TERIDIAN
SEMICONDUCTOR CORP.
A Maxim Integrated Products Brand

73S1215F, 73S1217F
USB CCID Guidelines

September 14, 2009
Rev. 2.10
AN_12xxF_028
Library Rev. 4.00(DFU)
Code/Firmware Rev. TSC12xx.2.10

Maxim cannot assume responsibility for use of any circuitry other than circuitry entirely embodied in a Maxim product. No circuit patent licenses are implied. Maxim reserves the right to change the circuitry and specifications without notice at any time.

Maxim Integrated Products, 120 San Gabriel Drive, Sunnyvale, CA 94086 408-737-7600

Table of Contents

Figures

Tables

1 Introduction

This document contains guidelines for the use of the TSC73S12xxF in a USB environment, particularly Microsoft® Windows XP® and PC/SC. The intent of this application note is to aid customers in the development of Smart Card Reader applications specific to their environment. It is not intended as a reference document for the software mentioned in this document.

2 Overview

Figure 1 shows an overview of the 73S12xxCCID Software. TSC has provided an embedded CCID firmware application that is compliant with the USB CCID Class Specification of the *USB Device Class Specification for USB Chip/Smart Card Interface Devices Specification*, Revision 1.1. The firmware has been developed on the TSC 73S12xxF Evaluation Board. The firmware is provided in the form of ANSI C source code and Keil uVision2/3 project files so that the customer may use them to modify and enhance the application to meet their requirements.

In the Windows environment, the customer may employ either the Microsoft-supplied CCID USB client driver or the TSC enhanced CCID USB client driver. The TSC driver offers additional hardware support for multiple card slots, LCD operation and PIN pad input.

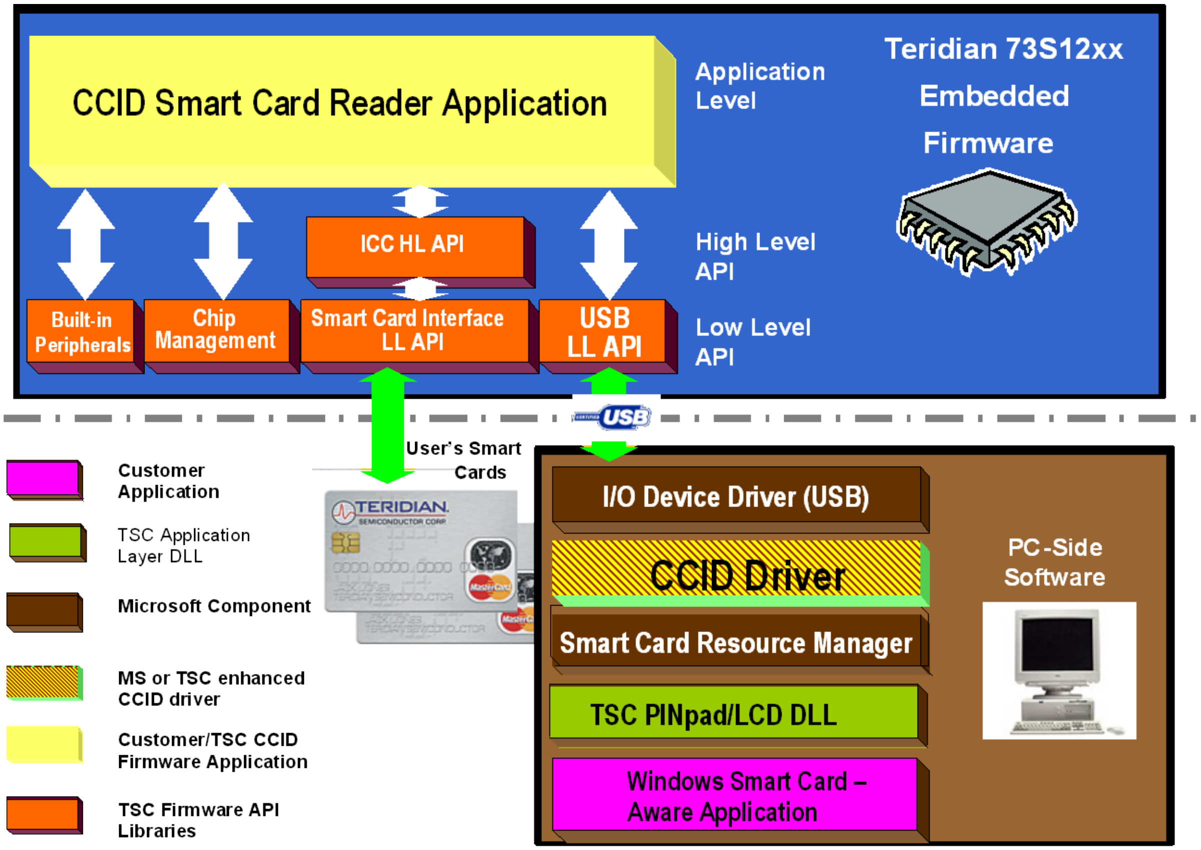

CCID Smart Card Reader Application
Application Level
Teridian 73S12xx Embedded Firmware
ICC HL API
High Level API
Built-in Peripherals
Chip Management
Smart Card Interface LL API
USB LL API
Low Level API
USB
User's Smart Cards
TERIDIAN SEMICONDUCTOR CORP
Customer Application
TSC Application Layer DLL
Microsoft Component
MS or TSC enhanced CCID driver
Customer/TSC CCID Firmware Application
TSC Firmware API Libraries
I/O Device Driver (USB)
CCID Driver
Smart Card Resource Manager
TSC PINpad/LCD DLL
Windows Smart Card – Aware Application
PC-Side Software

Figure 1: Overview of the 73S12xx CCID Software

On the Windows side, the customer develops application software using the Windows Smart Card API to access/control the reader.

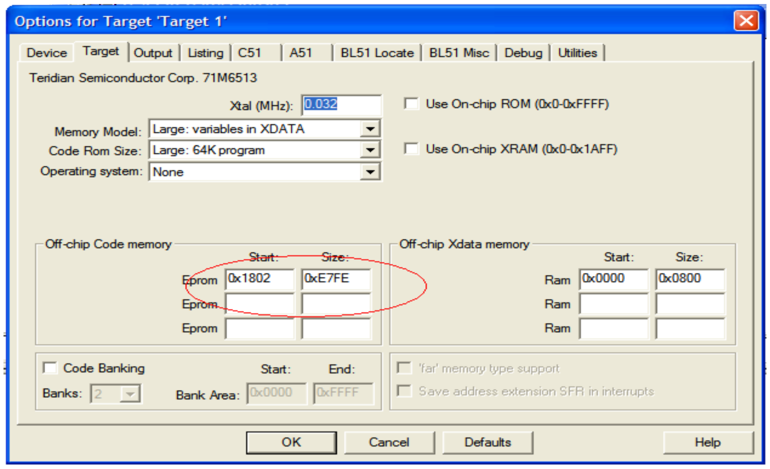
3 CCID Application Firmware

Once the 73S12xxF Development Kit is installed, the CCID Application Firmware is contained in the CCID USB folder. Refer to the *73S12xxF Software User's Guide* description of the contents of this installation disk. The firmware is delivered in the form of source code and Keil uVision (.uv2 or .uv3) project files. Customers can modify and build this software and download .hex files to the 73S12xxF Evaluation or Demo Board as detailed in the *73S12xxF Software User's Guide*, the *73S12xxF Evaluation Board User's Guide*, and the *Teridian Flash Programming Tool*.

