MiAPI DLL User Manual

Version 3.1a

MiTAC Computing Technology

Version History

Date	Version	Remark
2015/01/30	0.1	draft
2015/03/06	0.2	Revise all exporting APIs.
2015/06/25	0.3	Correct some function definitions.(Watchdog, display)
2015/0720	0.4	Add 2.4 tutorials. The APIS base v0.9.
2015/10/15	1.0	Official release for MiAPI.
2016/08/29	2.0	Support Intel Skylake platform. Remove obsolete APIs.
2017/03/08	2.1	1.Support Intel Appolo Lake platform 2. Revise MiAPI_SMBUS_Read() and MiAPI_SMBUS_Write() 3. Apply BIOS SMI call to watchdog
2017/11/16	2.2	1. Define the GPIO direction/voltageLevel. 2. Redefine Watchdog disable function.
2017/12/01	3.0	1.Redefine SMBUS features 2. Revise version to correspond BIOS SMI spec. 3.Redefine display features
2018/08/02	3.1	1. Add display on/off features. 2. Use errorcode to indicate MB's support instead of show old DLL version.
2019/02/21	3.1a	1. Add sample codes. 2. Revise user manual.

Index

1. Overview

MiTAC provides a suite of software APIs , called MiAPI, to speed up the external devices development and control on MiTAC embedded boards.

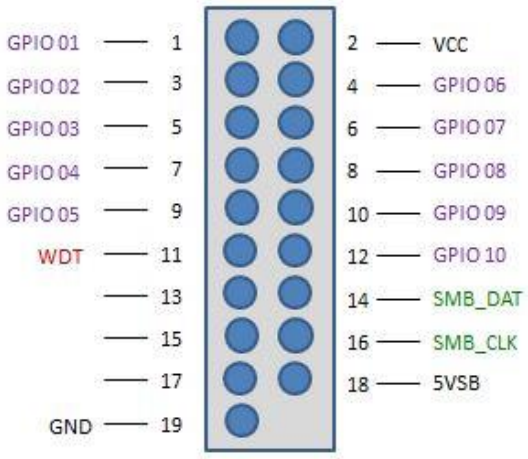
This software APIs provide not only the underlying and transparent drivers to access system interfaces, but also a rich set of easy-use and integrated function calls including GPIO, SMBUS, watchdog, and Hardware Monitor .

This document provides the programming details and interfaces exposed by the MiAPI (MiTAC Application Programming Interface) library for Windows.

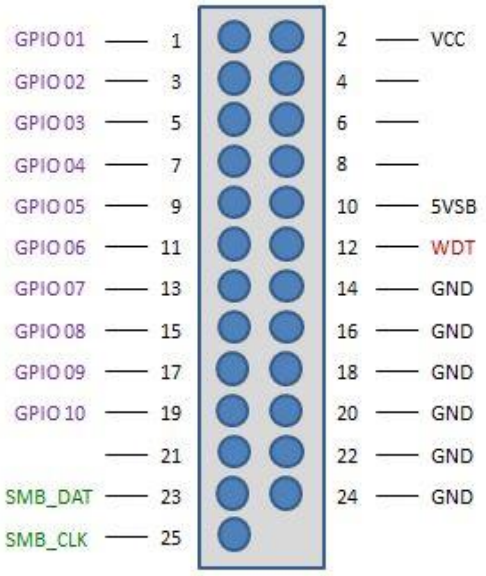
2. Specification

2.1 Hardware feature

2.1.1 Header pin definition



20 pins header



26 pins header

2.1.2 Pin list

- 10 GPIO pins.
- SMBUS data/SMBUS clock
- Watchdog pin
- VCC/5VSB/Ground

2.2 Software feature

2.2.1 OS environment

- Windows 7 32bit/64bit
- Windows 8.1 32bit/64bit
- Windows 10 32bit/64bit

2.2.2 Compiler tool

- Microsoft Visual Studio C++ 2010 sp1
- Microsoft Visual Studio C++ 2013

2.2.3 Package contents

Items	Folder/Files	Description
User Manual	MiAPI DLL User Manual_v3.1.PDF	This document
Library	1. Import library : SRC\LIB\MiAPI.lib 2. Dynamic link library: SRC\LIB\MiAPI.dll	1. The reference imported library to put and link in source code. 2. The DLL file must put into the same folder with application.
Include header	SRC\LIB\ MiAPI.h	The MiAPI header file to import DLL functions.
sample project source code	SRC\MiAPI_demo	Sample VC++ 2010 project to demonstrate MiAPI features.
Test plan	TestKit\MiAPI_R3.01_TestPlan.xls	MiAPI verification test procedure.
Test kit tools	Testkit\DisplayControl Testkit\GPIO Testkit\HWMonitoring TestKit\SMBUS TestKit\Watchdog	Test tools coded by MiAPI Including : Display Control, GPIO, HW Monitoring, SMBUS, Watchdog

3. How to Use

3.1 Code Guidance

1. Put **MiAPI.lib** and **MiAPI.h** into your project folder.
2. Add " **#include <windows.h>** " in code or insert into "stdafx.h".
3. Add **#include "MiAPI.h"** and specify the included folder that project can reference.
4. Add **pragma comment(lib,MiAPI.lib)** or use **Add Reference** dialog box lists the libraries **MPAI.lib** that you can reference.
5. Call **MiAPI_Start()** to start the DLL loading.
6. Call the DLL API functions for your application.
7. Call **MiAPI_Exit()** to release DLL resource when existed.
8. You must put **MiAPI.dl**l in the same folder of the executable application. To run your application, ensure it is run under **administrator** privilege.

Refer to the Chapter 4 or source code of MiAPI_demo (SRC\MiAPI_demo) for better understanding about MiAPI API usage.

3.2 User Guide for Test Tools

3.2.1 MiAPI_demo

Usage :

1. Open a command prompt run as **administrator.**
2. Find in \SRC\release and run "MiAPI_demo.exe".
3. Choose what you like to demo.

```
=== Demo Menu ===
  0 : Exit
  1 : Show BIOS & MiAPI version
  2 : HW monitoring
  3 : GPIO setting
  4 : SMBUS scan
  5 : Display Control
  6 : Watchdog
Choose (0 ~ 6) : 6
```

4. There are a sub-menu in Watchdog demo. Be cautious about this case 2. It may damage the hard drive due to its force shutdown.

