www.aaeon.com

Appendix A:

AAEON Embedded Controller

API Protocol

Rev 0.4
01/15/2015

Department: AAEON/SDD

Author: Elf Lo

AAEON Technology Inc.
5F, No.135, Lane 235, Pao Chiao Rd.,
Hsin-Tien City, New Taipei 231,
Taiwan, R.O.C.
TEL: +886-2-8919-1234
FAX: +886-2-8919-1056

History

VERSION	DATE	DESCRIPTION
V0.1	11/17/2014	Preliminary Specification V0.1, released by Elf.
V0.2	12/11/2014	Support SMART Fan Function.
V0.3	01/08/2015	Upgrade API_Check() function.
V0.4	01/15/2015	Support SMBUS Block Read/Write.

AAEON Technology Inc.
5F, No.135, Lane 235, Pao Chiao Rd.,
Hsin-Tien City, New Taipei 231,
Taiwan, R.O.C.
TEL: +886-2-8919-1234
FAX: +886-2-8919-1056

Module Features

Function list / Module	GPIO	HW Monitor		SMBus	WDT	I²C
		Voltage Temp	SMART FAN			
COM-CV-B10	√	√	√	√	√	ODM
NanoCOM-BT-A10	√	√	√	√	√	ODM

AAEON Technology Inc.
5F, No.135, Lane 235, Pao Chiao Rd.,
Hsin-Tien City, New Taipei 231,
Taiwan, R.O.C.
TEL: +886-2-8919-1234
FAX: +886-2-8919-1056

1. Introduction

Through ***AAEON EC API*** can communicate with AAEON EC Device by ***AAEON defined I/O Port*** 284h/285h.

AAEON Technology Inc.
5F, No.135, Lane 235, Pao Chiao Rd.,
Hsin-Tien City, New Taipei 231,
Taiwan, R.O.C.
TEL: +886-2-8919-1234
FAX: +886-2-8919-1056

EC API Registers Map Table:

Index	R/W	Description
10h	R/W	Logic Device Number Register
11h	R/W	Function and Device Register
12h	R/W	Configuration Register
13h	R/W	Option Register 0
14h	R/W	Option Register 1
15h	R/W	Option Register 2
16h	R/W	Option Register 3
17h	R/W	Option Register 4
18h	R/W	Option Register 5
19h	R/W	Option Register 6
1Ah	R/W	Option Register 7
1Bh	R/W	Option Register 8
1Ch	R/W	Option Register 9
1Dh	R/W	Option Register A
1Eh	R/W	Option Register B
1Fh	R/W	Option Register C

LDNR — Logic Device Number Register

Index Address: 10h

Bit	R/W	Description
7:0	R/W	**Logic Device Number** Reference with 'Logic Device and Function Map table' for more detail.

FNDR — Function and Device Register

Index Address: 11h

Bit	R/W	Description
7:0	R/W	**Function and Device** Reference with 'Logic Device and Function Map table' for more detail.

CONFR — Configuration Register

Index Address: 12h

Bit	R/W	Description
7:6	-	**Reserved**
5	R/W	**Data Register Read/Write Setting** 1b: Write 0b: Read
4	R/W	**API Active Setting** 1b: Enable API service. (It will be cleared after service activate)
3	R	**Reserved**
2	R	**API Progress status (It will set to b'1' during API Service enabled)** 1: API Progressing 0: None
1	R	**API Operation Fail Status (Return API operating status after API progress)** 1: API Operation fail. 0: None
0	R	**API Operation Done Status (Return API operating status after API progress)** 1: API Operation done 0: None

OPRn — Option Register (0Ch-00h)

Index Address: 1Fh-13h

Bit	R/W	Description
7-0	R/W	API Data Buffer.

AAEON Technology Inc.
5F, No.135, Lane 235, Pao Chiao Rd.,
Hsin-Tien City, New Taipei 231,
Taiwan, R.O.C.
TEL: +886-2-8919-1234
FAX: +886-2-8919-1056

Logic Device and Function Map Table:

LDNR	FNDR	Description
A1h	-	EC Firmware Information Function
	02h	EC Firmware Information
A2h	-	Dynamic Digital Input / Output Function
	00h	Digital Input / Output Mode Configuration
	01h	Digital Input / Output Value Configuration
A5h	-	Hardware Monitor Controller
	00h	System Temperature Information
	02-01h	Native ADC Function
A6h	-	SMART FAN Controller
	02-00h	SMART FAN 3-1 Configuration
A7h		SMBUS/I2C Host Controller
	01h	SMBUS Host Controller
A8h	-	System Protect Function
	00h	Watchdog Configuration

