APPLICATION NOTE

Library Reference for TDA8029

Release 1.1
ISO 7816 and EMV 3.1.1 compliant
C51 KEIL Compiler

AN01009

Philips Semiconductors

PHILIPS

Abstract

This document is the specification of the software library in 'C' language written for the TDA8029 to handle the communication between a TDA8029 and an asynchronous smart card.

APPLICATION NOTE

Library Reference for TDA8029

C51 KEIL Compiler

AN01009

Author(s):
Thierry LEJOSNE
Systems & Applications
Business Unit Identification – Business Line RIC
Caen - France

Keywords
TDA8029
Libraries
ISO 7816-3
E.M.V. 3.1.1

Number of pages : 44

Date : 2001/07/25

REVISION HISTORY

Version Number	Date	Description	Author
1.0	June 6th, 2001	• Initial release	T. Lejosne
1.1	July 25th, 2001	• The library is provided under 3 files : -TDA8029.lib -TDA8029Z.lib -TDA8029C.lib. • In EMV mode, the IFSD request in no more sent just after the ATR but just before the first APDU. • Addition of the activation of 1.8V cards • Example 'C' file upgraded	T. Lejosne

CONTENTS

1 INTRODUCTION

The TDA8029 is a complete one chip, low cost, low power, robust smart card reader.
Due to specific versatile hardware, a small embedded software allows the control of most cards available in the market.

This application note :

- describes all the functions available in the TDA8029 asynchronous library in order to handle a communication with asynchronous smart cards, either in T=0 or T=1 protocols,
- details the way to use the library,
- and finally gives an example in 'C' language to show calls to the different functions of the library.

2 ASYNCHRONOUS LIBRARY

The following functions are available in the asynchronous library :

check_pres_card()	Verifies if a card is inserted.
clock_stop()	Stops the clock card at the low or high level or set it to Fint/2 (~1.25MHz).
GetCardParam()	Returns the current parameters of the activated card (FiDi, clock card, protocol (T=0 or T=1)).
get_lib_version()	Returns the current version of the library
idle_mode()	Sets the microcontroller in idle mode and sets the card clock in the required state.
init_system()	Initializes the library.
Manual_Xmit_T1()	To be used for APDU transportation in T=1 protocol if the application layer provides the complete T=1 frame.
negotiate()	Assumes the PPS command.
power_down()	For deactivation of the smart card.
power_down_mode()	Sets the microcontroller in power down mode (card optionally deactivated or not).
power_up()	For activation of the smart card with analysis of the **A**nswer **T**o **R**eset.
power_up_iso()	For activation of the smart card with analysis of the **A**nswer **T**o **R**eset respecting ISO recommendations for the selection of the operating class.
read_alarm()	To be used to check if an alarm has occurred.
read_alarm_card_moved()	Gives the indication of a card movement.

read_bit()	Returns the value of bit(s) of an internal register of the UART.
read_card_active()	Returns the current state of the card (activated or not).
read_error()	Gives the error type when a function returns an error.
read_register()	Used for reading a byte from an UART register of the TDA8029.
send_Sblock_IFSD()	Negotiates IFSD with the card.
set_baud_rate()	Defines the ETU for the data exchange with the card.
set_clock_card()	Defines the clock frequency of the card.
SetNAD()	Defines the NAD parameter in T=1 protocol composed of the SAD and the DAD.
SetXTal()	To fixe the crystal frequency used in the application.
write_bit()	Sets or resets bit(s) of an UART register of the TDA8029 to a specified value.
write_register()	Used for writing a byte into an UART register of the TDA8029.
XmitAPDU()	For APDU transportation in T=0 and T=1 protocol. (ISO 7816-4).

3 ADVICES

3.1 Hardware considerations

This library is written for an P80C51RB+, which is the microcontroller core inside the TDA8029.
Pin P32 (INT0) is used to control pin INT of the ISO UART of the TDA8029.

3.2 ISO 7816 mode and EMV 3.11 mode

The choice between these two modes is made during the activation of the card (see power up function). At each activation it is possible to choose the mode.

For the EMV 3.11 mode, refer to the document :

_EMV 96 Integrated Circuit Card Specification For Payment Systems
Version 3.1.1 May 31, 1998

The asynchronous library has been validated according to the Europay document :

_ICC Terminal Type Approval: Test Bench Description Executable Tests August 27, 1998
+ Erratum TBD/EXE/T01 2.04, Erratum#3, November 29th 2000.

3.3 Compilation

The libraries are compiled in **small model**. This microcontroller inside the TDA8029 has **512 bytes of additional auxiliary RAM** addressable as an external memory.

3.4 Auxiliary RAM

The library 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 limited by the auxiliary RAM size. The exchange block length between the card reader and the card is limited at 512 bytes. The start address of the buffer is 0 in auxiliary RAM. It is possible to configure the size of the buffer.

3.5 Interrupts

Only the interrupt **INT0** is used by the library. Its interrupt service routine uses the **register bank 1.**
INT0 has a low priority level.
All the functions use the **register bank 0**.
The register bank 2 and the register bank 3 are available for the software application.

3.6 Timers

No internal timer of the microcontroller (timer 1, 2 or 3) is used by the libraries.
All the timing issues (Work Waiting Time, Extra Guard Time, Character Waiting Time, Block Waiting Time, Block Guard Time, Block Waiting Time extension) are fully supported by the specific timers of the ISO UART whatever the baud rate is on the I/O line.
Thus, the timers can be used for example to manage the baud rate or to survey a time-out communication on the serial line

3.7 System initialization

Before calling any function of the libraries, it is **mandatory to call the function init_system().**
init_system must be the **first instruction** of the application software. init_system() is described in *§4.6, page 17*.

Furthermore, if the external crystal or the external clock used for the TDA8029 has a frequency different of 14.745Mhz and when using TDA8029C.LIB, it is also **mandatory to call the SetXtal() function** described in *§4.23, page 26*.

3.8 Call to the functions

Any function must be called from **the register bank 0**.
The maximum stack level used by the library is 12. The minimum depth of the stack is 16 bytes.

3.9 Data exchange with the host

The libraries does not contain the software communication with the host. The software communication depends on the protocol used by the application.

3.10 Linker

In the command line of the link, one has to specify both the internal RAM size (256) and the start address of the indirect memory area (80h).

Command line example to generate the object code for the emulation:

```
MyAppli.obj, tda8029.lib
RAMSIZE(256)
IDATA(80H)
```

If the start address of indirect memory area is not specify, the indirect data can be located in direct memory area. It is mandatory to specify the RAM size.

3.11 Variables system

All variables used by the libraries are located in the 256 bytes of the internal RAM.

3.12 Internal RAM memory mapping and library code size

All variables used by the libraries are located in the 256 bytes of the internal RAM.
The amount of data used by the library slightly depend on which library is used.

Three different .LIB files are provided :

- **TDA8029.LIB** This is the normal version of the library
- **TDA8029Z.LIB** This is a reduced version of the normal version of the library witch may be used when the buffer used for card exchanges is located at address 0 in Xdata.
- **TDA8029C.LIB** This is the complete version of the library, including advanced functions.