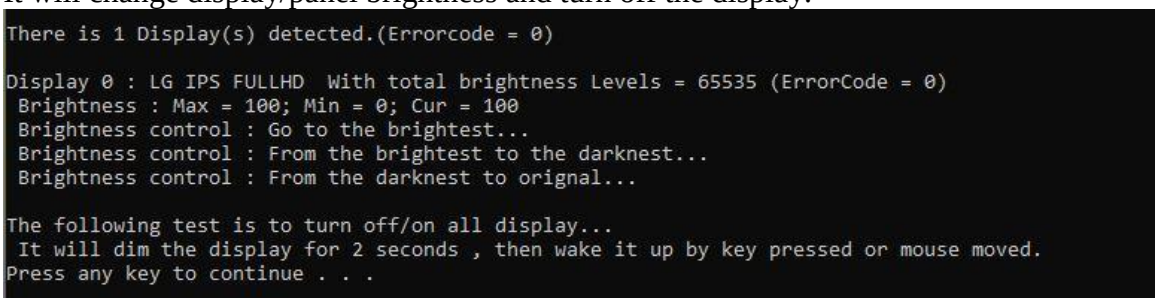
```
*** Watchdog demo ***

=== WDT Demo cases ===
  0 : Exit
  1 : App thread exits and halt WDT --- NORMAL
  2 : APP thread crash and system reboot --- DANGER
Choose (0 ~ 2) :
```

3.2.2 DisplayControl

Usage :

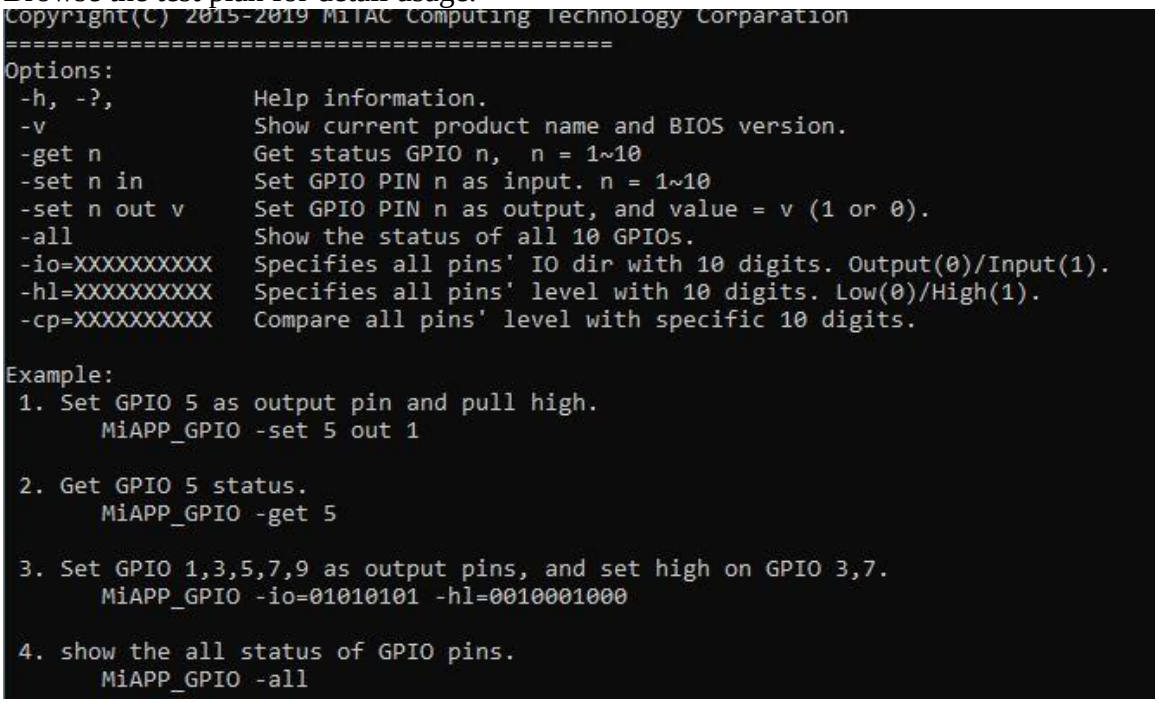
1. Open a command prompt run as **administrator.**
2. Find in \TestKit\DisplayControl and run "MiAPP_DisplayControl.exe".
3. It will change display/panel brightness and turn off the display.



3.2.3 GPIO

Usage :

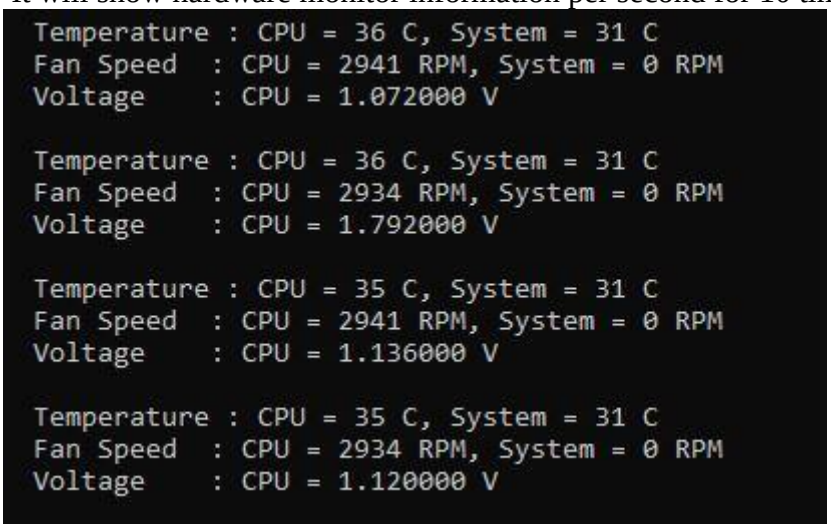
1. Open a command prompt run as **administrator.**
2. Find in \TestKit\GPIO and run "MiAPP_DisplayControl.exe –h" to show help.
3. Browse the test plan for detail usage.



