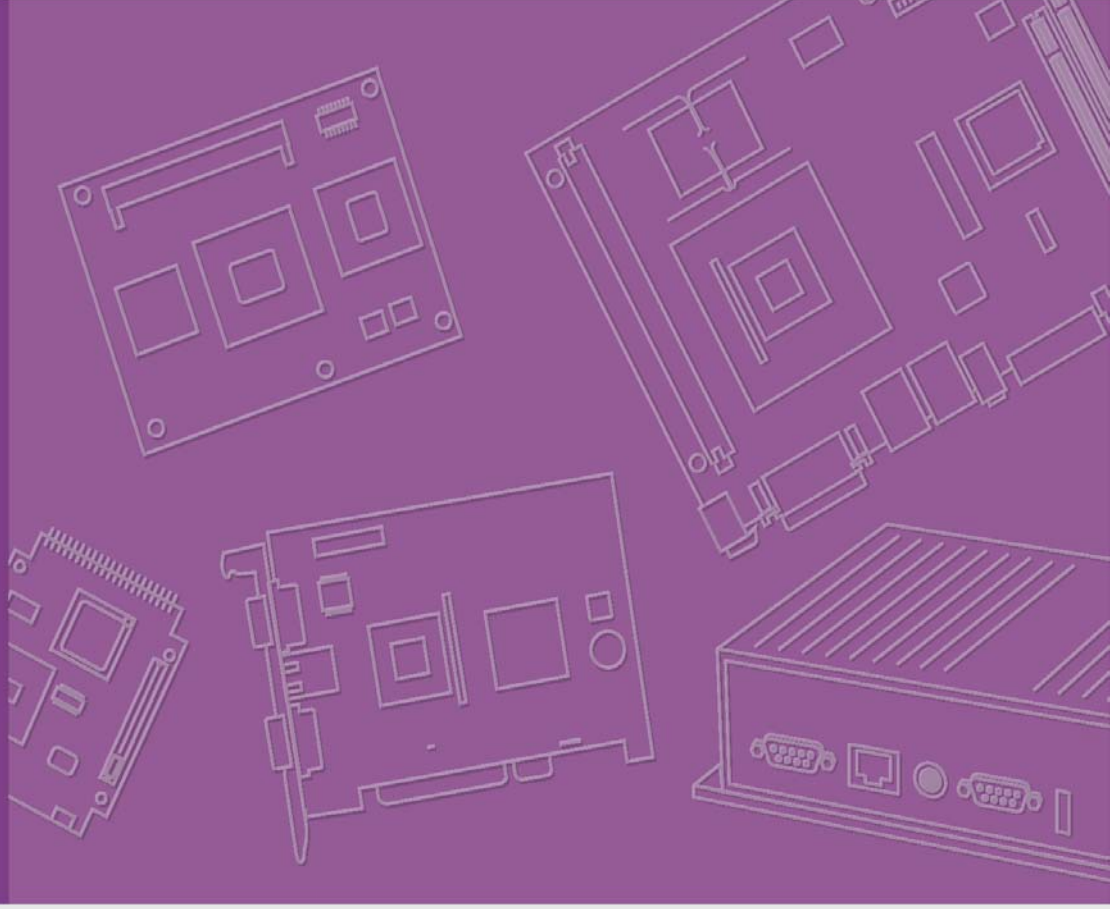


User Manual

iManager & Software API

ADVANTECH EmbCore

Embedded Boards, Modules and Software Services

Copyright

Part No. 200600SW00 Edition 1

Printed in Taiwan December 2009

Contents

Chapter 1

Introduction

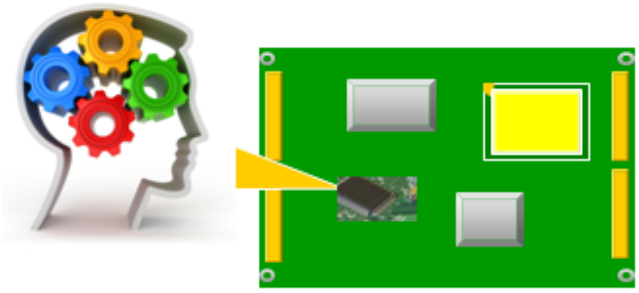
1.1 Intelligent Management for COM modules.

Advantech's new COM module comes equipped with "iManager" - a micro controller, providing embedded features for system integrators. Embedded features have been moved from the OS/BIOS level to the board level, to increase reliability and simplify integration.

iManager runs whether the system is powered on or off; it can count the boot times and running hours of the device, monitor device health, and provide an advanced watchdog if errors happen.

iManager also comes with a secure EEPROM for storing important security ID or other information.

All the embedded functions are configured by a utility. Advantech has done all the hard work for our customers with the release of a suite of Software APIs (Application Programming Interfaces). These provide 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.



1.2 Benefits

- **Simplify Integration**

  Unique embedded functions are built-in to the iManager's uniform set of APIs, such as watchdog, monitoring, smart battery, and so on.

  Offers a multi control interface for easy integration with all kind of peripherals, we have Standard I2C, SMBus and multi GPIO.

- **Enhance Reliability**

  Advanced watchdog, smart fan, hardware monitoring, CPU throttling; provided by eBrian independent from OS.

  Advantech eSOS is able to issue an alarm to customers when system crashes and further action can be taken from the remote side (Such as recovering the OS)

- **Secure the System**

  iManager provides an encryption space for customer data storage such as secure key for HDD lock, user ID and password, security ID to protect your application

- **Easy System Upgrade**

  Uniform and OS independent interface for cross hardware platforms

  Uniform API across different embedded OSs.

  Easily upgrade to other COM modules or different OS.

Chapter 2

Environments

2.1 iManager Utility

2.1.1 Hardware

This utility supports only Advantech ePlatforms with iManager module; please see the release notes to check the support list before using it.

2.1.2 Operating Systems

- Windows XP Professional
- Windows XP Embedded
- Windows Embedded Standard 2009
  - SUSI V4.0 driver and API are required
  - Dot Net Framework 2.0 required

2.2 iManager API:

2.2.1 Hardware

The Software API supports only Advantech ePlatforms with iManager module; please see the release notes to check the support list before using it.

2.2.2 Operating Systems

- Windows XP Professional
- Windows XP Embedded
- Windows Embedded Standard 2009
  - SUSI V4.0 driver and API are required
  - Dot Net Framework 2.0 required

Chapter 3

Installation

3.1 iManager Utility

Installation is not required; just copy all required files to a specific folder, then click "iManager.exe" to run. It will dynamically link the libraries. The required files are:

1.iManager.exe	Main program
2.PieChartControls.dll	iManager's library (Only for iManager.exe AP)
3.Susi.dll	iManager's external export Library (API)
4.SusiCore.dll	iManager's internal Library
5.SusiCore.sys	iManager's Driver

Chapter 4

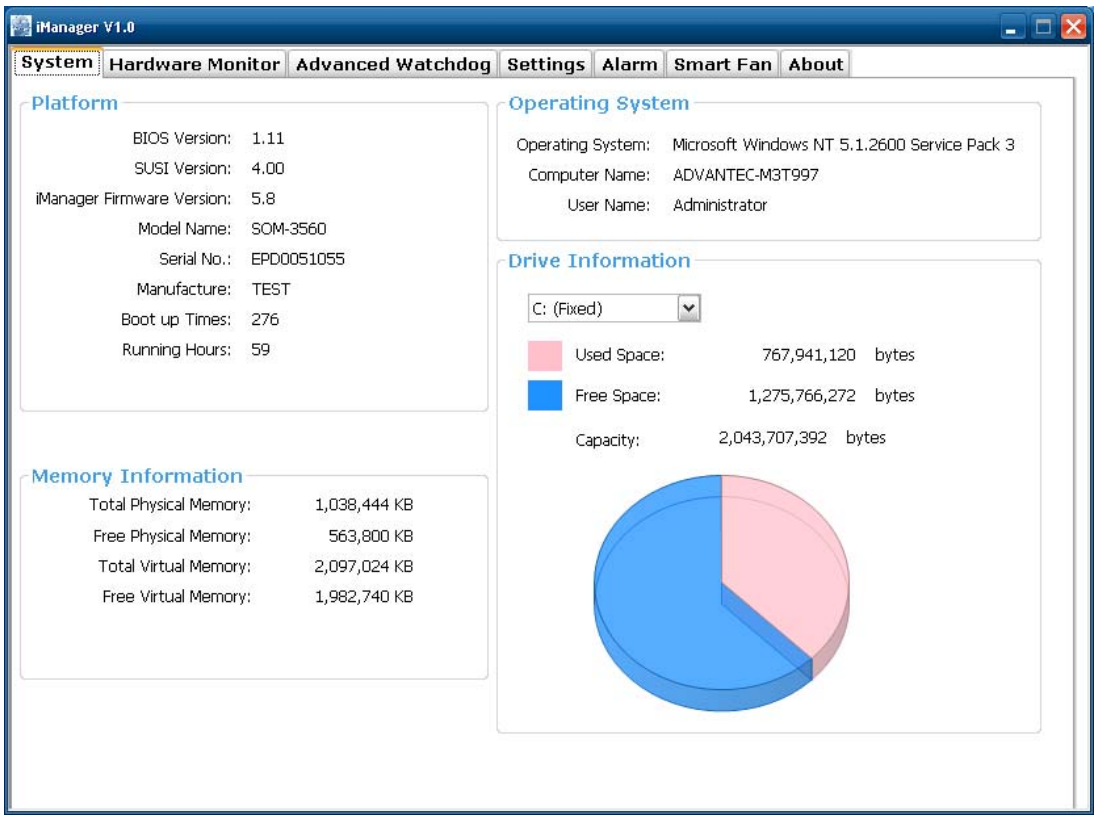
iManager Utility

iManager is a GUI utility which runs in Windows environment, It can be used to monitor the entire system and to popup a warning message in the system tray when something critical happens

Functions include: System, Hardware Monitor, Advanced Watchdog, Settings, Alarm and Smart Fan.

4.1 System Information

iManager can gather and record system information for users to manage their devices, including platform information, memory information, operating system information and hard disk information.

