APPLICATION NOTE

Serial PC/SC Smart Card Reader Application with TDA8029

AN01032

Philips
Semiconductors

PHILIPS

Abstract

This application note describes a serial smart card reader for PC environment based on specifications established by the Personal Computer Smart Card (PC/SC) Workgroup.

This is a typical example of a smart card reader for electronic commerce and network security applications, based on the smart card coupler I.C. TDA8029.

The application note describes the software implemented in TDA8029 that handles a communication between a system controller and a smart card. Asynchronous smart cards using either T=0 or T=1 protocol are supported. It also presents briefly the use of this reader in the Windows environment.

APPLICATION NOTE

Serial PC/SC Smart Card Reader Application with TDA8029

AN01032

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

Keywords
TDA8029
Smart card interface
ISO 7816-3 & 4
PC/SC

Number of pages : 51

Date : 01/07/23

CONTENTS

1 INTRODUCTION

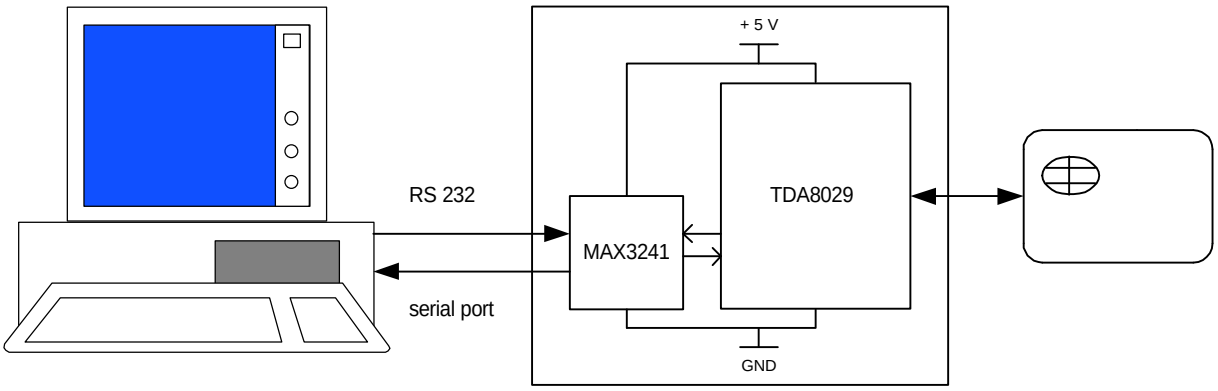
TDA8029 is a smart card coupler providing all the analogue electrical interface signals to the smart card. This coupler is able to manage asynchronous cards due to its specific ISO7816 UART and to its embedded 80C51 microcontroller core; it can also manage synchronous cards such as I2C cards or prepaid telephone cards.

The software embedded in this device is able to support any ISO7816 asynchronous smart card (T=0 or T=1 protocol) and completely handles the communication layer between the card and the host system.

A specific protocol called «ALPAR» has been defined on the serial interface between TDA8029 and the host system; it uses the APDUs frame types to convey the asynchronous card commands.

A demoboard (CAKE 8029_04-T) has been built in order to demonstrate a communication between a smart card and a host system which here is a PC. This demoboard is driven from the PC by means of a Windows device driver and an application named «SCManager».

The following diagram illustrates this application.



This demo board supports the Plug and Play functionality under the Microsoft Windows environment.

The PC/SC Workgroup is formed in partnership with major PC and smart card companies. Its main focus is to facilitate the development of smart card based applications for the PC by developing open specifications that ensure interoperability among cards, readers and PCs. The specifications are based on ISO 7816 standards and version 1.0 is published at http://www.pcscworkgroup.com/.

Microsoft has implemented the PC/SC 1.0 specifications for the 32-bit Windows platforms.

Microsoft has also created a Windows-compatible Logo Program for smart card readers to test that the reader is conforming to the PC99 hardware design requirements and to Microsoft's implementation of the PC/SC Workgroup 1.0 specifications. The smart card reader test kit can be downloaded from the Windows Hardware Quality Lab (WHQL) Web site at http://www.microsoft.com/hwtest.

2 HARDWARE ASPECT

The board CAKE 8029_04-T is made with a TDA8029 in a specific LQFP68 package combined with an external PROM or EPROM (see **Appendix: Hardware Information**, page 44). This specific package is only used for development purpose as this mask is not available in a TDA8029 romed version.
This board is supplied under +5V and is connected to the PC by means of the serial port at 38 400 bauds.

3 SOFTWARE ASPECT

This mask has been developed in order to be used in ISO7816-3 and PC/SC environments.

The demoboard CAKE 8029_04-T has been realised in order to be compliant with the HCT Smart Card Reader Self-Test Procedures for Microsoft Windows operating systems from Microsoft Windows Hardware Quality Labs.

4 SERIAL RS232 INTERFACE

4.1 General description

The serial interface between the TDA8029 and the host controller is a full duplex interface using the two lines RX and TX.
RX (pin 32) is used to receive data from the host controller, TX (pin 31) is used to send data to the host controller.

Serial data format

1	start bit
8	data bits
1	stop bit, no parity

Baud rate

38 400 bauds

A security feature has been implemented on the TDA8029 receiving procedure in order to avoid any blocking of the serial interface.

5 PLUG AND PLAY DETAILS

Three additional lines are needed for the plug and play mechanism:

- DTR (PC ready) from the PC
- RTS (Request to Send) from the PC
- DSR (COM device ready) from demo board CAKE 8029_04-T

The essential elements of plug-and-play serial devices are:

- attachment detection of serial devices,
- device identification, and operating system notification of its arrival,
- detachment detection of serial devices, and operating system notification of its removal.

The Serial Device ID String returned by the demo board is coded as follows:

Field Name	CAKE 8029_04-T
Other ID	
Begin PnP	28h
PnP Rev	01h 24h
EISA ID	PSC
Product ID	8029
Serial Number	\
Class ID	\SmartCardReader
Driver ID	\
User Name	\Philips SC Serial SmartCard Reader
Another ID	\
Checksum	39h 36h
End PnP	29h

The whole detection method is described in “Plug and Play External COM Device Specification, Version 1.00” and in “PC99 Smart Card Reader Self-Test” documents from Microsoft.
These two documents can be downloaded from http://www.microsoft.com/hwdev/respec/pnpspecs.htm and http://www.microsoft.com/hwtest/testkits.

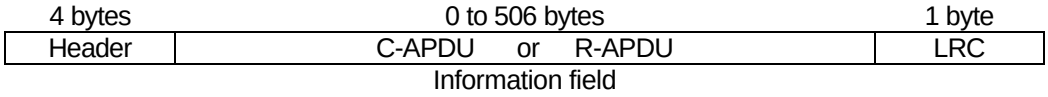
6 PROTOCOL ALPAR

The communication between the host controller and the TDA8029 uses a protocol named ALPAR.
This protocol encapsulates the useful data of a message in an invariant frame structure and defines a dialog structure of messages exchanges.

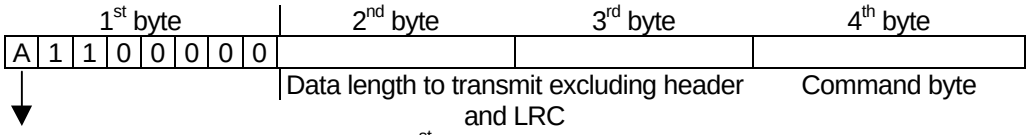
Frame structure :

Data is exchanged between the host controller and TDA8029 in blocks, each block made up of binary characters on one byte:

4 header characters
0 to 506 data characters (C-APDU or R-APDU)
1 LRC character



The 4 header bytes includes:



A=0 ➜ Acknowledge of the frame (1st byte = 60)
A=1 ➜ Nack of the frame (message with a status error, 1st byte = E0)

LRC byte:

The LRC (Longitudinal Redundancy Check) byte is such that the exclusive-oring of all bytes including LRC is null.

6.1 General dialog structure

The host controller is the master for the transmission; each command from the master is followed by an answer from TDA8029 including the same command byte as the input command.

However, in some cases (card insertion or extraction, a time out detection on Rx line or an automatic emergency deactivation of the card) the TDA8029 is able to initiate an exchange.

6.2 Successful command

System to TDA8029

60	XX XX	YY	nnnnnnnnnnnnnnnnnnn	ZZ
ACK	length	code	Data (C-APDU)	LRC

TDA8029 to System

60	UU UU	YY	mmmmmmmmmmmmmm	TT
ACK	length	code	Data (R-APDU)	LRC

The same command byte YY is returned in the answer from TDA8029.

6.3 Unsuccessful command

System to TDA8029

60	XX XX	YY	nnnnnnnnnnnnnnnnnnn	ZZ
ACK	length	code	Data (C-APDU)	LRC

TDA8029 to System

E0	UU UU	YY	SS	TT
NACK	length	code	status	LRC

The status contains the error code information (see **error list**, page 16).

6.4 Answer with an acknowledge (power_off, …)

System to TDA8029 (example : power_off)

60	00 00	4D	2D
ACK	Length	code	LRC

TDA8029 to System

60	00 00	4D	2D
ACK	Length	code	LRC

When the answer is an acknowledge of the command, the TDA8029 sends back a frame with the same content of the command.

6.5 Card insertion / Card removal

A card insertion will generate the following informational frame:

TDA8029 to System

60	00 01	A0	01	C0
ACK	Length	code	data	LRC

whereas in the case of a card extraction, the frame will be:

TDA8029 to System

60	00 01	A0	00	C1
ACK	Length	code	data	LRC

7 COMMAND BYTES

The following command bytes are available (listed in numerical order):

Command	***Code***	***Answer from reader***	***(page)***
card_command (APDU)	00_H	Card response (APDU) or error message	*(21)*
check_pres_card	09_H	Card presence information	*(19)*
send_num_mask	$0A_H$	1 parameter giving the mask number	*(19)*
set_card_baud_rate	$0B_H$	Acknowledge	*(22)*
IFSD_request	$0C_H$	Acknowledge or error message	*(22)*
negotiate (PPS)	10_H	Acknowledge or error message	*(21)*
set_clock_card	11_H	Acknowledge or error message	*(22)*
power_off	$4D_H$	Acknowledge	*(21)*
power_up_1V8	68_H	ATR from the card or error message	*(20)*
power_up_3V	$6D_H$	ATR from the card or error message	*(21)*
power_up_5V	$6E_H$	ATR from the card or error message	*(21)*
idle_mode	$A9_H$	Acknowledge	*(19)*
set_NAD	$A5_H$	Acknowledge or error message	*(24)*

Outgoing commands (only):

	Code	*Parameter*
Card_take_off	$A0_H$	00_H
Card_insertion	$A0_H$	01_H

These commands are sent as soon as a card is inserted or extracted without any command coming from the system. These commands use the same operating code but the extra parameter gives the additional information.