3.2.4 HWMonitoring

Usage :

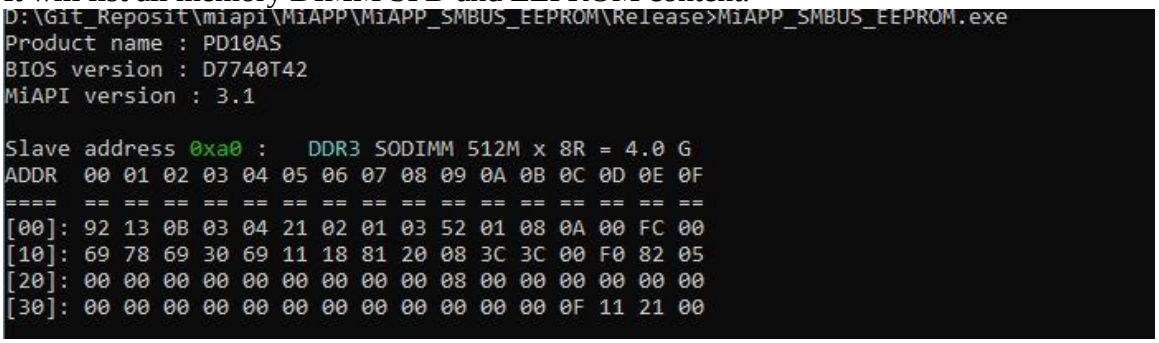
1. Open a command prompt run as **administrator.**
2. Find in \TestKit\HWMonitoring and run "MiAPP_HWMoniter.exe".
3. It will show hardware monitor information per second for 10 times.



3.2.5 SMBUS

Usage :

1. Open a command prompt run as **administrator.**
2. Find in \TestKit\SMBUS and run "MiAPP_SMBUS_EEPROM.exe".
3. It will list all memory DIMM SPD and EEPROM content.



3.2.6 Watchdog

Usage :

1. Open a command prompt run as **administrator.**
2. Find in \TestKit\Watchdog and run "MiAPP_WDT.exe –h" to show help.

```
MiAPP WatchDog Timer V0.01
Copyright(C) 2015-2016 MiTAC Computing Technology Corparation
==========================================
Options:
 -h, -?,              Help information.
 -timeout=XXX         Watchdog timeout in seconds, ranging 2 ~ 614 seconds.
 -reboot=X            Reboot system or not when timeout. No(0)/Yes(1).
 -status              Show WDT setting and current timer count.
 -go                  Start the watchdog timer.
 -halt                Halt or pause the watchdog timer countdown.
 -refresh             Refresh the WDT count to its timeout as set.
```

3. See "test_watchdog.cmd" for detail usage.

```
D:\Git_Reposit\miapi\MiAPI_Kit\TestKit\Watchdog>test_watchdog.cmd
 1. Check current watchdog timeout count...
WDT Min=2, Max=1023, Current countdown=10

 2. Let Watchdog go for 2 second...

Waiting for 0 seconds, press CTRL+C to quit ...
WDT Min=2, Max=1023, Current countdown=7
 Check if the countdown value decrese or not?

 3. After 2 seconds, Refresh the watchdog....

Waiting for 0 seconds, press CTRL+C to quit ...
WDT Min=2, Max=1023, Current countdown=10

 4. Halt the watchdog and set the timeout to 5 seconds(or 9 ticks)
WDT Min=2, Max=1023, Current countdown=5

 5. *** Make sure you have saved the log above... ***
 Next, hit any key to continue and system will ***REBOOT*** in 5 second...!
Press any key to continue . . .
```

4. MiAPI API Features

4.1 Basic functions

The basic functions are used to install MiAPI interface for launching API functions. Follow 3.1 Code Guidance and Sample Reference Code in the bottom to start application coding.

MiAPI_Start

Description

Initialize the MiAPI Library.

int MiAPI_Start(void)

Parameters

None.

Return Value

MiAPI_OK (0x00)	Success
MiAPI_INIT_FAIL (0x01)	Driver or library initialization fail
MiAPI_NOT_SUPPORT (0x02)	This board doesn't support MiAPI.

Remarks

An application must call MiAPI_Start before calling others MiAPI functions.

MiAPI_Exit

Description

Exit the MiAPI Library.

void MiAPI_Exit(void)

Parameters

None.

Return Value

None.

Remarks

Application has to call MiAPI_Exit to free the resource before it exits.

Sample Code

```
#include "stdafx.h"
#include <Windows.h>
#include "MiAPI.h"

// Alternatively add the following pragma comment, instead of setting up reference dependence
// in compiler environment setting.  Be aware to put the same bits version MiAPI.lib in the
//source folder.

#pragma comment(lib, "MiAPI.lib")

int _tmain(int argc, _TCHAR* argv[])
{

        //-- Start the MiAPI libary
          if( MiAPI_Start() != MiAPI_OK )
          {
                  printf("Error: Failed to initialize MAPI library.\n");
                  return MiAPI_INIT_FAIL;
          }

        //--  Start  MiAPI functions from here.
          Do_MiAPI_Version();
          DO_MiAPI_HWMonitoring();
          Do_MiAPI_GPIO();
         Do_MiAPI_SMBUS_SCAN();
          Do_MiAPI_DisplayControl();
          Do_MiAPI_WDT();

          //-- It must free the resource by call MiAPI_Exit () when application  exits.
          MiAPI_Exit();

          return 0;
}
```

4.2 Version

The feature is used to identify mother board BIOS and version of current MiAPI DLL.

MiAPI_GetBIOSVersion

Description

Get mother board BIOS version.

int MiAPI_GetBIOSVersion(CHAR *BIOSVersion, DWORD *size)

Parameters

BIOSVersion	[out]	Pointer to a string which the BIOS version is returned.
size	[out]	Pointer to a variable that specifies the size of string to BIOSVersion

.

