Kontron®
SuperVision

Technical Manual

Revision. 0.1

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

	Datum	**Name**	**Abteilung**
Prepared:	26.07.04	Michael Koch	TH (KECR)
Checked:			
Approved:			

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

Table of Contest:

1.0 Function-Description

The Supervision-IC provided a lot of useful features like a Watchdog, Operating-Time-Counter, Power-On-Counter, Voltage Metering, Temperature Measuring, Fan-Speed and Version-Number. As additional feature is a protected EEprom memory available.

All this features are implemented in a PIC16F818 from Microchip. Via the SM-Bus you can access the internal registers of the PIC.

The SM-Bus address is **0x50**[1].

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

Register:	Short-Description:
00	Watchdog High-Register
01	Watchdog Low-Register
02	Operating-Time-Counter Byte 0
03	Operating-Time-Counter Byte 1
04	Operating-Time-Counter Byte 2
05	Power-On-Counter Byte 0
06	Power-On-Counter Byte 1
07	Measured 12.0 Volts
08	Measured 5.0 Volts
09	Measured 3.3 Volts
0A	Board Temperature
0B	Program-Revision
0C	Fan-Revolution
0D	AD-Channel Error-Byte
0E	Error-Register
0F	Dummy Register
10	Protected-Data-Byte 0
11	Protected-Data-Byte 1
12	Protected-Data-Byte 2
13	Protected-Data-Byte 3
14	Protected-Data-Byte 4
15	Protected-Data-Byte 5
16	Protected-Data-Byte 6
17	Protected-Data-Byte 7
18	Protected-Data-Byte 8
19	Protected-Data-Byte 9
1A	Protected-Data-Byte A
1B	Protected-Data-Byte B
1C	Protected-Data-Byte C
1D	Protected-Data-Byte D
1E	Protected-Data-Byte E
1F	Protected-Data-Byte F
20	Key for protected bytes

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

1.1 Watchdog

The Supervision-IC provided a watchdog component that allows the system to be reset when the running application has stopped responding. This works by setting the Watchdog-Registers.

Register	Direction	Name	Default:
00	Read / Write	Watchdog High Register	1111 1111
01	Read / Write	Watchdog Low Register	1111 1111

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Watchdog-High-Register (Register 1)								**Watchdog-Low-Register (Register 0)**							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WDT Disable	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 retriggers the Watchdog-Timer.

Reading any Watchdog-Register retrieves the watchdog parameters that were set by the application or the default parameters.

Examples:

Set Watchdog timeout after 53 seconds:

Watchdog-Time = 1 * 53 + 256 * 0 = 53 seconds

```
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
```

Set Watchdog timeout after 584 seconds:

Watchdog-Time = 1 * 72 + 256 * 2 = 584 seconds

```
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
```

Disable Watchdog

Disable-Bit : 1000 000 = 128 decimal

```
I2C_Write(0x50,0x00,128);       // Write to Watchdog-High-Register the Value 128
```

Hint:

If the watchdog resets the system a flag form the error-byte is set.

1.2 Operating-Time-Counter

Returns the operating time in hours.

Register	Direction	Name	State at first power-on:
02	Read only	Operating-Time-Counter Byte 0	0000 0000
03	Read only	Operating-Time-Counter Byte 1	0000 0000
04	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																							

Area from 0-16777215 operating hours.
The Operating-Time-Counter has no overflow, the value is hold on max value.

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
```

1.3 Power-On-Counter

Showing the number of power-on events.

Register	Direction	Name	State at first power-on:
05	Read only	Power-On-Counter Byte 0	0000 0000
06	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															

Area from 0-65535 cold starts. Warm starts are not counted.
The Power-On-Counter has no overflow, the value is hold on max value.

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
```

1.4 Voltage metering: 12 Volts

Measure the 12 V – voltage.

Register	Direction	Name
07	Read only	Measured12V0

7	6	5	4	3	2	1	0
Measured12V0							

Reference: approximate 4.75 Volts

Voltage divider: 1 / 4.