3.1 CCID Application Build Environments

Starting from revision 2.00, the CCID Application with the DFU Boot Loader feature was partitioned to reside starting at address 0x1802 instead of the previous address of 0x0000. The CD ROM version 2.00 built for CCID USB version 2.00 has two distinct flavors: with the DFU Boot Loader and without the DFU Boot Loader (same as previous releases). Prior CCID projects (without the Boot Loader) remain the same. The CCID+DFU project file setup to accommodate the DFU Boot Loader feature has the following changes:

1. The Application starting address is set at 0x1802 as shown below:



Teridian Semiconductor Corp. 71M6513

Xtal (MHz): 0.032

□ Use On-chip ROM (0x0-0xFFFF)

Memory Model: Large: variables in XDATA

Code Rom Size: Large: 64K program

Operating system: None

□ Use On-chip XRAM (0x0-0x1AFF)

Off-chip Code memory

	Start:	Size:
Eprom	0x1802	0xE7FE
Eprom		
Eprom		

Off-chip Xdata memory

	Start:	Size:
Ram	0x0000	0x0800
Ram		
Ram		

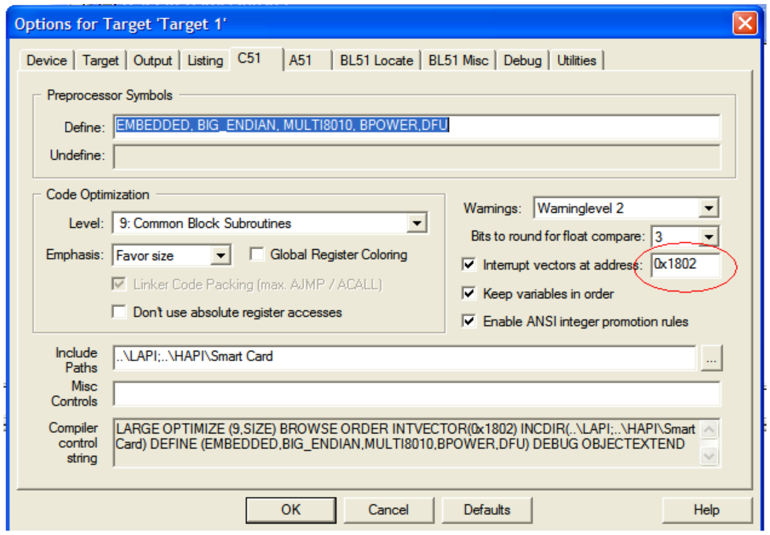
□ Code Banking Start: End:

Banks: 2 Bank Area: 0x0000 0xFFFF

□ 'far' memory type support

□ Save address extension SFR in interrupts

2. The Interrupt Vector address is also set to start at 0x1802 as shown below. The DFU directive (circled in black below) is used to specifically select the DFU feature into the build. The LEDMGT directive (circled in red below) indicates the new LED0 control feature as described in Section 5.2.4.



Preprocessor Symbols

Define: EMBEDDED, BIG_ENDIAN, MULTI8010, BPOWER,DFU

Undefine:

Code Optimization

Level: 9: Common Block Subroutines

Emphasis: Favor size □ Global Register Coloring

☑ Linker Code Packing (max. AJMP / ACALL)

□ Don't use absolute register accesses

Warnings: Warninglevel 2

Bits to round for float compare: 3

☑ Interrupt vectors at address: 0x1802

☑ Keep variables in order

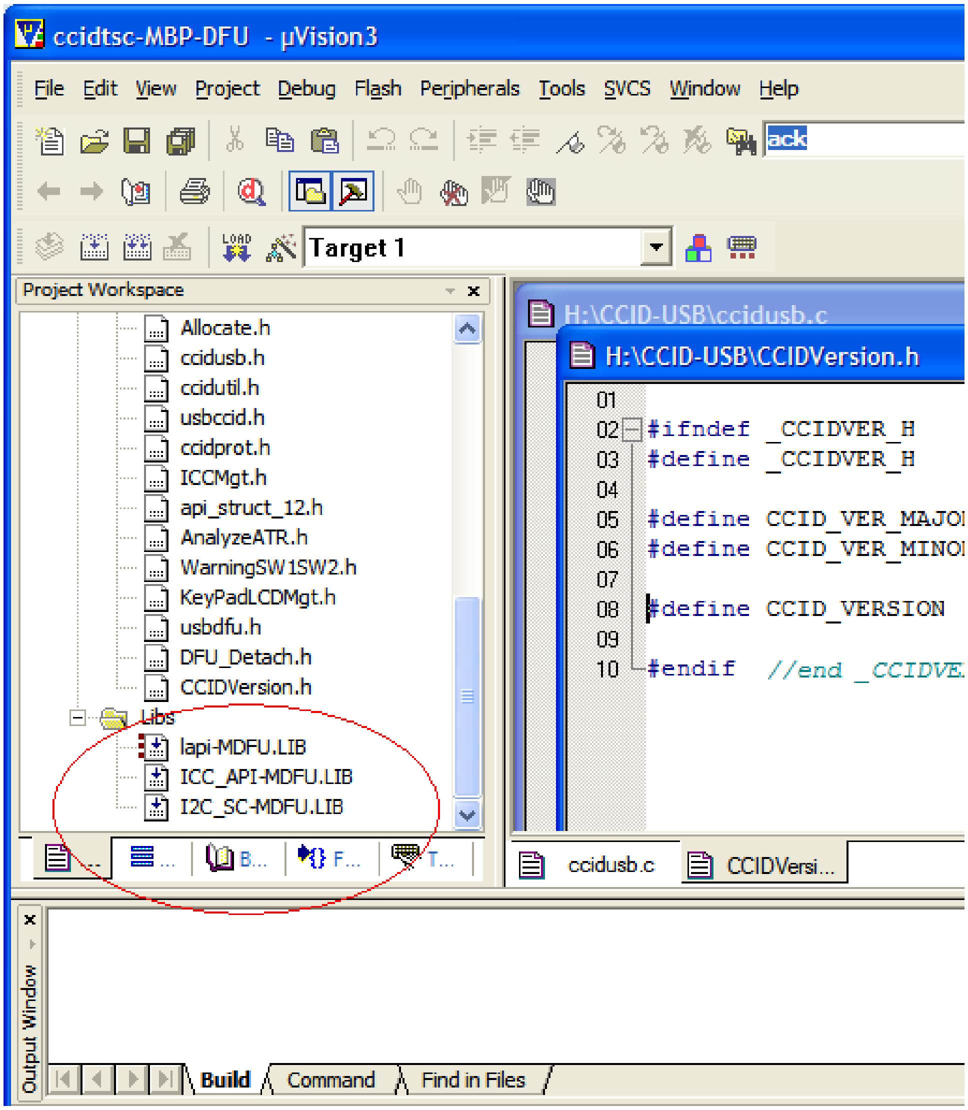
☑ Enable ANSI integer promotion rules

Include Paths: ..\LAPI;..\HAPI\Smart Card

Misc Controls:

Compiler control string: LARGE OPTIMIZE (9,SIZE) BROWSE ORDER INTVECTOR(0x1802) INCDIR(..\LAPI;..\HAPI\Smart Card) DEFINE (EMBEDDED,BIG_ENDIAN,MULTI8010,BPOWER,DFU) DEBUG OBJECTEXTEND

3. The LAPI and HAPI libraries are built specifically for DFU to stay clear of the DFU Flash area. Thus, the CCID+DFU project file is also setup to link to the DFU flavored libraries as shown below:



```
#ifndef  _CCIDVER_H
#define  _CCIDVER_H

#define CCID_VER_MAJO
#define CCID_VER_MINO

#define CCID_VERSION

#endif   //end _CCIDVE
```

The CCID firmware application is divided into functional source code modules as illustrated in Table 1. Customers are encouraged to browse the source files to better understand how they work. Special care should be taken when modifying these source files. Customers are discouraged from modifying code related to the USB CCID class processing found in ccidusb.c and ccidprot.c. Specific parts of the code that may be customized are identified in the following sections.

Table 1: CCID Firmware Source Code Modules

Functional Area	Module
main() and init function	ccidtsc.c
CCID USB layer	ccidusb.c
CCID Protocol Layer	ccidprot.c
Enhanced CCID support for pinpad and LCD	ccidhid.c
Utility functions	ccidutil.c

The USB CCID Class Specification specifies a USB CCID class descriptor, which is used to present all Smart Card Reader features to the host computer. The CCID firmware application employs the TSC ICC HAPI (detailed in the *73S12xxF Smart Card Terminal Controller Family Software User's Guide*) to provide Smart card functionality. The Smart Card HAPI layer is multi-standard compliant and encapsulates a great deal of functionality. This is reflected in the contents of the USB CCID class descriptor as defined in ccidusb.c. The descriptor contents specify, among other things, Short and Extended APDU level of transfer, automatic voltage selection, automatic protocol parameters selection and automatic clock frequency and data rate change.