The mappings shown hereafter can change according the release of the libraries. These mappings correspond to the library reference 1.1

3.12.1 TDA8029.LIB

This is a version in which the data passed to some of the library functions can be placed anywhere in the Xdata memory space using a ptr parameter.

```
TYPE     BASE       LENGTH     RELOCATION    SEGMENT NAME
-----------------------------------------------------
* * * * * * *   D A T A   M E M O R Y   * * * * * * *
REG      0000H      0008H      ABSOLUTE      "REG BANK 0"
REG      0008H      0008H      ABSOLUTE      "REG BANK 1"
DATA     0010H      0007H      UNIT          ?DT?EXAMPLE
DATA     0017H      0007H      UNIT          ?DT?GENLIB
DATA     001EH      0001H      UNIT          ?DT?_CLOCK_STOP?GENLIB
         001FH      0001H                    *** GAP ***
DATA     0020H      0001H      BIT_ADDR      ?BA?GENLIB
BIT      0021H.0    0003H.6    UNIT          ?BI?ASYNCLIB
BIT      0024H.6    0000H.7    UNIT          ?BI?GENLIB
BIT      0025H.5    0000H.3    UNIT          _BIT_GROUP_
BIT      0026H.0    0000H.1    UNIT          ?BI?WAITCOMMUTATION?GENLIB
         0026H.1    0000H.7                  *** GAP ***
DATA     0027H      0031H      UNIT          _DATA_GROUP_
DATA     0058H      0014H      UNIT          ?DT?ASYNCLIB
         006CH      0014H                    *** GAP ***
IDATA    0080H      0015H      UNIT          ?ID?ASYNCLIB
IDATA    0095H      0007H      UNIT          _IDATA_GROUP_
IDATA    009CH      0005H      UNIT          ?ID?GENLIB
IDATA    00A1H      0001H      UNIT          ?ID?EXAMPLE
IDATA    00A2H      0001H      UNIT          ?STACK
```

Direct memory area : 83 bytes are used by the library, thus 44 bytes are free for user application

Indirect memory area : 33 bytes are used by the library, 94 bytes are free for user.

Library code size : The total code size for the library reference 1.1 is about 9.7 Kbytes. This size can change according the release of the library.

3.12.2 TDA8029Z.LIB

The second version supposes that the data are always placed at @000 in the XData memory space. This saves memory space and code size.

```
TYPE     BASE       LENGTH     RELOCATION    SEGMENT NAME
-----------------------------------------------------
* * * * * * *   D A T A   M E M O R Y   * * * * * * *
REG      0000H      0008H      ABSOLUTE      "REG BANK 0"
REG      0008H      0008H      ABSOLUTE      "REG BANK 1"
DATA     0010H      0007H      UNIT          ?DT?EXAMPLE
DATA     0017H      0007H      UNIT          ?DT?GENLIB
DATA     001EH      0001H      UNIT          ?DT?_CLOCK_STOP?GENLIB
         001FH      0001H                    *** GAP ***
DATA     0020H      0001H      BIT_ADDR      ?BA?GENLIB
BIT      0021H.0    0003H.6    UNIT          ?BI?ASYNCLIB
BIT      0024H.6    0000H.7    UNIT          ?BI?GENLIB
BIT      0025H.5    0000H.3    UNIT          _BIT_GROUP_
BIT      0026H.0    0000H.1    UNIT          ?BI?WAITCOMMUTATION?GENLIB
         0026H.1    0000H.7                  *** GAP ***
DATA     0027H      002FH      UNIT          _DATA_GROUP_
DATA     0056H      0014H      UNIT          ?DT?ASYNCLIB
         006AH      0016H                    *** GAP ***
IDATA    0080H      0015H      UNIT          ?ID?ASYNCLIB
IDATA    0095H      0007H      UNIT          _IDATA_GROUP_
IDATA    009CH      0005H      UNIT          ?ID?GENLIB
IDATA    00A1H      0001H      UNIT          ?ID?EXAMPLE
IDATA    00A2H      0001H      UNIT          ?STACK
```

Direct memory area : 81 bytes are used by the library, thus 46 bytes are free for user application

Indirect memory area : 33 bytes are used by the library, 94 bytes are free for user.

Library code size : The total code size for the library reference 1.1 is about 9.4 Kbytes. This size can change according the release of the library.

3.12.3 TDA8029C.LIB

The third version includes advanced functions, not essentials most of the time. This version is the one that uses the biggest memory space and code size.

```
TYPE     BASE       LENGTH     RELOCATION    SEGMENT NAME
-----------------------------------------------------
* * * * * * *   D A T A   M E M O R Y   * * * * * * *
REG      0000H      0008H      ABSOLUTE      "REG BANK 0"
REG      0008H      0008H      ABSOLUTE      "REG BANK 1"
DATA     0010H      000BH      UNIT          ?DT?GENLIB
         001BH      0005H                    *** GAP ***
DATA     0020H      0001H      BIT_ADDR      ?BA?GENLIB
BIT      0021H.0    0003H.6    UNIT          ?BI?ASYNCLIB
BIT      0024H.6    0000H.7    UNIT          ?BI?GENLIB
BIT      0025H.5    0000H.3    UNIT          _BIT_GROUP_
DATA     0026H      0033H      UNIT          _DATA_GROUP_
DATA     0059H      0016H      UNIT          ?DT?ASYNCLIB
DATA     006FH      0007H      UNIT          ?DT?EXAMPLE
         0076H      000AH                    *** GAP ***
IDATA    0080H      0015H      UNIT          ?ID?ASYNCLIB
IDATA    0095H      0007H      UNIT          _IDATA_GROUP_
IDATA    009CH      0005H      UNIT          ?ID?GENLIB
IDATA    00A1H      0001H      UNIT          ?ID?EXAMPLE
IDATA    00A2H      0001H      UNIT          ?STACK
```

Direct memory area : 90 bytes are used by the library, thus 37 bytes are free for user application

Indirect memory area : 33 bytes are used by the library, 94 bytes are free for user.

Library code size : The total code size for the library reference 1.1 is about 10.8 Kbytes. This size can change according the release of the library.

4 ASYNCHRONOUS LIBRARY DETAILED DESCRIPTION

All the functions of the library are listed hereafter in alphabetical order.
In case of differences between the 3 libraries (TDA8029.LIB, TDA8029Z.LIB and TDA8029C.LIB), the two syntaxes used to call the function are described.

	TDA8029.LIB	TDA8029Z.LIB	TDA8029C.LIB	page
check_pres_card()				16
clock_stop()				16
GetCardParam()				16
get_lib_version()				17
idle_mode()				17
init_system()				17
Manual_Xmit_T1()				18
negotiate()				18
power_down()				19
power_down_mode()				19
power_up()				20
power_up_iso()				21
read_alarm()				22
read_alarm_card_moved()				23
read_bit()				23
read_card_active()				24
read_error()				24
read_register()				24
send_Sblock_IFSD()				25
set_baud_rate()				25
set_clock_card()				25
SetNAD()				26
SetXTal()				26
write_bit()				27
write_register()				27
XmitAPDU()				27
Rough size (v1.1)	**9.7 ko**	**9.4 ko**	**10.8 ko**	

	Function not available
	Standard function
	Buffer implicitly located at @0

4.1 check_pres_card

Syntax : bit check_pres_card(void)

Summary : #include “tdalib.h”

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

Return value : 1 = A card is present
0 = No card

Register bank used : 0	**Stack level used** : 8

4.2 clock_stop

Syntax : void clock_stop (unsigned char **clock_level**)

Summary : #include “tdalib.h”

Description : Depending on the **clock_level** parameter, the clock_stop function :

- sets the clock card at Fint/2 (~1.25MHz) (clock_level = 0)
- stops the clock card at high level (clock_level = 1)
- stops the clock card at low level (clock_level = 2)

Return value : None

Register bank used : 0	**Stack level used** : 6

4.3 GetCardParam

Syntaxes : unsigned int GetCardParam (unsigned int **ptr**) ***TDA8029C.LIB***

Summary : #include “tdalib.h”

Description : This function allows to get the current parameters of the card, if activated.

- the current FiDi is returned at address **ptr**
- the current clock card is returned at address **ptr**+1
  0 CLK = X_{TAL}
  2 CLK = X_{TAL} / 2
  4 CLK = X_{TAL} / 4
  6 CLK = X_{TAL} / 8
- the current protocol used is returned at address **ptr**+2
  0 protocol T=0
  1 protocol T=1.

Return value : 0 : the card is not activated
3 : the current parameters of the card are stocked in the buffer from address ptr.

Register bank used : 0	**Stack level used** : 8

4.4 get_lib_version

Syntax : unsigned char get_lib_version (void)

Summary : #include “tdalib.h”

Description : This function returns the current version of the library.

