Benutzerhandbuch/ User's Manual

Kontron SuperVision

kontron

If it's Embedded, it's Kontron.

Technical Manual

Revision. 0.2

Revision History			
Rev.	**Description**	**Date**	**Author**
0.1	Initial Release	24.07.04	M. Koch
0.2	Upgrade	12.09.06	M. Koch

	Date	Name	Department
Prepared:	19.09.06	Michael Koch	TH (KECR)
Checked:			
Approved:			

Kontron Embedded Computers GmbH
Werner-von-Siemens-Str. 1
93426 Roding / Germany

Table of Contents:

1.0 Function Descriptions

The SuperVision-IC provides many useful features such as watchdog timer, Operating-Time Counter, Power-on Counter, voltage metering, temperature metering, fan speed metering, and release number. A protected EEPROM memory is as an additional feature available.

Depending on the board/system used, the SuperVision-IC provides either the basic features or the basic features and the enhanced features. All basic features are implemented in the Main-Revision 1. With the Main-Revision 2, the enhanced features are also available.

The following chart shows the relationship between the revision, board/system, and available features.

Board/System	Main-Revision	Basic	Enhanced	PIC
PCI954	1	●		PIC16F818
STX	1	●		PIC16F818
ETX	1	●		PIC16F818
ETXExpress	2	●	●	PIC16F913

Via the SM-Bus can be accessed the internal registers of the PIC.
The SM-Bus address is **0x50**[1]. (Decimal: 80)

Boards based on ETX or ETXExpress modules can access the SM-Bus via JIDA32 Interface. The JIDA32 Interface has SM-Bus BIOS support and works with operating the following operating systems: Windows® 9x, Windows® NT, Windows® 2000, Windows® XP, Windows® CE, and Linux 2.2/2.4.

If you are using the PCI954 or STX-systems, please contact Kontron support for special SM-Bus drivers.

[1] 0x?? means a values in hexadecimal.

The following chart shows an overview of the PIC-Registers:

Register:	Short-Description:	Basic	Enhanced	Main-Revision
00	Watchdog High-Register	•	•	1 , 2
01	Watchdog Low-Register	•	•	1 , 2
02	Operating-Time Counter Byte 0	•	•	1 , 2
03	Operating-Time Counter Byte 1	•	•	1 , 2
04	Operating-Time Counter Byte 2	•	•	1 , 2
05	Power-On Counter Byte 0	•	•	1 , 2
06	Power-On Counter Byte 1	•	•	1 , 2
07	Measured 12.0 Volts	•	•	1 , 2
08	Measured 5.0 Volts	•	•	1 , 2
09	Measured 3.3 Volts	•	•	1 , 2
0A	Board Temperature	•	•	1 , 2
0B	PIC Program Version	•	•	1 , 2
0C	Fan-Speed 1	•	•	1 , 2
0D	AD-Channel Error Byte	•	•	1 , 2
0E	Error Register	•	•	1 , 2
0F	Dummy Register	•	•	1 , 2
10	Protected-Data Byte 0	•	•	1 , 2
11	Protected-Data Byte 1	•	•	1 , 2
12	Protected-Data Byte 2	•	•	1 , 2
13	Protected-Data Byte 3	•	•	1 , 2
14	Protected-Data Byte 4	•	•	1 , 2
15	Protected-Data Byte 5	•	•	1 , 2
16	Protected-Data Byte 6	•	•	1 , 2
17	Protected-Data Byte 7	•	•	1 , 2
18	Protected-Data Byte 8	•	•	1 , 2
19	Protected-Data Byte 9	•	•	1 , 2
1A	Protected-Data Byte A	•	•	1 , 2
1B	Protected-Data Byte B	•	•	1 , 2
1C	Protected-Data Byte C	•	•	1 , 2
1D	Protected-Data Byte D	•	•	1 , 2
1E	Protected-Data Byte E	•	•	1 , 2
1F	Protected-Data Byte F	•	•	1 , 2
20	Protected Bytes Key	•	•	1 , 2
21	Main-Program Revision		•	2
22	Measured Battery voltage		•	2
23	Fan-Speed 2		•	2
24	FAN-Settings		•	2
25	AD-Channel Error Byte 2		•	2
26	reserved		•	2
27	Board Power State		•	2
28	ON-Timer Second		•	2
29	ON-Timer Minute		•	2
2A	ON-Timer Hour 1		•	2
2B	ON-Timer Hour 2		•	2
2C	Status Monitoring		•	2
2D	Watchdog Count High-Register		•	2
2E	Watchdog Count Low-Register		•	2

1.1 PIC-Program Revision

Reading these registers returns the current version of the PIC-Program.

Register	Access	Name	Basic	Enhanced
0x0B	Read only	PIC Program Version	•	•

Register	Access	Name	Basic	Enhanced
0x21	Read only	PIC Main Program Revision	delivers 255	•

The Register (0x21) “Main Program Revision” is only available with the enhanced instruction set (Main Program Revision 2).

A board/system, with only basic instruction set returns 255.
The return value 255 is per definition Main Program Revision 1.

Example:

```
// Main_Revision 1 and Main_Revision 2

    Main_Revision= I2C_Read(0x50,0x21);          // Read the Main_Program_Revision

    if Main_Revison==255
            {
            Main_Revison=1;
            }

    Pic_Version =   I2C_Read(0x50,0x0B);         // PIC_Program_Version

    printf("Main Revison:%d Version:%d“,Main_Revison,Pic_Version);
```

1.2 Watchdog

The SuperVision-IC provides a watchdog component which resets the system when the application stops responding. Setting the Watchdog-Registers activates this function.

Registers associated with Watchdog operations:

Register	Access	Name	Default:	Basic	Enhanced
0x00	Read / Write	Watchdog High Register	1111 1111	•	•
0x01	Read / Write	Watchdog Low Register	1111 1111	•	•
0x2D	Read	Watchdog High Count Re.	1111 1111		•
0x2E	Read	Watchdog Low Count Re.	0000 0011		•

Watchdog High/Low Register

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Watchdog High Register (Register 01)								**Watchdog Low Register (Register 00)**							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WDT Disable	not used					Watchdog-Time									

Watchdog Count High/Low Register

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Watchdog High Count Re. (Register 2D)								**Watchdog Low Count Re. (Register 2E)**							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
not used						Watchdog-Time									

Watchdog Time: This is the watchdog timeout in seconds. (10 bits max. 1024 seconds)
WDT Disable-Bit: **0** Watchdog **enable**, **1** Watchdog disable

Writing to any Watchdog-Register (Register 00/01) retriggers the Watchdog-Timer.

