Related Products	MOPS/386A (P389)
Subject	I^2C bus on MOPS/386A
Document Name	I2CP389_E210.doc
Usage	Common

1. REVISION HISTORY

Date	Document Name	Subjects added, changed, deleted	Changed by
18-Dec-02	I2CP389_E210	Initial release of Application Note	H. Bruhn

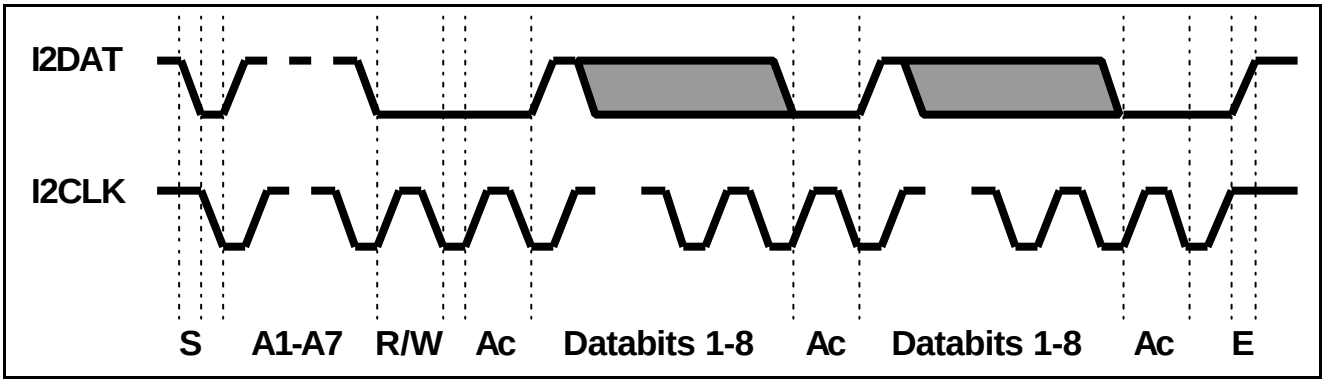
2. TABLE OF CONTENTS

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.

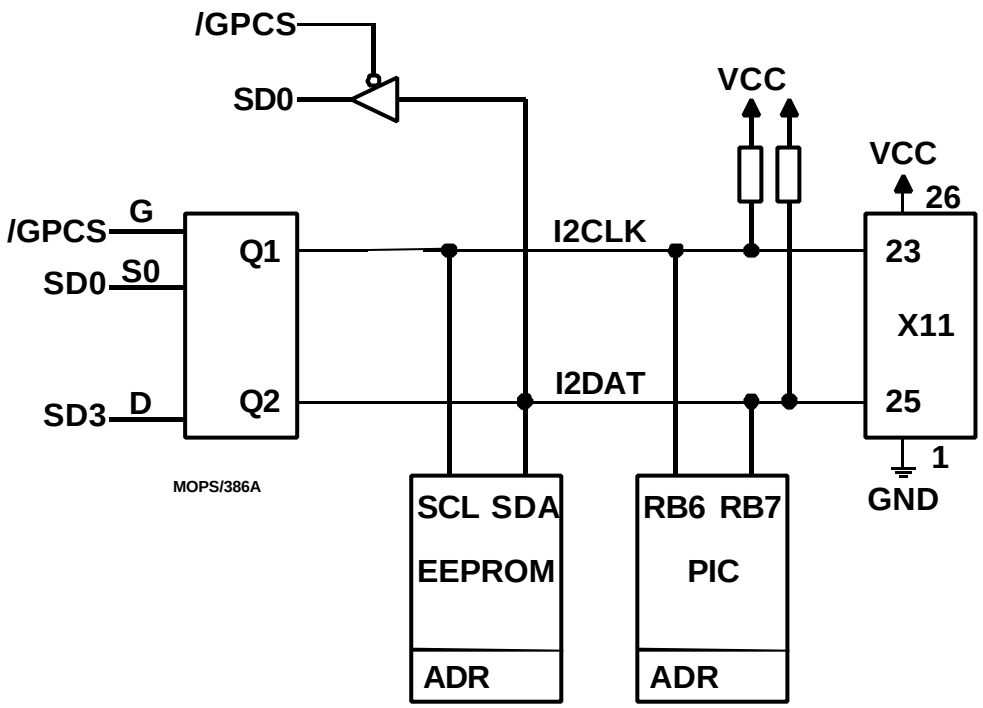
Note: This kind of interface does not support external masters.

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 MOPS/386A

4.1. Schematics



The signal line SD0 selects the I2CLK or I2DAT line for output.
The signal line SD3 holds the "data", when set to 0 the selected line I2CLK or I2DAT is pulled LOW.