Return Value

MiAPI_OK (0x00)	Success
MiAPI_READ_FAIL(0x04)	Fail

MiAPI_GetProductName

Description

Get the current product name

int MiAPI_GetProductName(CHAR *ProductName, DWORD *size)

Parameters

ProductName	[out]	Pointer to a string which the product name is returned.
size	[out]	Pointer to a variable that specifies the size of string to ProductName

Return Value

MiAPI_OK (0x00)	Success
MiAPI_READ_FAIL(0x04)	Fail

MiAPI_GetMiAPIVersion

Description

Get MiAPI version.

int MiAPI_GetMiAPIVersion(DWORD *major, DWORD *minor)

Parameters

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

Return Value

MiAPI_OK (0x00)	Success
MiAPI_NOT_SUPPORT (0x02)	This board doesn't support MiAPI.
MiAPI_OLD_VERSION(0x05)	Mother board support only limited or old features.

Remarks

If the return code of MiAPI_GetMiAPIVersion returns MiAPI_OLD_VERSION (0x05), it implies the MB's BIOS implement some limited features only, and might have few compatibility issues on watchdog,SMBUS and GPIO. Please contact vendor for issue report.

Sample Code

```
int Do_MiAPI_Version(void)
{
	int Major, Minor;
	char BIOSVersion[80];
	char ProductName[80];
	DWORD size;
	int ret = MiAPI_OK;

	ret = MiAPI_GetProductName(ProductName, &size);
	printf("Product name : %s\n", ProductName);

	ret = MiAPI_GetBIOSVersion(BIOSVersion, &size);
	printf("BIOS version : %s\n", BIOSVersion);

	ret = MiAPI_GetMiAPIVersion(&Major, &Minor);
	printf("MAPI DLL version : %d.%d \n",Major,Minor);

	return ret;
}
```

4.3 Display Control

These APIs provide display control features including display information, brightness, contrast, orientation, and screen on/off . Most of modern monitors/panels that MS Windows supports can work properly.

MiAPI_Display_GetAmountOfMonitors

Description

Get the current amount of monitors connected to the board.

Int MiAPI_Display_GetAmountOfMonitors(int *AmountOfMonitors)

Parameters

AmountOfMonitors	[out]	Pointer to a variable of amount of connected monitors .

Return Value

MiAPI_OK (0x00)	Success
MiAPI_VGA_GET_AMOUNT_OF_MONITORS_FAIL (0x54)	Fail

MiAPI_Display_GetMonitorInfo

Description

Get monitor information form specific monitor index..

Int MiAPI_Display_GetMonitorInfo(MIAPI_MONITOR_INFO *MiAPI_MonitorInfo, DWORD Index)

Parameters

MIAPI_MONITOR_INFO	[out]	Monitor info members: DeviceIndex, FriendlyDeviceName, Brightness, Orientation.
Index	[in]	Specifies the monitor to get.

```
typedef struct _MIAPI_MONITOR_INFO
{
   WORD Orientation;
   DWORD DeviceIndex;
   WCHAR FriendlyDeviceName[64];
   DWORD WMITotalBrightnessLevel;
} MIAPI_MONITOR_INFO;
```

Return Value

MiAPI_OK (0x00)	Success
MiAPI_VGA_INIT_FAIL (0x51)	Fail

MiAPI_Display_GetBrightness

Description

Get the current panel brightness.

int MiAPI_Display_GetBrightness(MIAPI_BRIGHTNESS *MiAPI_Brightness, DWORD Index)

Parameters

MIAPI_BRIGHTNESS	[out]	Pointer to a struct which contains members : Minimum Brightness, Maximum Brightness, Current Brightness
Index	[in]	Specifies the monitor to get its brightness.

```
typedef struct _MIAPI_BRIGHTNESS
{
    DWORD MinimumBrightness;
    DWORD MaximumBrightness;
    DWORD CurrentBrightness;
} MIAPI_BRIGHTNESS;
```

Return Value

MiAPI_OK (0x00)	Success
MiAPI_VGA_GETBRIGHTNESS_FAIL (0x55)	Fail

MiAPI_Display_SetBrightness

Description

Set current panel brightness.

int MiAPI_SetBrightness(DWORD NewBrightness, DWORD Index)

Parameters

NewBrightness	[in]	Specifies the brightness value to be set.
Index	[in]	Specifies the monitor to set its brightness.

Return Value

MiAPI_OK (0x00)	Success
MiAPI_VGA_SETBRIGHTNESS_FAIL	Fail

(0x56)	

MiAPI_Display_GetContrast

Description

Get minimum ,maximum and current contrast values from specific monitor.

MiAPI_Display_GetContrast(MIAPI_CONTRAST *MiAPI_Contrast, DWORD Index)

Parameters

MIAPI_CONTRAST	[out]	Pointer to a struct which contains members : Minimum Contrast, Maximum Contrast, Current Contrast
Index	[in]	Specifies the monitor to get its contrast.

```
typedef struct _MIAPI_CONTRAST
{
   DWORD MinimumContrast;
   DWORD MaximumContrast;
   DWORD CurrentContrast;
} MIAPI_CONTRAST;
```

Return Value

MiAPI_OK (0x00)	Success
MiAPI_VGA_GET_CONTRAST_FAIL (0x57)	Fail

MiAPI_ Display_SetContrast

Description

Set display's contrast of specific monitor.

MiAPI_Display_SetContrast(DWORD NewContrast, DWORD Index)

Parameters

NewContrast	[in]	The new contrast value to be set.
Index	[in]	Specifies the monitor to set its contract.

Return Value

MiAPI_OK (0x00)	Success
MiAPI_VGA_SET_CONTRAST_FAIL (0x58)	Fail

MiAPI_ Display_SetOrientation

Description

Set display's orientation of specific monitor.

MiAPI_Display_SetOrientation(short Orientation, DWORD Index)

Parameters

Orientation	[in]	Display orientation degrees to set: 0: natural orientation of the display device;. 90: rotated 90 degrees in clockwise. 180: rotated 180 degrees in clockwise. 270: rotated 270 degrees in clockwise.
Index	[in]	Specifies the monitor to set its brightness.

Return Value

MiAPI_OK (0x00)	Success
MiAPI_VGA_SET_ORIENTATION_FAIL (0x59)	Fail

MiAPI_ Display_Rescan

Description

The function is used to rescan monitors and renew the device list in case of monitors live changing..