Return value : The current version of the library is composed of a version number and a revision number.
The **version** is coded in the high nibble of the returned value as the **revision** is coded in the low nibble.

b7	b6	b5	b4	b3	b2	b1	b0
Version				Revision			

Register bank used : 0	**Stack level used** : 2

4.5 idle_mode

Syntax : void idle_mode(unsigned char **clock_level**) ***TDA8029C.LIB***

Summary : #include “tdalib.h”

Description : The idle_mode function :

- set the card clock at the level specified in **clock_level**.
  - clock_level = 0, the clock is set to Fint/2 (**~**1.25MHz),
  - clock_level = 1, the clock is stopped at low level,
  - clock_level = 2, the clock is stopped in high level.
- then the microcontroller is put in idle mode. This instruction is the last one executed in normal mode before the program execution is stopped. During this mode, the external interrupt 0 (EX0) and the serial port interrupt are enabled. Thus, if any of theses event happens and are serviced, the microcontroller returns in normal mode and sets the clock card in the same state it was before the call of this function (see “example.c” in page 33 for further details).

Return value : none

cRegister bank used : 0	**Stack level used** : 8

4.6 init_system

Syntax : void init_system (unsigned int **size**)

Summary : #include “tdalib.h”

Description : The init_system function defines the **size** of the **data exchange buffer in auxiliary Ram**. The data Exchange buffer is used for data communication between the smart card and the card reader. Its length is limited to the auxiliary RAM size. This function initializes the variables system of the libraries. The maximum length of the data exchange buffer is 512 bytes.

It is mandatory to call this function at the beginning of the main application software.

The data exchange buffer is used by the following functions :

- XmitAPDU,
- Manual_Xmit_T1,
- power_up,
- power_up_iso,
- negotiate,
- Send_Sblock_IFSD,
- GetCardParam.

Return value : none

Register bank used : 0	**Stack level used** : 12

4.7 Manual_Xmit_T1()

Syntax : unsigned char Manual_Xmit_T1 (unsigned int **ptr**, unsigned int **DataLength**)

TDA8029C.LIB

Summary : #include “tdalib.h”

Description : This function may be used when for T=1 protocol exchanges when the application layer provides the complete T=1 frame including prologue, information and epilogue fields. The frame is passed to this function from address **ptr** and is assumed to be **DataLength** length.
This function will send this frame to the smart card without any verification and will return the length of the received frame from the card.
Of course, the timings of a block (Character Waiting Time and Block Waiting Time) and the Waiting Time Extensions requests (WTX) from the card will be handled by the TDA8029.

Return value : 0 : the function has failed
<>0 : number of received characters from the card.

Register bank used : 0	**Stack level used** : 10

4.8 negotiate

Syntax : unsigned char negotiate(unsigned char **protocol**, unsigned char **FiDi**)
unsigned char negotiate(unsigned int **ptr**, unsigned char **protocol**, unsigned char **FiDi**)

TDA8029Z.LIB
TDA8029.LIB
TDA8029C.LIB

Summary : #include “tdalib.h”

Description : The negotiate function executes the Protocol Parameters Selection **PPS**.
ptr contains the offset address of the Data Exchange buffer ; if TDA8029Z.LIB is used data have to be located from address @000.
The negotiate function uses 6 bytes from this address. **protocol** is used to select the protocol and **Fi/Di** is used to select the new Fi/Di parameters.
The coding of **protocol** is the following :

- 0 for T=0 protocol
- 1 for T=1 protocol

The coding of **FiDi** is the coding of the TA1 parameter (see ISO7816-3).

If the card is in negotiable 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 PPS command can be given by using the negotiate command.
This negotiate command automatically generates a PPS command by using the selected Fi/Di parameters and the selected protocol.

Return value : 1 = Successful
0 = Error, in this case the error type is sent back by the read_error() function,
(see §5 : Error list, page 29).

Register bank used : 0 and 1	**Stack level used** : 10

4.9 power_down

Syntax : void power_down(void)

Summary : #include “tdalib.h”

Description : The power_down function deactivates the card whatever its current state.

Return value : none

Register bank used : 0	**Stack level used** : 4

4.10 power_down_mode

Syntax : void power_down_mode (bit **deactive**, unsigned char **clock_level**) ***TDA8029C.LIB***

Summary : #include "tdalib.h"

Description : The power_down_mode function :

- deactivates the card if the parameter **deactive** is set,
  else the clock is stopped according to the **clock_level** parameter (same coding than for the clock_stop function) :
  - clock_level = 0, the clock is set to Fint/2 (**~**1.25MHz),
  - clock_level = 1, the clock is stopped at low level,
  - clock_level = 2, the clock is stopped in high level.
- then the microcontroller is put in power down. This instruction is the last one executed in normal mode before the program execution is stopped. During this mode, the consumption of the TDA8029 is minimized. Either a hardware reset or an external interrupt (INT0, INT1 or Rx) can be used to exit from this mode. ***Note** : the user has to build Interrupt Service Routine corresponding to INT1 or Rx (ISR attached to INT0 belongs to the library). See Example provided in page 33 for further details.*
- if the card has not been deactivated (deactivate = 0), then the clock is restored at the same value it was before the call of this function.

Return value : none

Register bank used : 0	**Stack level used** : 8

4.11 power_up

Syntax : unsigned char power_up (unsigned char **voltage**) ***TDA8029Z.LIB***
unsigned char power_up (unsigned int **ptr**, unsigned char **voltage**) ***TDA8029.LIB***
TDA8029C.LIB

Summary : #include "tdalib.h"

Description : The power_up function :

- switches the libraries in 7816 mode or in EMV 3.11 mode depending of the content of the Data Exchange Buffer + ptr (or @000 in case of TDA8029Z.LIB). If '0' the 7816 mode will be selected, if other value the EMV 3.1.1 mode will be selected. This memory location will be overwritten during the ATR,
- asserts VCC (**1.8, 3 or 5 volts** depending of the **voltage**) on the card.
  voltage = 1 VCC=1.8V ***TDA8029C.LIB***
  voltage = 3 VCC=3V,
  voltage = 5 VCC=5V.
- sets Fcard = F_{Xtal} /4
- sets ETU = Fcard/372 during the Answer To Reset
- sets UART prescaler = /31
- activates the card,
- stores 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 (in case of TDA8029Z.LIB, data are stored from @000).
- decodes the protocol T=0 or T=1,

- memorizes WI, CWI, BWI which are used to control the communications with the card
- initializes the Guard Time Register GTR
- in specific mode, automatically applies the protocol indicated in TA2 and uses Fi/Di parameters given in the Answer To Reset.

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, see §5 : Error list, page 29.

Example :

init_system (512);	The auxiliary RAM = 512 bytes length
status = power_up (0, 3);	The ATR will be stored at the first address of the Data Exchange Buffer. The Vcc card is 3 Volts.
or	
status = power_up (3);	Using TDA8029Z.LIB library.

Register bank used : 0 and 1	**Stack level used** : 12

4.12 power_up_iso

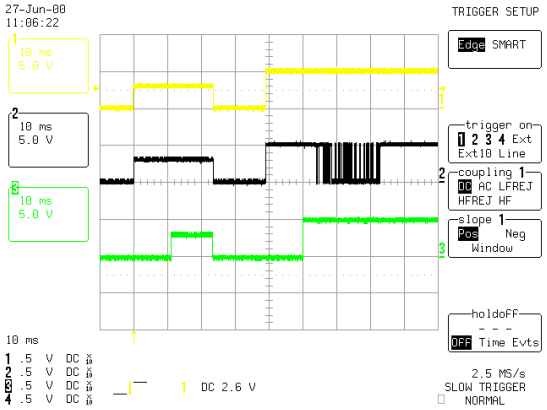
Syntax : unsigned char power_up_iso (unsigned int **ptr**) ***TDA8029C.LIB***

Summary : #include “tdalib.h”

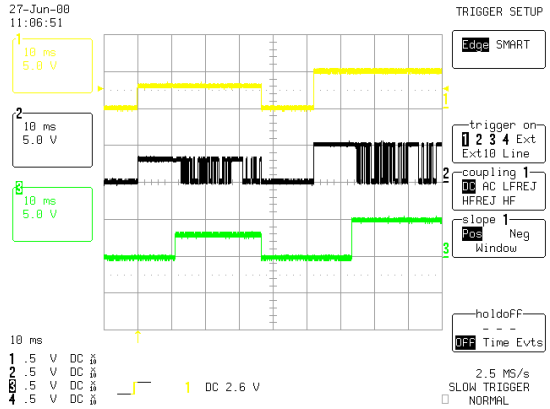
Description : The power_up_iso function :

- switches the libraries in 7816 mode,
- sets Fcard = F_{Xtal} /4,
- sets ETU = Fcard/372 during the Answer To Reset,
- sets UART prescaler = /31,
- asserts VCC 3 volts on the card and attempts to activate the card. If the card answers correctly and specifies that it is a class B or class AB card, then the session continues with VCC 3 volts,
- else asserts VCC 5 volts on the card and activates the card,
- stores 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,
- decodes the protocol T=0 or T=1,
- memorizes WI, CWI, BWI which are used to control the communications with the card,
- initializes the Guard Time Register GTR.