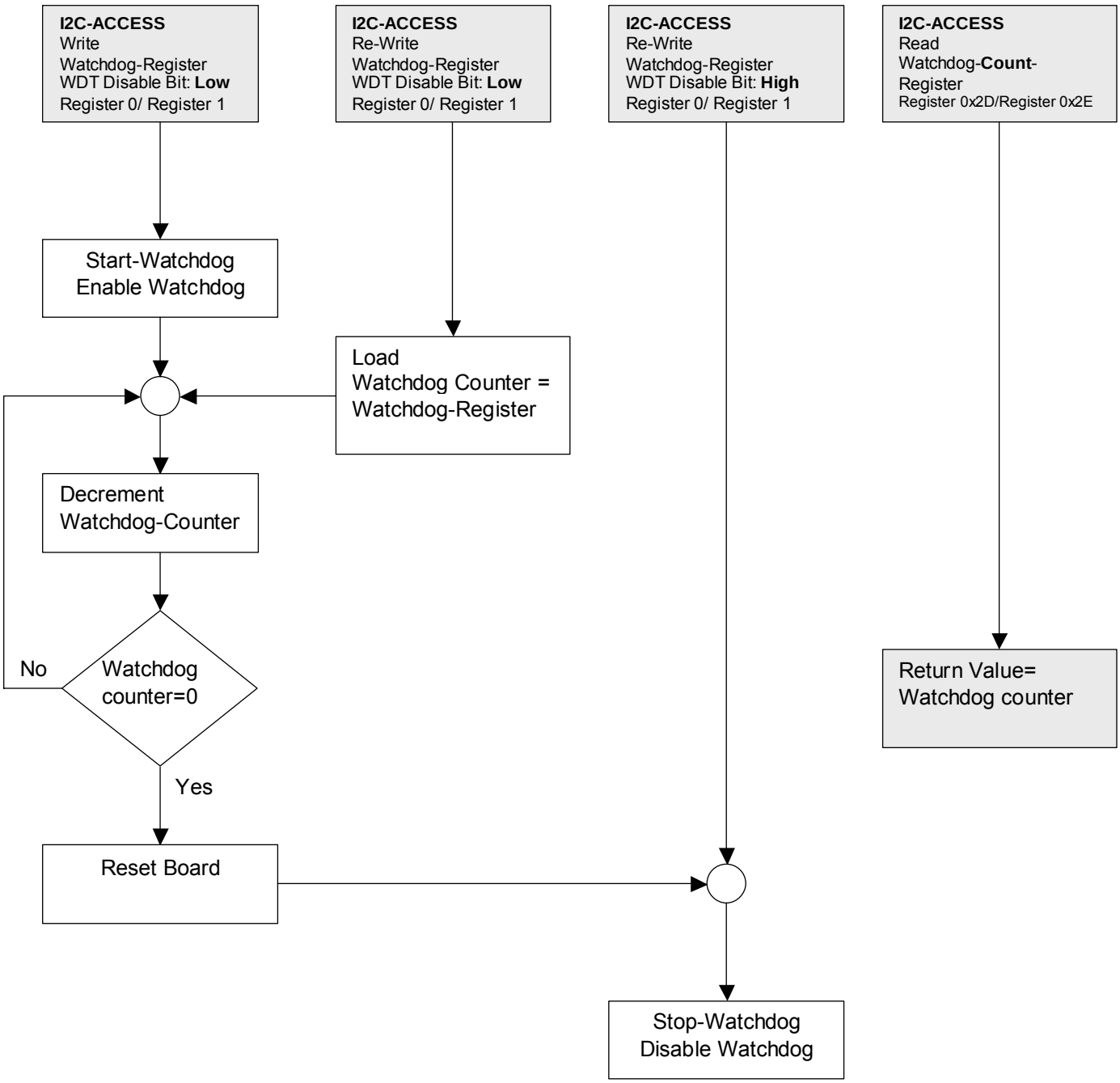
Reading any Watchdog-Register (Register 00/01) retrieves the current watchdog parameters.

Reading any Watchdog-Count-Register (Register 2D/2E) retrieves the current watchdog counter.

Recycling the board/system power disables the watchdog.

After a watchdog reset the watchdog will be disabled.

Flow Diagram:



A write access to SuperVison-IC watchdog registers (register 0/1) sets these with the values.

Example:

I2C_Write(0x50, 0x01,30);
I2C_Write(0x50, 0x00, 00);

The SuperVision "Watchdog-Time" then counts down once a second, provided that the WDT disable bit is 0.

There are 4 possibilities:

A) Condition:
SuperVision-"Watchdog Time "> 0

Action:
Write access to the watchdog register (register 0/1) and watchdog register 1=0xxx xxxx (enable Watchdog).

Example:

I2C_Write(0x50, 0x01, 30);
I2C_Write(0x50, 0x00, 00);

Result:
The SuperVision-IC "watchdog time" will be restarted with the passed values (here 30d).

B) Condition:
SuperVision "Watchdog Time ">0

Action:
Write access to the watchdog register 1 =1xxx xxxx (disable watchdog).

Example:

I2C_Write(0x50, 0x00, 255);

Result:
The SuperVision watchdog will be stopped → No system reset.
Restart SuperVision watchdog → rewrite SuperVision watchdog registers (0x00, 0x01).

C) Condition:
SuperVision-"Watchdog-Time ">0 and read access to the watchdog-register (via I2C)

Example:

Register1=I2C_Read(0x50, 0x01);
Register0=I2C_Read(0x50, 0x00);
printf("Watchdog High Register %d Watchdog Low Register";Register1,Register0);
Result: Register1=30; Register2=0

Result:
Reading any SuperVision watchdog register returns, the current watchdog parameter
(In this case: 30 seconds, look at item A)

D) Condition:
SuperVision-"Watchdog-Time " = 0

Result:
The watchdog resets the system.

Examples:

Example 1

- Set watchdog timeout to 53 seconds:
- Watchdog time = 1 * 53 + 256 * 0 = 53 seconds
- Every pass of the do-while loop retrigger the watchdog with 53 seconds

```
// Main Revision 1 and Main Revision 2


do {

        I2C_Write(0x50,0x01,53);        // Write to watchdog low register the Value 53
        I2C_Write(0x50,0x00,00);        // Write to watchdog high register the Value 00

        // Your program code
        …


        …
        // Your program code

while (x!=END_CONDITION)
```

Example 2

- Set watchdog timeout to 584 seconds:
- Watchdog time = 1 * 72 + 256 * 2 = 584 seconds
- After processing “your program code” the watchdog will be disabled

```
// Main Revision 1 and Main Revision 2


        I2C_Write(0x50,0x01,72);        // Write to watchdog low register the value 72
        I2C_Write(0x50,0x00,02);        // Write to watchdog high register the value 02

        // Your program code
        …


        …
        // Your program code


        I2C_Write(0x50,0x00,128);       // Write to watchdog high register the value 128
                                        // Disable bit : 1000 0000 = 128 decimal
```

Example 3

- Set watchdog timeout to 53 seconds:
- Watchdog time = 1 * 53 + 256 * 0 = 53 seconds
- Read the watchdog settings
- Read the down counter of the watchdog
- Every pass of the do-while loop retrigger the watchdog with 53 seconds

```
do {

        I2C_Write(0x50,0x01,53);            // Write to watchdog low register the value 53
        I2C_Write(0x50,0x00,00);            // Write to watchdog high register the value 00

        // Your program code
        …


        …
        // Your program code


        // Only for Main Revision 2

        if (Main_Revision!=1)
                {
                // Read Watchdog Counter
                Watchdog_Counter_High= I2C_Read(0x50,0x2d);    // Read the watchdog counter high byte
                Watchdog_Counter_Low= I2C_Read(0x50,0x2e);    // Read the watchdog counter low byte
                Watchdog_Counter=Watchdog_Counter_High*256+ Watchdog_Counter_Low;

                printf("Watchdog-Counter is: %d", Watchdog_Counter);
                }

        // Main_Revision 1 und Main_Revision 2

        Watchdog_High_Register=I2C_Read(0x50,0x01)          // Read watchdog high register
        Watchdog_Low_Register=I2C_Read(0x50,0x00)           // Read watchdog low register

        printf("Watchdog-Register High:%d   Low:%d", Watchdog_High_Register,Watchdog_Low_Register);


        printf("Watchdog is");

        if (Watchdog_High_Register  & 128)
                                {
                                        printf("disabled");
                                }
                        else
                                {
                                        printf("enabled");
                                }


         }

while (x!=END_CONDITION)
```

1.2 Counter

1.2.1 Operating-Time Counter

Reading these registers returns the operating time in hours.

Register	Access	Name	State at first power-on:	Basic	Enhanced
0x02	Read only	Operating-Time Counter Byte 0	0000 0000	•	•
0x03	Read only	Operating-Time Counter Byte 1	0000 0000	•	•
0x04	Read only	Operating-Time Counter Byte 2	0000 0000	•	•

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Operating-Time Counter Byte 2								**Operating-Time Counter Byte 1**								**Operating-Time Counter Byte 0**							
23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Operating-Time Counter																							

Range 0-16777215 operating hours.
The Operating-Time Counter has no overflow. The counter stops upon reaching the maximum value.

Example:

// Main Revision 1 and Main Revision 2

```
OTC0= I2C_Read(0x50,0x02);      // Read the Operating-Time Counter Byte 0
OTC1= I2C_Read(0x50,0x03);      // Read the Operating-Time Counter Byte 1
OTC2= I2C_Read(0x50,0x04);      // Read the Operating-Time Counter Byte 2

Hours=OTC2*65536+OTC1*256+OTC0;      // Calculate the Operating-Time
```

1.2.2 Power-On Counter

Reading these registers returns the number of cold starts.

Register	Access	Name	State at first power-on:	Basic	Enhanced
0x05	Read only	Power-On Counter Byte 0	0000 0000	•	•
0x06	Read only	Power-On Counter Byte 1	0000 0000	•	•

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Power-On Counter Byte 1								**Power-On Counter 0**							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Power-On Counter															

Range 0-65535 cold starts. Warm starts are not counted.
The Power-On Counter has no overflow. The counter stops upon reaching the maximum value.

