APPLICATION NOTE

USING I2C CARDS WITH THE TDA8002

AN96138

Philips
Semiconductors

PHILIPS

Abstract

This application note deals with the smart card interface TDA8002.
It explains how to handle I^2C cards with this interface.

APPLICATION NOTE

USING I2C CARDS WITH THE TDA8002

Author(s):
Cécile KOHLER

Application Laboratory - Paris
France

Keywords
TDA8002
Smart Cards
I^2C protocol

Date : November 1996

CONTENTS

1. Introduction

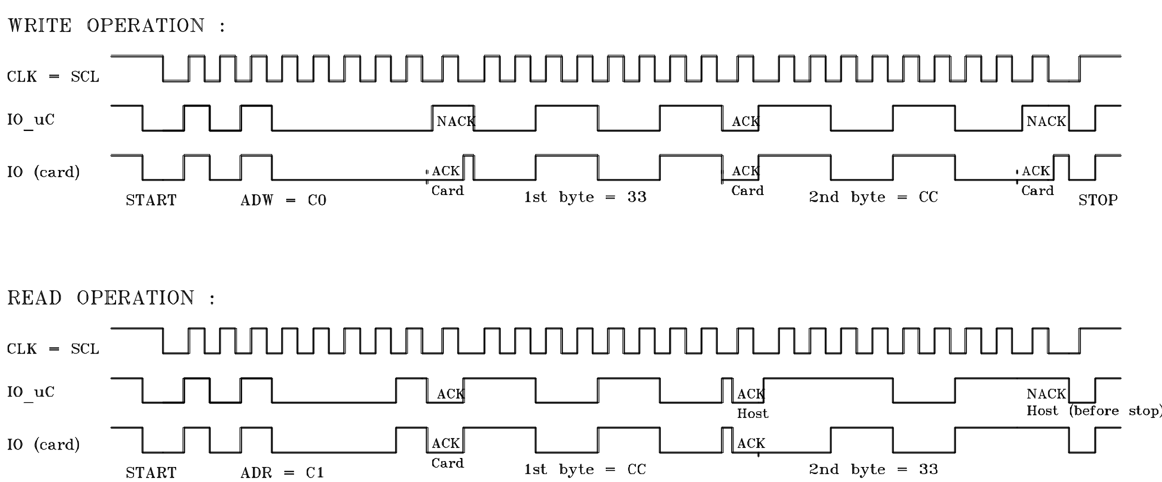
Due to the internal IO line configuration, it is not possible to communicate with standard I^2C cards using only IO as SDA line.
This document gives two hardware methods to solve this problem by separating the I/O input and output on the host side :
- the first method uses IO_uC as output and AUX2_uC (or AUX1_uC) as input,
- the second method uses IO_uC as output and a standard microcontroller port of the host plus a gate as input.

This second method should be implemented if the auxiliary outputs are already used to drive C4 and C8 contacts of other synchronous cards.

This report is written for the version C2 of the TDA8002, where CLK and STROBE have the same polarity.

2. TDA8002 problem description

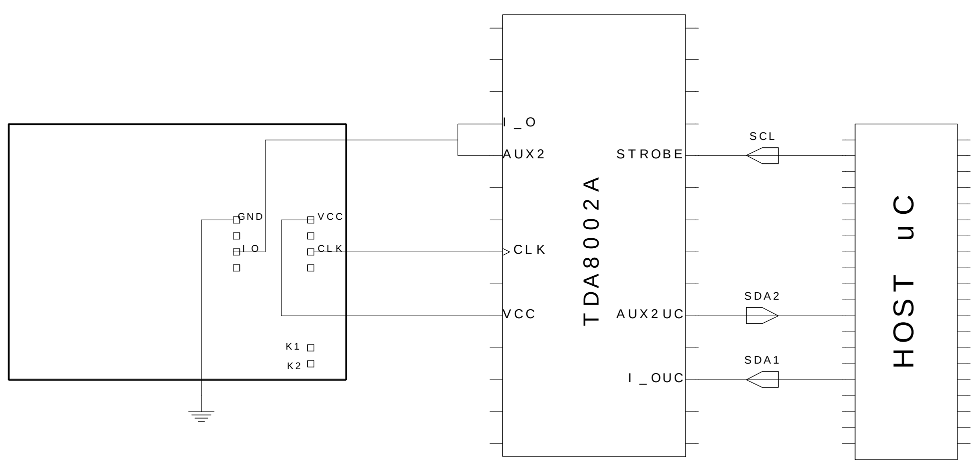
The TDA8002 I/O line is a false bi-directionnal line with IO pin on card side and IO_uC pin on host side. It is defined in the specification as follows :
« the first side on which a falling edge is detected becomes a master (input). An anti-latch circuitry first disables the detection of the falling edge on the other side, which becomes slave (output) ».
So, the I/O line can not switch from transmission configuration to reception without a falling edge. That is not compatible with I^{2}C specification where, during a communication (between start and stop conditions), the default level on SDA may be the low level and where the bit ACK set by the receiver may leave SDA at a low level.
The problem occurs in transmission to the card if the host sets the SDA line high for receiving the ACK bit after the card has set the SDA line low for transmitting the ACK bit ... Then, the SDA line is never high and there is no high to low transition enabling the receiver to become transmitter.
The same problem happens in reception from the card after the ACK bit has been sent by the host : if the host sets the SDA line high to receive the next byte after the card has prepared its first bit to low (if the byte is smaller than 80 H), there is no high to low transition and the first bits are lost until the card sets the SDA line high then low.



