User's Manual

Kontron SuperVision

kontron

If it's Embedded, it's Kontron.

Technical Manual

Revision: 3.3

Revision History			
Rev.	**Description**	**Date**	**Author**
1.0	Initial Release	24.07.2004	M. Koch
2.0	Upgrade	12.09.2006	M. Koch
3.0	Upgrade	14.02.2008	D. Brumfield
3.1	Upgrade	07.07.2008	D. Brumfield
3.2	Corrected Hyperlinks in document	23.07.2008	A. Kunz
3.3	Upgrade	13.11.2008	D. Brumfield

	Date	**Name**	**Department**
Prepared:	18.02.2008	Daniel Brumfield	(KEC) RD-B-R
Checked:	10.03.2008	Michael Koch	(KEC) RD-B-R
Approved:	10.03.2008	Michael Koch	(KEC) RD-B-R

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

Table of Contents:

1 Description

The SuperVision-IC provides many useful features such as:

- A watchdog timer
- An operating-time counter
- A power-on counter
- Voltage, temperature, and fan speed monitoring
- Release number
- A protected EEPROM memory

and so on.

Which of these features are provided by the SuperVison-IC depends upon the board/system used and can be determined with the “Feature Ident. No.” register (see “Feature Implementation” and/or the documentation for the board/system used). **Unimplemented features registers must not be accessed.**

The SuperVision-IC registers can be accessed though the SM-Bus.
The SM-Bus address is usually **0x50**[1] (Decimal: 80).
See your system documentation for details.

Boards based on ETX or ETXExpress modules can access the SM-Bus with the 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,
- Linux 2.2/2.4.

Most single board computers (SBCs), slot-CPUs, and STX-based systems do not have the JIDA32 Interface in BIOS and need separate drivers to communicate with the SuperVision-IC. If you are using one these types of system, please contact Kontron support for SM-Bus drivers.

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

2 Features

2.1 Register Map

The following table shows the possible features available:

<table>
<tr><th colspan="2">Register</th><th rowspan="2">Description</th><th rowspan="2">Byte</th><th rowspan="2">R/W Access</th><th rowspan="2">Section</th></tr>
<tr><th>Dec</th><th>Hex</th></tr>
<tr><td>0</td><td>00</td><td rowspan="2">Watchdog Settings</td><td>1</td><td rowspan="2">R/W</td><td rowspan="2">Watchdog</td></tr>
<tr><td>1</td><td>01</td><td>0</td></tr>
<tr><td>2</td><td>02</td><td rowspan="3">Operating Time Counter</td><td>0</td><td rowspan="3">Ro</td><td rowspan="3">Operating Time Counter</td></tr>
<tr><td>3</td><td>03</td><td>1</td></tr>
<tr><td>4</td><td>04</td><td>2</td></tr>
<tr><td>5</td><td>05</td><td rowspan="2">Power-On Counter</td><td>0</td><td rowspan="2">Ro</td><td rowspan="2">Power-On Counter</td></tr>
<tr><td>6</td><td>06</td><td>1</td></tr>
<tr><td>7</td><td>07</td><td colspan="2">Voltage Monitoring: 12V</td><td>Ro</td><td rowspan="3">Voltage Monitoring</td></tr>
<tr><td>8</td><td>08</td><td colspan="2">Voltage Monitoring: 5V</td><td>Ro</td></tr>
<tr><td>9</td><td>09</td><td colspan="2">Voltage Monitoring: 3.3V</td><td>Ro</td></tr>
<tr><td>10</td><td>0A</td><td colspan="2">Board Temperature (analog)</td><td>Ro</td><td>Temperature Monitoring</td></tr>
<tr><td>11</td><td>0B</td><td colspan="2">PIC Program Version</td><td>Ro</td><td>Program Version</td></tr>
<tr><td>12</td><td>0C</td><td colspan="2">Fan Speed 1</td><td>Ro</td><td>Fan Speed Monitoring</td></tr>
<tr><td>13</td><td>0D</td><td colspan="2">AD-Channel Error 1</td><td>Ro</td><td>AD Channel Errors</td></tr>
<tr><td>14</td><td>0E</td><td colspan="2">Error Register</td><td>Ro</td><td>Error Register</td></tr>
<tr><td>15</td><td>0F</td><td colspan="2">Dummy Register</td><td>R/W</td><td>Dummy Register</td></tr>
<tr><td>16</td><td>10</td><td rowspan="16">Protected Data</td><td>0</td><td rowspan="16">(R/W)</td><td rowspan="17">Protected Memory</td></tr>
<tr><td>17</td><td>11</td><td>1</td></tr>
<tr><td>18</td><td>12</td><td>2</td></tr>
<tr><td>19</td><td>13</td><td>3</td></tr>
<tr><td>20</td><td>14</td><td>4</td></tr>
<tr><td>21</td><td>15</td><td>5</td></tr>
<tr><td>22</td><td>16</td><td>6</td></tr>
<tr><td>23</td><td>17</td><td>7</td></tr>
<tr><td>24</td><td>18</td><td>8</td></tr>
<tr><td>25</td><td>19</td><td>9</td></tr>
<tr><td>26</td><td>1A</td><td>A</td></tr>
<tr><td>27</td><td>1B</td><td>B</td></tr>
<tr><td>28</td><td>1C</td><td>C</td></tr>
<tr><td>29</td><td>1D</td><td>D</td></tr>
<tr><td>30</td><td>1E</td><td>E</td></tr>
<tr><td>31</td><td>1F</td><td>F</td></tr>
<tr><td>32</td><td>20</td><td colspan="2">Protected Data Key</td><td>Wo</td></tr>
<tr><td>33</td><td>21</td><td colspan="2">Feature Ident. No.</td><td>Ro</td><td>Feature Ident. No.</td></tr>
<tr><td>34</td><td>22</td><td colspan="2">Voltage Monitoring: Battery</td><td>Ro</td><td>Voltage Monitoring</td></tr>
<tr><td>35</td><td>23</td><td colspan="2">Fan Speed 2</td><td>Ro</td><td rowspan="2">Fan Speed Monitoring</td></tr>
<tr><td>36</td><td>24</td><td colspan="2">Fan Settings</td><td>Ro</td></tr>
<tr><td>37</td><td>25</td><td colspan="2">AD-Channel Error 2</td><td>Ro</td><td>AD Channel Errors</td></tr>
<tr><td>38</td><td>26</td><td colspan="2">(reserved)</td><td>--</td><td></td></tr>
<tr><td>39</td><td>27</td><td colspan="2">(reserved)</td><td>--</td><td></td></tr>
</table>