4.2. Used I²C bus addresses

I/O address to generate /CS : 51h

Device address of EEPROM : 1010 000xb
Device address of PIC16F84 : 0101 101xb
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.**

4.3. Programming information

First of all it is good to define some global constant values, which can be used by the functions later on.

```
CTL_REG_1               EQU   051h                    ; control port address
PORT61                  EQU   061h                    ; XT port 61hex
REFRESH_STATUS          EQU   010h                    ; Refresh status bit port 61hex
I2DAT_HIGH              EQU   ((0 shl 0) + (1 shl 3)) ; bit 0 set to 0 and bit 3 set to 1
I2DAT_LOW               EQU   ((0 shl 0) + (0 shl 3)) ; bit 0 set to 0 and bit 3 set to 0
I2CLK_HIGH              EQU   ((0 shl 0) + (1 shl 3)) ; bit 1 set to 0 and bit 3 set to 1
I2CLK_LOW               EQU   ((0 shl 0) + (0 shl 3)) ; bit 1 set to 0 and bit 3 set to 0
```

Here are the functions for driving the I^2C lines.

```
;-------------------------------------------------------------------------------------------------
; Name:     I2cSclLow
;           I2cSclHigh
;           I2cSdaLow
;           I2cSdaHigh – functions for driving the I2C lines
; Entry:    none
; Exit:     none
; Modified: destroy AX and minor flags
;-------------------------------------------------------------------------------------------------

I2cSclLow   PROC NEAR PUBLIC
      mov   al, I2CLK_LOW
      out   CTL_REG_1, al
      ret
I2cSclLow   ENDP

I2cSclHigh  PROC NEAR PUBLIC
      mov   al, I2CLK_HIGH
      out   CTL_REG_1, al
      ret
I2cSclLow   ENDP

I2cSdaLow   PROC NEAR PUBLIC
      mov   al, I2DAT_LOW
      out   CTL_REG_1, al
      ret
I2cSclLow   ENDP

I2cSdaHigh  PROC NEAR PUBLIC
      mov   al, I2DAT_HIGH
      out   CTL_REG_1, al
      ret
I2cSclLow   ENDP
```

Here is the function to read the current status of the I^2C data line.

```
;-------------------------------------------------------------------------------------------------
; Name:     I2cReadSda
; Entry:    none
; Exit:     none
; Modified: destroy AX and minor flags
;-------------------------------------------------------------------------------------------------

I2cReadSda  PROC NEAR PUBLIC
      in    al, CTL_REG_1
      and   al, 001h
      ret
I2cReadSda  ENDP
```

True I²C bus devices allow a maximum speed of 100KHz. Therefore it is necessary to have a delay when driving the I²C bus. The following function offers a delay by number of 15 microsecond periods. This function should not be called with 0 in CX. The entry value in CX therefore should be minimum 1. For the Kontron onboard devices we recommend a value of 5 in CX which results in a clock delay of 75 microseconds.

However, there are many devices with I²C interface on the market that may not allow a speed of 100KHz. Please read the datasheet of your I²C device very carefully to find out the I2CLK delay. Use a suitable entry value in CX to call this function for your device.

```
;-------------------------------------------------------------------------------------------------
; Name:     FixedDelay
; Entry:    CX – Time in 15us (DO NOT CALL with CX==0 or CX>0FFF8h !!!)
; Exit:     none
; Modified: CX and flags
;-------------------------------------------------------------------------------------------------

FixedDelay  PROC NEAR PUBLIC
      pusha
      pushf
      mov   ah, NOT(REFRESH_STATUS)            ; force ah, al miscompare
      add   cx, 2
loopSampleRefresh:
      jcxz  $+2                                ; I/O delay
      in    al, PORT61                         ; load port 61hex status
      and   al, REFRESH_STATUS                 ; isolate refresh status
      cmp   al, ah                             ; match last status?
      je    loopSampleRefresh                  ; yes-loop if hasn´t toggled
saveStatus:
      mov   ah, al                             ; save new state of refresh status
      loop loopSampleRefresh                   ; repeat until timer expires
      popf
      popa
      ret
FixedDelay  ENDP
```

The MOPS/386A has an onboard keyboard matrix PIC which uses the lines I2CLK and I2DAT as matrix scan lines, too. Therefore it is necessary to stop the PIC from driving and scanning the keyboard matrix interface, thus freeing the I²C bus.
This must be done by holding the data line I2DAT low for at least 60 microseconds to ensure that the keyboard matrix PIC recognizes the I²C bus arbitration and stops scanning.
The PIC will start the matrix scan again after five millisceconds of no action on the I²C bus.
The following function does the necessary bus arbitration and has to be called whenever a transfer of data to a device on the I²C bus has to be done.

```
;-------------------------------------------------------------------------------------------------
; Name:     I2cArbit
; Entry:    none
; Exit:     none
; Modified: may change AX, DX and flags
;-------------------------------------------------------------------------------------------------

I2cArbit    PROC NEAR PUBLIC
      call  I2cSdaLow                          ; force I2DAT low
      push  cx
      mov   cx, (60/15)+1                      ; 60 microseconds delay
      call  FixedDelay
      pop   cx
      call I2cSdaHigh                          ; release I2DAT
      ret
I2cArbit    ENDP
```