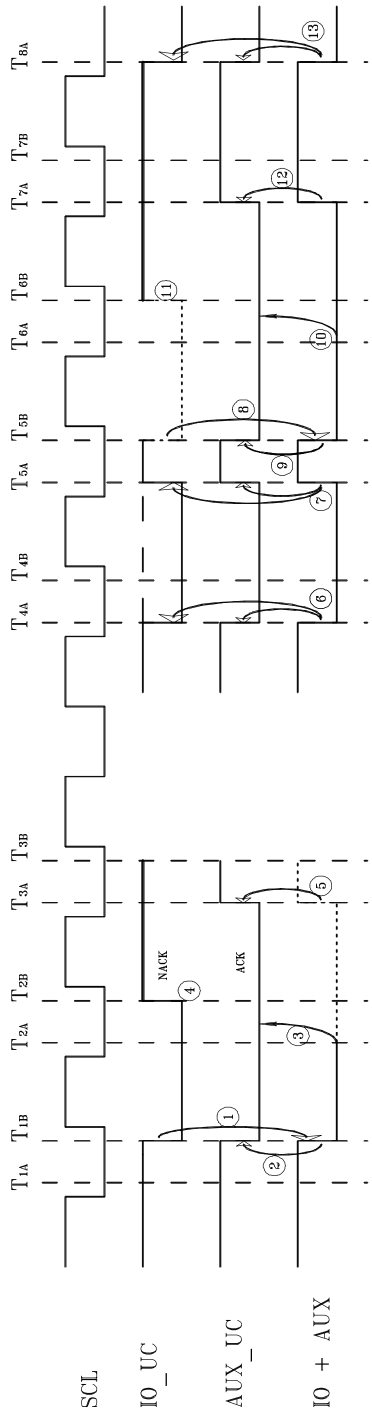
3. First solution : use of IO and AUX

The first solution proposed is to use two lines for handling the card SDA : IO and AUX2 (or AUX1 if AUX2 is not available). AUX2 has been chosen in this example because IO and AUX2 are physically close to each other and AUX1 may be used to drive the C4 contact of some synchronous cards.
On the card side, IO and AUX2 should be linked and on the system side, IO_uC is used as output (SDA1) and AUX2_uC is used as input (SDA2).
The I^2C clock SCL is transferred on the CLK pin of the card through the STROBE input of the TDA8002.

This solution is available if the microcontroller input is in high impedance state at high level. If it is not the case, it may be necessary to add an open collector gate (7407 type) between the TDA8002 AUX output and the microcontroller input.



Explanation :



Case A : write data from host to card
ACK from card to host

a-	IO_uC sets the last data bit at 0	at T_{1B}	
	IO falls low		1
	AUX2_uC copys the low level of IO		2
b-	The card sets IO low for ACK bit	at T_{2A}	3
	(IO_uC is still a 0)		
	AUX2_uC copys the ACK		
	if IO_uC is set high by the software	at T_{2B}	4
	(to ev. receive the ACK bit), it stays in emission mode		
c-	The card let IO high to be in reception mode	at T_{3A}	5
	AUX2_uC goes high and IO_uC is still in emission		

At T_{3B}, IO may write its first data bit to 1 or 0.

