Related Products	littleMOSTER2, LM2P/P (LEU2) coolMONSTER, CMP/P (LEU2) coolMONSTER/S, CMP/S (LEU3)
Subject	I^2C bus on Pentium Slot-CPUs
Document Name	I2CLEU2_3_E510.doc
Usage	Common

1. REVISION HISTORY

Date	Document Name	Subjects added, changed, deleted	Changed by
20-Dec-02	I2CLEU2_3_E510.DOC	Initial release of Application Note	H. Bruhn

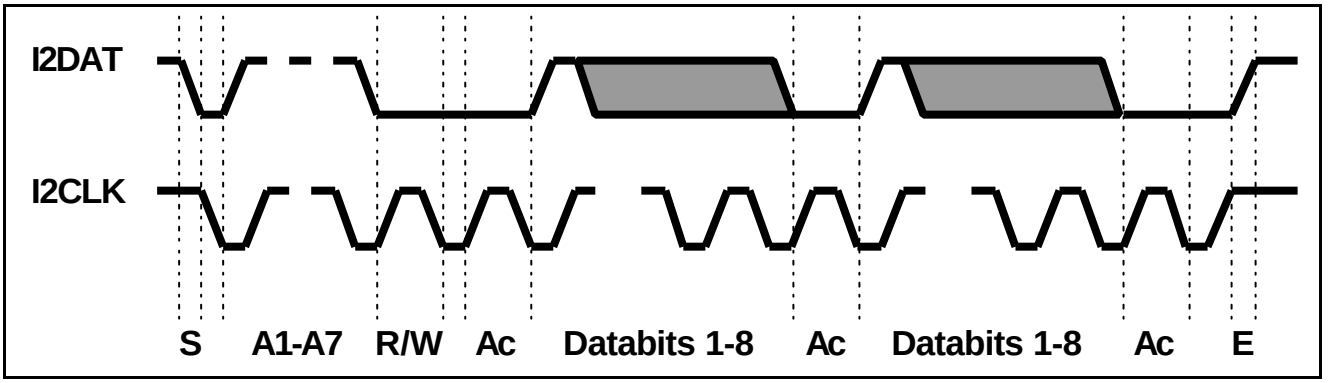
2. TABLE OF CONTENTS

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.

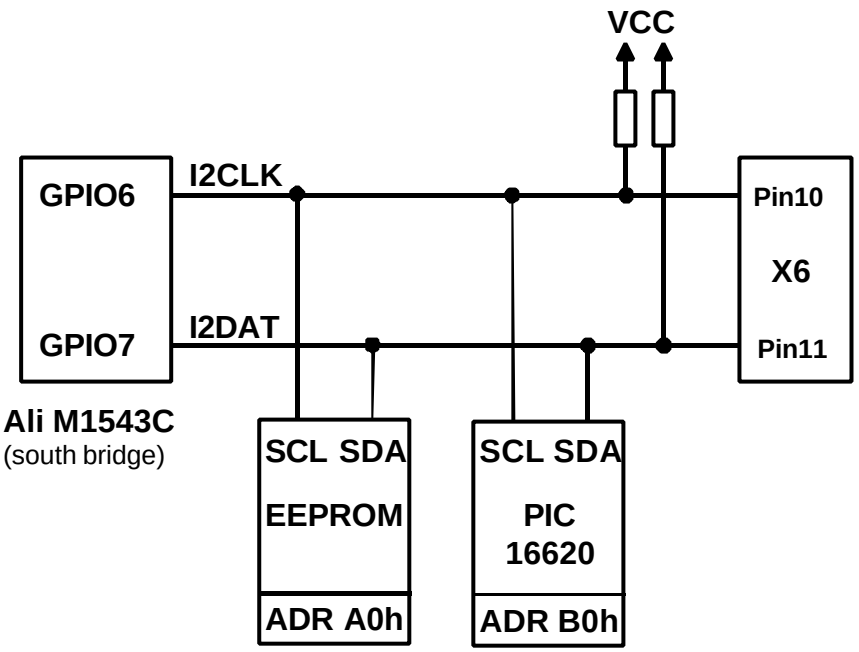
Note: This kind of interface does not support external masters.

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 PENTIUM SLOT CPU BOARDS

4.1. Schematics



4.2. Used I2C bus addresses

Device address of EEPROM	:	1010 000xb
Device address of PIC16620	:	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

The I²C Bus signals on these Pentium Slot CPU boards are controlled by two General Purpose I/Os 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 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 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)		
	Bus	Device	Function
coolMONSTER , coolMONSTER/S littleMONSTER2	00h	07h	00h

4.3.1. Driving the I²C bus lines

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

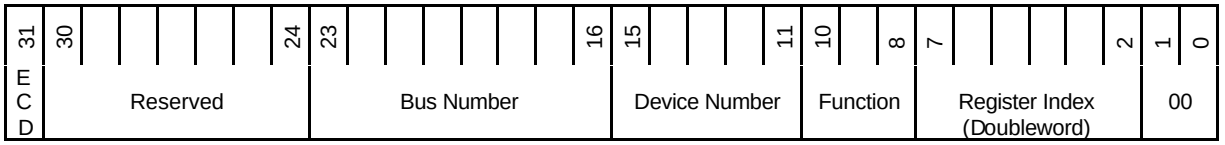
```
CONFIG_ADDR             EQU   0CF8h             ; configuration address register
CONFIG_DATA             EQU   0CFCh             ; configuration data register
CONFIG_ADDR_PMU         EQU   8000385Ch         ; PCI config cycle to bus 00h, device 02h,
                                                ; function 00h, register index 5Ch (make visible/
                                                ; hide PMU register – bit2 in reg 5Fh)
CONFIG_ADDR_SET         EQU   8000387Ch         ; PCI config cycle to bus 00h, device 02h, 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 visible. ---------------------------------

      mov eax, CONFIG_ADDR_PMU                  ; make PMU device visible
      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
;-------------------------------------------------------------------------------------
```

4.3.2. PCI configuration port information

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 (M1543C). ***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**



Bit [1:0] - are reserved and must be zero (because of doubleword 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.**