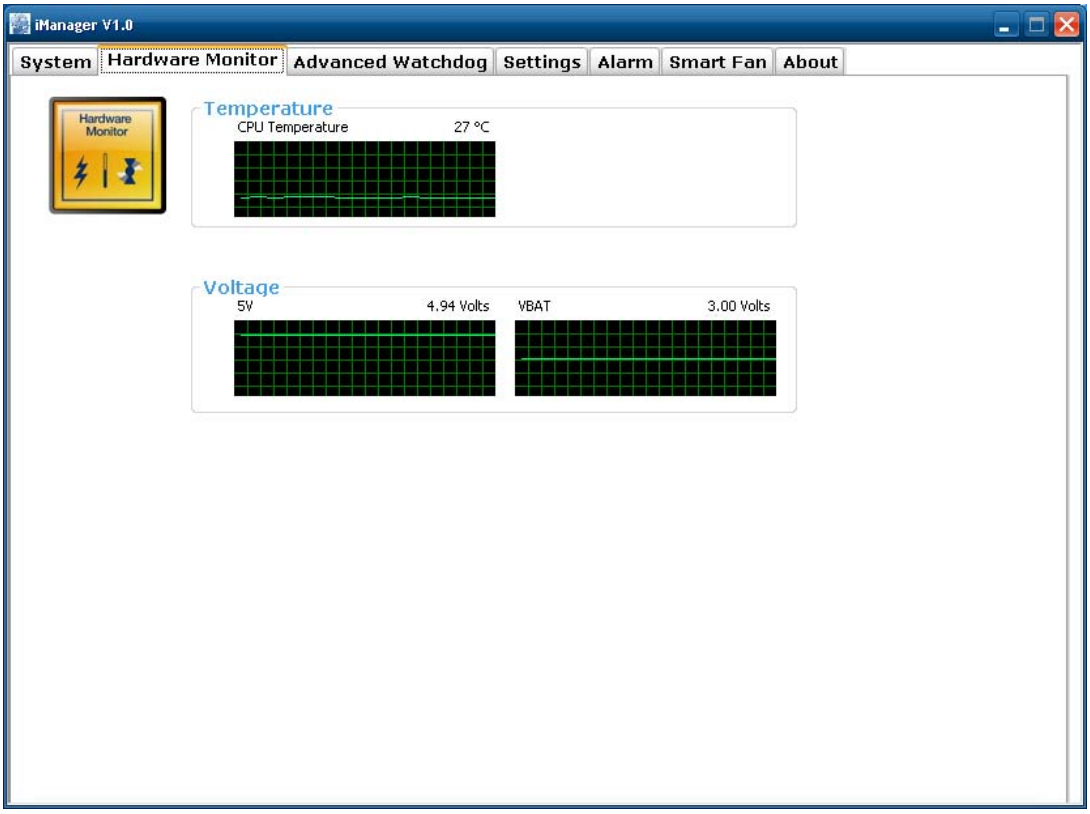


Platform information: The following information can be recorded on an Advantech board:

- BIOS Version: The version of BIOS file.
- SUSI Version: This is the driver version.
- iManager Firmware Version: This is the controller firmware version
- Model Name: This is platform name
- Serial Number: This number is input by the factory, used for sales tracking and service.
- Manufacturer: The creator of this platform
- Boot Counter: Boot up times
- Running Hours: Running times in hours

4.2 Hardware Monitor (HWM)

The hardware monitor contains three features: temperature, voltage and fan speed. It monitors critical items including power supply voltage, CPU & system fan speed, and CPU & system temperature. These items are important to the operation of the system because when errors happen, they may cause permanent damage to the PC. If an monitored item is outside its normal range, a warning message will pop up to remind the user to take corrective actions.

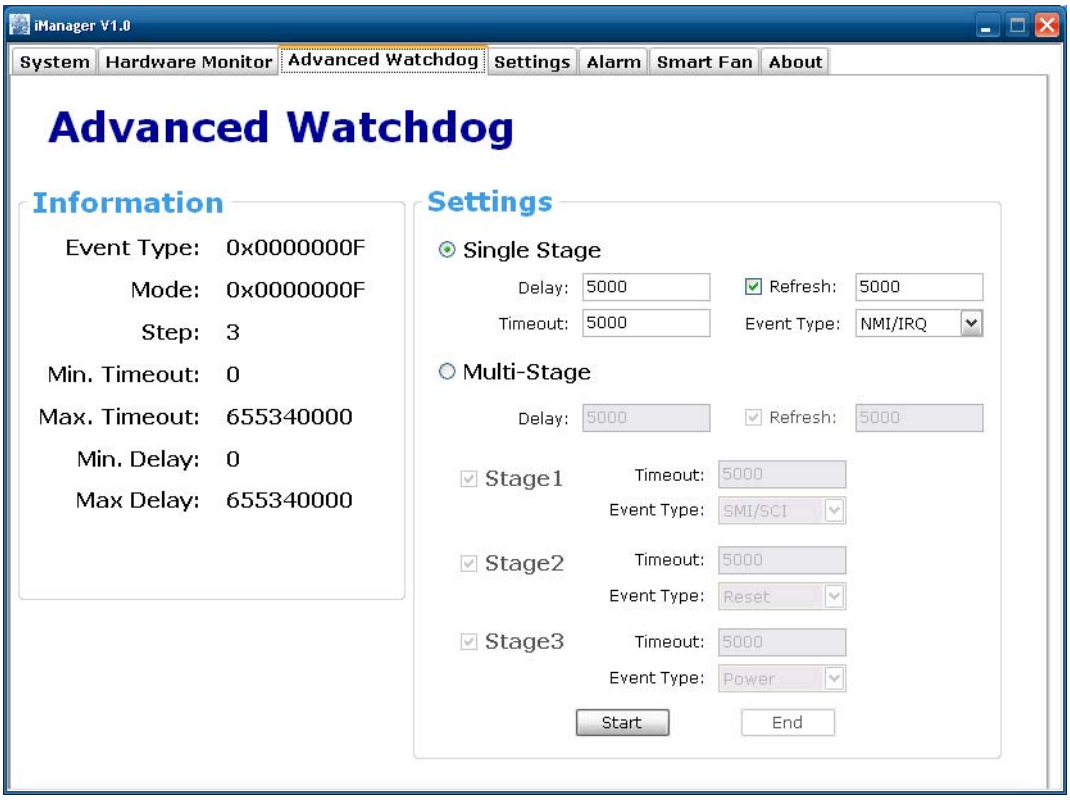


4.3 Advanced WatchDog

In general, a watchdog timer (WDT) is a function that performs a specific operation after a certain period of time when something goes wrong with the system.

A watchdog timer can be programmed to restart the system after a certain time period when a program or computer fails to respond or hangs.

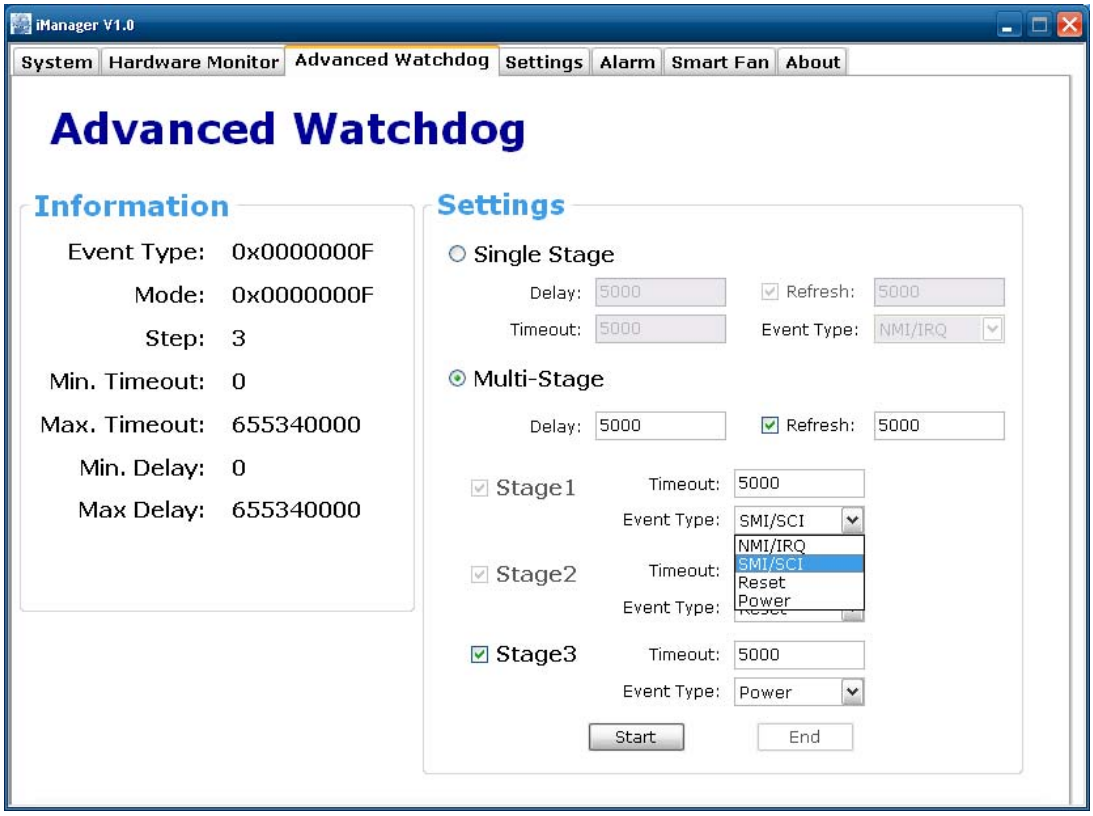
Since many customers like to program different responses to different events, Advantech has designed an advanced watchdog which consists of both a single stage and a multi-stage timer.



Single stage: In this stage, you can set delay time, refresh time, timeout in milliseconds and event type.

For example:

Type 3000 (3 sec.) in the "Timeout" text box, 5000 (5 sec.) in the "Refresh" text box and optionally type 5000 (5 sec.) in the "Delay" text box. Click the "Start" button. Because the refresh time is longer than the timeout time (5 > 3), the watchdog cannot get response and will execute the event by issuing an IRQ event.



Multi-stage: The multi stage watchdog allows up to 3 actions in each stage; one can set a different timeout in milliseconds based on event type.

For example:

Type 5000 (5 sec.) in the “Delay” box, Type 5000 (5 sec) in “Refresh”, then for each stage, set the "Timeout" text box to 5000 (5 sec).

For “Stage 1”, set the event type to NMI/IRQ, for “Stage 2”, set the event type to SMI/SCI, and for “Stage 3”, set the event type to Reset

Now, click "Start". The advanced watchdog will run the 3 stages. An application can be written to receive the events.