Example:

```
// Main Revision 1 and Main Revision 2

    PTC0= I2C_Read(0x50,0x05);      // Read the Power-On Counter Byte 0
    PTC1= I2C_Read(0x50,0x06);      // Read the Power-On Counter Byte 1

    PTC_SUM=PTC1*256+PTC0;          // Calculate the Power-On Counter
```

1.2.3 On-Timer

Returns the on time since power up.

Register	Access	Name	State at power-on:	Basic	Enhanced
28	Read only	ON-Timer Second Byte	0000 0000		•
29	Read only	ON-Timer Minute Byte	0000 0000		•
2A	Read only	ON-Timer Hour 1 Byte	0000 0000		•
2B	Read only	ON-Timer Hour 2 Byte	0000 0000		•

7	6	5	4	3	2	1	0
unused	unused						
ON-Timer Second Byte							

7	6	5	4	3	2	1	0
unused	unused						
ON-Timer Minute Byte							

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
On-Timer Hour 2 Byte								**On-Timer Hour 1 Byte**							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
On Timer Hour Word															

Range 0-65536 operating hours.
The On-Timer has an overflow. When maximum value of 65536h:59m:59s has been reached the counter resets to 0h:0m:0s.

Example:

```
// Only for Main Revision 2

if (Main_Revision!=1)

        {
        OT_Second     =  I2C_Read(0x50,0x28);        // Read the ON-Timer Second Byte
        OT_Minute     =  I2C_Read(0x50,0x29);        // Read the ON-Timer Minute Byte
        OT_Hour_1     =  I2C_Read(0x50,0x2A);        // Read the ON-Timer Hour   Byte 1
        OT_Hour_2     =  I2C_Read(0x50,0x2B);        // Read the ON-Timer Hour   Byte 2


        OT_Hours      = OT_Hour_1*256+OT_Hour_2;     // Calculate the ON_Timer Hours

        }
```

1.3 Hardware Monitoring

1.3.1 Voltage Metering: 12 Volts

Reading this register returns the measured 12 V voltage.

Register	Access	Name	Basic	Enhanced
07	Read only	Measured12V0	•	•

Reference: 4.75 Volts (approximately)

Voltage divider: 1 / 4.

Minimum: Measured12V0 = 0 → 0 Volt
Maximum: Measured12V0 = 255 → 19 Volts.
Resolution: 0.075 Volts

$$Result\_12V0 = \frac{Measured\_Value\_12V0}{255} \cdot \frac{4.75\ V}{4}$$

Example:

```
// Main Revision 1 and  Main Revision 2

    double Result12V0;                      // Result12V0 should be a float or double

    Measured12V0= I2C_Read(0x50,0x07);      // Read the voltage information (12 Volts)

    Result12V0=Measured12V0/255*4.75*4;     // Calculate the voltage
```

1.3.2 Voltage Metering: 5 Volts

Reading this register returns the measured 5 V voltage.

Register	Access	Name	Basic	Enhanced
08	Read only	Measured5V0	•	•

Reference: 4.75 Volts (approximately)

Voltage divider: 1 / 2.

Minimum: Measured5V0 = 0 → 0 Volt
Maximum: Measured5V0 = 255 → 9.5 Volts.
Resolution: 0.037 Volts

$$\text{Result}\_5V0 = \frac{\text{Measured\_Value\_5V0}}{255} \cdot \frac{4.75\ V}{2}$$

Example:

```
// Main Revision 1 and Main Revision 2

    double Result5V0;                    // Result 5V0 should be a float or double

    Measured5V0= I2C_Read(0x50,0x08);    // Read the voltage information (5 Volts)

    Result5V0=Measured5V0/255*4.75*2;    // Calculate the voltage
```

1.3.3 Voltage Metering: 3.3 Volts

Reading this register returns the measured 3.3 V voltage.

Register	Access	Name	Basic	Enhanced
09	Read only	Measured3V3	•	•

Reference: 4.75 Volts (approximately)

Voltage divider: 1 / 1.

Minimum: Measured3V3 = 0 → 0 Volt
Maximum: Measured3V3 = 255 → 4.75 Volts.
Resolution: 0.018 Volts

$$\text{Result\_}3V3 = \frac{\text{Measured\_Value\_3V3}}{255} \cdot \frac{4.75\ V}{1}$$

Example:

```
// Main Revision 1 and  Main Revision 2

    double Result3V3;                   // Result3V3 should be a float or double

    Measured3V3= I2C_Read(0x50,0x09);   // Read the voltage information (3.3 Volts)

    Result3V3=Measured3V3/255*4.75*1;   // Calculate the voltage
```

1.3.4 Voltage Metering: Battery

Reading this register returns the measured battery voltage.

Register	Access	Name	Basic	Enhanced
22	Read only	Measured Battery Voltage		•

Reference: 4.75 Volts (approximately)

Voltage divider: 1 / 1.

Minimum: MeasuredBattery = 0 → 0 Volt
Maximum: MeasuredBattery = 255 → 4.75 Volts.
Resolution: 0.018 Volts

$$\text{Result\_Battery} = \frac{\text{Measured\_Value\_Battery}}{255} \cdot \frac{4.75\ V}{1}$$

Example:

```
// Only for Main Revision 2

if (Main_Revision!=1)

        {

        double ResultBattery;                   // ResultBattery should be a float or double

        MeasuredBattery= I2C_Read(0x50,0x22);   // Read the voltage information (Battery)

        ResultBattery=MeasuredBattery/255*4.75*1;   // Calculate the voltage

        // ResultBattery should be a float or double variable

        }
```

1.3.5 Board Temperature

Reading this register returns the measured board temperature.

The board temperature measured with a precision temperature sensor (LM135).
The LM135 has a breakdown voltage of +10 mV/°K that is directly proportional to absolute temperature.

The sensor is not implemented on every board. Please check your board specification.

This function returns an undefined value, if the board/system does not have a sensor implemented.

Register	Access	Name	Basic	Enhanced
0A	Read only	Board Temperature	•	•

Reference: 4.75 Volts (approximately)
Voltage divider: 1 / 1.

Minimum: Board_Temp = 0 → 0 Kelvin → -273 °C
Maximum: Board_Temp = 255 → 475 Kelvin → +202 °C

Resolution: 1.85 Kelvin

$$Temp(°C) = \frac{Board\_Temp}{255} \cdot \frac{4.75\ V}{10 \cdot 10^{-3}\ \frac{V}{K}} - 273K$$

Example:

```
// Main Revision 1 and Main Revision 2

    double Board_Temp;                          // Board_Temp should be a float or double

    Measured_Temp= I2C_Read(0x50,0x0A);         // Read the temperature information

    Board_Temp=Measured_Temp/255*4.75/10e-3-273;  // Calculate the Temperature
```

1.3.6 Fan Speed

Reading any Fan Speed register returns the measured fan speed.
Reading the Fan Settings register returns the settings of the DIP-Switch "Fan-Setting".

Register	Access	Name	Basic	Enhanced
0x0C	Read only	Fan Speed 1	•	•

Divider: 1 / 100

Register	Access	Name	Basic	Enhanced
0x23	Read only	Fan Speed 2		•

Divider: 1 / 100

Register	Access	Name	Basic	Enhanced
0x24	Read only	Fan Settings		•

FAN Settings							
7	6	5	4	3	2	1	0
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Enable Measuring FAN 1	1 Enable Measuring FAN 2

Bit 0: Enable Measuring Fan 2 1: Enable 0: Disable

Bit 1: Enable Measuring Fan 1 1: Enable 0: Disable

Bit 2..7: Reserved

The DIP-Switch „FAN-Setting" sets the „FAN-Settings". Please check your board specification.

Main Revision 1: (basic instruction set)

A fan connector is not implemented on every board. Please check your board specification.

The function returns 0, if the board does not have a fan or fan-connector.