These outgoing commands are sent only when the host is waiting for a reply or is in stand by; when the card is extracted whereas the host is sending a frame to TDA8029, the card_take_off message will be sent from TDA8029 only when it has received the complete frame coming from the host controller. This system prevents any conflict on the serial line.

	Code	*Parameter*
Card deactivated	XX_H	$A1_H$ The card is deactivated due to a hardware problem (short circuit on Vcc, overcurrent)

Time out	XX_H	FF_H Time out problem on (TDA8029) Rx line This command is used in order to warn the host controller that the last communication has broken down (time out problem) so that the Rx line of TDA8029 does not remain blocked. The time out condition is a silence greater than 10 ms in the host command frame.

In these two commands, the code value is the previous code value used during a normal exchange.

8 ERROR LIST

The error list gives the status code identification and a brief explanation of the status error code.

8.1 Exhaustive list of possible error code

Status code	*Meaning*
08_H	Length of the data buffer too short
$0A_H$	3 consecutive errors from the card in T=1 protocol
20_H	Wrong APDU
21_H	Too short APDU
22_H	Card mute now (during T=1 exchange)
24_H	Bad NAD
25_H	Bad LRC
26_H	Resynchronized
27_H	Chain aborted
28_H	Bad PCB
29_H	Overflow from card
30_H	Non negotiable mode (TA2 present)
31_H	Protocol is neither T=0 nor T=1 (negotiate command)
32_H	T=1 is not accepted (negotiate command)
33_H	PPS answer is different from PPS request
34_H	Error on PCK (negotiate command)
35_H	Bad parameter in command
38_H	TB3 absent
39_H	PPS not accepted (no answer from card)
$3B_H$	Early answer of the card during the activation
55_H	Unknown command
80_H	Card mute (after power on)
81_H	Time out (waiting time exceeded)
83_H	5 parity errors in reception
84_H	5 parity errors in transmission
86_H	Bad FiDi
88_H	ATR duration greater than 19200 etus (E.M.V.)
89_H	CWI not supported (E.M.V.)
$8A_H$	BWI not supported (E.M.V.)
$8B_H$	WI (Work waiting time) not supported (E.M.V.)
$8C_H$	TC3 not accepted (E.M.V.)
$8D_H$	Parity error during ATR
90_H	3 consecutive parity errors in T=1 protocol
91_H	SW1 different from 6X or 9X
92_H	Specific mode byte TA2 with b5 byte=1
93_H	TB1 absent during a cold reset (E.M.V.)
94_H	TB1different from 00 during a cold reset (E.M.V.)
95_H	IFSC<10H or IFSC=FFH

96_H	Wrong TDi
97_H	TB2 is present in the ATR (E.M.V.)
98_H	TC1 is not compatible with CWT
$9B_H$	Not T=1 card
$A0_H$	Procedure byte error
$A1_H$	Card deactivated due to a hardware problem
$C0_H$	Card absent
$C3_H$	Checksum error
$C4_H$	TS is neither 3B nor 3F
$C6_H$	ATR not supported
$C7_H$	VPP is not supported
$E1_H$	Card clock frequency not accepted (after a set_clock_card command)
$E2_H$	UART overflow
$E3_H$	Supply voltage drop-off
$E4_H$	Temperature alarm
$E9_H$	Framing error
$F0_H$	Serial LRC error
FF_H	Serial time out

8.2 Error code for each command

COMMAND		POSSIBLE RETURNED ERROR CODE
Power up 1V8, 3V, 5V		31h, 35h, 38h, 3Bh, 80h, 85h, 86h, 88h, 89h, 8Ah, 8Bh, 8Ch, 8Dh, 92h, 93h, 94h, 95h, 96h, 97h, 98h, C0h, C3h, C4h, C6h, C7h, E2h, E3h, E4h, E9h, F0h, FFh
Card command	T=0	08h, 20h, 21h, A1h, 81h 83h, 84h, 91h, A0h, C0h, E2h, E3h, E4h, E9h, F0h, FFh
	T=1	08h, 22h, 24h, 25h, 26h, 27h, 28h, 29h, A1h, 83h, 90h, C0h, E2h, E3h, E4h, E9h, F0h, FFh
Negotiate		30h, 31h, 33h, 34h, 35h, 39h, A1h, C0h, E2h, E3h, E4h, E9h, F0h, FFh
Set clock card		C0h, E1h, F0h, FFh
Set card baud rate		86h, C0h, F0h, FFh
Set NAD		24h, F0h, FFh
IFSD request		0Ah, A1h, 9Bh, C0h, E2h, E3h, E4h, E9h, F0h, FFh
Send mask number		F0h, FFh
Check presence card		F0h, FFh
Power off		F0h, FFh
Idle mode		F0h, FFh

9 COMMANDS DESCRIPTION

9.1 General commands

9.1.1 Send_num_mask

This command is used to identify the software version which is masked in TDA8029 ROM.

For example the current software will be coded as : “8029 PNP 1.0” (12 ASCII characters)
System to TDA8029 : 60 00 00 0A 6A
TDA8029 to System : 60 00 0C 0A 38 30 32 39 20 50 4E 50 20 31 2E 30 04

9.1.2 Check_presence_card

This command is used to check the card presence.

System to TDA8029 : 60 00 00 09 69
TDA8029 to System : 60 00 01 09 PRES LRC

Where PRES indicates the presence of a card (00_H if there is no card, 01_H if a card is present).

9.1.3 Idle_mode

This command is used to set the microcontroller in idle mode. The card, if activated, has its clock (CLK) set to low or high level or switched to Fint/2 but is still active.

Waking up conditions :

- any command from the host on the serial line. In that case, the command is normally executed by the TDA8029 and the corresponding answer is sent to the host.
- any hardware event on the card side. In that case, an outgoing command is sent to the host to warn it of the event occurred.
- a hardware reset.

System to TDA8029 : 60 00 00 A9 CLK LRC
TDA8029 to System : 60 00 00 A9 C9

Where CLK is the clock to be used

CLK = 0x00	clock is stopped at low level
CLK = 0x01	clock is stopped at high level
CLK = 0x02	clock is switched to Fint/2

9.1.4 Time_out

This command is sent from TDA8029 to the host controller if, during a transmission from the host controller to TDA8029, the time interval between 2 characters exceeds 10ms. This timing is calculated between each character of a frame, starts after the first character, and is disabled after the last character of the frame. This feature has been implemented in order to avoid any blocking of the transmission line between the host controller and TDA8029.

TDA8029 to System : E0 00 01 6F FF 71

9.1.5 Card_take_off and card_insertion

These two commands are sent directly to the system processor as soon as a card extraction or insertion has occurred.

TDA8029 to System :	60	00 01	A0	00	C1	for a card extraction
	60	00 01	A0	01	C0	for a card insertion.

9.2 Asynchronous card related commands

9.2.1 power_up commands

There are two different power up commands (Power_up_3V and Power_up_5V). These commands have to be followed by the parameter $00_{H,}$ indicating that all the ATR card parameters compliant with ISO7816-3 will be taken into account.

9.2.1.1 Power_up_5V

This command allows to activate the card at a VCC of 5V. All the signals going to the card will be referenced to this VCC=5V.
An activation sequence is processed following the ISO7816-3 normalisation (VCC is rising, I/O is enabled, CLK is started, and RST is processed). If the card answers to this command, the answer will content all the ATR parameters; these parameters are memorised in TDA8029 and will be taken into account during the whole card session (until the card is deactivated or until a warm reset is processed). The structure of the answer is the following:

System to TDA8029

60	00 01	6E	00	0F
ACK	length	code	ISO	LRC

TDA8029 to System

60	XX XX	6E	nnnnnnnnnnnnnnnnnn	ZZ
ACK	length	code	ATR parameters	LRC

If the card is in specific mode, TDA8029 will process the next command directly using the new interface parameters of this specific mode. If the card proposes a different Fi/Di in the ATR than the default value (Fi/Di=372), it is up to the application to make a PPS command by using the negotiate command. If the card proposes 2 different protocols in its ATR, it is up to the application to make a PPS command by using the negotiate command.
If the card does not answer to the reset, an error code is returned to the application.

The power_up_5V command can be used to generate a warm reset if the card is already activated.

9.2.1.2 Power_up_3V

This command allows to activate the card at a VCC of 3V. Every signal going to the card will be referenced to this VCC=3V.
See power_up_5V for the other characteristics.

9.2.1.3 Power_up_1V8

This command allows to activate the card at a VCC of 1.8V. Every signal going to the card will be referenced to this VCC=1.8V.
See power_up_5V for the other characteristics.

9.2.2 Power_off

This command is used to deactivate the card whatever it has been activated for 3V or 5V operation. A deactivation sequence is processed following the ISO 7816-3 normalisation in about 100µs.

System to TDA8029 : 60 00 00 4D 2D
TDA8029 to System : 60 00 00 4D 2D

9.2.3 Card_command (APDU)

This command is used to transmit card commands under APDU format from system to TDA8029 whatever T=0 or T=1 protocol are used. Only short commands can be used.
An answer to such a command is also made in APDU format from TDA8029 to the system.

Example :
System to TDA8029 : 60 00 07 00 00 A4 00 00 02 4F 00 8E (SELECT FILE 4F 00)
TDA8029 to System : 60 00 02 00 90 00 F2

9.2.4 Negotiate

This command is used to make a PPS (Protocol and Parameter Selection) to the card, if in its ATR the card proposes a different Fi/Di or 2 different protocols. By using this command, a PPS will be made to the card with the Fi or Di and protocol type entered as a parameter (PP). It is up to the host to make the correct Fi/Di submission to the card.

