APPLICATION NOTE

EASY SOFTWARE DEVELOPMENT FOR TDA8006
Preliminary Library Reference
Release 1.3

AN/97080

C51 KEIL Compiler

Philips Semiconductors

PHILIPS

APPLICATION NOTE

EASY SOFTWARE DEVELOPMENT FOR TDA8006 Preliminary Library Reference

C51 KEIL Compiler

Author(s) :
Jean Pierre BOURNAS
Application Laboratory - Paris
France

Keywords
TDA8006
Smart card
Software
T=0, T=1 protocol
ISO7816-3, ISO7816-4

Date : November 1997

Preface

This document is the specification of the library software in C language written for the TDA8006 to handle the communication between a TDA8006 and a smart card.

The library offers the functions to handle the complete communication with asynchronous smart card following **ISO 7816-3 standard protocol T=0 and T=1.**

A function for the **APDU transportation according the ISO 7816-4** is also available.

6 main functions are available:

1- **in_instr()** for incoming card instruction in T=0 protocol. (ISO 7816-3)

2-**out_instr()** for outgoing card instruction in T=0 protocol. (ISO 7816-3)

3-**XmitAPDU()** for APDU transportation in T=0 and T= 1 protocol. (ISO 7816-4)

4-**power_up()** for activation of the smard_card with anlysis of the **A**nswer **T**o **R**eset

5-**power_down** for deactivation of the smart_card.

6-**negoce()** assumes the PTS command.

Warning

This version uses a **buffer in auxiliary RAM** to handle the communications between the host and the card reader and between the smart card and the card reader. The buffer length is not limited. Its length depends of the data length exchanged between the host and the card reader and between the smart card and the card reader. The same buffer is used for both exchanges.

The characters received from the card are handled by **interruption**, the characters sent to the card are handled by **polling**. Only the interrupts **INT0** and **TF0** (Timer 0) are used by the library. Both interrupt services routine use the **register bank 1** and they have the same priority level**,** all functions use the register bank 0. The Timer 0 is used to check the differents Time_out. The register bank 2 and the register bank 3 are available for the software application. The call to the library functions must always use the **register bank 0**.

The library used about **10.5 Kbytes** of ROM code and **78** variables in data RAM area.

The maximum stack level used by the library is 8. The minimum depth of the stack is 16 bytes. The stack pointer is not initialized.

TABLE OF CONTENTS

1. init_clock

Syntax : unsigned char init_clock(divider)

Summary : #include « tdalib.h »

Possible divider value = 2,4,8

Description : The init_clock function set the clock card with Xtal/divider and set the UART programmable divider register (PDR) for the initial ETU = 372/F_{card}, Fosc = Xtal/4, CLKout = Xtal/4.

Return value : 1 : successful
other : divider value not supported

After execution :
Fcard = Xtal/divider
ETU = 372/Fcard
Fosc = Xtal
Fclkout = Xtal/4
UART presc. = /31

Register bank used : 0
Stack level used : 3

2. set_clock_card

Syntax : unsigned char set_clock_card(unsigned char **divider**)

Summary : #include « tdalib.h »

Description : The set_clock_card function select the clock to be sent to the card.

divider	clock card
2	Xtal/2
4	Xtal/4
6	Xtal/8
8	Fint/2(~1.25Mhz)

Return value : 1 = Successful

other value = divider not supported

Register bank used : 0
Stack level used : 2

3. set_clock_uart

Syntax : void set_clock_uart(bit **prescaler**, unsigned char **divider**)

Summary : #include »tdalib.h »

Description : The set_clock_uart function load the Programmable Divider Register PDR with the divider value and set the Clock Configuration Register CCR with the presc value.

F Note : ETU = ((256 - **divider**) * **prescaler**)/Xtal

If bit **prescaler** = 0, prescaler = 31
If bit **prescaler** = 1, prescaler = 32

Return value : None

Register bank used : 0
Stack level used : 1

4. set_clocks_card_uart

Syntax:unsigned char set_clocks_card_uart(unsigned char **div**)

Summary : #include « tdalib.h »

Description : The set_clock_card_uart function select the clock to be sent to the card and programs the uart baud rate defined by Fi and Di sent by the card during the Answer To reset.

div	clock card
2	Xtal/2
4	Xtal/4
6	Xtal/8

Prerequisites : Before using the set_clocks_card_uart function, the card must be powered and to have sent its Answer To Reset.

Return value : 1 = Successful

other value = Divider not supported.