There are four types to select from:

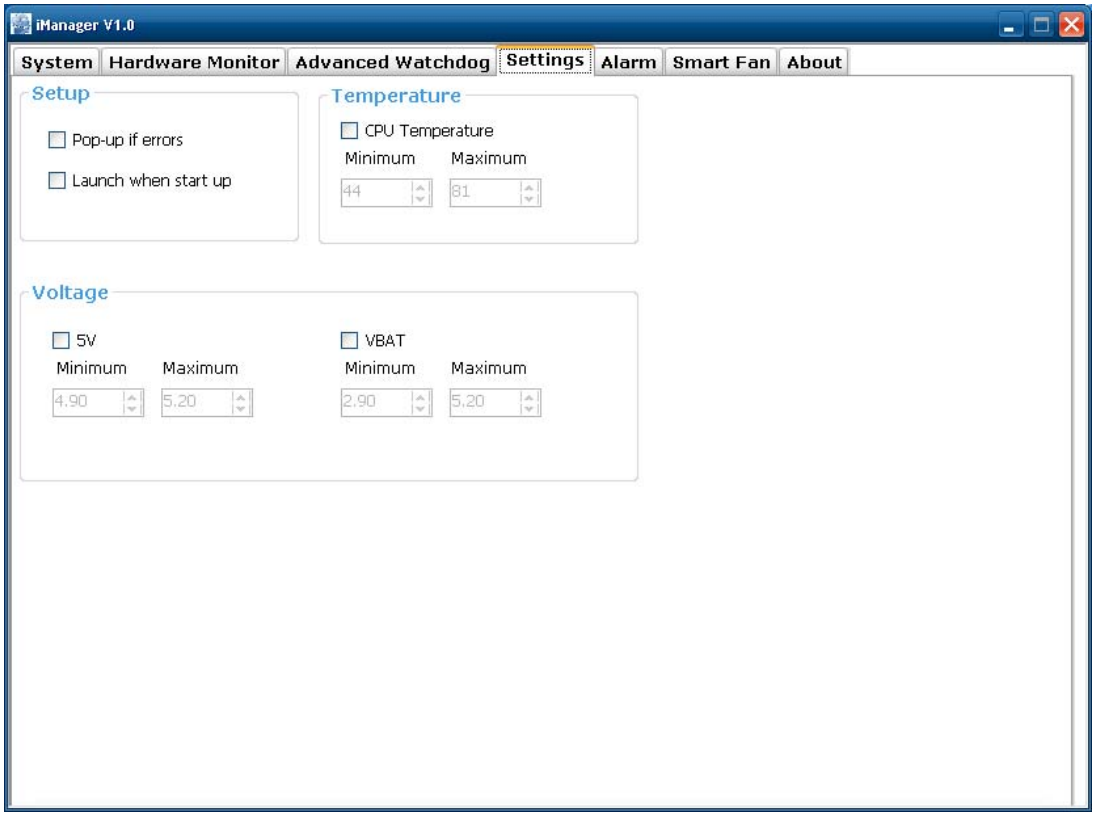
- NMI/IRQ
- SMI/SCI
- Reset
- Power

Note! *Valid event types will change for different platforms due to hardware limitations. Please reference the hardware platform user manual to get detailed information.*

4.4 Settings

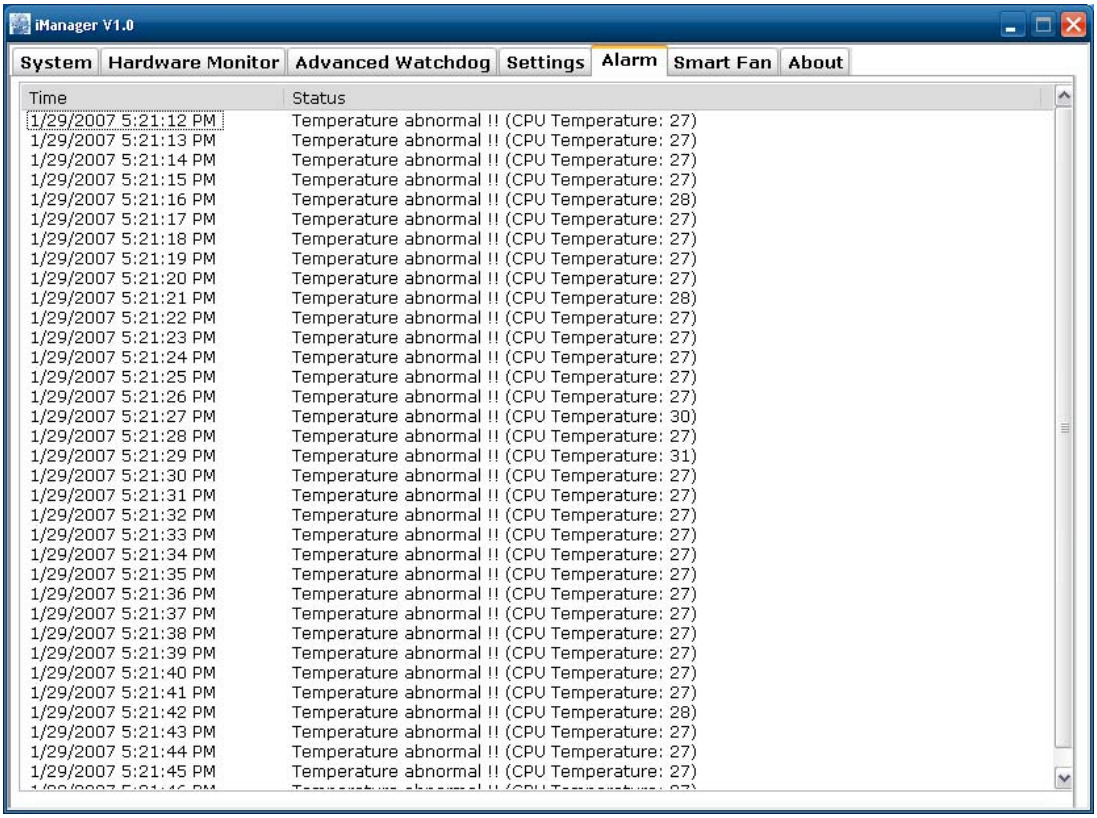
The settings page is used to set the minimum and maximum threshold values for temperature, fan speed and voltages. When the real value is outside this range, an error will be recorded on the alarm page.

In “Setup”, pop-up errors and the monitoring utility can be selected to run at system startup.



4.5 Alarm

On the alarm page, all errors are tracked by time and status



4.6 SmartFan

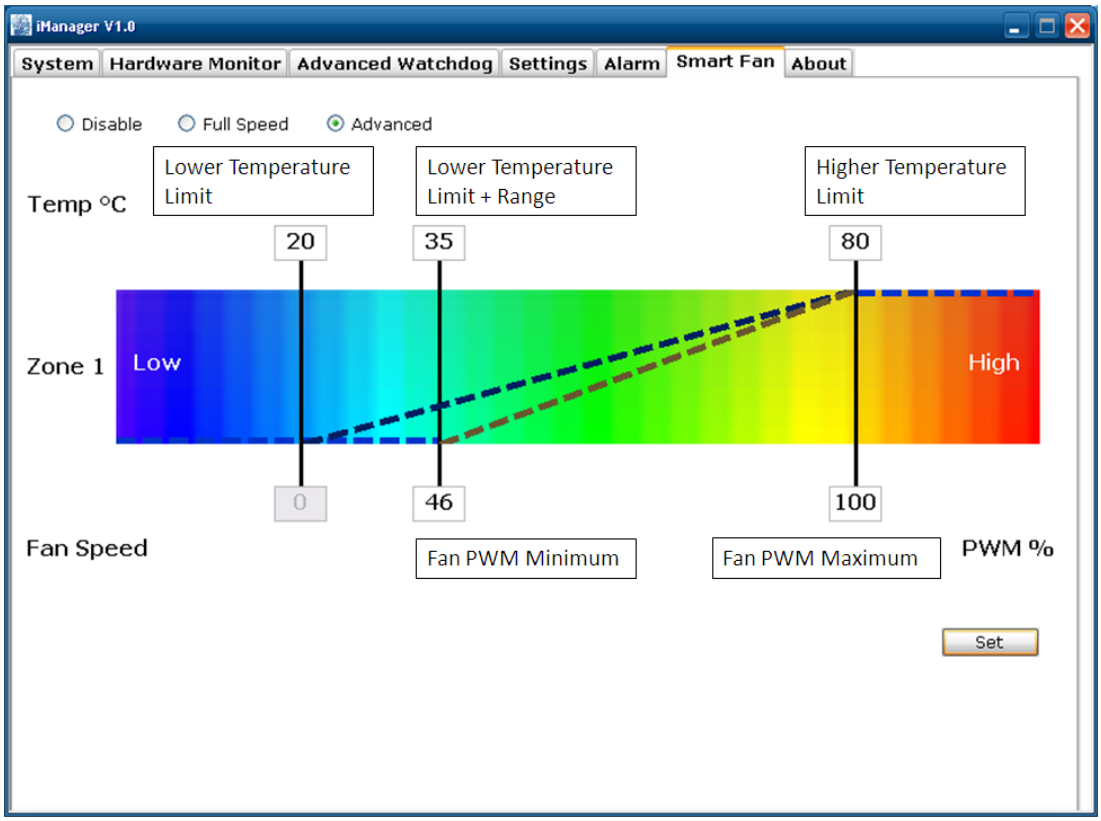
iManager's design provides a SmartFan for the user to pre-define the fan speed based on the system temperature. There are three modes for selection:

- Disable: Disable the SmartFan
- Full Speed: Fan always runs in full speed.
- Advanced: Users can set the "Lower Temperature Limit" (Fan speed = 0), "Lower Temperature Limit+ Range" (FAN PWM Minimum) and "Higher Temperature Limit" (FAN PWM Maximum). The SmartFan cycle will operate automatically; see Figure-8.

In Advanced Mode, when the temperature for a zone is between the "Lower Temperature Limit" and "Temperature Limit + Range", the speed of the fan assigned is determined as follows:

When the temperature reaches the "Fan Temp Limit" for a zone, the PWM output assigned to that zone will be "Fan PWM Minimum".

Between "Lower Temperature Limit + Range" and "Higher Temperature Limit", the PWM duty cycle will increase linearly according to the temperature, as shown in the figure below. The PWM duty cycle will be 100% at "Higher Temperature Limit".



Chapter 5

Installing the iManager API

The iManager API is easy to install. You don't need to run the setup program. Supported operating systems are:

- Microsoft® Windows NT/2000/XP/XP Embedded
- Microsoft® Windows CE
- Linux
- QNX®
- VxWorks

5.1 Microsoft Windows 2000/ XP/ XP Embedded

To use the iManager API, just copy the following files to your application folder. There is no need to do an installation.

The files required are:

1.	Susi.dll	iManager's external export Library (API)
2.	SusiCore.dll	iManager's internal Library
3.	SusiCore.sys	iManager's Driver

5.2 Microsoft WindowsCE

Windows CE 5.0: Double-click the iManager installation file and add it from catalog items.

Windows CE 6.0: Extract the contents of the archive, iManager, to $(WINCEROOT)\public\ and add it from catalog items.

5.3 Linux

Extract the contents of the archive linux_susi.tar.gz to a driver folder on the Linux target and run the shell file: susi_install.sh. On the Linux target, the kernel version and distribution should match the Advantech release. Refer to the readme file for a detailed description of how to setup the driver.

5.4 QNX

Extract the contents of the archive: qnx_susi.tar.gz to the driver folder (/lib/dll) on a QNX target. Refer to the readme file for a detailed description of how to setup & use the driver.

5.5 WindRiver VxWorks

The API for VxWorks is provided by request. For more information contact our technical support department

Chapter 6

Programming Overview

The SUSI API functions are based on a dynamic library, so they can be installed and used at run time.

Header Files

- SUSI.H includes the API declaration, constants and flags that are required for programming.
- DEBUG.H / ERRDRV.H / ERRLIB.H are for debug code definitions.