Main Revision 2: (enhance instruction set)

Fan Setting Enable Measuring Fan 1	Enable Measuring Fan 2	Fan Speed 1 return value	Fan Speed 2 return value
disable	disable	0	0
enable	disable	1/100 the measured Fan Speed	0
disable	enable	0	1/100 the measured Fan Speed
enable	enable	1/100 the measured Fan Speed	1/100 the measured Fan Speed

Example:

```
// Main Revision 1 and Main Revision 2

Fan_Speed_Read_1= I2C_Read(0x50,0x0C);   // Read the Fan Speed 1 information
FanSpeed_1=Fan_Speed_Read_1*100;         // Calculate the Fan Speed 1
printf("Fan Speed 1:%-5d", FanSpeed1);   // Output of Fan Speed 1

// Only for Main Revision 2

if (Main_Revision!=1)
        {
Fan_ Speed_Read _2= I2C_Read(0x50,0x23); // Read the Fan Speed 2 information
FanSpeed_2=Fan_ Speed_Read _2*100;       // Calculate the Fan Speed 2
printf("Fan Speed 2:%-5d", FanSpeed2);   // Output of Fan Speed 2

FanSetting= I2C_Read(0x50,0x24);         // Read the Fan Setting information


                printf("Fan1-Setting:");
                 if (FanSetting & 2)
                        {
                                printf("ON");
                        }
                 else
                        {
                                printf("OFF");
                        }

                 printf("Fan2-Setting:",13);

                 if (FanSetting & 1)
                        {
                                printf("ON");
                        }
                 else
                        {
                                printf("OFF");
                        }
        }
```

1.3.7 AD-Channel Errors

These registers show if any monitored value is not within certain limits.
The limits are programmed and not chargeable. An out of limit condition will return a “1” in the corresponding bit.

Register	Access	Name	Default	Basic	Enhanced
0x02	Read only	AD-Channel Error Byte 1	0000 0000	•	•
0x25	Read only	AD-Channel Error Byte 2	0000 0000		•

AD-Channel Error-Byte 1							
7	6	5	4	3	2	1	0
Temperature too high	Temperature too low	3.3 Volts **over** voltage	3.3 Volts **under** voltage	5.0 Volts **over** voltage	5.0 Volts **under** voltage	12.0 Volts **over** voltage	12.0 Volts **under** voltage

Limits:

Measured	Condition	Bit	Register	Measured Value	Equivalent (ca.):
12V	**under voltage**	**0**	**0x02**	**<155**	**<11.4 Volts**
12V	**over voltage**	**1**	**0x02**	**>172**	**>12.6 Volts**
5V	under voltage	2	0x02	<129	<4.7 Volts
5V	over voltage	3	0x02	>142	>5.2 Volts
3.3V	**under voltage**	**4**	**0x02**	**<171**	**<3.1 Volts**
3.3V	**over voltage**	**5**	**0x02**	**>189**	**>3.5 Volts**
Temp.	Temperature too low	6	0x02	<143	<-10°C
Temp.	Temperature too high	7	0x02	>189	>+79°C

The bits 6/7 from register 0x02 are undefined, if the system/board does not have a temperature sensor implemented, please check your board/system specification.

AD-Channel Error-Byte 2							
7	6	5	4	3	2	1	0
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Battery Volts **over** voltage	Battery Volts **under** voltage

Limits:

Measured	Condition	Bit	Register	Measured Value	Equivalent (ca.):
Battery	**under voltage**	**0**	**0x25**	**<135**	**<2.5 Volts**
Battery	**over voltage**	**1**	**0x25**	**>189**	**>3.5 Volts**
	reserved	2	0x25		
	reserved	3	0x25		
	reserved	**4**	**0x25**		
	reserved	**5**	**0x25**		
	reserved	6	0x25		
	reserved	7	0x25		

Example:

```
// Main Revision 1 and Main Revision 2

        AD_Error_1= I2C_Read(0x50,0x0D);                    // Read the AD-Error 1 information

        if (AD_Error_1 & 0x01) printf("\n12 Volt under voltage");
        if (AD_Error_1 & 0x02) printf("\n12 Volt over voltage");
        if (AD_Error_1 & 0x04) printf("\n5 Volt under voltage");
        if (AD_Error_1 & 0x08) printf("\n5 Volt over voltage");
        if (AD_Error_1 & 0x10) printf("\n3.3 Volt under voltage");
        if (AD_Error_1 & 0x20) printf("\n3.3 Volt over voltage");
        if (AD_Error_1 & 0x40) printf("\nBoard Temperature to high");
        if (AD_Error_1 & 0x80) printf("\nBoard Temperature to low");

// Only for Main_Revision 2

        if (Main_Revision!=1)
                {
                AD_Error_2= I2C_Read(0x50,0x25);            // Read the AD-Error 2  information

                if (AD_Error_2 & 0x01) printf("\nBattery under voltage");
                if (AD_Error_2 & 0x02) printf("\nBattery over voltage");
                }
```

1.3.8 Status Monitoring Register

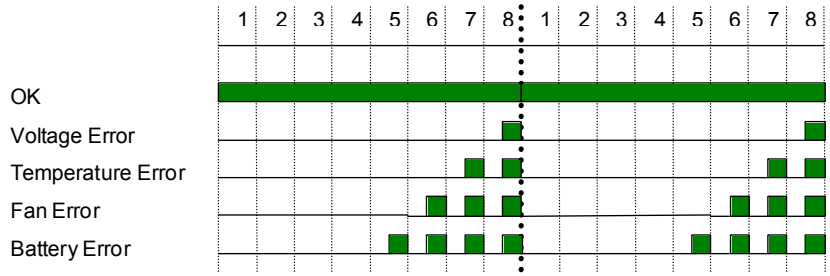
Reading this register returns the current status monitoring.

Register	Access	Name	Basic	Enhanced
0x2C	Read only	Status Monitoring		•

This function display four important fail conditions:

- Voltage Error
- Temperature Error
- Fan Error
- Battery Error

All these errors will be notified by the status led and by a beeper. The following illustration shows the blink/beep for different conditions:



If more than one fail condition occurs simultaneously, only the highest priority error code will be shown. The priority order is:

1. most — Voltage Error
2. less — Temperature Error
3. less — Fan Error
4. at least — Battery Error

E.g. are there a temperature error and a fan error, only the temperature error will be displayed.

Error range (see 1.3.7):

		Error	Good:	Error
Voltage Error	12V	0..<11,4V	11,4V..12.6V	>12.6V..
	5V	0..<4.7V	4.7V..5.2V	>5.2V
	3.3V	0..<3.1V	3.1V..3.5V	>3.5V
Temperature Error		…<-10°C	-10°C… +79°C	>79°C
Fan Error	1	…<900 1/min	900 1/min...20000 1/min	>20000 1/min
	2	…<900 1/min	900 1/min...20000 1/min	>20000 1/min
Battery Error		0..<2.5V	2.5V … 3.5V	>3.5V

Register value allocation:

Condition:	Register Value: binary	decimal:
OK	0000 0000	0
Voltage Error	1111 1110	254
Temp. Error	1111 1100	252
Fan Error	1111 1000	248
Battery Error	1111 0000	240

Example:

```
// Only for Main Revision 2

        if (Main_Revision!=1)
                {
                        StatusMonitoring= I2C_Read(0x50,0x2C);          // Read the information

                        printf("Status-Monitoring::");


                                        switch (StatusMonitoring)
                                {
                                        case 254:
                                                printf("Voltage Error");
                                        break;

                                        case 252:
                                                printf("Temp Error");
                                        break;

                                        case 248:
                                                printf("Fan Error");
                                        break;

                                        case 240:
                                                printf("Battery Error");
                                        break;



                                        default:
                                          printf("OK");

                                }
                }
```

1.5 Memory

1.5.1 Dummy Register

This register is for test purposes and can be used as a 1-byte scratch pad.

Register	Access	Name	Default	Basic	Enhanced
0x0F	Read / Write	Dummy Register	0000 0000	•	•

Example:

// Main Revision 1 and Main Revision 2

```
I2C_Write(0x50,0x0F,10);                    // Write the Dummy Register
Dummy_Register= I2C_Read(0x50,0x0F);        // Read the Dummy Register
```

1.5.2 Protected Memory

These registers provide access to the protected EEPROM memory.

Register	Access	Name	Default	Basic	Enhanced
0x20	Write	Protected Key	0000 0000	•	•

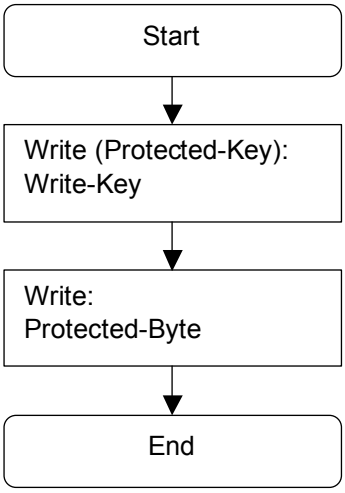
The key is needed to write or read the protected memory.