Register bank used : 0
Stack level used : 2

5. clock_stop_high

Syntax : void clock_stop_high(void)

Summary : #include « tdalib.h »

Description : The clock_stop_high function stops the clock card at the high level

Return value : None

Register bank used : 0
Stack level used : 2

6. clock_stop_low

Syntax : #void clock_stop_low(void)

Summary : #include « tdalib.h »

Description : The clock_stop_low function stops the clock card at the low level

Return value : None

Register bank used : 0
Stack level used : 2

7. set_clock_uc

Syntax : unsigned char set_clock_uc(unsigned char **divider**)

Summary : #include « tdalib.h »

Description : The set_clock_uc select the clock frequency which can be used for the Uc clock.

Divider	Osc
00	Fint/8
0x40	Xtal
0x80	Xtal/2
0xC0	Fint/2

Return value : 1 = Successful

Other value = divider not supported

Register bank used : 0
Stack level used : 2

8. set_clockout

Syntax : unsigned char set_clock_out(unsigned char **divider**)

Summary : #include « tdalib.h »

Description : The set_clock_out function selects the frequency using by the external logic circuitry.

Divider	clk_out
00	Xtal/4
0x10	Xtal
0x20	Xtal/2

Return value : 1 = successful
other value = divider not supported

Register bank used : 0
Stack level used : 2

9. ReadFiDi

Syntax : unsigned char ReadFiDi(void)

Summary : #include « tdalib.h »

Description : The readFiDi function returns the current FiDi

Return value : FiDi value given during Answer To Reset (default value = 11)

Register bank used : 0
Stack level used : 1

10. verify_pres_card

Syntax : unsigned char verify_pres_card(void)

Summary : #include » tdalib.h »

Description : The verify_pres_card function verify the card snatching.

Return value : 1 = The card has not been snatched
other = The card has been snatched

Register bank used : 0
Stack level used : 1

11. check_pres_card

Syntax : unsigned char check_pres_card(void)

Summary : #include « tdalib.h »

Description : The check_pres_card function checks if the card is inserted in the connector.

Return value : 1 = The card is present
other = No card
Register bank used : 0
Stack level used : 1

12. init_io_buff_address

syntax : void init_io_buff_address(unsigned int **address**)

Summary : #include »tdalib.h »

Description : The init_io_buff_address function defines the start address of **Data exchange buffer in auxiliary Ram**. The Data Exchange buffer is used for data communication between the host and the card reader and between the smart card and the card reader. Its length is not limited.

It is mandatory to call this function once for instance at the begining of the main application software.

The Data Exchange buffer is used the function below :

- in_instr function
- out_instr function
- XmitAPDU function
- power_up function
- negoce function

Register bank used : 0
Stack level used : 1

13. write_aux

Syntax : void write_aux(unsigned int offset, unsigned char **data**)

Summary : #include « tdalib.h »

Description : The write_aux function write **data** in auxiliary Ram at the memory location selected by **init_io_buff_address + offset.**

Return value : None

Register bank used : 0
Stack level used : 2

Example : Write 0x55 at the address 0x10 of the data exchange buffer

```
init_io_buff_address(0x100);  //start of the data exchange buffer
write_aux(0x10,0x55);         //absolute address = 0x110
```

14. read_aux

Syntax : unsigned char read_aux(unsigned int **offset**)

Summary : #include « tda_lib.h »

Description : The read_aux function read a data in auxiliary Ram at the memory location selected by init_io_buff_address **+ offset.**

Return value : read data

Register bank used : 0
Stack level used : 2

Example : Read 2 bytes from the address 0x10 of the data exchange buffer

```
init_io_buff_address(0x100);        //start address of the data exchange buffer
byte1 = read_aux(0x10);             //absolute address = 0x110
byte2 = read_aux(0x11);             //absolute address = 0x111
```

15. power_up

Syntax:unsigned char power_up(unsigned int **ptr**,unsigned char **clock_div**,unsigned char **voltage**)

Summary : #include « tdalib.h »

Description : The power_up function :

- assert VCC (**3 or 5 volts** depending of the **voltage**)on the card. Only 2 values voltage are possible 3 or 5.
- set Fcard = Xtal/**clock_div**
- set ETU = Fcard/372 during the Answer To Reset
- set Fuc = Xtal
- set Fclkout = Xtal/4
- set UART prescaler = /31
- reset the card
- store the **A**nswer **T**o **R**eset into the Data Exchange Buffer in the auxiliary RAM, **ptr** contains the offset address of the Data Exchange Buffer.
- Decode the protocol T=0 or T=1
- Memorize WI, CWT, BWT which are used to control the communications with the card
- initialize the Guard Time Register GTR
- perform automatically the Protocol Type Selection PTS in specific mode (TA2 is present). In this case the Fi/Di sent during the Answer To Reset will be active at the end of Answer To reset.

clock_div	clock card
2	Xtal/2
4	Xtal/4
6	Xtal/8

Return value : Number of bytes of the Answer to Reset
0 = error, in this case the error type is sent back by the read_error() function.