Int MiAPI_Display_Rescan()

Parameters

None

Return Value

None

MiAPI_Display_On

Description

The function is used to turn on monitors. Note: It might not work for some legacy monitors that it is not fully compatible with Windows API.

Int MiAPI_Display_On()

Parameters

None

Return Value

None

MiAPI_Display_Off

Description

The function is used to turn off monitors. Note: It might not work for some legacy monitors that it is not fully compatible with Windows API. And this display will wake up easily by event notification such as mouse moving or key pressing.

Int MiAPI_Display_Off()

Parameters

None

Return Value

None

Sample Code

```
int Do_MiAPI_DisplayControl(void)
{
        int ret = MiAPI_OK;
        int nDisplays;
        MIAPI_MONITOR_INFO disp;
        MIAPI_BRIGHTNESS brg;
        MIAPI_CONTRAST crst;


        ret = MiAPI_Display_GetAmountOfMonitors(&nDisplays);
        printf("\nThere is %d Display(s) detected.(Errorcode = %X)\n", nDisplays, ret);

        for(int i = 0; i < nDisplays; i++)
        {
            //-- Get Display infomation
                ret = MiAPI_Display_GetMonitorInfo(&disp, i);
```

```
        if(ret == MiAPI_OK) {

                wprintf(L"\nDisplay #%02d : %ls\n",
                disp.DeviceIndex,  disp.FriendlyDeviceName);

                //-- Get Brightness info
                   ret = MiAPI_Display_GetBrightness(&brg, i);
                  printf("Brightness: Min = %d, Max = %d, Current = %d\n",
                        brg.MinimumBrightness,
                        brg.MaximumBrightness,
                        brg.CurrentBrightness );

                //-- Set brightness to 50%.
                 ret = MiAPI_Display_SetBrightness(50, i);

                //-- Get Contrast info
                   ret = MiAPI_Display_GetContrast (&crst, i);
                   printf("Contrast: Min = %d, Max = %d, Current = %d\n",
                          crst.MinimumContrast,
                          crst.MaximumContrast,
                          crst.CurrentContrast );

                //-- Set contrast  to 50%.
                 ret = MiAPI_Display_SetContrast (50, i);

                //-- Rotate 90 degree
                 ret=MiAPI_Display_SetOrientation(90, i);

                }
        }

   //-- Display On & OFF
         printf(" The display will dim the display for 2 seconds ,
                  then wake it up by key pressed or mouse moved.\n");

         // Turn off monitor
         ret = MiAPI_Display_Off();

         Sleep(2000);
         // Turn on monitor
         ret = MiAPI_Display_On();

        return ret;
}
```

4.4 GPIO

A **general-purpose input/output** (**GPIO**) is an uncommitted digital signal pin on an integrated circuit or electronic circuit board whose behavior—including whether it acts as input or output—is controllable by the user at run time.

MiAPI_GPIO_GetStatus

Description

Read current status of one GPIO pin.

int MiAPI_GPIO_GetStatus(BYTE PinNum, GPIO *status)

Parameters

PinNum	[in]	GPIO pin to be read, ranging from 1~10.
status	[out]	Pointer to a structure for GPIO status including its direction and voltage level.
GPIO.Direction	[out]	GPIO status member to indicate input or output direction. 1 = Input ; 0 = Output.
GPIO.VoltageLevel	[out]	GPIO status member to indicate pin high or low voltage level. 1 = High ; 0 = Low.

```
typedef struct GPIOStatus
    {
      BYTE Direction;
      BYTE VoltageLevel;
    } GPIO;
```

Direction : 1 = Input ; 0 = Output.
VoltageLevel : 1 = High; 0 = Low

Return Value

MiAPI_OK (0x00)	Success
MiAPI_GPIO_GETSTATUS_FAIL(0x34)	Fail

Remarks

The GPIO direction is input(1) and voltage level(1) for these 10 pins by default.

MiAPI_GPIO_SetStatus

Description

Set one GPIO output pin as status high or low.

int MiAPI_GPIO_SetStatus(BYTE PinNum, GPIO status)

Parameters

PinNum	[in]	GPIO pin to be read, ranging from 1~10.
status	[in]	Pointer to a structure for GPIO status including its direction and voltage level.
GPIO.Direction	[in]	GPIO status member to indicate input or output direction. 1 = Input ; 0 = Output.
GPIO.VoltageLevel	[in]	GPIO status member to indicate pin high or low voltage level. 1 = High ; 0 = Low.

```
typedef struct GPIOStatus
    {
      BYTE Direction;
      BYTE VoltageLevel;
    } GPIO;
```

Direction : 1 = Input ; 0 = Output.
VoltageLevel : 1 = High; 0 = Low

Return Value

MiAPI_OK (0x00)	Success
MiAPI_GPIO_SETSTATUS_FAIL(0x35)	Fail

Remarks

The voltage level will be ignored when its direction is set as input(1).

Sample Code

```
int Do_MiAPI_GPIO(void)
{
        int ret = MiAPI_OK;
        MIAPI_GPIO_STATUS gpio1, gpio2, gpio3;

        //--Set GPIO 1 to input; for input mode, this voltage level is dummy as no use.
         gpio1.Direction = 0x01;    //input = 0x01; output = 0x00;
         ret = MiAPI_GPIO_SetStatus(1, gpio1);

        //--Set GPIO2 to output low
         gpio2.Direction = 0x00;    //input = 0x01; output = 0x00 ;
         gpio2.VoltageLevel = 0x00;  // Low = 0x00; High = 0x01
         ret = MiAPI_GPIO_SetStatus(2, gpio2);

        //-- Show  GPIO 3 status
         ret = MiAPI_GPIO_GetStatus(3, &gpio3);
        printf("  GPIO 3  :  DIR=%d  LEVEL=%d \n",  gpio3.Direction, gpio3.VoltageLevel);
        return ret;
}
```

4.5 Hardware Monitoring

Hardware Monitoring provides user the system health information including fan speed, temperature and CPU voltage.

