HM-PCI104 Manual

Health Monitor
For
PCI104 BUS

Manufactured by
TRI-M ENGINEERING
Engineered Solutions for Embedded Applications

Technical Manual

P/N: HMPCI-MAN
Revision: 25 April 2006

TRI-M ENGINEERING
1407 Kebet Way, Unit 100
Port Coquitlam, BC V3C 6L3
Canada
http://www.Tri-M.com
Tel 604.945.9565
North America 800.665.5600
Fax 604.945.9566

PREFACE

This manual is for integrators of applications of embedded systems. It contains information on hardware requirements and interconnection to other embedded electronics.

DISCLAIMER

Tri-M Engineering makes no representations or warranties with respect to the contents of this manual, and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Tri-M Engineering shall under no circumstances be liable for incidental or consequential damages or related expenses resulting from the use of this product, even if it has been notified of the possibility of such damages. Tri-M Engineering reserves the right to revise this publication from time to time without obligation to notify any person of such revisions. If errors are found, please contact Tri-M Engineering at the address listed on the title page of this document.

CHAPTER 1: INTRODUCTION

1.1 General Description

The HM-PCI104 is a compact board design to monitor the overall health of a PCI/104 stack. It can simultaneously monitor five internal analog voltage input, four external analog voltage inputs, one on board temperature sensor, 2 external temperature sensors and 2 fan tachometer inputs. It has pulse width modulation outputs to control the speed rotation of two fans. The HM-PCI104 also provides visual and audio warning when any measurement goes out of a programmable range.

1.2 Specifications

1.2.1 Analog voltage inputs

- Five internal : +5V, +12V, -12V, +3.3V, VIO.
- Four external : Vcore, Vext, Vbat, 5V stby.

1.2.2 Temperature sensors

- One on board temperature sensor using a 10K NTC thermistor (optional 2N3904 NPN-type transistor).
- Two external sensor inputs using thermistor, 2N3904 NPN-type transistor or CPU thermal diode.

1.2.3 Tachometer inputs

- 2 Fan speed monitoring inputs.

1.2.4 Fan control outputs

- 2 PWM outputs for fan speed control.

1.2.5 Alarm

- LED + Beep tone warning.
- SMI# and OVT# signals to active system protection.

1.2.6 Programming and control

- I^2C bus

1.2.7 Mechanical/Environmental

- PC/104 form factor compliant, 3.55" x 1.5" x 0.9" (90mm x 38mm x 23mm)
- Standard PC/104 16-bit stackthrough connector for PC/104-compliant modules
- Operating temperature -40° to 185°F (0° to 70°C)
- Storage temperature: -67° to 185°F (-55° to 150°C)
- WEIGHT: 0.1 LB. (45G)

CHAPTER 2: EMBEDDED FEATURES

2.1 Customizable External Analog Inputs

- VCORE: default set to +3V (max +4V), can be adjusted with R20.
- VBAT: default set to +3V (max +4V), can be adjusted with R24.
- 5VSB: default set to +5V (max +6.5V), can be adjusted with R29 and R25.
- VEXT: default set to +120V (max +130V), can be adjusted with R16 and R18.

2.2 Customizable internal Temperature sensor

- 10K NTC thermistor or 2N3904 NPN-type transistor.

2.3 Selectable I^2C address

- From 50_h to $5E_h$.

2.4 Visual and Audio Alarm

- Programmable Out of range activates LED and Speaker.

2.5 Software Compatibility

- Linux (kernel 2.6, lm_sensor).
- Windows (Motherboard Monitor, Hardware Doctor, Hardware monitor).

Note: Normally every software supporting the Winbond W83782D monitoring IC on I^2C bus

CHAPTER 3: INSTALLATION

3.1 Locating the Connectors & JUMPERS

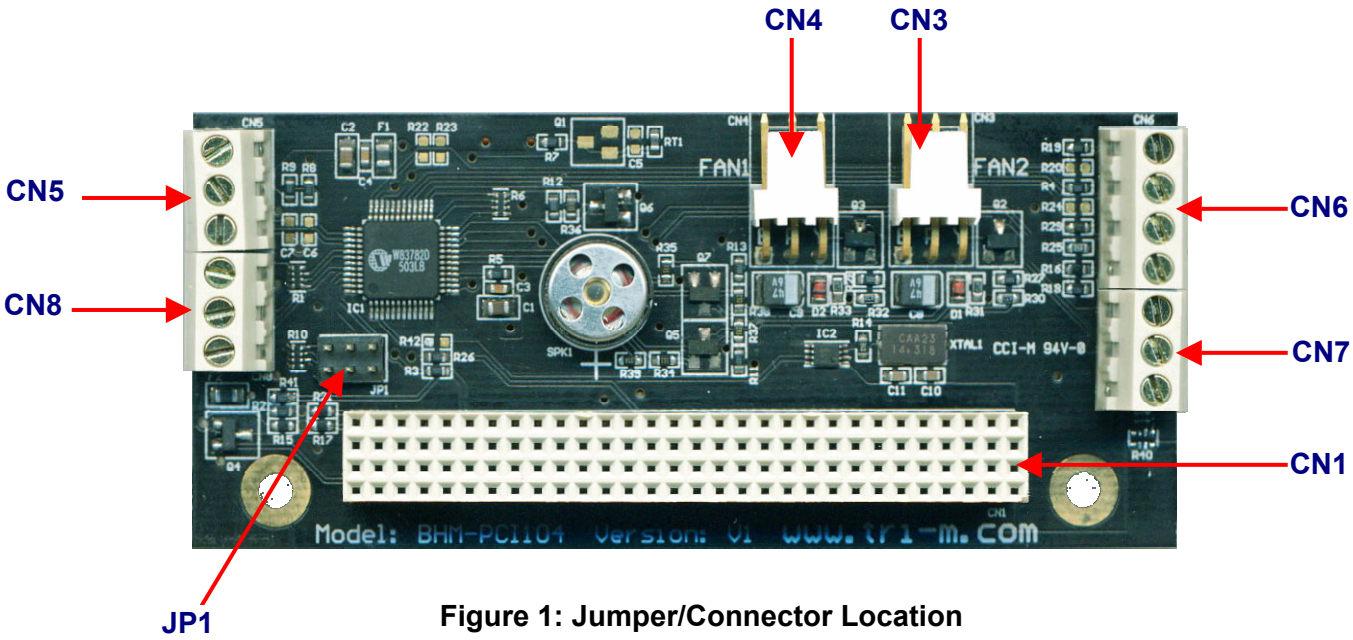


Figure 1: Jumper/Connector Location

CHAPTER 4: JUMPERS

Jumper is provided on the HM-PCI104 to select the I^2C address of the internal address register from 50_h to $5E_h$.

Jumpers	
Label	**Function**
JP1	I^2C base address

Table 1: List of Jumpers

4.1 I^2C address select (JP1)

Jumper JP1 allows to select the I^2C base address of the internal address register. The internal address register provides read/write access to every registers of the HM-PCI104 excepted BANK1 and BANK2. The internal address register can also be modified through the control register 48_h.

The I^2C base address is defined as follow: 0101.[A2][A1][A0][R/W]

I^2C address	**JP1**		
	A2	A1	A0
$5E_h$	OFF	OFF	OFF
$5C_h$	OFF	OFF	ON
$5A_h$	OFF	ON	OFF
58_h	OFF	ON	ON
56_h	ON	OFF	OFF
54_h	ON	OFF	ON
52_h	ON	ON	OFF
50_h	ON	ON	ON

Table 2: I^2C address Select

CHAPTER 5: CONNECTORS

Connectors on the HM-PCI104 are provided to communicate with the board, measure bus/external inputs, generate external events and control the fans.