Error type : 0x80 = Mute card
0x83 = 3 parity errors
0x82 = 3 parity errors on TS

0xC0 = Card absent
0xC1 = I/O line at 0 or no VCC
0xC2 = The protocol is neither T0 nor T1
0xC3 = In the case where TCK is present in ATR and not correct
0xC4 = Not supported. TS is neither 0x3B or 3F
0xC5 = Bad FiDi
0xC6 = ATR not supported
0xC7 = Requested VPP not supported
0xC8 = voltage parameter error;

Register bank used : 0 and 1
Stack level used : 8

Example : init_io_buff_address(0x100) ; //address of the Data Exchange Buffer in //the auxiliary RAM

status = power_up(0,4,3) //The ATR is stored at the first address of //the Data Exchange Buffer, the clock // card is Xtal/4 //The Vcc card is 3 Volts

16. power_down

Syntax : void power_down(void)

Summary : #include « tdalib.h »

Description : The power_down function

- assert Vcc card OFF
- active reset card
- enable the INT0 interrupt
- reset the ISO Uart

Return value : none

Register bank used : 0
Stack level used : 2

17. out_instr

Syntax : unsigned int in_instr(unsigned int **offset**)

Summary : #include « tdalib.h »

Description : The out_instr function send an outgoing instruction to the card in protocol T=0. offset contains the offset address of the Data exchange buffer where the first byte of the card instruction is stored (Data exchange buffer address + **offset**). The characters received from the card are stored at the Data exchange buffer address + **offse**t, the command is overwritten.

CLA	INS	P1	P2	P3

Return value :
Not equal to zero = number of characters receive from the card, ptr points -> characters received from the card.
0 = error, in this case the error type is sent back by the read_error() function

Error type : 0xC0 : card absent
0xC1 : I/O line locked at low level or no VCC
0x81 : Time out
0x83 : 3 parity errors in reception
0x84 : 3 parity errors in transmission

Register bank used : 0 and 1
Stack level used : 6

18. in_instr

Syntax : unsigned char in_instr(unsigned int **offset**)

Summary : #include « tdalib.h »

Description : The in_instr function send an incoming instruction to the card in protocol T=0. offset contains the **offset** address of the Data exchange buffer where the first byte of the card instruction is stored (Data exchange buffer address + **offset**).

SW1, SW2 are stored at the Data exchange buffer address + offset, the command is overwritten.

CLA	INS	P1	P2	P3	Data

Return value :2 = Successful

0 = error, in this case the error type is sent back by the read_error() function

Error type : 0xC0 : card absent
0xC1 : I/O line locked at low level or no VCC
0x81 : Time out
0x83 : 3 parity errors in reception
0x84 : 3 parity errors in transmission

Register bank used : 0 and 1
Stack level used : 6

19. XmitAPDU

Syntax : Unsigned int XmitAPDU(unsigned int **offset,** unsigned char **length**)

Summary : #include « tdalib.h »

Description : TheXmitAPDU function is used for the transportation of the APDU in T=0 protocol or T=1 protocol. **offset** contains the offset address of the Data exchange buffer (**512 bytes length max**) where the first byte APDU is stored (Data exchange buffer address + offset).**Length** is APDU length.

The characters received from the card are stored at the Data exchange buffer address + offset, the command is overwritten.

Return value : number of bytes returned by the card, offset -> received byte from the card in the data exchange buffer. The command is overwritten.

0 = error, in this case the error type is sent back by the read_error() function

Error type : 1: In T=0 or T= 1 protocol :
0xC0 : card absent
0xC1 : I/O line locked at low level or no VCC
0x81 : Time out
0x83 : 3 parity errors in reception
0x84 : 3 parity errors in transmission
0x20 : wrong APDU
0x21 : Too short APDU

2: In T=1 protocol :
0x22 : card mute during T=1 exchange
0x24 : bad NAD
0x25 : bad LRC
0x26 : Resynchronised
0x27 : Chain aborted
0x28 : bad PCB
0x29 : overflow from card(512 byte max)

Register bank used : 0 and 1
Stack level used : 8

20. read_ error

Syntax : unsigned char read _error(void)

Summary : #include « tdalib.h »

Description : This function return the error type when a function listed below return an error:

- power_up
- in_instr
-out_instr
-xmitAPDU
-negoce

Return value : Error Type(defined for each function)

Register bank used : 0
Stack level used : 1

21. read_IO_line

Syntax : unsigned char read_IO_line(void)

Summary : #include « tdalib.h »

Description :read_IO_line function returns the level of the card IO line.

Return value : 0 = IO line at 0 volt or no VCC
1 = IO line at 5 volts

Register bank used : 0
Stack level used : 2

22. negoce

Syntax :unsigned char negoce(unsigned int **ptr**, unsigned char **protocol**)

Summary : #include « tdalib.h »

Description : The negoce function executes the Protocol Type Selection PTS. **ptr** contains the offset address of the Data Exchange buffer, the negoce function use 6 bytes from this address. **protocol** = 0 or 1 to select the protocol T=0 or T=1. The FiDi given by the card during the Answer To Reset shall be used.