The following two figures show the power_up_iso sequence for two different cards.
Channel 1. VCC
Channel 2. I/O
Channel 3. RST



In this first case, the card does not answer under VCC=3 volts.
A second activation is done with VCC=5 volts.



In this second case, the card answers under VCC=3 volts but it does not specify its operating class.
So, a second activation is done with VCC=5 volts.

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, see §5 : Error list, page 29.

Register bank used : 0 and 1	**Stack level used** : 10

4.13 read_alarm

Syntax : bit read_alarm(void)

Summary : #include “tdalib.h”

Description : This function allows to check the current alarm state. If the returned value is different of zero, an alarm problem occurred; the alarm type can be obtained by calling read_error() function. Note that calling this function will erase the alarm marker.
It’s mandatory to use this function to check if an alarm has occurred.

Return value : This function returns :
0 when there is no alarm,
1 if an alarm problem has occured since the last call of this function

Example : Alarm is activated when : the card is inserted or removed, and when the TDA8029 has been deactivated due to a hardware problem (overcurrent on VCC, overheating, voltage drop on VDD).

```
MyAlarm = read_alarm();     // will return value 1
MyError = read_error();     // will give the alarm cause
MyAlarm = read_alarm();     // will return value 0, because current alarm
                            // has been erased by the first call of the function
```

Register bank used : 0	**Stack level used** : 2

4.14 read_alarm_card_moved

Syntax : bit read_alarm_card_moved(void)

Summary : #include “tdalib.h”

Description : This command returns a bit giving information on card movement. The application has to test if the card is inserted or removed after a debouncing delay by means of read_register(MSR) function.

Return value : 0 : no movement,
1 : movement detected

It is mandatory to call this function when read_alarm() function is different of zero and when the read_error() function returns zero because in this case the alarm must be due to card movement.

Register bank used : 0	**Stack level used** : 2

4.15 read_bit

Syntax : bit read_bit (unsigned char **reg**, unsigned char **mapping**)

Summary : #include “tdalib.h”

Description : This function gives the value of the bit(s) in the register located at **reg** address, following the bitmap given by **mapping**.

Return value : If all the bits specified in **mapping** are equal to one, then 1 is returned, 0 otherwise.

Example : Read bit PR1 in MSR register :
MSR register = 0x1C => suppose BGT bit and PR1 set
bit_value = Read_bit(0x0C, 0x04) gives bit_value = 1

Register bank used : 0	**Stack level used** : 4

4.16 read_card_active

Syntax : bit read_card_active (void)

Summary : #include “tdalib.h”

Description : This function returns 1 if the card is active.

Return value : 1 card active
0 card inactive

Register bank used : 0	**Stack level used** : 6

4.17 read_error

Syntax : unsigned char read_error(void)

Summary : #include “tdalib.h”

Description : This function returns the error type when the functions listed below return an error.

- power_up,
- power_up_iso,
- XmitAPDU,
- negotiate,
- send_Sblock_IFSD,
- set_clock_card.

This function must be called if the read_alarm() function returns 1.

Return value : Error Type (defined for each function, see §5 : Error list, page 29)

Register bank used : 0	**Stack level used** : 2

4.18 read_register

Syntax : unsigned char read_register(unsigned char pdata ***address**)

Summary : #include “tdalib.h”

Description : This function allows to read all readable registers of the UART of theTDA8029. The register is selected by the **address** parameter.
Address of the readable registers : one can use reserved words, see tdalib.h

Return value : read value.

Register bank used : 0	**Stack level used** : 2

4.19 send_Sblock_IFSD

Syntax : unsigned char send_Sblock_IFSD (unsigned char IFSD)

Summary : #include "tdalib.h"

Description : This function performs an IFSD negotiation with the card. An S(IFS)request is sent to the card with the value of the **IFSD** parameter given as an input.

Return value : 1 successful
0 IFSD value not supported. (read error function)

Register bank used : 0	**Stack level used** : 10

4.20 set_baud_rate

Syntax : void set_baud_rate(unsigned char **FiDi,** unsigned char **CKU**)

Summary : #include “tdalib.h”

Description : The set_baud_rate function programs the ETU of the ISO uart according the **FiDi** value (see ISO 7813-3) and the **CKU** parameter value. If **CKU** is set to 0, the baud rate is the one defined by **FiDi**; if **CKU** is set to one, the baud rate on I/O line is the double of the baud rate defined by **FiDi** (the ETU value is divided by 2).

Return value : 0 = FiDi not accepted
1 = FiDi accepted

Register bank used : 0	**Stack level used** : 10

4.21 set_clock_card

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

Summary : #include “tdalib.h”

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

Divider	clock card
0	F_{Xtal}
1	F_{Xtal} /2
2	F_{Xtal} /4
3	F_{Xtal} /8
4, 5, 6, 7	Fint/2(~1.25Mhz)

Return value : 1 = Successful
0 = Divider not supported or not compatible with Fi

Register bank used : 0	**Stack level used** : 6

4.22 SetNAD

Syntax : void SetNAD (unsigned char **DadSad**) ***TDA8029C.LIB***

Summary : #include “tdalib.h”

Description : The SetNAD function may be used to define the current Node address byte (NAD) used in T=1 protocol exchanges with **DadSad**.
If accepted, all the following exchanges in T=1 will use this value until it is changed again.
The default value for NAD is 0x00.

Return value : 0 : the DadSad parameter is refused
1 : the DadSad parameter is accepted

Register bank used : 0	**Stack level used** : 2

4.23 SetXtal

Syntax : bit SetXtal (float **DefXtal**) ***TDA8029C.LIB***

Summary : #include “tdalib.h”

Description : The SetXtal function defines the working clock frequency of the microcontroller (**DefXtal** in Mhz).

The default value for Xtal is supposed to be 14.745Mhz. It is mandatory to call this function at the beginning of the main application software if the used frequency is different of 14.745 Mhz.

Return value : none

Register bank used : 0	**Stack level used** : 2

4.24 write_bit

Syntax : void write_bit(unsigned char pdata ***address**, unsigned char **mask_bit**, bit **value**)

Summary : #include “tdalib.h”

Description : This function allows to write bit(s) of a readable registers of the UART of the TDA8029. The register is selected by the **address** parameter.
Address of the readable registers:reserved words, see tdalib.h
mask_bit contains the mapping of the bits of the register that are to be written. If value is 1, then all the bits specified in mask_bit of the register are set to one, 0 otherwise.

Return value : None

Register bank used : 0	**Stack level used** : 4

4.25 write_register

Syntax : void write_register(unsigned char pdata ***address**, unsigned char **data**)

Summary : #include “tdalib.h”

Description : This function writes a **data** into any writable register of the TDA8029. The register is selected by the **address** parameter.
Address of the writable registers: reserved words, see tdalib.h

Return value : None

Register bank used : 0	**Stack level used** : 2

4.26 XmitAPDU

Syntax : unsigned int XmitAPDU(unsigned int **length**) ***TDA8029Z.LIB***
unsigned int XmitAPDU(unsigned int **offset,** unsigned int **length**)
DA8029.LIB
TDA8029C.LIB

Summary : #include “tdalib.h”

Description : The XmitAPDU function is used for the transportation of the APDU in T=0 protocol or T=1 protocol according to **ISO 7816-3 and 7816-4**. **offset** contains the offset address of the data exchange buffer where the first byte APDU is stored (data exchange buffer address + offset, data exchange buffer address in case of TDA8029Z.LIB). **Length** is APDU length **(512 bytes max)**.
The characters received from the card are stored at the data exchange buffer address + offset, the command is overwritten.
If the selected mode is EMV 3.1.1 and if the protocol is T=1, in the case of the first exchange with the card following the ATR, then this function performs an automatic IFSD negotiation between the card and the reader (with 0xFE bytes).

