UM1577
User manual

DEMO_CR95HF and STM3210E-EVAL board firmware

Introduction

This document describes the firmware of the DEMO_CR95HF board and details the available commands. It helps the developer to understand how this firmware works and how to tailor his/her own application.

The PC software can communicate with the CR95HF device through the MCU by using the USB bus.

Two families of commands are available:

- The commands dedicated to the CR95HF device. These commands are described in the CR95HF datasheet. In this case, the MCU translates the USB command to SPI or UART commands.
- The advanced commands. In this case, a specific function or process will be managed by the MCU.

Table 1 lists the development tools concerned by this user manual.

Table 1. Applicable tools

Type	Applicable tools
Development tools	DEMO_CR95HF, PLUG_CR95HF, STM3210E-EVAL

Contents

List of tables

List of figures

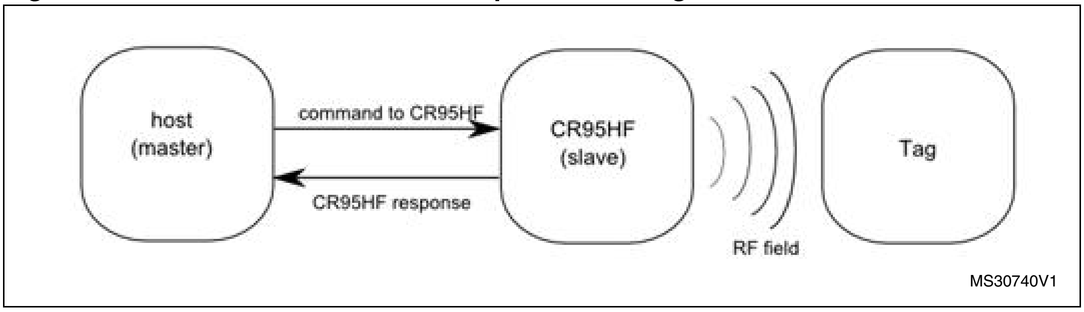
1 Overview

CR95HF is an RF transceiver IC for contactless application (ISO/IEC 15693, 14443A and B, and ISO/IEC 18092). It manages the RF communication with RFID or NFC tags that includes the frame coding, the RF modulation and the contactless tag response decoding.

The CR95HF also supports the detection, reading and writing of NFC Forum Type 1, 2, 3 and 4 tags.

The CR95HF is a slave device. Thus, a host (MCU) is required to control it.

Figure 1. CR95HF command and response exchange



For more details about the CR95HF device, refer to its datasheet.

1.1 STM32F103 overview

The STM32F103xx incorporates the high-performance ARM Cortex™-M3 32-bit RISC core operating at a 72 MHz frequency, high-speed embedded memories (Flash memory up to 128 Kbytes and SRAM up to 20 Kbytes), and an extensive range of enhanced I/Os and peripherals connected to two APB buses. All devices offer two 12-bit ADCs, three general purpose 16-bit timers plus one PWM timer, as well as standard and advanced communication interfaces: up to two I2Cs and SPIs, three USARTs, a USB and a CAN.

These features make the STM32F103xx microcontroller suitable for a wide range of applications such as motor drives, application control, medical and handheld equipment, PC and gaming peripherals, GPS platforms, industrial applications, PLCs, inverters, printers, scanners, alarm systems, video intercoms, and HVACs.

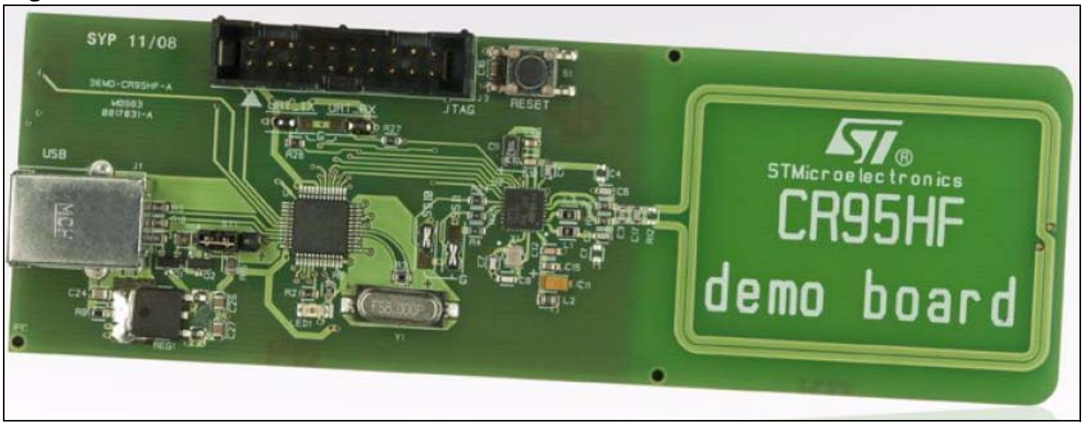
1.2 DEMO_CR95HF board

The DEMO-CR95HF-A is a demonstration kit which allows to evaluate the performances of an ST CR95HF 13.56 MHz multiprotocol contactless transceiver. It includes a ready-to-use board to interface with the CR95HF host PC demonstration software through a USB interface.