USB communications are achieved with the use of the USB LAPI functions as described in detail in the *73S12xxF Smart Card Terminal Controller Family Software User's Guide*. The CCID application is a single threaded C program whose main() function is placed in ccidtsc.c. The main() function contains a do..while(1) loop which polls the various external events and performs the necessary processing. First, the main loop calls the USB_OUT_1() LAPI to check for data reception on the CCID bulk command pipe. If a CCID command has been received this is processed in CCID_CommandDispatch() as described in Section 3.4. If no data is present on the bulk pipe as indicated by USB_OUT1() returns RX_PENDING. The firmware then checks for the following:

- The status of the EP0 control pipe using CheckEP0Status() as described in Section 3.2.
- The status of ICC slots using CheckCardStatus() as described in Section 3.3.
- The status of USB state for suspend/reset/resume as described in Section 3.1.

3.2 Suspend and Resume Processing

The functions uReset(), uSuspend() and uResume() in ccidusb.c are callback functions for the USB LAPI that are set up during initialization with a call to USB_Init(). The functions are called by the LAPI subsystem when USB suspend and resume conditions occur on the bus. In the CCID application firmware these functions are used to power down/up the ICC and PIN pad to meet USB suspend power requirements.

The customer application may use these callback functions to power down/up additional components and/or to perform additional suspend/resume processing.

3.3 Endpoint 0 Processing

USB device requests are made over the control pipe on endpoint 0. The status of EP0 is checked by calling the LAPI USB_Status() from within the CheckEP0Status() function in ccidtsc.c LAPI returns an enumerated value indicating what type of request, if any, has been received on the control pipe. Refer to the *73S12xxF Smart Card Terminal Controller Family Software User's Guide* for further information on the USB_Status() LAPI.

The CheckEP0Status() function handles both class device requests (ABORT) and TSC vendor specific device requests detailed below. The customer should not need to modify any of this code and the following is provided for information only.

The CCID_SET_DEVICE_MODE device request is used to set the device mode to either Microsoft CCID mode or TSC enhanced CCID mode. The device request control bytes are defined in Table 2. The firmware sets a global variable called gFirmwareMode. This global is used throughout the firmware to control access to multiple slots and enhanced features. In Microsoft CCID mode (default firmware mode), only one slot (slot 0) is supported and all of the enhanced features such as LCD and PIN pad are disabled.

Table 2: CCID_SET_DEVICE_MODE Device Request

bmRerquestType	bRequest	wValue	wIndex	wLength	Data
01000000	CCID_SET_DEVICE_MODE (0)	Device Mode*	0	0	none

* 0 = Microsoft CCID mode, 1 = TSC Enhanced CCID mode

3.4 Card Slot Status Processing via (interrupt) Endpoint 2

Thestatus of the card slots is checked by calling the ICC_Status() HAPI from within the CheckCardStatus() function in ccidtsc.c. The USB CCID Class Specification describes how the host computer is notified of card slot insertions and removals and the customer should not need to modify this code.

3.5 CCID Command Processing

CCID bulk command processing is performed in the CCID_CommandDispatch function contained in ccidusb.c. This function contains a large switch statement with cases for all of the CCID Bulk command-response pairs detailed in the USB CCID Class Specification. This source module also contains all the message-specific processing, including accesses to the ICC HAPI. The customer is strongly discouraged from modifying any of this code, since this code has been thoroughly tested using the Microsoft HCT and DTM test tools and is standard-compliant. However the customer may modify the Mechanical and Escape Command processing as detailed below.

3.6 Command Error Codes

CCID bulk command is processed and executed. An error code is returned for each command according to the USB CCID Class Specification. Teridian added a few more error codes, mostly following an escape command. The error codes provided in the accompanied CCID source code are defined as follows:

```
// the following #defines are used on the bError byte
#define CMD_ABORTED           0xff
#define ICC_MUTE              0xfe
#define HW_ERROR              0xfb
#define BAD_ATR_TS            0xf8
#define BAD_ATR_TCK           0xf7
#define UNKNOWN_ESC_CMD       0xff
#define KEYPAD_TIMEOUT        0xf9
```

3.7 Mechanical Features Porcessing

Since the 73S12xxF evaluation board does not have any mechanical features, no specific mechanical processing is performed in the CCID firmware. Customers who wish to develop readers with mechanical features are directed to the CCID_Mechanical() function in ccidusb.c. The decoding software for this function has switch cases for LOCK_CARD and EJECT_CARD and has provided stub functions (LockCard() and EjectCard()) for these two cases. The stub functions simply return success and the customer should place any mechanical processing code in these functions.

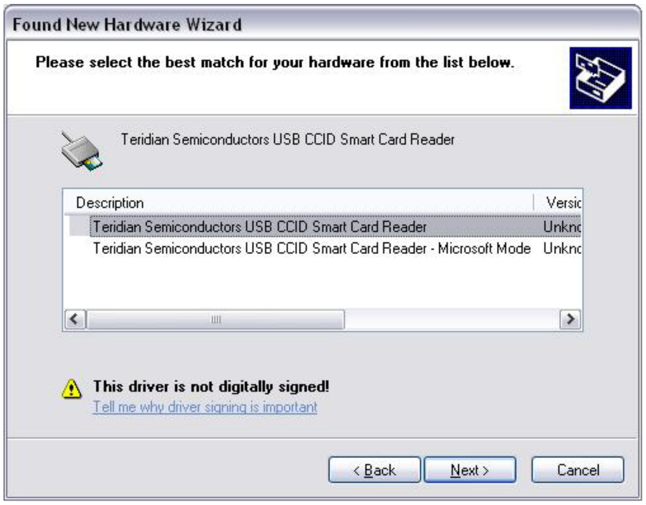
The USB *Device Class Specification for USB Chip/Smart Card Interface Devices Specification*, Revision 1.1 only supports LOCK_CARD and EJECT_CARD, although other mechanical functions are defined. The customer should add any other required mechanical functions to the switch statement contained in CCID_Mechanical() and add handler functions as required.

3.8 Escape Command Processing

The USB CCID Class Specification provides the PC_to_RDR_Escape bulk command to allow card reader vendors to send and receive custom data between the host computer and reader. The CCID application firmware provides a handler function in ccidusb.c called CCID_Escape(). This handler can be used to implement additional customization features such as LCD and Keypad as defined outside the scope of CCID specification. Any customer-specific escape processing should be added to this handler function. The application software may send and receive escape data via both the Microsoft and TSC driver as described in Section 5.2.

4 Driver Selection and Installation

When the 73S12xxF evaluation board containing the CCID firmware application is plugged into the USB port of a Windows host PC, the Windows Add New Hardware Wizard is launched. The wizard prompts the user to install a driver for the newly found hardware. The CCID application firmware operates with both the standard Microsoft CCD driver, usbccid.sys, and the TSC enhanced CCID driver, ccidtsc.sys. The software installation disk contains Windows .inf file for both of these drivers. If the user browses to the directory containing the .inf files, the wizard displays the dialog shown in Figure 2. The first choice, highlighted is the ccidtsc.sys driver; the second choice is Microsoft's usbccid.sys.



Please select the best match for your hardware from the list below.

Teridian Semiconductors USB CCID Smart Card Reader

Description	Versic
Teridian Semiconductors USB CCID Smart Card Reader	Unknc
Teridian Semiconductors USB CCID Smart Card Reader - Microsoft Mode	Unknc

This driver is not digitally signed!

Tell me why driver signing is important

Figure 2: Add New Hardware Wizard Showing Microsoft and TSC CCID Drivers

The features provided by Microsoft's usbccid.sys are detailed fully in *Microsoft Class Drivers for USB CCID Smart Cards* (http://www.microsoft.com/whdc/device/input/smartcard/USB_CCID.mspx).