Return value: Number of bytes returned by the card. They are available in the data exchange buffer from location **offset** or @000 (TDA8029Z.LIB). The command is overwritten.

This function returns 0 when an error occurred, in this case the error type can be obtained using the read_error() function (*see §5 : Error list, page 29*).

It returns 0xFFFF when the automatic IFSD negotiation has failed.

Register bank used : 0 and 1	**Stack level used** : 14

5 ERROR LIST

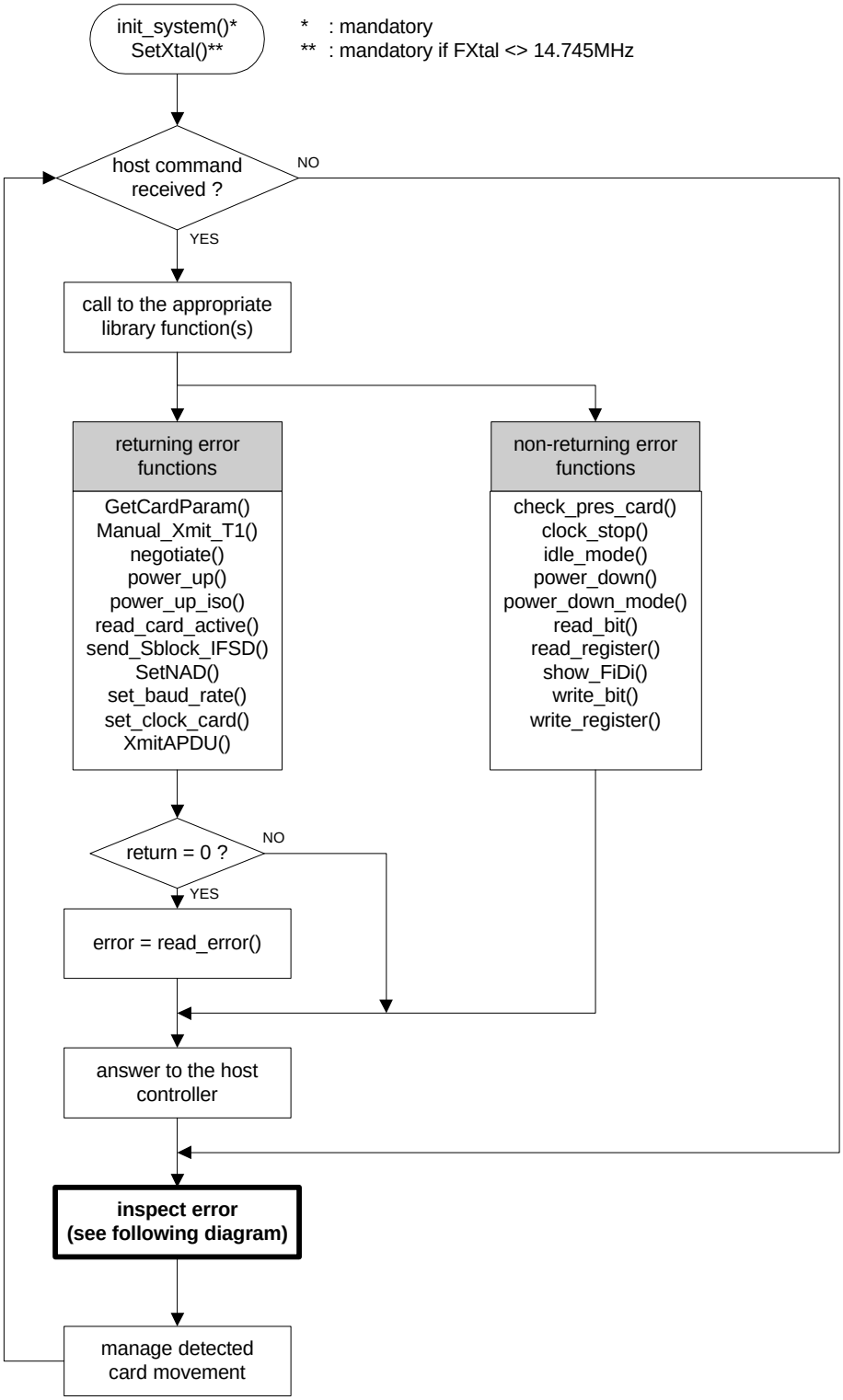
The following error list matrix gives for each status code identification a brief signification of the status error code and the name of function in which it is susceptible to occur. The error code is returned by the read_error() function. When this function returns 0, there is no error.

Code	Meaning	GetCardParam()	Manual_Xmit_T1()	negotiate()	power_up()	power_up_iso()	read_card_active()	send_Sblock_IFSD()	SetNAD()	set_baud_rate()	set_clock_card()	XmitAPDU()
00h	No error	X	X	X	X	X	X	X	X	X	X	X
08h	Length of the data buffer too short			X				X				X
0Ah	3 consecutive errors from the card in T=1 protocol							X				X
20h	Wrong APDU											X
21h	Too short APDU											X
22h	Card mute now (during T=1 exchange)											X
24h	Bad NAD								X			X
25h	Bad LRC											X
26h	Resynchronized											X
27h	Chain aborted											X
28h	Bad PCB											X
29h	Overflow from card (512 bytes max.)											X
30h	Non negotiable mode (TA2 is present)			X								
31h	Protocol is neither T=0 nor T=1 (negotiate command)			X								
32h	T=1 is not accepted (negotiate command)			X								
33h	PPS answer is different from PPS request			X								
34h	Error on TCK (negotiate command)			X								
35h	Error on parameter passed to a function			X								
38h	TB3 absent				X							
39h	PPS not accepted (no answer from the card)			X								
3Bh	Early answer of the card during the activation				X	X						
80h	Card mute (after power on)				X	X						
81h	Time out (waiting time exceeded)		X	X				X				X
83h	Too many parity errors in reception			X	X	X		X				X
84h	Too many parity errors in transmission			X				X				X
86h	Bad FiDi			X						X		
88h	ATR duration greater than 19200 etus (E.M.V.)				X							
89h	CWI not supported (E.M.V.)				X							
8Ah	BWI not supported (E.M.V.)				X							
8Bh	WI (Work waiting time) not supported (E.M.V.)				X							
8Ch	TC3 not accepted (E.M.V.)				X							
8Dh	Parity error during ATR				X	X						