MiAPI_GetFanSpeed

Description

Read the current value of one of the fan speed sensors.

int MiAPI_GetFanSpeed(WORD fanType, WORD *retval)

Parameters

fanType	[in]	Specifies a fan speed sensor to get. 1 = CPUFAN , 2 = SYSFAN
retval	[out]	Point to a variable of the fan speed in RPM

Return Value

MiAPI_OK (0x00)	Success
MiAPI_FANSPEED_GET_FAIL(0x61)	Fail
MiAPI_NOT_SUPPORT (0x02)	Board does support this function.

MiAPI_SetFanSpeed

Description

Control the speed of one of the fans.

int MiAPI_SetFanSpeed(WORD fanType, WORD setval)

Parameters

fanType	[in]	Specifies a fan speed sensor to get. 0 = Automatic Fan curve control. 1 = CPUFAN , 2 = SYSFAN
setval	[in]	Fan speed in RPM

Return Value

MiAPI_OK (0x00)	Success
MiAPI_FANSPEED_SET_FAIL(0x62)	Fail
MiAPI_NOT_SUPPORT (0x02)	Board does support this function.

Remarks

FanType is suggested to set back to Automatic Fan curve control(0) when manual control ends. And the RPM setval will be ignored as fantype is 0. .

MiAPI_GetTemperature

Description

Read the current value of one of the temperature sensors

BOOL MiAPI_GetTemperature(WORD tempType, WORD *retval)

Parameters

tempType	[in]	Specify a temperature sensor to get. 1 = CPU , 2 = SYSTEM
retval	[out]	Point to a variable of the temperature in Celsius.

Return Value

MiAPI_OK (0x00)	Success
MiAPI_TEMPERATURE_GET_FAIL (0x64)	Fail
MiAPI_NOT_SUPPORT (0x02)	Board does support this function.

MiAPI_GetVoltage

Description

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

int MiAPI_GetVoltage(DWORD voltType, WORD *retval)

Parameters

voltType	[in]	Specify a temperature sensor to get. 1 = CPU , 2 = MEMORY DIMM
retval	[out]	Point to a variable of the voltage in Volt.

Return Value

MiAPI_OK (0x00)	Success
MiAPI_HWMON_GETVOLT_FAIL (0x63)	Fail
MiAPI_NOT_SUPPORT (0x02)	Board does support this function.

Sample Code

```
int DO_MiAPI_HWMonitoring(void)
{
        int ret = MiAPI_OK;
        WORD dummy = 0;
        WORD T_CPU = 0 , T_SYS;
        WORD RPM_CPU = 0, RPM_SYS = 0;
        WORD Volt_CPU = 0;

        ret = MiAPI_GetTemperature(1, &T_CPU, &dummy);
        ret = MiAPI_GetTemperature(2, &T_SYS, &dummy);
        printf("\n  Temperature : CPU = %d C, System = %d C\n", T_CPU, T_SYS );

        ret = MiAPI_GetFanSpeed(1, &RPM_CPU, &dummy);
        ret = MiAPI_GetFanSpeed(2, &RPM_SYS, &dummy);
        printf("  Fan Speed   : CPU = %d RPM, System = %d RPM\n", RPM_CPU, RPM_SYS);

        ret = MiAPI_GetVoltage(1, &Volt_CPU, &dummy);
        printf("  Voltage     : CPU = %3.3f V\n", (float)Volt_CPU/1000.0);

        return ret;
}
```

4.6 SMBUS

The **System Management Bus** (abbreviated to **SMBus** or **SMB**) is a single-ended simple two-wire bus for the purpose of lightweight communication. SMBus is used as an interconnect in several platform management standards such as I2C devices,EEPROM.

MiAPI_SMBusReadQuick

Description

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

`int MiAPI_SMBusReadQuick(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

`MiAPI_OK (0x00)`	`Success`
`SMBUS_TIMEOUT (0x41)`	The transaction did not complete within an internally specified timeout period, or the controller is not available for use.
`SMBUS_INVALID_PARAMETER (0x42)`	Length or Buffer is NULL for any operation besides quick read or quick write..
`SMBUS_UNSUPPORTED (0x43)`	The operation is unsupported
`SMBUS_BUFFER_TOO_SMALL (0x44)`	The buffer was not enough for the command operation. Choose other commands for the larger size.
`SMBUS_CRC_ERROR (0x45)`	Packet Error Code Checking was mismatch.
`SMBUS_DEVICE_ERROR (0x46)`	There was an SMBUS error (NACK) during the operation. Slave device is not present or is in a hung condition.

MiAPI_SMBusWriteQuick

Description

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

`int MiAPI_SMBusWriteQuick(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

`MiAPI_OK (0x00)`	`Success`
`SMBUS_TIMEOUT (0x41)`	The transaction did not complete within an internally specified timeout period, or the controller is not available for use.
`SMBUS_INVALID_PARAMETER (0x42)`	Length or Buffer is NULL for any operation besides quick read or quick write..
`SMBUS_UNSUPPORTED (0x43)`	The operation is unsupported
`SMBUS_BUFFER_TOO_SMALL (0x44)`	The buffer was not enough for the command operation. Choose other commands for the larger size.
`SMBUS_CRC_ERROR (0x45)`	Packet Error Code Checking was mismatch.
`SMBUS_DEVICE_ERROR (0x46)`	There was an SMBUS error (NACK) during the operation. Slave device is not present or is in a hung condition.

MiAPI_SMBusReceiveByte

Description

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

`int MiAPI_SMBusReceiveByte(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

`MiAPI_OK (0x00)`	`Success`
`SMBUS_TIMEOUT (0x41)`	The transaction did not complete within an internally specified timeout period, or the controller is not available for use.
`SMBUS_INVALID_PARAMETER (0x42)`	Length or Buffer is NULL for any operation besides quick read or quick write..
`SMBUS_UNSUPPORTED (0x43)`	The operation is unsupported
`SMBUS_BUFFER_TOO_SMALL (0x44)`	The buffer was not enough for the command operation. Choose other commands for the larger size.
`SMBUS_CRC_ERROR (0x45)`	Packet Error Code Checking was mismatch.
`SMBUS_DEVICE_ERROR (0x46)`	There was an SMBUS error (NACK) during the operation. Slave device is not present or is in a hung condition.