The TSC enhanced CCID driver provides CCID compliant APDU level transfer of data and in addition addresses many of the shortcomings of the Microsoft driver, specifically:

- Support for multiple reader slots. See Section 5.1.
- PIN pad support. See Section 5.2.
- LCD support. See Section 5.2.
- Vendor/device-specific string name support in the device manager.
- Support for mechanical features.
- DFU Device Class (0xFE)

Once the TSC enhanced CCID driver is loaded it issues a CCID_SET_DEVICE_MODE device request to enable the enhanced features in the firmware.

The TSC enhanced CCID driver uses the following registry settings, set from the ccidtsc.inf installation file:

- HKLM,System\CurrentControlSet\Services\ccidtsc\Parameters,SelectSuspendEnable
  This setting is used to control whether the driver attempts to selectively suspend the card reader in the case of 3 seconds of bus traffic inactivity. This should be left at the default state of 0, disabled, until the hardware supports remote wakeup capability.
- HKLM,System\CurrentControlSet\Services\ccidtsc\Parameters,AbortEnable
  This setting is used to control whether the driver will activate the CCID class ABORT sequence specified in the USB Device Class Specification in the event of bulk command/response timeout. The ccidtsc.inf file is delivered with ABORT disabled for development purposes. Once the customer has developed a stable firmware application this feature may be enabled.

5 Smart Card Application

A Windows Smart Card Application uses the Win32 Smart Card to access the TSC 73S12xxF based reader. The APIs operate with both the Microsoft and the TSC CCID drivers in an identical manner. For a full Windows Smart Card API reference see the *MSDN Library – April 2005. Platform SDK: Smart Card*.

The application program first establishes a context to the Smart Card Resource Manager using the API ScardEstablishContext(). Then the ScardConnect() API is called to open a handle on an individual smart card basis. This handle is then used for subsequent operations on the Smart Card, such as power on/off, data transmission and reception and protocol selection.

The TSC 73S12xx Software Installation disk contains a sample Smart Card application called SmartCard, which demonstrates the use of the lower-level Windows Smart Card APIs. The application uses the APIs in Table 3 to demonstrate card/device related access. Refer to the code for example uses of the APIs.

Table 3: Windows Smart Card APIs used in the SmartCard Sample Application

API	Description
ScardEstablishContext	Establishes the resource manager context.
ScardListReaders	Receives a list of Smart Card Readers attached to the system.
SCardConnect	Establishes a connection between the application and a smart card contained by a specific reader.
SCardReconnect	Re-establishes an existing connection between the calling application and a smart card.
ScardTransmit	Sends a service request to the smart card and expects to receive data back from the card. Used to send ADPUs.
SCardControl	Used for direct control of the reader via IOCTL method.
ScardGetAttrib	Used to get the current reader attributes for the given handle e.g. ATR string, IFSD, data rates, vendor strings.
ScardStatus	Provides the current status of a smart card in a reader.
ScardDisconnect	Terminates a connection previously opened between the calling application and a smart card in the target reader.
ScardReleaseContext	Releases the resource manager context.

The enhanced TSC features such as multiple card slots, PIN pad and LCD also use the handle returned from ScardConnect() API. Multiple slot operation is achieved as described in Section 5.1 of this application note. PIN pad and LCD services are provided through a TSC-supplied DLL. The application links to this DLL and calls the DLL services, passing in the Smart Card handle as a parameter. The services are detailed in below sections of this application note. It is noted that these enhanced services are provided by the TSC enhanced CCID driver, ccidtsc-*.sys. If the Microsoft CCID driver is installed, these enhanced function calls will fail.

The CCID application firmware will automatically control the LED0 indicator. Whenever the card in slot 0 is activated with a valid ATR, the LED0 indicator will automatically turn on solidly. Whenever the card is deactivated (by card removal or card deactivation via command or fault), the LED0 will be turned off. The LED ESC command can manually control the LED0, but the firmware will always override the LED ESC control setting upon card activation or deactivation.

5.1 Multiple Slot Operation

When a reader is selected for the first time, the application program retrieves slot information from the driver to determine the number of slots the selected reader supports. When changing active slot on a multiple-slot reader, the application program calls the ScardControl() API with the IOCTL code set to

IOCTL_CCIDTSC_SELECT_SLOT. The new slot number is contained in the first member of the I/O buffer byte array and is a zero-based slot index. A smart card should be already inserted in the new slot, otherwise the TSC enhanced CCID driver will fail the API. After a successful slot change, the application program is ready to communicate normally with the smart card in the new slot.

When the user clicks on the Connect button, the application program calls ScardConnect() API to power on the smart card. From this point on, all subsequent card operations refer to the card in the new slot.

Two additional IOCTL options are provided for slot management. They are IOCTL_CCIDTSC_GET_NO_SLOTS and IOCTL_CCIDTSC_GET_SELECTED_SLOT. While initializing for a selected reader, the application program uses these options to retrieve the number of slots available in the reader, and the currently selected slot. The application program calls these options in the same manner using the ScardControl() API. The return value is returned in the first byte of the I/O buffer. Example use of these IOCTL options is demonstrated in the btnSelecteReader_Click() function in the SmartCard sample application.

5.2 Sending Escape Commands

In order to send/receive escape commands to/from the reader the application programmer calls the ScardControl() API with the IOCTL set to IOCTL_CCID_ESCAPE. The ScardControl() API has transmit and receive buffers to send and receive data. See the *MSDN Library – April 2005. Platform SDK: Smart Card* for a full definition of the use of this API. It is noted that this IOCTL may be used with both the Microsoft CCID driver and the TSC enhanced CCID driver. However, for the Microsoft driver, this feature must be enabled via a registry setting as detailed in *Microsoft Class Drivers for USB CCID Smart Cards* (http://www.microsoft.com/whdc/device/input/smartcard/USB_CCID.mspx).

The current version of CCID firmware application support escape commands in the format as described in table below. The supported extended features are LCD and PinPad controls but more can be added and/or modified:

bMessageType	dwLength	bSlot	bSeq	RFU	T1B1B2B3B4 ..abData
6Bh	**xxxxxxxx**	**xx**	**xx**	**xxxxxx**	**n1...nx**
1Byte	4Bytes	1 Byte	1 Byte	3 Bytes	1 - xBytes
Offset = 0	Offset = 1	Offset = 5	Offset = 6	Offset = 7	Offset = 10

Since these extended features are mostly outside of the Smart Card functions; thus, references to smart card's slot number is typically RFU. These additional features are supported as defined in T1 column below. The Parameter field (B1B2B3) is a raw text format that can be used to control the specific non-smart card features.

Table 4 defines the extended features.

Table 4: Extended Features

T1	B1	B2	B3	B4	Any Other Bytes
1Byte	1Byte	1Byte	1Byte	1Byte	Up to 255 bytes
0x31	0x00 = CLEAR	0x00 = RFU	0x00 = RFU	0x00 = RFU	LCD Control
	0x01 = POSX	0x00 = RFU	rc – r=Row/c=Column	0x00 = RFU	
	0x02 = DISPLAY	0x00 = RFU	i - Index	0x00 = RFU	
	0x03 = WRITE	0x00 = RFU	0x00 = RFU	l = length	
0x32	0x00 = Config	WaitTime	rc – r=Row/c=col	0 - RFU	KeyPad Control
	0x01 = KeyWait	WaitTime	rc – r=Row/c=col	0 – RFU	
	0x02 = WaitKeys	WaitTime	# Keys	0 – RFU	
0x33	0xX1 = ISO Mode 0xX0 = EMV Mode	RFU	RFU	RFU	EMV Level 1 Test Automation
	0x00 = not a test mode 0x10 = MasterCard® (Cetecom lab) 0x20 = VISA® (FIME, RFI labs) 0x40 = VISA (ICTK lab)	RFU	RFU	RFU	
0x34	0x00 = Turn off LED0	RFU	RFU	RFU	LED0 turned off.
	0x01 = Turn on LED0		Brightness: 0x01 = Dim (2mA) 0x02 = Normal (4mA) 0x03 = Bright (10mA)	RFU	LED0 is turned on at this brightness.
	0x02 = Enable LED0 blinking.	Blinking rate, in 100 ms. Slowest rate is 25.5 seconds.	Brightness (same as above)	RFU	LED0 will be blinking at this rate and this brightness until a card is removed or deactivated and LED0 will be turned off. When a card is activated, the LED0 will turn on solidly.