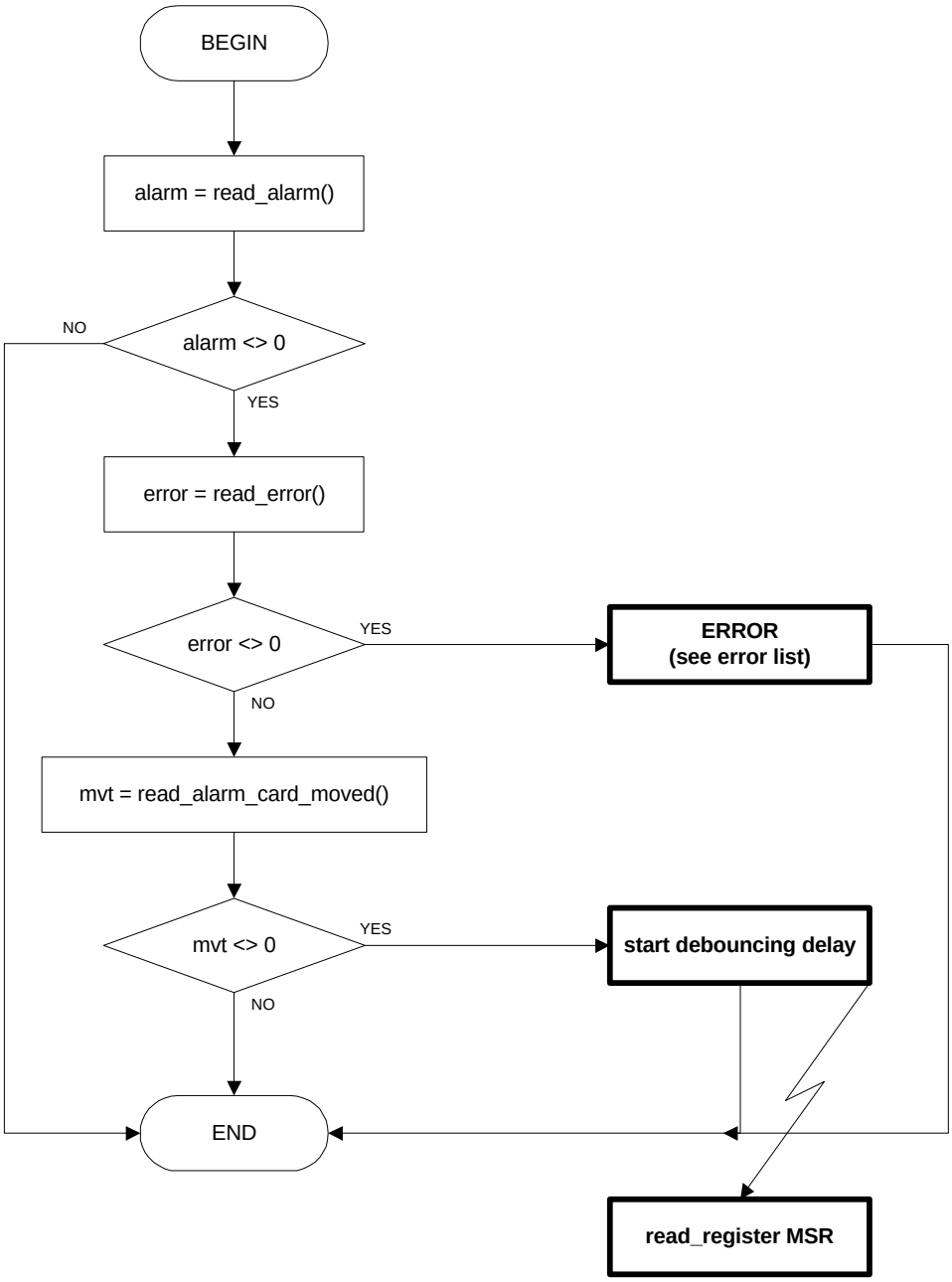
Code	Meaning	GetCardParam()	Manual_Xmit_T1()	negotiate()	power_up()	power_up_iso()	read_card_active()	send_Sblock_IFSD()	SetNAD()	set_baud_rate()	set_clock_card()	XmitAPDU()
90h	3 consecutive parity errors in T=1 protocol											X
91h	SW1 different from 6X or 9X											X
92h	Specific mode byte TA2 with b5 bit=1				X							
93h	TB1 absent during a cold reset (E.M.V.)				X							
94h	TB1different from 00 during a cold reset (E.M.V.)				X							
95h	IFSC<10H or IFSC=FFH				X							
96h	Wrong TDi				X	X						
97h	TB2 is present in the ATR (E.M.V.)				X							
98h	TC1 is not compatible with CWT				X							
9Bh	Not T=1 card							X				
A0h	Procedure byte error											X
A1h	Card deactivated	X	X	X			X	X				X
C0h	Card absent	(has to be checked at application layer)										
C3h	Checksum error				X	X						
C4h	TS is neither 3B nor 3F				X	X						
C6h	ATR not supported (more than 32 interface characters [i>8])				X	X						
C7h	VPP is not supported				X	X						
E0h	Bad number of card in select_card() function											
E1h	Bad clock card										X	
E2h	UART has received a character from the card whilst FIFO was full		X	X	X	X		X				X
E3h	Supply voltage drop-off		X	X	X	X		X				X
E4h	Temperature alarm		X	X	X	X		X				X
E9h	Framing error		X	X	X	X		X				X

6 PRINCIPLE OF UTILIZATION OF THE LIBRARY

The two following diagrams show the logical use of the library functions.



Alarm monitoring :