Example :
System to TDA8029 : 60 00 02 10 PP FD LRC
TDA8029 to System : 60 00 00 10 70

Where FD is the ratio Fi/Di given by TA1 parameter of the ATR and PP is the protocol to be used.

If the command is acknowledged, any subsequent exchanges between the card and TDA8029 will be made by using the new parameters.

9.2.5 IFSD_request

This command is used to send a S(IFS request) block to the card indicating the maximum length of information field of blocks which can be received by the interface device in T=1 protocol. The initial size following the answer to reset is 32 bytes and this size shall be used throughout the rest of the card session or until a new value is negotiated by the terminal by sending a S(IFS request) block to the card.

System to TDA8029 : 60 00 01 0C PAR LRC
TDA8029 to System : 60 00 00 0C 6C

Where PAR is the IFSD size.

9.2.6 Set_clock_card

This command is used for changing the card clock frequency. The default value is set to FXTAL/4 which is 3.68625 MHz.
A parameter has to be transmitted in order to choose the card clock frequency:

System to TDA8029 : 60 00 01 11 PAR LRC

Frequency	Parameter
Fxtal =14.745MHz	00
Fxtal/2=7.37MHz	02
Fxtal/4=3.68MHz	04
Fxtal/8=1.84MHz	06

After a card clock frequency change, all the waiting times are internally set to the new value.

9.2.7 Set_card_baud_rate

This command is used mainly for cards which are not fully ISO 7816-3 compliant with specific and negotiable modes. As a matter of fact, some cards are in specific mode but they do not give TA2 parameter in their answer to reset. So the UART has to be set to the right baud rate by means of this specific command which programs the baud rate. For non ISO baud rates, there is a possibility to increase the capability of the reader by setting the bit CKU which divides by 2 the number of clock cycles of the etu and thus doubles the baud rate of the ISO UART.

Example :
System to TDA8029 : 60 00 02 0B XX CKU LRC
TDA8029 to System : 60 00 00 0B LRC

Where XX is the value of FiDi
if CKU=0, the baud rate is defined by FiDi
if CKU=1, the baud rate is 2 * the baud rate is defined by FiDi

For an etu of 372 clock cycles : XX=FiDi=0x11
prescaler = 31, divider = 12 ➜ 31 * 12 = 372, CKU=0.

The following baud rates are supported:

TA1	0x01	0x02	0x03	0x04	0x08
CLK/ETU	372	186	93	46.5	31

TA1	0x11	0x12	0x13	0x14	0x18
CLK/ETU	372	186	93	46.5	31

TA1	0x21	0x22	0x23	0x28
CLK/ETU	558	279	139.5	46.5

TA1	0x31	0x32	0x33	0x34	0x35	0x38
CLK/ETU	744	372	186	93	46.5	62

TA1	0x41	0x42	0x43	0x44	0x48
CLK/ETU	1116	558	279	139.5	93

TA1	0x51	0x52	0x53	0x54	0x55	0x56	0x58
CLK/ETU	1488	744	372	186	93	46.5	124

TA1	0x61	0x62	0x63	0x64	0x68	0x69
CLK/ETU	1860	930	465	232.5	155	93

TA1	0x91	0x92	0x93	0x94	0x95	0x96
CLK/ETU	512	256	128	64	32	16

TA1	0xA1	0xA2	0xA3	0xA4	0xA5	0xA6	0xA8
CLK/ETU	768	384	192	96	48	24	64

TA1	0xB1	0xB2	0xB3	0xB4	0xB5	0xB6
CLK/ETU	1024	512	256	128	64	32

TA1	0xC1	0xC2	0xC3	0xC4	0xC5	0xC6	0xC8
CLK/ETU	1536	768	384	192	96	48	128

TA1	0xD1	0xD2	0xD3	0xD4	0xD5	0xD6
CLK/ETU	2048	1024	512	256	128	64

Note : As the baud rates in dark boxes are using CKU bit, they are not reachable when CLK = Xtal.

9.2.8 Set_NAD

This command is used from the application layer in order to specify a SAD (source address) and a DAD (destination address) for a logical connection using T=1 protocol as defined in ISO7816-3. The default value is 00 and will be kept until the send NAD command has been notified to the TDA8029. Any NAD submission where SAD and DAD are identical (except 00) will be rejected. If bits b4 or b8 of the NAD required are set to 1 (VPP programming) the NAD will be rejected.
The NAD shall be initialised before any information exchange with the card using T=1 protocol, otherwise an error message will be generated.

System to TDA8029 : 60 00 01 A5 NAD LRC
TDA8029 to System : 60 00 00 A5 LRC

Where NAD is the new value of NAD immediately taken into account.

10 INFORMATION FIELD FOR ASYNCHRONOUS CARDS

The data buffer has a size of 512 bytes whose 6 bytes located at the end of the buffer are used by the internal library; so the data buffer has a real size of 506 bytes.
The information field that can include up to 506 bytes is composed of APDUs (Application Protocol Data Unit) according to the ISO7816-4 normalisation definition.
Different examples are given according to Annex A of the EMV'96 in T = 0.

TAL (System)		**TTL (TDA8029)**
Case 1 command {60, 00, 04, 00, CLA, INS, P1, P2, LRC}	⇒	
4 header bytes		
	⇐	{60, 00, 02, 00, 90, 00, LRC}
Case 2 command {60, 00, 05, 00, CLA, INS, P1, P2, 00, LRC}	⇒	
	⇐	{60, Licc+2, 00, [data (Licc)], 90, 00, LRC}
Case 3 command {60, Lc+5, 00, CLA, INS, P1, P2, Lc, [data Lc], LRC}	⇒	
	⇐	{60, 00, 02, 00, 90, 00, LRC}
Case 4 command {60, Lc+5+1, 00, CLA, INS, P1, P2, Lc, [data Lc], 00, LRC}	⇒	
	⇐	{60, Licc+2, 00, [data Licc], 90, 00, LRC}
Case 2 command using the 61 and 6C procedure byte Le = Licc or Le ≥ Licc {60, 00, 05, 00, CLA, INS, P1, P2, 00, LRC}	⇒	
	⇐	{60, D1+D2+Dn+2, 00, [data D1+D2+Dn], 90, 00, LRC}

11 SMART CARD READER DEVICE DRIVER

The demo board CAKE 8029_04-T is delivered with a Windows device driver so that it can be accessible to all Windows applications concerned with smart cards.
A device driver is a software component that enables a computer to communicate with a device. It manipulates the hardware in order to transmit the data to the device.

A device driver for a specific reader maps the functionality of the reader to the native services provided by the Windows platform and the smart card infrastructure. It is the responsibility of the reader device driver to communicate card insertion and removal events to the Resource Manager, and to provide data communications capabilities to and from the card by any or all of the T=0 or T=1 protocols.

All requests for smart card access go through the Resource Manager and are routed to the smart card reader containing the requested card. Therefore, the Resource Manager is responsible for managing and controlling all application access to any smart card inserted into any reader attached to a Windows-based PC. The Resource Manager provides a given application with a virtual direct connection to the requested smart card.

As described in paragraph 1, TDA8029 firmware handles the low-level protocols T=0 and T=1.

12 PC SMART CARD READER APPLICATION DESCRIPTION

12.1 Introduction

This PC application named « SCManager » is intended to operate with the demo board CAKE 8029_04-T. It uses the Microsoft Smart Card Base Components. They provide the necessary files needed to enable smart card aware application(s) or service provider(s) to communicate with a smart card through a reader attached to a PC and its corresponding device driver.

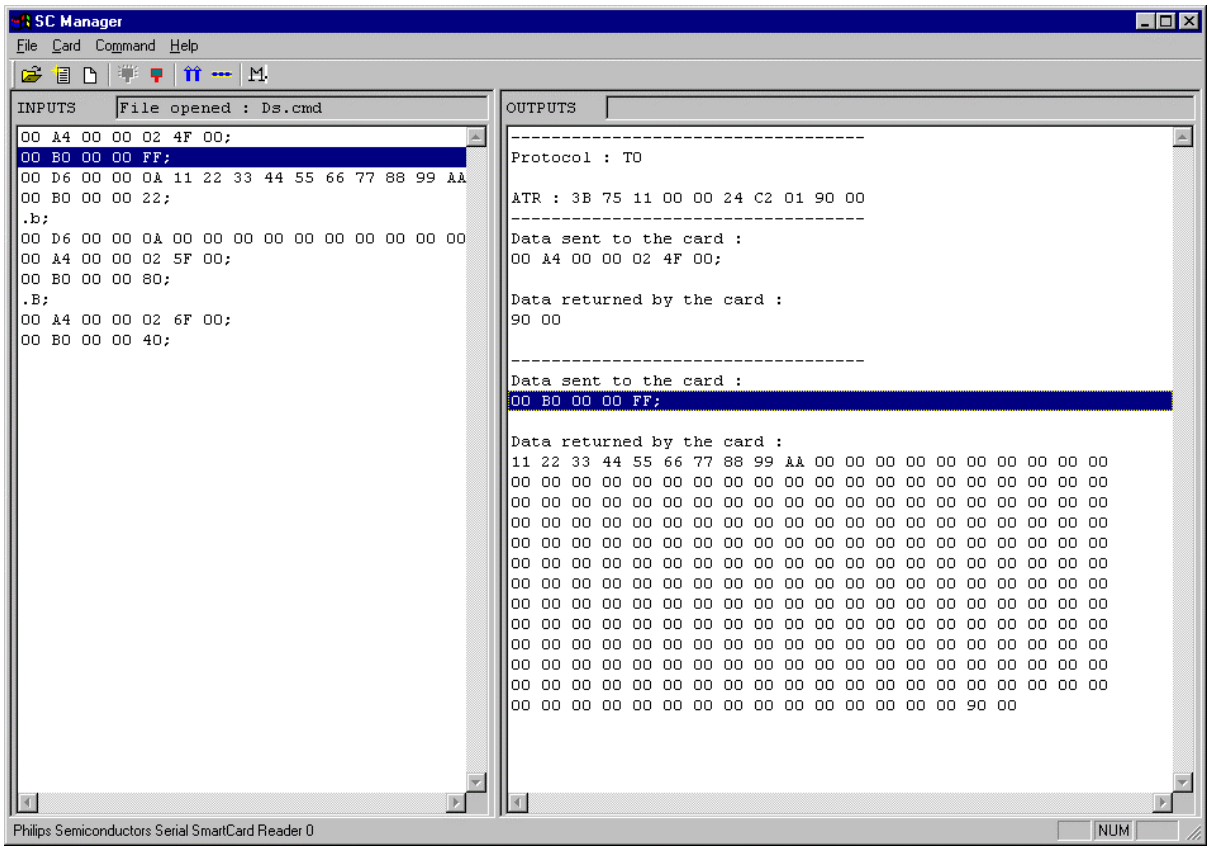
12.2 SCManager installation

Before executing SCManager.exe, the CAKE 8029_04-T device driver has to be installed from the PHILIPS SCR disk. More information about this installation is given in the file Readme.doc present on the disk.

