Application note

Number	12
Date	24.06.96
Author	H. Bruhn
Subject	I²C bus on **Kontron** boards
Related Products	all **Kontron** modules with I²C bus feature

Date	Related Products	Added by
24.06.96	SuperMOPS/486DX-1, ModulAT/486DX-2, PC/104-MIO2, AT96/ISA96/ISA-Multi-4, ISA-VGALCD-4, PC/104-VGALCD-4	H.Bruhn
08.04.98	added littleMONSTER/586 (Rev. 1.10 and later), added littleMONSTER/P (Rev. 1.21 and later)	C. Riesinger
09.10.98	added DIMM-PC/386-B (Rev. 2.1 and later) added DIMM-PC/386-I (Rev. 2.0 and later) added MOPSplus/lite (Rev. 2.1 and later) Corrected IO-adress of littleMONSTER/P from 50h to 51h	M. Hofmeister
17.11.98	added MOPSlcd4	C. Riesinger
16.03.99	added Table of Contents, JAP revised	R. Barth
21.05.99	added DIMM-PC/486-I	C. Riesinger
24.09.99	added CM, LM2P and MOPSlcd6	C. Riesinger
12.10.00	added coolMonster/S, littleMONSTER/P3, ETX-MGX and ETX-P1	C. Riesinger
16.10.00	added DIMM-PC/386-I	C. Hoch
12.07.01	added DIMM-PC/386-B new hardware revision (beginning with CE ?30) and ETX-P3/C3	C. Hoch
04.10.01	added DIMM-PC/520-I, MOPS/520 and modified description for DIMM-PC/486-I, modified example in chapter 3.2.	C. Hoch
05.02.02	Modified source code example in chapter 3.2, added index	C. Hoch
26.03.02	Added MOPSlcdGX1, corrected MOPSlcd6/686+ connector	G. Vogl
24.04.02	English proofreading	D. Gunter
23.08.02	Changed to Kontron style	H. Bruhn

1. TABLE OF CONTENTS

2. INDEX

ETX	DIMM-PC	SLOT	PC/104
ETX-P3E/C3E (MOD7)	DIMM-PC/586-IE (D502)	coolMONSTER/P3/C3 (LEU6)	MOPSlcd6, MOPS686+ (P588)
ETX-P3/C3 (MOD6)	DIMM-PC/520-I (D501)	coolMONSTER/S (LEU3)	MOPSlcdGX1 (PGX1)
ETX-P1 (MOD5)	DIMM-PC/486-I (D401)	littleMONSTER2 (LEU2)	MOPSlcd4, MOPS586 (P488)
ETX-mgx (MOD1)	DIMM-PC/386-IE (D201)	littleMONSTER (LEU1)	MOPS520 (P489)
	DIMM-PC/386-B (D101)	littleMONSTER+ (LEV1)	MOPSMZ, MOPSlcdMZ (PMZ1)
		littleMONSTER/586 (PISB)	MOPS/386A (P389)

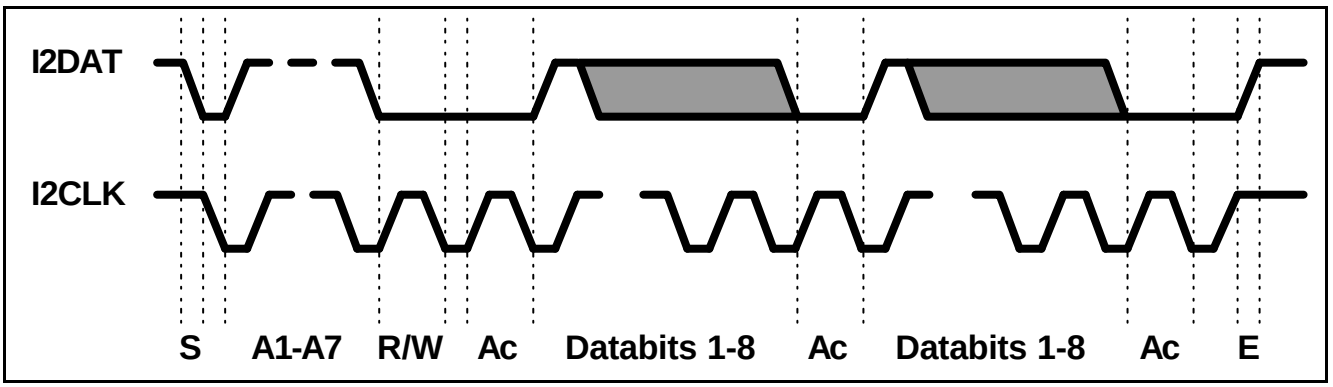
Old Products

3. GENERAL INFORMATION ABOUT I²C-BUS

3.1. Introduction to I²C-bus

The Inter-IC bus (I²C) is a two-wired serial bus and provides a sort of small area network between the circuits of one system and between different systems. Any device with built-in I²C bus interface can be connected to the system by simply clipping it to the I²C bus. It consists of two bi-directional lines for serial data (I2DAT) and serial clock (I2CLK). Every device connected can be master or slave, so there is no central master. A device addressed as a slave during one data transfer could possibly be the master for the next data transfer. Devices are also free to transmit or receive data during a transfer. The inherent synchronization process in connection with the wired AND technique allows fast devices to communicate with slower ones.

For each data bit transferred one clock pulse has to be generated. The data on the I2DAT line must be stable during the high period of the clock. The data lines state can only change when the I2CLK line is low. Data transfer is entered by a start condition and ended by a stop condition. A high to low transition of the I2DAT line, while the I2CLK is high, signals the start condition and a low to high transition, while I2CLK is high, indicates the stop-condition. Data transfer follows the format below:



After the start condition (S) the slave address byte is sent. This byte consists of seven address bits (A1-A7) and one direction bit (R/W) with low level indicating a transmission (WRITE) and high level indicating a request for data (READ).

After the addressing of a slave device the master's next clock pulse is used for acknowledgement (Ac). During this acknowledge pulse the I2DAT line has to be pulled down to low by the receiving device. A data transfer is always terminated by a stop condition (E) generated by the master. However, if the master wants to communicate with another device on the bus it generates another start condition to address another slave without the necessity of first generating a stop condition.

This was only a short summary concerning the I²C bus. For detailed information (e.g. timing problems, characteristics of devices) refer to I²C bus specifications, data books and specialized textbooks.

3.2. I²C bus on Kontron boards

The I²C bus interface on **Kontron** boards has to be implemented by the customer via software, which drives the two lines I2DAT and I2CLK, following the I²C bus specifications. The basic hardware to design the software interface is standard on the devices mentioned in this application note.

Note: This kind of interface does not support external masters.

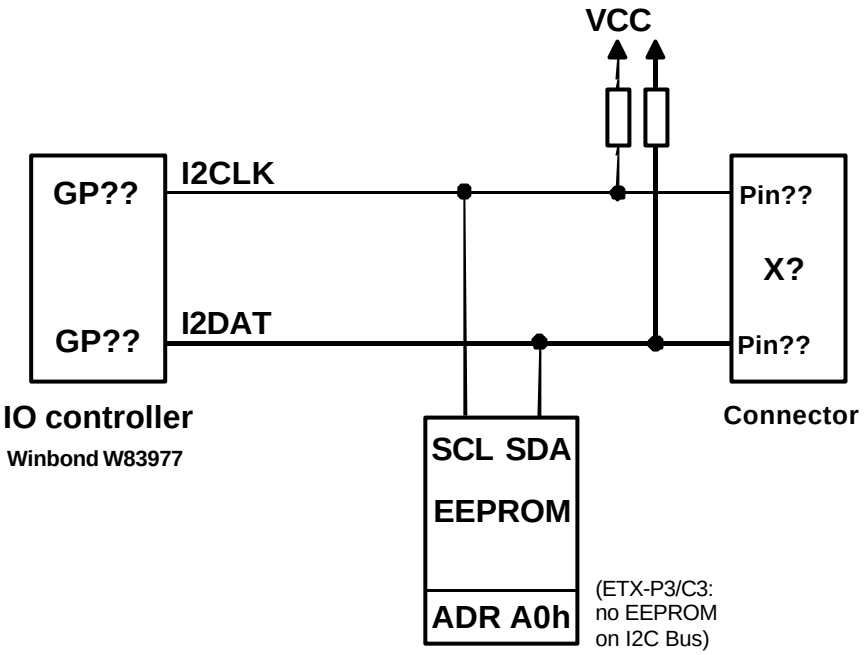
On different **Kontron** boards the two I²C bus lines are not offered on identical connectors. Refer to your manual if you're not sure you're using the right connector or pins for your I²C application.