Case B : read data from card to host
ACK from host to card

IO_uC is set high during the reception mode

a-	The card sets its last data bit to 0	at T_{4A}	6
	AUX2_uC copys the low level		
b-	The card sets IO high for the reception of ACK	at T_{5A}	7
	AUX2_uC and IO_uC go high		
c-	IO_uC is set low for ACK	at T_{5B}	
	IO falls low		8
	AUX2_uC copys IO low		9
d-	The card sets the first data bit at 0 on IO	at T_{6A}	10
	AUX2_uC stays at 0 during this bit		
e-	IO is set high for reception but	at T_{6B}	11
	stays in emission mode until next low level		
	on IO (13)		

In this diagram, the microcontroller IOs are bi-directionnal, and are put in reception by beeing set high (8051 IO types). If it is not the case, IO_uC may stay in emission mode, but it must be set high during the read operations.
In this configuration, and almost during the read operation **it is important that the microcontroller always sets its level on IO_uC at T_{nB}** (low level for ACK) **after the card has set the IO line at T_{nA}** (high during host ACK). The solution is to set the IO_uC bit just before setting SCL high, because the card always sets its IO bit on the falling edge of SCL and reads the bit on the rising edge.

Example :

The following routines give an example of use with a 80C51 microcontroller. A complete example software and its schematic are given in annex 1.

```
start:                          ;generation of a start condition
        setb    SDA2
        setb    SCL
        setb    SDA1
        call    wait6
        clr     SDA1
        call    wait6
        clr     SCL


;----------------------
iicwbyte:                       ;routine for writing 1 byte
        setb    c               ;acknow bit
        mov     r7,#9           ;8 bits + acknow
byte_loop:
        rlc     a
        mov     SDA1,c          ;write the bit
        nop
        setb    SCL
        call    wait6
        clr             SCL
        djnz    r7,byte_loop
```

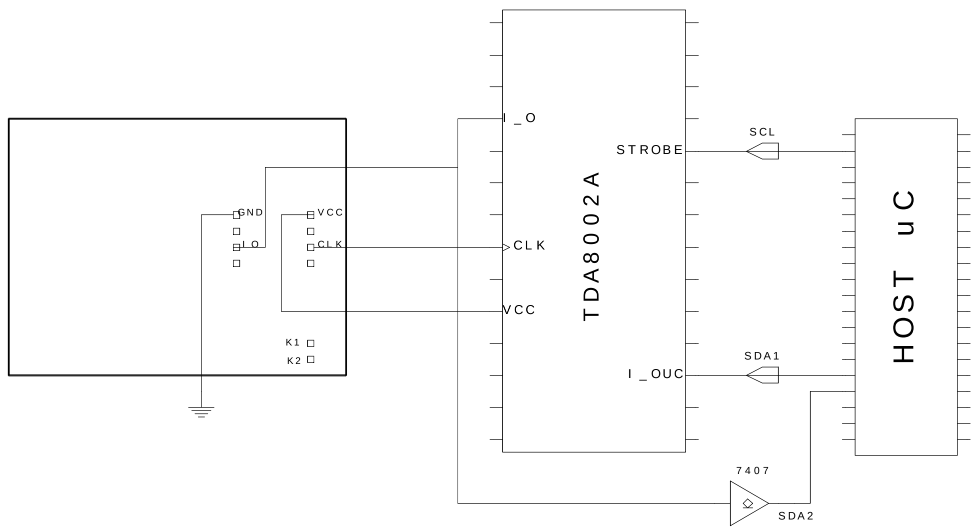
```
;-----------------------
iicrbyte:                       ;read one byte and send ACK
        clr     c               ;acknow bit
iicr_nack_byte:                 ;(read one byte and send NACK)
        mov     a,#0ffh
        mov     r7,#9           ;8 bits + acknow
byte_loop:
        rlc     a
        nop
        nop
        setb    SCL
        call    wait4
        mov     c,SDA2          ;read the bit
        clr     SCL
        djnz    7,byte_loop

;-----------------------
stop:                           ;generation of a stop condition
        clr     SDA1
        setb    SCL
        call    wait6
        setb    SDA1
        setb    SDA2

;-----------------------
wait6:
        nop
        nop
wait4:
        ret
```

4. Second solution : use of IO and one microcontroller port

This second solution is also the use of two lines for driving the card SDA, but only one goes through the TDA8002, the IO line which is used as output, the second line beeing directly derived from the host microcontroller and used as input.
In order to be sure that this second line is always in a high impedance state, a 7407 gate (non inverter, open collector output) is put between the host port and the IO pin.
The two lines are linked on the card I/O pin.



The driving software is the same as above.

This solution requires to protect the 7407 gate against ESD up to 6 kV while the first solution does not require any external component because the AUX1 or AUX2 lines have the same level of ESD protection as I/O.