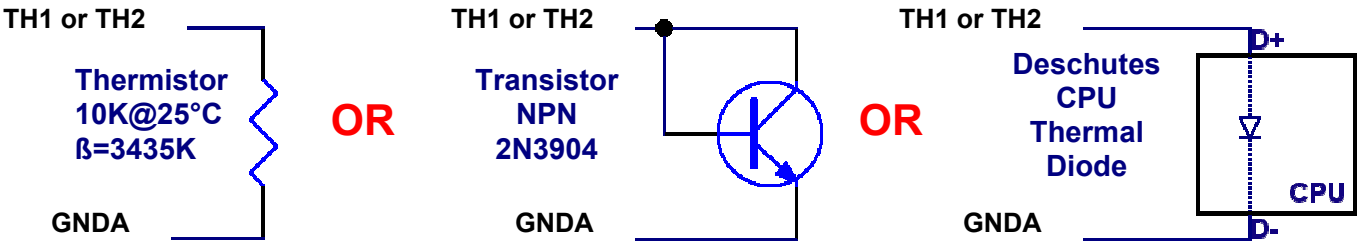
HM104 Connector List	
Connector Label	Function
CN1	PCI104+ BUS
CN3	FAN 2
CN4	FAN 1
CN5	External Temperature sensor input TH1 and TH2
CN6	External analog voltage inputs VCORE, VBAT, 5Vsb and VEXT
CN7	I^2C BUS
CN8	External Event output SMI# and OVT#

Table 3: HM-PCI104 Connector List

Note: CN5 to CN8 are screw terminal connectors, each input is individually named on the back side of the board.

5.1 External Temperature sensor input connectors (CN5)

The remote temperature sensing can be performed by thermistors, or 2N3904 NPN-type transistors, or directly from Deschutes CPU thermal diode output.



Temperature sensor connector (CN5)			
Pin #	**Marking**	**Type**	**Description**
1	GNDA	Power	Analog Ground
2	TH2	Input	Temperature sensor input 2
3	TH1	Input	Temperature sensor input 1

Table 4: Temperature sensor connector

5.2 Fan connectors (CN3 and CN4)

Use to connect of the shelve 12VDC PC fans, they allow to measure and control the fan speed rotation.

Fan connector (CN3 and CN4)			
Pin #	**Marking**	**Type**	**Description**
1	B	Power	Power Ground
2	R	Output	Programmed Voltage output
3	Y	Input	Tachometer input

Table 5: Fan connector

5.3 Analog Voltage input connectors (CN6)

These connectors allow to measure up to four external voltage inputs. Each input is connected to the Winbond monitoring IC through a resistor divider to lower the input voltage in a 0 to 4V range.

Analog Voltage input connector (CN6 and CN7)				
Pin	**Type**	**Range**	**Step**	**Description**
VCORE	Input	0 to 4V	0.016V	Vcore dedicated input
VBAT	Input	0 to 4V	0.016V	Battery dedicated input
5VSB	Input	0 to 6.5V	0.027	5V standby dedicated input
VEXT	Input	0 to 130V	0.51 V	High voltage dedicated input

Table 6: Analog Voltage inputs connectors

5.4 External event connector (CN8)

The HM-PCI104 provides two output activated on programmable events.

External event connector (CN8)			
Pin #	**Marking**	**Type**	**Description**
1	GNDD	Power	Digital Ground
2	OVT#	Output	Over temperature flag
3	SMI#	Output	Out of Range alarm flag

Table 7: External Event connector

5.5 I²C Bus connector (CN7)

This connectors interface the HM-PCI104 with a computer to configure the board and retrieve the measurement.

I²C BUS connector (CN7)			
Pin #	**Marking**	**Type**	**Description**
1	SDA	I/O	I²C Data signal
2	SCL	Input	I²C Clock signal
3	GND	Power	Digital Ground

Table 8: I²C Bus connector

5.6 PC/104+ connector (CN1)

This is only used the measure the bus power input +3.3V, +5V, +12V, -12V and VI/O.

PC/104+ Bus Connector (CN1)				
Pin	**A**	**B**	**C**	**D**
1	GND	NC	+5V	AD00
2	VI/O	AD02	AD01	+5V
3	AD05	GND	AD04	AD03
4	C/BE0	AD07	GND	AD06
5	GND	AD09	AD08	GND
6	AD11	VI/O	AD10	PMEX
7	AD14	AD13	GND	AD12
8	+3.3V	CBE1	AD15	+3.3V
9	SERR	GND	SB0	PAR
10	GND	PERR	+3.3V	SDONE
11	STOP	+3.3V	LOCK	GND
12	+3.3V	TRDY	GND	DEVSEL
13	FRAME	GND	IRDY	+3.3V
14	GND	AD16	+3.3V	C/BE2
15	AD18	+3.3V	AD17	GND
16	AD21	AD20	GND	AD19
17	+3.3V	AD23	AD22	+3.3V
18	N/A	GND	N/A	IDSEL2
19	AD24	C/BE3	VI/O	IDSEL3
20	GND	AD26	AD25	GND
21	AD29	+5V	AD28	AD27
22	+5V	AD30	GND	AD31
23	N/A	GND	N/A	VI/O
24	GND	REQ2	+5V	N/A
25	N/A	VI/O	GNT2	GND
26	+5V	N/A	GND	N/A
27	CLK_PCI_2	+5V	CLK_PCI_3	GND
28	GND	INTD	+5V	RST
29	+12V	N/A	N/A	INTB
30	-12V	NC	NC	GND

Table 9: PC/104+ Bus

CHAPTER 6: INTERFACE

The HM-PC104 is interfaced through the I^2C bus. The board uses three serial bus addresses to provide access to all the internal registers.

6.1 Internal Address Register

The first address selected with the jumper JP1 or defined the control register 48_h is used to access all the internal registers excluding BANK 1 and BANK2.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	1	0	1	A2	A1	A0	R/W#

6.2 External Temperature Sensor TH1

The second address defined in the control register $4A_h$ bit 2-0 only read/write registers of external temperature sensor TH1.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	0	1	$4A_h$-2	$4A_h$-1	$4A_h$-0	R/W#

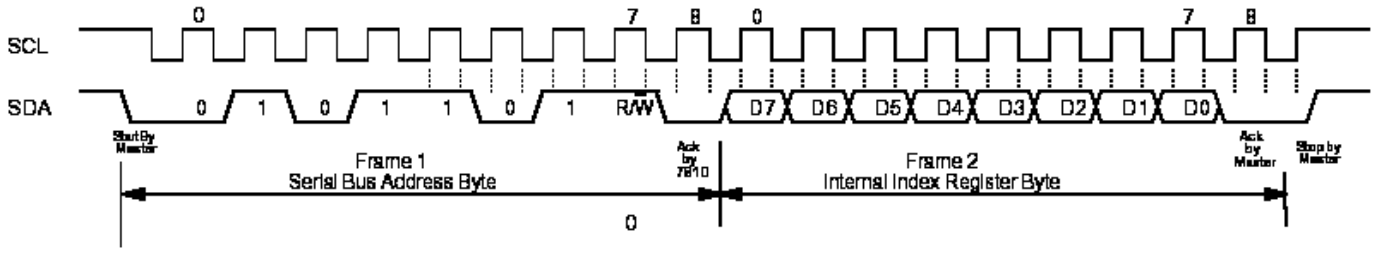
6.3 External Temperature Sensor TH2

The third address defined in register $4A_h$ bit 6-4 only read/write registers of external temperature sensor TH2.