The following schematics show the bus interface and the onboard devices connected to the I²C bus on the **Kontron** boards.

INDEX

4. ACCESS TO I²C-BUS

4.1. ETX-P3/C3, ETX-mgx, DIMM-PC/520-I, DIMM-PC/486-I, coolMONSTER/P3, MOPSlcdGX1



- I²C-bus Addresses

Device address of EEPROM: 1010 000xb
Reserved address: 1011 000xb
0101 100xb

- Programming Information

Two General Purpose IO pins of the Winbond W83977 I/O controller control the I²C Bus signals. If the IOs are set to be inputs, I2CLK and I2DAT are high because of the pull-ups. To drive I2CLK and I2DAT low, one must set GP's to output and set the respective bit in a register mapped to I/O port 100h (or 102h). The programming example below shows exactly how the I²C Bus signals can be controlled.

	GP I/O (Winbond 83977)				Connector	Pin Nr.	
	GP Port	Base Address	I2CLK (SCL)	I2DAT (SDA)		I2CLK	I2DAT
ETX-P3/C3	2 (device 8)	102h	GP20 (bit0)	GP21 (bit1)	ETX-bus: X4	Pin16	Pin22
ETX-mgx	1 (device 7)	100h	GP14 (bit4)	GP15 (bit5)	ETX-bus: X4	Pin16	Pin22
DIMM-PC/520-I	1 (device 7)	100h	GP14 (bit4)	GP15 (bit5)	DIMM-bus: X1	Pin74	Pin72
DIMM-PC/486-I	1 (device 7)	100h	GP13 (bit3)	GP12 (bit2)	DIMM-bus: X1	Pin74	Pin72
CM/P3	1 (device 7)	100h	GP14 (bit4)	GP15 (bit5)	X6	Pin10	Pin11
MOPSlcdGX1	1 (device 7)	100h	GP14 (bit4)	GP15 (bit5)	Feature X8	Pin3	Pin4

Attention: **These devices are for BIOS-access only; reading from or writing to them may cause data corruption and system failure.**

Example:

- **These routines are used to drive the I²C bus lines SCL (clock) and SDA (data):**

```
W83977GP1Base     EQU   100h              ; General Purpose Port 1 base address
; W83977GP2Base   EQU   102h              ; General Purpose Port 2 base address (ETX-P3/C3)
I2CCLK_MASK       EQU   010h              ; GP14 (bit4 of GP1)used for I2C bus clock (SCL)
; I2CCLK_MASK     EQU   001h              ; GP20 (bit0 of GP2)used for I2C bus clock (ETX-P3/C3)
; I2CCLK_MASK     EQU   004h              ; GP13 (bit3 of GP1)used for I2C bus clock (DIMM-PC/486-I)
I2CDAT_MASK       EQU   020h              ; GP15 (bit5 of GP1)used for I2C bus data (SDA)
; I2CDAT_MASK     EQU   002h              ; GP21 (bit1 of GP2)used for I2C bus data (ETX-P3/C3)
; I2CDAT_MASK     EQU   008h              ; GP12 (bit2 of GP1)used for I2C bus data (DIMM-PC/486-I)

I2CDAT_BIT        EQU   5

I2CLK low:      call    i2cSetSclToOutput
                mov     dx,W83977GP1Base        ; W83977GP2Base (ETX-P3/C3)
                in      al, dx
                and     al, NOT I2CCLK_MASK
                out     dx, al

I2CLK high:     call    i2cSetSclToInput

I2CDAT low:     call    i2cSe tSdaToOutput
                mov     dx,W83977GP1Base        ; W83977GP2Base (ETX-P3/C3)
                in      al, dx
                and     al, NOT I2CDAT_MASK
                out     dx, al

I2DAT high:     call    i2cSetSdaToInput

ReadI2DAT:      mov     dx, W83977GP1Base       ; W83977GP2Base (ETX-P3/C3)
                in      al, dx
                and     al, I2CDAT_MASK
                shr     al, I2CDAT_BIT
```

- **These routines are used to set the data direction of the I²C bus lines SCL (clock) and SDA (data):**

```
;-----------------------------------------------------------------------------------------------
; Name: i2cSetSclToInput
; Desc: This function sets direction of GPxx (SCL)to input.
; Inp:  none
; Outp: none
; Regs: none
;-----------------------------------------------------------------------------------------------
i2cSetSclToInput  PROC NEAR PRIVATE
                pushf
                cli                                     ; disable interrupts
                mov     ax, 0707h                       ; select dev7 (0807h for dev8 – ETX-P3/C3)
                call    sioWb977RegWrite
                mov     al, 0E4h                        ; GP14 control
                ; mov   al, 0E8h                       ; GP20 control (ETX-P3/C3)
                ; mov   al, 0E3h                       ; GP13 control (DIMM-PC/486-I)
                call    sioWb977RegRead
                or      ah, 01h                         ; set to input
                call    sioWb977RegWrite
                popf
                ret
i2cSetSclToInput  ENDP
```

... Continued

```
;--------------------------------------------------------------------------------------------------
; Name: i2cSetSclToOutput
; Desc: This function sets direction of GPxx (SCL)to output.
; Inp:  none
; Outp: none
; Regs: none
;--------------------------------------------------------------------------------------------------
i2cSetSclToOutput PROC NEAR PRIVATE
                pushf
                cli                                     ; disable interrupts
                mov     ax, 0707h                       ; select dev7 (0807h for dev8 – ETX-P3/C3)
                call    sioWb977RegWrite
                mov     al, 0E4h                        ; GP14 control
                ; mov   al, 0E8h                        ; GP20 control (ETX-P3/C3)
                ; mov   al, 0E3h                        ; GP13 control (DIMM-PC/486-I)
                call    sioWb977RegRead
                and     ah, NOT 01h                     ; set to output
                call    sioWb977RegWrite
                popf
                ret
i2cSetSclToOutput ENDP

;--------------------------------------------------------------------------------------------------
; Name: i2cSetSdaToInput
; Desc: This function sets direction of GPxx (SDA) to input.
; Inp:  none
; Outp: none
; Regs: none
;--------------------------------------------------------------------------------------------------
i2cSetSdaToInput  PROC NEAR PRIVATE
                pushf
                cli                                     ; disable interrupts
                mov     ax, 0707h                       ; select dev7 (0807h for dev8 – ETX-P3)
                call    sioWb977RegWrite
                mov     al, 0E5h                        ; GP15 control
                ; mov   al, 0E9h                        ; GP21 control (ETX-P3/C3)
                ; mov   al, 0E2h                        ; GP12 control (DIMM-PC/486-I)
                call    sioWb977RegRead
                or      ah, 01h                         ; set to input
                call    sioWb977RegWrite
                popf
                ret
i2cSetSdaToInput  ENDP

;--------------------------------------------------------------------------------------------------
; Name: i2cSetSdaToInput
; Desc: This function sets direction of GPxx (SDA) to output.
; Inp:  none
; Outp: none
; Regs: none
;--------------------------------------------------------------------------------------------------
i2cSetSdaToOutput PROC NEAR PRIVATE
                pushf
                cli                                     ; disable interrupts
                mov     ax, 0707h                       ; select dev7 (0807h dev8 – ETX-P3/C3)
                call    sioWb977RegWrite
                mov     al, 0E5h                        ; GP15 control
                ; mov   al, 0E9h                        ; GP21 control (ETX-P3/C3)
                ; mov   al, 0E2h                        ; GP12 control (DIMM-PC/486-I)
                call    sioWb977RegRead
                and     ah, NOT 01h                     ; set to output
                call    sioWb977RegWrite
                popf
                ret
i2cSetSdaToOutput ENDP
```

The programming above requires low-level access to the internal registers of the Winbond W83977 I/O controller

The reader and writer routines `sioWb977RegRead` and `sioWb977RegWrite` access these registers.