  DEBUG.H - Function index codes

  ERRLIB.H - Library error codes

  ERRDRV.H - Driver error codes

Library Files

- Susi.dll is a dynamic link library that exports all the API functions.
- SusiCore.dll is a dynamic link library that talks to the driver.

Demo Program

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

Drivers

- SusiCore.sys is the driver that controls the hardware.

Initialize the DLL function

- Before using the API functions, make a call to SusiDllInitialize to initialize the library first, then call SusiDllInstall(1) to dynamically load the driver.
- After using the API functions, first make a call to SusiDllInstall(0) to dynamically unload the driver, and then call SusiDllInitialize to uninitialize the library.

6.1 Generic Board information

The iManager has the capability to keep account of all information about your platform. Use the function SusiBoardGetInfo with the SUSIBOARDINFO structure to easily get the data.

SusiBoardGetInfoA: ASCII code version.

SusiBoardGetInfoW: Unicode version.

SUSIBOARDINFO

unsigned long dwSize

size of this structure itself. please use sizeof(This strucrure name) to set this value.

unsigned long dwFlags

reserved. Always set to 0.

char szReserved[SUSI_BOARD_MAX_SIZE_ID_STRING]

reserved. Always set to 0.

char szBoard[SUSI_BOARD_MAX_SIZE_ID_STRING]
name of Platform.

char szBoardSub[SUSI_BOARD_MAX_SIZE_ID_STRING]
sub name of Platform, extracted from the manufacturing data

char szManufacturer[SUSI_BOARD_MAX_SIZE_ID_STRING]
name of the board manufacturer, usually ADVANTECH.

SUSITIME stManufacturingDate
date of manufacturing

SUSITIME stLastRepairDate
date of last repair

char szSerialNumber[SUSI_BOARD_MAX_SIZE_SERIAL_STRING]
serial number of platform, e.g. 000000000020

unsigned short wBoardRevision
board revision in ASCII notation, major revision in high-byte,
minor revision in low-byte, e.g. 0x4130 for revision A.0

unsigned short wBiosRevision
BIOS revision, major revision in high-byte,
minor revision in low-byte, e.g. 0x0110 for revision 1.10

unsigned short wOemBiosRevision
OEM BIOS revision

unsigned short wFirmwareRevision
firmware revision in ASCII notation, major revision in high-byte,
minor revision in low-byte, e.g. 0x0110 for revision 1.10

unsigned long dwClasses
represents all function classes supported in iManager.

unsigned long dwPrimaryClass
represents primary function class supported in iManager.

unsigned long dwRepairCounter
repair counter

char szPartNumber[SUSI_BOARD_MAX_SIZE_PART_STRING]
part number

char szEAN[SUSI_BOARD_MAX_SIZE_EAN_STRING]
EAN code of the platform

unsigned long dwReserved
sub manufacturer of the platform

6.2 Watchdog (WDog) Functions Class

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

There are multiple stage hardware watchdogs in iManager. Use the config function SusiWDogSetConfigStruct to set each unit's working rule.

The iManager API provides the following functions, which are used to control the behavior or to get information about the state of the watchdog:

- SusiWDogCount
- SusiWDogIsAvailable
- SusiWDogTrigger
- SusiWDogGetConfigStruct
- SusiWDogSetConfigStruct
- SusiWDogSetConfig
- SusiWDogDisable
- SusiWDogGetInfo
- SusiWDogSetIntCallBack

Mode

The mode defines the major behavior of the watchdog:

- SUSI_WDOG_MODE_REBOOT_PC:
  Run software reboot when watchdog happens.
- SUSI_WDOG_MODE_RESTART_OS:
  Trigger power button to run standard shutdown when a watchdog timeout occurs.
- SUSI_WDOG_MODE_STAGED:
  Set this mode to use staged mode watchdog.

Operatiing Modes

In staged mode, the watchdog might offer one or more various operating modes:

- SUSI_WDOG_OPMODE_DISABLED:
  Disable this stage.
- SUSI_WDOG_OPMODE_SINGLE_EVENT:
  Enable this stage, and send single event when a watchdog timeout occurs.

Events

An event is implemented by the onboard hardware during a situation when a watchdog timeout occurs. The following events are defined:

- SUSI_WDOG_EVENT_INT:
  defines a IRQ event.
  This event can cooperate SusiWDogSetIntCallBack to insert the call back function which will auto execute when a Watchdog timeout occurs.
- SUSI_WDOG_EVENT_SCI:
  defines a SCI event.
- SUSI_WDOG_EVENT_RST:
  defines a system reset event.
- SUSI_WDOG_EVENT_RST:
  defines a power button event.

Watchdog Types

The following watchdog types are currently defined:

- SUSI_WDOG_TYPE_UNKNOWN:
  used when the type is not known.
- SUSI_WDOG_TYPE_BC:
  the watchdog is implemented via the ADVANTECH onboard controller.
- SUSI_WDOG_TYPE_CHIPSET:
  watchdog functionality is available only through the board's chipset.

Information Structure

The SusiWDogGetInfo function call is used to get information about the current configuration and state of the watchdog. It takes a pointer to an instance of structure SUSIWDINFO, which is defined as follows:

SUSIWDINFO

unsigned long dwSize

size of this structure itself. please use sizeof(This structure name) to set this value.

unsigned long dwFlags

reserved. Always set to 0.

unsigned long dwMinTimeout

value depends on the hardware implementation of the watchdog and specifies the minimum value for the watchdog trigger timeout.

unsigned long dwMaxTimeout

value depends on the hardware implementation of the watchdog and specifies the maximum value for the watchdog trigger timeout.

unsigned long dwMinDelay

value depends on the hardware implementation of the watchdog and specifies the minimum value for the watchdog enable delay.

unsigned long dwMaxDelay

value depends on the hardware implementation of the Watchdog and specifies the maximum value for the Watchdog enable delay.

unsigned long dwOpModes

mask of the supported operating modes, see section: Operating Modes.

unsigned long dwMaxStageCount

amount of supported watchdog stages.

unsigned long dwEvents

mask of supported watchdog events, see section: Events.

unsigned long dwType

see section: Watchdog Types.

Configuration

The SusiWDogSetConfigStruct and SusiWDogGetConfigStruct function calls are used to set and to determine the Watchdog configuration. Both of them take a pointer to an instance of structure SUSIWDCONFIG which is defined as follows:

SUSIWDCONFIG

unsigned long dwSize

size of this structure itself. please use sizeof(This strucrure name) to set this value.

unsigned long dwTimeout

specifies the value for the watchdog timeout. It must be in the range, SUSIWDINFO: dwMinTimeout, and SUSIWDINFO: dwMaxTimeout. In case of multiple stages, this value is not used because the configuration occurs through the appropriate stage structure.

unsigned long dwDelay

value specifies the value for the watchdog enable delay

unsigned long dwMode

current mode, see section: Mode

--Optional parameters for staged watchdog--

unsigned long dwOpMode

mask of the supported operating modes, see section: Operating Modes

value is only used in multistage mode

unsigned long dwStageCount

number of available watchdog stages.

value is only used in multistage mode

SUSIWDSTAGE stStages[SUSI_WDOG_EVENT_MAX_STAGES]

array holds the state definition of each defined stage

values are only used in multistage mode

The SusiWDogSetConfig and the config structure contain time values with a millisecond resolution. Timeout is the basic time during which a SusiWDogTrigger function must be called. Delay adds an initial time period for the first trigger call.

SUSIWDSTAGE

unsigned long dwTimeout

specifies the time value for the affected stage. The value must be in the range SUSIWDINFO:dwMinTimeout and SUSIWDINFO:dwMaxTimeout.

unsigned long dwEvent

contains the event definition for the affected stage, see section Events.

Triggering

After configuring the watchdog using SusiWDogSetConfigStruct, the application must continuously call SusiWDogTrigger to trigger the watchdog.

Disabling the Watchdog

An enabled watchdog can be disabled by calling SusiWDogDisable.

6.3 GPIO (I/O) functions

Use iManager API to set IO direction and IO status (Hi | Low).

- SusiIOCount
- SusiIOIsAvailable
- SusiIORead
- SusiIOWrite
- SusiIOGetDirection
- SusiIOSetDirection
- SusiIOGetDirectionCaps

6.4 SMBus Functions

SMBus 2.0 compliant protocols are supported in SusiSMBus- APIs :

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

An additional API for probing is also supported:

- 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.

All APIs except SusiSMBusScanDevice use the parameter SlaveAddress as an 8-bit address; users don't need to be concerned about giving it as a read or write address, since the actual R/W is taken care by the API itself. As an example, using a write address of 0x41 for the APIs with a write operation or not using it, the correct result would be still be obtained in either case.

SusiSMBusScanDevice is used to probe whether an address is currently used by certain devices on a platform. The addresses which are occupied can be determined by scanning from 0x00 to 0x7f. An example of usage would be scanning for occupied addresses and avoiding them when connecting a new device; or probing before and after connecting a new device, to quickly discover its address. The SlaveAddress_7 parameter given in this API is a 7-bit address.

- SusiSMBusScanDevice
- SusiSMBusReadQuick
- SusiSMBusWriteQuick
- SusiSMBusReceiveByte
- SusiSMBusSendByte
- SusiSMBusReadByte
- SusiSMBusWriteByte
- SusiSMBusReadWord
- SusiSMBusWriteWord

6.5 IIC Functions

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

- SusiI2CCount
- SusiI2CType
- SusiI2CIsAvailable
- SusiI2CRead
- SusiI2CWrite
- SusiI2CReadRegister
- SusiI2CWriteRegister
- SusiI2CWriteReadCombined
- SusiI2CGetMaxFrequency
- SusiI2CGetFrequency
- SusiI2CSetFrequency

6.6 VGA Control (VC) Functions

SusiVC- functions support LCD brightness adjustment.

- SusiVgaCount
- SusiVgaGetBacklight
- SusiVgaSetBacklight
- SusiVgaGetInfo

6.7 Hardware Monitoring Functions

The SUSI interface provides access to hardware monitoring functions such as voltage sensor, temperature sensor and fan control.

The function calls “SusiVoltageGetCount”, “SusiTemperatureGetCount” and “SusiFanGetCount” are used to determine the number of attached sensors per type.

The function calls “SusiVoltageGetInfo”, “SusiTemperatureGetInfo” and “SusiFanGetInfo” are used to determine the state and the configuration of an attached sensor.

The function calls “SusiVoltageGetCurrent”, “SusiTemperatureGetCurrent” and “SusiFanGetCurrent” are used to determine the actual measured value of an attached sensor.