Register Dec	Register Hex	Description	Byte	R/W Access	Section
40	28	On-Timer Seconds		Ro	Operating Time Counter
41	29	On-Timer Minutes		Ro	
42	2A	On-Timer Hours	0	Ro	
43	2B		1		
44	2C	Status Monitoring		Ro	Status Monitoring
45	2D	Watchdog Count	1	Ro	Watchdog
46	2E		0		
47	2F	Measured Voltage 5V Standby		Ro	Voltage Monitoring
48	30	Measured Voltage 1.8V		Ro	
49	31	Measured Voltage 0.9V		Ro	
50	32	Measured Voltage 1.5V		Ro	
51	33	Measured Voltage 1.05V		Ro	
52	34	Measured Voltage Core		Ro	
53	35	AD-Channel Error 3		Ro	AD Channel Errors
54	36	Board Temperature (digital) 1		Ro	Temperature Monitoring
55	37	Board Temperature (digital) 2		Ro	
56	38	Board Temperature (digital) 3		Ro	
57	39	Board Temperature (digital) 4		Ro	
58	3A	Board Temperature (digital) 5		Ro	
59	3B	AD-Channel Error 4		Ro	AD Channel Errors
60	3C	Measured Voltage 2.5V		Ro	Voltage Monitoring
61	3D	AD-Channel Error 5		Ro	AD Channel Errors
62	3E	Measured Voltage 3.3V Standby		Ro	Voltage Monitoring
63	3F	Measured Voltage 12V Display		Ro	Voltage Monitoring
64	40	Measured Voltage 1.25V		Ro	Voltage Monitoring
65	41	Measured Voltage GFX rail		Ro	Voltage Monitoring
66	42	(reserved)		-	
...	...	...		...	
255	FF	(reserved)		-	

Not all features are available on all systems. The “Feature Ident. No.”-register(0x21) identifies the features available on the board/system used. See Section “Feature Implementation List” and/or your system documentation for further details.

2.2 Feature Descriptions

The descriptions of the individual features are sorted according to functional groups.

2.2.1 Feature Ident. No.

Register		Description	R/W Access	Default
Dec	Hex			
33	21	Feature Ident. No.	Ro	see below

(this register was formerly referred to as the “Main Revision” register.

Reading this register, returns the identity number for the features implemented in the system/board. All system/boards with the same features identity number have the same features and functions implemented. Unimplemented feature registers return 0xFF, when read.

The following table shows relationship between the identity number and system/board, current as of this document’s release:

Board/System	Board Type	Indent. No.	PIC Type
PCI-954	1	0xFF (255)	PIC16F818
STX-Baseboards	1	0xFF (255)	PIC16F818
ETX-Baseboards	1	0xFF (255)	PIC16F818
ETXExpress-Baseboards	2	0x02 (2)	PIC16F913
ETX-Baseboard B611 (enhanced)	3	0x03 (3)	PIC18F6410
Single Board B628	4	0x04 (4)	PIC18F6410

2.2.2 Program Version

Register		Description	R/W Access	Default
Dec	Hex			
11	0B	PIC Program Version	Ro	see below

This registers returns the current version of the PIC-software program. This value is relevant only to the board/system used.

2.2.3 Watchdog

The SuperVision-IC provides a watchdog function which resets the system when the application stops responding. The watchdog is activated and reset by writing to the “Watchdog Settings”-register.

Registers associated with the Watchdog function:

Register Dec	Register Hex	Description	Byte	R/W Access	Default
0	00	Watchdog Settings	1	R/W	1xxx xx11
1	01		0		1111 1111
45	2D	Watchdog Count	1	Ro	0000 0011
46	2E		0		1111 1111
14	0E	Error Register		Ro	Error Register

Watchdog Settings Bits:

Byte	0x00 (Byte 1)								0x01 (Byte 0)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
	15	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
	Disable	X	X	X	X	X	watchdog time (in s)									

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 either “Watchdog Settings”-register retriggers the Watchdog.
Reading the “Watchdog Settings”-registers retrieves the current watchdog settings.

Watchdog Count Bits:

Byte	0x2D (Byte 1)								0x2E (Byte 0)							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
	15	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
	X	X	X	X	X	X	watchdog time left to reset (in s)									

This register is not implemented in all SuperVisor-ICs.

Reading the “Watchdog Count”-registers retrieves the current watchdog counter (time left in seconds).

Watchdog Function:

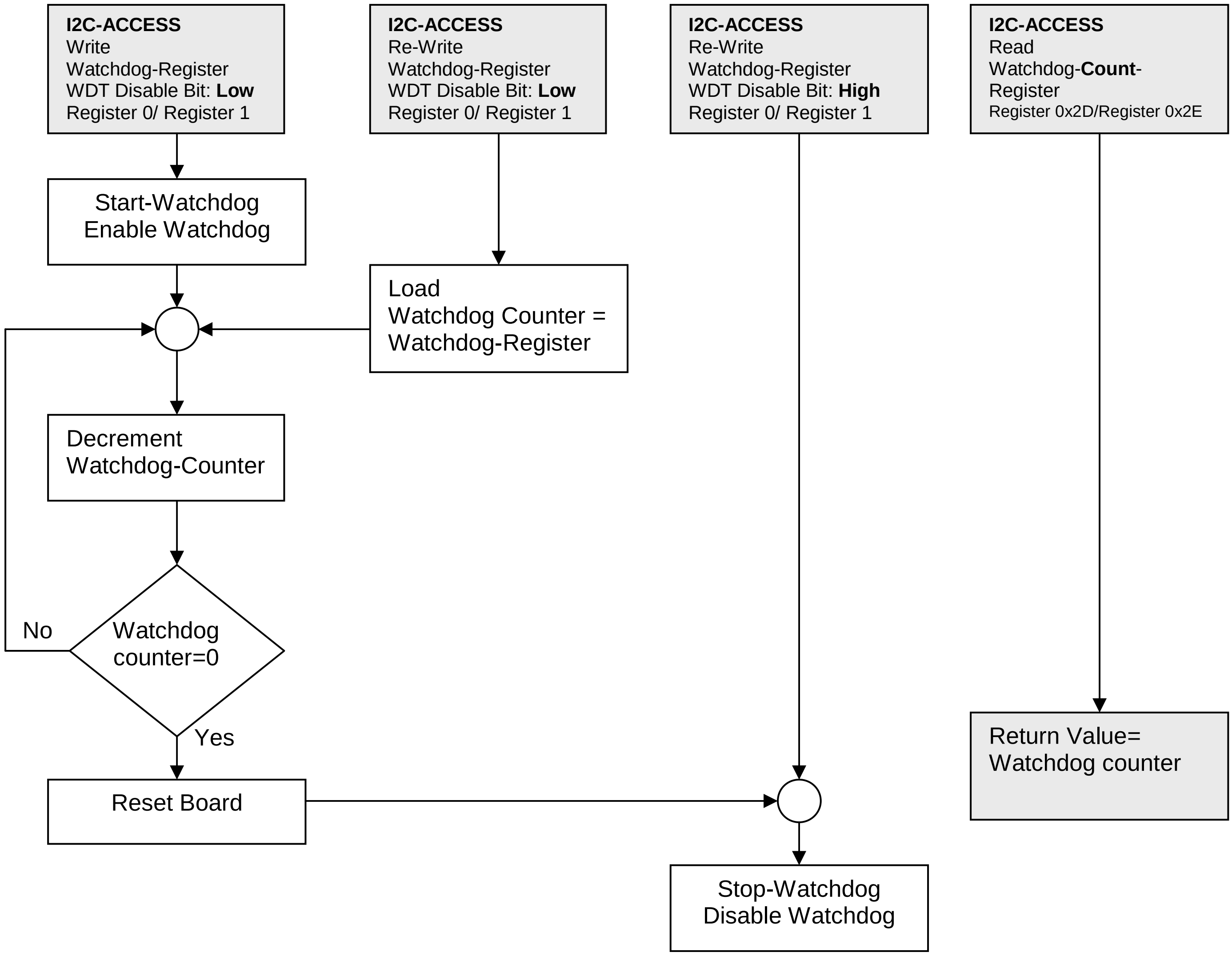
The watchdog is disabled by default.

After enabling the watch by writing to the “Watchdog Settings” high register with the bit 7 = 0, the watchdog starts to count down. The watchdog can be reset by writing to either of the “Watchdog Settings”-registers.
The “Watchdog Count” registers contain the number of seconds the watchdog has until it times out.
If the watchdog reaches 0 without being reset (time out), the system is reset.

After a watchdog reset the watchdog will be again be disabled.
Recycling the board/system power also disables the watchdog.

Writing to the “Watchdog Settings” high register with the bit 7 = 1, will disable the watchdog.

Flow Diagram:



Example:

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

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

The SuperVision "Watchdog Time" then counts down once a second (the WDT disable bit is 0).

There are now four 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 Count“ >0 and read access to the watchdog-register

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, see item A)

D) Condition: SuperVision-“Watchdog-Time “ = 0
Result: The watchdog resets the system.

Code Examples:

<u>Example 1:</u>

- 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

```
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)
```

<u>Example 2:</u>

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

```
    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
```

<u>Example 3</u> *(for versions with "Watchdog Count")*

- 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 versions with watchdog count

                // 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);


        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)
```

2.2.4 Operation Statistics

2.2.4.1 Power-On Counter

Registers associated with the Power-on Counter function:

Register Dec	Register Hex	Description	Byte	R/W Access	Default
5	05	Power-On Counter	0	Ro	see below
6	06		1		
14	0E	Error Register		Ro	Error Register

Reading these registers returns the number of cold starts.

Range: 0-65535 cold starts. Warm starts are not counted.

The Power-On Counter does not overflow. The counter stops upon reaching the maximum value (0xFFFF).

Code Example:

```
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
```

2.2.4.2 On- Timer

Registers associated with the On-Timer function:

Register Dec	Register Hex	Description	Byte	R/W Access	Default
40	28	On-Timer Seconds		Ro	see below
41	29	On-Timer Minutes		Ro	
42	2A	On-Timer Hours	0	Ro	
43	2B		1		
14	0E	Error Register		Ro	Error Register

These registers are not implemented in all SuperVisor-ICs.

Range: 0-65536 operating hours.

The On-Timer overflows when maximum value of 65536h:59m:59s has been reached. The timer then resets to 0h:0m:0s.

Code Example:

```
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
```

2.2.4.3 Operating Time Counter

Register		Description	Byte	R/W Access	Default
Dec	Hex				
2	02	Operating Time Counter	0	Ro	See below
3	03		1		
4	04		2		

Reading these registers returns the total operating time in hours.

Range: 0-16777215 operating hours.

The Operating-Time Counter does not overflow. The counter stops upon reaching the maximum value.

Code Example:

```
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
```

2.2.5 Hardware Monitoring

2.2.5.1 Voltage Monitoring

The voltage monitoring register return the current value of the voltage rail specified as a raw value. These values must be converted according to the specified formula to obtain the voltage.
Voltage rails which are not implemented return 0xFF.

Code Example: (12V)

