Related Products	coolMONSTER/C3 (LEU6) coolMONSTER/P3 (LEU6)
Subject	I^2C bus on coolMONSTER/P3 and /C3
Document Name	I2CLEU6_E610.doc
Usage	Common

1. REVISION HISTORY

Date	Document Name	Subjects added, changed, deleted	Changed by
20-Dec-02	I2CLEU6_E610	Initial release of Application Note	H. Bruhn

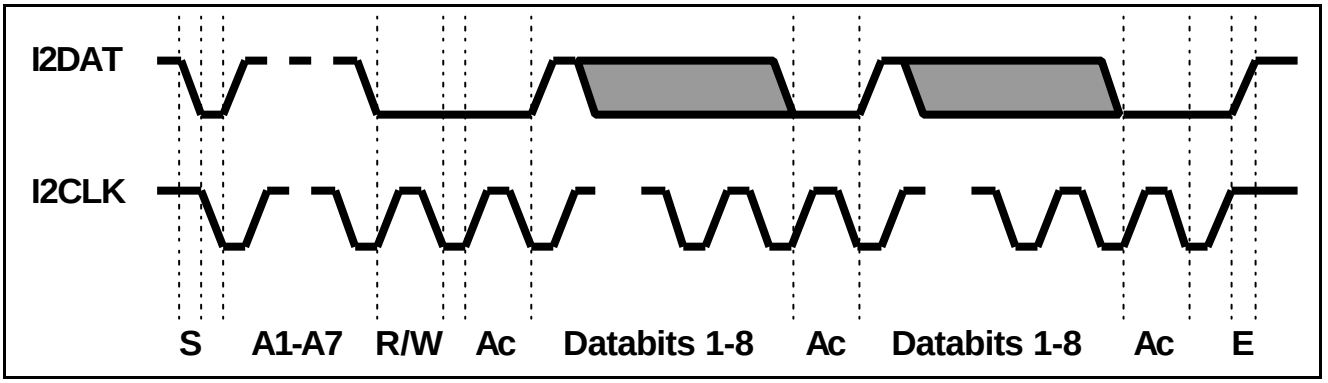
2. TABLE OF CONTENTS

3. GENERAL INFORMATION ABOUT I²C BUS

3.1. Introduction to I2C Bus

The Inter-IC bus (I^2C) 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 I^2C bus interface can be connected to the system by simply clipping it to the I^2C 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 I^2C bus. For detailed information (e.g. timing problems, characteristics of devices) refer to I^2C bus specifications, data books and specialized textbooks.

3.2. I²C Bus on Kontron Embedded Modules GmbH Boards

The I²C bus interface on **Kontron Embedded Modules GmbH** 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.

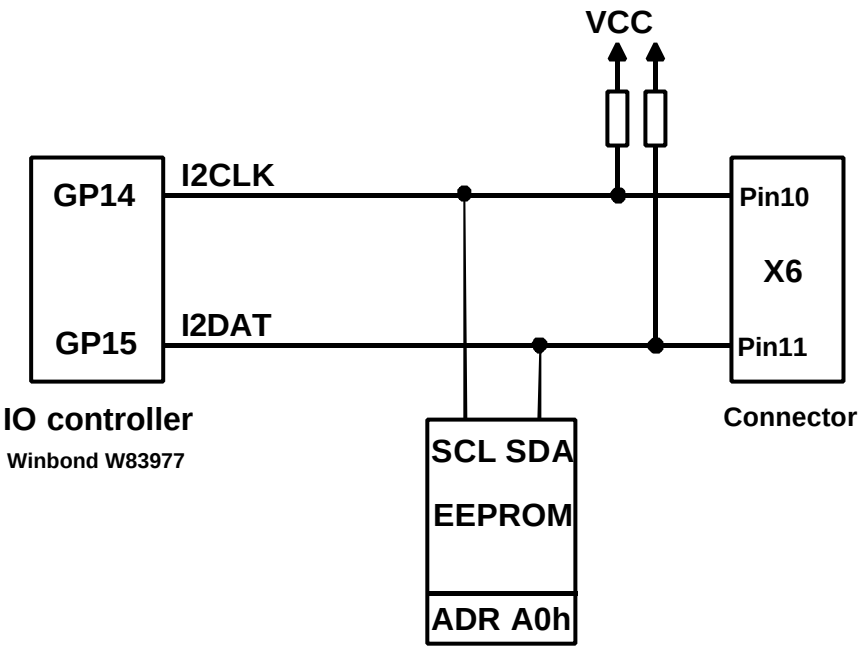
Note: This kind of interface does not support external masters.

On different **Kontron Embedded Modules GmbH** boards the two I²C bus lines are not offered on identical connectors. They are also not driven the same way. 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 special **Kontron** board the application note is related to. Therefore the information herein cannot be used for other products of **Kontron**.

4. ACCESS TO I2C BUS ON COOLMONSTER/C3 AND P3

4.1. Schematics



4.2. Used I2C bus addresses

I/O address to generate /CS	:	100h
Device address of EEPROM	:	1010 000xb
Reserved address	:	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.**

4.3. Programming information

Two General Purpose I/O pins of the Winbond W83977 I/O controller control the I²C Bus signals. If the I/Os 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. The programming example below shows exactly how the I²C Bus signals can be controlled.

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

```
W83977GP1Base     EQU   100h              ; General Purpose Port 1 base address
I2CCLK_MASK       EQU   010h              ; GP14 (bit4 of GP1)used for I2C bus clock (SCL)
I2CDAT_MASK       EQU   020h              ; GP15 (bit5 of GP1)used for I2C bus data (SDA)

I2CDAT_BIT        EQU   5

I2CLKlow:      call     i2cSetSclToOutput
               mov      dx,W83977GP1Base
               in       al, dx
               and      al, NOT I2CCLK_MASK
               out      dx, al

I2CLKhigh:     call     i2cSetSclToInput

I2CDATlow:     call     i2cSetSdaToOutput
               mov      dx,W83977GP1Base
               in       al, dx
               and      al, NOT I2CDAT_MASK
               out      dx, al

I2DAThigh:     call     i2cSetSdaToInput

ReadI2DAT:     mov      dx, W83977GP1Base
               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
               call     sioWb977RegWrite
               mov      al, 0E4h                  ; GP14 control
               call     sioWb977RegRead
               or       ah, 01h                   ; set to input
               call     sioWb977RegWrite
               popf
               ret
i2cSetSclToInput  ENDP
```

```
;-------------------------------------------------------------------------------------------------
; 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
                call    sioWb977RegWrite
                mov     al, 0E4h                        ; GP14 control
                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
                call    sioWb977RegWrite
                mov     al, 0E5h                        ; GP15 control
                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
                call    sioWb977RegWrite
                mov     al, 0E5h                        ; GP15 control
                call    sioWb977RegRead
                and     ah, NOT 01h                     ; set to output
                call    sioWb977RegWrite
                popf
                ret
i2cSetSdaToOutput ENDP
```

The programming described 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.

```
;-------------------------------------------------------------------------------------------------------
; 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.**