12.3 SCManager presentation

SCManager is an application test developed with the SmartCard SDK provided by Microsoft and is using the CAKE 8029_04-T device driver. This application is able to power up, power down a card, and to transmit data to the smart card. Only APDUs can be sent to the card.

Here is a print screen of SCManager.exe:



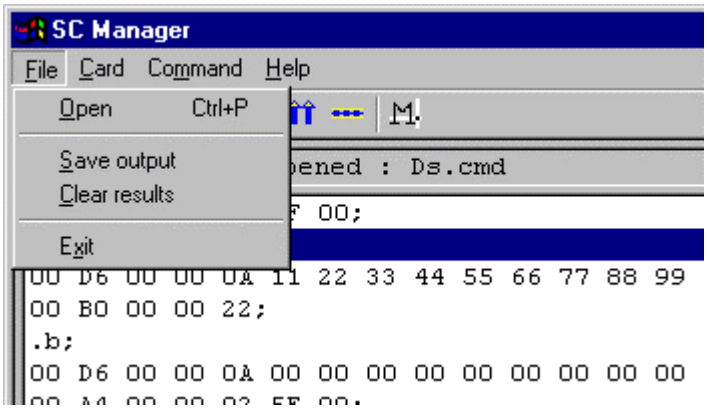
Two main windows are available:

- the left window INPUTS displays a list of C-APDUs from a command file selected via the File menu;
- the right window OUTPUTS displays the data exchange between the smart card and the PC (C-APDUs with R-APDUs associated).

Different menus are available on the first line. They will be explained into detail further.
Each menu line has a button equivalent.

12.4 Description of the different menus

File menu:



The Open item allows to select one existing command file (.cmd file).
This .cmd file is built with card commands (C-APDUs) and each command line has to be closed by a semi coma (;).
Some comments can also been added preceded by a *.

Example:

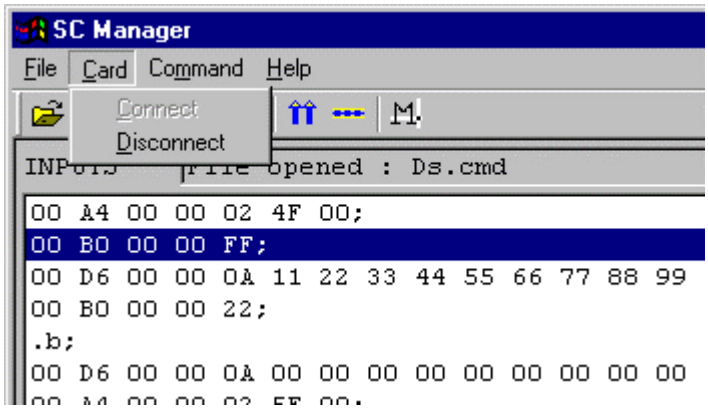
*Selection of a file (4F 00) in the card;
00 A4 00 00 02 4F 00;
*Read 10 bytes;
00 B0 00 00 0A;

The Save output item allows to store the content of the OUTPUTS window into a file .log.

The Clear results item deletes the content of the OUTPUTS window.

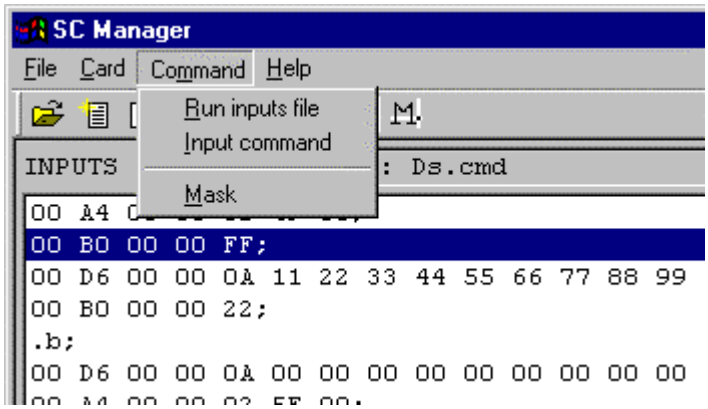
The Exit item allows to quit “SCManager” application.

Card menu:



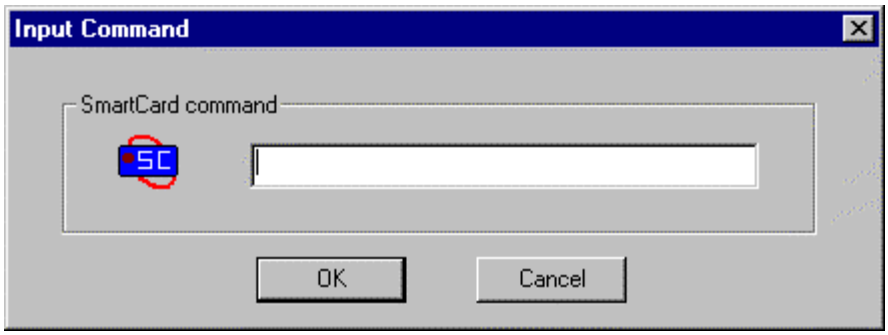
The Connect item will open a connection between the application and the smart card inserted through the Resource Manager, whereas Disconnect item will close it.

Command menu:



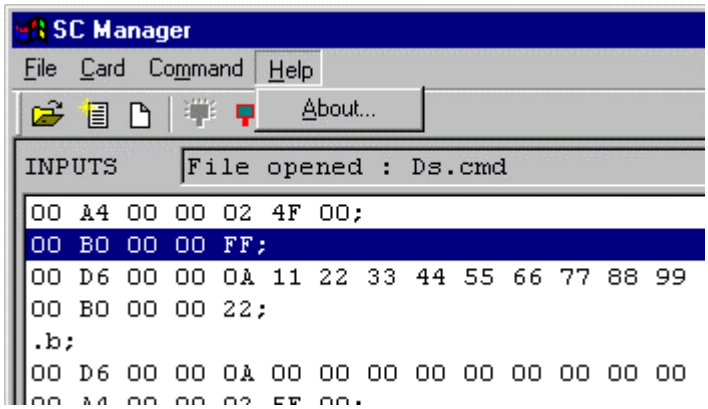
The Run inputs file item will execute the complete command file opened until its end.
A break point can be added in the command file by editing a line with .b; or .B;.
This will stop the command file execution. Select Run inputs file in the menu to continue the command file execution.

The Input command item allows to enter manually a C-APDU in the following dialog box:



The Mask item displays the TDA8029 mask number version. The card must be powered on before this operation.

Help menu:



The About… menu will give details on “SCManager” application.

12.5 Microsoft tools

Microsoft provides a smart card reader/driver test program (ifdtest.exe) to check the driver compliance with the Microsoft implementation of PC/SC. This application is used for the Microsoft Windows Logo program.

This test tool:

- tests if the driver correctly reports card insertion and removal events;
- checks basic reader attributes and basic driver functionality;
- simulates the behaviour of the smart card resource manager;
- tests the reader with a set of test smart cards.

Ifdtest.exe is part of “SCard99.exe” that can be downloaded from http://www.microsoft.com/hwtest, following the links Test Kits and smart card reader.

In order to use ifdtest.exe, the smart card resource manager (scardsvr.exe) must not be running.

13 APPLICATION RECOMMENDATIONS FOR LOW TEMPERATURE OPERATION

For low temperature operation (-20°C, -30°C), the following procedure at supply voltage powering on is recommended :

1. VDD and DCIN are rising,
2. Wait 30ms (or less till CDEL is loaded) and set RESET pin high for about 20µs,
3. Wait another 30ms,
4. TDA8029 is ready to operate.

This operating mode can of course be used on the whole temperature range.

14 CONCLUSION

This serial PC/SC smart card reader can be used in any security-sensitive or personal data application such as electronic commerce, home banking or e-purse facilities, secure computer access, digital signature, secure e-mails, certificate-based authentication, etc.

The following features give the general characteristics of this mask:

- 1.8V, 3V and 5V cards supported
- PC/SC compliance in mode ISO and with power_up_5V function call
- Data buffer up to 506 bytes
- Asynchronous protocols (T=0 and T=1) supported
- Control and communication through a serial interface at 38 400 bauds
- Automatic hardware protections in the event of card take off, supply voltage drop, short circuit or overheating
- All ISO7816-3 baud rates supported on the I/O line
- Possible selection of card clock frequencies
- Communication with the host made at the APDU level (asynchronous cards)
- Single +5V supply voltage