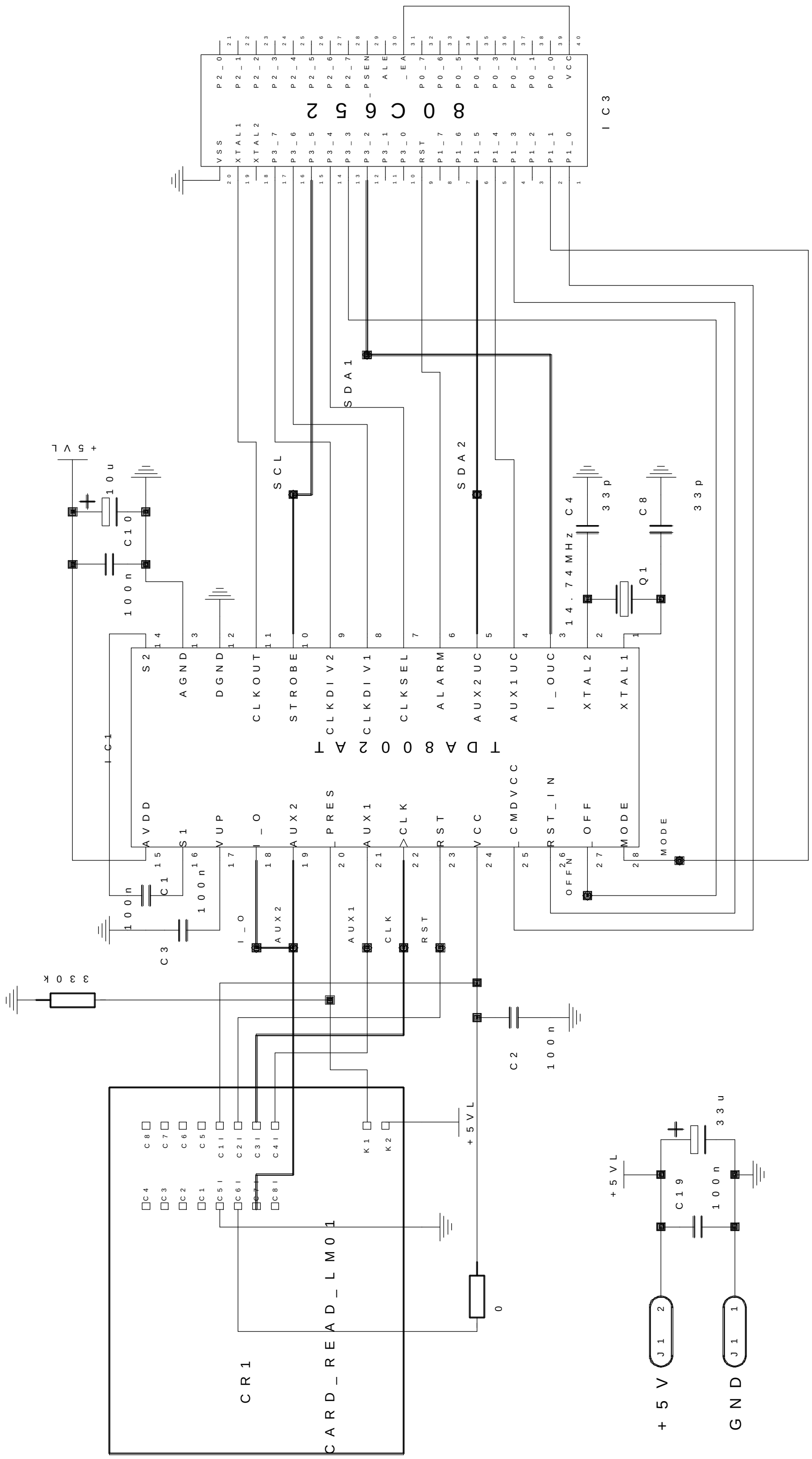
ANNEX

Example using AUX2 and IO

This example is a standard application of the TDA8002 driving asynchronous and synchronous smart cards where the C8 contact is not connected.

Only the routines for driving I^2C cards are given here, with a very simple application layer written for the Philips D2000 card:
- read 4 bytes at current address
- write FF, FE, FD, FC at address 0
- read 4 bytes at address 04.

The case of a non acknowledge is not handled.