Sensor Status Flags

The sensor status flags (unsigned long dwFlags), which are defined in the SUSI*INFO structure, represent the capabilities of the related sensor. The status flags can be determined using a “Susi*GetInfo” function call. The following sensor status flags are defined:

- SUSI_SENSOR_ACTIVE:
  sensor is active and usable
- SUSI_SENSOR_ALARM:
  sensor supports alarm indication
- SUSI_SENSOR_BROKEN:
  no physical sensor is attached
- SUSI_SENSOR_SHORTCIRCUIT:
  sensor has a short circuit

Temperature Sensor Types

The following types of temperature sensors are defined and are dependent on their location within the system:

- SUSI_TEMP_CPU:
  sensor which measures CPU temperature
- SUSI_TEMP_ENV:
  sensor which measures the temperature of the system environment
- SUSI_TEMP_BOARD:
  sensor which measures board temperature
- SUSI_TEMP_BACKPLANE:
  sensor which measures temperature on the backplane
- SUSI_TEMP_CHIPSETS:
  sensor which measures temperature of the chipset
- SUSI_TEMP_VIDEO:
  sensor which measures temperature of the video chip
- SUSI_TEMP_TOPDIMM_ENV:
  sensor which measures temperature of the DRAM module on the topside of the CPU module
- SUSI_TEMP_BOTDIMM_ENV:
  sensor which measures temperature of the DRAM module on the bottom side of the CPU module
- SUSI_TEMP_OTHER:

all other temperature sensors found within the system

Temperature Information Structure

The “SusiTemperatureGetInfo” function call is used to get information about the current configuration and state of the temperature sensor. It takes a pointer to an instance of structure SUSITEMPERARUREINFO, which is defined as follows:

SUSITEMPERATUREINFO

unsigned long dwSize

size of the structure itself, must be initialized with sizeof(SUSITEMPERATUREINFO)

unsigned long dwType

see section: Temperature Sensor Types

unsigned long dwRes

this value defines the granularity of the temperature sensor

unsigned long dwMin

this is the minimum value that can be measured by the sensor

unsigned long dwMax

this is the maximum value that can be measured by the sensor

All temperature values are in units of 1/1000th degree centigrade.

Fan Sensor Types

The following types of fan sensors are defined and are dependent on their location within the system:

- SUSI_FAN_CPU:
  sensor which represents the CPU fan
- SUSI_FAN_BOX:
  sensor which represents the fan on the chassis
- SUSI_FAN_CHIPSET:
  sensor which represents the fan on the chipset
- SUSI_FAN_VIDEO:
  sensor which represents the fan on the video chip
- SUSI_FAN_OTHER:
  all other fan sensors found within the system

Fan Information Structure

The “SusiFanGetInfo” function call is used to get information about the current configuration and state of the fan control. It takes a pointer to an instance of structure SUSIFANINFO, which is defined as follows:

SUSIFANINFO

unsigned long dwSize

size of the structure itself, must be initialized with sizeof(SUSIFANINFO)

unsigned long dwType

see section: Fan Sensor Types

unsigned long dwSpeedNom

this value defines the nominal speed of the fan.

If the value is -1 then the nominal speed is not supported or known

unsigned long dwMin

this is the minimum speed of the fan

unsigned long dwMax

this is the maximum speed of the fan

All fan speed values are in RPM (revolutions per minute).

Fan Speed Control

The “SusiFanSetConfigStruct” function call is used to set fan speed configuration. You can use this function to control the fan speed easily as you want. It takes a pointer to an instance of structure SUSIFANCONFIG, which is defined as follows:

SUSIFANCONFIG

unsigned long dwSize

size of the structure itself, must be initialized with sizeof(SUSIFANCONFIG)

unsigned long dwMode

mask of the supported fan modes.

unsigned long dwPWM

pwm speed value, only for SUSI_FAN_MODE_MANUAL mode.

SUSIAUTOFANCONFIG safConfig

pwm automatic algorithm, only for SUSI_FAN_MODE_AUTO mode.

Some modes as shown below can be used in dwMode:

- SUSI_FAN_MODE_OFF:

disable the fan speed function.

- SUSI_FAN_MODE_FULL:
  set fan to full speed.
- SUSI_FAN_MODE_MANUAL:
  set fan speed manually.
- SUSI_FAN_MODE_AUTO:
  use auto fan mode to control the fan speed.

Auto Fan Speed Mode

When you use the auto fan mode, you have to set “SUSIFANCONFIG:dwMode” in the “SUSI_FAN_MODE_AUTO” then set “SUSIFANCONFIG:safConfig”. “SUSIAUTOFANCONFIG” is defined as follows:

SUSIAUTOFANCONFIG

unsigned long dwZone
temperature sensor you want to refer to

unsigned long dwOpMode
set fan speed module as PWM or RPM

unsigned long dwLowStopTemp
when the temperature drop to the value, the fan will stop.

unsigned long dwLowTemp
when the temperature rises to the value, the fan will work in dwLow* speed.

unsigned long dwHighTemp
when the temperature rises to the value, the fan will work in dwHigh* speed.

unsigned long dwLowPWM
fan speed in low status using the PWM module

unsigned long dwHighPWM
fan speed in high status using the PWM module

unsigned long dwLowRPM
fan speed in low status using the RPM module

unsigned long dwHighRPM
fan speed in high status using the RPM module

Voltage Sensor Types

The following types of voltage sensors are defined and are dependent on their location within the system:

- SUSI_VOLTAGE_BAT_CMOS:
  sensor that measures the CMOS battery
- SUSI_VOLTAGE_BAT_POWER:
  sensor that measures the battery voltage in a mobile system
- SUSI_VOLTAGE_5V_S0:
  sensor that measures the 5V input voltage
- SUSI_VOLTAGE_5V_S5:
  sensor that measures the 5V standby voltage
- SUSI_VOLTAGE_33V_S0:
  sensor that measures the 3.3V onboard voltage
- SUSI_VOLTAGE_33V_S5:
  sensor that measures the 3.3V standby voltage
- SUSI_VOLTAGE_12V_S0:
  sensor that measures the 12V onboard voltage
- SUSI_VOLTAGE_VCOREA:
  sensor that measures the first core voltage (often used as CPU voltage)
- SUSI_VOLTAGE_VCOREB:
  sensor that measures the second core voltage (often used as memory and chipset voltage)
- SUSI_VOLTAGE_DC:
  any sensor that measures an onboard voltage that can't be covered by the previous definitions
- SUSI_VOLTAGE_DC_STANDBY:
  any sensor that measures a standby voltage that can't be covered by the previous definitions
- SUSI_VOLTAGE_OTHER:
  specified if none of the above can be applied

Voltage Information Structure

The “SusiVoltageGetInfo” function call is used to get information about the current configuration and state of the voltage control. It takes a pointer to an instance of structure SUSIVOLTAGEINFO, which is defined as follows:

SUSIVOLTAGEINFO

unsigned long dwSize

size of the structure itself, must be initialized with sizeof(SUSIVOLTAGEINFO)

unsigned long dwType

see section: Voltage Sensor Types

unsigned long dwNom

this value defines the nominal voltage of the sensor.

If the value is -1 then the nominal voltage is not supported or known

unsigned long dwRes

this value defines the granularity of the voltage sensor

unsigned long dwMin

this is the minimum value that can be determined by the sensor

unsigned long dwMax

this is the maximum value that can be determined by the sensor

All of the above mentioned voltage values are in units of 1/1000th volt.

Storage Area Functions

Each board is usually equipped with a number of different storage areas. They may be located in Flash, EEPROM, CMOS RAM, etc. A storage area is defined as a portion of physical memory that can provide constant storage for the user's application. Every SusiStorageArea* function call takes a type or a unit number as a second parameter, which identifies the affected area.

6.8 Storage Area Types

The storage areas are distinguished depending on their location in memory:

- SUSI_STORAGE_AREA_EEPROM:
  provides access to the user eeprom
- SUSI_STORAGE_AREA_FLASH:
  provides access to the flash
- SUSI_STORAGE_AREA_CMOS:
  provides access to the CMOS
- SUSI_STORAGE_AREA_RAM:
  provides access to the user RAM
- SUSI_STORAGE_AREA_UNKNOWN:
  this type is used to determine all installed areas (not just a certain type) during a SusiStorageAreaCount call

During any “SusiStorageArea*” function call, the pure type is located in the high word and the enumerated unit number within that pure type (if more units of the same type exist) is located in the low word of parameter dwUnit.

Chapter 7

SUSI API Programmer's Documentation

The iManager API provides access to ADVANTECH specific board information and features.

Return Values

Unless they return a count or version number, all SUSI* functions return 1 for success and 0 for failure. Other return values are stored in pointers passed to the function.

Information Structures

The API defines several information structures in susi.h They are used to store the returned values during Susi*GetInfo calls. Before using these structures, the dwSize entry of each info structure must be initialized with the size of the structure itself (sizeof(SUSI*INFO)). This provides independence between the application and the library if the structure is extended in future releases of the library.

Unit Numbers

Almost all function calls take a unique unit number that is used to identify a dedicated unit. Usually the unit number is between 0 and the return value -1 of the related Susi*Count function call. It can be taken as an index for devices of the same type.

7.1 SusiDllUninitialize

Uninitialize the iManager API Library.

SUSIRET_BOOL SusiDllUninitialize(void);

Parameters

None.

Return Value

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

Remarks

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

7.2 SusiDllIsAvailable

Checks if the iManager API library has already been initialized.

SUSIRET_BOOL SusiDllIsAvailable(void);

Parameters

Return Value

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

Remarks

Checks if the iManager API library has already been initialized by a prior call to function SusiDllInitialize.

7.3 SusiDllInstall

Retrieve the version numbers of iManager Library.

SUSIRET_BOOL SusiDllInstall(int install);

Parameters

install

[in] 1 - installs the low level SUSI driver

0 - removes the low level SUSI driver

Return Value

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

Remarks

This function can be used to install the low level iManager driver if a prior call of SusiLibInitialize failed.

Keep in mind that you might need administrative privileges for executing this function successfully.

7.4 SusiDllGetDrvVersion

Retrieve the version numbers of iManager low level driver.

SUSIRET_ULONG SusiDllGetDrvVersion(void);

Parameters

Return Value

version of the low level SUSI driver.

Remarks

7.5 SusiDllGetLastError

This function returns the last error code value.

SUSIRET_ULONG SusiDllGetLastError(void);

Parameters

None

Return Value

description	error code
generic error -1	(0xFFFF FFFF)
invalid parameter -2	(0xFFFF FFFE)
function not found -3	(0xFFFF FFFD)
read error -4	(0xFFFF FFFC)
write error -5	(0xFFFF FFFB)
timeout -6	(0xFFFF FFFA)

Remarks

Returns the last known error code of the low level iManager driver. Notice that this function really delivers the code of the last known iManager driver error and not the result of the last iManager API function call. A succeeding iManager API call doesn't affect the return value of this function.

7.6 SusiDllInstall

Set the last error code's buffer location.

SUSIRET_BOOL SusiDllSetLastErrorAddress(unsigned long *pErrNo);

Parameters

pErrNo buffer where the error code will be stored

Return Value

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

Remarks

With this function it's possible to specify a local memory location in the context of the application where the last error code will be stored. It provides a convenient way of implementing error handling without calling the SusiLibGetLastError function after each regular iManager API function call.