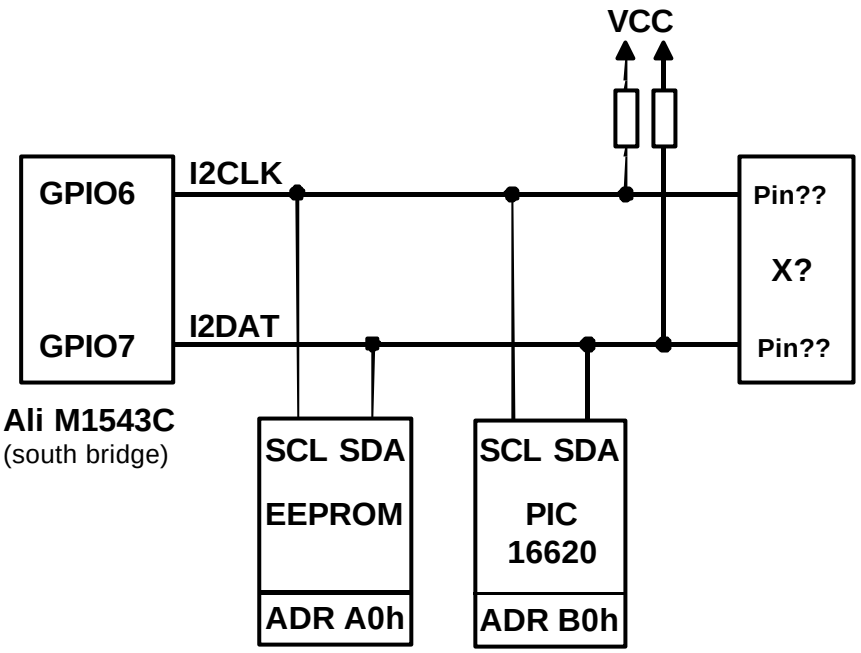
- **W83977 reader and writer routines:**

```
;-------------------------------------------------------------------------------------------------------
; Name: sioWb977RegRead
; Desc: Read data from Winbond83977 register.
; Inp:  AL – register index
; Outp: AH - register value;
; Regs: none
;-------------------------------------------------------------------------------------------------------
sioWb977RegRead PROC NEAR PUBLIC
                push    dx
                mov     dx, 3F0h                    ; configuration base address
                push    ax
                mov     al, 87h                     ; Enter configuration mode
                out     dx, al
                out     dx, al
                pop     ax
                out     dx, al                      ; write register index
                inc     dx                          ; Point to the data register
                xchg    ah, al                      ; Move index into AH
                in      al, dx                      ; Read the data
                xchg    ah, al                      ; AL = Index, AH = Data
                push    ax
                dec     dx                          ; Point to the index register
                mov     al, 0AAh                    ; Exit configuration mode
                out     dx, al
                pop     ax
                pop     dx
             ret
sioWb977RegRead ENDP

;-------------------------------------------------------------------------------------------------------
; Name: sioWb977RegWrite
; Desc: Write data to Winbond83977 register.
; Inp:  AL – register index
; Outp: AH - register value;
; Regs: none
;-------------------------------------------------------------------------------------------------------
sioWb977RegWrite PROC NEAR PUBLIC
                push    dx
                mov     dx, 3F0h                    ; configuration base address
                push    ax
                mov     al, 87h                     ; Enter configuration mode
                out     dx, al
                out     dx, al
                pop     ax
                out     dx, al                      ; write register index
                inc     dx                          ; Point to the data register
                xchg    ah, al                      ; Move index into AH
                out     dx, al                      ; Write the data
                xchg    ah, al                      ; Restore AX to original condition
                push    ax
                dec     dx                          ; Point to the index register
                mov     al, 0AAh                    ; Exit configuration mode
                out     dx, al
                pop     ax
                pop     dx
             ret
sioWb977RegRead ENDP
```

NOTE: **DO NOT MODIFY ANY OTHER BIT AND REGISTER AS DESCRIBED HERE! THIS COULD LEAD TO INCORRECT SYSTEM BEHAVIOUR.**

INDEX

4.2. ETX-P1, coolMONSTER and littleMONSTER2, coolMONSTER/S, MOPSlcd6 and MOPS/686+



- I²C Addresses

Device address of EEPROM : 1010 000xb
Device address of PIC16620 : 1011 000xb
Reserved address: : 0101 100xb

- Programming Information

The I²C Bus signals are controlled by two General Purpose IOs of the PMU (power management unit PMU M7101) device in the south bridge M1543C (ISA bridge). See source code example below how to enable PMU.
If the IOs are set to be inputs, I2CLK and I2DAT are high because of the pull-ups. To drive I2CLK and I2DAT low one must set GP6/7 to output and set the respective bit in a register of the PMU to 0. The programming example below shows exactly how the I²C Bus signals can be controlled.

	ISA bridge (PMU device M7101)			Connector	Pin Nr.	
	Bus	Device	Function		I2CLK	I2DAT
ETX-P1	00h	07h	00h	ETX-Bus: X4	Pin16	Pin22
MOPSlcd6 / MOPS/686+	00h	02h	00h	Not accessible		
coolMONSTER, littleMONSTER2, coolMONSTER/S	00h	07h	00h	X6	Pin10	Pin11

Attention: **These devices are for BIOS-access only; reading from or writing to them may cause data corruption and system failure.**

- **These routines are used to drive the I2C bus lines SCL (clock) and SDA (data):**

```
CONFIG_ADDR             EQU   0CF8h             ; configuration address register
CONFIG_DATA             EQU   0CFCh             ; configuration data register
CONFIG_ADDR_PMU         EQU   8000385Ch         ; PCI config cycle to bus 00h, device 07h,
                                                ; function 00h, register index 5Ch (make vizible/
                                                ; hide PMU register – bit2 in reg 5Fh)
CONFIG_ADDR_SET         EQU   8000387Ch         ; PCI config cycle to bus 00h, device 07h, function 00h,
                                                ; register index 7Ch (PMU respectively ISA Bridge)
I2CCLK_MASK             EQU   00404000h         ; set GPIO6 to output and high (SCL)
I2CDAT_MASK             EQU   00808000h         ; set GPIO7 to output and high (SDA)
I2CDAT_BIT              EQU   01Fh

;------------------------- Make PMU register vizible. ---------------------------------

      mov eax, CONFIG_ADDR_PMU                  ; make PMU device vizible
      call PCI_RegRead                          ; returns contain of CONFIG_DATA in EBX
      and ebx, NOT 04000000h                    ; set bit 2 in register 5Fh to 0 (0 = PMU enabled)

      call PCI_RegWrite                         ; write back CONFIG_DATA
      (delay 150us)                             ; give 150us delay
;-------------------------------------------------------------------------------------

I2CLK_low:     mov eax, CONFIG_ADDR_SET
               call PCI_RegRead                 ; returns contain of CONFIG_DATA in EBX
               or ebx, I2CCLK_MASK              ; set GPIO6 to output and low
               call PCI_RegWrite

I2CLK_high:    mov eax, CONFIG_ADDR_SET
               call PCI_RegRead                 ; returns contain of CONFIG_DATA in EBX
               and ebx, NOT I2CCLK_MASK         ; set GPIO6 to input (high)
               call PCI_RegWrite

I2DAT_low:     mov eax, CONFIG_ADDR_SET
               call PCI_RegRead                 ; returns contain of CONFIG_DATA in EBX
               or ebx, I2CDAT_MASK              ; set GPIO7 to output and low
               call PCI_RegWrite

I2DAT_high:    mov eax, CONFIG_ADDR_SET
               call PCI_RegRead                 ; returns contain of CONFIG_DATA in EBX
               and ebx, NOT I2CDAT_MASK         ; set GPIO7 to input (high)
               call PCI_RegWrite

Read_I2DAT:    mov eax, CONFIG_ADDR_SET
               call PCI_RegRead                 ; returns contain of CONFIG_DATA in EBX
               and ebx, NOT I2CDAT_MASK         ; set GPIO7 to input
               call PCI_RegWrite
               call PCI_RegRead                 ; read I2DAT
               shr ebx, I2CDAT_BIT              ; I2DAT is now BL[0]

;---------------------------------- Hide PMU register -----------------------------------

      mov eax, CONFIG_ADDR_PMU                  ; hide PMU device
      call PCI_RegRead                          ; returns contain of CONFIG_DATA in EBX
      or ebx, 04000000h                         ; set bit 2 in register 5Fh to 1 (1 = PMU disabled)

      call PCI_RegWrite                         ; write back CONFIG_DATA
      (delay 150us)                             ; give 150us delay
;-------------------------------------------------------------------------------------
```

The programming of this GPIO requires low-level access to the internal registers of the on chip PCI PMU device (M7101) respectively ISA bridge device (M1533). ***PCI configuration cycles mechanism #1*** via port CF8h (CONFIG_ADDRESS) and CFCh (CONFIG_DATA), are required to modify the internal PMU registers. See literature for more information on PCI configuration cycles.