7 APPLICATION EXAMPLE

```
/*****************************************************************
*     ALPAR PROTOCOL USING THE TDA8029 LIBRARY V1.1              *
*           PHILIPS SEMICONDUCTORS                               *
*                                                                *
*  date   : July 2001                                            *
*                                                                *
*  Author : Thierry LEJOSNE                                      *
*****************************************************************/
// C51 file description
#include "reg51rd.h"

// Library file description
#include "tdalib.h"


/********** Prototypes  application**********************************/
void          sendchar_to_host(unsigned char);
unsigned char receive_host(unsigned int);
void          send_io_buff_host(unsigned int);
unsigned char rcv_host(void);
void          send_acknowledge(unsigned char);
void          send_error_message(unsigned char, unsigned char);
void          send_message_pres_card(void);
void          nb_bytes_response(unsigned int);
void          send_num_mask(void);
unsigned char send_header_to_host(unsigned char *);
void          send_response(unsigned int);
void          init_uc_uart(void);
/*********************************************************************/

/*********************************************************************/
char code message_pres_card[] = {0x60, 0x00, 0x01, 0xA0};
char code message_time_out[]  = {0xE0, 0x00, 0x00, 0xFF};
char code header_num_mask[]   = {0x60, 0x00,   13, 0x0A};
char code num_mask[]          = {"Lib 8029"};
/*********************************************************************/

/*********************************************************************/
unsigned char       lrc;
unsigned char       header_buff[6];
unsigned char idata error;
/*********************************************************************/

/*********************************************************************/
#define LRC_ERROR             0x01
#define TIME_OUT_SERIAL_LINK  0xFF
#define UNKNOWN_INS           0x55
#define CARD_ABSENT           0xC0
#define OK                       1
#define SMODE                 0x80
#define GATE_TIMER_1          0x80
#define CT_TIMER_1            0x40
#define M0_TIMER_1            0x10
#define M1_TIMER_1            0x20
/*********************************************************************/


//====================================================================//
void main(void)
//====================================================================//
{
   unsigned int  idata i;
   unsigned char idata ins;
#ifdef COMPLETE_VERSION
   unsigned char       Para1, Para2;
#endif
```

```
// Initialisation of the TDA8029 and host UART
// ===========================================
init_system(512);              // mandatory
#ifdef COMPLETE_VERSION
   SetXtal (14.745);           // f = 14.745 MHz
#endif
init_uc_uart();                // 38400 Bauds


// Working until the end...
// ========================
while (1)
{
   if (receive_host(0) != OK)
   {
      if (error != TIME_OUT_SERIAL_LINK)
      {
         error = read_error();

         if (error == 0)
         {
            if (read_alarm_card_moved())
            {
                                          send_message_pres_card();
            }
         }
         else
         {
            send_error_message(ins, error);
         }
      }
      else
      {
         send_error_message(ins, TIME_OUT_SERIAL_LINK);
         RI = TI = 0;
      }
      error = 0;
      continue;
   }

   if (lrc != 0)
   {
      send_error_message(ins, LRC_ERROR);
      continue;
   }


   // Dispatch the received command
   // =============================
   ins = header_buff[3];

   // check card presence when needed
   // ===============================
   if ((ins == 0x0C) || (ins == 0x10) || (ins == 0x69) || (ins == 0x6D) || (ins == 0x6E) ||
       (ins == 0x00) || (ins == 0x01) || (ins == 0xA6))
   {
      if (!check_pres_card())
      {
         send_error_message(ins, CARD_ABSENT);
         continue;
      }
   }

   switch (ins)
   {
      case 0x09 : // Check presence of the selected card
                  // ===================================
                  data_exch_buff[0] = check_pres_card();
                  nb_bytes_response(1);
                  send_response(1);
                  break;
```

```
case 0x0A : // Mask number
            // ===========
            send_num_mask();
            break;


case 0x0B : // Set card baud rate
            // ==================
            if (set_baud_rate(data_exch_buff[0], data_exch_buff[1]))
               send_acknowledge(ins);
            break;


case 0x0C : // Send an IFSD request to the card
            // ===============================
            if (send_Sblock_IFSD(data_exch_buff[0]))
               send_acknowledge(ins);
            else
               send_error_message(ins, read_error());
            break;


case 0x10 : // Manage a negociation with the card
            // ==================================
            #ifdef OFFSET_ZERO
               i = negotiate(data_exch_buff[0], data_exch_buff[1]);
            #else
               i = negotiate(10, data_exch_buff[0], data_exch_buff[1]);
            #endif

            if (i==0)
               send_error_message(ins, read_error());
            else
               send_acknowledge(ins);
            break;


case 0x11 : // Set the clock used for the card exchange
            // ========================================
            if (set_clock_card(data_exch_buff[0]) == !OK)
               send_error_message(ins, read_error());
            else
               send_acknowledge(ins);
            break;


case 0x6D : // Asynchronous card @ 3 volts
            // ===========================
            #ifdef OFFSET_ZERO
               i = power_up(V3);
            #else
               i = power_up(0, V3);
            #endif

            if (i == 0)
            {
               send_error_message(ins, read_error());
               break;
            }

            nb_bytes_response(i);
            send_response(i);
            break;


case 0x6E : // Asynchronous card @ 5 volts
            // ===========================
            #ifdef OFFSET_ZERO
               i = power_up(V5);
            #else
               i = power_up(0, V5);
            #endif
```

```
                    if (i == 0)
                    {
                       send_error_message(ins, read_error());
                       break;
                    }

                    nb_bytes_response(i);
                    send_response(i);
                    break;


         case 0x4D : // Power down
                     // ==========
                     power_down();
                     send_acknowledge(0x4D);
                     break;


         case 0x00 : // Card command
                     // ============
                     i  = header_buff[1]*256;
                     i += header_buff[2];

                     #ifdef OFFSET_ZERO
                        i = XmitAPDU(i);
                     #else
                        i = XmitAPDU(0, i);
                     #endif

                     if (i == 0)
                     {
                        send_error_message(ins, read_error());
                     }
                     // Test if succesful IFS REQUEST
                     // =============================
                     else if (i == 0xFFFF)
                     {
                        send_error_message(0x0C, read_error());
                     }
                     else
                     {
                        nb_bytes_response(i);
                        send_response(i);
                     }
                     break;


#ifdef COMPLETE_VERSION
         case 0x01 : // Card command : Manual T=1 APDU
                     // ==============================
                     i  = header_buff[1]*256;
                     i += header_buff[2];

                     i  = Manual_Xmit_T1(0, i);

                     if (i == 0)
                     {
                        send_error_message(ins, read_error());
                     }
                     else
                     {
                        nb_bytes_response(i);
                        send_response(i);
                     }
                     break;


         case 0x68 : // Asynchronous card @ 1.8 volts
                     // ============================
                     #ifdef OFFSET_ZERO
                        i = power_up(V1_8);
                     #else
                        i = power_up(0, V1_8);
```

```
            #endif

            if (i == 0)
            {
               send_error_message(ins, read_error());
               break;
            }

            nb_bytes_response(i);
            send_response(i);
            break;


case 0x69 : // Asynchronous card activation following ISO recommandations
            // ========================================================
            i = power_up_iso(0);

            if (i == 0)
            {
               send_error_message(ins, read_error());
               break;
            }

            nb_bytes_response(i);
            send_response(i);
            break;


case 0xA2 : // Idle Mode with clock card stopped low
            // ====================================
            // This example does not manage UART interrupt, it should
            // be implemented to properly leave IdleMode
            send_acknowledge(0xA2);
            idle_mode(0x01);
            break;


case 0xA3 : // Power down mode
            // ===============
            // External interrupt 0 is enabled in the library
            // External interrupt 1 is free for user
            Para1 = data_exch_buff[0];
            Para2 = data_exch_buff[1];
            send_acknowledge(0xA3);
            EX1 = 1;
            power_down_mode(Para1, Para2);
            break;


case 0xA4 : // Idle Mode with clock card stopped high
            // =====================================
            // This example does not manage UART interrupt, it should
            // be implemented to properly leave IdleMode
            send_acknowledge(0xA4);
            idle_mode(0x02);
            break;


case 0xA5 : // Set NAD (used in T=1 protocol)
            // ============================
            i = SetNAD(data_exch_buff[0]);

            if (i == 0)
            {
               send_error_message(ins, read_error());
            }
            else
            {
               nb_bytes_response(i);
               send_response(i);
            }
            break;
```

```
         case 0xA6 : // Get card parameters
                     // ===================
                     i = GetCardParam(0);

                     if (i == 0)
                     {
                        send_error_message(ins, read_error());
                     }
                     else
                     {
                        nb_bytes_response(3);
                        send_response(3);
                     }
                     break;
#endif


         default   : // Unknown command
                     // ===============
                     send_error_message(ins, UNKNOWN_INS);
                     break;
      }
   }
}


//=========================================================================//
void nb_bytes_response(unsigned int nb)
//=========================================================================//
{
   header_buff[0] = 0x60;
   header_buff[1] = (nb/256);
   header_buff[2] = (char)(nb);
}


//=========================================================================//
unsigned char receive_host(unsigned int ptr)
//=========================================================================//
{
   unsigned int data i;
   unsigned int data nb_char;

   lrc = 0;

   // Receive 4 bytes header
   // ======================
   for (i=0; i<4; i++)
   {
      while(rcv_host() == 0)
      {
         if (read_alarm() != 0)
            return(!OK);
         if (error != 0)
            return(!OK);
      }
      header_buff[i] = SBUF;
      lrc ^= header_buff[i];
   }

   nb_char  = header_buff[1]*256;
   nb_char += header_buff[2];
   nb_char++;

   // Receive Data
   // ============
   for (i=0; nb_char!=0; nb_char--, i++)
   {
      while (rcv_host() == 0)
      {
```

```
         if (read_alarm() != 0)
            return(!OK);
      }

      lrc ^= SBUF;

      if (nb_char != 1)
         data_exch_buff[ptr+i] = SBUF;
   }

   return(OK);
}


//=========================================================================//
void send_acknowledge(unsigned char ins)
//=========================================================================//
{
   data_exch_buff[0] = 0x60;
   data_exch_buff[1] = 0;
   data_exch_buff[2] = 0;
   data_exch_buff[3] = ins;
   send_io_buff_host(4);
}


//=========================================================================//
void send_error_message(unsigned char ins,unsigned char code_error)
//=========================================================================//
{
   data_exch_buff[0] = 0xE0;
   data_exch_buff[1] = 0;
   data_exch_buff[2] = 1;
   data_exch_buff[3] = ins;
   data_exch_buff[4] = code_error;
   send_io_buff_host(5);
}


//=========================================================================//
void send_io_buff_host(unsigned int nb)
//=========================================================================//
{
   unsigned int i;

   lrc = 0;
   for (i=0; nb!=0; i++, nb--)
   {
      sendchar_to_host(data_exch_buff[i]);
      lrc = data_exch_buff[i] ^ lrc;
   }
   sendchar_to_host(lrc);
   return;
}


//=========================================================================//
void send_response(unsigned int nb)
//=========================================================================//
{
   unsigned int i;

   lrc = 0;
   for (i=0; i<4; i++)
   {
      sendchar_to_host(header_buff[i]);
      lrc = header_buff[i] ^ lrc;
   }
```

```
   for (i=0; nb!=0; i++, nb--)
   {
      sendchar_to_host(data_exch_buff[i]);
      lrc = data_exch_buff[i] ^ lrc;
   }
   sendchar_to_host(lrc);
   return;
}


//=========================================================================//
unsigned char rcv_host(void)
//=========================================================================//
{
   if (RI == 0)
      return(0);

   RI = 0;
   return(1);
}


//=========================================================================//
void sendchar_to_host(unsigned char c)
//=========================================================================//
{
   SBUF = c;
   while (TI == 0);
   TI = 0;
}


//=========================================================================//
void send_num_mask(void)
//=========================================================================//
{
   unsigned char i;
   unsigned char checksum;

   checksum = send_header_to_host(&header_num_mask);

   for (i=0; i<8; i++)
   {
      sendchar_to_host(num_mask[i]);
      checksum = checksum ^ num_mask[i];
   }

   #ifdef COMPLETE_VERSION
      sendchar_to_host('C');
      checksum = checksum ^ 'C';
   #elif defined (OFFSET_ZERO)
      sendchar_to_host('Z');
      checksum = checksum ^ 'Z';
   #else
      sendchar_to_host(' ');
      checksum = checksum ^ ' ';
   #endif

   sendchar_to_host(' ');
   checksum = checksum ^ ' ';

   i = get_lib_version();
   sendchar_to_host(0x30 + ((i&0xF0)>>4));
   checksum = checksum ^ (0x30 + ((i&0xF0)>>4));

   sendchar_to_host('.');
   checksum = checksum ^ '.';

   sendchar_to_host(0x30 + (i&0x0F));
   checksum = checksum ^ (0x30 + (i&0x0F));
```

```
   sendchar_to_host(checksum);
}


//=========================================================================//
void init_uc_uart(void)
//=========================================================================//
{
   PCON = PCON | SMODE;
   TMOD = TMOD & ~GATE_TIMER_1 & ~CT_TIMER_1 & ~M0_TIMER_1 | M1_TIMER_1;
   TH1  = 254;
   TL1  = 254;
   TR1  = 1;
   SM0  = 0;
   SM1  = 1;                // UART mode 1
   SM2  = 0;
   REN  = 1;
}


//=========================================================================//
void send_message_pres_card(void)
//=========================================================================//
{
   unsigned char lrc;
   unsigned char i;

   lrc = send_header_to_host(&message_pres_card);
   i   = check_pres_card();
   sendchar_to_host(i);
   lrc = lrc ^ i;
   sendchar_to_host(lrc);
}


//=========================================================================//
unsigned char send_header_to_host(unsigned char *ptr_header)
//=========================================================================//
{
   unsigned char lrc;
   unsigned char i;

   lrc = 0;
   for (i=0; i<4; i++)
   {
      sendchar_to_host(*(ptr_header+i));
      lrc = lrc ^ *(ptr_header+i);
   }
   return(lrc);
}


#ifdef COMPLETE_VERSION
//=========================================================================//
int_TDA8029_INT1() interrupt 2 using 1
//=========================================================================//
{
   // This interrupt is only used to wake-up the device from Power Down mode
   // ====================================================================
   // (Specific code may be added here)
}


//=========================================================================//
void UAR_int_uart(void) interrupt 4 using 1
//=========================================================================//
{
   // This interrupt is only used to wake-up the device from Idle mode
```

```
   // ================================================================
   // (Specific code have to be added here to allow a good restart)
   RI = 0;
   TI = 0;
   ES = 0;
   error = TIME_OUT_SERIAL_LINK;
}
#endif
```