```
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
```

2.2.5.1.1 12 Volt Monitoring

Registers associated with 12V monitoring:

Register		Description	R/W Access	Default
Dec	Hex			
7	07	Voltage Monitoring: 12V	Ro	see below
13	0D	AD Channel Error 1	Ro	AD Channel Errors
44	2C	Status Monitoring	Ro	Status Monitoring

These registers are not implemented in all SuperVisor-ICs.

Formula:

$$\text{Result\_12}V = \frac{\text{Value}}{255} * (4.75V * 4)$$

Minimum : 0x00 0 → 0V
Maximum: 0xFF 255 → 19V

2.2.5.1.2 5 Volt Monitoring

Registers associated with 5V monitoring:

Register		Description	R/W Access	Default
Dec	Hex			
8	08	Voltage Monitoring: 5V	Ro	see below
13	0D	AD Channel Error 1	Ro	AD Channel Errors
44	2C	Status Monitoring	Ro	Status Monitoring

These registers are not implemented in all SuperVisor-ICs.

Formula:

$$\text{Result\_5}V = \frac{\text{Value}}{255} * (4.75V * 2)$$

Minimum : 0x00 0 → 0V
Maximum: 0xFF 255 → 9.5V

2.2.5.1.3 3.3 Volt Monitoring

Registers associated with 3.3V monitoring:

Register Dec	Register Hex	Description	R/W Access	Default
9	09	Voltage Monitoring: 3V3	Ro	see below
13	0D	AD Channel Error 1	Ro	AD Channel Errors
44	2C	Status Monitoring	Ro	Status Monitoring

These registers are not implemented in all SuperVisor-ICs.

Formula: $$\mathrm{Result\_}3V3 = \frac{\mathrm{Value}}{255} * 4.75V$$

Minimum : 0x00 0 → 0V
Maximum: 0xFF 255 → 4.75V

2.2.5.1.4 Battery Monitoring

Registers associated with Battery monitoring:

Register Dec	Register Hex	Description	R/W Access	Default
34	22	Voltage Monitoring: Battery	Ro	see below
37	25	AD-Channel Error 2	Ro	AD Channel Errors
44	2C	Status Monitoring	Ro	Status Monitoring

These registers are not implemented in all SuperVisor-ICs.

Formula: $$\mathrm{Result\_}Bat = \frac{\mathrm{Value}}{255} * 4.75V$$

Minimum : 0x00 0 → 0V
Maximum: 0xFF 255 → 4.75V

2.2.5.1.5 5 Volt Standby Monitoring

Registers associated with 5V Standby monitoring:

Register Dec	Register Hex	Description	R/W Access	Default
47	2F	Voltage Monitoring: 5V Standby	Ro	see below
37	25	AD-Channel Error 2	Ro	AD Channel Errors
44	2C	Status Monitoring	Ro	Status Monitoring

These registers are not implemented in all SuperVisor-ICs.

Formula: $$\mathrm{Result\_}5VSby = \frac{\mathrm{Value}}{255} * (4.75V * 2)$$

Minimum : 0x00 0 → 0V
Maximum: 0xFF 255 → 9.5V

2.2.5.1.6 1.8 Volt Monitoring

Registers associated with 1.8V monitoring:

Register		Description	R/W Access	Default
Dec	Hex			
48	30	Voltage Monitoring: 1.8V	Ro	see below
37	25	AD-Channel Error 2	Ro	AD Channel Errors
44	2C	Status Monitoring	Ro	Status Monitoring

These registers are not implemented in all SuperVisor-ICs.

Formula: $$\text{Result\_}1V8 = \frac{\text{Value}}{255} * 3V$$

Minimum : 0x00 0 → 0V
Maximum: 0xFF 255 → 3V

2.2.5.1.7 0.9 Volt Monitoring

Registers associated with 0.9V monitoring:

Register		Description	R/W Access	Default
Dec	Hex			
49	31	Voltage Monitoring: 0.9V	Ro	see below
13	0D	AD Channel Error 2	Ro	AD Channel Errors
44	2C	Status Monitoring	Ro	Status Monitoring

These registers are not implemented in all SuperVisor-ICs.

Formula: $$\text{Result\_}0V9 = \frac{\text{Value}}{255} * 3V$$

Minimum : 0x00 0 → 0V
Maximum: 0xFF 255 → 3V

2.2.5.1.8 1.5 Volt Monitoring

Registers associated with 1.5V monitoring:

Register		Description	R/W Access	Default
Dec	Hex			
50	32	Voltage Monitoring: 1.5V	Ro	see below
53	35	AD Channel Error 3	Ro	AD Channel Errors
44	2C	Status Monitoring	Ro	Status Monitoring

These registers are not implemented in all SuperVisor-ICs.

Formula: $$\text{Result\_}1V5 = \frac{\text{Value}}{255} * 3V$$

Minimum : 0x00 0 → 0V
Maximum: 0xFF 255 → 3V

2.2.5.1.9 1.05 Volt Monitoring

Registers associated with 1.05V monitoring:

Register Dec	Register Hex	Description	R/W Access	Default
51	33	Voltage Monitoring: 1.05V	Ro	see below
53	35	AD Channel Error 3	Ro	AD Channel Errors
44	2C	Status Monitoring	Ro	Status Monitoring

These registers are not implemented in all SuperVisor-ICs.

Formula:

$$\boxed{Result\_1V05 = \frac{Value}{255} * 3V}$$

Minimum : 0x00 0 → 0V
Maximum: 0xFF 255 → 3V

2.2.5.1.10 Core Voltage Monitoring

Registers associated with VCore monitoring:

Register Dec	Register Hex	Description	R/W Access	Default
52	34	Voltage Monitoring: VCore	Ro	see below

These registers are not implemented in all SuperVisor-ICs.

Formula:

$$\boxed{Result\_VCore = \frac{Value}{255} * 3V}$$

Minimum : 0x00 0 → 0V
Maximum: 0xFF 255 → 3V

This voltage is monitored only and does not have limits.

2.2.5.1.11 2.5 Voltage Monitoring

Registers associated with 2.5V monitoring:

Register Dec	Register Hex	Description	R/W Access	Default
60	3C	Voltage Monitoring: 2.5V	Ro	see below
61	3D	AD Channel Error 5	Ro	AD Channel Errors
44	2C	Status Monitoring	Ro	Status Monitoring

These registers are not implemented in all SuperVisor-ICs.

Formula:

$$\boxed{Result\_2V5 = \frac{Value}{255} * 3V}$$