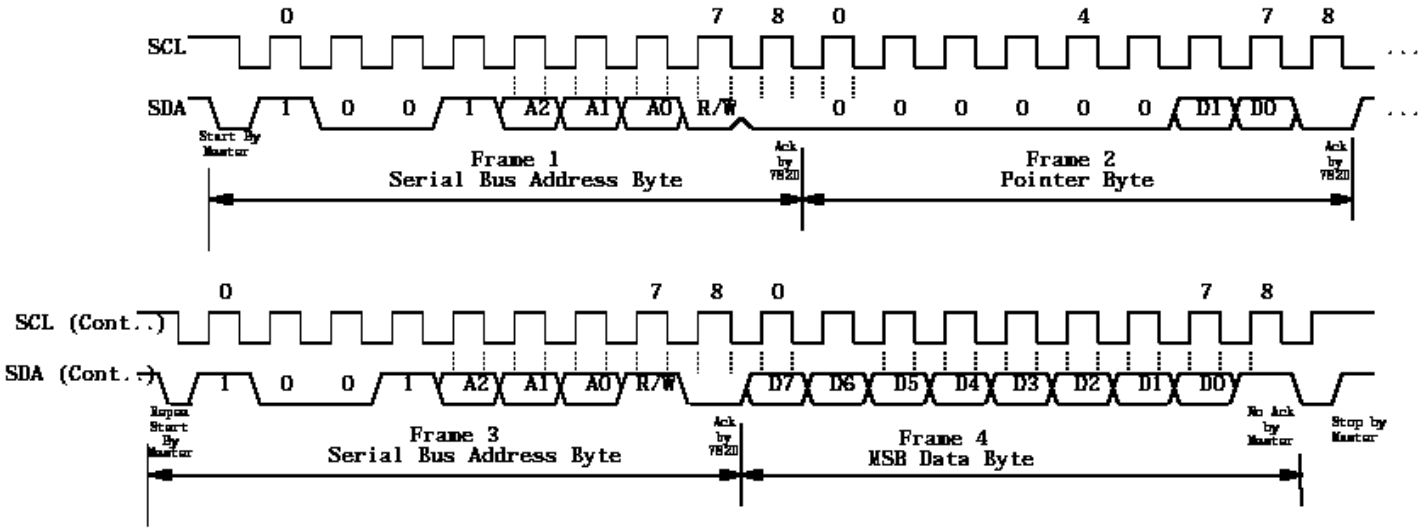
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	0	1	$4A_h$-6	$4A_h$-5	$4A_h$-4	R/W#

6.4 I^2C BUS Read

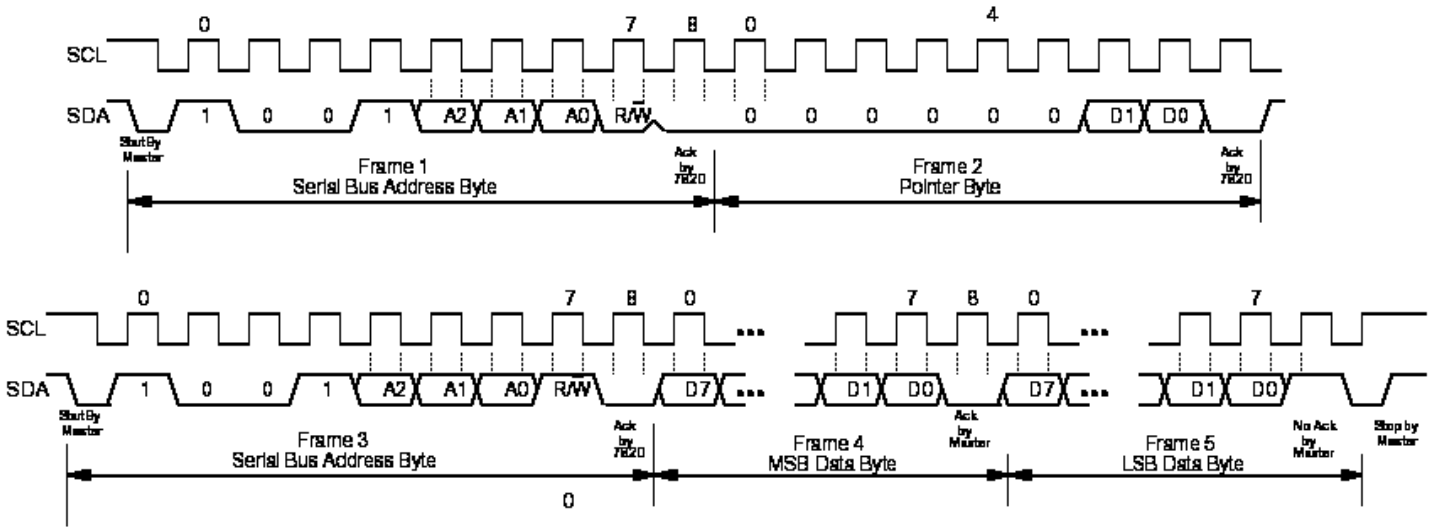
Read the index register:



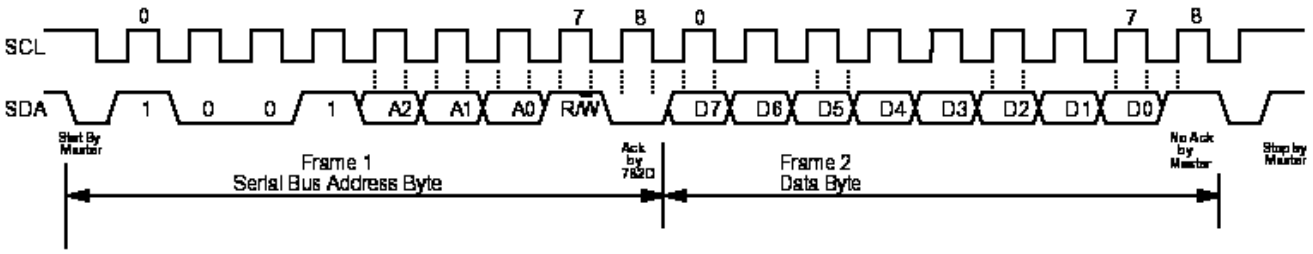
Read the content of a selected register:



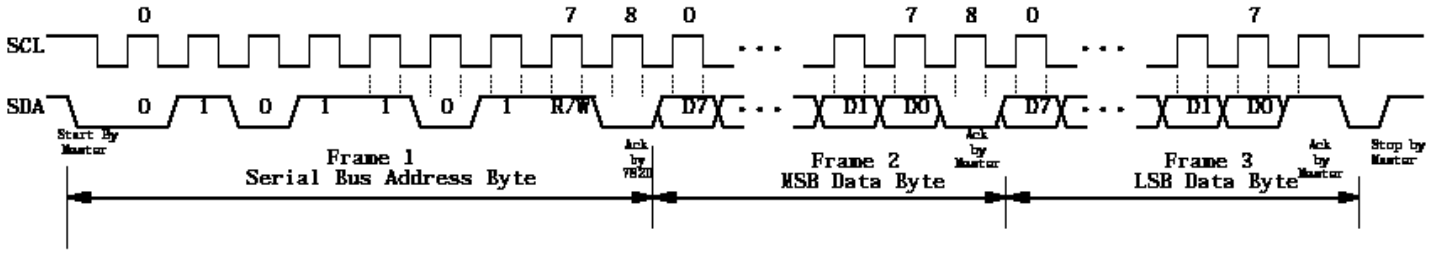
Read two byte from a specified location:



Read data from the current pointer location:

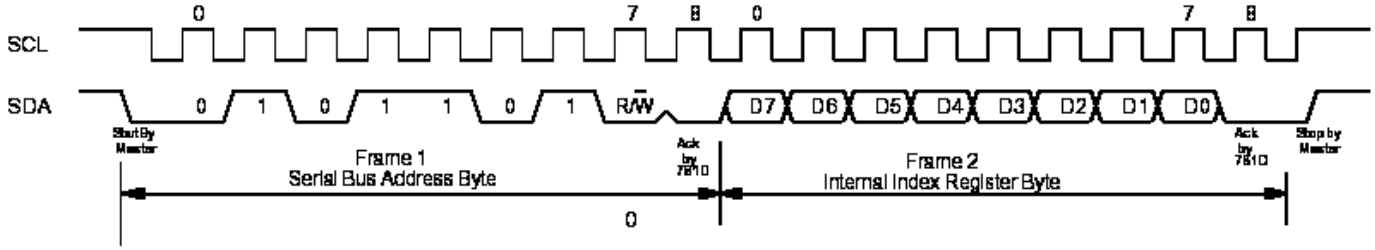


Read two bytes from the current pointer location:

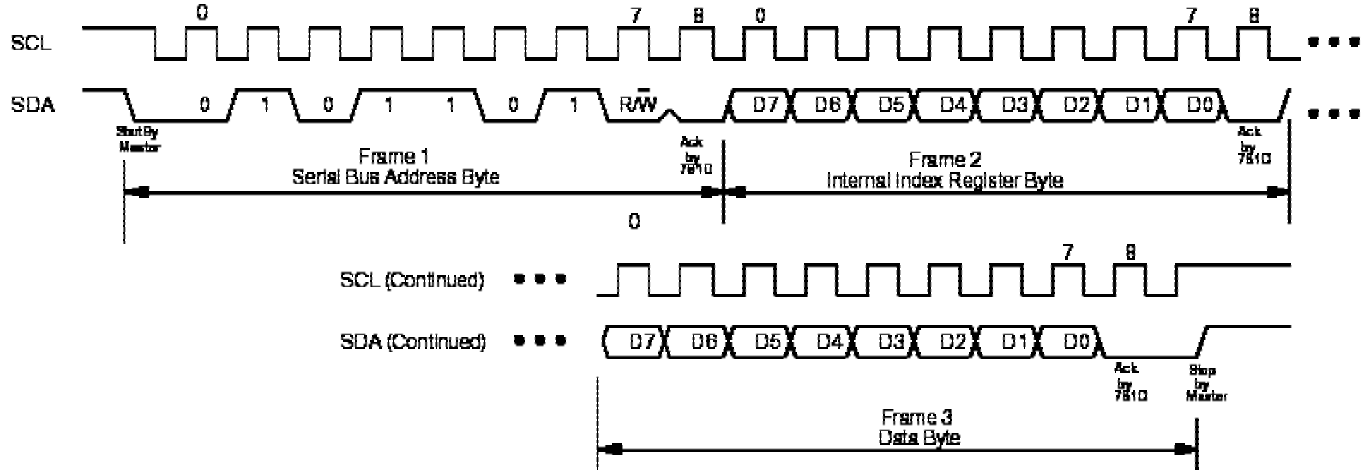


6.5 I^2C BUS Write

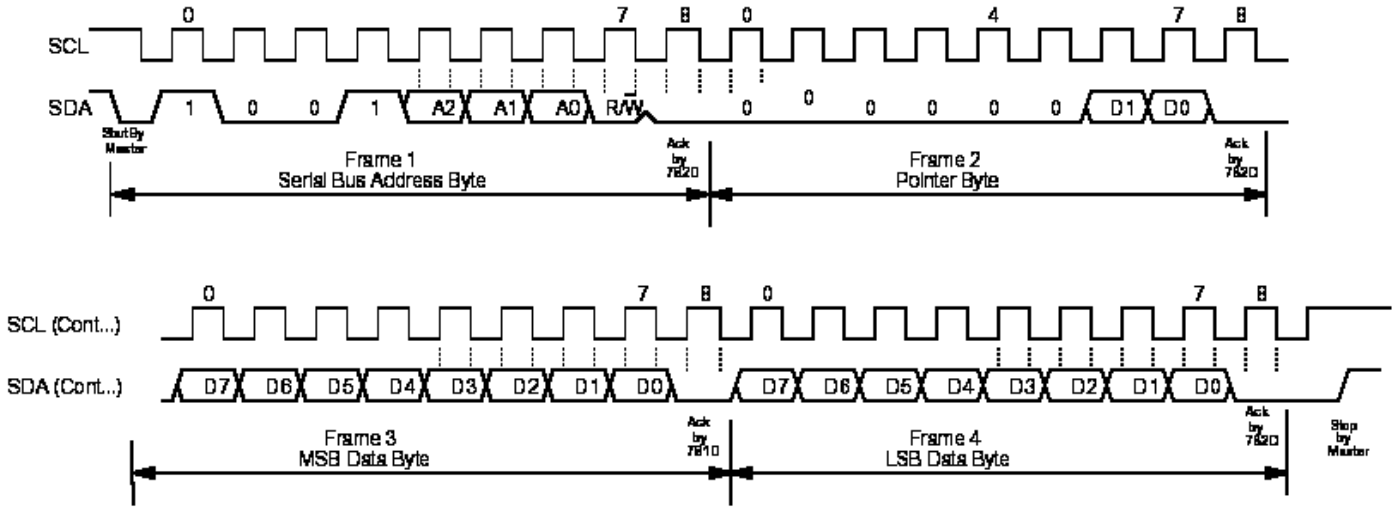
Write the index register:



Write data to the specified location:



Write two bytes to the specified location:



Note: Some registers use auto-increment, accessing the Data Register Port will automatically increment the Index Register Port. See the next chapter for more details.

CHAPTER 7: REGISTERS

7.1 Base registers

Register	Address	Auto-incremental Add.	Power on default
VCORE Reading	20h	60h	/
VI/O Reading	21h	61h	/
3V Reading	22h	62h	/
+5V Reading	23h	63h	/
+12V Reading	24h	64h	/
-12V Reading	25h	65h	/
VEXT Reading	26h	66h	/
On board Temperature sensor Reading	27h	67h	/
FAN1 Reading	28h	68h	/
FAN2 Reading	29h	69h	/
VCORE limit high	2Bh	6Bh	FFh
VCORE limit low	2Ch	6Ch	00h
VI/O limit high	2Dh	6Dh	FFh
VI/O limit low	2Eh	6Eh	00h
3V limit high	2Fh	6Fh	EDh
3V limit low	30h	70h	B0h
+5V limit high	31h	71h	01h
+5V limit low	32h	72h	0Ah
+12V limit high	33h	73h	00h
+12V limit low	34h	74h	0Ch
-12V limit high	35h	75h	04h
-12V limit low	36h	76h	10h
VEXT limit high	37h	77h	04h
VEXT limit low	38h	78h	44h
On board Temperature sensor limit low	39h	79h	00h
On board Temperature sensor hysteresis	3Ah	7Ah	00h
FAN1 count limit low	3Bh	7Bh	41h
FAN2 count limit low	3Ch	7Ch	20h
Configuration register	40h	/	01h
Interrupt Status register 1	/	41h	00h
Interrupt Status register 2	42h	/	00h
Interrupt Mask register 1	/	43h	00h
Interrupt Mask register 2	44h	/	00h
VID/FAN divisor register	47h	/	50h
Serial Bus Address Register	48h	/	2Dh
Voltage ID (VID4) and Device ID	49h	/	01h
TH1 and TH2 Serial Bus Address Register	4Ah	/	01h
IRQ#/OVT# property select	4Ch	/	01h
FAN, BEEP and GPO control register	4Dh	/	15h
Register 50h to 5Fh bank select	4Eh	/	80h
Winbond Vendor ID	4Fh	/	5Ch

Table 10: W83782D Base Registers

7.2 BANK 0

Register	Address	Auto-incremental Address	Power on default
BEEP control register 1	56h	/	00h
BEEP control register 2	57h	/	80h
Chip ID	58h	/	30h
Diode selection register	59h	/	70h
PWMOUT 2 control	5Ah	/	FFh
PWMOUT 1 control	5Bh	/	FFh
PWMOUT 1 and 2 clock select	5Ch	/	11h
VBAT monitor control register	5Dh	/	00h

Table 11: W83782D Bank 0 Registers

7.3 BANK 1

Register	Address	Auto-incremental Address	Power on default
Temperature sensor 2 high byte	50h	/	/
Temperature sensor 2 low byte	51h	/	/
Temperature sensor 2 configuration register	52h	/	00h
Temperature sensor 2 hysteresis high byte	53h	/	4Bh
Temperature sensor 2 hysteresis low byte	54h	/	00h
Temperature sensor 2 over-temperature high byte	55h	/	50h
Temperature sensor 2 over-temperature low byte	56h	/	00h

Table 12: W83782D Bank 1 Registers

7.4 BANK 2

Register	Address	Auto-incremental Address	Power on default
Temperature sensor 3 high byte	50h	/	/
Temperature sensor 3 low byte	51h	/	/
Temperature sensor 3 configuration register	52h	/	00h
Temperature sensor 3 hysteresis high byte	53h	/	4Bh
Temperature sensor 3 hysteresis low byte	54h	/	00h
Temperature sensor 3 over-temperature high byte	55h	/	50h
Temperature sensor 3 over-temperature low byte	56h	/	00h