T1	B1	B2	B3	B4	Any Other Bytes
0x41	RFU	RFU	RFU	RFU	DFU Detach
	0x00 = Config	0xYy where: Y = Baud, * y = NoRetries	WaitTime This time specifies a number in seconds that has passed before a retry is attempted.	TxRxBuffSize Maximum Buffer size. Note:This field is RFU for now due to limited RAM space. CCIDUSB FW defines the maximum Rx/Tx size for Serial Interface according to available RAM space.	Assuming 8 Data bit, No Parity, No Xon/off.
0x50	0x01 = Transmit 0x02 = Receive 0x03 = Transmit then Receive 0x04 = Receive then Transmit	TxSize RFU TxSize TxSize	RFU RxSize RxSize RxSize	RFU RFU RFU RFU	Data Data Data

* Valid values (Y) for baud rates at CPU CLK=24MHz are specified as follows:

0x3y = 4800
0x4y = 9600
0x5y = 14400
0x6y = 19200
0x7y = 28800
0x9y = 57600
0xBy = 125000
0xCy = 250000
0xDy = 375000

Any other values of Y will revert to default baud of 9600.

Value (y) at CPU CLK=24MHz for the number of retries can be anything up to 15 (0xYF). An error code will be returned when the number of retries is exhausted.

When the command is executed successfully, the device's response will be as follows:

bMessageType	dwLength	bSlot	bSeq	bStatus	bError	Parameter	abData
83h	**xxxxxxxx**	**xx**	**xx**	**xx**	**xx**	**T1B1B2B3B4**	**n1...nx**
1Byte	4Bytes	1 Byte	1 Byte	1 Bytes	1 Bytes	5 Bytes	1 – x Bytes
Offset = 0	Offset = 1	Offset = 5	Offset = 6	Offset = 7	Offset = 8	Offset = 9	Offset = 13

When the command is executed unsuccessfully, such as in the case of Serial Interface, the device's response will be as follows:

bMessageType	***dwLength***	***bSlot***	***bSeq***	***bStatus***	***bError***	***Parameter***
83h	**xxxxxxxx**	**xx**	**xx**	**xx**	**xx**	**T1B1B2B3B4**
1Byte	4Bytes	1 Byte	1 Byte	1 Bytes	1 Bytes	5 Bytes
Offset = 0	Offset = 1	Offset = 5	Offset = 6	Offset = 7	Offset = 8	Offset = 9

Where bStatus will show current Smart Card's status (0x4X) and bError will show the error code according to CCID spec. The Parameter field will show the values of the original command.

5.2.1 EscapeCommand – LCD Control: 0x31

T1 = 0x31, LCD Control.
B1 = 0, Clear LCD.
= 1, Position cursor on LCD, B2 = '00', B3 = row/column (maximum is 2x16).
= 2, Display selected canned message on LCD at current cursor, B2 = '00', B3 = message index, B4 = '00'.
Message index (B3) is defined as follows:
0x00 – " ENTER SC PIN "
0x01 – " ENTER NEW PIN "
0x02 – " CONFIRM NEW PIN"
0x03 – " RFU "
More messages can be added if firmware program space permits.
= 3, Write string of characters to LCD from cursor, B2 = '00', B3 = '00', B4 = length of string.

When B1=0x03, the host application should be responsible for making sure the string length is within the hardware size limit (i.e. 2x16).

This escape command should never return an error. Field *abData* should always be empty.

5.2.2 EscapeCommand – KeyPad Control:0x32

T1 = 0x32, KeyPad Control.
B1 = 0, Configure KeyPad, B2 = Wait time (up to 255 seconds, 0=indefinite wait), B3 = rows/columns (maximum 6 rows x 5 columns).
= 1, KeyWait, B2 = Wait time (up to 255 seconds, 0=indefinite wait), B3 = row high nibble, column low nibble of key.
= 2, Wait for keys, B2 = Wait time (up to 255 seconds, 0=indefinite wait), B3 = number of keys to wait to be pressed.
Note: *B2 will be used as current Waittime for all 3 KeyPad control commands.
The Keypad Control response will return each key pressed in the row-nibble/column-nibble format in B2....Bn parameters field in the order that the key(s) is/are pressed. The row/column position starts from 0. When B1=0x01 or 0x02, the response will be in the same format in which B2 denotes the number of keypress and B3..Bn denote the actual keypress position(s) (row/column) in the order that the keys are pressed (FIFO).

When B1 of the keypad control = 0x01 or 0x02, the response will be as follows:

bMessage Type	dwLength	bSlot	bSeq	bStatus	bError	abData
83h	**xxxxxxxx**	**xx**	**xx**	**xx**	**xx**	**32 01 (or 02) Z K1 K2…Kn**
1Byte	4 Bytes	1Byte	1Byte	1Byte	1Byte	nBytes
Offset = 0	Offset = 1	Offset =5	Offset = 6	Offset = 7	Offset = 8	Offset = 9

Z represents the number of key presses. K1, K2..Kn represents the key positions that were pressed.

5.2.3 Escape Command – EMV Level 1 Test Control: 0x33

T1 = 0x33, EMV Level 1 Test Automation Control

B1 = 0xX1, (least significant nibble) – ISO mode (where default IFS size would be different than EMV, broader range of FiDi values are accepted and a few other differences);
B1 = 0xX0, (least significant nibble) – EMV mode. When in EMV mode, acceptance of certain ATR is more restricted. The IFS size will be 0xFE (as opposed to 0x20 as specified in ISO). In EMV test mode, different value is used in determining if a test suite used for EMV testing is MasterCard, VISA following specification version 4.0 or VISA following specification version 4.1. The loopback implementation is different for each different test mode. Value of this byte is classified as follows:

0x00 – Not a test mode.
0x10 – MasterCard's loopback (Cetecom).
0x20 – VISA's loopback(FIME, RFI).
0x30 – VISA's loopback(ICTK).

In order to invoke the EMV Level 1 Automation Test mode, this escape command should be sent first to setup the firmware prior to sending the ICC_PowerOn command. The ICC_PowerOn command then follows. If the response from the ICC_PowerOn command is successful (with an ATR), the host should then send the BlockTransfer (0x6F) command to start the loopback test; otherwise, it should send an ICC_PowerOFF command to conclude the test. Upon setting the device in EMV Test mode, if Slot 0 is currently activated, it will be deactivated by the device in order to prepare for testing.

Block Transfer (0x6F command) in EMV Test Mode:

When a card is activated in EMV TEST mode for either VISA or MasterCard environment, as described above, it is not required to include the APDU command as part of the Block Transfer packet or the content of the Block Transfer command will be ignored. The VISA and MasterCard PSE test environment is written specifically for a specific test lab, the firmware will handle the loopback APDU command. The host only needs to facilitate and initiate this test mode by sending an empty Block Transfer packet after a successful activation of the card (ICC_PowerOn in EMV and either VISA or MCI mode). After the device completed the Block Transfer command, i.e. the host receives the response for the Block Transfer command, the host should send an ICC_PowerOff command to conclude the EMV Level 1 Test.

5.2.3.1 EMV LEVEL I Certification Tests

The EMV compliant test suite follows its specification written for Payment System Environment.
There are several test labs, listed on the EMVco.com website, qualified to perform these tests. There are two Protocol test suites that can be used to qualify for EMV Level I compliance. In other words, passing either one of these suites will qualify as EMV Level I compliance. The CCID firmware code is written to link to the two TSC libraries (LAPI and HAPI) that comply with both tests. However, since each lab has its own test scripts and the test scripts are different according to the lab's setup, the host application layer must be written, and catered, specifically for each test lab's requirements. The following two subsections describe both loopback tests that must be written on the host side's application in order to invoke the appropriate test.