Minimum : 0x00 0 → 0V
Maximum: 0xFF 255 → 3V

2.2.5.1.12 3.3 Volt Standby Monitoring

Registers associated with 3.3V Standby monitoring:

Register Dec	Register Hex	Description	R/W Access	Default
62	3E	Voltage Monitoring: 3V3	Ro	see below
61	3D	AD Channel Error 5	Ro	AD Channel Errors
44	2C	Status Monitoring	Ro	Status Monitoring

These registers are not implemented in all SuperVisor-ICs.

Formula: $$\mathrm{Result\_}3V3S = \frac{\mathrm{Value}}{255} * 4.75V$$

Minimum : 0x00 0 → 0V
Maximum: 0xFF 255 → 4.75V

2.2.5.1.13 12 Volt Display Monitoring

Registers associated with 12V Display monitoring:

Register Dec	Register Hex	Description	R/W Access	Default
63	3F	Voltage Monitoring: 12V Display	Ro	see below
61	3D	AD Channel Error 5	Ro	AD Channel Errors
44	2C	Status Monitoring	Ro	Status Monitoring

These registers are not implemented in all SuperVisor-ICs.

Formula: $$\mathrm{Result\_}12VDisp = \frac{\mathrm{Value}}{255} * (4.75V * 4)$$

Minimum : 0x00 0 → 0V
Maximum: 0xFF 255 → 19V

2.2.5.1.14 1.25 Volt Monitoring

Registers associated with 1.25V monitoring:

Register		Description	R/W Access	Default
Dec	Hex			
64	40	Voltage Monitoring: 1.25V	Ro	see below
61	3D	AD Channel Error 5	Ro	AD Channel Errors
44	2C	Status Monitoring	Ro	Status Monitoring

These registers are not implemented in all SuperVisor-ICs.

Formula: $$\mathrm{Result\_1V25} = \frac{\mathrm{Value}}{255} * 3V$$

Minimum :	0x00	0	→	0V
Maximum:	0xFF	255	→	3V

2.2.5.1.15 GFX Rail Monitoring

Registers associated with GFX rail monitoring:

Register		Description	R/W Access	Default
Dec	Hex			
65	41	Voltage Monitoring: GFX Rail	Ro	see below

These registers are not implemented in all SuperVisor-ICs.

Formula: $$\mathrm{Result\_}GFX = \frac{\mathrm{Value}}{255} * 3V$$

Minimum :	0x00	0	→	0V
Maximum:	0xFF	255	→	3V

This voltage is monitored only and does not have limits.

2.2.5.2 Temperature Monitoring

2.2.5.2.1 Board Temperature (Analog)

Registers associated with analog temperature monitoring:

Register		Description	R/W Access	Default
Dec	Hex			
10	0A	Board Temperature (Analog)	Ro	see below
13	0D	AD Channel Error 1	Ro	AD Channel Errors
44	2C	Status Monitoring	Ro	Status Monitoring

These registers are not implemented in all SuperVisor-ICs.

This register returns the current board temperature measured with a precision temperature sensor (LM135).

The LM135 has a breakdown voltage of (+10 mV/°K), w hich is directly proportional to absolute temperature.

This sensor type is not implemented on every board. Please check your board specification.
This function returns an undefined value, if the board/system does not implement a sensor of this type.

Formula:

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

Minimum :	0x00	0	→	0K	→	-273°C
Maximum:	0xFF	255	→	475K	→	+202°C

2.2.5.2.2 Board Temperature (Digital)

Registers associated with digital temperature monitoring:

Register		Description	R/W Access	Default
Dec	Hex			
54	36	Board Temperature (digital) 1	Ro	see below
55	37	Board Temperature (digital) 2	Ro	
56	38	Board Temperature (digital) 3	Ro	
57	39	Board Temperature (digital) 4	Ro	
58	3A	Board Temperature (digital) 5	Ro	
53	35	AD Channel Error 3	Ro	AD Channel Errors
59	3B	AD Channel Error 4	Ro	
44	2C	Status Monitoring	Ro	Status Monitoring

These registers are not implemented in all SuperVisor-ICs.

This register returns the current board temperature measured with a digital precision temperature sensor such as the LM75 (or similar).
The LM75 returns a value which can be directly read a °C in a signed value.

This sensor type is not implemented on every board. Please check your board specification.
This function returns an undefined value, if the board/system does not implement a sensor of this type.

Formula:

$$\boxed{Temp(°C) = (signed\_char)Value}$$

Minimum :	0x80	128	→	--128°C
	0xFF	255	→	--1°C
	0x00	0	→	0°C
	0x01	1	→	+1°C
Maximum:	0x7F	127	→	+127°C

2.2.5.3 Fan Speed Monitoring

Registers associated with fan speed monitoring:

Register Dec	Register Hex	Description	R/W Access	Default
12	0C	Fan Speed 1	Ro	see below
35	23	Fan Speed 2	Ro	
36	24	Fan Settings	Ro	
44	2C	Status Monitoring	Ro	Status Monitoring

These registers are not implemented in all SuperVisor-ICs.

The “Fan Speed” registers return the fan speed in 100RPM.

Formula: $\boxed{Fan\_Speed(RPM) = Value * 100}$

For boards which do not implement a fan connector, the return value is no defined.
Some Type 1 boards do not implement a fan speed register (these will always return 0).

Fan Settings Bits:

Byte	0x24							
Bit	7	6	5	4	3	2	1	0
	X	X	X	X	X	X	Fan 1 Enable	Fan 2 Enable

This register is not implemented in all SuperVisor-ICs.

The Fan Enable Bits show the state of the Fan Monitoring DIP-Switch.
- 0 → Monitoring enabled
- 1 → Monitoring disabled

When this register is implemented and a Fan is disabled per switch, the Fan Speed register will always return 0.

Code Example:

```
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

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");
```

2.2.5.4 AD Channel Errors

Registers associated with AD-Channel errors:

Register		Description	R/W Access	Default
Dec	Hex			
13	0D	AD Channel Error 1	Ro	see below
37	25	AD Channel Error 2	Ro	
53	35	AD Channel Error 3	Ro	
59	3B	AD Channel Error 4	Ro	
61	3D	AD Channel Error 5	Ro	
44	2C	Status Monitoring	Ro	Status Monitoring

These registers are not all implemented in all SuperVisor-ICs.

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

Unimplemented bits/bytes are undefined and should be masked out to prevent reacting to an error that does not exist.

2.2.5.4.1 Monitoring Limits

The following tables list the upper and lower limits for the monitored values:

Voltage Limits:

Monitored Voltage	Lower Limit (Undervoltage)			Upper Limit (Overvoltage)		
	in decimal	in hex	in Volts	in decimal	in hex	in Volts
12V	< 155	< 0x9B	< 11.4V	> 172	> 0xAC	> 12.6V
5V	< 129	< 0x81	< 4.7V	> 142	> 0x8E	> 5.2V
3.3V	< 171	< 0xAB	< 3.1V	> 189	> 0xBD	> 3.5V
Battery	< 135	< 0x87	< 2.5V	> 189	> 0xBD	> 3.5V
5V Sby	< 129 *	< 0x81 *	< 4.7V *	> 142	> 0x8E	> 5.2V
1.8V	< 138	< 0x8A	< 1.62V	> 167	> 0xA7	> 1.96V
0.9V	< 69	< 0x45	< 0.81V	> 83	> 0x53	> 0.97V
1.5V	< 115	< 0x73	< 1.35V	> 140	> 0x8C	> 1.65V
1.05V	< 81	< 0x51	< 0.95V	> 98	> 0x62	> 1.15V
Vcore	monitored only					
2.5V	< 192	< 0xC0	< 2.25V	> 233	> 0xE9	> 2.74V
3.3V Sby	< 171	< 0xAB	< 3.1V	> 189	> 0xBD	> 3.5V
12V Display	< 155	< 0x9B	< 11.4V	> 172	> 0xAC	> 12.6V
1.25V	< 96	< 0x60	< 1.13V	> 117	> 0x75	> 1.38V
GFX	monitored only					

Temperature Limits:

Monitored Temperature	Lower Limit (Undertemp.)			Upper Limit (Overtemp.)		
	in decimal	in hex	in ℃	in decimal	in hex	in ℃
Analog	< 143	< 0x8F	< -10℃	> 189	> 0xBD	> 79℃
Digital	< -10	< 0xF6	< -10℃	> 79	> 0x4F	> 79℃

2.2.5.4.2 AD-Channel Error Bits

AD-Channel Error 1 Bits:

Byte	0x0D							
Bit	7	6	5	4	3	2	1	0
	Temp. (analog) Overtemp.	Temp. (analog) Undertemp.	3.3V Overvoltage	3.3V Undervoltage	5V Overvoltage	5V Undervoltage	12V Overvoltage	12V Undervoltage

AD-Channel Error 2 Bits:

Byte	0x25							
Bit	7	6	5	4	3	2	1	0
	0.9V Overvoltage	0.9V Undervoltage	1.8V Overvoltage	1.8V Undervoltage	5V Standby Overvoltage	5V Standby Undervoltage	Battery Overvoltage	Battery Undervoltage

AD-Channel Error 3 Bits:

Byte	0x35							
Bit	7	6	5	4	3	2	1	0
	Temp. (digital 1) Undertemp.	Temp. (digital 1) Undertemp.	Reserved	Reserved	1.05V Overvoltage	1.05V Undervoltage	1.5V Overvoltage	1.5V Undervoltage

AD-Channel Error 4 Bits:

Byte	0x3B							
Bit	7	6	5	4	3	2	1	0
	Temp. (digital 5) Overtemp.	Temp. (digital 5) Undertemp.	Temp. (digital 4) Overtemp.	Temp. (digital 4) Undertemp.	Temp. (digital 3) Overtemp.	Temp. (digital 3) Undertemp.	Temp. (digital 2) Overtemp.	Temp. (digital 2) Undertemp.

AD-Channel Error 5 Bits:

Byte	0x3D							
Bit	7	6	5	4	3	2	1	0
	1.25V Overvoltage	1.25V Undervoltage	12V Display Overvoltage	12V Display Undervoltage	3.3V Standby Overvoltage	3.3V Standby Undervoltage	2.5V Overvoltage	2.5V Undervoltage

Programming Example:

```
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");

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");
```

2.2.5.4.3 Status Monitoring

Registers associated with status monitoring:

Register Dec	Register Hex	Description	R/W Access	Default
44	2C	Status Monitoring	Ro	see below
13	0D	AD Channel Error 1	Ro	AD Channel Errors
37	25	AD Channel Error 2	Ro	
53	35	AD Channel Error 3	Ro	
59	3B	AD Channel Error 4	Ro	
61	3D	AD Channel Error 5	Ro	

These registers are not all implemented in all SuperVisor-ICs.

This register returns the current state of the status monitoring. As long as the hardware monitoring has no error, the value returned is 0.

If there is an error, the value returned is a code (≠ 0). If there is more than 1 error, only the error code with the highest priority (lowest priority number) is returned.
Each error code is also shown by a blinking LED (and beep on some systems).
The error codes and blink/beep codes are only returned or shown during the duration of the error.
The limits for the Voltage and Temperature monitoring are as in 2.1.5.4. The limits for the Fan Speed Monitoring are below.

2.2.5.4.4 Fan Speed Limits

	Lower Limit			Upper Limit		
	in decimal	in hex	in RPM	in decimal	in hex	in RPM
Fan Speed 1	< 9	< 0x09	< 900	> 200	> 0xC8	> 20,000
Fan Speed 2	< 9	< 0x09	< 900	> 200	> 0xC8	> 20,000

2.2.5.4.5 Error codes

The following table shows the error codes and their priority:

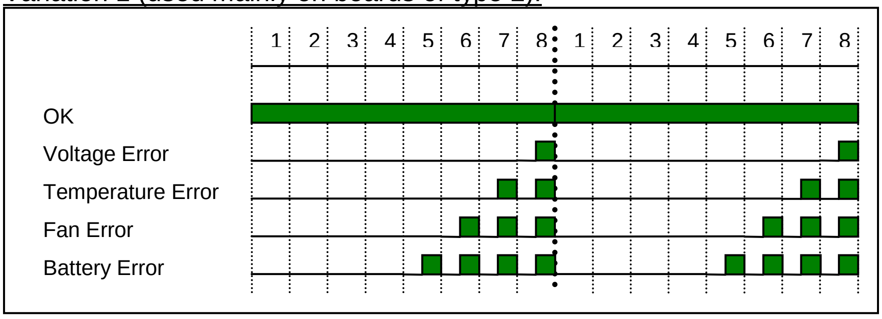
State / Error	Code in Hex	Code in dec.l	Priority (blinks)
OK	0x00	0	-
Voltage Error	0xFE	254	1
Temp. Error	0xFC	252	2
Fan Error	0xF8	248	3
Battery Error	0xF0	240	4

2.2.5.4.6 Blink/Beep Codes

There each board/system may have different variations of the blink/beep codes, depending upon several factors such as the type/color of LEDs on board, beeper implementation, additional PIC functions such as over-temperature shutdown, etc. For an exact description of the blink/beep codes on your system, see the system documentation.

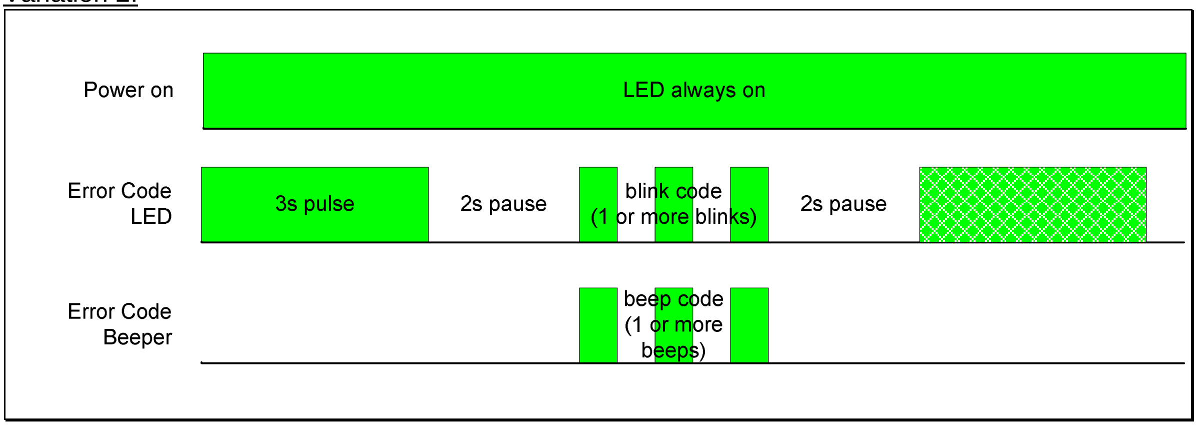
Two variations of the possible blink/beep codes available are show below as a reference, only:

Variation 1 (used mainly on boards of type 2):



In this variation the LED is on were shown as green and off where not. The beeper, if implemented, gives out a tone when the LED is off.

Variation 2:



Programming Example:

```
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");
}
```

2.2.6 Memory

2.2.6.1 Dummy Register

Register Dec	Register Hex	Description	R/W Access	Default
15	0F	Dummy Register	Ro	0x00

This register can be used as a 1 byte scratch pad. The data is stored in the controllers RAM and is not saved in event of a power failure.

Programming Example:

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

2.2.6.2 Protected Memory

Registers associated with protected memory:

Register Dec	Register Hex	Description	Byte	R/W Access	Default
32	20	Protected Data Key		Wo	0x00
16	10	Protected Data	0	(R/W)	0xFF
17	11		1		
18	12		2		
19	13		3		
20	14		4		
21	15		5		
22	16		6		
23	17		7		
24	18		8		
25	19		9		
26	1A		A		
27	1B		B		
28	1C		C		
29	1D		D		
30	1E		E		
31	1F		F		
14	0E	Error Register		Ro	Error Register

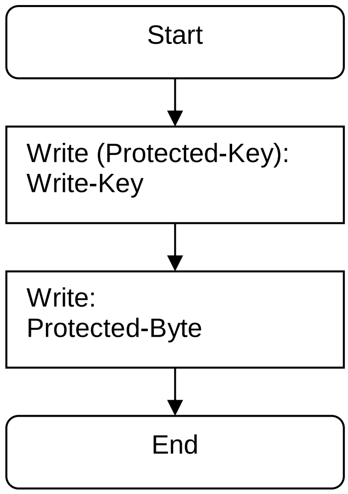
These registers provide access 16-bytes of EEPROM memory, protected with a read/write code.

Access to each byte of the protected data registers must be preceded by a write to the key register with the appropriate key. The key is invalidated after every byte access. This means that sequential read/writes to the protected data will only work for the first byte read/written and therefore do not make sense to use.

Keys:

Access Type	Key
Read	0xF0
Write	0x0F

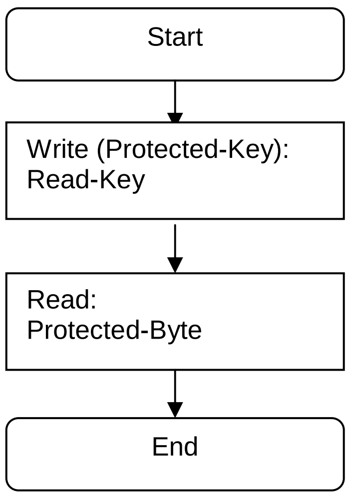
2.2.6.2.1 Write Access Protected Data



I2C_Write(0x50,0x20,Write_Key);

I2C_Write(0x50,0x10,Byte_to_write);

2.2.6.2.2 Read Access from Protected Data



I2C_Write(0x50,0x20,Read_Key);

Protected00=2C_Read(0x50,0x10)

2.2.7 Error Register

Register Dec	Register Hex	Description	Byte	R/W Access	Default
14	0E	Error Register		Ro	see below

Error Register Bits:

Byte	0x25							
Bit	7	6	5	4	3	2	1	0
	Ext. EEPROM Fatal Error	Reserved	SM(I2C)-Read Error	SM(I2C)-Write Error	EEPROM-Read Error	EEPROM-Write Error	On-Timer Overflow	Watchdog Reset

Bit Descriptions:

Bit 0: Watchdog Reset: 1, when the Watchdog function caused a reset.

Bit 1: On-Timer Overflow: 1, when the On-Timer function has overflowed.

Bit 2: EEPROM-Write Error: 1, when a non-fatal EEPROM write error occurred.
Bit 3: EEPROM-Read Error: 1, when a non-fatal EEPROM read error occurred.
Description:
This means that the EEPROM data may be corrupt or the last protected data read/write may not have been correctly written to the EEPROM.

Bit 4: SM(I2C)-Write Error: 1, when a error occurred while writing to the SuperVisor-IC.
Bit 5: SM(I2C)-Read Error: 1, when a error occurred while reading from the SuperVisor-IC.
Description:
This can mean the following:
- An I2C access was incorrectly performed
- There was a read/write to a unimplemented register.
- A write was performed to a read-only register.
- The protected data was access without the correct key.

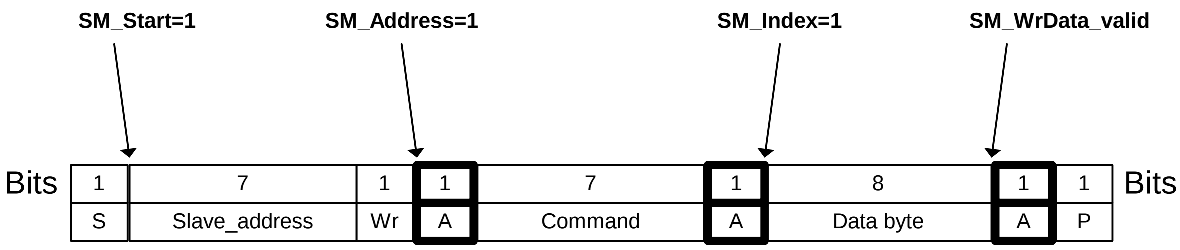
Bit 7: Ext. EEPROM Fatal Error: 1, when the SuperVisor-IC's external EEPROM has a fatal error.
Description:
This bit means the SuperVisor-IC cannot access its external EEPROM and ALL functions that use the EEPROM are either non-functional or corrupt. These functions are:
- Operating Time Counter
- Power-on Counter
- Protected Data

3 SM/I2C-Bus Access

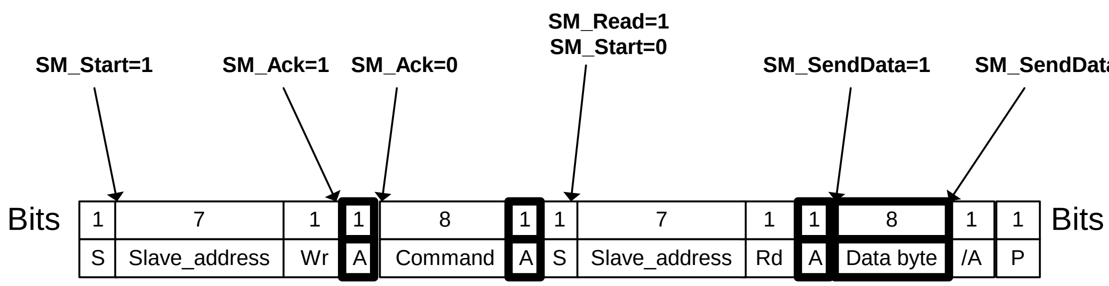