- **Accessing a PCI function's configuration port is a four step process:**
  - Write the target bus number, physical device number, function number and doubleword number to the configuration address port (CF8h). **This must be a 32 bit (doubleWord) access!**
  - Perform an I/O read from the configuration data port (CFCh)
  - Modify the respective bits of the configuration data port (CFCh)
  - Perform an I/O write to the configuration data port (CFCh)

- **Configuration Address Register at CF8h**

31	30 ... 24	23 ... 16	15 ... 11	10 ... 8	7 ... 2	1	0
ECD	Reserved	Bus Number	Device Number	Function	Register Index (Doubleword)	00	

Bit [1:0] - are reserved and must be zero (because ofdoubleword register index access)
Bit [7:2] - identify the target doubleword within the target function's configuration space
Bit [10:8] - identify the target function number within the target physical PCI device
Bit [15:11] - identify the target physical PCI device number
Bit [23:16] - identify the target PCI bus number
Bit [30:24] - are reserved and must be zero
Bit [31] - enable CONFIG_DATA
= 0, CONFIG_DATA register not active
= 1, CONFIG_DATA register active

See literature for more information on PCI configuration cycles.

- **Register Index (PMU device respectively ISA bridge)**

Index 7Dh
Bit6; Bit7: Direction Control of GPIO[6;7]
= 0, GPIO[6;7] is a General purpose input pin
= 1, GPIO[6;7] is a General purpose output pin

Index 7Eh
Bit6; Bit7: Data Output to GPIO[6;7] when is set as General purpose output pin

Index 7Fh
Bit6; Bit7: Data Input from GPIO[6;7] when is set as General purpose input pin

- **Example: read the CONFIG_DATA register**

```
;-------------------------------------------------------------------------------------------------
; Name:     PCI_RegRead - read CONFIG_DATA register (32bit)
; Entry:    EAX  - PCI configuration cycle
; Exit:     EBX - data for CONFIG_DATA register
; Modified: EBX
;-------------------------------------------------------------------------------------------------

PCI_RegRead PROC NEAR PUBLIC
            push dx
            mov dx, CONFIG_ADDR               ; configuration address register
            out dx, eax                       ; write CONFIG_ADDRESS port
            jcxz $+2
            mov ebx, eax                      ; save EAX to EBX

            mov dx, CONFIG_DATA               ; configuration data register
            in eax, dx                        ; read CONFIG_DATA port
            jcxz $+2
            xchg eax, ebx                     ; EBX now holds CONFIG_DATA dword
            pop dx
      ret
PCI_RegRead ENDP
```

- **Example: write the CONFIG_DATA register**

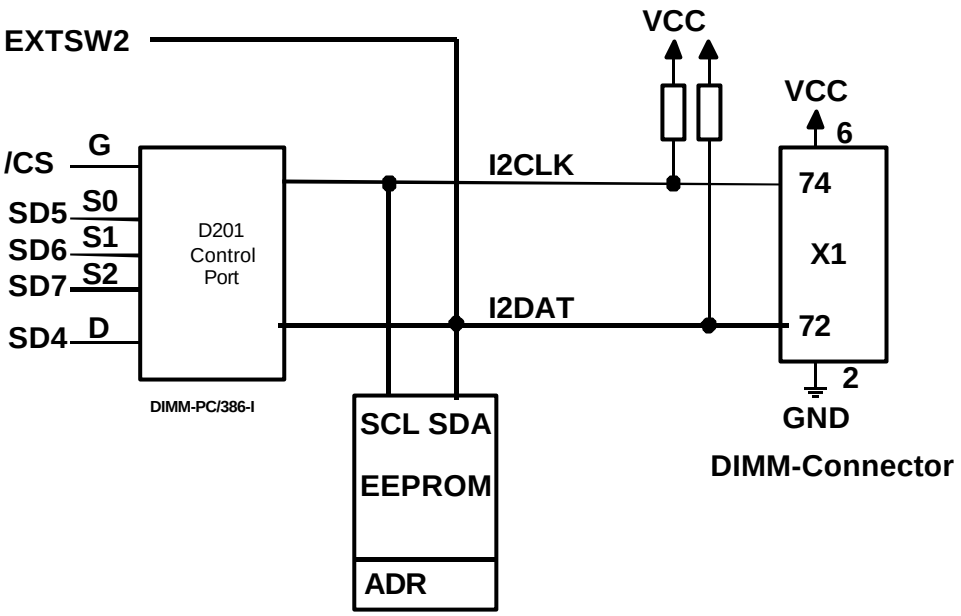
```
;--------------------------------------------------------------------------------------------------------
; Name:     PCI_RegWrite - write CONFIG_DATA register (32bit)
; Entry:    EAX - PCI configuration cycle
;           EBX - data for CONFIG_DATA register
; Exit:     none
; Modified: EBX, DX
;--------------------------------------------------------------------------------------------------------

PCI_RegWrite PROC NEAR PUBLIC
            push dx
            mov dx, CONFIG_ADDR               ; configuration address register
            out dx, eax                       ; write CONFIG_ADDRESS port
            jcxz $+2
            xchg eax, ebx                     ; exchange EAX and EBX

            mov dx, CONFIG_DATA               ; configuration data register
            out dx, eax                       ; write EAX to CONFIG_DATA port
            jcxz $+2
            xchg eax, ebx                     ; exchange EAX and EBX
            pop dx
      ret
PCI_RegWrite ENDP
```

NOTE: **If one wants to write board independent software, it is good programming practice to search the ISA bridge device instead of using fix devices. The vendor and device ID of the ISA bridge are: 10B9h/1533h**

NOTE: **DO NOT MODIFY ANY OTHER BIT AND REGISTER AS DESCRIBED HERE! THIS COULD LEAD TO INCORRECT SYSTEM BEHAVIOUR.**

INDEX

4.3. DIMM-PC/386-IE



Device address of EEPROM : 1010 000xb
Reserved address: : 1011 000xb
0101 100xb

Attention: **These devices are for BIOS-access only; reading from or writing to them may cause data corruption and system failure.**

The handling of I^2C-bus must be done by direct programming of the Ali M6117C registers.

- **Unlock the configuration registers of the Ali M6117C:**

```
out   22h, 13h              ; enable 13h
out   23h, C5h              ; unlock registers
```

- **Read/write the configuration registers:**

  On board I/O port 22h is the index register and I/O port 23h is the data register. To read a configuration register, write the index value to I/O port 22h in advance, and then read data from I/O port 23h. To write a configuration register, write the index value to I/O port 22h, and then write data to I/O port 23h.
  For instance, if we want to write the data (example 55h) of configuration register which index is 10h, the steps are:

```
out 22h, 10h                ; write 10h (index) to I/O port 22h
out 23h, 55h                ; write data 55h to I/O port 23h
```

- **The following procedure shows how to access the D201 control port:**

```
;--------------------------------------------------------------------------
; Name: WriteControlPort
; Desc: This routine writes to the control port.
; Inp:  AH = value to be written to the control port
; Outp: none
; Regs: all registers preserved
;--------------------------------------------------------------------------

WriteControlPort        PROC NEAR PRIVATE

            pushf
            cli
            call    open_config
            mov     al, 068h            ; Prepare xbus.
            call    write_22_23         ; write data to config. register
            mov     al, 073h            ; Activate ENPOWER to latch
            call    write_22_23         ; xbus.
            call    close_config
            popf
            ret

WriteControlPort        ENDP
```

