AN1527
APPLICATION NOTE

DEVELOPING A USB SMARTCARD READER WITH ST7SCR

by Microcontroller Division Applications

INTRODUCTION

This document describes a firmware implementation developed by STMicroelectronics for a USB Smart Card Reader. This firmware is for the ST7SCR microcontroller and can be used with the associated Smart Card Reader board available from STMicroelectronics.

It is divided into 4 parts:

– Universal Serial Bus (USB) management: This is the USB library which manages the USB hardware.

– Chip Card Interface Device (CCID) implementation: This contains high level functions for using the USB in compliance with CCID specifications (messages for Bulk-in, Bulk-out, interrupt and class requests)

– Interface Device (IFD) implementation: This contains high level functions for ISO 7816 specification implementation and the smartcard-specific command interpreter.

– 7816 UART Smartcard Interface (CRD) management: This contains low level functions for the hardware management of the 7816 UART Smartcard Interface (CRD).

The main loop polls USB transactions (functions in Ccid_usb.c) using a state machine process:

– USB_Polling() function from library for USB low level and endpoint 0 management.

– Receive USB Bulk-out message (CCID part).

– Execute the function corresponding to the Bulk-out message (IFD part).

– Send USB interrupt in message (CCID part) if necessary.

– Send USB Bulk-in response (CCID part).

The Bulk-out/in messages are managed by a state machine. When the Bulk-out message reception is completed, the message function is executed and returns only after the completion of the action to be done in the IFD part. As this may take several 10 ms (e.g. ATR reading), the response is sent to the host by Bulk-in message. The Bulk-in/out messages are received in several transactions with a single transaction at every main loop if the USB endpoint is available.

If there are any interrupt messages, they are sent just before the Bulk-in message is sent.

Specific CCID class requests are managed by the setup management function in the USB library (USER_USB_setup() function).

1 CCID IMPLEMENTATION

1.1 GENERALITIES

The Chip Card Interface Device (CCID) firmware implementation conforms to the “Universal Serial Bus Device Class Specification for USB Chip/Smart Card Interface Devices” revision 1.0. It contains two files:

– Ccid_usb.c: This file contains the USB - CCID interface functions used to manage the state machine that generates Bulk-in/Bulk-out messages.

– Ifd_ccid.c: This file contains the PC_to_RDR_xxx() and RDR_to_PC_xxx() message functions for the CCID.

The following commands are not supported but can be added easily:

– PC_To_RDR_Secure,

– PC_To_RDR_SetDataRateAndClockFrequency,

– PC_To_RDR_ToAPDU,

– PC_To_RDR_Mechanical.

This CCID part uses a 271-byte buffer for all messages. Messages are composed of two parts:

– header (10 bytes: fixed size)

– data (up to 261 bytes).

The size is based on the largest message managed at the short APDU transaction level. Due to this large size, the ST7SCR can have only 1 buffer and cannot store more than one message at a given time.

The CCID_BulkOutMessage() function uses several Bulk-out USB transactions to verify the message header and store the entire message in the buffer.

When the message reception is completed, the CCID_DispatchMessage() function identifies the message type and calls the corresponding function (PC_to_RDR_xxx()) for processing. Then, the IFD part of the firmware executes a command on the ICC if necessary. This command is included in the data field of the buffer. The PC_to_RDR_xxx() function receives from the IFD the answer to the message in the buffer data field and returns an error code to be included in the Bulk-in response.

The RDR_to_PC_xxx() function, which corresponds to the PC_to_RDR_xxx() previously executed, is launched and enters the correct values for error and status codes in the header.

At this time, the Bulk-in message is ready to be sent. But before the sending, the CCID_IntMessage() function is executed to detect any hardware problems or if a slot change has occurred. In this case, the corresponding interrupt message is sent to the host.

To finish the main loop, the CCID_BulkInMessage() function sends the Bulk-in message as an answer to the previous Bulk-out message. The message is sent in a process that requires several Bulk-in transactions (same concept as for the Bulk-out process).

In addition, the Abort request is implemented through the CCIDClassRequestAbort() function called in the USB_Polling() function. For this purpose, the USB_Polling() function is called in the PC_to_RDR_xxx() function. When this request is received from endpoint 0, a flag is set and the command sequence number is memorized.