AAEON Technology Inc.
5F, No.135, Lane 235, Pao Chiao Rd.,
Hsin-Tien City, New Taipei 231,
Taiwan, R.O.C.
TEL: +886-2-8919-1234
FAX: +886-2-8919-1056

Logical Device A1h: EC Firmware Information Function

Function and Device 02h: EC Firmware Information

OPRn	R/W	Description
00h	R	Reserved
01h	R	Year Register (High Byte)
02h	R	Year Register (Low Byte)
03h	R	Month Register
04h	R	Date Register
0Ch-05h	R	EC firmware version Register.

Option Register 01h: Year Register (High Byte)

Bit	R/W	Default	Description
7-0	R	-	**Indicate the build date of years (High Byte)** ex. Return 20h as year 2012.

Option Register 02h: Year Register (High Byte)

Bit	R/W	Default	Description
7-0	R	-	**Indicate the build date of years (Low Byte)** ex. Return 12h as year 2012.

Option Register 03h: Month Register

Bit	R/W	Default	Description
7-0	R	-	**Indicate the build date of Month.** ex. Return 10h as October.

AAEON Technology Inc.
5F, No.135, Lane 235, Pao Chiao Rd.,
Hsin-Tien City, New Taipei 231,
Taiwan, R.O.C.
TEL: +886-2-8919-1234
FAX: +886-2-8919-1056

Option Register 04h: Date Register

Bit	R/W	Default	Description
7-0	R	-	**Indicate the build date of date.** ex. Return 21h as twenty-first.

Option Register 0Ch-05h: EC firmware version Register

Bit	R/W	Default	Description
7-0	R	-	**Indicate EC Firmware Version String in ASCII Code.** ex. CM77BE10 Return 43h of Option Register 05. Return 47h of Option Register 06. ….. Return 31h of Option Register 0B. Return 30h of Option Register 0C.

Logical Device A2h: Dynamic Digital Input/Output Function

Function and Device 00h: Digital Input/Output Mode Configuration

OPRn	R/W	Description
00h	R	Digital Input/Output amount
08h-01h	R/W	Digital Input/Output Pin Mode Register. (DIO 63-55) … (DIO 7-0)
0Ch-09h	-	Reserved

Option Register 00h: Digital Input/Output amount

Bit	R/W	Default	Description
7-0	R/W	-	The number of “Digital Input/Output” supports in the project.

Option Register 08h-01h: Digital Input/Output Pin Mode Register

Bit	R/W	Default	Description
7-0	R/W	-	Indicate the Digital Input/Output Pin Operating mode for (DIO 63-55) … (DIO 7-0). 0: Output 1: Input

AAEON Technology Inc.
5F, No.135, Lane 235, Pao Chiao Rd.,
Hsin-Tien City, New Taipei 231,
Taiwan, R.O.C.
TEL: +886-2-8919-1234
FAX: +886-2-8919-1056

Function and Device 01h: Digital Input/Output Value Configuration

OPRn	R/W	Description
00h	R	Digital Input/Output amount
08h-01h	R/W	Digital Input/Output Data Register
0Ch-09h	-	Reserved

Option Register 00h: Digital Input/Output amount

Bit	R/W	Default	Description
7-0	R/W	-	The number of "Digital Input/Output" supports in the project.

Option Register 08h-01h: Digital Input/Output Data Register

Bit	R/W	Default	Description
7-0	R/W	-	**For each individual DIO pin (DIO 63-55) … (DIO 7-0).** **In the output mode, reading returns the last written date.** **In the input mode, reading this register returns the pin level status.** 0: Pin Level States Low. 1: Pin Level States High.

Logical Device A5h: Hardware Monitor Controller

Function and Device 00h: Hardware Monitor

OPRn	R/W	Description
00h	R	System Thermal Sensor amount.
0Ch-01h	R	System Temperature Register.

Option Register 00h: System Thermal Sensor amount

Bit	R/W	Default	Description
7-0	R	-	The number of “System Thermal Sensor” supports in the project.

Option Register 0Ch-01h: System Temperature Register

Bit	R/W	Default	Description
7-0	R/W	-	The System Temperature Register data format is a binary twos complement signed byte format as follow: Returns 7Dh when system temperature reaches 125℃. Returns 19h when system temperature reaches 25℃. Returns 01h when system temperature reaches 1℃. Returns FFh when system temperature reaches -1℃. Returns E7h when system temperature reaches -25℃. Returns C9h when system temperature reaches -55℃.