Before Starting EMV Test Mode:

When a card is inserted in the slot, the host's Smart Card Resource Manager automatically activates it as part of its scanning/polling process (this is true for both Microsoft's Generic CCID driver and TSC's custom driver). These steps below outline a procedure to start the EMV Automation test. Follow these steps in their exact order to guarantee the correct test session:

1. From the HostU: Connect the device via a USB port from the host computer running either Windows XP or Windows Vista. Wait until the driver is successfully loaded and enumerated (driver appears in the Device Manager window).
2. UFrom the HostU: Bring up the C# application, select the TSC device. Then select option 'EMV Level 1 Test Automation' and the appropriate test environment under 'Mode' to send the ESCAPE command to the device to be ready for EMV Test Mode (VISA or MCI). Accept the default value or enter a desired delay value (in seconds) to space out the loopback test as required by the EMV test environment/test suite. For USB interface, with Windows XP, any value of 10 seconds or higher is desirable.
3. From the EMV Test Side: Select to start running the multi-test mode (or automatic test mode) to start the EMV test on the card.
4. Both the card/EMV Test side and the host C# + device should be synchronized and test should start at this point.

After the testing is completed, reset the 73S12xxF device to take it back into ISO mode, or out of EMV test mode.

5.2.3.2 EMV Test Mode

An EMV test (or session) is defined to be commands that run from Activation of the card to Deactivation of the card. A Block Transfer may or may not happen in the session depends on the card's ATR response. The host may setup the EMV PSE test environment via the Escape (0x33) command. The first parameter byte (B1) of the Escape command needs to specifically tell whether a test mode is invoked and if so, it should be invoked using MCI or VISA test environment. Please review the Escape – EMV Level 1 Test Control section for details about this test mode. Following the successful PowerOn command, the host needs to send the Block Transfer command. This command packet can be with or without the APDU command (command length = 0). Since the APDU command will be ignored by the firmware in test mode, the Block Transfer can be empty or with any data The test loopback will be handled by the firmware and upon finishing up one test, the firmware will respond to the host with status of whether a test session run with successful return code or not. NOTE : an unsuccessful return code may or may not be a failed test. The test verdict (test passed or test failed) is determined only on the card side. The following flow chart depicts the minimal coding required on the host application to invoke EMV PSE test environment

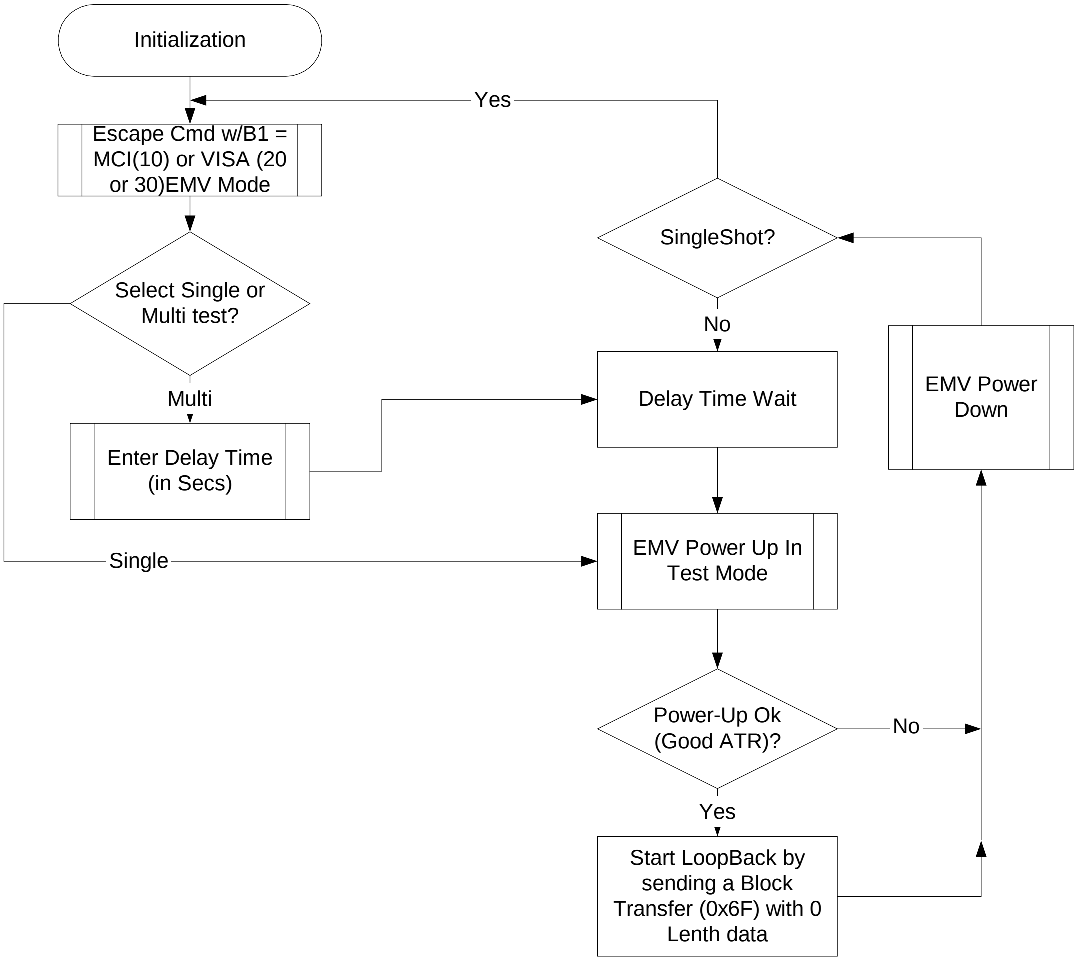

Initialization
Yes
Escape Cmd w/B1 = MCI(10) or VISA (20 or 30)EMV Mode
SingleShot?
Select Single or Multi test?
No
Multi
Delay Time Wait
EMV Power Down
Enter Delay Time (in Secs)
Single
EMV Power Up In Test Mode
Power-Up Ok (Good ATR)?
No
Yes
Start LoopBack by sending a Block Transfer (0x6F) with 0 Lenth data

Figure 3: Add EMV Test Mode

5.2.3.3 MasterCard Loopback Test

TSC used CETECOM test lab (in Germany – as listed on EMVco website) and FIME test lab (in France – as listed on EMVco website); who used MCI test suite to qualify their EMV Level I test services. The loopback flowchart below is specific to both FIME and Cetecom's Level I Protocol test scripts. This flowchart shows in details flow of the entire MCI test suite with coding to be done on both host (invoking the test) and the device side (manages all aspect of smart card's EMV test). Source code is also included in the CD ROM.