All commands are aborted until the next PC_to_RDR_Abort command with the correct sequence number is received.

1.2 FUNCTION DESCRIPTION

- **void CCID_Init(void)**

This function initializes the flags and status variables for the state machine process.

- **void CCID_Init_IT(void)**

This function initializes the state machine and switches off the VccCard after a USB reset done by the PC Host. It can only be called by an interrupt routine.

- **void CCID_Suspend_IT(void)**

This function initializes the state machine and the card interface hardware after a resume from USB suspend mode.

- **void CCID_BulkOutMessage(void)**

This function manages the state machine during USB Bulk Out message reception. It fills the message buffer with a maximum of 271 bytes.

- **void CCID_BulkInMessage(void)**

This function manages the state machine during the USB Bulk In message transmission. It sends the buffer content as a message with a maximum of 271 bytes.

- **void CCID_DispatchMessage(void)**

This function identifies the USB Bulk Out received message and calls the corresponding functions to process it : PC_to_RDR_xxx() and RDR_to_PC_xxx().

- **void CCID_IntMessage(void)**

This function verifies the slot status and sends an interrupt In message if needed.

- **void CcidClassRequestAbort(void)**

This function is called from the USER_USB_Setup() function to process an Abort request.

- **unsigned char PC_to_RDR_IccPowerOn(void)**

This function verifies the PC_TO_RDR_ICCPOWERON command format and calls the IFD_IccPowerOn() function (from Interface Device level). If IFD_IccPowerOn() returns no error, the message header is filled with the ATR length. This PC_to_RDR function returns an error code.

- **unsigned char PC_to_RDR_IccPowerOff(void)**

This function verifies the PC_TO_RDR_ICCPOWEROFF command format, switches off the smartcard power supply and returns the NOERROR code.

- **unsigned char PC_to_RDR_GetSlotStatus(void)**

This function verifies the PC_TO_RDR_GETSLOTSTATUS command format, verifies the slot hardware state and returns the corresponding error code.

- **unsigned char PC_to_RDR_XfrBlock(void)**

This function verifies the PC_TO_RDR_XFRBLOCK command format, checks the slot status and returns an error if needed. If there is no error, it calls the IFD_XfrBlock() function (from the Interface Device level). The message buffer contains the data to be transferred and is updated by this IFD function. This PC_to_RDR function returns an error code.

- **unsigned char PC_to_RDR_GetParameters(void)**

This function verifies the PC_TO_RDR_GETPARAMETERS command format, checks the slot hardware state and returns the corresponding error code.

- **unsigned char PC_to_RDR_ResetParameters(void)**

This function verifies the PC_TO_RDR_RESETPARAMETERS command format, fills the message buffer with the default values of the reader parameters and calls the IFD_SetParameters() function (from the Interface Device level). This PC_to_RDR function returns an error code.

- **unsigned char PC_to_RDR_SetParameters(void)**

This function verifies the PC_TO_RDR_SETPARAMETERS command format and calls the IFD_SetParameters() function (from the Interface Device level). This PC_to_RDR function returns an error code.

- **unsigned char PC_to_RDR_Escape(void)**

This function verifies the PC_TO_RDR_ESCAPE command format and calls the IFD_Escape() function (from the Interface Device level). This PC_to_RDR function returns an error code (from IFD_Escape(), for example).

- **unsigned char PC_to_RDR_IccClock(void)**

This function verifies the PC_TO_RDR_ICCCLOCK command format and calls the IFD_SetClock() function (from the Interface Device level). This PC_to_RDR function returns an error code (from IFD_SetClock(), for example).

- **unsigned char PC_to_RDR_Abort(void)**

This function verifies the PC_TO_RDR_ABORT command format, checks the abort conditions and sets the corresponding flag. It returns the corresponding error code.

- **void RDR_to_PC_DataBlock(unsigned char ErrorCode)**

This function fills the whole command buffer header including the error code given as input to prepare the RDR_TO_PC_DATABLOCK message.

- **void RDR_to_PC_SlotStatus(unsigned char ErrorCode)**

This function fills the whole command buffer header including the error code given as input to prepare the RDR_TO_PC_SLOTSTATUS message.