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

Example:

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

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

// Result12V0 should be a float or double variable
```

1.5 Voltage metering: 5 Volts

Measure the 5 V – voltage.

Register	Direction	Name
08	Read only	Measured5V0

7	6	5	4	3	2	1	0
Measured5V0							

Reference: approximate 4.75 Volts

Voltage divider: 1 / 2.

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

Example:

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

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

// Result5V0 should be a float or double variable
```

1.6 Voltage metering: 3.3 Volts

Measure the 3.3V – Voltage.

Register	Direction	Name
09	Read only	Measured3V3

7	6	5	4	3	2	1	0
Measured3V3							

Reference: approximate 4.75 Volts

Voltage divider: 1 / 1.

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

Example:

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

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

// Result3V3 should be a float or double variable
```

1.7 Board temperature

Measure the board temperature with a precision temperature sensor (LM135).
The LM135 has a breakdown voltage directly proportional to absolute temperature at +10 mV/°K.

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

The function returns an undefined value if the board haven't got a sensor.

Register	Direction	Name
0A	Read only	Board_Temp

7	6	5	4	3	2	1	0
Board_Temp							

Reference: approximate 4.75 Volts
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:

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

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


// Board_Temp should be a float or double variable
```

1.8 PIC-Program Revision

Showing the current version of the PIC-Program.

Register	**Direction**	**Name**
0B	Read only	Revision

7	6	5	4	3	2	1	0
Revision							

Example:

```
Revision= I2C_Read(0x50,0x0B);          // Read the Version-Register
```

1.9 Fan-Revolution

Measure the fan revolution per minute divided by 100.

The fan-connector is not implemented at every board. Please check your board specification.

The function returns 0 if the board haven't got a fan or fan-connector.

Register	Direction	Name
0C	Read only	Revolution

7	6	5	4	3	2	1	0
Revolution							

Divider: 1 / 100

Example:

```
Revolution= I2C_Read(0x50,0x0C);        // Read the revolution information

FanSpeed=Revolution*100;                // Calculate the fan speed
```

1.10 AD-Channel-Errors

Check all limits of all AD-Channels.

Register	Direction	Name	Default:
0D	Read only	AD_Errors	0000 0000

AD_Errors							
7	6	5	4	3	2	1	0
Temperature to high	Temperature to 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 of the AD-Channels:

Bit 0:	under-voltage	Measured12V0 < 155	→	ca. 11.4 Volts
Bit 1:	over-voltage	Measured12V0 > 172	→	ca. 12.6 Volts

5 Volts:

Bit 2:	under-voltage	Measured5V0 < 129	→	ca. 4.7 Volts
Bit 3:	over-voltage	Measured5V0 > 142	→	ca. 5.2 Volts

3.3 Volts:

Bit 4:	under-voltage	Measured3V3 < 171	→	ca. 3.1 Volts
Bit 5:	over-voltage	Measured3V3 > 189	→	ca. 3.5 Volts

Board-Temperature[2]:

Bit 6:	low-temperature	Board_Temp< 143	→	ca. –10 °C
Bit 7:	high-temperature	Board_Temp> 189	→	ca. +79 °C

Example:

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

        if (AD_ERROR & 0x01) printf("\n12 Volt Under-voltage");
        if (AD_ERROR & 0x02) printf("\n12 Volt Over-voltage");
        if (AD_ERROR & 0x04) printf("\n5 Volt Under-voltage");
        if (AD_ERROR & 0x08) printf("\n5 Volt Over-voltage");
        if (AD_ERROR & 0x10) printf("\n3.3 Volt Under-voltage");
        if (AD_ERROR & 0x20) printf("\n3.3 Volt Over-voltage");
        if (AD_ERROR & 0x40) printf("\nBoard-Temperature to high");
        if (AD_ERROR & 0x80) printf("\nBoard-Temperature to low");
```

[2] The function returns an undefined value if the board haven't got a temperature-sensor.

1.11 Error-Register

Register	Direction	Name	Default:
0E	Read only	Error_Register	0000 0000

Error Register

7	6	5	4	3	2	1	0
For future use	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: For future use

Example:

```
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");
}
```

1.12 Dummy-Register

Test-Register

Register	Direction	Name	Default:
0F	Read / Write	Dummy_Register	0000 0000

You can read and write to this register with any effects.

Example:

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

1.13 Protected Memory

Access to the protected Eeprom memory.

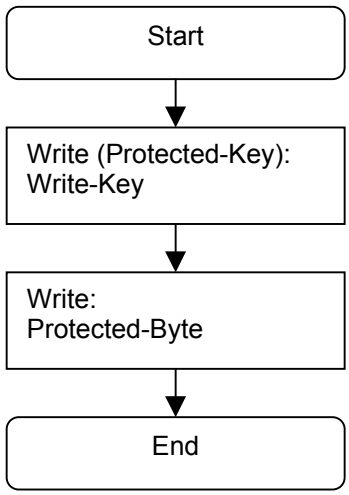
Register	Direction	Name
20	Write	Protected_Key

To write or read the protected memory you need the key to write or read the protected memory.

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

1.13.1 Write access to protected memory

Write_Key: **0x0F**



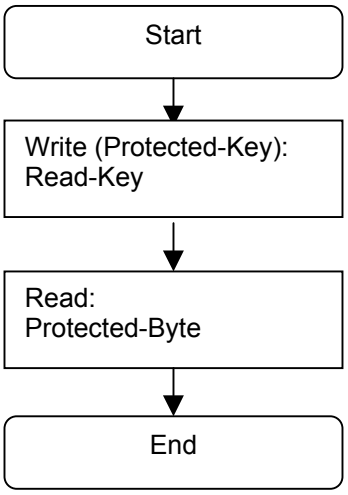
I2C_Write(0x50,0x20,Write_Key);

I2C_Write(0x50,0x10,Byte_to_write);

Every protected memory access need to write the protected write-key and the byte to write.

1.13.2 Read access to protected memory

Read_Key: **0xF0**



I2C_Write(0x50,0x20,Read_Key);

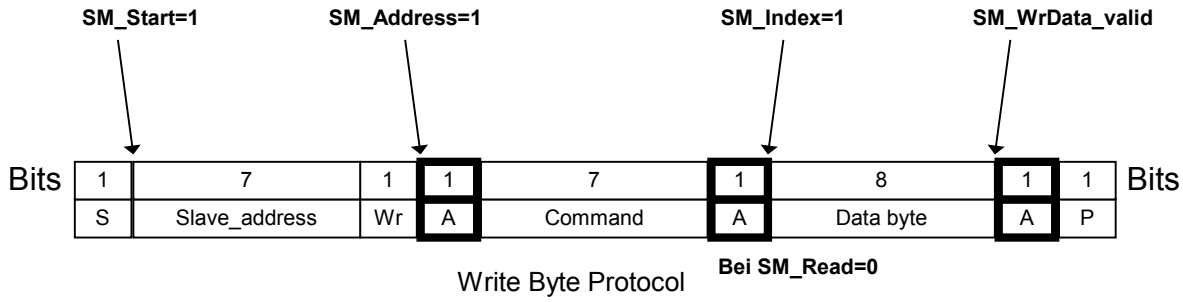
Protected00=2C_Read(0x50,0x10)

Every protected memory access need to write the protected read-key.

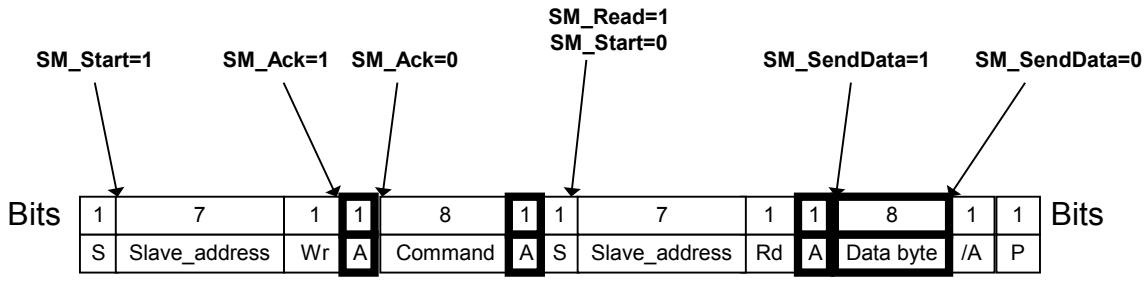
2.0 SM(I2C)-Bus –Access

The SuperVision-IC only allows SM-Bus-Register-Read or SM-Bus-Register-Write. All other accesses like sequential SM-Bus-Read are not possible and could cause unpredictably conditions.

2.1 SM-Bus-Protocol



Write Byte Protocol



Read Byte Protocol

3.0 PIC- Revision History

Revision	Description	Date
51	Initial Release	18.05.04
01	I2C-Sequentiel Read Handling	26.07.04
	I2C-Exception Handling	
	Interrupt-Exception Handling	
	Fast - Protected Memory Access	
02	Reset-Handling PCI954	08.12.04

4.0 PIC-Demoprogram

Version: ETX
I²C-Access via JIDA32-Library