Table 13: W83782D Bank 2 Registers

7.5 BANK 4

Register	Address	Auto-incremental Address	Power on default
Interrupt status register 3	50h	/	00h
Interrupt mask register 3	51h	/	00h
BEEP control register 3	53h	/	00h
Real Time Hardware status register 1	59h	/	00h
Real Time Hardware status register 2	5Ah	/	00h
Real Time Hardware status register 3	5Bh	/	00h
PWMOUT 3 & 4 clock select	5Ch	/	11h

Table 14: W83782D Bank 4 Registers

7.6 BANK 5

Register	Address	Auto-incremental Address	Power on default
5VSB Reading	50h	/	/
VBAT Reading	51h	/	/
5VSB limit high	54h	/	10h
5VSB limit low	55h	/	00h
VBAT limit high	56h	/	00h
VBAT limit low	57h	/	00h

Table 15: W83782D Bank 5 Registers

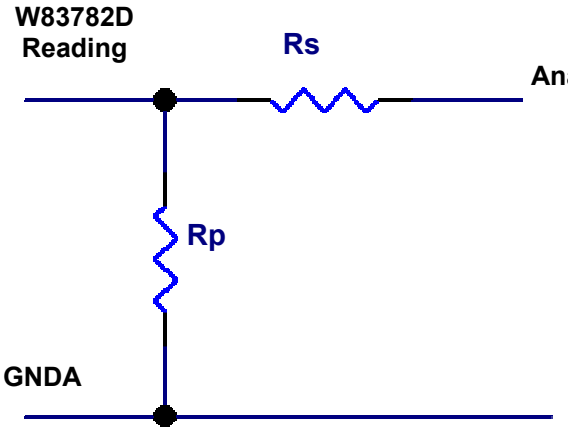
Note: BANK 3 is not used by the Winbond W83783D monitoring IC.

More detailed information about these registers is available in the W83782D Winbond H/W Monitoring IC datasheet.

CHAPTER 8: MEASUREMENT & CONTROL

8.1 Positive Analog Voltage Inputs

The positive analog voltage inputs reading reports the voltage measured on the input resistor divider and not the real input voltage. The resistor values of the input divider will have to be taken in account to calculate the real input voltage.



$$INPUT = \frac{READING \times 4.08 \times (Rs + Rp)}{255 \times Rp}$$

Analog Voltage inputs measurement			
Input	**Rp Value**	**Rs Value**	**Input calculation (V)**
+5V	50K	34K	READING x 4.08 x 84 / 255 / 50
+12V	10K	28K	READING x 4.08 x 38 / 255 / 10
VIO	10K	10K	READING x 4.08 x 20 / 255 / 10
VCORE	NL	10K	READING x 4.08 / 255
VEXT	7K5	232K	READING x 4.08 x 239.5 / 255 / 7.5
3V	NL	10K	READING x 4.08 / 255
VBAT	NL	1K	READING x 4.08 / 255
5VSB	7K5	5K1	READING x 4.08 x 12.6 / 255 / 7.5

Table 16: Positive Analog Voltage input measurement

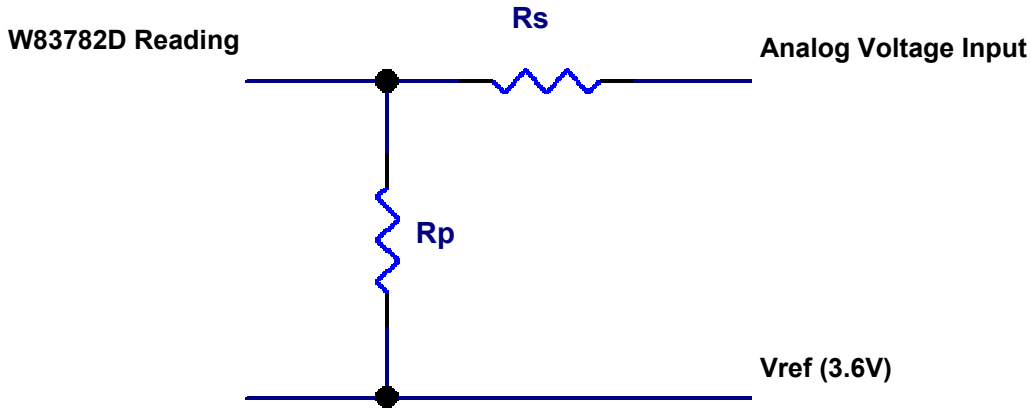
Note: NL indicate a resistor not populated on the board in the default configuration

Analog Voltage Reading Examples					
Input	**Reading**	**Rp Value**	**Rs Value**	**Input calculation (V)**	**Real input value**
3V	C7h	NL	10K	**199** x 4.08 / 255	3.18V
+5V	B8h	50K	34K	**184** x 4.08 x 84 / 255 / 50	4.95V
+12V	C6h	10K	28K	**198** x 4.08 x 38 / 255 / 10	12.04V

Table 17: Positive Analog Voltage Reading Example

8.2 Negative Analog Voltage Inputs

The negative analog voltage inputs reading reports the voltage measured on the input resistor divider referenced to 3.6V and not the real input voltage. The resistor values of the input divider will have to be taken in account to calculate the real input voltage.



$$INPUT = Vref - \left(Vref - \frac{READING \times 4.08}{255} \right) \left(\frac{Rs + Rp}{Rp} \right)$$

Analog Voltage inputs measurement			
Input	**Rp Value**	**Rs Value**	**Input calculation (V)**
-12V	56K	232K	3.6 - (3.6 - READING x 0.016) x (288 / 56)

Table 18: Negative Analog Voltage input measurement

Analog Voltage Reading Examples					
Input	**Reading**	**Rp Value**	**Rs Value**	**Input calculation (V)**	**Real input value**
-12V	23h	56K	232K	3.6 – (3.6 – **35** x 0.016) x (288 / 56)	-12.03V

Table 19: Negative Analog Voltage Reading Example

8.3 Temperature sensor inputs

The temperature date format is 8 bits two-complement for the internal sensor and 9 bits two-complement for both external sensors. The temperature reading register of the internal sensor reports the temperature in degree Celsius. The temperature reading registers of the external sensors report the temperature in ½ degree Celsius.

Temperature input readings		
Temperature	**Internal sensor**	**External sensors**
+125 °C	$0111.1101_B : 7D_H : 125_D$	$0.1111.1010_B : 0FA_H : 250_D$
+25 °C	$0001.1001_B : 19_H : 25_D$	$0.0011.0010_B : 032_H : 50_D$
+1 °C	$0000.0001_B : 01_H : 1_D$	$0.0000.0010_B : 002_H : 2_D$
+0.5 °C	/	$0.0000.0001_B : 001_H : 1_D$
0 °C	$0000.0000_B : 00_H : 0_D$	$0.0000.0000_B : 000_H : 0_D$
-0.5 °C	/	$1.1111.1111_B : 1FF_H : -1_D$
-1 °C	$1111.1111_B : FF_H : –1_D$	$1.1111.1110_B : 1FE_H : -2_D$
-25 °C	$1110.1110_B : E7_H : -25_D$	$1.1100.1110_B : 1CE_H : -50_D$
-55 °C	$1100.1001_B : C9_H : -55_D$	$1.1001.0010_B : 192_H : -110_D$

Table 20: Temperature input readings

8.4 FAN tachometer inputs

The HM-PCI104 allow fan speed measurement when the connected fan provides a tachometer output. The value recorded in the fan reading register will be invert proportional to the fan speed and the programmed divisor. The value FFh represents a reading out of range. Increasing the divider will allow to measure lower speed, and decreasing it will allow to measure higher speed. The Divider can be programmed from 1 to 128

$$READING = \frac{1{,}350{,}000}{RPM \times DIVIDER}$$

Tachometer input reading examples		
RPM	**Divider**	**Reading**
4440	2	98h
4440	16	13h