Function and Device 01h: Native ADC Function channel 1 to 4

OPRn	R/W	Description
00h	R	Reserved
01h-02h	R	ADC value of channel 1.
03h-04h	R	ADC value of channel 2
05h-06h	R	ADC value of channel 3
07h-08h	R	ADC value of channel 4

Function and Device 02h: Native ADC Function channel 5 to 8

OPRn	R/W	Description
00h	R	Reserved
01h-02h	R	ADC value of channel 5
03h-04h	R	ADC value of channel 6
05h-06h	R	ADC value of channel 7
07h-08h	R	ADC value of channel 8

Option Register 08h-01h: ADC value of channels

Bit	R/W	Default	Description
15-10	R	-	Reserved
9-0	R	-	It is a 10-bit unsigned integer for ADC voltage input in each channel. 3.0V is returned as 3FFh. 1.5V is returned as 200h. 0.0V is returned as 000h.

SMART Fan Controller

SMART Fan Mode is dynamic changed output PWM Duty Cycle depending on reference temperature, when temperature exceeds “Start Temperature” and the temperature between Start Limit and Full Limit, the output PWM Duty Cycle value will be as below:

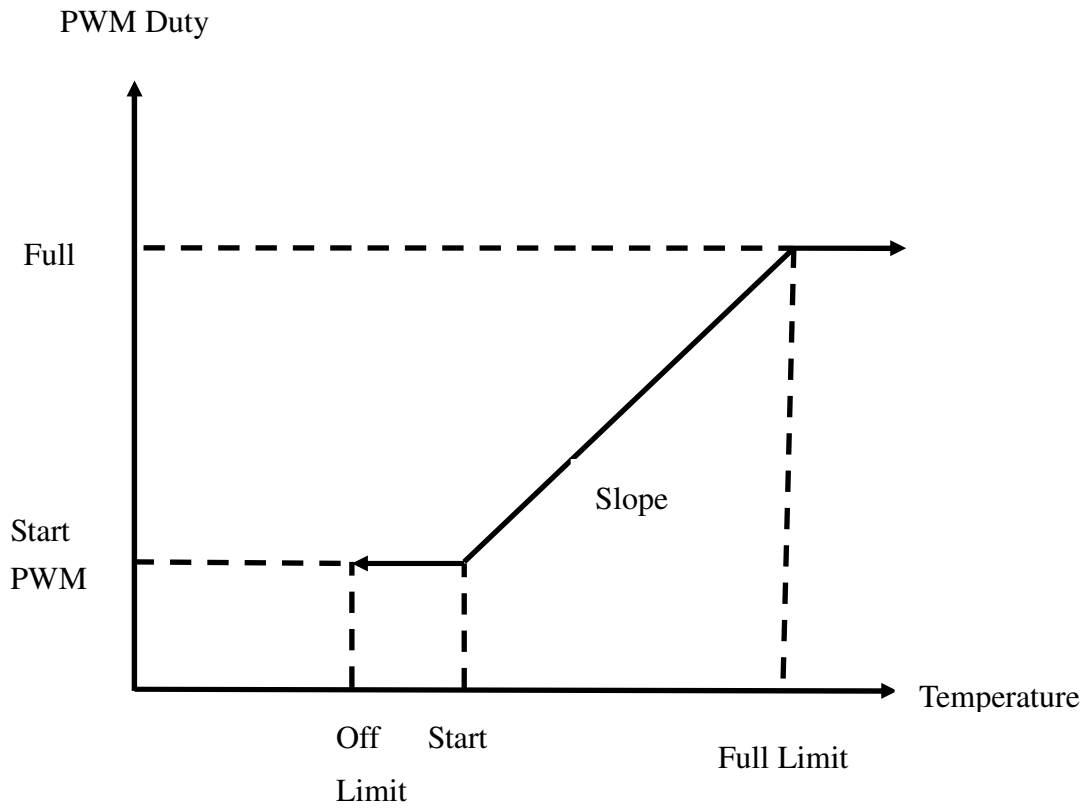
Duty Cycle % = Start PWM + (Reading Temperature – Start Temperature) x Slope;

If SMART trigged and reading temperature under “Start Temperature” and the temperature between Start Limit and Off Limit, the output PWM Duty Cycle value will equal Start PWM:

Duty Cycle % = Start PWM;

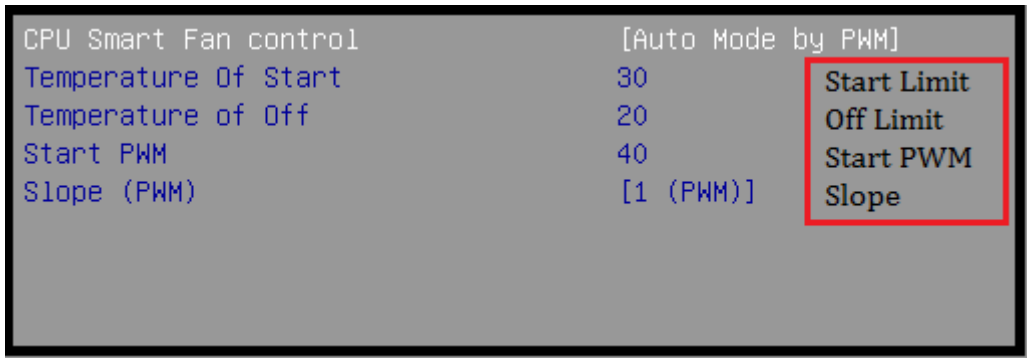
If reading temper under Off Limit, the output PWM Duty Cycle value will equal Zero.

Duty Cycle % = 0;



AAEON Technology Inc.
5F, No.135, Lane 235, Pao Chiao Rd.,
Hsin-Tien City, New Taipei 231,
Taiwan, R.O.C.
TEL: +886-2-8919-1234
FAX: +886-2-8919-1056

Options in BIOS setup menu:



Note:

(1) The "Full Limit" value is generated by "Start Limit" , "Start PWM" and "Slope" 's setting.

(2) Different fan requires different PWM driving to "spin up". Please check your fan specification before use.

Logical Device A6h: SMART FAN Controller

Function and Device 02h-00h: SMART FAN Controller

OPRn	R/W	Description
00h	R/W	SMART FAN1 Configuration
01h	R/W	SMART FAN2 Configuration
02h	R/W	SMART FAN3 Configuration
0Ch-03h	-	Reserved

Option Register 00h: SMART FAN amount

Bit	R/W	Default	Description
7-0	R/W	0h	Supported SMART FAN numbers.

Option Register 03h: FAN Configuration

Bit	R/W	Default	Description
7	R/W	0b	FAN mode selection. 0: Manual Mode 1: Auto Mode.
6	R/W	0b	Inverting of output PWM signal. 0: High Active 1: Low Active
5-4	-	-	Reserved
3-0	R/W	0b	Select which monitoring thermal sensor.

Option Register 04h: FAN Temperature Of Start

Bit	R/W	Default	Description
7-0	R/W	0h	It's triggered to start output PWM single when the monitoring temperature exceeds this value. (Use for Auto Mode)

Option Register 05h: FAN Temperature Of Off

Bit	R/W	Default	Description
7-0	R/W	0h	It's triggered to stop output PWM single when the monitoring temperature exceeds this value. (Use for Auto Mode)

Option Register 06h: FAN Start Of PWM

Bit	R/W	Default	Description
7-0	R/W	0h	The initial output PWM value when Temperature Of Start is triggered. (Use for Auto Mode)

Option Register 07h: FAN Slope Of Start

Bit	R/W	Default	Description
7-0	R/W	0h	It related output PWM when Temperature Of Start is triggered. (Use for Auto Mode)

Option Register 08h: FAN Output PWM

Bit	R/W	Default	Description
7-0	R/W	0h	Selection output PWM value of Manual Mode.

Logical Device A7h: SMBUS Host Controller

Function and Device 01h: SMBUS/I2C Host Controller

OPRn	R/W	Description
00h	R	Reserved
01h	R/W	Host Control Register
02h	R/W	Transmit Slave Address Register
03h	R/W	Host Command Register
04h	R/W	Data Low Byte Register
04h	R/W	Byte Count Register (Block Mode)
05h	R/W	Data High Byte Register
0Ch-05h	R/W	Data Byte 8-1 Register (Block Mode)

Option Register 01h: Host Control Register

Bit	R/W	Default	Description
7-3	R/W	0	Reserved
4-2	R/W	0	001: Send Byte / Receive Byte 010: Write Byte / Read Byte 011: Write Word / Read Word 101: Block Write / Block Read Other: reserve
1-0	R/W	0	Reserved

