusiWDDisable
20400000	ESusiWDGetRange
20500000	ESusiWDSetConfig
20600000	ESusiWDTrigger
20800000	ESusiWDTriggerEx
20900000	ESusiWDDisableEx
21000000	ESusiWDSetConfigEx
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
41600000	ESusiSMBusI2CReadBlock
41700000	ESusiSMBusI2CWriteBlock
41800000	ESusiSMBusReset
IIC	
50100000	ESusiIICInit
50200000	ESusiIICAvailable
50300000	ESusiIICReadByte
50400000	ESusiIICWriteByte
50500000	ESusiIICWriteReadCombine
50600000	ESusiIICRead
50700000	ESusiIICWrite
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
00000520	ERRLIB_CORE_CAN_NOT_WRITE_PWR_SCHEME
00000521	ERRLIB_CORE_GET_PWR_SCHEME_FAILED
00000522	ERRLIB_CORE_SET_PWR_SCHEME_FAILED
00000523	ERRLIB_CORE_DETECT_PWR_PROFILE_FAILED
00000524	ERRLIB_CORE_DELETE_PWR_PROFILE_FAILED
00000525	ERRLIB_CORE_CREATE_PWR_PROFILE_FAILED
00000526	ERRLIB_CORE_PWR_PROFILE_INVALID

00000527	ERRLIB_CORE_WRITE_PWR_SCHEME_FAILED
00000528	ERRLIB_CORE_READ_PWR_SCHEME_FAILED
Watchdog Functions	
00001024	ERRLIB_RESERVED1024
GPIO Functions	
00001200	ERRLIB_GPIO_DEVICE_INIT_FAIL
00001201	ERRLIB_GPIO_DEVICE_SETDIR_FAIL
00001202	ERRLIB_GPIO_DEVICE_GETDIR_FAIL
00001203	ERRLIB_GPIO_DEVICE_SETIO_FAIL
00001204	ERRLIB_GPIO_DEVICE_GETIO_FAIL
00001205	ERRLIB_GPIO_DEVICE_FUNC_INIT_FAIL
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)	