If the protected memory accessed without the protected keys, the access will be ignored.

Every single protected memory access (read/write) must be preceded directly by the appropriate key.

1.5.2.1 Write access to protected memory

Write Key: **0x0F**

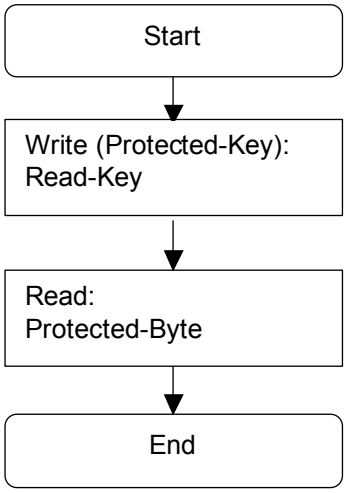


I2C_Write(0x50,0x20,Write_Key);

I2C_Write(0x50,0x10,Byte_to_write);

1.5.2.2 Read access to protected memory

Read Key: **0xF0**



I2C_Write(0x50,0x20,Read_Key);

Protected00=2C_Read(0x50,0x10)

1.6 Error Register

This register returns the SuperVision state error flags.

Register	Access	Name	Default	Basic	Enhanced
0x0E	Read only	Error Register	0000 0000	•	•

Error Register

7	6	5	4	3	2	1	0
Reserved	Interrupt Error	SM(I2C)-Read Error	SM(I2C)-Write Error	EEPROM Read Error	EEPROM Write Error	OTC overflow	Watchdog reset

Bit 0: Watchdog Reset: Watchdog reset occurred

Bit 1: OTC overflow: Operating Time Counter overflow occurred

Bit 2: EEPROM Write Error: EEPROM write error occurred

Bit 3: EEPROM Read Error: EEPROM read error occurred

Bit 5: SM(I2C)-Write Error: SM(I2C-)-write error occurred

Bit 5: SM(I2C)-Read Error SM(I2C-)-read error occurred

Bit 6: Interrupt-Error: Undefined Interrupt occurred

Bit 7: Reserved

Example:

```
// Main Revision 1 and  Main Revision 2

        Error_Register= I2C_Read(0x50,0x0E);          // Read the Error-Register

        if (Error_Register & 0x01)
        {
                printf("\n A Watchdog reset occurred");
        }
        else
        {
                printf("\n Unknown reset");
        }
```

1.7 Board Power State Register

This register returns the state of certain board power state signals.

Register	Access	Name	Basic	Enhanced
0x27	Read only	Board Power State		•

Board-Power-State Register

7	6	5	4	3	2	1	0
Reserved	Reserved	Reserved	ATX Power-Good	Power-Good 3,3V-Standby	Power-Good 5,0V-Standby	Power-Good Board	PS_ON#

Bit 0: PS_ON#

Bit 1: Power-Good Board

Bit 2: Power Good 5,0V-Standby

Bit 3: Power Good 3,3V-Standby

Bit 4: ATX Power-Good

Bit 5: Reserved

Bit 6: Reserved

Bit 7: Reserved

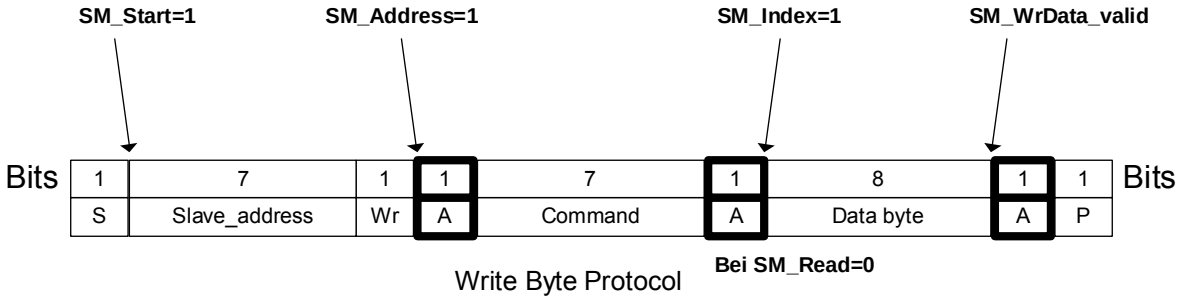
Example:

```
// Only for Main Revision 2

        if (Main_Revision!=1)
                {
                Board_Power_State= I2C_Read(0x50,0x27);          // Read the information

                if (!(Board_Power_State & 0x08)) printf("\n3,3V-Standby problem");
                }
```

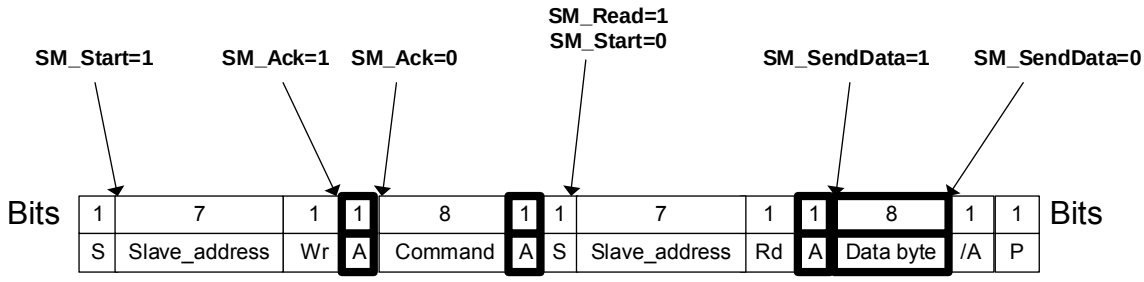
2.0 SM(I2C)-Bus Access

The SuperVision-IC only allows byte wise SM-Bus read any SM-Bus write access.
All other forms of access, such as sequential SM-Bus reads, are not allowed and could cause unpredictable conditions.

SM-Bus Protocol:



Write Byte Protocol



Read Byte Protocol

3.0 Revision History

Main Revision		Version	Description	Date
1	2			
x		51	Initial Release Revision 1	18.05.04
x		01	I2C-Sequentiel Read Handling I2C-Exception Handling Interrupt-Exception Handling Fast Protected Memory Access	26.07.04
x		02	Reset Handling for PCI954	08.12.04
x		03	Improved storage routine for Power on Counter + Operating Time Counter	03.06.05
	x	01	Initial Release Revision 2	12.09.06
	x	02	Expand PIC-Supply Voltage, only BTX-Boards	13.10.06