Table 21: Tachometer input reading examples

8.5 FAN speed control outputs

The HM-PCI104 can control the speed of two fan using two PWM outputs. The PWM output duty-cycle can be programmed by a 8 bits register. Modifying the duty-cycle allows the HM-PCI104 to drive the fans with a specific voltage. Both fans can be programmed from off condition to full speed.

$$Duty - Cycle(\%) = \frac{PROGRAMMED\_VALUE}{255} \times 100$$

PWM output examples		
Programmed value	**Output voltage**	**RPM**
00h	10mV	0
10h	4.5V	1300
20h	7.7V	2700
30h	9.3V	3400
40h	10V	3750
50h	10.6V	4000
60h	11V	4100
80h	11.3V	4220
A0h	11.6V	4330
FFh	11.9V	4380

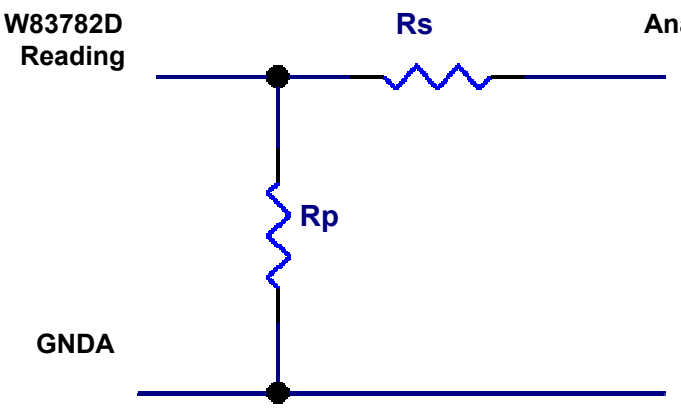
Table 22: PWM output examples

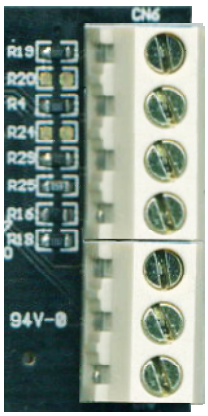
The measurements were performed using a 12V, 0.28A fan, part number F08G-12B2S1.

CHAPTER 9: CUSTOMISATION

9.1 Analog Voltage inputs

All analog voltage inputs of the Winbond monitor IC allow to measure a voltage from 0 to 4.096V with a precision of 0.016V. The HM-PCI104 provides a resistor divider to every input to enlarge the range of measurement. The divider can be modified to match a specific measurement need in range and precision.





Analog Voltage Maximum inputs							
Input	**Rp**	**Rs**	**Rp Value**	**Rs Value**	**Vmax calculation (V)**	**Vmax (V)**	**Precision (mV) (Vmax / 255)**
VCORE	R20	R19	NL	10K	4.08/ R20 * (R20 + R19)	4.08	16
VEXT	R18	R16	7K5	232K	4.08 / R18 * (R16 + R18)	130.8	511
3V	R23	R26	NL	10K	4.08 / R23 * (R26 + R23)	4.08	16
VIO	R22	R3	10K	10K	4.08 / R22 * (R22 + R3)	8.16	32
VBAT	R24	R4	NL	1K	4.08 / R24 * (R24 + R4)	4.08	16
5VSB	R25	R29	7K5	5K1	4.08 / R25 * (R25 + R29)	6.88	27

Table 23: Analog Voltage Maximum inputs

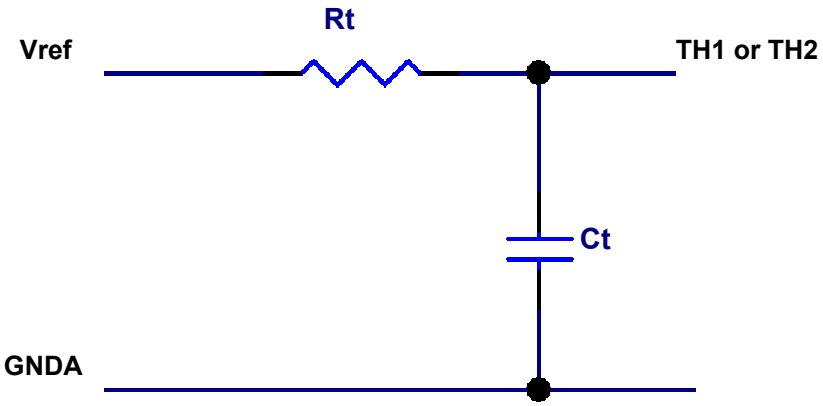
Note: NL indicate a resistor not populated on the board in the default configuration

Analog Voltage Example inputs				
Rp Value	**Rs Value**	**Vmax calculation (V)**	**Vmax (V)**	**Precision (mV) (Vmax / 256)**
10K	15K	4.08/ 10 * (10 + 15)	10	40
12K	47K	4.08 / 12 * (12 + 47)	20	79
12K	105K	4.08 / 12 * (12 + 105)	40	156

Table 24: Analog Voltage Example inputs

9.2 Temperature inputs

The HM-PCI104 provides two external connections for remote temperature sensors. These connections allows use of Thermistors (10K@25°C - ß=3435K) or 2N3904 NPN-type transistors or directly from Deschutes CPU thermal diode output. Each input has a circuit composed of a resistor and a capacitor that need to match the temperature sensor used. The HM104 is populated by default with the necessary components to use thermistor on both inputs.



Temperature inputs default					
Input	**Rt**	**Ct**	**Rt Value**	**Ct Value**	**Sensor**
TH1	R8	C6	10K	NL	Thermistors
TH2	R9	C7	10K	NL	Thermistors

Table 25: Temperature inputs default

Note: NL indicate a resistor not populated on the board in the default configuration

Temperature input components match		
Sensor	**Rt Value**	**Ct Value**
Thermistors (10K@25°C - ß=3435K)	10K	NL
2N3904 NPN-type transistors	30K	3.3nF
Deschutes CPU thermal diode output	30K	3.3nF

Table 26: Temperature inputs components match

CHAPTER 10: EXAMPLE PROGRAMS

10.1 Program 1 : read specified register

```
/*
br-i2c.c is a small program to access the registers of the HM-PCI104.
the program display the content of the specified HM-PCI104 register.

usage : br-i2c REGISTER_NUMBER

br-i2c.c can be compiled with the command : gcc br-i2c.c -o br-i2c

br-i2c.c was tested on a TMZ104 running Linux Slackware 7.0

*/

#include <stdio.h>
#include <unistd.h>
#include <asm/io.h>
#include <sys/types.h>

#define PCIADDRREF 0x0CF8 /* PCI Configuration ADDRESS Register */
#define PCIDATAREF 0x0CFC /* PCI Configuration DATA Register */


int main(int argc, char *argv[])
{
 unsigned long IOBase;
 unsigned long IReading;
 unsigned int SMBaddr;
 unsigned int unFunction;
 unsigned int Loop;
 unsigned char Status;
 unsigned char bIndex;
 unsigned char Buffer;

 if (argc != 2)
 {
        printf("\nbr-i2c need the register index as parameter\n\n");
   return -1;
 }

 sscanf(argv[1],"%i",&unFunction);
 bIndex = (unsigned char)(unFunction);
```

```
if ((bIndex < 0x20) || (bIndex > 0x7D))
{
        printf("\nIndex %02X is not valid \n\n",bIndex);
        return -2;
}

/* Get access to the ports */
if (iopl(3) < 0) {perror("iopl"); return(-1);}

/* Read the System Management Bus I/O Base */
IOBase = (unsigned long)(0x80000000 |
                         ((0x00 & 0xFF) << 16) |
                         ((0x07 & 0x1F) << 11) |
                         ((0x04 & 0x07) << 8)  |
                         (0x90 & 0xFC));
outl(IOBase, PCIADDRREF);
lReading = inl(PCIDATAREF);
SMBaddr = (unsigned int)(lReading & 0x0000FF00);
if (SMBaddr < 0x1000)
{
 printf("\n***** Error : '%08lX' invalid SMBus I/O Base address *****\n\n",SMBaddr);
 return -3;
}

/* Reset the System Management Bus */
Status = inb(SMBaddr + 0x00);
outb(Status,SMBaddr + 0x00);
if (inb(SMBaddr + 0x00) != 0x00)
{
 printf("\n***** Error : Reading, SMBus not ready *****\n\n");
 return -10;
}

/* Select the HM-PCI104 address in read mode */

outb(0x5F,SMBaddr + 0x04);

if (inb(SMBaddr + 0x04) != 0x5F)
{
 printf("\n***** Error : Setting the SMBus device address *****\n\n");
 return -8;
}
```

```
/* Set the index of the HM-PCI104 register*/

outb(bIndex,SMBaddr + 0x03);

if (inb(SMBaddr + 0x03) != bIndex)
{
 printf("\n***** Error : Setting the SMBus command register *****\n\n");
 return -9;
}

/* Read init data from the HM-PCI104 */

outb(0x48,SMBaddr + 0x02);

Loop = 10;
while ((inb(SMBaddr + 0x00) != 0x02) && (Loop-- > 0))
{
 usleep (100000);
}
if (inb(SMBaddr + 0x00) != 0x02)
{
 printf("\n***** Error : Reading data from the HM-PCI104 *****\n\n");
 return -11;
}

 Buffer = inb(SMBaddr + 0x05);

 printf("\nReading %02X from %02X\n\n",Buffer,bIndex);
 return 0;
}
```