7.7 SusiBoardCount

Check number of installed iManager compliant boards.

SUSIRET_ULONG SusiBoardCount(unsigned long dwClass, unsigned long dwFlags);

Parameters

dwClass	the hardware class of the board.
dwFlags	either SUSI_BOARD_OPEN_FLAGS_DEFAULT or SUSI_BOARD_OPEN_FLAGS_PRIMARYONLY
	SUSI_BOARD_OPEN_FLAGS_DEFAULT counts all boards of the given hardware class
	SUSI_BOARD_OPEN_FLAGS_PRIMARYONLY counts only boards which primary board class matches the given hardware class

Return Value

the number of installed iManager compliant boards with the specified board class dwClass. In case of dwClass is 0, the total number of boards in the system will be returned.

Remarks

7.8 SusiBoardOpen

Open the iManager compliant board and get handle

SUSIRET_BOOL SusiBoardOpen(unsigned long dwClass, unsigned long dwNum, unsigned long dwFlags, HSUSI *phSusi);

Parameters

dwClass — the hardware class of the board, see also 4.2 subsection: "Board classes"

dwNum — the subsequent number of the selected board in it's class, starting from 0

dwFlags — either SUSI_BOARD_OPEN_FLAGS_DEFAULT or SUSI_BOARD_OPEN_FLAGS_PRIMARYONLY

SUSI_BOARD_OPEN_FLAGS_DEFAULT
scans for all boards of the specified hardware class,
regardless if it's the primary class or the secondary class

SUSI_BOARD_OPEN_FLAGS_PRIMARYONLY
scans for boards which primary board class
matches the specified hardware class

phSusi — buffer where the board handle will be stored

Return Value

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

Remarks

Each iManager compliant board in the system will be addressed by its own unique board handle. This function is used to open such a board and to obtain a valid board handle. If there is more then one iManager board in the system, each board can be individually selected by its board class dwClass and a subsequent enumeration of dwNum. On success, the function returns the board handle in * phSusi.

SUSI_BOARD_OPEN_FLAGS_PRIMARYONLY might be used for dwFlags to select a board of a dedicated board class. Together with an enumerated counter starting from 0 the board can be addressed exactly.

7.9 SusiBoardOpenByNameA

Open the iManager compliant board and get handle by the name, ASCII code Ver.

SUSIRET_BOOL SusiBoardOpenByNameA(const char *pszName, HSUSI *phSusi);

Parameters

pszName	the name of the board.
phSusi	buffer where the board handle will be stored

Return Value

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

Remarks

This function behaves like SusiBoardOpen except that the board is specified by its name. On success, the function returns the board handle in *phSusi.

7.10 SusiBoardOpenByNameW

Open the iManager compliant board and get handle by the name, Unicode Ver.

SUSIRET_BOOL SusiBoardOpenByNameW(const wchar_t *pszName, HSUSI *phSusi);

Parameters

pszName	the name of the board.
phSusi	buffer where the board handle will be stored

Return Value

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

Remarks

This function behaves like SusiBoardOpen except that the board is specified by its name. On success, the function returns the board handle in *phSusi.

7.11 SusiBoardClose

Close the iManager compliant board after using.

SUSIRET_BOOL SusiBoardClose(HSUSI hSusi);

Parameters

hSusi the board handle

Return Value

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

Remarks

Closes a board which was previously opened by either SusiBoardOpen or SusiBoardOpenByName*.

7.12 SusiBoardGetNameA

Get platform name, ASCII code Ver.

SUSIRET_BOOL SusiBoardGetNameA(HSUSI hSusi, char *pszName, unsigned long dwSize);

Parameters

hSusi	the board handle
pszName	buffer where the board name will be stored
dwSize	size of the buffer in bytes, should be at least SUSI_BOARD_MAX_SIZE_ID_STRING

Return Value

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

Remarks

Determines the name of the board addressed by hSusi.

7.13 SusiBoardGetNameW

Get platform name, Unicode Ver.

SUSIRET_BOOL SusiBoardGetNameW(HSUSI hSusi, wchar_t *pszName, unsigned long dwSize);

Parameters

hSusi	the board handle
pszName	buffer where the board name will be stored
dwSize	size of the buffer in bytes, should be at least SUSI_BOARD_MAX_SIZE_ID_STRING

Return Value

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

Remarks

Determines the name of the board addressed by hSusi.

7.14 SusiBoardGetInfoA

Get platform informateion, ASCII code Ver.

SUSIRET_BOOL SusiBoardGetInfoA(HSUSI hSusi, SUSIBOARDINFOA *pBoardInfo);

Parameters

hSusi	the board handle
pBoardInfo	the buffer where the board information will be stored

Return Value

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

Remarks

Gets the board information of a iManager API compliant board addressed by hSusi.

7.15 SusiBoardGetInfoW

Get platform information, Unicode Version

SUSIRET_BOOL SusiBoardGetInfoW(HSUSI hSusi, SUSIBOARDINFOW *pBoardInfo);

Parameters

hSusi	the board handle
pBoardInfo	the buffer where the board information will be stored

Return Value

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

Remarks

Gets the board information of a iManager API compliant board addressed by hSusi.

7.16 SusiBoardGetBootCounter

Gets the current value of the boot counter.

SUSIRET_BOOL SusiBoardGetBootCounter(HSUSI hSusi, unsigned long *pdw-Count);

Parameters

hSusi	the board handle
pdwCount	the variable where the boot counter value will be stored

Return Value

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

Remarks

7.17 SusiBoardGetRunningTimeMeter

Gets the current running time of the board measured in hours.

SUSIRET_BOOL SusiBoardGetRunningTimeMeter(HSUSI hSusi, unsigned long *pdwCount);

Parameters

hSusi	the board handle
pdwCount	the variable where the value of the running time meter will be stored

Return Value

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

Remarks

7.18 SusiWDogCount

Check number of watchdogs function on the platform.

SUSIRET_ULONG SusiWDogCount(HSUSI hSusi);

Parameters

hSusi	the board handle

Return Value

the number of installed Watchdogs in the system.

Remarks

7.19 SusiWDogIsAvailable

Check that the watchdog function unit is workable.

SUSIRET_BOOL SusiWDogIsAvailable(HSUSI hSusi, unsigned long dwUnit);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control

Return Value

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

Remarks

7.20 SusiWDogTrigger

Triggers the watchdog; it doesn't timeout.

SUSIRET_BOOL SusiWDogTrigger(HSUSI hSusi, unsigned long dwUnit);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control

Return Value

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

Remarks

7.21 SusiWDogGetConfigStruct

Get the configuration which is the watchdog working rule on the platform.

SUSIRET_BOOL SusiWDogGetConfigStruct(HSUSI hSusi, unsigned long dwUnit, SUSIWDCONFIG *pConfig);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
pConfig	the pointer to the configuration structure

Return Value

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

Remarks

7.22 SusiWDogSetConfigStruct

Set the configuration which is the watchdog working rule on the platform.

SUSIRET_BOOL SusiWDogSetConfigStruct(HSUSI hSusi, unsigned long dwUnit, SUSIWDCONFIG *pConfig);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
pConfig	the pointer to the configuration structure

Return Value

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

Remarks

7.23 SusiWDogSetConfig

Set the single watchdog working rule on the platform.

SUSIRET_BOOL SusiWDogSetConfig(HSUSI hSusi, unsigned long dwUnit, unsigned long timeout, unsigned long delay, unsigned long mode);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
pConfig	the pointer to the configuration structure
timeout	the value in milliseconds before the watchdog times out. An application which is observed by the watchdog must call SusiWDogTrigger within the specified time.
delay	the delay before the watchdog starts working. This is required to prevent a reboot while the operating system or the application initializes.

Return Value

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

Remarks

Sets the configuration of the watchdog. WhileSusiWDogSetConfigStruct takes a complete structure, SusiWDogSetConfig takes single values. Use SusiWDogSetConfigStruct to benefit from the advantages of a staged watchdog.

7.24 SusiWDogDisable

Disable the watchdog function.

SUSIRET_BOOL SusiWDogDisable(HSUSI hSusi, unsigned long dwUnit);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control

Return Value

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

Remarks

7.25 SusiWDogGetInfo

Gets the information structure of the watchdog.

SUSIRET_BOOL SusiWDogGetInfo(HSUSI hSusi, unsigned long dwUnit, SUSIWDINFO *pInfo);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
pInfo	pointer to the Watchdog information structure

Return Value

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

Remarks

7.26 SusiWDogSetIntCallBack

Register the interrupt active function.

SUSIRET_BOOL SusiWDogSetIntCallBack(HSUSI hSusi, unsigned long dwUnit, SUSI_WDOG_CALLBACK_EVENT_INT *fnCallBack);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
fnCallBack	pointer of call back function which will be called when interrupt happened.

Return Value

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

Remarks

Before using this function, the SUSIWDCONFIG:dwOpMode must be set to SUSI_WDOG_EVENT_INT.

7.27 SusiIOCount

Check number of IO function on the platform.

SUSIRET_ULONG SusiIOCount(HSUSI hSusi);

Parameters

hSusi the board handle

Return Value

the number of installed I/Os in the system.

Remarks

7.28 SusiIOIsAvailable

Check I/O function unit is workable.

SUSIRET_BOOL SusiIOIsAvailable(HSUSI hSusi, unsigned long dwUnit);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control

Return Value

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

Remarks

7.29 SusiIORead

Read the I/O pin's state.

SUSIRET_BOOL SusiIORead(HSUSI hSusi, unsigned long dwUnit, unsigned long *pdwData);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
pdwData	the pointer to the destination buffer

Return Value

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

Remarks

Reads the value of the input pins of IO unit dwUnit. It's recommended to combine this value with the result of SusiIOGetDirectionCaps.

7.30 SusiIOWrite

Write the I/O pin's state.

SUSIRET_BOOL SusiIOWrite(HSUSI hSusi, unsigned long dwUnit, unsigned long dwData);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
dwData	the data to write

Return Value

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

Remarks

Writes the value dwData to the output pins of I/O unit dwUnit. It's recommended to combine this value with the result of SusiIOGetDirectionCaps.

7.31 SusiIOGetDirectionCaps

Get I/O direction's capability.

SUSIRET_BOOL SusiIOGetDirectionCaps(HSUSI hSusi, unsigned long dwUnit, unsigned long *pdwInputs, unsigned long *pdwOutputs);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
pdwInputs	the pointer to the destination buffer of the input capabilities
pdwOutputs	the pointer to the destination buffer of the output capabilities

Return Value

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

Remarks

Determines the input and the output capabilities of the I/O unit dwUnit. Each GPI/GPO/GPIO is represented by a bit in the variables pdwInputs and pdwOutputs. If the pin has input capabilities, the respective pin in pdwInputs is set to 1. If the pin has output capabilities, the respective pin in pdwOutputs is set to 1. If the pin has input and output capabilities, both respective bits in pdwInputs and pdwOutputs are set to 1. In this case, the data direction (if input or output) may be controlled by the SusiIOSetDirection function call.