AAEON Technology Inc.
5F, No.135, Lane 235, Pao Chiao Rd.,
Hsin-Tien City, New Taipei 231,
Taiwan, R.O.C.
TEL: +886-2-8919-1234
FAX: +886-2-8919-1056

Option Register 02h: Transmit Slave Address Register

Bit	R/W	Default	Description
7-1	R/W	00h	Address of the targeted slave.
0	R/W	0b	Direction of the host transfer. 0: Write 1: Read

Option Register 03h: SMBUS/I2C Host Command Register

Bit	R/W	Default	Description
7-0	R/W	-	These bits are transmitted in the command field of the SMBus protocol.

Option Register 04h: Data Low Byte Register

Bit	R/W	Default	Description
7-0	R/W	00h	The Data Byte Low (first transaction Data Byte) of the SMBus protocol.

Option Register 04h: Byte Count Register (Block Mode)

Bit	R/W	Default	Description
7-0	R/W	00h	How many Byte Data of Block Read/Write process.

Option Register 05h: Data 1 Register

Bit	R/W	Default	Description
7-0	R/W	00h	The Data Byte High of the SMBus protocol.

Option Register 0Ch-05h: Data Byte 8-1 Register (Block Mode)

Bit	R/W	Default	Description
7-0	R/W	00h	The Data Byte High of the SMBus protocol.

AAEON Technology Inc.
5F, No.135, Lane 235, Pao Chiao Rd.,
Hsin-Tien City, New Taipei 231,
Taiwan, R.O.C.
TEL: +886-2-8919-1234
FAX: +886-2-8919-1056

Function and Device 05h: SMBUS Block Byte Buffer1

OPRn	R/W	Description
00h	R	Reserved
0Ch-01h	R/W	Data Byte 20-9 Register (Block Mode)

Option Register 0Ch-01h: Data Byte 20-9 Register (Block Mode)

Bit	R/W	Default	Description
7-0	R/W	00h	The Data Byte High of the SMBus protocol.

Function and Device 06h: SMBUS Block Byte Buffer2

OPRn	R/W	Description
00h	R	Reserved
0Ch-01h	R/W	Data Byte 32-21 Register (Block Mode)

Option Register 0Ch-01h: Data Byte 32-21 Register (Block Mode)

Bit	R/W	Default	Description
7-0	R/W	00h	The Data Byte High of the SMBus protocol.

Logical Device A8h: System Protection

Function and Device 00h: Watchdog Configuration

OPRn	R/W	Description
00h	R/W	Watchdog Timer Configuration Register
01h	R/W	Watchdog Configuration Register.
0C-02h	-	Reserved

Option Register 00h: Watchdog Timer Register

Bit	R/W	Default	Description
7-0	R/W	00h	**Watchdog timer unit. (1~255)** 0: Stop watchdog counter.

Option Register 01h: Watchdog Configuration Register

Bit	R/W	Default	Description
7:2	R/W	-	Reserved
1	R/W	0b	**Pulse Type** 0 = High Active 1 = Low Active
0	R/W	0b	**Time Setting** 0 = Second 1 = Minute

Sample code for common function:

```
//---------------------------------------------------------------------------
// Type define
//---------------------------------------------------------------------------
typedef unsigned char             BYTE;     // 8bit
typedef unsigned short int        WORD;     //16bit
//---------------------------------------------------------------------------
// Common define
//---------------------------------------------------------------------------
#define EcPortIndex         0x284
#define EcPortData          0x285
#define EcReg_Dev           0x10
#define EcReg_Type          0x11
#define EcReg_Conf          0x12
#define EcReg_Dat0          0x13
#define EcReg_Dat1          0x14
#define EcReg_Dat2          0x15
#define EcReg_Dat3          0x16
#define EcReg_Dat4          0x17
#define EcReg_Dat5          0x18
#define EcReg_Dat6          0x19
#define EcReg_Dat7          0x1A
//---------------------------------------------------------------------------
#define CMD_Read            0x10
#define CMD_Write           0x30
//---------------------------------------------------------------------------
#define Flag_Fail           0x02
#define Flag_Done           0x01
//---------------------------------------------------------------------------
// Common Function.
//---------------------------------------------------------------------------
BYTE EcReadByte(BYTE offset)
{
        outportb(EcPortIndex, offset);
        return(inportb(EcPortData));
}

void EcWriteByte(BYTE offset,BYTE data)
{
        outportb(EcPortIndex, offset);
        outportb(EcPortData, data);
}

void API_Check()
{
   WORD timeout;
   BYTE value;

        for (timeout = 0; timeout < 20; timeout++)
        {
                value =EcReadByte(EcReg_Conf);
                sleep(1);

                if (value&(Flag_Fail|Flag_Done))
                {
                        if (value&Flag_Fail)
                        {     printf("Process fail.\n");     }
                        return;
                }
        }
        printf("Time Out, EC response time about 20ms.\n");

}
```