The DEMO-CR95HF-A is powered through the USB bus and no external power supply is required. It includes a CR95HF contactless transceiver, a 47 x 34 mm 13.56 MHz inductive etched antenna and its associated tuning components.

By default, the CR95HF communicates with the STM32F103CB 32-bit MCU via the SPI bus. The interface can then be changed to UART.

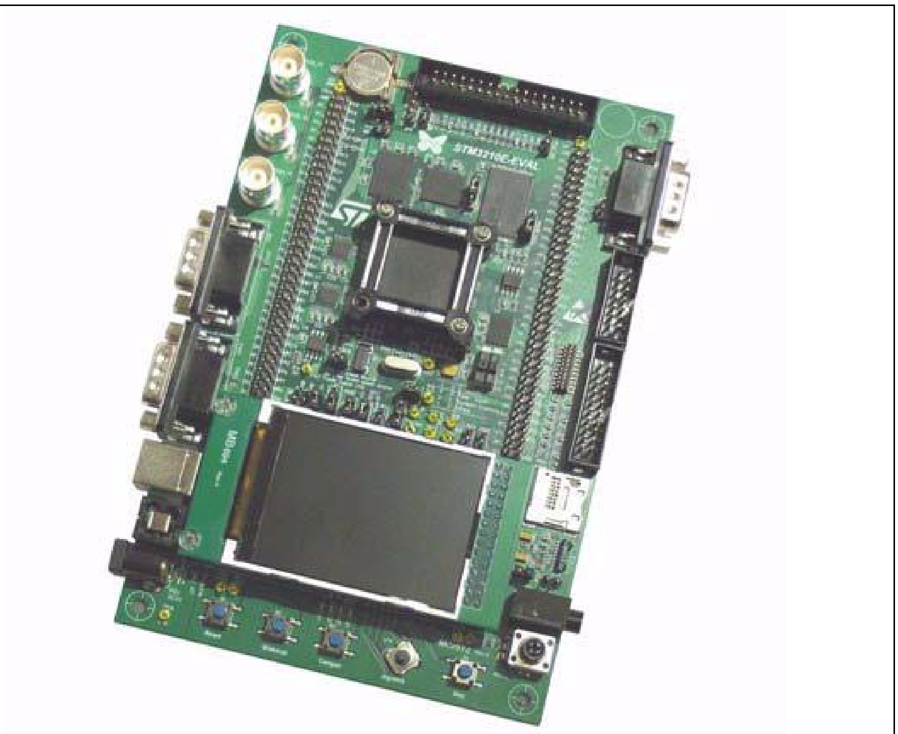
Figure 2. DEMO_CR95HF demonstration board



1.3 STM3210E-EVAL board

The STM3210E-EVAL evaluation board is a complete development platform for STMicroelectronic's ARM Cortex-M3 core-based STM32F103ZET6 or STM32F103ZGT6 microcontroller. The range of hardware features on the board help you to evaluate all peripherals (LCD, SPI Flash, USART, IrDA, USB, audio, CAN bus, Smartcard, MicroSD Card, NOR Flash, NAND Flash, SRAM, temperature sensor, audio DAC and motor control) and develop your own applications.

Figure 3. STM3210E-EVAL board

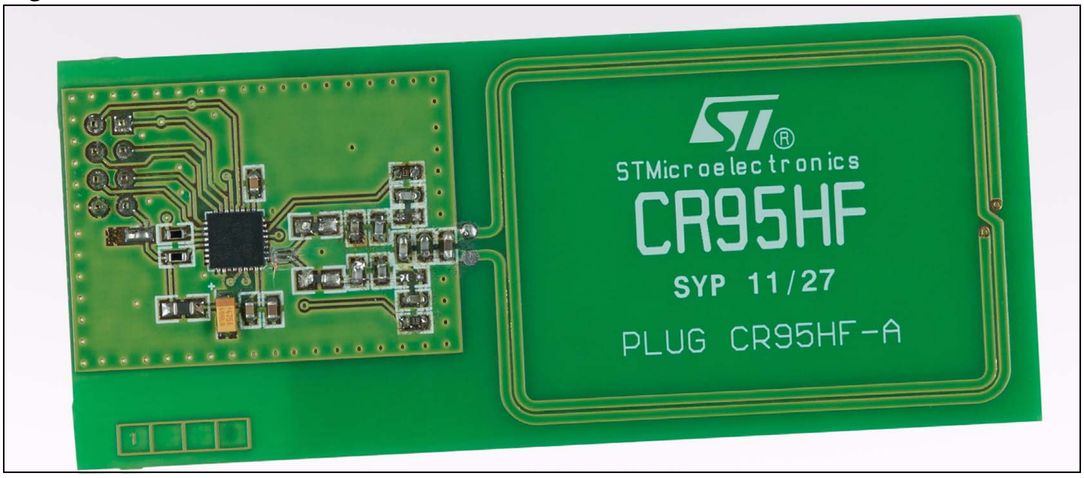


1.4 PLUG_CR95HF

The PLUG-CR95HF is a daughter board which includes a CR95HF contactless transceiver, a 47 x 34 mm 13.56 MHz inductive etched antenna and its associated tuning components.