7.32 SusiIOGetDirection

Get the I/O pin's direction.

SUSIRET_BOOL SusiIOGetDirection(HSUSI hSusi, unsigned long dwUnit, unsigned long *pdwData);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
pdwData	the pointer to the destination buffer of the direction information

Return Value

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

Remarks

Determines the current data direction of the respective GPI/GPO/GPIO pin. A bit set to 1 in this field indicates that the respective pin is configured as an input, a bit set to 0 indicates that the respective pin is configured as an output. Notice that the binary values for pins that are not implemented are unspecified and can be 0 or 1. Therefore, it's recommended to cross check the result of SusiIOGetDirection with the result of SusiIOGetDirectionCaps.

7.33 SusiIOSetDirection

Set the I/O pin's direction.

SUSIRET_BOOL SusiIOSetDirection(HSUSI hSusi, unsigned long dwUnit, unsigned long dwData);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
dwData	the direction information

Return Value

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

Remarks

Sets the current data direction of the respective GPI/GPO/GPIO pin. A bit set to 1 in this field indicates that the related pin is configured to be an input, a bit set to 0 indicates that the related pin is configured to be an output. Notice that the binary values for pins that are not implemented are unspecified and should be written as 0.

7.34 SusiSMBusScanDevice

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

SUSIRET_INT SusiSMBusScanDevice(HSUSI hSusi, unsigned char bAddr_7);

Parameters

hSusi	the board handle
SlaveAddress	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 many as 128 devices connected to a single SMBus. For more information about how to use this API, please refer to the "Programming Overview", section "SMBus functions".

7.35 SusiSMBusReadQuick

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

SUSIRET_BOOL SusiSMBusReadQuick(HSUSI hSusi, unsigned char bAddr);

Parameters

hSusi	the board handle
SlaveAddress	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 the "Programming Overview", section "SMBus functions".

7.36 SusiSMBusWriteQuick

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

SUSIRET_BOOL SusiSMBusWriteQuick(HSUSI hSusi, unsigned char bAddr);

Parameters

hSusi the board handle

SlaveAddress 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 the "Programming Overview", section "SMBus functions".

7.37 SusiSMBusReceiveByte

Receive information in bytes from the target slave device in the SMBus.

SUSIRET_BOOL SusiSMBusReceiveByte(HSUSI hSusi, unsigned char bAddr, unsigned char *pDataByte);

Parameters

hSusi the board handle

SlaveAddress 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 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 the "Programming Overview", section "SMBus functions".

7.38 SusiSMBusSendByte

Send information in bytes to the target slave device in the SMBus.

SUSIRET_BOOL SusiSMBusSendByte(HSUSI hSusi, unsigned char bAddr, unsigned char bData);

Parameters

hSusi	the board handle
SlaveAddress	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	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 the "Programming Overview", section "SMBus functions".

7.39 SusiSMBusReadByte

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

SUSIRET_BOOL SusiSMBusReadByte(HSUSI hSusi, unsigned char bAddr, unsigned char bReg, unsigned char *pDataByte);

Parameters

hSusi	the board handle
SlaveAddress	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	Specifies the offset of the device register to read data from.
Result	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 the "Programming Overview", section "SMBus functions".

7.40 SusiSMBusWriteByte

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

SUSIRET_BOOL SusiSMBusWriteByte(HSUSI hSusi, unsigned char bAddr, unsigned char bReg, unsigned char bData);

Parameters

hSusi	the board handle
SlaveAddress	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	Specifies the offset of the device register to read data from.
Result	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 the "Programming Overview", section "SMBus functions".

7.41 SusiSMBusReadWord

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

SUSIRET_BOOL SusiSMBusReadWord(HSUSI hSusi, unsigned char bAddr, unsigned char bReg, unsigned short *pDataWord);

Parameters

hSusi	the board handle
SlaveAddress	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	Specifies the offset of the device register to word data from.
Result	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 the "Programming Overview", section "SMBus functions".

7.42 SusiSMBusWriteWord

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

SUSIRET_BOOL SusiSMBusWriteWord(HSUSI hSusi, unsigned char bAddr, unsigned char bReg, unsigned short wData);

Parameters

hSusi	the board handle
SlaveAddress	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	Specifies the offset of the device register to word data from.
Result	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 the "Programming Overview", section "SMBus functions"

7.43 SusiI2CCount

Gets the number of installed I2C buses on the platform.

SUSIRET_ULONG SusiI2CCount(HSUSI hSusi);

Parameters

hSusi	the board handle

Return Value

the number of installed I2C buses in the system.

Remarks

7.44 SusiI2CType

Gets the type of the addressed I2C bus.

SUSIRET_ULONG SusiI2CType(HSUSI hSusi, unsigned long dwUnit);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control

Return Value

SUSI_I2C_TYPE_PRIMARY	the primary I2C bus
SUSI_I2C_TYPE_SMB	the system management bus
SUSI_I2C_TYPE_DDC	the I2C bus of the DDC interface
or	
SUSI_I2C_TYPE_UNKNOWN	for unknown or special purposes if the type is not known.

Remarks

7.45 SusiI2CIsAvailable

Check I2C function unit is workable.

SUSIRET_BOOL SusiI2CIsAvailable(HSUSI hSusi, unsigned long dwUnit);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control

Return Value

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

Remarks

7.46 SusiI2CRead

Read data from I2C device.

SUSIRET_BOOL SusiI2CRead(HSUSI hSusi, unsigned long dwUnit, unsigned char bAddr, unsigned char *pBytes, unsigned long dwLen);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
bAddr	the 8bit address of the affected device on the bus (bit 0 must be logical 1 to indicate a read operation)
pBytes	the pointer to the destination buffer
dwLen	the number of sequential bytes to read

Return Value

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

Remarks

Reads dwLen subsequent bytes from the device with address bAddr at I2C bus dwUnit to buffer pBytes.

7.47 SusiI2CWrite

Write data from I2C device.

SUSIRET_BOOL SusiI2CWrite(HSUSI hSusi, unsigned long dwUnit, unsigned char bAddr, unsigned char *pBytes, unsigned long dwLen);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
bAddr	the 8-bit address of the affected device on the bus (bit 0 must be logical 0 to indicate a write operation)
pBytes	the pointer to the destination buffer
dwLen	the number of sequential bytes to write

Return Value

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

Remarks

Writes dwLen subsequent bytes from the buffer pBytes to the device with address bAddr at I2C bus dwUnit.

7.48 SusiI2CReadRegister

Read data from I2C device register.

SUSIRET_BOOL SusiI2CReadRegister(HSUSI hSusi, unsigned long dwUnit, unsigned char bAddr, unsigned short wReg, unsigned char *pDataByte);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
bAddr	the 8bit address of the affected device on the bus (bit 0 must be logical 1 to indicate a read operation)
wReg	the number of the register to read
pDataByte	the pointer to the destination buffer

Return Value

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

Remarks

Reads one byte from the register wReg in the device with address bAddr at I2C bus dwUnit to buffer pDataByte.

7.49 SusiI2CWriteRegister

Write data from I2C device register.

SUSIRET_BOOL SusiI2CWriteRegister(HSUSI hSusi, unsigned long dwUnit, unsigned char bAddr, unsigned short wReg, unsigned char bData);

Parameters

hSusi the board handle

dwUnitthe unit number you want to control

bAddr the 8bit address of the affected device on the bus (bit 0 must be logical 0 to indicate a write operation)

wReg the number of the register to write to

bData the byte value to write

Return Value

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

Remarks

Writes the value of bData to the register wReg in the device with address bAddr at I2C bus dwUnit to buffer pDataByte.

7.50 SusiI2CWriteReadCombined

Combines writing to and reading from a device on the I2C bus in one step.

SUSIRET_BOOL SusiI2CWriteReadCombined(HSUSI hSusi, unsigned long dwUnit, unsigned char bAddr, unsigned char *pBytesWrite,

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
bAddr	the 8bit address of the affected device on the bus (bit 0 must be logical 0)
pBytesWrite	the pointer to the source buffer which contains the bytes to write
dwLenWrite	the amount of bytes to write
pBytesRead	the pointer to the destination buffer
dwLenRead	the amount of bytes to read

Return Value

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

Remarks

There will be no stop condition after writing to the device, the subsequent read cycle will be initiated with a leading start condition.

7.51 SusiI2CGetMaxFrequency

Gets the maximum operating frequency.

SUSIRET_BOOL SusiI2CGetMaxFrequency(HSUSI hSusi, unsigned long dwUnit, unsigned long *pdwSetting);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
pdwSetting	the variable where the maximum frequency setting will be stored

Return Value

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

Remarks

Gets the maximum operating frequency of the I2C bus specified by unit number dwUnit in Hz.

7.52 SusiI2CGetFrequency

Gets the current operating frequency.

SUSIRET_BOOL SusiI2CGetFrequency(HSUSI hSusi, unsigned long dwUnit, unsigned long *pdwSetting);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
pdwSetting	the variable where the current frequency setting will be stored

Return Value

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

Remarks

Gets the current operating frequency of the I2C bus specified by unit number dwUnit in Hz.

7.53 SusiI2CGetMaxFrequency

Sets the maximum operating frequency.

SUSIRET_BOOL SusiI2CSetFrequency(HSUSI hSusi, unsigned long dwUnit, unsigned long dwSetting);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
pdwSetting	the frequency setting in Hz

Return Value

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

Remarks

Sets the current operating frequency of the I2C bus specified by unit number dwUnit in Hz. Commonly used values are 100000 and 400000.

7.54 SusiVgaCount

Check number of VGA function on the platform.

SUSIRET_ULONG SusiVgaCount(HSUSI hSusi);

Parameters

hSusi the board handle

Return Value

the number of installed VGA in the system.

Remarks

7.55 SusiVgaGetBacklight

Gets the backlight brightness value.

SUSIRET_BOOL SusiVgaGetBacklight(HSUSI hSusi, unsigned long dwUnit, unsigned long *pdwSetting);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
pdwSetting	the variable where the backlight brigthness will be stored

Return Value

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

Remarks

The range of the value is between 0 and SUSI_VGA_BACKLIGHT_MAX (100), respectively 0 and 100%.

7.56 SusiVgaSetBacklight

Sets the backlight brigthness value.

SUSIRET_BOOL SusiVgaSetBacklight(HSUSI hSusi, unsigned long dwUnit, unsigned long dwSetting);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
dwSetting	the backlight value

Return Value

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

Remarks

The range of the value is between 0 and SUSI_VGA_BACKLIGHT_MAX (100), respectively 0 and 100%.

7.57 SusiVgaGetInfo

Gets the VGA board information.

SUSIRET_BOOL SusiVgaGetInfo(HSUSI hSusi, unsigned long dwUnit, SUSIVGAINFO *pInfo);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
pInfo	the buffer where the VGA information will be stored

Return Value

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

Remarks

Gets the VGA board information of a iManager API compliant board addressed by hSusi.