- **The following procedures show how to set the I²C bus lines of the D201:**

```
;--------------------------------------------------------------------------
; Name: I2cSclLow, I2cSclHigh, I2cSdaLow, I2cSdaHigh
; Desc: These routines are used to drive the I2C bus lines SCL (clock) and
;       SDA (data).
; Inp:  CH = combined I2C device address, page address and data direction
; Outp: none
; Regs: may change AX, DX and FLAGS
;--------------------------------------------------------------------------

D201_I2DAT_LOW          equ     ((6 shl 5) + (0 shl 4))
D201_I2DAT_HIGH         equ     ((6 shl 5) + (1 shl 4))
D201_I2CLK_LOW          equ     ((7 shl 5) + (0 shl 4))
D201_I2CLK_HIGH         equ     ((7 shl 5) + (1 shl 4))

_J_I2cSclLow    PROC NEAR PUBLIC
            mov     ah, D201_I2CLK_LOW
            jmp     WriteControlPort
_J_I2cSclLow    ENDP

_J_I2cSclHigh   PROC NEAR PUBLIC
            mov     ah, D201_I2CLK_HIGH
            jmp     WriteControlPort
_J_I2cSclHigh   ENDP

_J_I2cSdaLow    PROC NEAR PUBLIC
            mov     ah, D201_I2DAT_LOW
            jmp     WriteControlPort
_J_I2cSdaLow    ENDP

_J_I2cSdaHigh   PROC NEAR PUBLIC
            mov     ah, D201_I2DAT_HIGH
            jmp     WriteControlPort

_J_I2cSdaHigh   ENDP
```

- **The following procedure shows how to read the EXTSW2 signal on the D201:**

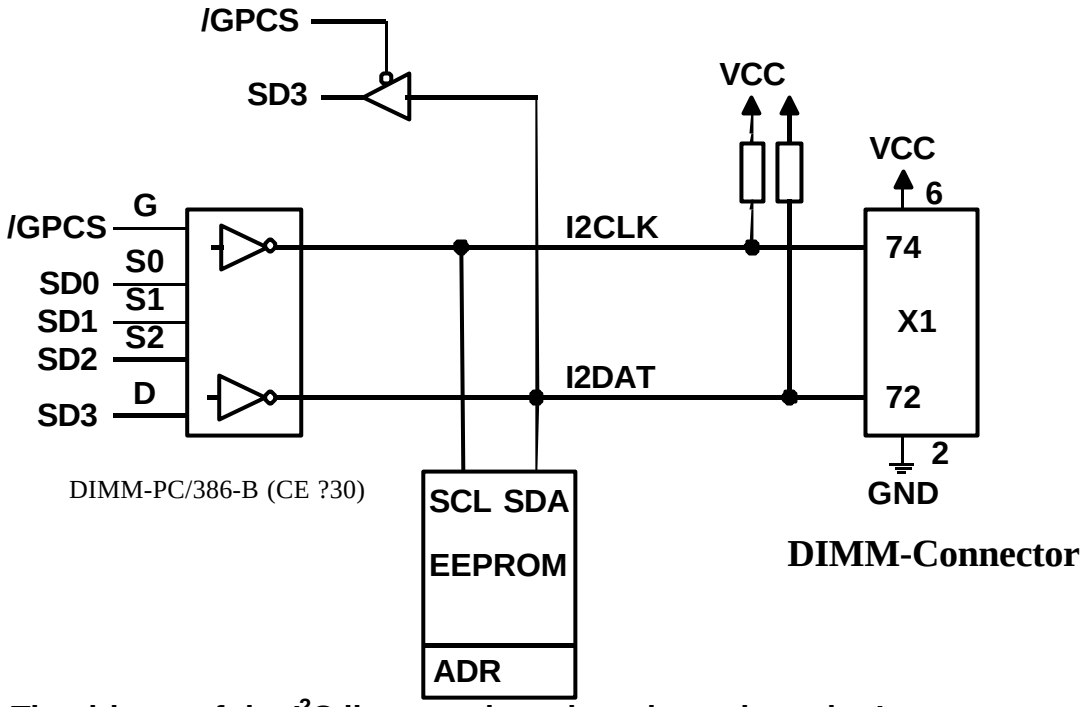
```
;--------------------------------------------------------------------------
; Name: I2cReadSda
; Desc: This routine reads the current status of the I2C data line (SDA).
; Inp:  CH = combined I2C device address, page address and data direction
; Outp: AL = 0 = SDA forced to GND, i.e. logical state is 0.
;          = 1 = SDA released which is a logical 1
; Regs: AL changed
;--------------------------------------------------------------------------

_J_I2cReadSda   PROC NEAR PUBLIC

            pushf
            cli
            call    open_config
            mov     al, 067h
            call    read_22_23          ; read data from config. register
            and     al, 004h            ; SDA is on EXTSW2.
            shr     al, 2
            push    ax
            call    close_config
            pop     ax
            popf
            ret

_J_I2cReadSda   ENDP
```

NOTE: **DO NOT MODIFY ANY OTHER BIT AND REGISTER AS DESCRIBED HERE! THIS COULD LEAD TO INCORRECT SYSTEM BEHAVIOUR.**

INDEX

4.4. DIMM-PC/386-B

4.4.1. New Hardware Revision (beginning from CE ?30)



Attention:The drivers of the I²C lines on these boards are inverting!

I/O address to generate /CS	:	51h
Device address of EEPROM	:	1010 000xb
Reserved address:	:	1011 000xb
		0101 100xb

Changes to the old Hardware Revision (see 3.10.2):

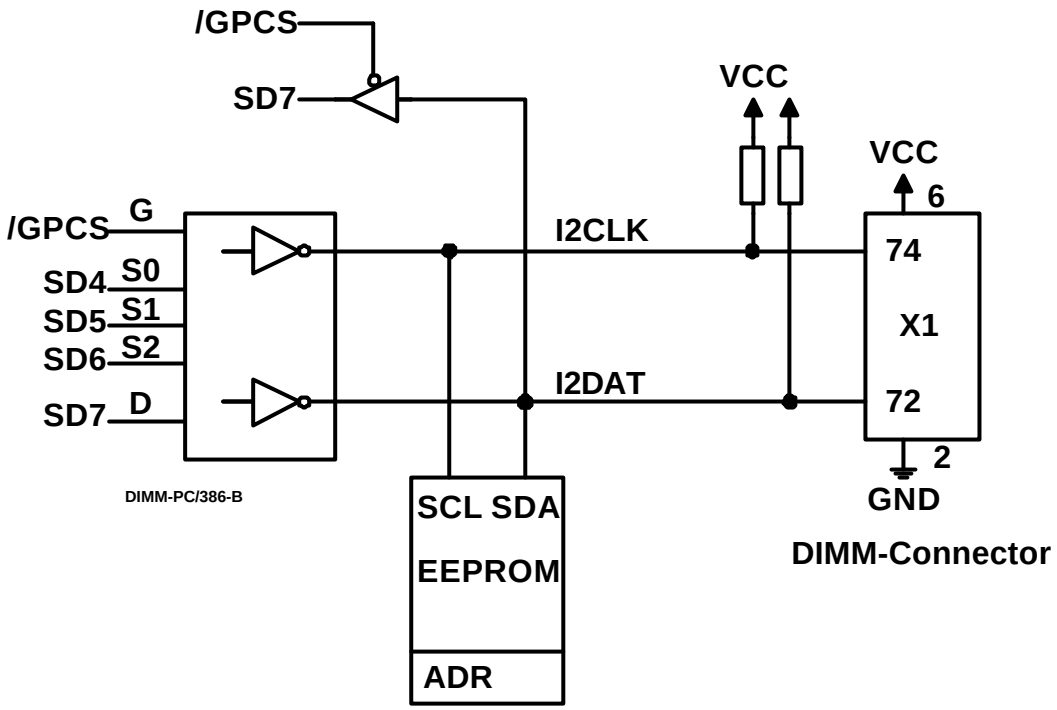
Data line is now SD3 (former SD7) and the addressing of I2DAT and I2CLK is now controlled by SD2-SD0 (former SD6-SD4)

Attention: **These devices are for BIOS-access only; reading from or writing to them may cause data corruption and system failure.**

Example: Assume you're writing a software utility and want the I2DAT line to be pulled HIGH. You have to use I/O-port 51h and set the Data line SD3=0 (gets HIGH because of inversion) and SD2-SD1-SD0 = 000 to address I2DAT.
The corresponding OUT instruction would be: **OUT 51h, x0h**
to reset I2DAT, bit SD3 must be set (because of inversion).
The corresponding OUT instruction would be: **OUT 51h, x8h**
Writing x1h and x9h to port 51h affects I2CLK the same way!
When reading form port 51h, SD3 holds the level of I2DAT.