```
// **
// ** Title: SuperVision - Demoprogram ETX
// **
// ** Author: Dipl-Ing(FH) Michael Koch
// ** Kontron Roding
// **
// ** Last Change: 23-07-04
// **
// 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;

#define Base_Number_I2C    0x0
#define PIC_Address_I2C    0x50
#define Protected_Write    0x0F
#define Protected_Read             0xF0

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

```
// 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)
{
        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.

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
//

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);

            // Output Watchdog Settings

            Watchdog(&Output_Byte[0],&Output_Byte[1]);
            sprintf(Output_String,"%3d",Output_Byte[0]);
            TextOut(hdc,40,40,"Watchdog-High-Register:",23);
            sprintf(Output_String,"%-3d",Output_Byte[0]);
            TextOut(hdc,240,40,Output_String,3);
            TextOut(hdc,300,40,"Watchdog-Low-Register:",22);
            sprintf(Output_String,"%3d",Output_Byte[1]);
            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,40,60,"Operating-Time-Counter[h]:",26);
            sprintf(Output_String,"%-8d",Output_Long);
            TextOut(hdc,240,60,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,40,80,"Power-On-Counter:",17);
            sprintf(Output_String,"%-5d",Output_Long);
            TextOut(hdc,240,80,Output_String,5);

            // Output Voltage metering 12 Volts


            TextOut(hdc,40,100,"12.0V - measured[V]:",20);
            sprintf(Output_String,"%-2.2f",Measured12V0());
            TextOut(hdc,240,100,Output_String,5);

            // Output Voltage metering 5 Volts

            TextOut(hdc,40,120,"5.0V - measured[V]:",19);
            sprintf(Output_String,"%-1.2f",Measured5V0());
```

```
TextOut(hdc,240,120,Output_String,4);

// Output Voltage metering 3.3 Volts

TextOut(hdc,40,140,"3.3V - measured[V]:",19);
sprintf(Output_String,"%-1.2f",Measured3V3());
TextOut(hdc,240,140,Output_String,4);

// Output Board Temperature

TextOut(hdc,40,160,"Board-Temp[°C]:",15);
sprintf(Output_String,"%-5f",BoardTemp());
TextOut(hdc,240,160,Output_String,5);

// Output Revision

Output_Byte[0]=Revision();
sprintf(Output_String,"%-3d",Output_Byte[0]);
TextOut(hdc, 40,180,"Revision:",9);
TextOut(hdc,240,180,Output_String,3);

// Output Revolution

sprintf(Output_String,"%-5d",Revolution());
TextOut(hdc, 40,200,"Fan-Revolution [rpm]:",21);
TextOut(hdc,240,200,Output_String,5);

// 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 Dummy

Temp=ReadDummy();
sprintf(Output_String,"%-3d",Temp);
TextOut(hdc, 40,260,"1.Read-Dummy:",13);
TextOut(hdc,240,260,Output_String,3);
TextOut(hdc, 40,280,"1.Write-Dummy[OLD+1]:",21);
WriteDummy(++Temp);
sprintf(Output_String,"%-3d",Temp);
TextOut(hdc,240,280,Output_String,3);
Temp=ReadDummy();
sprintf(Output_String,"%-3d",Temp);
TextOut(hdc,300,280,"2.Read-Dummy:",13);
TextOut(hdc,460,280,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, 40,300,"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);
                                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);
}


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);
}
```