7.58 SusiTemperatureCount

Check number of Temperature function on the platform.

SUSIRET_ULONG SusiTemperatureCount(HSUSI hSusi);

Parameters

hSusi the board handle

Return Value

the number of installed temperature sensors in the system.

Remarks

7.59 SusiTemperatureGetInfo

Gets the temperature sensor information.

SUSIRET_BOOL SusiTemperatureGetInfo(HSUSI hSusi, unsigned long dwUnit, SUSITEMPERATUREINFO *pInfo);

Parameters

hSusi the board handle

dwUnit the unit number you want to control

pInfo pointer to the sensor information structure

Return Value

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

Remarks

Gets the information structure of the specified temperature sensor.

7.60 SusiTemperatureGetCurrent

Gets the temperature sensor current value.

SUSIRET_BOOL SusiTemperatureGetCurrent(HSUSI hSusi, unsigned long dwUnit, unsigned long *pdwSetting, unsigned long *pdwStatus);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
pdwSetting	pointer to the sensor's current measured value
pdwStatus	pointer to the sensor's current status value

Return Value

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

Remarks

Gets the actual value of the specified temperature sensor.

7.61 SusiTemperatureSetLimits

Sets the temperature limit for alarm.

SUSIRET_BOOL SusiTemperatureSetLimits(HSUSI hSusi, unsigned long dwUnit, SUSITEMPERATUREINFO *pInfo);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
pInfo	pointer to the sensor information structure.

Return Value

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

Remarks

7.62 SusiFanCount

Check number of fan functions on the platform.

SUSIRET_ULONG SusiFanCount(HSUSI hSusi);

Parameters

hSusi the board handle

Return Value

the number of installed fan sensors in the system.

Remarks

7.63 SusiFanIsAvailable

Check that the fan function unit is workable.

SUSIRET_BOOL SusiFanIsAvailable(HSUSI hSusi, unsigned long dwUnit);

Parameters

hSusi the board handle

dwUnit the unit number you want to control

Return Value

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

Remarks

7.64 SusiFanSetConfigStruct

Set auto fan function mode or alarm mode.

SUSIRET_BOOL SusiFanSetConfigStruct(HSUSI hSusi, unsigned long dwUnit, SUSIFANCONFIG *pConfig);

Parameters

hSusi the board handle

dwUnit the unit number you want to control

pConfig pointer to the auto fan function config.

Return Value

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

Remarks

For more information about how to use this API, please refer to the "Programming Overview", section "HWM functions".

7.65 SusiFanGetConfigStruct

Get information about auto fan function mode or alarm mode.

SUSIRET_BOOL SusiFanGetConfigStruct(HSUSI hSusi, unsigned long dwUnit, SUSIFANCONFIG *pConfig);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
pConfig	pointer to the auto fan function config.

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 "HWM functions".

7.66 SusiFanGetInfo

Gets the fan sensor information.

SUSIRET_BOOL SusiFanGetInfo(HSUSI hSusi, unsigned long dwUnit, SUSIFANINFO *pInfo);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
pInfo	pointer to the sensor information structure

Return Value

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

Remarks

Gets the information structure of the specified temperature sensor.

7.67 SusiFanGetCurrent

Gets the fan sensor current value.

SUSIRET_BOOL SusiFanGetCurrent(HSUSI hSusi, unsigned long dwUnit, unsigned long *pdwSetting, unsigned long *pdwStatus);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
pdwSetting	pointer to the sensor's current measured value
pdwStatus	pointer to the sensor's current status value

Return Value

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

Remarks

Gets the actual value of the specified fan sensor.

7.68 SusiFanSetLimits

Sets the fan limit for alarm.

SUSIRET_BOOL SusiFanSetLimits(HSUSI hSusi, unsigned long dwUnit, SUSI-FANINFO *pInfo);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
pInfo	pointer to the sensor information structure.

Return Value

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

Remarks

7.69 SusiVoltageCount

Check number of voltage function on the platform.

SUSIRET_ULONG SusiVoltageCount(HSUSI hSusi);

Parameters

hSusi	the board handle

Return Value

the number of installed voltage sensors in the system.

Remarks

7.70 SusiVoltageGetInfo

Gets the voltage sensor information.

SUSIRET_BOOL SusiVoltageGetInfo(HSUSI hSusi, unsigned long dwUnit, SUSIVOLTAGEINFO *pInfo);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
pInfo	pointer to the sensor information structure

Return Value

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

Remarks

Gets the information structure of the specified voltage sensor.

7.71 SusiVoltageGetCurrent

Gets the voltage sensor current value.

SUSIRET_BOOL SusiVoltageGetCurrent(HSUSI hSusi, unsigned long dwUnit, unsigned long *pdwSetting, unsigned long *pdwStatus);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
pdwSetting	pointer to the sensor's current measured value
pdwStatus	pointer to the sensor's current status value

Return Value

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

Remarks

Gets the actual value of the specified voltage sensor.

7.72 SusiVoltageSetLimits

Sets the voltage limit for alarm.

SUSIRET_BOOL SusiVoltageSetLimits(HSUSI hSusi, unsigned long dwUnit, SUSIVOLTAGEINFO *pInfo);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
pInfo	pointer to the sensor information structure.

Return Value

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

Remarks

7.73 SusiStorageAreaCount

Check number of storage area function on the platform.

SUSIRET_ULONG SusiStorageAreaCount(HSUSI hSusi, unsigned long dwUnit);

Parameters

hSusi the board handle

Return Value

the number of installed storage area sensors in the system.

Remarks

7.74 SusiStorageAreaType

Gets the Type of storage area on the platform.

SUSIRET_ULONG SusiStorageAreaType(HSUSI hSusi, unsigned long dwUnit);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control

Return Value

SUSI_STORAGE_AREA_EEPROM
SUSI_STORAGE_AREA_FLASH
SUSI_STORAGE_AREA_CMOS
SUSI_STORAGE_AREA_RAM
or
SUSI_STORAGE_AREA_UNKNOWN if the type is not known.

Remarks

This function is also used to determine the pure type of a dedicated storage area (by separating it from the unit number).

7.75 SusiStorageAreaSize

Gets the Size of storage area on the platform.

SUSIRET_ULONG SusiStorageAreaSize(HSUSI hSusi, unsigned long dwUnit);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control

Return Value

the size of the storage area in bytes.

Remarks

7.76 SusiStorageAreaBlockSize

Gets the block size of storage area on the platform.

SUSIRET_ULONG SusiStorageAreaBlockSize(HSUSI hSusi, unsigned long dwUnit);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control

Return Value

the block size of a storage area block in bytes.

Remarks

7.77 SusiStorageAreaRead

Read data form the storage area on the platform.

SUSIRET_BOOL SusiStorageAreaRead(HSUSI hSusi, unsigned long dwUnit, unsigned long dwOffset, unsigned char *pBytes, unsigned long dwLen);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
dwOffset	byte offset where the data is read from
pBytes	pointer to the destination buffer
dwLen	number of bytes to read

Return Value

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

Remarks

Reads dwLen bytes from the storage area into buffer pBytes.

7.78 SusiStorageAreaWrite

Write data to the storage area on the platform.

SUSIRET_BOOL SusiStorageAreaWrite(HSUSI hSusi, unsigned long dwUnit, unsigned long dwOffset, unsigned char *pBytes, unsigned long dwLen);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
dwOffset	byte offset where the data writes to
pBytes	pointer to the source buffer
dwLen	number of bytes to write

Return Value

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

Remarks

Writes dwLen bytes from the buffer pBytes to the storage area .

7.79 SusiStorageAreaErase

Erase data of the storage area on the platform.

SUSIRET_BOOL SusiStorageAreaErase(HSUSI hSusi, unsigned long dwUnit, unsigned long dwOffset, unsigned long dwLen);

Parameters

hSusi	the board handle
dwUnitt	he unit number you want to control
dwOffset	byte offset to the area, which will be erased
dwLen	number of bytes to erase

Return Value

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

Remarks

Erases dwLen bytes from the storage area starting at offset dwOffset.

7.80 SusiStorageAreaEraseStatus

Get erase data of the storage area on the platform.

SUSIRET_BOOL SusiStorageAreaEraseStatus(HSUSI hSusi, unsigned long dwUnit, unsigned long dwOffset, unsigned long dwLen, unsigned long *lpStatus);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
dwOffset	byte offset to the which will be erased
dwLen	number of bytes to erase
lpStatus	pointer to the status

Return Value

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

Remarks

Return the status of the current area erase progress in lpStatus:

0	Erasing the specified area finished successfully
1	Erasing in progress
2	Erase error

7.81 SusiStorageAreaLock

Lock a storage area for write protect.

SUSIRET_BOOL SusiStorageAreaLock(HSUSI hSusi, unsigned long dwUnit, unsigned long dwFlags, unsigned char *pBytes, unsigned long dwLen);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
dwFlags	reserved for future use, set to 0
pBytes	pointer to the source buffer containing the secret string
dwLen	number of bytes to write

Return Value

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

Remarks

This function is used to write protect a storage area. Write access to a locked storage area is rejected as long as the area is unlocked with the SusiStorageAreaUnlock function call. Read access to a locked storage area isn't affected by this mechanism and therefore still permitted at any time. This kind of implementation allows you to set up features such as protected custom serial numbers or the selective enabling of software features. This function fails if the selected area is already locked.

The current release of the software only supports the locking of storage areas of type SUSI_STORAGE_AREA_EEPROM. The protection mechanism for this type expects a secret string with up to 6 characters. The length of the string must be specified in dwLen.

7.82 SusiStorageAreaUnlock

Unlock a storage area for write protect.

SUSIRET_BOOL SusiStorageAreaUnlock(HSUSI hSusi, unsigned long dwUnit, unsigned long dwFlags, unsigned char *pBytes, unsigned long dwLen);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
dwFlags	reserved for future use, set to 0
pBytes	pointer to the source buffer containing the secret string
dwLen	number of bytes to write

Return Value

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

Remarks

This function is used to unlock a write protected storage area that was previously locked using SusiStorageAreaLock. To unlock an area the secret string must be exactly the same as the string that was used to lock the area. If the attempt to unlock an area fails, any further try to unlock the area requires a preceding power off/on cycle of the system.

This function fails if the selected area is already unlocked.

7.83 SusiStorageAreaIsLocked

Check the storage area is locked.

SUSIRET_BOOL SusiStorageAreaIsLocked(HSUSI hSusi, unsigned long dwUnit, unsigned long dwFlags);

Parameters

hSusi	the board handle
dwUnit	the unit number you want to control
dwFlags	reserved for future use, set to 0

Return Value

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

Remarks

This function is used to determine the locking state of a storage area. It returns true if the selected area is locked. It returns false if the area isn't locked or if the functionality isn't implemented.

Trusted ePlatform Services

ADVANTECH

www.advantech.com