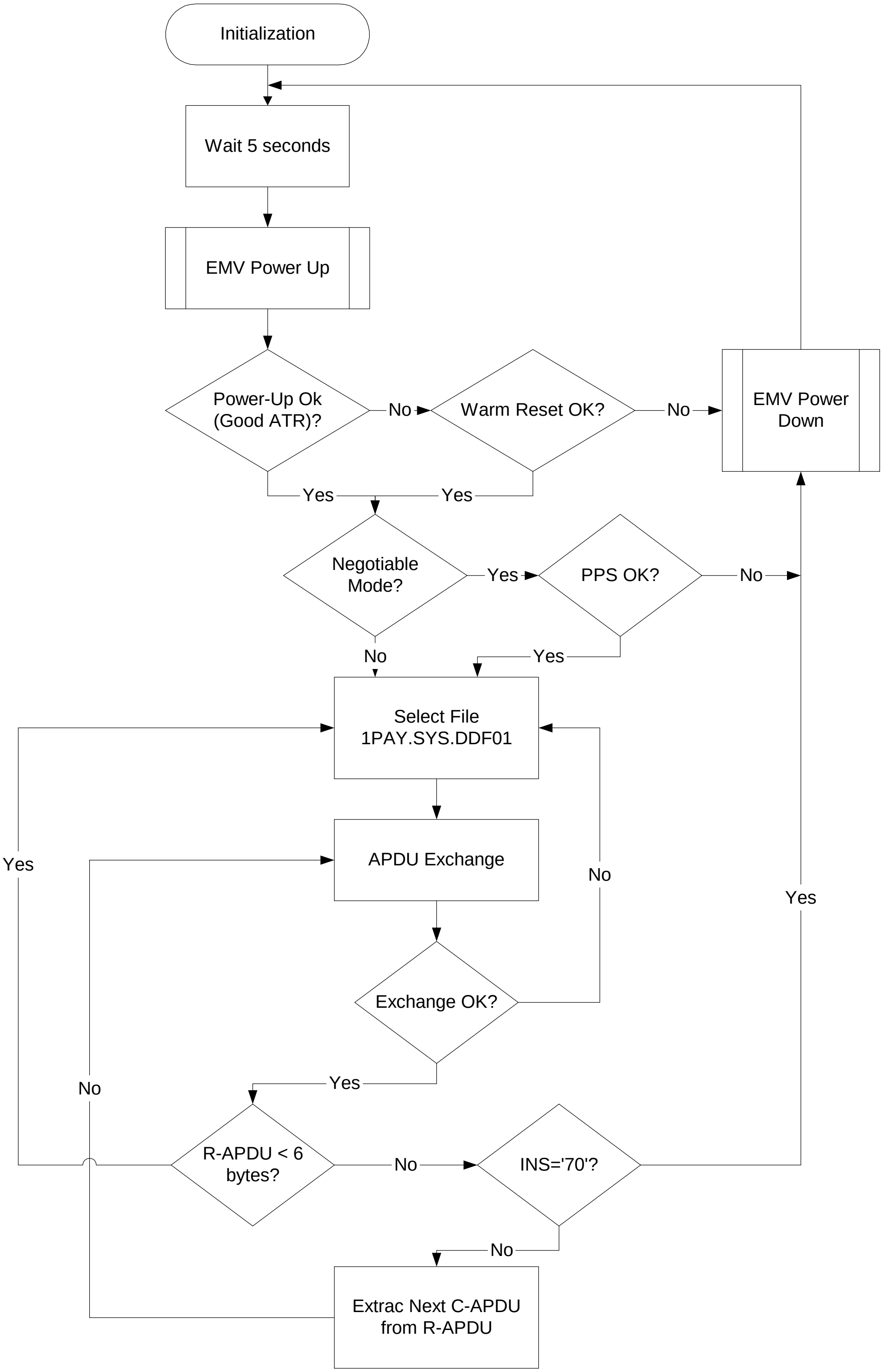


Figure 4: MasterCard Loopback Test

5.2.3.4 VISA-1 Loopback Test

TSC used RFI Global test lab (in UK – as listed on EMVco website); who used VISA test suite to qualify their EMV Level I test services. The Loopback flowchart below is specific to RFI's test scripts. Source code is also included in the CD ROM.

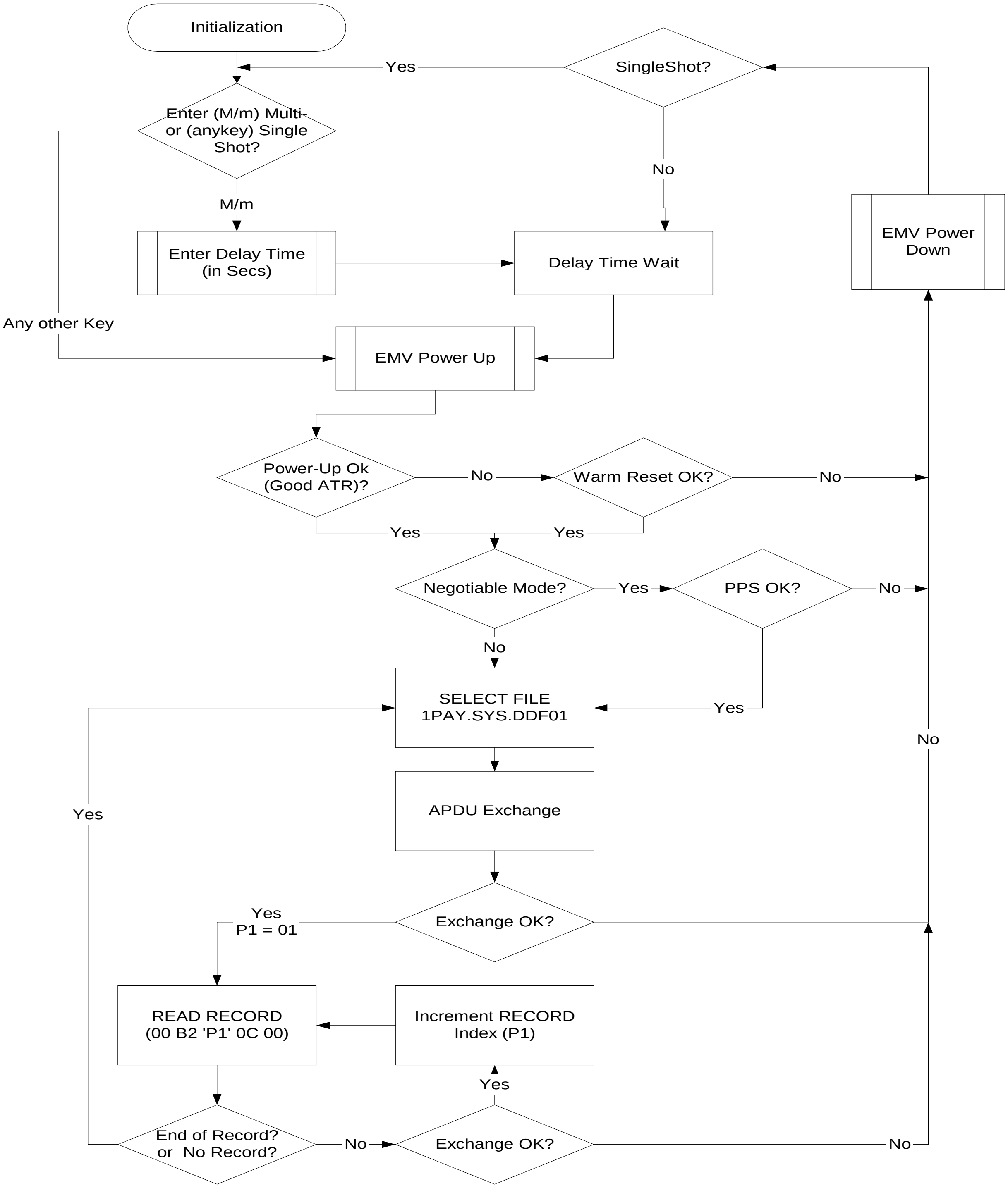


Figure 5: VISA-1 Loopback Test

5.2.3.5 VISA-2 Loopback Test

TSC used ICT-K test lab (in Korea – as listed on EMVco website); who used VISA test suite version 4.1 to qualify their EMV Level I test services. This CD ROM includes source code that implements the Loopback flowchart below.