If the card is in negociable mode (TA2 is not present) and answers in its ATR a Fi/Di different from the default value (Fi/Di = 372) or proposes 2 different protocols (T=0 and T=1) a PTS command can be done by using the negoce command.

This negoce command generates automatically a PTS command by using the proposed Fi/Di parameters and the only parameter necessary to be given is the **protocol.**

Return value : 1 = Successful

0 = error, in this case the error type is sent back by the read_error() function

error type : 0x30 = no negociable mode (TA2 is present in the ATR)
0x31 = protocol is neither T=0 nor T=1
0x32 = T=1 is not accepted
0x33 = Answer from card does not match with the PTS command
0x34 = PCK error

Register bank used : 0 and 1
Stack level used : 6

23. idle_mode

Syntax : void idle_mode(void)

Summary : #include « tdalib.h »

Description : After execution the microcontroller is in idle mode. This instruction is the last one in normal mode, the program execution is stopped. During this mode the interrupt serial port is enabled. If an interrupt serial port is serviced, the microcontroller returns in normal mode and executes the instructions below idle_mode(), the interrupt serial port is now disabled. The library TDALIB.h includes the serial port interrupt routine. This routine does not read the received character from the serial port.

See **CE560** microcontroller specifications

Return value : None

Register bank used : 0
Stack level used : 1

24. read_clock_card

Summary : #include « tdalib.h »

Syntax : unsigned char read_clock_card(void)

Description : This function allows to know the clock card before execution of the clock_stop_low() or clock_stop_high() functions.

Return value : 0x02 = Xtal/2
0x04 = Xtal/4
0x06 = Xtal/8
0x08 = FINT/2

This value is usable by the set_clock_card function.

Register bank used : 0
Stack level used : 1

Example :

```
clock_stop_low() ;          // clock card = 0
clock = read_clock_card()   //clock = current clock card
clock_stop_low() ;          //clock card = low
set_clock_card(clock) ;     //restore clock card
```

25. protocol

Summary : #include « tdalib.h »

Syntax : unsigned char protocol(void)

Description : The protocol function allows to know which protocol is active in the card.

Return value : 0 The active protocol in the card is T = 0
1 The active protocol in the card is T =1

Register bank used : 0
Stack level used : 1

26. select_address

Syntax : void address_ram(unsigned int **address**)

Summary : #include « tdalib.h »

Description : The select_ram function selects the auxiliary RAM absolute **address** to be written or to be read by the write_ram function or the read_ram_function.

Return value : None

Register bank used : 0
Stack level used : 1

27. write_ram

Syntax : void write_ram(unsigned char **data**)

Summary : #include « tdalib.h »

Description : The write_ram function write a **data** in auxiliary RAM at the location selected by the select address function.

Note : After execution of the write_ram function or read_ram function, the Ram address register is automatically incremented. The Ram address points on the next character.

Return value : None

Register bank used : 0
Stack level used : 1

28. read_ram

Syntax : unsigned char read_ram(void)

Summary : #include « tdalib.h »

Description : The read_ram function read a data from the auxiliary RAM at the location selected by the select_address function..

Return value : Read data from the auxiliary RAM.

Note : After execution of the write_ram function or read_ram function, the Ram address register is automatically incremented. The Ram address points on the next character.

Register bank used : 0
Stack level used : 1

29. Error list

The error list gives the status code identification and a brief signification of the satus error code. The error code is returned by the read_error() function.

Status :	Meaning:
0x80	Card mute (after_power on)
0x81	Time out (waiting time exceeded)
0x82	3 parity errors on TS byte
0x83	3 parity errors in reception from the card
0x84	3 parity errors in transmission to the card
0x40	Card deactivated
0xC0	Card absent
0xC1	I/O line locked at low level (a deactivation ic processed)
0xC2	Protocol is neither T=0 nor T=1
0xC3	Checksum error
0xC4	TS is neither 0x3F nor 0x3B
0xC5	Bad Fi/Di
0xC6	ATR not supported(more than 32 interfece bytes[i>8])
0xC7	VPP not supported
0x30	No negociable mode (TA2 is present in the ATR)
0x31	Protocole is neither T=0 nor T=1 (negoce command)
0x32	T=1 is not accepted(negoce command)
0x33	PTS answer is different from the PTS request
0xA0	Procedure byte error
0x20	Wrong APDU
0x21	Too short APDU
0x22	Card mute now (during T=1 exchange)
0x24	Bad NAD
0x25	Bad LRC
0x26	Resynchronised
0x27	Chain aborted
0x28	Bad PCB
0x29	Overflow from card (512 bytes max)