- **void RDR_to_PC_Parameters(unsigned char ErrorCode)**

This function fills the whole command buffer header including the error code given as input to prepare the RDR_TO_PC_PARAMETERS message and call the IFD_GetParameters() function to fill the message data field.

- **void RDR_to_PC_Escape(unsigned char ErrorCode)**

This function fills the whole command buffer header including the error code given as input to prepare the RDR_TO_PC_ESCAPE message.

2 IFD IMPLEMENTATION

The Interface Device (IFD) implementation firmware contains one file:

– Ifd_protocol.c: This is the ISO 7816 implementation for Protocol types T=0 and T=1 and for character, TPDU and short APDU levels.

The protocol type is also managed at the CRD level by the interrupt system for data reception from the ICC. For this release, only the T=0 with character level has been implemented in the current firmware.

This part has to be modified in depth to integrate the secure function and other communication types and levels.

All the functions in this part are called by a CCID function. These functions are typically those where the reader interprets the messages sent to/from the ICC and manages the parameters for ICC communication (speed, type, etc....).

An IFD structure is used to save all the current parameters in compliance with CCID specifications.

The following functions are used for ICC and parameter management:

– IFD_Init(): This sets the parameter structure to the default value.

– IFD_GetParameters(): This returns the current parameters in a buffer.

– IFD_SetParameters(): This enters the new parameters in the IFD structure and programs the CRD for use with the new configuration.

– IFD_ApplyParametersStructure(): This programs the CRD to use the configuration described by the IFD structure.

– IFD_UpdateConvParameterStructure(): This changes the convention parameter of the structure with the value programmed in the CRD.

– IFD_IccPowerOn(): This switches ON the V_{CC} Card with automatic voltage selection and returns the Answer to Reset (ATR) variable.

– IFD_XfrBlock(): This sends a command and receives the answer with current parameters.

– IFD_XfrCharT0(): This is given by the IFD structure. (Character level and T=0 type).

– IFD_Escape(): This is dedicated to specific communication between reader and PC. (see CCID spec).

– IFD_SetClock(): This changes the clock state as configured in the IFD structure.

3 ISO 7816 UART LIBRARY (CRD)

3.1 GENERAL INTRODUCTION

The aim of this library is to provide the user with a set of functions for using the Smartcard Interface (ISO7816-3 UART Interface) to directly access the hardware and to communicate with an Integrated Circuit Card (ICC) through a few simple functions. This library is composed of 4 files:

– crd.c

– crd.h

– int_crd.c

– int_crd.h

The crd.x files provide the general functions for the main loop program. The int_crd.x files are used for interrupt management.

The transactions between the ST7 microcontroller and the ICC are performed by the ICC_xxx functions. These functions poll flags waiting for the end of the transaction. The transaction itself and the flag settings are managed by interrupts.

3.2 SMARTCARD INTERFACE FUNCTIONS

The Smartcard Interface (CRD) functions described below are very low level functions used to interact directly with the hardware.

- **void CRD_Init(void)**

This function resets and initialize the CRD and the local software flags.

- **unsigned char CRD_GetHwError(unsigned char * pHwErrorCode)**

This function checks the hardware state and returns one of the following error codes:

– Slot_No_Error

– SlotError_HW_Error

– SlotError_ICC_Mute (no card present)

If the function returns SlotError_HW_Error, pHwErrorCode will contain one of the following error codes:

– HardwareErrorCode_OverCurrent

– HardwareErrorCode_VoltageError

– HardwareErrorCode_OverCurrent_IT

– HardwareErrorCode_VoltageError_IT

Note: In case of a hardware error, the slot is inactivated (VCC Card off).

■ **unsigned char CRD_VccOn(unsigned char Voltage)**

This function switches ON the V_{CC} Card with value given by the Voltage variable. There are three possible voltage values:

– CRD_Voltage18V (for 1.8 volts)

– CRD_Voltage3V (for 3.0 volts)

– CRD_Voltage5V (for 5.0 volts)

This function returns one of the following error codes:

– Slot_No_Error

– SlotError_HW_Error

– SlotError_ICC_Mute (no card present)

■ **void CRD_VccOff(void)**

This function switches OFF the V_{CC} Card. The inactivating sequence is managed by hardware.