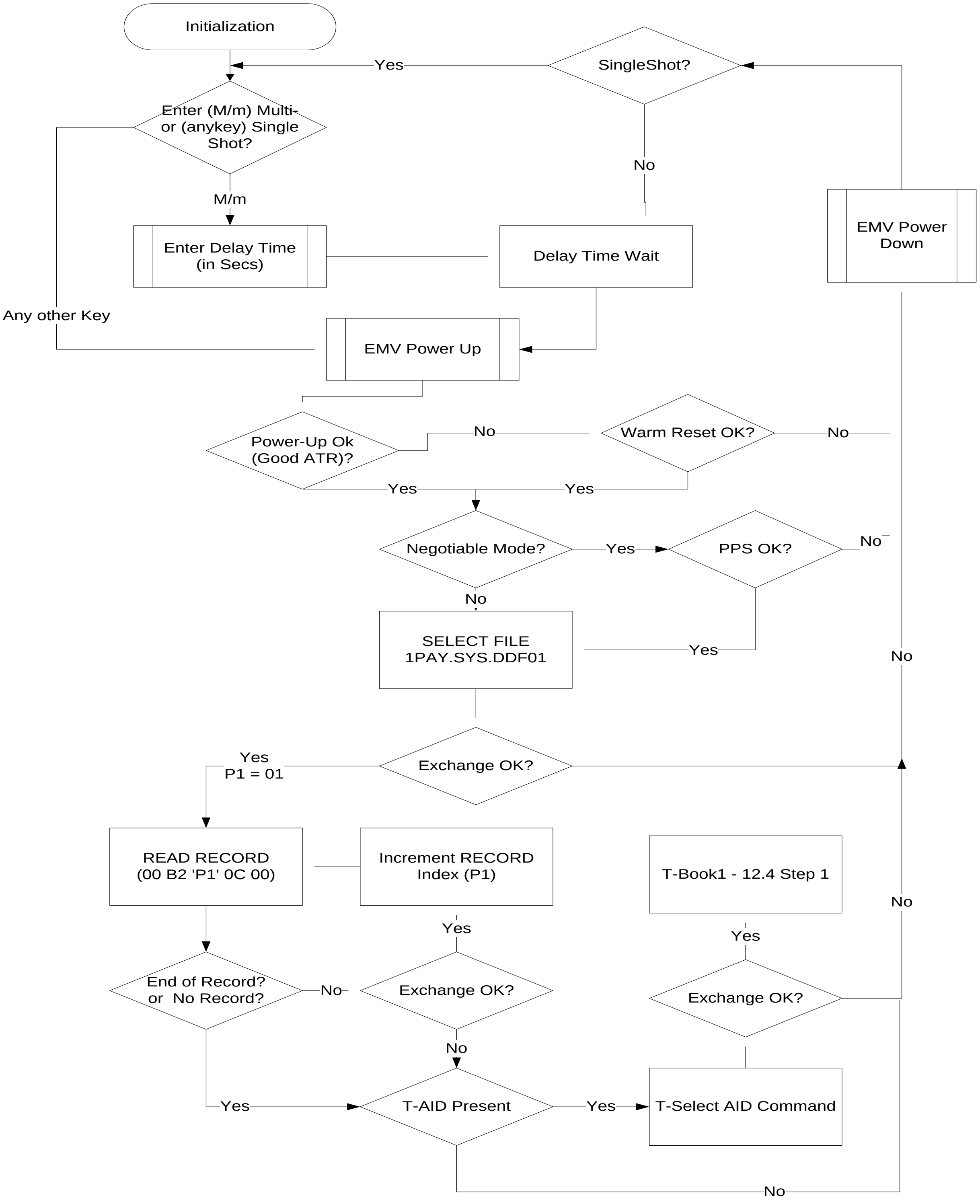


Figure 6: VISA-2 Loopback Test

5.2.4 EscapeCommand – LED Control: 0x34

This escape command can be used to control LED0. The default purpose is to use LED as a smart card ready indicator (card activated and powered on), but it can be used for any other indicator function. When the blinking setting (such as blinking rate) is set, it will stay globally set until the next setting or card deactivation in slot 0. For example, if an escape command is sent from the host to set the blinking to be 500 ms apart, it will stay at 500ms until it is changed by the next escape command modifying the rate or turning the led solidly off or on. The parameters for this command are executed as follows:

T1 = 0x34, LED Control command.

B1 = 0x00,
Turn off LED0.
= 0x01,
Turn ON LED0
= 0x02,
Enable blinking of LED0. When a card is removed, LED0 will be turned off. When a card in slot 0 is reactivated LED0 will be turned on but will not be blinking.

B2 = #. Time in 100 ms that indicates blinking rate. Only applicable when B1 = 0x02.

B3 = Brightness control. When B1 <> 0xX0, use this value to set level of LED's brightness as indicated.

This escape command should never return an error.

5.2.5 EscapeCommand – DFU Detach: 0x41

Upon receiving this ESCAPE command from the host, the firmware will complete the command with a successful response. It then erases the checkcode in preparing for the DFU Boot Loader to take over. A Soft_Reset will follow to restart the device, which will cause a USB power cycle. See the *73S12xxF Smart Card Terminal Controller Family. DFU Application Note* for more details.

5.2.6 EscapeCommand – Serial Interface Control: 0x50

T1 = 0x50, Serial Interface Control.

B1 = 0, Serial Interface Initialization. Supported baud rates are based on the device running the CPU clock. The rates shown here are assuming the CPU clock is running at 24Mhz. The Serial rates are specified in api_struct_12.h.

= 1, Transmit data through Serial Interface. The number of bytes to be transmitted is specified in TxSize (B2).
Wait until data is completely transmitted before sending response.
= 2, Receive data from Serial Interface. Number of expected bytes is specified in RxSize (B3).
= 3, Transmit then Receive through Serial Interface. First, transmit data through Serial Interface. The number of bytes to be transmitted is specified in TxSize (B2). After data is transmitted, expect to receive the number of bytes as specified in RxSize (B3) from Serial Interface. If there is an error during either transmission or reception, the command will be abandoned and error code will be returned to the host.
= 4, Receive data then Transmit the given data through Serial Interface. First, receive data from Serial Interface. The number of expected bytes to receive is specified in RxSize (B3). Once all bytes, as specified in RxSize (B3), are received, transmit the TxSize (B2) data bytes through Serial Interface. If there is an error during either transmission or reception, the command will be abandoned and error code will be returned to the host.

B2 = Number of bytes to be transmitted.
B3 = Number of bytes to be received.

//Please note: Serial data rates are dependent on the device's CPU Clock rate. The following table
//Shows supported rates based on specific CPU Clock:

//		CPU CLK		
//Rates	3.69Mhz	6Mhz	12Mhz	24Mhz
//600	OK	N/A	N/A	N/A
//1200	OK	OK	OK	N/A
//2400	OK	OK	OK	N/A
//4800	OK	OK	N/A	OK
//9600	OK	N/A	OK	OK
//14400	OK	OK	OK	OK
//19200	OK	N/A	N/A	OK
//28800	OK	N/A	OK	OK
//38400	OK	N/A	N/A	N/A
//57600	OK	N/A	N/A	OK
//115200	OK	N/A	N/A	N/A
//125000	N/A	N/A	OK	OK
//250000	N/A	N/A	N/A	OK
//375000	N/A	N/A	OK	OK

6 Acronyms

APDU	Application Protocol Data Unit
API	Application Programming Interface
ATR	Answer To Reset
COM	Communication Port
DFU	Device Firmware Upgrade
DTK	Development ToolKit
EMV	Euro, Master, Visa
HAPI	High Level API
HCT	Hardware Compatibility Test
ISO	International Standards Organization
ISP	In-System Programming
JICSAP	Japan Ic Card System Application council
LAPI	Low Level API
LCD	Liquid Crystal Display
PC	Personal Computer
PIN	Personal Indentification
RAM	Random Access Memory
ROM	Read Only Memory
TSC	Teridian Semiconductor Corporation
USB	Universal Serial Bus
WHQL	Windows Hardware Quality Lab

7 Related Documentation

The following 73S12xxF documents are available from Teridian Semiconductor Corporation:

71S1215F Data Sheet
71S1217F Data Sheet
73S12xxF Smart Card Terminal Controller Family Software User's Guide
73S12xxF Evaluation Board User's Guide
Teridian Flash Programming Tool
73S1215F, 73S1217F Boot Loader – DFU Class Firmware Application Note
73S1215F, 73S1217F Windows XP 32 USB CCID and DFU Drivers Installation Guide
73S1215, 73S1217F CCID Application Note

8 Contact Information

For more information about Maxim products or to check the availability of the 73S12xxF, contact technical support at www.maxim-ic.com/support.

Revision History

Revision	Date	Description
2.0	1/6/2009	Documents Library Rev. 4.00(DFU), Code/Firmware Rev. TSC12xx.2.00.
2.00	4/27/2009	Added DFU build/project description. The DFU Boot Loader is a new feature added to this release. Added a new escape command to indicate card events by manipulating LED0.
2.10	9/14/2009	Added a paragraph at the end of Section 5 about LED0 indicator control. In Table 4, rewrote the 0x034 Any Other Bytes description. Rewrote Section 5.2.4, EscapeCommand – LED Control: 0x34. Change document title from *73S1217F, 73S1217F CCID Application Note* to *73S1217F, 73S1217F USB CCID Guidelines*.