Remarks

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

MiAPI_SMBusSendByte

Description

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

`int MiAPI_SMBusSendByte(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

`MiAPI_OK (0x00)`	`Success`
`SMBUS_TIMEOUT (0x41)`	The transaction did not complete within an internally specified timeout period, or the controller is not available for use.
`SMBUS_INVALID_PARAMETER (0x42)`	Length or Buffer is NULL for any operation besides quick read or quick write..
`SMBUS_UNSUPPORTED (0x43)`	The operation is unsupported
`SMBUS_BUFFER_TOO_SMALL (0x44)`	The buffer was not enough for the command operation. Choose other commands for the larger size.
`SMBUS_CRC_ERROR (0x45)`	Packet Error Code Checking was mismatch.
`SMBUS_DEVICE_ERROR (0x46)`	There was an SMBUS error (NACK) during the operation. Slave device is not present or is in a hung condition.

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`.

MiAPI_SMBusReadByte

Description

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

```
int MiAPI_SMBusReadByte(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 receives the byte data.

Return Value

`MiAPI_OK (0x00)`	`Success`
`SMBUS_TIMEOUT (0x41)`	The transaction did not complete within an internally specified timeout period, or the controller is not available for use.
`SMBUS_INVALID_PARAMETER (0x42)`	Length or Buffer is NULL for any operation besides quick read or quick write..
`SMBUS_UNSUPPORTED (0x43)`	The operation is unsupported
`SMBUS_BUFFER_TOO_SMALL (0x44)`	The buffer was not enough for the command operation. Choose other commands for the larger size.
`SMBUS_CRC_ERROR (0x45)`	Packet Error Code Checking was mismatch.
`SMBUS_DEVICE_ERROR (0x46)`	There was an SMBUS error (NACK) during the operation. Slave device is not present or is in a hung condition.

Remarks

MiAPI_SMBusWriteByte

Description

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

`int MiAPI_SMBusWriteByte(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

`MiAPI_OK (0x00)`	`Success`
`SMBUS_TIMEOUT (0x41)`	The transaction did not complete within an internally specified timeout period, or the controller is not available for use.
`SMBUS_INVALID_PARAMETER (0x42)`	Length or Buffer is NULL for any operation besides quick read or quick write..
`SMBUS_UNSUPPORTED (0x43)`	The operation is unsupported
`SMBUS_BUFFER_TOO_SMALL (0x44)`	The buffer was not enough for the command operation. Choose other commands for the larger size.
`SMBUS_CRC_ERROR (0x45)`	Packet Error Code Checking was mismatch.
`SMBUS_DEVICE_ERROR (0x46)`	There was an SMBUS error (NACK) during the operation. Slave device is not present or is in a hung condition.

Remarks

MiAPI_SMBusReadWord

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

```
int MiAPI_SMBusReadWord(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

`MiAPI_OK (0x00)`	`Success`
`SMBUS_TIMEOUT (0x41)`	The transaction did not complete within an internally specified timeout period, or the controller is not available for use.
`SMBUS_INVALID_PARAMETER (0x42)`	Length or Buffer is NULL for any operation besides quick read or quick write..
`SMBUS_UNSUPPORTED (0x43)`	The operation is unsupported
`SMBUS_BUFFER_TOO_SMALL (0x44)`	The buffer was not enough for the command operation. Choose other commands for the larger size.
`SMBUS_CRC_ERROR (0x45)`	Packet Error Code Checking was mismatch.
`SMBUS_DEVICE_ERROR (0x46)`	There was an SMBUS error (NACK) during the operation. Slave device is not present or is in a hung condition.

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.

MiAPI_SMBusWriteWord

Description

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

`int MiAPI_SMBusWriteWord(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

`MiAPI_OK (0x00)`	`Success`
`SMBUS_TIMEOUT (0x41)`	The transaction did not complete within an internally specified timeout period, or the controller is not available for use.
`SMBUS_INVALID_PARAMETER (0x42)`	Length or Buffer is NULL for any operation besides quick read or quick write..
`SMBUS_UNSUPPORTED (0x43)`	The operation is unsupported
`SMBUS_BUFFER_TOO_SMALL (0x44)`	The buffer was not enough for the command operation. Choose other commands for the larger size.
`SMBUS_CRC_ERROR (0x45)`	Packet Error Code Checking was mismatch.
`SMBUS_DEVICE_ERROR (0x46)`	There was an SMBUS error (NACK) during the operation. Slave device is not present or is in a hung condition.

Remarks

The low byte of `Result` will be send to the slave device first and then the high byte.

MiAPI_SMBusReadBlock

Description

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

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

Return Value

`MiAPI_OK (0x00)`	`Success`
`SMBUS_TIMEOUT (0x41)`	The transaction did not complete within an internally specified timeout period, or the controller is not available for use.
`SMBUS_INVALID_PARAMETER (0x42)`	Length or Buffer is NULL for any operation besides quick read or quick write..
`SMBUS_UNSUPPORTED (0x43)`	The operation is unsupported
`SMBUS_BUFFER_TOO_SMALL (0x44)`	The buffer was not enough for the command operation. Choose other commands for the larger size.
`SMBUS_CRC_ERROR (0x45)`	Packet Error Code Checking was mismatch.
`SMBUS_DEVICE_ERROR (0x46)`	There was an SMBUS error (NACK) during the operation. Slave device is not present or is in a hung condition.

Remarks

None.