Several OUTs and INs will be necessary to address one I²C bus device and write or read a byte, because the start and stop conditions and every single bit have to be set separately as well as the I2CLK line.

4.4.2. Old Hardware Revision (until CE ?23)



Attention:The drivers of the I²C lines on these boards are inverting!

I/O address to generate /CS	:	51h
Device address of EEPROM	:	1010 000xb
Reserved address:	:	1011 000xb
		0101 100xb

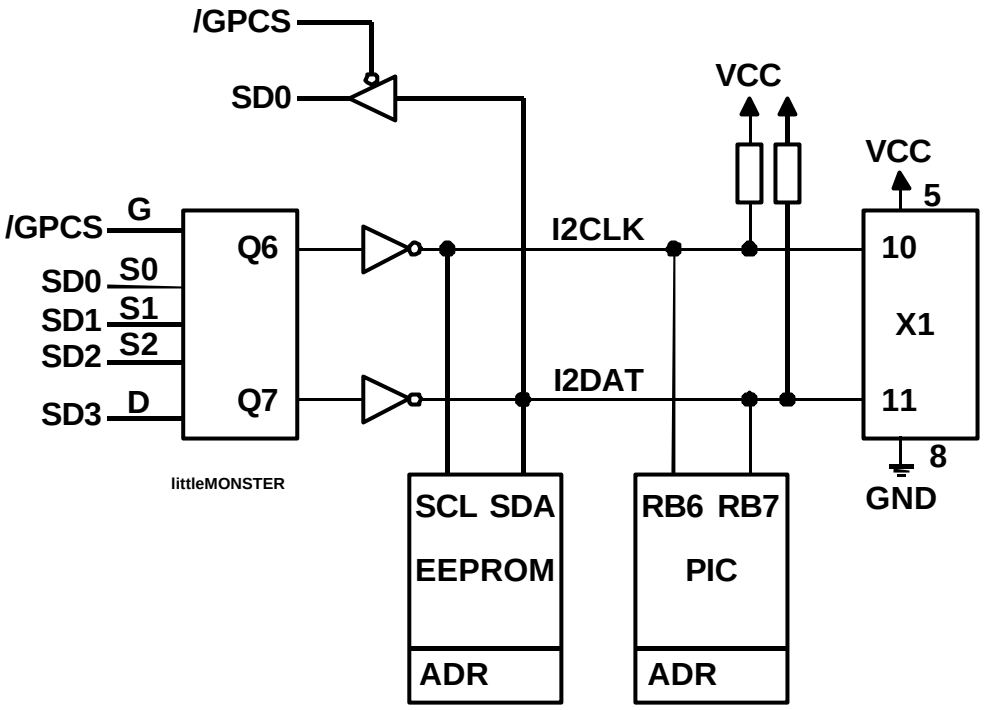
Attention: **These devices are for BIOS-access only; reading from or writing to them may cause data corruption and system failure.**

Example: Assume you're writing a software utility and want the I2DAT line to be pulled HIGH. You have to use I/O-port 51h and set the Data line SD7=0 (gets HIGH because of inversion) and SD6-SD5-SD4 = 000 to address I2DAT.
The corresponding OUT instruction would be: **OUT 51h, 0xh**
To reset I2DAT bit SD7 must be set (because of inversion).
The corresponding OUT instruction would be: **OUT 51h, 8xh**
Writing 1xh and 9xh to port 51h affects I2CLK the same way!
When reading form port 51h, SD7 holds the level of I2DAT.

Several OUTs and INs will be necessary to address one I²C bus device and write or read a byte because the start and stop conditions and every single bit have to be set separately as well as the I2CLK line.

INDEX

4.5. littleMONSTER (LEU1 / LEV1)



Attention: **The drivers of the I²C lines on these boards are inverting!**

I/O address to generate /CS : 51h

Device address of EEPROM : 1010 000xb
Device address of PIC16F84 : 1011 000xb
Device address of MAX517 : 0101 100xb

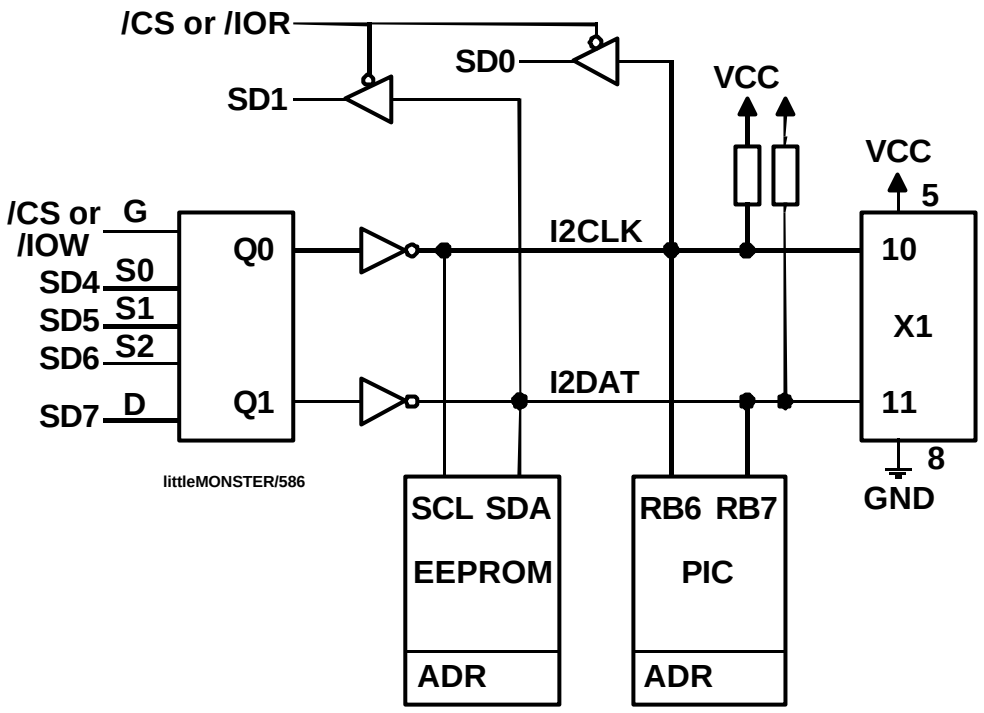
Attention: **These devices are for BIOS-access only; reading from or writing to them may cause data corruption and system failure.**

Example: Assume you're writing a software utility and want the I2DAT line to be pulled HIGH. You have to use I/O-port 51h and set the Data line D3=0 (gets HIGH because of inversion) and D2-D1-D0 = 111 to address I2DAT.
The corresponding OUT instruction would be: **OUT 51h, 07h**
To reset I2DAT bit SD3 must be set (because of inversion).
The corresponding OUT instruction would be: **OUT 51h, 0Fh**
Writing 06h and 0Eh to port 51h affects I2CLK the same way!
When reading from port 51h, SD0 holds the level of I2DAT.

Several OUTs and INs will be necessary to address one I²C bus device and write or read a byte, because the start and stop conditions and every single bit have to be set separately as well as the I2CLK line.