15 APPENDIX: DRIVER PROTOCOL DOCUMENTATION

INTERACTIONS BETWEEN SMART CARD COMPONENTS AND DRIVERS

15.1 Smart Card Components

15.1.1 Overview

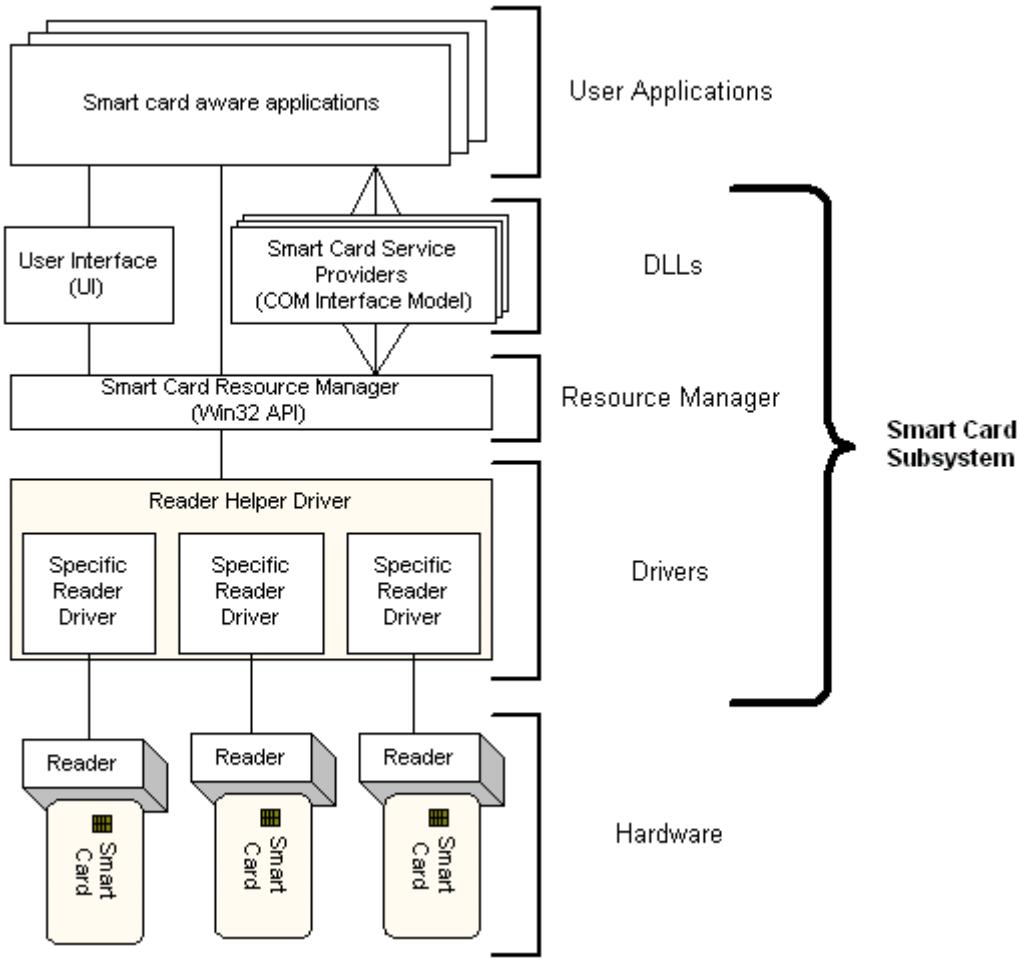
The Smart Card Components are the Microsoft way to provide the users access to smart cards.
The basic components of the smart card subsystem are defined in the *Interoperability Specification for ICCs and Personal Computer Systems* (see Documents at http://www.pcscworkgroup.com/).

These basic components include:

- a resource manager which uses a Win32® application programming interface (API);
- a user interface (UI) which works with the resource manager;
- several base service providers which provide access to specific services.

In contrast to the resource manager's Win32® API, service providers use a COM interface model to provide smart card services.

The following figure shows the relationship of these components in the overall smart card architecture:



15.1.2 Resource Manager

The smart card resource manager manages the access to readers and to smart cards. To manage these resources, it performs the following three functions:

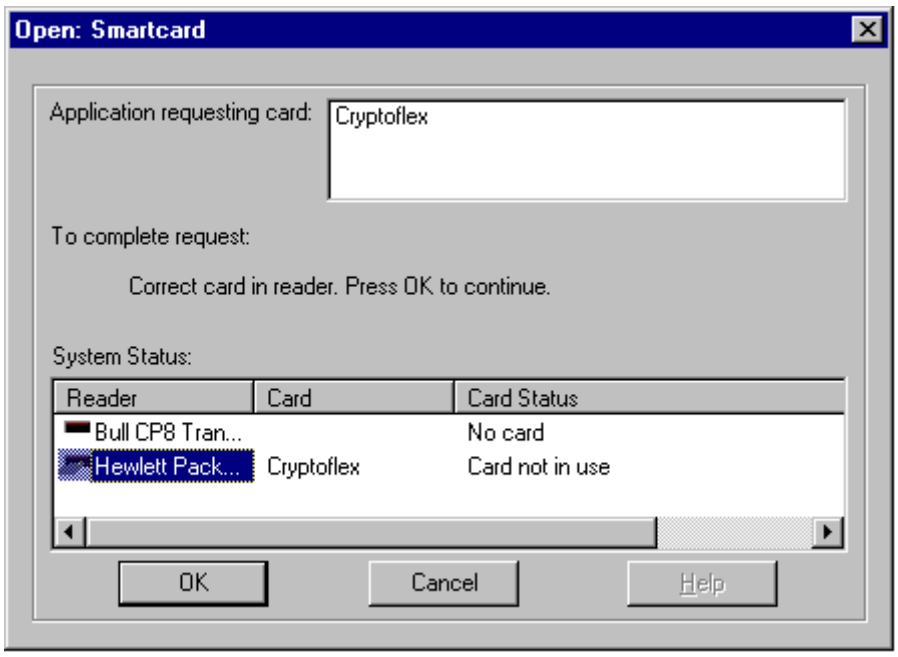
- identifies and tracks resources;
- allocates readers and resources across multiple applications;
- supports transaction primitives for accessing services available on a given card.

Note: This is an important point because current cards are single-threaded devices that often require the execution of multiple commands to complete a single function. Transactions allow multiple commands to be executed without interruption, ensuring that intermediate state information is not corrupted.

The resource manager can be accessed directly via the resource manager API or indirectly through any smart card service provider.
The resource manager API is a set of Win32® functions which provides direct access to the resource manager's services. In comparison, smart card service providers use COM interfaces.

15.1.3 Smart Card User Interface

The smart card user interface (UI) is a single common dialog that lets the user specify or search for a smart card to open (that is, connect to and use in an application).



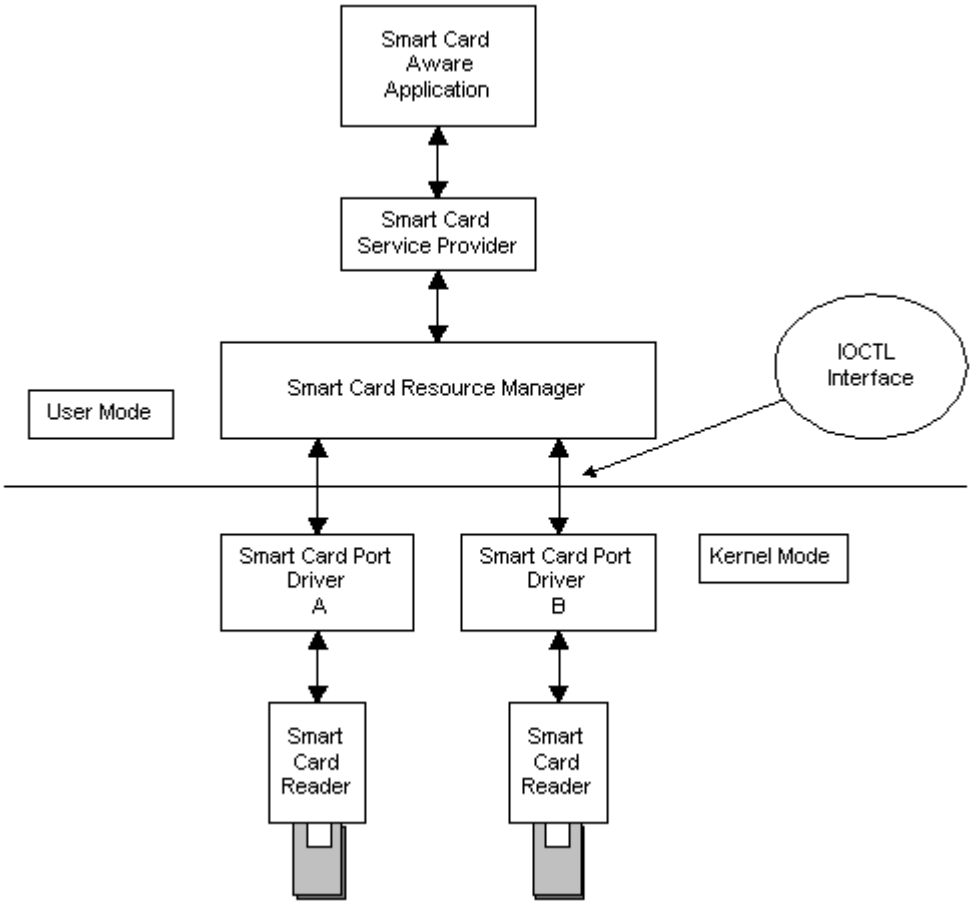
15.1.4 Smart Card Service Providers

Service providers (SCSP) provide access to Smart Card capabilities. They can provide access to a single capability, such as the base service providers provided by the Smart Card SDK, or they can provide access to several capabilities in order to accomplish a more complex task.
Service providers provide access through COM interfaces. The Smart Card SDK provides the COM interfaces used by its own base service providers, as well as several application interfaces that can be used when developing custom service providers.

15.2 Smart Card Drivers Overview

15.2.1 Environment

The following figure illustrates the standard environment for a smart card driver. The driver resides in kernel space and is called only by the smart card resource manager that resides in user space. The interface used by the resource manager and the driver is based on IOCTL calls.