MiAPI_SMBusWriteBlock

Description

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

```
int MiAPI_SMBusWriteBlock(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

`MiAPI_OK (0x00)`	`Success`
`SMBUS_TIMEOUT (0x41)`	The transaction did not complete within an internally specified timeout period, or the controller is not available for use.
`SMBUS_INVALID_PARAMETER (0x42)`	Length or Buffer is NULL for any operation besides quick read or quick write..
`SMBUS_UNSUPPORTED (0x43)`	The operation is unsupported
`SMBUS_BUFFER_TOO_SMALL (0x44)`	The buffer was not enough for the command operation. Choose other commands for the larger size.
`SMBUS_CRC_ERROR (0x45)`	Packet Error Code Checking was mismatch.
`SMBUS_DEVICE_ERROR (0x46)`	There was an SMBUS error (NACK) during the operation. Slave device is not present or is in a hung condition.

Sample Code

Please refer to I2C devices data sheet and code snippet in source code of Project MiAPI_demo.

4.7 Watchdog

A **watchdog timer** is an electronic timer that is used to detect and recover from computer malfunctions. During normal operation, the computer regularly resets the watchdog timer to prevent it from elapsing, or "timing out". If, due to a hardware fault or program error, the computer fails to reset the watchdog, the timer will elapse and generate a timeout signal. The timeout signal is used to initiate corrective action or actions.

MiAPI_Watchdog_SetConfig

Description

Set watchdog timer with specified timeout value and define the action to reboot or trigger a WD_TIME pin when expired.

Int MiAPI_Watchdog_SetConfig (DWORD Timeout, BOOL Reboot)

Parameters

Timeout	[in]	Specifies a value in seconds for the watchdog timeout.
Reboot	[in]	True to reboot system when expired; False to trigger a low pulse on MiAPI WD_TIME pin.

Return Value

MiAPI_OK (0x00)	Success
MiAPI_WDT_SET_FAIL (0x22)	Fail

Remarks

Before starting watchdog, it must specify the watchdog timeout to expire and the behavior when it expires. The default timeout is 4 seconds, and reboot is false.

MiAPI_Watchdog_GetRange

Description

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

int MiAPI_Watchdog_GetRange(DWORD *min, DWORD *max, DWORD *cur)

Parameters

min	[out]	Pointer to a variable containing the minimum timeout value in seconds.
max	[out]	Pointer to a variable containing the maximum timeout value in seconds.
cur	[out]	Pointer to a variable containing the current count of the timer in seconds.

Return Value

MiAPI_OK (0x00)	Success

MiAPI_NOT_SUPPORT (0x02)	Watchdog doesn't support.
MiAPI_WDT_GET_FAIL (0x21)	Fail

Remarks

This function provides an indicator to show time range and the current remained time before watchdog expires. They are read-only, and will not alter watchdog's countdown.

MiAPI_Watchdog_Start

Description

Start the watchdog timer.

int MiAPI_Watchdog_Start(void)

Parameters

None

Return Value

MiAPI_OK (0x00)	Success
MiAPI_WDT_SET_FAIL (0x22)	Fail

MiAPI_Watchdog_Disable

Description

Disable the watchdog timer.

int MiAPI_Watchdog_Disable(void)

Parameters

None

Return Value

MiAPI_OK (0x00)	Success
MiAPI_WDT_SET_FAIL (0x22)	Fail

Remarks

Watchdog won't keep the timer count and may reset the count when it start again.

MiAPI_Watchdog_Refresh

Description

Reset the watchdog timer to the timeout value set by MiAPI_Watchdog_SetConfig. It is always inserted in application main loop to prevent watchdog expires.

int MiAPI_Watchdog_Refresh (void)

Parameters

None

Return Value

MiAPI_OK (0x00)	Success
MiAPI_WDT_SET_FAIL (0x22)	Fail

Remarks

It is better for users to set a longer 1.5~2 times timeout than user's service loop. Once system busy causes user service delays, it will be a safe tolerance for application refreshing the timer before watchdog expires.

Sample Code

```
//--Global variables
bool running;

unsigned int __stdcall thread_WDT(void* data)
// This is application worker thread to demonstrate the WDT refresh and disable feature.
// Main thread pass a running time into loading while loop for WDT refresh. After it exits,
// disable the WDT timer to stop the countdown.
{
        //-- running  loop for WDT refresh before WDT countdown to 0.
        while(running )
        {
             MiAPI_Watchdog_Refresh();

            //--Simulate a time-consume loading
               loading();
        }

        //--Stop the watchdog for safety when it exits the while loop.
        MiAPI_Watchdog_Disable();

        return 0;
}

int Do_MiAPI_WDT(void)
{
        int ret = MiAPI_OK;
```

```
        HANDLE handleWDT;

        //-- Set up 10 seconds timeout to system reboot
          ret = MiAPI_Watchdog_SetConfig(10, true);

        //-- Start WDT
          MiAPI_Watchdog_Start();

        //-- Create the worker thread
          handleWDT = (HANDLE)_beginthreadex(0, 0, &thread_WDT, &running, 0, 0);

        //-- Wait for worker thread terminated
            WaitForSingleObject(handleWDT, INFINITE);
            CloseHandle(handleWDT);

        //-- Disable the Watchdog
        MiAPI_Watchdog_Disable();

        return ret;
}
```

Appendix A – API Error Codes

General	
0x00	MiAPI_OK
0x01	MiAPI_INIT_FAIL
0x02	MiAPI_NOT_SUPPORT
0x03	MiAPI_UNLOAD_FAIL
0x04	MiAPI_READ_FAIL
0x05	MiAPI_OLD_VERSION *
Watchdog	
0x21	MiAPI_WDT_GET_FAIL
0x22	MiAPI_WDT_SET_FAIL
GPIO	
0x31	MiAPI_GPIO_QUERY_FAIL
0x32	MiAPI_GPIO_MUX_FAIL
0x33	MiAPI_GPIO_SETDIR_FAIL
0x34	MiAPI_GPIO_GETSTATUS_FAIL
0x35	MiAPI_GPIO_SETSTATUS_FAIL
SMBUS	
0x41	SMBUS_TIMEOUT
0x42	SMBUS_INVALID_PARAMETER
0x43	SMBUS_UNSUPPORTED
0x44	SMBUS_BUFFER_TOO_SMALL
0x45	SMBUS_CRC_ERROR
0x46	SMBUS_DEVICE_ERROR
VGA Control	
0x51	MiAPI_VGA_WRONG_RANGE
0x52	MiAPI_VGA_GETBRIGHTNESS_FAIL
0x53	MiAPI_VGA_SETBRIGHTNESS_FAIL
Hardware Monitor	
0x61	MiAPI_GET_CPUFAN_SPEED_FAIL
0x62	MiAPI_SET_CPUFAN_SPEED_FAIL
0x63	MiAPI_GET_SYSFAN_SPEED_FAIL
0x64	MiAPI_SET_SYSFAN_SPEED_FAIL

*** This error code is to identify current mother board might not fully compatible with MiAPI v3.1 specification.**