80C652
IC3
TDA8002AT
IC1
SCL
SDA1
SDA2
+5VL
10u
C10
100n
C4
33p
C8
33p
Q1
14.74MHz
C1
100n
C3
100n
C2
100n
330K
CR1
CARD_READ_LM01
C19
100n
33u
J1
+5 V
GND

```
SDA2 :  p1.5     =       AUX2_uC of TDA8002
SDA1 :  p3.2     =       IO_uC of TDA8002
SCL  :  p3.5     =       STROBE of TDA8002

adw  = 0a0h              ;i2c write address (D2000 card)
adr    = 0a1h            ;i2c read address

nbroct: 1 byte   =       number of data to read or write
                         (including memory address if needed)
stadd:  1 byte   =       start address of the data buffer
addbuf: 1 byte   =       card memory address buffer
datbuf: 8 bytes  =       data buffer

;* datbuf must be placed just after addbuf in RAM



;****************************************
;       Master I2C
;
; - Start condition sub-routine
; - The iicbyte routine write the data on
; SDA1 on a low level of SCL and read the
; data on SDA2 on the high level of SCL :
; - the data to send must be in the Acc,
; and the read data is available in Acc.
; - To read, Acc must be first set to FFh.
; - ACK bit is available in the carry.
;
; - The ACK error routine depends on the
; application, and is not handled here.
;****************************************

start:                          ;generation of a start condition
        setb    SDA2            ;AUX2 is used as input
        setb    SCL
        setb    SDA1
        call    wait6
        clr     SDA1
        call    wait6
        clr     SCL
```

```
iicwbyte:                       ;routine for writing or reading one byte
        setb    c               ;acknow bit

iicbyte:
        mov     r7,#9           ;8 bits + acknow
byte_loop:
        rlc     a
        mov     SDA1,c          ;write the bit
        nop
        setb    SCL
        call    wait4
        mov     c,SDA2          ;read the bit
        clr     SCL
        djnz    r7,byte_loop

        ret

;----------------------

iicrbyte:                       ;read one byte and send ACK
        clr     c               ;acknow bit
iicr_l_byte                     ;(read one byte and send NACK)
        mov     a,#0ffh
        jmp     iicbyte

;----------------------

stop:                           ;generation of a stop condition
        clr     SDA1
        setb    SCL
        call    wait6
        setb    SDA1
        setb    SDA2

;----------------------
```

```
wait6:
        nop
        nop
wait4:
        ret
;******************************

write_iic:
        mov     r0,stadd        ;data buffer base address
        mov     r1,nbroct       ;number of data to transmit
        mov     a,#adw          ;write address

        call    start           ;start condition + write address
        jc      ack_erro        ;

write_loop:
        mov     a,@r0
        call    iicwbyte
        jc      ack_error
        inc     r0              ;increment data address
        djnz    r1,write_loop   ;decrement data counter

        ret

;******************************

read_iic:
        mov     r0,stadd        ;data buffer base address
        mov     r1,nbroct       ;number of data to receive
        mov     a,#adr          ;read address

        call    start           ;start condition + read address
        jc      ack_error

read_loop:
        djnz    r1,read_bytes
        jmp     read_last

read_bytes:
        call    iicrbyte        ;read the byte and send a ACK bit
        mov     @r0,a
        inc     r0
        jmp     read_loop
```

```
read_last:
        setb    c
        call    iicr_l_byte     ;read the last byte and send a NACK
        mov     @r0,a

        ret

;---------------
ack_error:
        nop
        ret

;********************************
; Initialisation of the TDA8002
; for the I2C communication
;********************************
init:
        mov     r0,#0ffh
        clr     a
clr_ram:
        mov     @r0,a
        djnz    r0,clr_ram

init_tda8002:
        setb    mode
        setb    clksel
        setb    SDA1            ;AUX_µC and IO_µC must be set high before activation
        setb    SDA2
        setb    SCL

reset_card:                     ;activation of the VCC
        clr     cmdvccn
        mov     r2,#250
```

```
wait_activ:
        djnz    r2,$

        call    seq1

        call    seq2

        call    seq3

end:
        jmp     end

;---------------

seq1:                           ;read 4 bytes at current address
        mov     r0,#datbuf
        mov     stadd,r0
        mov     nbroct,#5
        call    read_iic
        call    stop
        ret

;---------------

seq2:                           ;write FF FE FD FC at address 0
        mov     r0,#addbuf
        mov     stadd,r0
        mov     nbroct,#5       ;address + 4 bytes
        mov     addbuf,#0       ;Card memory address
        mov     r0,#datbuf
        mov     r1,#0ffh
        mov     r2,#4
s2_wloop:
        mov     a,r1
        mov     @r0,a           ;write FF FE FD FC in datbuf
        inc     r0
        dec     r1
        djnz    r2,s2_wloop

        call    write_iic
        call    stop
        ret
```

```
;---------------

seq3                            ;read 4 bytes at address 4
        mov     r0,#addbuf
        mov     stadd,r0
        mov     nbroct,#1
        mov     addbuf,#4       ;card memory address
        call    write_iic

        mov     r0,#datbuf
        mov     stadd,r0
        mov     nbroct,#4
        call    read_iic
        call    stop
        ret
```