This example can be compiled and linked by :

```
C:\> C51  Example.c
C:\> L51  @Example.lin
```

With EXAMPLE.LIN file :

```
C:\TDA8029\Example\Example.OBJ,
C:\TDA8029\Example\TDA8029.LIB
to C:\TDA8029\Example\Example.ABS idata(80h) pdata(00) ramsize(256) ixref
```

To use for example the complete version of the library :

```
C:\> C51  Example.c DF(COMPLETE_VERSION)
C:\> L51  @ExampleC.lin
```

With EXAMPLEC.LIN file :

```
C:\TDA8029\Example\Example.OBJ,
C:\TDA8029\Example\TDA8029C.LIB
to C:\TDA8029\Example\Example.ABS idata(80h) pdata(00) ramsize(256) ixref
```

8 ANSWER TO RESET : CARDS ACCEPTED WITH THE LIBRARY

8.1 EMV mode

Character	Basic EMV value	Remarks	Values accepted
TS	3B or 3F	Direct or inverse convention	3B or 3F
T0	T=0 : 6X T=1 : EX	TB1 and TC1 present TB1, TC1 and TD1 present X : number of historical bytes	M_1X M_1 0 to F (mapping of TA1 to TD1) X 0 to F (number of historical bytes)
TA1	absent	FI / DI	Any value accepted by the TDA8029 01, 02, 03, 04, 08 11, 12, 13, 14, 18 21, 22, 23, 28 31, 32, 33, 34, 35, 38 41, 42, 43, 44, 48 51, 52, 53, 54, 55, 56, 58 61, 62, 63, 64, 68, 69 91, 92, 93, 94, 95, 96 A1, A2, A3, A4, A5, A6, A8 B1, B2, B3, B4, B5, B6 C1, C2, C3, C4, C5, C6, C8 D1, D2, D3, D4, D5, D6
TB1	00	VPP not required	00 in response to a cold reset Any value in response to a warm reset
TC1	00 to FF	Extra guard time	00 to FF
TD1	T=0 : absent T=1 : 81	Protocol type	$M_2$0 or $M_2$1 M_2 mapping of TA2 to TD2 0 or 1 protocol type (T=0 or T=1)
TA2	absent	Specific mode byte	
TB2	absent	Programming voltage	none ➔ ATR rejected
TC2	absent	WI in T=0	Any value from 01 to 0A
TD2	T=0 : absent T=1 : 31	Protocol type	$M_3$1 or M_3E M_3 mapping of TA3 to TD3 1 or E protocol type (T=1 or T=14) E accepted if l.s nibble of TD1=0
TA3	T=0 : absent T=1 : 10 to FE	IFSC of the card	Any value from 10 to FE
TB3	T=0 : absent T=1 : BC	B : BWI and C : CWI	BWI : 0 to 4 CWI : 0 to 5 with 2^{CWI} > (TC1 + 1)
TC3	T=0 : absent T=1 : absent	CRC or LRC	00
TH_i	absent	Historical bytes	Any values
TCK	T=0 : absent other cases : mandatory	Check character	TCK present and correct value

8.2 ISO7816 mode

Character	Basic value	Remarks	Values accepted
TS	3B or 3F	Direct or inverse convention	3B or 3F
T0	NX	N : mapping of TA1 to TD1 X : number of historical bytes	M_1X M_1 0 to F X 0 to F
TA1	YY	FI / DI	Any value accepted by the TDA8029 01, 02, 03, 04, 08 11, 12, 13, 14, 18 21, 22, 23, 28 31, 32, 33, 34, 35, 38 41, 42, 43, 44, 48 51, 52, 53, 54, 55, 56, 58 61, 62, 63, 64, 68, 69 91, 92, 93, 94, 95, 96 A1, A2, A3, A4, A5, A6, A8 B1, B2, B3, B4, B5, B6 C1, C2, C3, C4, C5, C6, C8 D1, D2, D3, D4, D5, D6
TB1	ZZ	Programming current and voltage	Any value
TC1	00 to FF	Extra guard time	00 to FF
TD1	T=0 : $M_2$0 T=1 : $M_2$1	Protocol type	$M_2$0 or $M_2$1 M_2 mapping of TA2 to TD2 0 or 1 protocol type (T=0 or T=1)
TA2		Specific mode byte	
TB2		Programming voltage	50
TC2	XX	WI in T=0	Any value from 00 to FF
TD2	T=0 : $M_3$0 T=1 : $M_3$1	Protocol type	$M_3$1 or M_3E M_3 mapping of TA3 to TD3 0, 1, E or F protocol type 0 accepted if l.s nibble of TD1=0
TA3	T=0 : absent T=1 : 01 to FE	IFSC of the card	Any value from 01 to FE
TB3	T=0 : absent T=1 : BC	B : BWI and C : CWI	BWI : 0 to 9 CWI : 0 to 15
TC3	T=0 : absent T=1 : absent	CRC or LRC	Any value
TH_i	absent	Historical bytes	Any values
TCK	T=0 : absent other cases : mandatory	Check character	TCK present and correct value