INDEX

4.6. littleMONSTER/586 (PISB)



Attention: **The drivers of the I^2C lines on these boards are inverting!**

I/O address to generate /CS : 100h

Device address of EEPROM : 1010 000xb
Device address of PIC16F84 : 1011 000xb
Device address of MAX517 : 0101 100xb

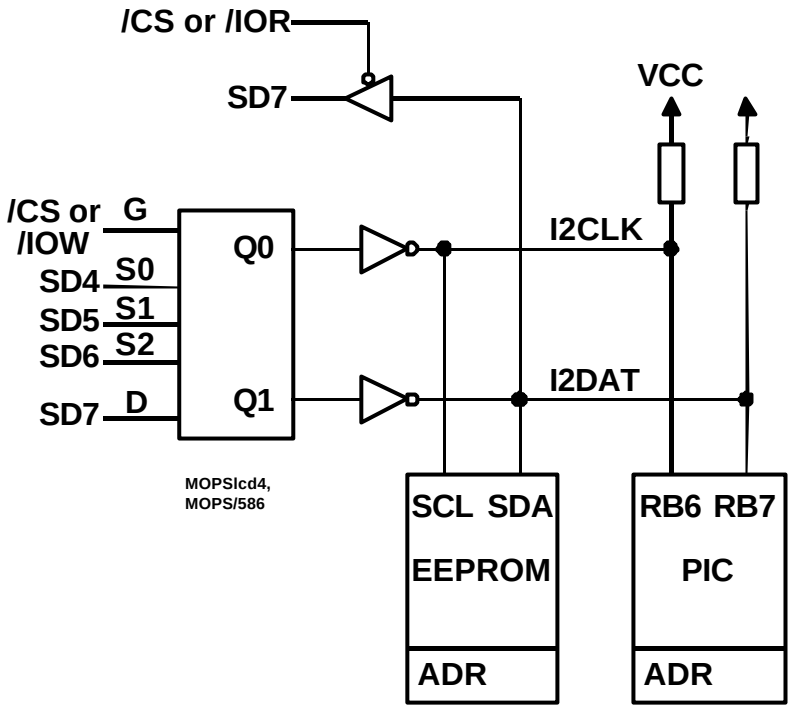
Attention: **These devices are for BIOS-access only; reading from or writing to them may cause data corruption and system failure.**

Example: Assume you're writing a software utility and want the I2DAT line to be pulled HIGH. You have to use I/O-port 100h and set the Data line D7=0 (gets HIGH because of inversion) and D6-D5-D4 = 001 to address I2DAT.
The corresponding OUT instruction would be: **OUT 100h, 10h**
To reset I2DAT bit SD7 must be set (because of inversion).
The corresponding OUT instruction would be: **OUT 100h, 90h**
Writing 00h and 80h to port 100h affects I2CLK the same way!
When reading from port 100h, SD0 holds the level of I2CLK and SD1 the level of I2DAT.

Several OUTs and INs will be necessary to address one I^2C bus device and write or read a byte, because the start and stop conditions and every single bit have to be set separately as well as the I2CLK line.

INDEX

4.7. MOPSlcd4 / MOPS/586 (P488)



I/O address to generate /CS : 101h

Device address of EEPROM : 1010 000xb
Device address of PIC16F84 : 1011 000xb
Reserved address: : 0101 100xb

Attention: **These devices are for BIOS-access only; reading from or writing to them may cause data corruption and system failure.**

Example: Assume you're writing a software utility and want the I2DAT line to be pulled HIGH. You have to use I/O-port 101h and set the Data line D7=1 and D6-D5-D4 = 001 to address I2DAT.
The corresponding OUT instruction would be: **OUT 101h, 90h**
To reset I2DAT bit SD7 must be reset.
The corresponding OUT instruction would be: **OUT 101h, 10h**
Writing 80h and 00h to port 101h affects I2CLK the same way!
When reading from port 101h, SD7 holds the level of I2DAT.

Several OUTs and INs will be necessary to address one I^2C bus device and write or read a byte, because the start and stop conditions and every single bit have to be set separately as well as the I2CLK line.

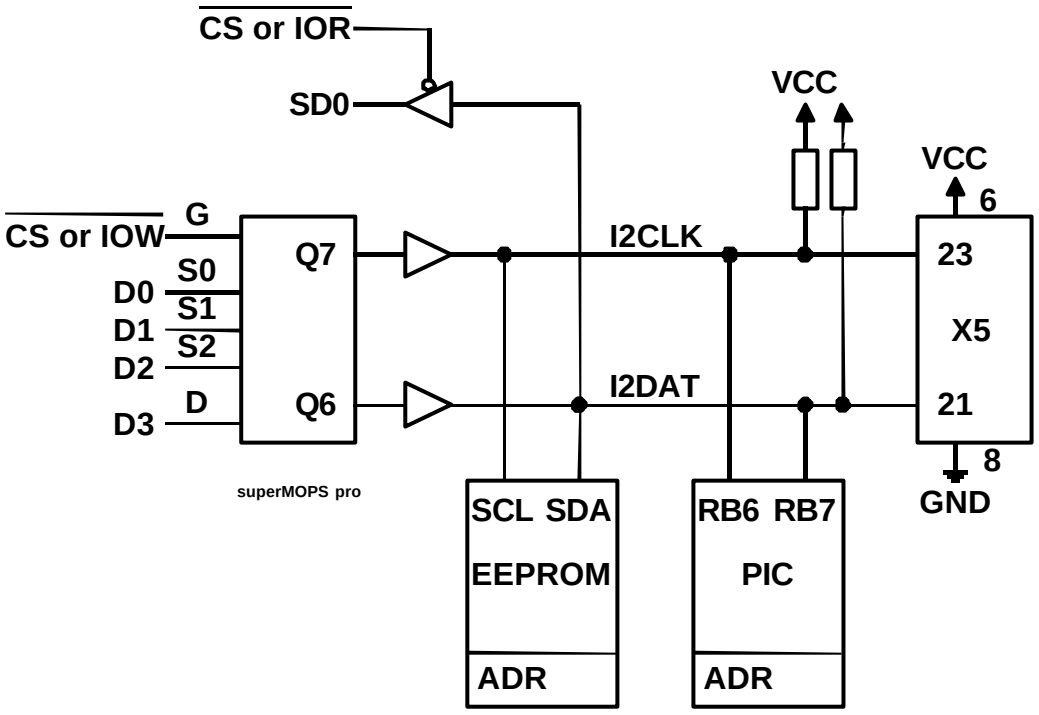
INDEX

4.8. MOPSplus / MOPSlite (P386), MOPS/520 (P489)

Attention: **I²C-bus on MOPSplus/lite and MOPS/520 is not supported for external applications.**

INDEX

4.9. superMOPSpro (P487)



I/O address to generate /CS : 50h

Device address of EEPROM : 1010 000xb
Device address of PIC16C84 : 1011 000xb
Reserved address: : 0101 100xb

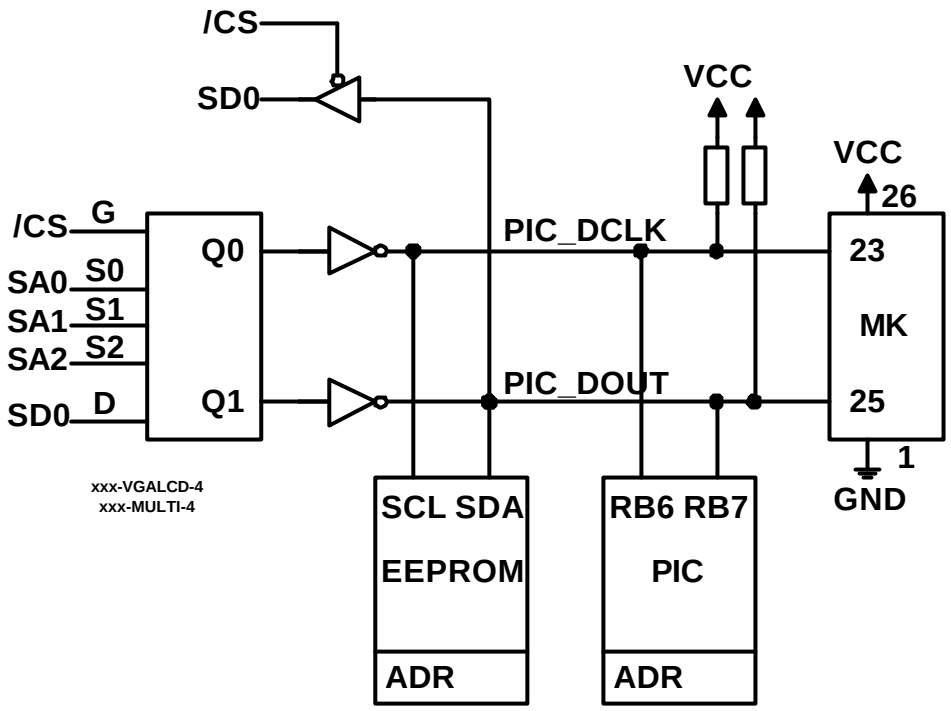
Attention: **These devices are for BIOS-access only; reading from or writing to them may cause data corruption and system failure.**

Example: Assume you're writing a software utility and want the I2DAT line to be pulled HIGH.
You have to use I/O-port 50h and set the data lines D0 to D3, with D3=1 holding the output level for I2DAT and D2 to D0=110 choosing the right output line Q6.
The corresponding OUT instruction would be: **OUT 50h,0Eh**

Several OUTs and INs will be necessary to address one I²C bus device and write or read a byte, because the start and stop conditions and every single bit have to be set separately as well as the I2CLK line.