Sample code for read EC Firmware Information:

```
//---------------------------------------------------------------------------
// Firmware Information function define
//---------------------------------------------------------------------------
#define API_Ver_Dev             0xA1
#define API_Ver_Type            0x02
#define Ver_Reg_Year1           EcReg_Dat1
#define Ver_Reg_Year2           EcReg_Dat2
#define Ver_Reg_Month           EcReg_Dat3
#define Ver_Reg_Day             EcReg_Dat4
#define Ver_Reg_Char1           EcReg_Dat5
#define Ver_Reg_Char2           EcReg_Dat6
#define Ver_Reg_Char3           EcReg_Dat7
#define Ver_Reg_Char4           EcReg_Dat8
#define Ver_Reg_Char5           EcReg_Dat9
#define Ver_Reg_Char6           EcReg_DatA
#define Ver_Reg_Char7           EcReg_DatB
#define Ver_Reg_Char8           EcReg_DatC
//---------------------------------------------------------------------------
// Main function
//---------------------------------------------------------------------------
void main()
{
        BYTE i;
        WORD value;

        EcWriteByte(EcReg_Dev,API_Ver_Dev);     //Setting Device
        EcWriteByte(EcReg_Type,API_Ver_Type);   //Setting Type
        EcWriteByte(EcReg_Conf,CMD_Read);       //Read Command
        API_Check();


        printf("Released Year:%x%x.\n",EcReadByte(Ver_Reg_Year1),EcReadByte(Ver_Reg_Year2));
        printf("Released Date:%x/%x.\n",EcReadByte(Ver_Reg_Month),EcReadByte(Ver_Reg_Day));
        printf("Released Version:");

        for (i=0;i<8;i++)
        {
                printf("%c",EcReadByte(Ver_Reg_Char1+i));
        }
                printf(".\n");
}}
```