4.0 Demo program

Version: ETX
I²C-Access via JIDA32-Library

```
// **
// ** Title: SuperVision - Demo program ETX-EtxExpress
// **
// ** Author: Dipl-Ing(FH) Michael Koch
// ** Kontron Roding
// **
// ** Last Change: 12-09-2006
// **
// Changes: --

#include "stdafx.h"
#include "resource.h"
#include "stdio.h"
#include "Jida.h"
#include "time.h"


#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                                    // current instance
TCHAR szTitle[MAX_LOADSTRING];                                                  // The
title bar text
TCHAR szWindowClass[MAX_LOADSTRING];

HJIDA  hJida;

char Output_String[50];
char Output_Protected[50];
BYTE Output_Byte[16];
BYTE Temp,Counter;
unsigned long int Output_Long;
time_t rawtime;
struct tm * timeinfo;
int month,Main_Revision;



#define Base_Number_I2C 0x0
#define PIC_Address_I2C      0x50
#define Protected_Write      0x0F
#define Protected_Read0xF0

void JidaInit();

void Set_Watchdog(BYTE Watchdog_High,BYTE Watchdog_Low);
void Watchdog(BYTE *Watchdog_High,BYTE *Watchdog_Low);
void OTC(BYTE *OTC_0,BYTE *OTC_1,BYTE *OTC_2);
void PTC(BYTE *PTC_0,BYTE *PTC_1);
double Measured12V0();
double Measured5V0();
double Measured3V3();
double BoardTemp();
BYTE Revision();
int Revolution();
BYTE AD_Errors();
BYTE Error_Register();
void WriteDummy(BYTE ByteToDummy);
BYTE ReadDummy();
BYTE Read_Protected_Byte(BYTE Register,BYTE Key_to_Read);
void Write_Protected_Byte(BYTE Register,BYTE Byte_to_Write,BYTE Key_to_Write);
BYTE MainRevision();
double MeasuredBattery();
int Revolution_2();
```

```
BYTE StatusDipSwitch();
BYTE AD_Errors_2();
BYTE ThermalTrip();
BYTE BoardPowerState();
BYTE ONTimerSecond();
BYTE ONTimerMinute();
BYTE ONTimerHour1();
BYTE ONTimerHour2();
BYTE StatusMonitoring();
BYTE WatchdogHighCount();
BYTE WatchdogLowCount();



BYTE Readbyte(BYTE Register);
void Writebyte(BYTE Register,BYTE Byte_to_Write);

void delay();

// Foward declarations of functions included in this code module:
ATOM                        MyRegisterClass(HINSTANCE hInstance);
BOOL                        InitInstance(HINSTANCE, int);
LRESULT CALLBACK      WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK      About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
        // TODO: Place code here.
        MSG msg;
        HACCEL hAccelTable;

        // Initialize global strings
        LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
        LoadString(hInstance, IDC_PICETX, szWindowClass, MAX_LOADSTRING);
        MyRegisterClass(hInstance);

        JidaInit();

        // Perform application initialization:
        if (!InitInstance (hInstance, nCmdShow))
        {
                return FALSE;
        }

        hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_PICETX);

        // Main message loop:
        while (GetMessage(&msg, NULL, 0, 0))
        {
                if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
                {
                        TranslateMessage(&msg);
                        DispatchMessage(&msg);
                }
        }

        return msg.wParam;
}

//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
//
//  COMMENTS:
//
//    This function and its usage is only necessary if you want this code
//    to be compatible with Win32 systems prior to the 'RegisterClassEx'
//    function that was added to Windows 95. It is important to call this function
//    so that the application will get 'well formed' small icons associated
//    with it.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
```

```
        WNDCLASSEX wcex;

        wcex.cbSize = sizeof(WNDCLASSEX);

        wcex.style                      = CS_HREDRAW | CS_VREDRAW;
        wcex.lpfnWndProc        = (WNDPROC)WndProc;
        wcex.cbClsExtra                 = 0;
        wcex.cbWndExtra                 = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon                      = LoadIcon(hInstance, (LPCTSTR)IDI_PICETX);
        wcex.hCursor            = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground      = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName       = (LPCSTR)IDC_PICETX;
        wcex.lpszClassName      = szWindowClass;
        wcex.hIconSm            = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

        return RegisterClassEx(&wcex);
}

//
//   FUNCTION: InitInstance(HANDLE, int)
//
//   PURPOSE: Saves instance handle and creates main window
//
//   COMMENTS:
//
//        In this function, we save the instance handle in a global variable and
//        create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   HWND hWnd;

   hInst = hInstance; // Store instance handle in our global variable

   hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
      CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

   if (!hWnd)
   {
      return FALSE;
   }

   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);

   return TRUE;
}

//
//  FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
//
//  PURPOSE:  Processes messages for the main window.
//
//  WM_COMMAND - process the application menu
//  WM_PAINT   - Paint the main window
//  WM_DESTROY - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
        int wmId, wmEvent;
        PAINTSTRUCT ps;
        HDC hdc;
        TCHAR szHello[MAX_LOADSTRING];
        LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);

        switch (message)
        {
                case WM_COMMAND:
                        wmId    = LOWORD(wParam);
                        wmEvent = HIWORD(wParam);
                        // Parse the menu selections:
                        switch (wmId)
                        {
                                case IDM_REFRESH:
                                InvalidateRect(hWnd,NULL,TRUE);
                                break;
```

```
                        case IDM_ABOUT:
                           DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd, (DLGPROC)About);
                           break;

                        case IDM_Watch_10:
                               MessageBox(hWnd, "Watchdog is set to 10 seconds","",48+256);

                               Set_Watchdog(0x0, 0x0A);
                               InvalidateRect(hWnd,NULL,TRUE);
                        break;

                        case IDM_Watch_60:
                               MessageBox(hWnd, "Watchdog is set to 60 seconds","",48+256);
                               Set_Watchdog(0x0, 60);
                               InvalidateRect(hWnd,NULL,TRUE);

                        break;

                         case IDM_Watch_OFF:
                               MessageBox(hWnd, "Watchdog is set OFF","",48+256);
                               Set_Watchdog(0xff,0xff);
                               InvalidateRect(hWnd,NULL,TRUE);

                        break;

                        case IDM_PROTECTEDMEMORY:
                               MessageBox(hWnd, "Write to Protected Memory","",48+256);
                               time( &rawtime);
                               timeinfo = localtime(&rawtime);
                               month=(timeinfo->tm_mon)+1;

                               sprintf(Output_String,"%s",asctime(timeinfo));

                               Write_Protected_Byte(0x00,Output_String[0x0B]-
48,Protected_Write);
                               Write_Protected_Byte(0x01,Output_String[0x0C]-
48,Protected_Write);
                               Write_Protected_Byte(0x02,Output_String[0x0E]-
48,Protected_Write);
                               Write_Protected_Byte(0x03,Output_String[0x0F]-
48,Protected_Write);
                               Write_Protected_Byte(0x04,Output_String[0x11]-
48,Protected_Write);
                               Write_Protected_Byte(0x05,Output_String[0x12]-
48,Protected_Write);
                               Write_Protected_Byte(0x06,0xee,Protected_Write);
                               Write_Protected_Byte(0x07,Output_String[0x08]-
48,Protected_Write);
                               Write_Protected_Byte(0x08,Output_String[0x09]-
48,Protected_Write);
                               if (month<10)
                               {
                               Write_Protected_Byte(0x09,0x00,Protected_Write);
                               Write_Protected_Byte(0x0A,month,Protected_Write);
                               }