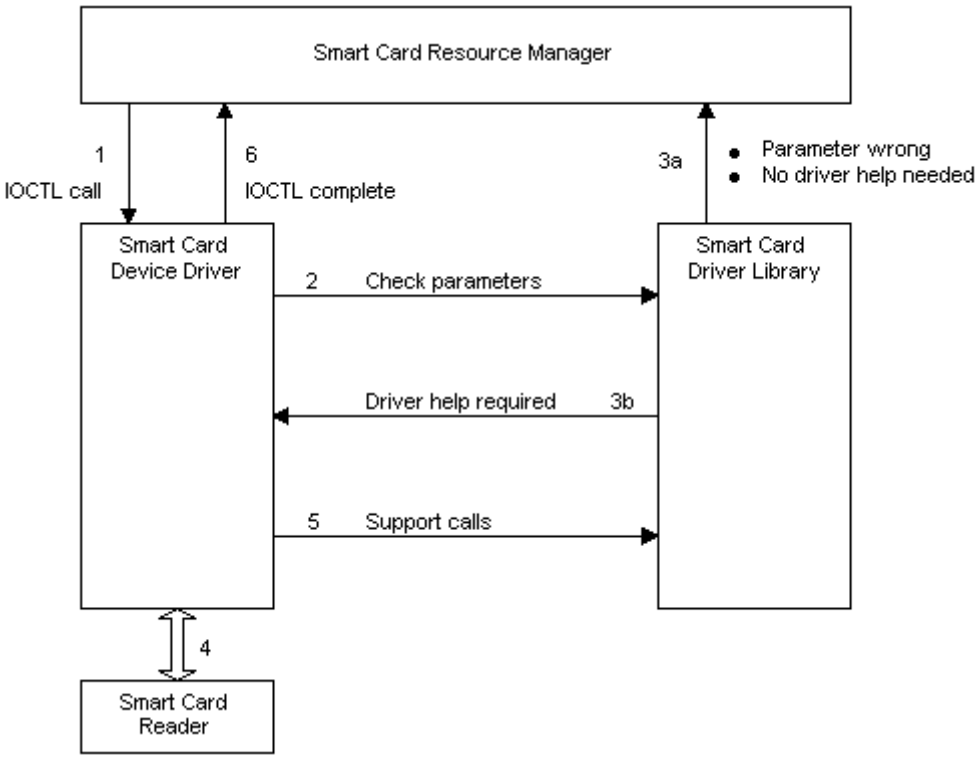
15.2.2 IOCTL Codes

The following table lists the available smart card IOCTL codes. Except IOCTL_SMARTCARD_GET_LAST_ERROR, these codes apply to both WDM and VxD drivers.

IOCTL Code	*Description*
IOCTL_SMARTCARD_EJECT	*Ejects the currently inserted smart card from the smart card reader.*
IOCTL_SMARTCARD_GET_ATTRIBUTE	*Queries smart card and smart card reader attributes*
IOCTL_SMARTCARD_GET_LAST_ERROR	*VxD drivers only. Retrieves the error code of the last operation.*
IOCTL_SMARTCARD_GET_STATE	*Gets the current status of the smart card reader.*
IOCTL_SMARTCARD_IS_ABSENT	*Either returns immediately if no card is currently inserted, or installs an event handler to track card removals.*
IOCTL_SMARTCARD_IS_PRESENT	*Either returns immediately if no card is currently inserted, or installs an event handler to track card insertions.*
IOCTL_SMARTCARD_POWER	*Either powers down or warm resets the card.*
IOCTL_SMARTCARD_SET_ATTRIBUTE	*Sets various attributes in the driver.*
IOCTL_SMARTCARD_SET_PROTOCOL	*Sets the protocol to be used with the currently inserted card.*
IOCTL_SMARTCARD_SWALLOW	*Causes the reader to swallow the card.*
IOCTL_SMARTCARD_TRANSMIT	*Transmits data to and receives data from the currently inserted smart card.*

15.2.3 Smart Card Driver IOCTL Calling Scheme

The following figure shows the calling scheme of an IOCTL call:



This is the process that takes place in the calling scheme:

- The resource manager calls the driver (1). The driver calls the library (2). The library checks the parameters of this call. If the driver's help is not needed to satisfy the call, the library returns the requested data, or an error code, to the caller (3a). If the driver's help is needed to satisfy the call, for example if I/O is required or if this is a vendor-specific call, the library calls the driver with all parameters set up correctly (3b). The driver performs I/O and uses support functions of the library (4 and 5). The driver then completes the request and returns the call to the resource manager (6).

- The smart card library synchronises access to your driver. Two callback functions will never be called at the same time. However, card insertion and removal event handling must be processed asynchronously.

15.2.4 Smart Card Driver Library Callback

The following table lists the callback function available from both libraries. Some functions are mandatory and others are optional. The driver needs to set up only mandatory functions. Each function that has not been set up will return STATUS_NOT_SUPPORTED to the smart card resource manager.

Callback Function	*Requirements*	*Description*
RDF_CARD_POWER	Mandatory	*Resets or powers down an inserted smart card.*
RDF_CARD_EJECT	Optional	*Ejects an inserted smart card.*
RDF_CARD_TRACKING	Mandatory	*Installs an event handler to track card insertions and removals.*
RDF_IOCTL_VENDOR	Optional	*Performs vendor-specific IOCTL operations.*
RDF_READER_SWALLOW	Optional	*Does a mechanical swallow.*
RDF_SET_PROTOCOL	Mandatory	*Selects a transmission protocol for the currently inserted card.*
RDF_TRANSMIT	Mandatory	*Performs data transmissions.*

Note : The demo board CAKE 8029_04-T does not handle cards swallowing and ejecting (the card connector has no specific mechanism), thus the corresponding callbacks (RDF_CARD_EJECT and RDF_READER_SWALLOW) are not implemented.

15.3 Driver Implementation

As the protocol is common on all platforms, the functions used by the different versions of the drivers are the same. They implement the necessary materials to handle communication between the Smart Card Components and the reader.
As described here above, the mandatory functions are callbacks for card tracking, powering on and off the card, setting the protocol, transmitting data and performing vendor specific calls. These callbacks are simple functions: they usually send commands to the reader. Most of the work is thus done by the **PhSC_SendCommand** function.

15.3.1 Callbacks calling scheme

When an IOCTL call is done by the Resource Manager, it calls the **DeviceIoControl** driver's function. This function checks that the state of the reader enables such a command, then calls the **SmartcardDeviceControl** function of the Smart Card Components.
This function performs the actual parameters checking, and if the call cannot be satisfied without the driver, it calls one of the callback methods (RDF_xxx).
When the callback returns, the results are sent back to the Resource Manager.

15.3.2 Sending commands to the reader

As all transmissions between the PC and the reader (except card tracking notifications) are composed of a command sent to the reader and its answer, these transactions are performed by the **PhSC_SendCommand** function.

- The first phase is the preparation of the command. If the command is a simple one (all except sending an APDU to the card), there is no need for computation as the commands never change. If the command is an APDU, the data are encapsulated in the ALPAR protocol by the **PhSC_GenSendDataBlock** function.
- The second phase is the transmission and reception of data, using a driver/platform specific method, **PhSC_WriteAndRead**.
- The last phase is to check the answer and to update information accordingly. This is done by the **PhSC_UpdateCardStatus** function, which performs the smart card/reader status checking. When the status has changed, the function notifies the Smart Card Components of the changes.

15.3.3 Shared callbacks functions

15.3.3.1 Card Power (PhSC_CardPower)

The **RDF_CARD_POWER** callback function resets or powers down an inserted smart card.
Depending of the subfunction (power up, power down, warm reset), the driver sends the appropriate command (power_up_5V ($6E_H$) or power_down ($4D_H$)) to the reader. If a reset of the card is asked, the ATR is stored in the card capabilities buffer.

15.3.3.2 Set Protocol (PhSC_SetProtocol)

The **RDF_SET_PROTOCOL** callback function selects a transmission protocol for the currently inserted card.
The driver sends the negotiate command (10_H) to the reader (with two parameters set to reflect which protocol and which Fi/Di will be used) and returns the result. For T=1 protocol, an IFSD request is sent to the reader before returning, using the ifsd_request command ($0C_H$).

15.3.3.3 Transmit (PhSC_Transmit)

The **RDF_TRANSMIT** callback function performs data transmission.
The driver sends a card_command function (00_H) to the reader with the data passed as parameters.

15.3.3.4 IOCTL Vendor (PhSC_IoctlVendor)

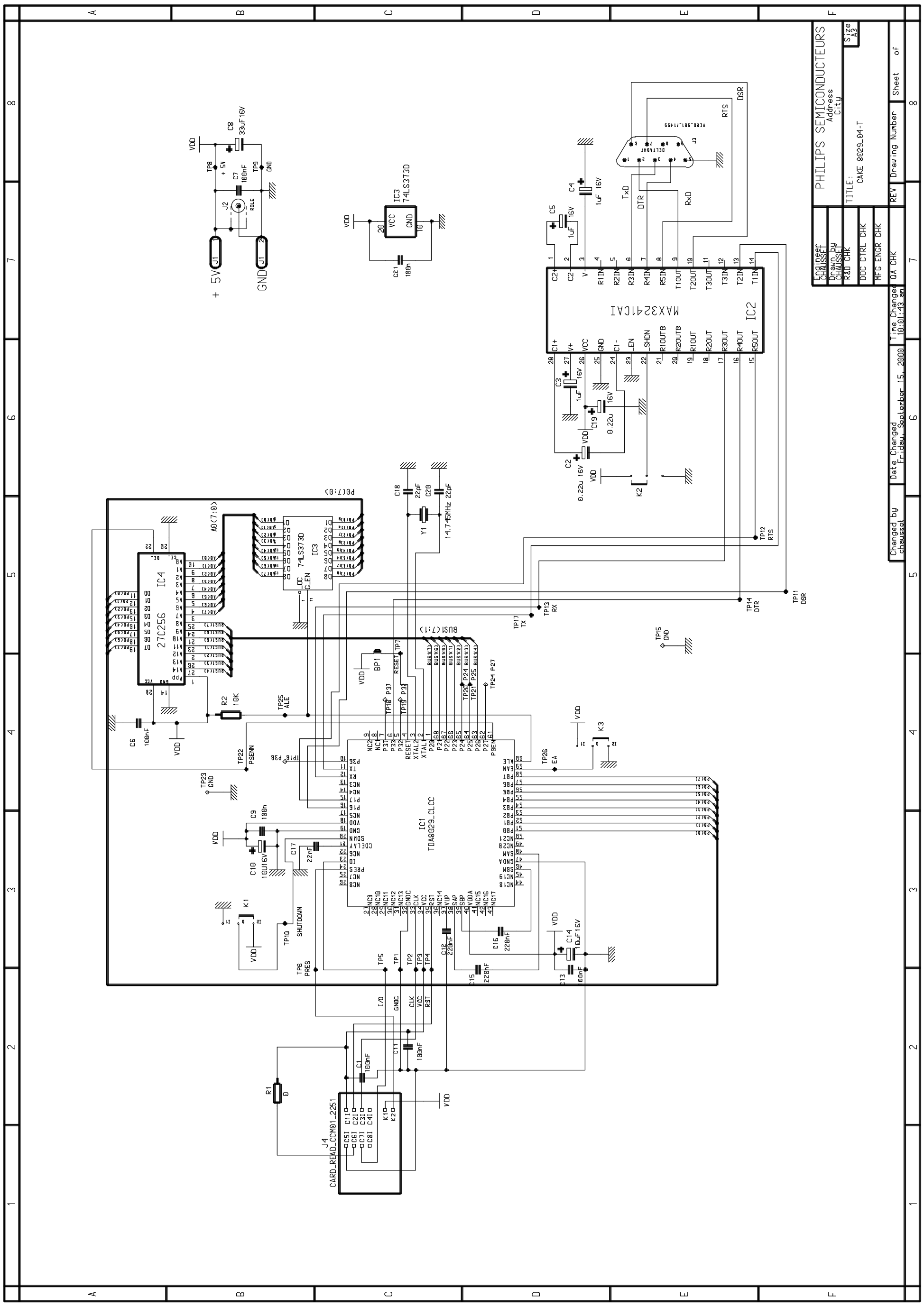
The **RDF_IOCTL_VENDOR** callback function performs vendor-specific IOCTL operations.
Two operations are currently implemented:

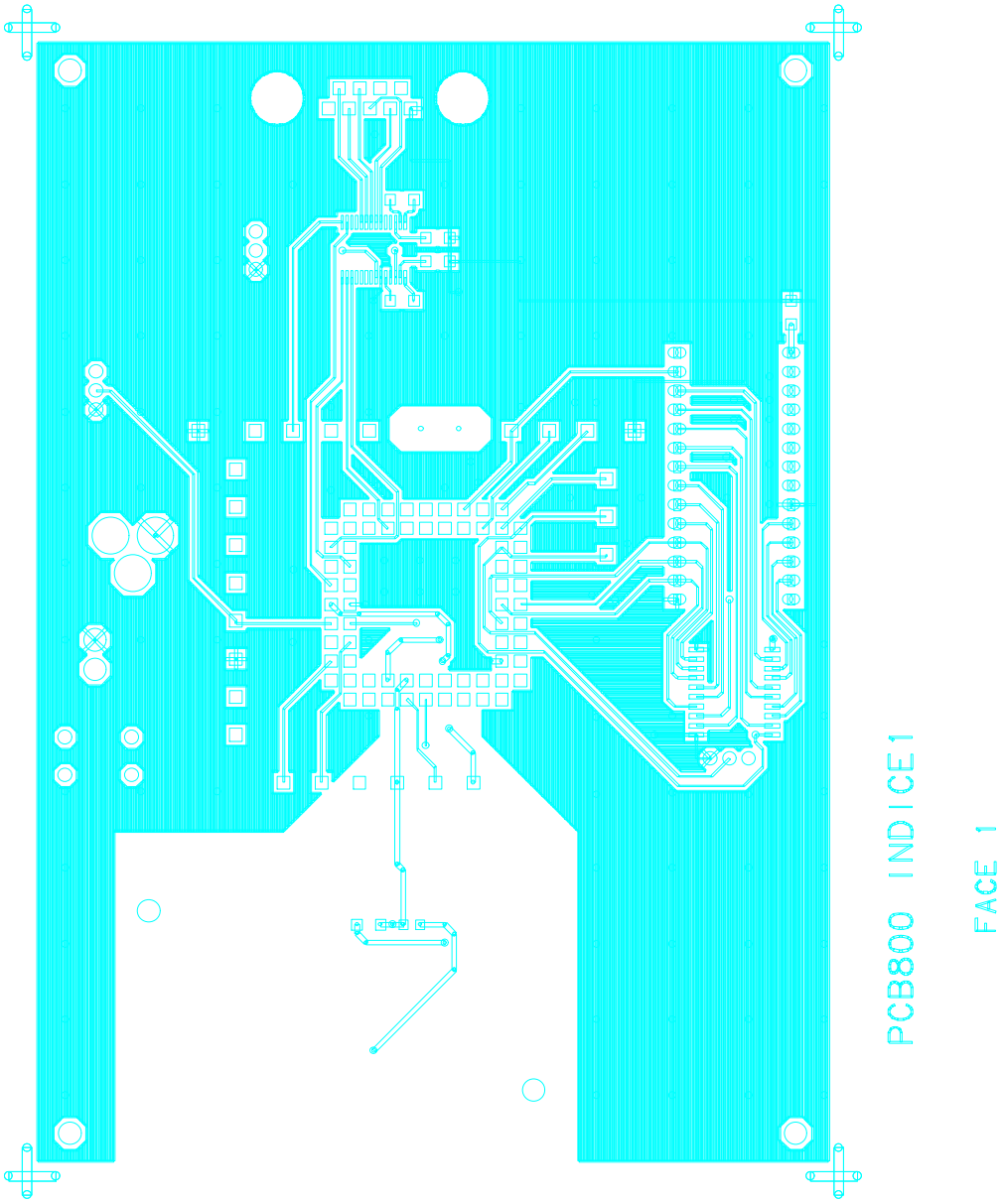
- IOCTL_GET_READER_MASK: used to retrieve the mask number of the connected reader. This function sends the send_num_mask command ($0A_H$) to the reader and returns the character string sent back by the reader.
- IOCTL_GET_LAST_ERROR_CODE: used to retrieve the last error code sent by the reader. This has been implemented to allow the user to access more specific error codes than those used by the Smart Card Components. This function only returns the error code that has been saved in the last transmission.

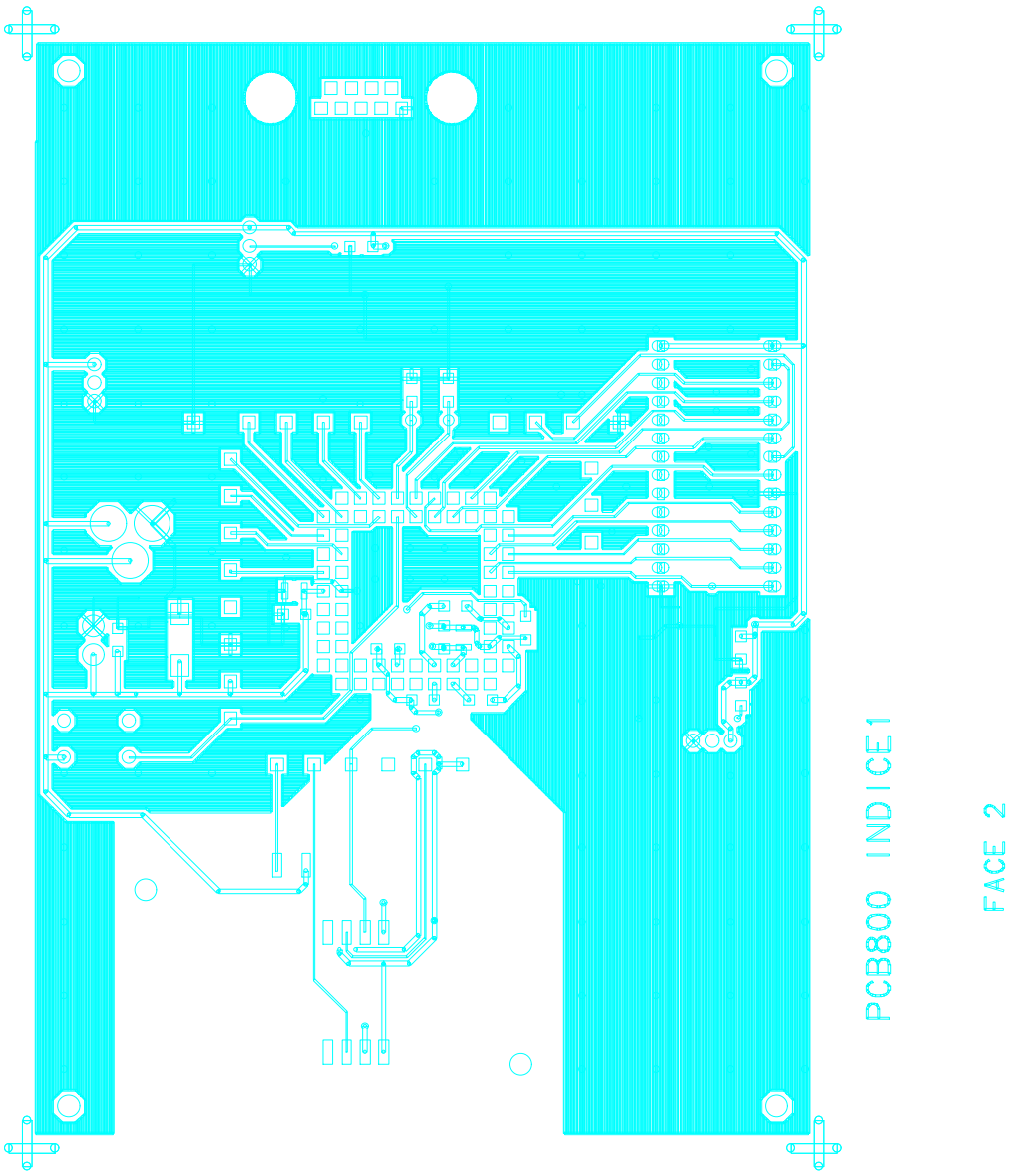
15.3.3.5 Card Tracking (PhSC_CardTracking)