Sample code for Hardware Monitor function:

```
//---------------------------------------------------------------------------
// Hardwre Monitor functiuon define
//---------------------------------------------------------------------------
#define API_HWM_Dev             0xA5
#define API_HWM_Type0           0x00 //System Temperature Information
#define API_HWM_Type1           0x01 //Native ADC Fucntion
#define API_HWM_Type2           0x02 //Native ADC Fucntion
#define HWM_Reg_Cnt             EcReg_Dat0
#define HWM_Reg_Temp1           EcReg_Dat1
#define HWM_Reg_Temp2           EcReg_Dat2
#define HWM_Reg_ADC1            EcReg_Dat1
#define HWM_Reg_ADC2            EcReg_Dat3
#define HWM_Reg_ADC3            EcReg_Dat5
#define HWM_Reg_ADC4            EcReg_Dat7

//---------------------------------------------------------------------------
// Main function
//---------------------------------------------------------------------------
void main()
{
        BYTE i;
        WORD value;

        EcWriteByte(EcReg_Dev,API_HWM_Dev);         //Setting Device
        EcWriteByte(EcReg_Type,API_HWM_Type0);      //Setting Type
        EcWriteByte(EcReg_Conf,CMD_Read);           //Read Command
        API_Check();

        printf("There are %d temperautre sensor is supported.\n",EcReadByte(HWM_Reg_Cnt));
        printf("System Temperature 1, %d degree \n",EcReadByte(HWM_Reg_Temp1));
        printf("System Temperature 2, %d degree \n",EcReadByte(HWM_Reg_Temp2));


        EcWriteByte(EcReg_Dev,API_HWM_Dev);         //Setting Device
        EcWriteByte(EcReg_Type,API_HWM_Type1);      //Setting Type
        EcWriteByte(EcReg_Conf,CMD_Read);           //Read Command
        API_Check();

        for (i=0;i<4;i++)
        {
                value =     (WORD)EcReadByte(HWM_Reg_ADC1+i*2)<<8| \
                            EcReadByte(HWM_Reg_ADC1+1+i*2);

                            printf("ADC channel %d value 0x%x.\n",i+1,value);

                switch(i)
                {
                        case 0:
                        case 1:
                                printf(" Use for thermal sensor.\n");
                                break;

                        case 2:
                                value = (DWORD)value*100/341;
                                printf("ADC channel %d +1.8V, value = %d.%d%dV. \n", \
                                i+1,value/100,value%100/10,value%100%10);
                                break;

                        case 3: // R1=2K, R2=3K.
                                value = (DWORD)value*5/3*100/341;
                                printf("ADC channel %d +5V, value = %d.%d%dV. \n", \
                                i+1,value/100,value%100/10,value%100%10);
                                break;

                        default:
```

```
                        break;

                }
        }

        EcWriteByte(EcReg_Dev,API_HWM_Dev);          //Setting Device
        EcWriteByte(EcReg_Type,API_HWM_Type2);       //Setting Type
        EcWriteByte(EcReg_Conf,CMD_Read);            //Read Command
        API_Check();

        for (i=0;i<4;i++)
        {
                value = (WORD)EcReadByte(HWM_Reg_ADC1+i*2)<<8|\
                EcReadByte(HWM_Reg_ADC1+1+i*2);
                printf("ADC channel %d value 0x%x.\n",i+5,value);

                switch(i)
                {
                        case 0:      //R1=100,R2=1K.
                                value = (DWORD)value*11/10*100/341;
                                printf("ADC channel %d +3.3V, value = %d.%d%dV. \n", \
                                i+5,value/100,value%100/10,value%100%10);
                                break;

                        case 1: //R1=100,R2=1M.
                                value = (DWORD)value*100/341;
                                printf("ADC channel %d +1.5V, value = %d.%d%dV. \n", \
                                i+5,value/100,value%100/10,value%100%10);
                                break;

                        case 2:
                                value = (DWORD)value*100/341;
                                printf("ADC channel %d +1.05V, value = %d.%d%dV. \n", \
                                i+5,value/100,value%100/10,value%100%10);
                                break;

                        case 3: // R1=2K, R2=3K.
                                value = (DWORD)value*100/341;
                                printf("ADC channel %d VGFX, value = %d.%d%dV. \n", \
                                i+5,value/100,value%100/10,value%100%10);
                                break;

                        default:
                                break;

                }
        }
}
```

Sample code for WDT function setting:

```
//---------------------------------------------------------------------------
// WDT function define
//---------------------------------------------------------------------------
#define API_WDT_Dev             0xA8
#define API_WDT_Type            0x00
//---------------------------------------------------------------------------
// Main function
//---------------------------------------------------------------------------
void main()
{
        BYTE WDT_count;
        BYTE WDT_Conf;

        EcWriteByte(EcReg_Dev,API_WDT_Dev);         //Setting Device
        EcWriteByte(EcReg_Type,API_WDT_Type);       //Setting Type
        EcWriteByte(EcReg_Conf,CMD_Read);           //Read Command

        API_Check();
        WDT_count = 0x05;                           // 5 units
        WDT_Conf    = 0x00;                         // bit0, 0: Second.
                                                    // bit0, 1: Minute.
        EcWriteByte(EcReg_Dev,API_WDT_Dev);         // Setting Device
        EcWriteByte(EcReg_Type,API_WDT_Type);       // Setting Type
        EcWriteByte(EcReg_Dat0,WDT_count);          // Setting watchdog timer
                                                    // Set to zero will stop watchdog function.
        EcWriteByte(EcReg_Dat1,WDT_Conf);           // Setting watchdog unit
        EcWriteByte(EcReg_Conf,CMD_Write);          // Read Command
        API_Check();
}
```