                               else
                               {
                               month-=10;
                               Write_Protected_Byte(0x09,0x01,Protected_Write);
                               Write_Protected_Byte(0x0A,month,Protected_Write);
                               }

                               Write_Protected_Byte(0x0B,Output_String[0x14]-
48,Protected_Write);
                               Write_Protected_Byte(0x0C,Output_String[0x15]-
48,Protected_Write);
                               Write_Protected_Byte(0x0D,Output_String[0x16]-
48,Protected_Write);
                               Write_Protected_Byte(0x0E,Output_String[0x17]-
48,Protected_Write);

                               Write_Protected_Byte(0x0F,0xEE,Protected_Write);

                               InvalidateRect(hWnd,NULL,TRUE);

                        break;
```

```
                case IDM_EXIT:
                   DestroyWindow(hWnd);
                   break;
                default:
                   return DefWindowProc(hWnd, message, wParam, lParam);
            }
            break;

    case WM_PAINT:

            hdc = BeginPaint(hWnd, &ps);
            RECT rt;
            GetClientRect(hWnd, &rt);

            TextOut(hdc,40,12,"Supervision Register Overview:",30);

            // Test Main-Revision

            Main_Revision=MainRevision();

            if (Main_Revision>250) Main_Revision=1;


            // Mask

            MoveToEx(hdc,10,35,0);
            LineTo(hdc,700,35);
            MoveToEx(hdc,10,60,0);
            LineTo(hdc,700,60);
            MoveToEx(hdc,10,105,0);
            LineTo(hdc,700,105);
            MoveToEx(hdc,10,170,0);
            LineTo(hdc,700,170);
            MoveToEx(hdc,10,215,0);
            LineTo(hdc,700,215);
            MoveToEx(hdc,10,260,0);
            LineTo(hdc,700,260);
            MoveToEx(hdc,10,285,0);
            LineTo(hdc,700,285);
            MoveToEx(hdc,10,330,0);
            LineTo(hdc,700,330);


            // Output Main-Revision

            Output_Byte[0]=Main_Revision;

            sprintf(Output_String,"%-3d",Output_Byte[0]);

            TextOut(hdc,40,40,"Main-Revision:",14);
            TextOut(hdc,220,40,Output_String,3);

                    // Output Revision

            Output_Byte[0]=Revision();
            sprintf(Output_String,"%-3d",Output_Byte[0]);

            TextOut(hdc,300,40,"Revision:",9);
            TextOut(hdc,460,40,Output_String,3);



            // Output Operating Time Counter


            OTC(&Output_Byte[0],&Output_Byte[1],&Output_Byte[2]);
            Output_Long=Output_Byte[2]*65536+Output_Byte[1]*256+Output_Byte[0];
            TextOut(hdc,40,65,"Operating-Time-Counter[h]:",26);
            sprintf(Output_String,"%-8d",Output_Long);
            TextOut(hdc,220,65,Output_String,8);

            // Output Power On Counter


            PTC(&Output_Byte[0],&Output_Byte[1]);
            Output_Long=Output_Byte[1]*256+Output_Byte[0];
            TextOut(hdc,40,85,"Power-On-Counter:",17);
```

```
                        sprintf(Output_String,"%-5d",Output_Long);
                        TextOut(hdc,220,85,Output_String,5);

                        // Output On-Timer
                        // only Main-Revison 2

                        if (Main_Revision!=1)
                        {
                         Output_Long=ONTimerHour2()*255+ONTimerHour1();
                         sprintf(Output_String,"%-5d:%-2d: %-
2d",Output_Long,ONTimerMinute(),ONTimerSecond());
                         TextOut(hdc,300,65,"On-Timer [hh:mm:ss]:",19);
                         TextOut(hdc,460,65,Output_String,12);
                        }

                        // Output Voltage metering 12 Volts

                        TextOut(hdc,40,110,"12.0V - measured:",17);
                        sprintf(Output_String,"%-2.1f V",Measured12V0());
                        TextOut(hdc,220,110,Output_String,6);

                        // Output Voltage metering 5 Volts

                        TextOut(hdc,300,110,"5.0V - measured:",16);
                        sprintf(Output_String,"%-1.1f V",Measured5V0());
                        TextOut(hdc,460,110,Output_String,5);

                        // Output Voltage metering 3.3 Volts

                        TextOut(hdc,40,130,"3.3V - measured:",16);
                        sprintf(Output_String,"%-1.1f V",Measured3V3());
                        TextOut(hdc,220,130,Output_String,5);

                        // Output Measured Battery
                        // only Main-Revison 2

                        if (Main_Revision!=1)
                        {
                         TextOut(hdc,300,130,"Battery - measured:",19);
                         sprintf(Output_String,"%-1.1f V",MeasuredBattery());
                         TextOut(hdc,460,130,Output_String,5);
                        }

                        // Output Board Temperature

                        TextOut(hdc,40,150,"Board-Temp:",11);
                        sprintf(Output_String,"%-3.1f °C",BoardTemp());
                        TextOut(hdc,220,150,Output_String,7);

                        // Output Revolution Fan 1


                        sprintf(Output_String,"%-5d rpm",Revolution());
                        TextOut(hdc, 40,175,"Fan1-Revolution:",16);
                        TextOut(hdc,220,175,Output_String,9);

                        // Output Revolution Fan 2
                        // only Main-Revison 2

                        if (Main_Revision!=1)
                        {
                         sprintf(Output_String,"%-5d rpm",Revolution_2());
                         TextOut(hdc,40,195,"Fan2-Revolution:",16);
                         TextOut(hdc,220,195,Output_String,9);
                        }

                        // Output FanSettings
                        // only Main-Revison 2

                        if (Main_Revision!=1)
                        {

                         TextOut(hdc,300,175,"Fan1-Setting:",13);
                         if ( StatusDipSwitch() & 2)
                               {
                                      TextOut(hdc,460,175,"ON",2);
                               }
                         else
```

```
        {
                TextOut(hdc,460,175,"OFF",3);
        }

 TextOut(hdc,300,195,"Fan2-Setting:",13);

 if ( StatusDipSwitch() & 1)
        {
                TextOut(hdc,460,195,"ON",2);
        }
 else
        {
                TextOut(hdc,460,195,"OFF",3);
        }
}

// Output Watchdog Settings


Watchdog(&Output_Byte[0],&Output_Byte[1]);
sprintf(Output_String,"%3d",Output_Byte[0]);
TextOut(hdc,40,220,"Watchdog-High-Register:",23);
sprintf(Output_String,"%-3d",Output_Byte[0]);
TextOut(hdc,220,220,Output_String,3);
TextOut(hdc,300,220,"Watchdog-Low-Register:",22);
sprintf(Output_String,"%3d",Output_Byte[1]);
TextOut(hdc,460,220,Output_String,3);

// Output Watchdog Counter
// only Main-Revison 2

if (Main_Revision!=1)
{
Output_Byte[0]=WatchdogHighCount();
Output_Byte[1]=WatchdogLowCount();

sprintf(Output_String,"%-3d",Output_Byte[0]);
TextOut(hdc,40,240,"Watchdog-High-Counter:",22);
TextOut(hdc,220,240,Output_String,3);

sprintf(Output_String,"%-3d",Output_Byte[1]);
TextOut(hdc,300,240,"Watchdog-Low-Counter:",21);
TextOut(hdc,460,240,Output_String,3);
}

// Output StatusMonitoring

Output_Byte[0]=StatusMonitoring();
sprintf(Output_String,"%-2x",Output_Byte[0]);

TextOut(hdc,40,265,"Status-Monitoring:",18);


        switch (StatusMonitoring())
{
        case 254:
                TextOut(hdc,220,265,"Voltage Error",13);
        break;

        case 252:
                TextOut(hdc,220,265,"Temp Error",10);
        break;

        case 248:
                TextOut(hdc,220,265,"Fan Error",9);
        break;

        case 240:
                TextOut(hdc,220,265,"Battery Error",13);
        break;



        default:
           TextOut(hdc,220,265,"OK",2);

        }
```

```
		// Demo Dummy

		Temp=ReadDummy();
		sprintf(Output_String,"%-3d",Temp);
		TextOut(hdc, 40,290,"1.Read-Dummy:",13);
		TextOut(hdc,220,290,Output_String,3);

		TextOut(hdc, 40,310,"1.Write-Dummy[OLD+1]:",21);
		WriteDummy(++Temp);
		sprintf(Output_String,"%-3d",Temp);
		TextOut(hdc,220,310,Output_String,3);

		Temp=ReadDummy();
		sprintf(Output_String,"%-3d",Temp);