■ **void CRD_SetMode(unsigned char Mode)**

This function sets the operating mode of the CRD. There are two operating modes:

– CRD_ManualMode

– CRD_UARTMode

■ **void CRD_SetConvention(unsigned char Convention)**

This function sets the convention used by the CRD to communicate with the ICC. There are two conventions:

– CRD_DirectConv

– CRD_InverseConv

■ **unsigned char CRD_GetSlotStatus(void)**

This function returns the current status of the slot (only 1 slot is possible with this device). There are three possible states for the slot:

– CRD_NotPresent

– CRD_PresentInactive

– CRD_PresentActive

■ **unsigned char CRD_GetConvention(void)**

This function returns the convention currently in use in the CRD. There are two conventions:

– CRD_DirectConv

– CRD_InverseConv

■ **unsigned char CRD_GetClockStatus(void)**

This function returns the status of the CRD clock signal. There are three possible states for the clock signal:

– CRD_ClockRunning

– CRD_ClockStoppedLow

– CRD_ClockStoppedHigh

■ **unsigned char CRD_SetClock(unsigned char ClockState)**

This function modifies the state of the CRD clock signal. ClockState can be one of the following ones:

– CRD_ClockRunning

– CRD_ClockStoppedLow

– CRD_ClockStoppedHigh

– CRD_ClockStoppedLowOrHigh

This function returns one of the following error codes:

– Slot_No_Error

– SlotError_ICC_Mute

■ **unsigned char CRD_SetEtu(unsigned int Etu, unsigned char Comp)**

This function sets the Elementary Time Unit registers to the value of the Etu variable. The value of this variable must be a positive integer between 12 and 2048. If the Etu value is 2048, the hardware registers are cleared and the CRD behaves as if it was set to 2048. The Comp value can be 0 or 1. It is written to the COMP bit of register CRDETU1. For more information, please refer to the ST7SCR datasheet.

This function will return 0 if the CRDETU registers are correct and 0xFF if the input values are incorrect.

■ **void CRD_SetGuardTime(unsigned int GuardTime)**
void CRD_SetWaitingTime(unsigned long WaitingTime)

These functions enter positive integer values in the hardware timer registers.

- **void CRD_StartWaitingTime(void)**

This function starts the Waiting Timer. The bWaitingTimeFlag software flag is set by an interrupt when the count is finished. For more information, please refer to the crd.h file.

This counter is used by the ICC level functions to determine the end of the transmission.

- **void CRD_StopWaitingTime(void)**

This function stops the Waiting Timer and resets the corresponding flag.

- **void CRD_WaitingTime(unsigned long WaitingTimeInEtu)**

This function creates a wait loop for WaitingTimeInEtu. The waiting time is:

– WaitingTimeInEtu * Etu

This function is used to set the V_{CC} card stabilization delay or delays during the reset procedure. The Etu value is set by the CRD_SetEtu function.

- **void CRD_InitReceive(unsigned int ReceiveBufSize, unsigned char * pBuffer)**

This function will configure the CRD for reception and begin the process.

– pBuffer is the address of the buffer where the data will be copied.

– ReceiveBufSize is the size of the buffer.

If the ICC transmits more bytes than the number specified in the ReceiveBufSize variable, data saving is stopped and a flag is set. This reception is done entirely by interrupt processing.

- **unsigned int CRD_NumberOfBytesReceived(void)**

This function returns the number of bytes already received during the reception process.

- **unsigned int CRD_EndReceive(void)**

This function stops the reception process and returns the number of bytes received and copied in the buffer.

- **void CRD_InitTransmit(unsigned int TransmitBufSize, unsigned char * pBuffer)**

This function configures the CRD for transmission and begins the process.

– pBuffer is the address of the buffer containing the data to be transmitted.

– TransmitBufSize is the number of bytes to be transmitted.

This transmission is entirely done by interrupt processing.

- **unsigned int CRD_NumberOfBytesToTransmit(void)**

This function returns the remaining number of bytes to be transmitted.

■ **unsigned int CRD_EndTransmit(void)**

This function finishes the transmission by disabling the interrupt. This function returns the remaining number of bytes to be transmitted.