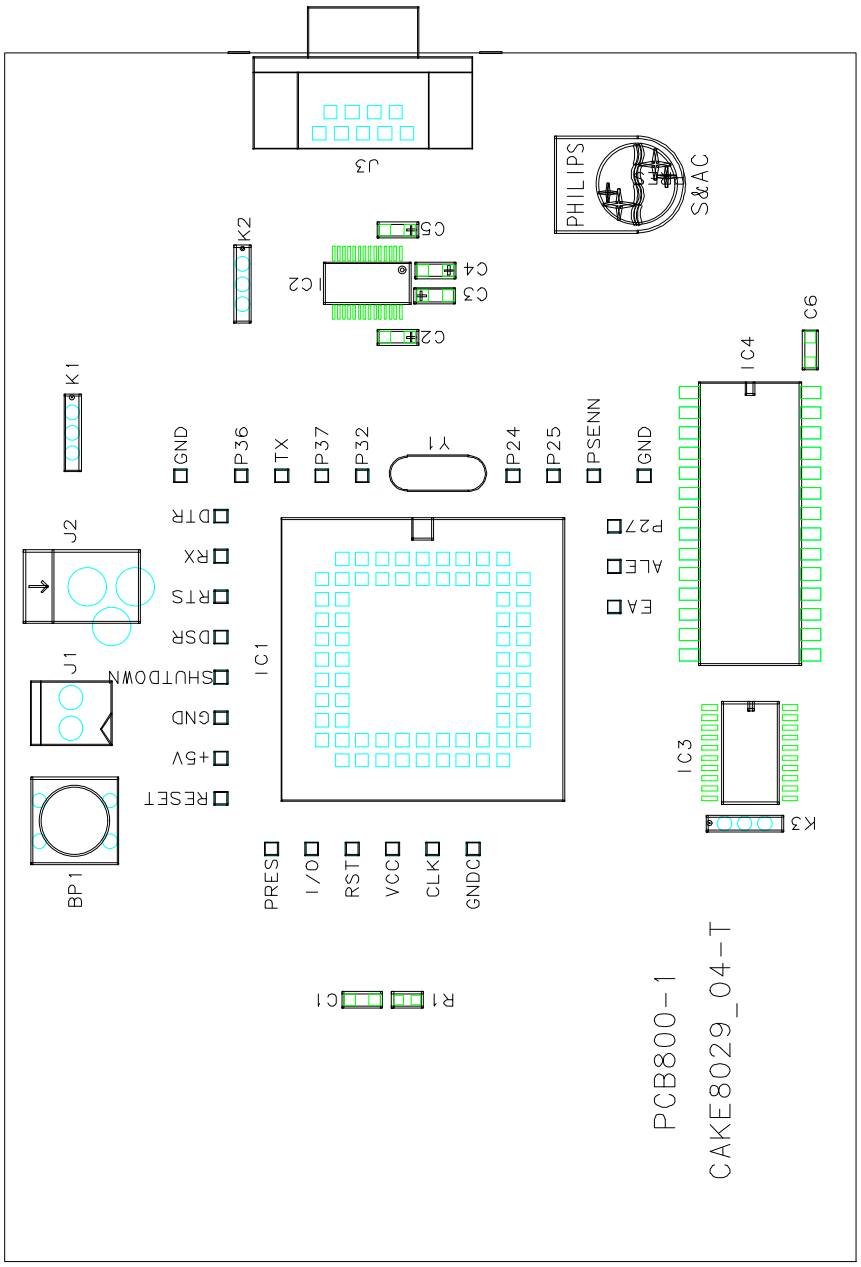
The **RDF_CARD_TRACKING** callback function installs an event handler to track card insertions and removals.
As card tracking must be handled asynchronously, the callback function **PhSC_CardTracking** only notifies the driver that the Smart Card Components are waiting for an event.
The actual card tracking is handled differently under platforms. The serial reader sends a command to the PC (the only one sent by the initiative of the reader) to notify if a card was inserted or removed.
When an event occurs the driver calls the **PhSC_CompleteCardTracking** to notify the Smart Card Components of the change.

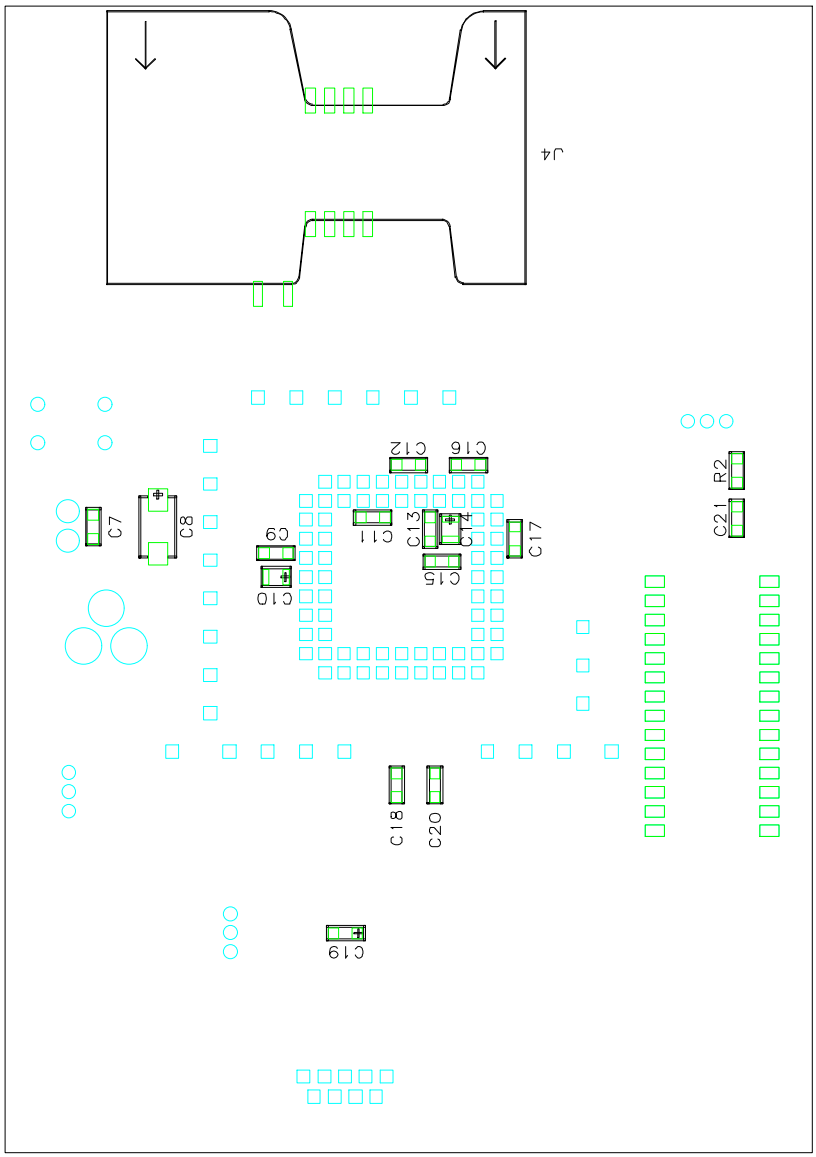
13 APPENDIX: HARDWARE INFORMATION


PHILIPS SEMICONDUCTEURS
MAX3241CAI
IC2
TDA8029_CLCC
IC1
27C256
IC4
74LS373D
IC3
CARD_READ_CCM01_2251
J4


PCB800 INDICE1
FACE 1


PCB800 INDICE1
FACE 2


J3
PHILIPS
S&AC
K2
C5
C4
C3
C2
IC2
C6
IC4
K1
GND
P36
TX
P37
P32
Y1
P24
P25
PSENN
GND
P27
ALE
EA
DTR
RX
RTS
DSR
SHUTDOWN
GND
+5V
RESET
J2
J1
IC1
IC3
K3
BP1
PRES
I/O
RST
VCC
CLK
GNDC
R1
C1
PCB800-1
CAKE8029_04-T
SERIGRAPHIE FACE 1



SERIGRAPHIE FACE 2

LABORATOIRES D APPLICATIONS PHILIPS
2 Route de Girafe
BP 5120
14079 CAEN CEDEX 5
R.BASTIEN TEL: 02.31.45.31.17
FAX: 02.31.45.30.70

CARTE : PCB800 IND : 1 ETUDE: CAKE8029_04-T

PROJET : CAKE8029 LE : 25 09 00

FAIT PAR : bastien

LISTE DES COMPOSANTS FACE 1 FACE 2

REFERENCE	GEOMETRY	DESCRIPTION
BP1	MICROCOSMOS	POUS_MICRO
C1	1206	C1206, 100nF
C2	293D_A	C293D_A, 0.22u
C3	293D_A	C293D_A, 1uF
C4	293D_A	C293D_A, 1uF
C5	293D_A	C293D_A, 1uF
C6	1206	C1206, 100nF
C7	1206	C1206, 100nF
C8	293D_D	C293D_D, 33uF
C9	1206	C1206, 100n
C10	595D_A	C595D_A, 10U
C11	1206	C1206, 100nF
C12	1206	C1206, 220nF
C13	1206	C1206, 100nF
C14	595D_A	C595D_A, 10uF
C15	1206	C1206, 220nF
C16	1206	C1206, 220nF
C17	1206	C1206, 22nF
C18	1206	C1206, 22pF
C19	293D_A	C293D_A, 0.22u
C20	1206	C1206, 22pF
C21	1206	C1206, 100n
IC1	sp_plcc68	tda8029_clcc
IC2	SOT341_1	MAX3241CAI
IC3	SO20L	74LS373D, 74LS373D
IC4	SOT117_1	27C256
J1	CONN353MV2	EDGE
J2	JACK2.5_H	JACK2.5_H
J3	DELTA9HF	DELTA9HF
J4	CARD_READ_CCM01_2251	CARD_READ_CCM01_2251
K1	CAV_1C2P	CAV_1C2P
K2	CAV_1C2P	CAV_1C2P
K3	CAV_1C2P	CAV_1C2P
R1	0805	RC11, 0
R2	1206	RC01, 10K

TP1	TP	POINT_TEST
TP2	TP	POINT_TEST
TP3	TP	POINT_TEST
TP4	TP	POINT_TEST
TP5	TP	POINT_TEST
TP6	TP	POINT_TEST
TP7	TP	POINT_TEST
TP8	TP	POINT_TEST
TP9	TP	POINT_TEST
TP10	TP	POINT_TEST
TP11	TP	POINT_TEST
TP12	TP	POINT_TEST
TP13	TP	POINT_TEST
TP14	TP	POINT_TEST
TP15	TP	POINT_TEST
TP16	TP	POINT_TEST
TP17	TP	POINT_TEST
TP18	TP	POINT_TEST
TP19	TP	POINT_TEST
TP20	TP	POINT_TEST
TP21	TP	POINT_TEST
TP22	TP	POINT_TEST
TP23	TP	POINT_TEST
TP24	TP	POINT_TEST
TP25	TP	POINT_TEST
TP26	TP	POINT_TEST
Y1	RW43V	HC49U_V, 14.745MHz