INDEX

4.10. AT96/ISA96/ISA-Multi-4 , ISA-VGALCD-4, PC/104-VGALCD-4



Attention:The drivers of the I²C lines on these boards are inverting!

I/O address to generate /CS	:	C190h for PIC_DCLK C191h for PIC_DOUT
Device address of EEPROM	:	1010 000xb
Device address of PIC16C84	:	0101 101xb
Reserved address:	:	0101 100xb

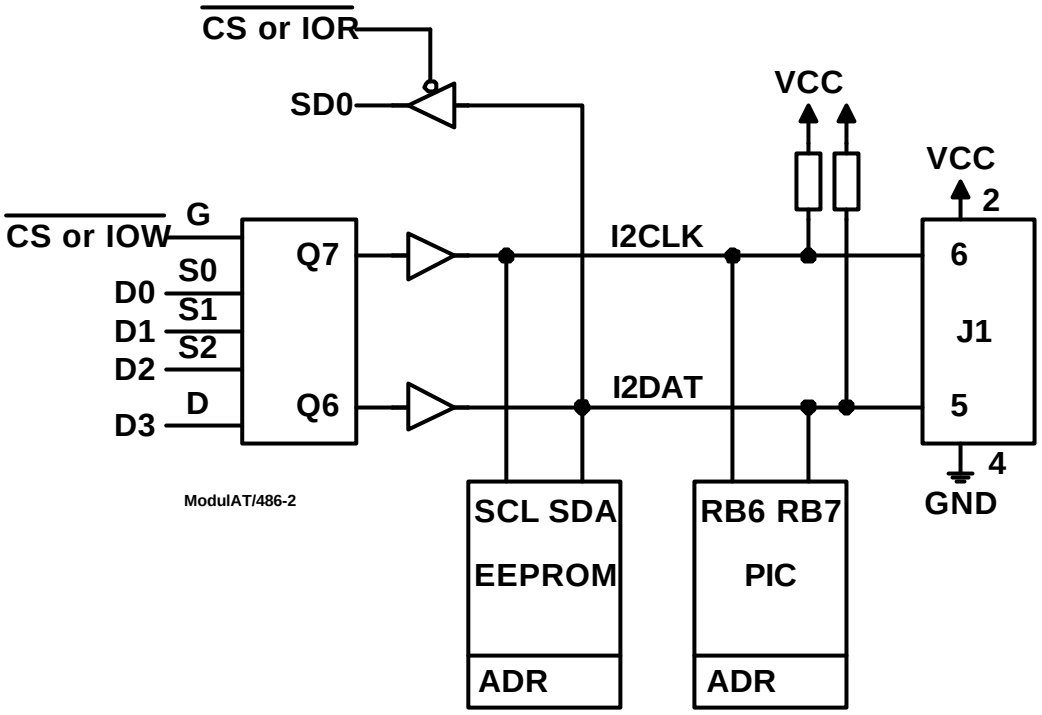
Attention: **These devices are for BIOS-access only; reading from or writing to them may cause data corruption and system failure.**

Example: Assume you're writing a software utility and want the PIC_DOUT line to be pulled HIGH. You have to use I/O-port C191h and set the data line D0=0 (gets HIGH because of inversion) holding the output level for PIC_DOUT.
The corresponding OUT instruction would be: **OUT 0C191h,00h**

Several OUTs and INs will be necessary to address one I²C bus device and write or read a byte, because the start and stop conditions and every single bit have to be set separately as well as the I2CLK line.

INDEX

4.11. ModulAT/486-2



I/O address to generate /CS : 50h

Device address of EEPROM : 1010 000xb
Device address of PIC16C84 : 1011 000xb
Reserved address: : 0101 100xb

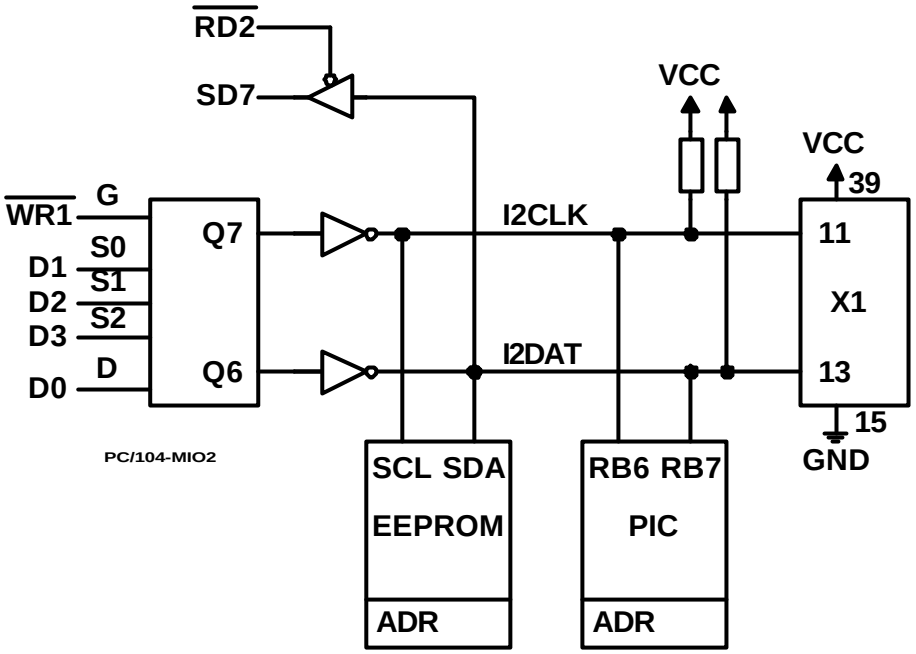
Attention: **These devices are for BIOS-access only; reading from or writing to them may cause data corruption and system failure.**

Example: Assume you're writing a software utility and want the I2DAT line to be pulled HIGH.
You have to use I/O-port 50h and set the data lines D0 to D3, with D3=1 holding the output level for I2DAT and D2 to D0=110 choosing the right output line Q6.
The corresponding OUT instruction would be : **OUT 50h,0Eh**

Several OUTs and INs will be necessary to address one I^2C bus device and write or read a byte, because the start and stop conditions and every single bit have to be set separately as well as the I2CLK line.

INDEX

4.12. PC/104-MIO2



Attention: The drivers of the I²C lines on these boards are inverting!

I/O address to generate /WR1	:	depending on BIOS location	
		C000,D000	: 018Fh
		C400,D400	: 0210h
		C800,D800	: 0300h
		CC00,DC00	: 0410h
I/O address to generate /RD2	:	depending on BIOS location	
		C000,D000	: 118Fh
		C400,D400	: 1210h
		C800,D800	: 1300h
		CC00,DC00	: 1410h

Device address of EEPROM	:	1010 000xb
Device address of PIC16C84	:	0101 101xb
Reserved address:	:	0101 100xb

Attention: **These devices are for BIOS-access only; reading from or writing to them may cause data corruption and system failure.**

Example: Assume you're writing a software utility and want the I2DAT line to be pulled HIGH. Your BIOS location is jumpered to D800.
You have to use I/O-port 0300h and set the data lines D0 to D3, with D0=0 (gets HIGH because of inversion) holding the output level for I2DAT and D3 to D1=110 choosing the right output line Q6.
The corresponding OUT instruction would be: **OUT 0300h,0Ch**

Several OUTs and INs will be necessary to address one I²C bus device and write or read a byte, because the start and stop conditions and every single bit have to be set separately as well as the I2CLK line.

INDEX