The SuperVision-IC only allows byte-wise SM-Bus reads. SM-Bus write access may be sequential.
All other forms of access, such as sequential SM-Bus reads, are not allowed and could cause unpredictable conditions.

Unimplemented features registers must not be accessed!

SM-Bus Protocol:



Write Byte Protocol



Read Byte Protocol

4 Feature Implementation List

Register	Description	Implemented (Board Type)						
Hex		1	2	3	4	5	6	7
00-01	Watchdog	●	●	●	●	●	●	●
02-05	Operating Time Counter	●	●	●	●	●	●	●
05-06	Power On Counter	●	●	●	●	●	●	●
07	Voltage Monitoring: 12V	●	●	●	●	-	●	●
08	Voltage Monitoring: 5V	●	●	●	●	●	●	●
09	Voltage Monitoring: 3.3V	●	●	●	●	●	●	●
0A	Board Temperature (analog)	●	●	-	-	-	-	-
0B	PIC Program Version	●	●	●	●	●	●	●
0C	Fan Speed 1	●	●	-	-	-	●	-
0D	AD-Channel Error 1	●	●	●	●	●	●	●
0E	Error Register	●	●	●	●	●	●	●
0F	Dummy Register	●	●	●	●	●	●	●
10-20	Protected Data (w. Key)	●	●	●	●	●	●	●
21	Feature Ident. No.	-	●	●	●	●	●	●
22	Voltage Monitoring: Battery	-	●	-	●	●	●	●
23	Fan Speed 2	-	●	-	-	-	●	-
24	Fan Settings	-	●	-	-	-	●	-
25	AD-Channel Error 2	-	●	●	●	●	●	●
26-27	(reserved)	-	-	-	-	-	-	-
28-2B	On-Timer	-	●	●	●	●	●	●
2C	Status Monitoring	-	●	●	●	●	●	●
2D-2E	Watchdog Count	-	●	●	●	●	●	●
2F	Measured Voltage 5V SBY	-	-	●	●	●	●	●
30	Measured Voltage 1.8V	-	-	-	●	●	●	●
31	Measured Voltage 0.9V	-	-	-	●	-	-	-
32	Measured Voltage 1.5V	-	-	-	●	●	●	●
33	Measured Voltage 1.05V	-	-	-	●	●	●	●
34	Measured Voltage Core	-	-	-	●	●	●	●
35	AD-Channel Error 3	-	-	●	●	●	●	●
36	Board Temperature (digital) 1	-	-	●	●	●	●	●
37	Board Temperature (digital) 2	-	-	-	●	-	●	-
38	Board Temperature (digital) 3	-	-	-	●	-	-	-
39	Board Temperature (digital) 4	-	-	-	●	-	●	-
3A	Board Temperature (digital) 5	-	-	-	●	-	●	-
3B	AD-Channel Error 4	-	-	●	●	-	●	-
3C	Measured Voltage 2.5V	-	-	-	●	-	-	-
3D	AD-Channel Error 5	-	-	-	●	●	●	●
3E	Measured Voltage 3.3V Standby	-	-	-	-	●	-	●
3F	Measured Voltage 12VDisplay	-	-	-	-	●	-	-
40	Measured Voltage 1.25V	-	-	-	-	-	●	-
41	Measured Voltage GFX rail	-	-	-	-	-	●	-
40--FF	(reserved)	-	-	-	-	-	-	-

Current Board Types:

Feature ID	Board/System	PIC
1 (255)	PCI-945, STX-Baseboards, ETX-Baseboards	PIC16F818
2	ETXe-BB B542	PIC16F913
3	ETX-BB B611R2	PIC18F6410
4	SBC B628	PIC18F6410
5	SBC B635	PIC18F6410
6	SBC B636	PIC18F6410
7	SBC B649	PIC18F6410

5 History

Main Revision 1	2	3	Version	Description	Date
x			51	Initial Release Revision 1	18.05.2004
x			01	I2C-Sequentiel Read Handling I2C-Exception Handling Interrupt-Exception Handling Fast Protected Memory Access	26.07.2004
x			02	Reset Handling for PCI954	08.12.2004
x			03	Improved storage routine for Power on Counter + Operating Time Counter	03.06.2005
	x		01	Initial Release Revision 2	12.09.2006
	x		02	Expand PIC-Supply Voltage, only BTX-Boards	13.10.2006
		x	3.0	Initial Release Revision 3	14.02.2008
		x	3.1	Corrected typo in error register; Changed error register bit 6 to reserved; Added 3V3 S5 and 12V Display Registers with error bit in ADC Error 5; added Feature ID5	07.07.2008
		x	3.2	Corrected hyperlinks	23.07.2008
		x	3.3	Added 1.25V and GFX; added Feature ID6; removed VCore limits; added Feature ID7	13.11.2008

6 Demo Program

Version: ETX
I^2C-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);

}
```