		TextOut(hdc,300,310,"2.Read-Dummy:",13);
		TextOut(hdc,460,310,Output_String,3);


		// Demo Protected Read

		// Wrong Key

		for(Counter=0;Counter<0x10;Counter++)
		{
		 Output_Byte[Counter]=Read_Protected_Byte(Counter,0x00);
		 sprintf(Output_String,"%2x ",Output_Byte[Counter]);
		 for(Temp=0;Temp<4;Temp++)
		 {
			Output_Protected[Temp+Counter*3]=Output_String[Temp];
		 }
		}
		Output_Protected[49]=0x0;
		TextOut(hdc, 40,335,"Read prot. memory(wrong key):",29);
		TextOut(hdc,245,335,Output_Protected,48);

		// Right Key

		for(Counter=0;Counter<0x10;Counter++)
		{
		 Output_Byte[Counter]=Read_Protected_Byte(Counter,Protected_Read);
		 sprintf(Output_String,"%2x ",Output_Byte[Counter]);
		 for(Temp=0;Temp<4;Temp++)
		 {
			Output_Protected[Temp+Counter*3]=Output_String[Temp];
		 }
		}
		Output_Protected[49]=0x0;
		TextOut(hdc, 40,355,"Read prot. memory (right key):",30);
		TextOut(hdc,240,355,Output_Protected,48);


	/*


		// Output AD-Channel-Errors

		sprintf(Output_String,"%-2x",AD_Errors());
		TextOut(hdc, 40,220,"AD-Channel-Errors[hex]:",23);
		TextOut(hdc,240,220,Output_String,2);

		// Output Error-Register

		sprintf(Output_String,"%-2x",Error_Register());
		TextOut(hdc, 40,240,"Error-Register[hex]:",20);
		TextOut(hdc,240,240,Output_String,2);



		// Demo Protected Read

		// Wrong Key

		for(Counter=0;Counter<0x10;Counter++)
		{
		 Output_Byte[Counter]=Read_Protected_Byte(Counter,0x00);
		 sprintf(Output_String,"%2x ",Output_Byte[Counter]);
		 for(Temp=0;Temp<4;Temp++)
```

```
                        {
                                Output_Protected[Temp+Counter*3]=Output_String[Temp];
                        }
                       }
                       Output_Protected[49]=0x0;
                       TextOut(hdc, 40,300,"Read prot. memory(wrong key):",29);
                       TextOut(hdc,245,300,Output_Protected,48);

                       // Right Key

                       for(Counter=0;Counter<0x10;Counter++)
                       {
                        Output_Byte[Counter]=Read_Protected_Byte(Counter,Protected_Read);
                        sprintf(Output_String,"%2x ",Output_Byte[Counter]);
                        for(Temp=0;Temp<4;Temp++)
                        {
                                Output_Protected[Temp+Counter*3]=Output_String[Temp];
                        }
                       }
                       Output_Protected[49]=0x0;
                       TextOut(hdc, 40,320,"Read prot. memory (right key):",30);
                       TextOut(hdc,240,320,Output_Protected,48);









                       // Output AD-Channel-Errors

                       sprintf(Output_String,"%-2x",AD_Errors_2());
                       TextOut(hdc, 40,420,"AD-Channel-2-Errors[hex]:",25);
                       TextOut(hdc,240,420,Output_String,2);

                       // Output ThermalTrip-Counter

                       Output_Byte[0]=ThermalTrip();
                       sprintf(Output_String,"%-3d",Output_Byte[0]);

                       TextOut(hdc,40,440,"ThermalTrip-Counter:",20);
                       TextOut(hdc,240,440,Output_String,3);

                       // Output Board-Power-State

                       Output_Byte[0]=BoardPowerState();
                       sprintf(Output_String,"%-2x",Output_Byte[0]);

                       TextOut(hdc,40,460,"Board-Power-State[hex]:",23);
                       TextOut(hdc,240,460,Output_String,2);




               */

                       EndPaint(hWnd, &ps);

                       break;
               case WM_DESTROY:
                       PostQuitMessage(0);
                       break;
               default:
                       return DefWindowProc(hWnd, message, wParam, lParam);
   }
   return 0;
}

// Mesage handler for about box.
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
       switch (message)
       {
               case WM_INITDIALOG:
                               return TRUE;

               case WM_COMMAND:
```

```
                        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
                        {
                                EndDialog(hDlg, LOWORD(wParam));
                                return TRUE;
                        }
                        break;
        }
    return FALSE;
}


void JidaInit()
{

        JidaDllInitialize();
        JidaBoardOpen(JIDA_BOARD_CLASS_CPU,0,0,&hJida);
}


BYTE Readbyte(BYTE Register)
{
BYTE pReadBytes[1],RetValue;
int Success_Read;

        Success_Read=JidaI2CReadRegister(hJida,Base_Number_I2C,PIC_Address_I2C,Register,&pReadBytes[0
]);
        if (!Success_Read) return(0xFF);
        RetValue=(BYTE)pReadBytes[0];

return(RetValue);
}


void Writebyte(BYTE Register,BYTE Byte_to_Write)
{
        JidaI2CWriteRegister(hJida,Base_Number_I2C,PIC_Address_I2C,Register,Byte_to_Write);
}


void Set_Watchdog(BYTE Watchdog_High, BYTE Watchdog_Low)
{
Writebyte(0x00,Watchdog_High);
Writebyte(0x01,Watchdog_Low);
}

void Watchdog(BYTE *Watchdog_High,BYTE *Watchdog_Low)
{
 *Watchdog_High       =Readbyte(0x00);
 *Watchdog_Low =Readbyte(0x01);
}

void OTC(BYTE *OTC_0,BYTE *OTC_1,BYTE *OTC_2)
{
 *OTC_0 =Readbyte(0x02);
 *OTC_1 =Readbyte(0x03);
 *OTC_2 =Readbyte(0x04);
}


void PTC(BYTE *PTC_0,BYTE *PTC_1)
{
 *PTC_0 =Readbyte(0x05);
 *PTC_1 =Readbyte(0x06);
}


double Measured12V0()
{
 BYTE ReadValue;
 double ReturnValue;

 ReadValue=Readbyte(0x07);
 ReturnValue=(double)ReadValue/255*4.75*4;
 return(ReturnValue);
}

double Measured5V0()
{
```

```
 BYTE ReadValue;
 double ReturnValue;

 ReadValue=Readbyte(0x08);
 ReturnValue=(double)ReadValue/255*4.75*2;
 return(ReturnValue);
}


double Measured3V3()
{
 BYTE ReadValue;
 double ReturnValue;

 ReadValue=Readbyte(0x09);
 ReturnValue=(double)ReadValue/255*4.75*1;
 return(ReturnValue);
}

double BoardTemp()
{
 BYTE ReadValue;
 double ReturnValue;

 ReadValue=Readbyte(0x0A);
 ReturnValue=(double)ReadValue/255*4.75/10e-3-273;
 return(ReturnValue);
}

BYTE Revision()
{
 BYTE RetValue;

 RetValue=Readbyte(0x0B);
 return(RetValue);
}

int Revolution()
{
 BYTE ReadValue;
 int ReturnValue;

 ReadValue=Readbyte(0x0C);
 ReturnValue=(BYTE)ReadValue*100;
 return(ReturnValue);
}

BYTE AD_Errors()
{
 BYTE RetValue;

 RetValue=Readbyte(0x0D);
 return(RetValue);
}



BYTE Error_Register()
{
 BYTE RetValue;
 RetValue=Readbyte(0x0E);
 return(RetValue);
}


BYTE MainRevision()
{
 BYTE RetValue;
 RetValue=Readbyte(0x21);
 return(RetValue);
}


double MeasuredBattery()
{
 BYTE ReadValue;
 double ReturnValue;
```

```
 ReadValue=Readbyte(0x22);
 ReturnValue=(double)ReadValue/255*4.75*1;
 return(ReturnValue);
}

int Revolution_2()
{
 BYTE ReadValue;
 int ReturnValue;

 ReadValue=Readbyte(0x23);
 ReturnValue=(BYTE)ReadValue*100;
 return(ReturnValue);
}

BYTE StatusDipSwitch()
{
 BYTE RetValue;
 RetValue=Readbyte(0x24);
 return(RetValue);
}

BYTE AD_Errors_2()
{
 BYTE RetValue;
 RetValue=Readbyte(0x25);
 return(RetValue);
}

BYTE ThermalTrip()
{
 BYTE RetValue;
 RetValue=Readbyte(0x26);
 return(RetValue);
}


BYTE BoardPowerState()
{
 BYTE RetValue;
 RetValue=Readbyte(0x27);
 return(RetValue);
}

BYTE ONTimerSecond()
{
 BYTE RetValue;

 RetValue=Readbyte(0x28);
 return(RetValue);
}


BYTE ONTimerMinute()
{
 BYTE RetValue;

 RetValue=Readbyte(0x29);
 return(RetValue);
}

BYTE ONTimerHour1()
{
 BYTE RetValue;

 RetValue=Readbyte(0x2a);
 return(RetValue);
}

BYTE ONTimerHour2()
{
 BYTE RetValue;

 RetValue=Readbyte(0x2b);
 return(RetValue);
}

BYTE StatusMonitoring()
```

```
{
 BYTE RetValue;

 RetValue=Readbyte(0x2c);
 return(RetValue);
}


BYTE WatchdogHighCount()
{
 BYTE RetValue;

 RetValue=Readbyte(0x2d);
 return(RetValue);
}


BYTE WatchdogLowCount()
{
 BYTE RetValue;

 RetValue=Readbyte(0x2e);
 return(RetValue);
}


void WriteDummy(BYTE ByteToDummy)
{
       Writebyte(0x0F,ByteToDummy);

}

BYTE ReadDummy()
{
 BYTE RetValue;

 RetValue=Readbyte(0x0F);
 return(RetValue);
}

BYTE Read_Protected_Byte(BYTE Register,BYTE Key_to_Read)
{
 BYTE RetValue;

 Writebyte(0x20,Key_to_Read);
 RetValue=Readbyte(0x10+Register);
 return(RetValue);
}

void Write_Protected_Byte(BYTE Register,BYTE Byte_to_Write,BYTE Key_to_Write)
{
 Writebyte(0x20,Key_to_Write);
 Writebyte(0x10+Register,Byte_to_Write);
 //delay();
}

void delay()
{
 time_t start_time,cur_time;
 time(&start_time);
 do
 {
        time(&cur_time);
 }
 while((cur_time-start_time)<1);

}
```