3.3 INTERRUPT FUNCTIONS

The following two functions are executed when an interrupt occurs:

■ **void INT_UART(void)**

This function is used for all CRD interrupts: parity, receive, transmit, waitingtime, voltage_error, over_current and transmit_empty.

– PARITY: A flag is set if 4 automatic retries have failed.

– WAITINGTIME: A flag is set when a counter overflow is detected.

– VOLTAGE and CURRENT: The corresponding flags are set, the corresponding interrupt is invalidated and the V_{CC} Card is switched OFF.

– TRANSMIT: The transmit register is loaded with the next byte to be sent. If the last byte is sent, the interrupt is disabled and a flag is set.

– RECEIVE: This function uses a Protocol Type Flag (software) to identify the received bytes and manage the T=0 and the T=1 special cases.

– TRANSMITEMPTY: Not used.

■ **void INT_CARDDET(void)**

This function will set a flag with Rebound Management (see the Time Base Unit (TBU) function).

3.4 ICC FUNCTIONS

The following functions execute a global action on the ICC.

■ **unsigned char ICC_PowerOnAsync(unsigned char Voltage)**

This function executes the Power-on sequence in compliance with ISO 7816 specifications for asynchronous ICCs, including the generation of the rising edge for the reset signal. The V_{CC} Card voltage is either 1.8 volts, 3.0 volts or 5.0 volts depending on the value of the Voltage variable:

– CRD_Voltage18V (for 1.8 volts)

– CRD_Voltage3V (for 3.0 volts)

– CRD_Voltage5V (for 5.0 volts)

This function returns one of the following error codes:

– Slot_No_Error

– SlotError_HW_Error

– SlotError_ICC_Mute

Note: At the end of this function, the CRD is in Manual mode.

- **void ICC_ResetAsync(void)**

This function executes a warm reset sequence in compliance with ISO 7816 specifications for asynchronous ICCs.

Note: At the end of this function, the CRD is in Manual mode.

- **unsigned char ICC_GetAtrAsync(unsigned char Voltage, unsigned char * pReceiveBuffer, unsigned int * pAtrSize)**

This function executes the Power-on or Reset sequence and answer to reset reading for an asynchronous ICC. The “voltage” variable can be:

– CRD_Voltage18V (for 1.8 volts)

– CRD_Voltage3V (for 3.0 volts)

– CRD_Voltage5V (for 5.0 volts)

Note: The “voltage” variable is used only if the slot status is inactive at the beginning of the function. In this case, the cold Answer to Reset (ATR) is returned. If the ICC is already active, a simple reset is generated and a warm ATR is returned.

– pReceiveBuffer is the address of the buffer to fill with the ATR.

– pAtrSize is the address of the data containing:

– Input: the size of the ReceiveBuffer in bytes.

– Output: the length of the ATR written in the ReceiveBuffer in bytes.

This function returns one of the following error codes:

– Slot_No_Error

– SlotError_HW_Error

– SlotError_ICC_Mute

– SlotError_BAD_ATR_TS

– SlotError_XFR_Overrun

- **unsigned char ICC_SendCommandAsync(unsigned char * pTransmitBuffer, unsigned int CommandSize, unsigned char ProtocolType)**

This function sends a packet of bytes to the ICC with the Guard Time currently in use.

– pTransmitBuffer is the address of the buffer containing the data to be sent.

– CommandSize is the size of the buffer in bytes.
– ProtocolType is 0 (for T=0) or 1 (for T=1). This variable is used to set a flag tested in the interrupt process at transmission and reception levels.

This function will return one of the following error codes:
– Slot_No_Error
– SlotError_ICC_Mute (if transfer not completed)

- **unsigned char ICC_ReceiveAnswerAsync(unsigned char * pReceiveBuffer, unsigned int * pAnswerSize)**

This function will receive a packet of bytes from the ICC with its waiting time already programmed.
– pReceiveBuffer is the address of the buffer to fill with the received data.
– pAnswerSize is the address of the data containing:
– Input: the size of the ReceiveBuffer in bytes.
– Output: the length in bytes of the received data written in the ReceiveBuffer.

This function will return one of the following error codes:
– Slot_No_Error
– SlotError_XFR_Overrun
– SlotError_ICC_Mute