10.2 Program 2 : write specified register

```
/*
bw-i2c.c is a small program to modify the registers of the HM104.
the program display the read back content of the specified HM104
register.

usage : bw-i2c REGISTER_NUMBER

bw-isa.c can be compiled with the command : gcc bw-i2c.c -o bw-i2c

bw-i2c.c was tested on a TMZ104 running Linux Slackware 7.0

*/

#include <stdio.h>
#include <unistd.h>
#include <asm/io.h>
#include <sys/types.h>

#define PCIADDRREF 0x0CF8 /* PCI Configuration ADDRESS Register */
#define PCIDATAREF 0x0CFC /* PCI Configuration DATA Register */


int main(int argc, char *argv[])
{
 unsigned long IOBase;
 unsigned long IReading;
 unsigned int SMBaddr;
 unsigned int unParameter;
 unsigned int Loop;
 unsigned char Status;
 unsigned char bIndex,bData;

 if (argc != 3)
 {
        printf("\nbw-i2c need the register index and data as parameters\n\n");
   return -1;
 }

 sscanf(argv[1],"%i",&unParameter);
 bIndex = (unsigned char)(unParameter);
 sscanf(argv[2],"%i",&unParameter);
 bData = (unsigned char)(unParameter);
```

```
if ((bIndex < 0x20) || (bIndex > 0x7D))
{
        printf("\nIndex %02X is not valid \n\n",bIndex);
        return -2;
}

/* Get access to the ports */
if (iopl(3) < 0) {perror("iopl"); return(-1);}

/* Read the System Management Bus I/O Base */
IOBase = (unsigned long)(0x80000000 |
                         ((0x00 & 0xFF) << 16) |
                         ((0x07 & 0x1F) << 11) |
                         ((0x04 & 0x07) << 8)  |
                         (0x90 & 0xFC));
outl(IOBase, PCIADDRREF);
lReading = inl(PCIDATAREF);
SMBaddr = (unsigned int)(lReading & 0x0000FF00);
if (SMBaddr < 0x1000)
{
 printf("\n***** Error : '%08lX' invalid SMBus I/O Base address *****\n\n",SMBaddr);
 return -3;
}

/* Reset the System Management Bus */
Status = inb(SMBaddr + 0x00);
outb(Status,SMBaddr + 0x00);
if (inb(SMBaddr + 0x00) != 0x00)
{
 printf("\n***** Error : Reading, SMBus not ready *****\n\n");
 return -10;
}

/* Select the HM-PCI104 address in write mode */

outb(0x5E,SMBaddr + 0x04);

if (inb(SMBaddr + 0x04) != 0x5E)
{
 printf("\n***** Error : Setting the SMBus device address *****\n\n");
 return -8;
}
```

```
/* Set the index of the HM-PCI104 register*/

outb(bIndex,SMBaddr + 0x03);

if (inb(SMBaddr + 0x03) != bIndex)
{
 printf("\n***** Error : Setting the SMBus command register *****\n\n");
 return -9;
}

/* Set the data of the HM-PCI104 register*/

outb(bData,SMBaddr + 0x05);

if (inb(SMBaddr + 0x05) != bData)
{
 printf("\n***** Error : Setting the SMBus data register *****\n\n");
 return -9;
}

/* Read init data from the HM-PCI104 */

outb(0x48,SMBaddr + 0x02);

Loop = 10;
while ((inb(SMBaddr + 0x00) != 0x02) && (Loop-- > 0))
{
 usleep (100000);
}
if (inb(SMBaddr + 0x00) != 0x02)
{
 printf("\n***** Error : Reading data from the HM-PCI104 *****\n\n");
 return -11;
}


printf("\nWriting %02X to %02X\n\n",bData,bIndex);
return 0;
}
```