Sample code for dynamic Digital IO function:

```
//---------------------------------------------------------------------------
// DIO functiuon define
//---------------------------------------------------------------------------
#define API_DIO_Dev         0xA2
#define API_DIO_Type        0x00
#define API_DIO_Ctrl        0x01
//---------------------------------------------------------------------------
// Main function
//---------------------------------------------------------------------------
void main()
{

        BYTE value;

        // Setting DIO direction
        EcWriteByte(EcReg_Dev,API_DIO_Dev);             //Setting Device
        EcWriteByte(EcReg_Type,API_DIO_Type);           //Setting Type
        EcWriteByte(EcReg_Conf,CMD_Read);               //Read Command
        API_Check();

        value = EcReadByte(EcReg_Dat1);                 // 0: Output, 1: Input
        printf("The DIO direction setting. 0x%x \n",value);

        value = value&0xfe;
        EcWriteByte(EcReg_Dat1,value);                  // Setting DIO0 to output
        EcWriteByte(EcReg_Conf,CMD_Write);              // Write Command
        API_Check();

        EcWriteByte(EcReg_Conf,CMD_Read);               //Read Command
        API_Check();
        printf("The DIO direction setting. 0x%x \n",EcReadByte(EcReg_Dat1));

        // Setting DIO value;
        EcWriteByte(EcReg_Dev,API_DIO_Dev);             //Setting Device
        EcWriteByte(EcReg_Type,API_DIO_Ctrl);           //Setting Control
        EcWriteByte(EcReg_Conf,CMD_Read);               //Read Command
        API_Check();

        value = EcReadByte(EcReg_Dat1);                 // 0: Low, 1: High
        printf("The polarity of DIO 0x%x \n",value);
        value = value&0xfe;
        EcWriteByte(EcReg_Dat1,value);                  // Setting DIO0 output low
        EcWriteByte(EcReg_Conf,CMD_Write);              // Write Command
        API_Check();

        EcWriteByte(EcReg_Conf,CMD_Read);               //Read Command
        API_Check();
        printf("The polarity of DIO 0x%x \n",EcReadByte(EcReg_Dat1));
}
```

Sample code for SMBUS/I2C Host Controller function:

```
//---------------------------------------------------------------------------
// SMBUS/I2C Host Controller function define
//---------------------------------------------------------------------------
#define API_I2C_Dev             0xA7
#define API_I2C_Type            0x01
#define I2C_Reg_Ctrl            EcReg_Dat1
#define I2C_Reg_Addr            EcReg_Dat2
#define I2C_Reg_Cmd             EcReg_Dat3
#define I2C_Reg_Dat0            EcReg_Dat4
#define I2C_Reg_Dat1            EcReg_Dat5
//---------------------------------------------------------------------------
#define API_I2C_SendByte        0x04
#define API_I2C_Receive         0x04
#define API_I2C_ReadByte        0x08
#define API_I2C_WriteByte       0x08
#define API_I2C_ReadWord        0x0C
#define API_I2C_WriteWord       0x0C
//---------------------------------------------------------------------------
#define API_I2C_Read            0x01
#define API_I2C_Write           0x00
//---------------------------------------------------------------------------
// Main function
//---------------------------------------------------------------------------
void main()
{
        BYTE SLV_Addr;
        WORD value;

        EcWriteByte(EcReg_Dev,API_I2C_Dev);                     //Setting Device
        EcWriteByte(EcReg_Type,API_I2C_Type);                   //Setting Type

// Sample code for access AD5247 on ECB-917T
        SLV_Addr = 0x2E<<1;                                     //Refer to AD5247 spec.
        printf("ECB-917's DAC module test \n");
        EcWriteByte(I2C_Reg_Addr,(SLV_Addr|API_I2C_Read));      //Setting AD5247 Slave address
        EcWriteByte(I2C_Reg_Ctrl,API_I2C_Receive);              //Setting to Receive function.
        EcWriteByte(EcReg_Conf,CMD_Write);                      //Write Command
        API_Check();

        printf("Current output DAC value. 0x%x \n",EcReadByte(I2C_Reg_Dat0));
        printf("Select output DAC value, range 0x00~0x7F. \n");
        scanf("%x", &value);
        printf("Setting DAC value: %x \n",value);

        EcWriteByte(I2C_Reg_Addr,(SLV_Addr|API_I2C_Write));     //Setting AD5247 Slave address
        EcWriteByte(I2C_Reg_Ctrl,API_I2C_SendByte);             //Setting to Send Byte function.
        EcWriteByte(I2C_Reg_Cmd,value);                         //Setting to Send Byte function.
        EcWriteByte(EcReg_Conf,CMD_Write);                      //Write Command
        API_Check();

        EcWriteByte(I2C_Reg_Addr,(SLV_Addr|API_I2C_Read));      //Setting AD5247 Slave address
        EcWriteByte(I2C_Reg_Ctrl,API_I2C_Receive);              //Setting to Receive function.
        EcWriteByte(EcReg_Conf,CMD_Write);                      //Write Command
        API_Check();

        printf("Current DAC output value. 0x%x \n",EcReadByte(I2C_Reg_Dat0));

}
```