10.3 Program 3 : monitoring example

```
/*
mon-i2c.c is a small program to access the memory of the HM104.
the program display the content of the HM104 memory.

usage : mon-i2c

mon-i2c.c can be compiled with the command : gcc mon-i2c.c -o mon-i2c

mon-i2c.c was tested on a TMZ104 running Linux Slackware 7.0
*/

#include <stdio.h>
#include <stdlib.h>
#include <sys/io.h>

#define PCIADDRREF 0x0CF8 /* PCI Configuration ADDRESS Register */
#define PCIDATAREF 0x0CFC /* PCI Configuration DATA Register */

int SmbInit(unsigned int *unSmbAddr);
int ReadData(unsigned int unSmbAddr, unsigned char bIndexValue, unsigned char *bDataValue);

int main(int argc, char *argv[])
{

 unsigned int SMBaddr;
 unsigned char bInByte,bOutByte,bIndex;
 unsigned char bMem[30];
 unsigned char bSMI1,bSMI2,bSMImask1,bSMImask2;
 unsigned int unFan1Div,unFan2Div;
 float fValue,fLimitH,fLimitL;

 printf("\n");

 // Parameter test
 if (argc != 1)
 {
  printf ("Command 'mon-i2c' doesn´t need any parameter\n\n");
  return -1;
 }

 if (SmbInit(&SMBaddr) < 0) // Retrieve the SMBus base address
  return -2;

for  (bIndex = 0;bIndex < 30;bIndex++)
{
        if (ReadData(SMBaddr, 0x20 + bIndex, bMem + bIndex) < 0) // read the data present in the memory
    return -3;
}
```

```
/* VCORE display */
fValue = (float)(bMem[0]) * 0.016;
fLimitH = (float)(bMem[11]) * 0.016;
fLimitL = (float)(bMem[12]) * 0.016;
printf("Vcore = %5.3f\t - Limit high = %5.3f\t - Limit low = %5.3f\n",fValue,fLimitH,fLimitL);

/* VI/O display */
fValue = (float)(bMem[1]) * 0.016 * (10 + 10)  / 10;
fLimitH = (float)(bMem[13]) * 0.016 * (10 + 10)  / 10;
fLimitL = (float)(bMem[14]) * 0.016 * (10 + 10)  / 10;
printf("VI/O  = %5.3f\t - Limit high = %5.3f\t - Limit low = %5.3f\n",fValue,fLimitH,fLimitL);

/* 3.3V display */
fValue = (float)(bMem[2]) * 0.016;
fLimitH = (float)(bMem[15]) * 0.016;
fLimitL = (float)(bMem[16]) * 0.016;
printf("3.3V  = %5.3f\t - Limit high = %5.3f\t - Limit low = %5.3f\n",fValue,fLimitH,fLimitL);

/* +5V display */
fValue = (float)(bMem[3]) * 0.016 * (34 + 50)  / 50;
fLimitH = (float)(bMem[17]) * 0.016 * (34 + 50)  / 50;
fLimitL = (float)(bMem[18]) * 0.016 * (34 + 50)  / 50;
printf("+5V   = %5.3f\t - Limit high = %5.3f\t - Limit low = %5.3f\n",fValue,fLimitH,fLimitL);

/* +12V display */
fValue = (float)(bMem[4]) * 0.016 * (28 + 10)  / 10;
fLimitH = (float)(bMem[19]) * 0.016 * (28 + 10)  / 10;
fLimitL = (float)(bMem[20]) * 0.016 * (28 + 10)  / 10;
printf("+12V  = %5.3f\t - Limit high = %5.3f\t - Limit low = %5.3f\n",fValue,fLimitH,fLimitL);

/* -12V display */
fValue = - ((3.6 - (float)(bMem[5]) * 0.016) / 56 *  (56 + 232)  - 3.6);
fLimitH = - ((3.6 - (float)(bMem[21]) * 0.016) / 56 *  (56 + 232)  - 3.6);
fLimitL = - ((3.6 - (float)(bMem[22]) * 0.016) / 56 *  (56 + 232)  - 3.6);
printf("-12V  = %5.3f\t - Limit high = %5.3f\t - Limit low = %5.3f\n",fValue,fLimitH,fLimitL);

/* VEXT display */
fValue = (float)(bMem[6]) * 0.016 * (232 + 7.5)  / 7.5;
fLimitH = (float)(bMem[23]) * 0.016 * (232 + 7.5)  / 7.5;
fLimitL = (float)(bMem[24]) * 0.016 * (232 + 7.5)  / 7.5;
printf("VEXT  = %5.3f\t - Limit high = %5.3f\t - Limit low = %5.3f\n",fValue,fLimitH,fLimitL);

/* Temp1 display */
fValue = (float)(bMem[7]);
fLimitH = (float)(bMem[25]);
fLimitL = (float)(bMem[26]);
printf("Temp  = %5.1f\t - Limit high = %5.1f\t - Hysteresis= %5.1f\n",fValue,fLimitH,fLimitL);
```

```
if (ReadData(SMBaddr, 0x47, &bInByte) < 0) // read the FAN Divider low register
   return -4;
```

```
unFan1Div = (bInByte & 0x30) >> 4;
unFan2Div = (bInByte & 0xC0) >> 6;

if (ReadData(SMBaddr, 0x5D, &bInByte) < 0) // read the FAN Divider high register
   return -5;

unFan1Div = unFan1Div + ((bInByte & 0x20) >> 3);
unFan2Div = unFan2Div + ((bInByte & 0x40) >> 4);
unFan1Div = 1 << unFan1Div;
unFan2Div = 1 << unFan2Div;

/* FAN1 display */
fValue = 1350000 / (float)(bMem[8]) / (float)(unFan1Div);
fLimitL = 1350000 / (float)(bMem[27]) / (float)(unFan1Div);
if (bMem[8] == 0xFF)
 printf("FAN1  = ???????\t ",fValue);
else
 printf("FAN1  = %5.0f\t ",fValue);
printf("- Limit Low  = %5.0f\t ",fLimitL);
printf("- Divider = %d \n",unFan1Div);

/* FAN2 display */
fValue = 1350000 / (float)(bMem[9]) / (float)(unFan2Div);
fLimitL = 1350000 / (float)(bMem[28]) / (float)(unFan2Div);
if (bMem[9] == 0xFF)
 printf("FAN2  = ???????\t ",fValue);
else
 printf("FAN2  = %5.0f\t ",fValue);
printf("- Limit Low  = %5.0f\t ",fLimitL);
printf("- Divider = %d \n",unFan2Div);

if (ReadData(SMBaddr, 0x41, &bSMI1) < 0) // read the SMI# status register
   return -6;

if (ReadData(SMBaddr, 0x42, &bSMI2) < 0) // read the SMI# status register 2
   return -7;

if (ReadData(SMBaddr, 0x43, &bSMImask1) < 0) // read the SMI# mask register 1 index
   return -8;

if (ReadData(SMBaddr, 0x44, &bSMImask2) < 0) // read the SMI# mask register 2 index
   return -9;

printf("\nInterrupts  = %02X%02X - mask  = %02X%02X \n\n",bSMI2,bSMI1,bSMImask2,bSMImask1);

 return 0;
}
```

```
int SmbInit(unsigned int *unSmbAddr)
{
 unsigned long IOBase;
 unsigned long IReading;

 /* Get access to the ports */
 if (iopl(3) < 0) {perror("iopl"); return(-1);}

 /* Read the System Management Bus I/O Base */
 IOBase = (unsigned long)(0x80000000 |
                          ((0x00 & 0xFF) << 16) |
                          ((0x07 & 0x1F) << 11) |
                          ((0x04 & 0x07) << 8)  |
                          (0x90 & 0xFC));
 outl(IOBase, PCIADDRREF);
 IReading = inl(PCIDATAREF);
 *unSmbAddr = (unsigned int)(IReading & 0x0000FF00);
 if (*unSmbAddr < 0x1000)
 {
  printf("\n***** Error : '%08lX' invalid SMBus I/O Base address *****\n\n",*unSmbAddr);
  return -2;
 }
 else
   return 0;
}
```

```
int ReadData(unsigned int unSmbAddr, unsigned char bIndexValue, unsigned char *bDataValue)
{
  unsigned int Loop;
  unsigned char Status;

 /* Reset the System Management Bus */
 Status = inb(unSmbAddr + 0x00);
 outb(Status,unSmbAddr + 0x00);
 if (inb(unSmbAddr + 0x00) != 0x00)
 {
  printf("\n***** Error : Reading, SMBus not ready *****\n\n");
  return -1;
 }

 /* Set the index of the HM-PCI104 register*/
 outb(bIndexValue,unSmbAddr + 0x03);
 if (inb(unSmbAddr + 0x03) != bIndexValue)
 {
  printf("\n***** Error : Setting the SMBus command register *****\n\n");
  return -3;
 }

 /* Select the HM-PCI104 address in read mode */
 outb(0x5F,unSmbAddr + 0x04);
 if (inb(unSmbAddr + 0x04) != 0x5F)
 {
  printf("\n***** Error : Setting the SMBus device address *****\n\n");
  return -2;
 }

 /* Read init data from the HM-PCI104 */
 outb(0x48,unSmbAddr + 0x02);
 Loop = 10;
 while ((inb(unSmbAddr + 0x00) != 0x02) && (Loop-- > 0))
 {
  usleep (100000);
 }
 if (inb(unSmbAddr + 0x00) != 0x02)
 {
  printf("\n***** Error : Reading data from the HM-PCI104 *****\n\n");
  return -3;
 }

  *bDataValue = inb(unSmbAddr + 0x05);
  return 0;
}
```

CHAPTER 11: LITERATURE REFERENCES

The following references are for information about the PC/104 architecture, the PC DOS, and the PC BIOS.

11.1 ISA System Architecture

MindShare, Inc., Tom Shanley and Don Anderson
Internet: mindshar@interserv.com
CompuServe: 72507,1054
Published by Addison Wesley, Inc.

11.2 AT Bus Design

Edward Solari
Anabooks
12145 Alta Carmel Ct., Suite 250
San Diego, CA 92128
ISBN 0-929392-08-6

11.3 Personal Computer Bus Standard P996

Institute of Electrical and Electronic Engineers, Inc.
445 Hoes Lane
Piscataway, NJ 08854

11.4 PC Interrupts

PC Interrupts, Ralf Brown, Addison/Wesley.

11.5 PC/104 Consortium

809 B-175 Cuesta Drive,
Mountain View, CA 94040
Phone: 415 903-8304
FAX: 415 967-0995

11.6 W83782D Data Sheet

Winbond
4, Creation Rd,
Hsinchu, 300, Taiwan R.O.C.
Phone: 886-3-577-0066

11.7 V82C686B "Super South" South Bridge Data Sheet

VIA Technologies, Inc
940 Mission Court
Fremont, CA 94539
Phone: 1 510 683 3300
FAX: 1 510 687 4654