The user must connect a host to the board through the UART or the SPI connector. It allows controlling the 13.56 MHz CR95HF multi-protocol transceiver IC from the host.

Figure 4. PLUG_CR95HF board



1.5 CR95HF development software

The CR95HF development software is a PC software which allows to configure, evaluate, and communicate with an ST CR95HF 13.56 MHz multiprotocol contactless transceiver.

The software must be used in conjunction with the DEMO-CR95HF-A demonstration kit (see Figure 1) which includes a ready-to-use board to interface with the host PC through a USB interface.

The DEMO-CR95HF-A is powered through the USB bus and no external power supply is required. It includes a CR95HF contactless transceiver, a 48 x 34 mm 13.56 MHz inductive etched antenna and the associated tuning components. The CR95HF communicates with the STM32F103CB 32-bit core MCU via the SPI bus or the UART bus.

The CR95HF development software is available on the ST internet web site.

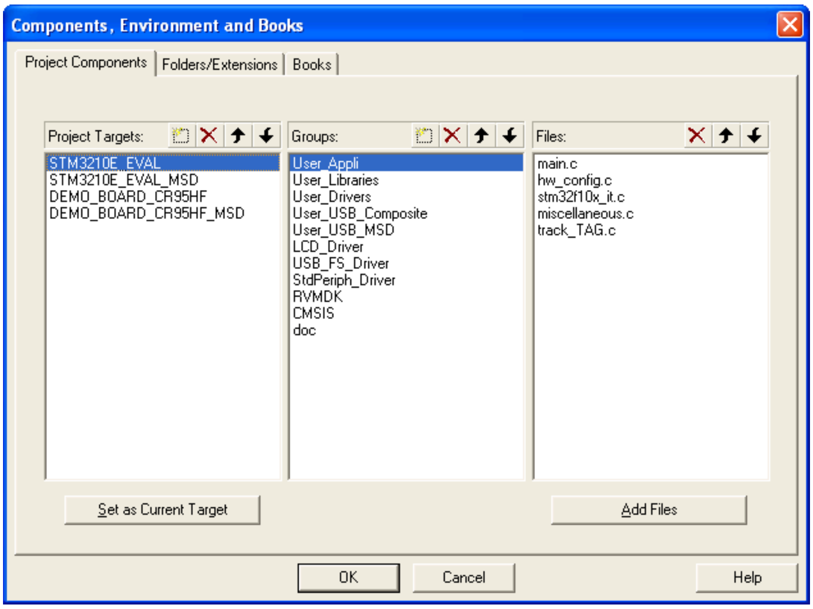
2 Firmware overview

2.1 The four project targets

The firmware was developed using KEIL µVision and can be used on the DEMO_CR95HF or on the STM3210E_EVAL board with the conjunction of a PLUG_CR95HF board. The USB mass storage feature can be activated or not. Thus, the Keil project contains four project targets, as can be seen on *Figure 5*:

- STM3210E Eval board without the USB mass storage feature
- STM3210E Eval board with the USB mass storage feature
- DEMO_CR95HF without the USB mass storage feature
- DEMO_CR95HF with the USB mass storage feature

Figure 5. Four project targets



2.2 Firmware package

The firmware, developed using KEIL µVision 4.2, is delivered in a .zip file and contains all the subdirectories and .h and .c source code files that make up the core of the application. The related KEIL workspace/project files are also included.

The KEIL project is organized in project folders coherent with file-system folders.

Figure 6. Project organization

The firmware contains all the application task source files and related module files, and consists of the following project folders:

- User_Appli: the application layer.
- User_Libraries: the CR95HF and the contactless tag libraries. These libraries include the commands to communicate with the CR95HF transceiver or with a contactless tag.
- User_Drivers: the drivers let the GPIO of the MCU communicate with the CR95HF transceiver via the UART or SPI bus. This folder includes the LED management.
- User_USB_composite: the source code that manages the HID communication.
- User_USB_MSD: the source code that manages the Mass storage device.
- LCD_driver: the source file that manages the LCD of the STM3210E EVAL board.
- USB_FS_Drive: the STM32 MCU standard library for the USB bus.
- StdPeriph_Driver: the STM32 MCU standard library.

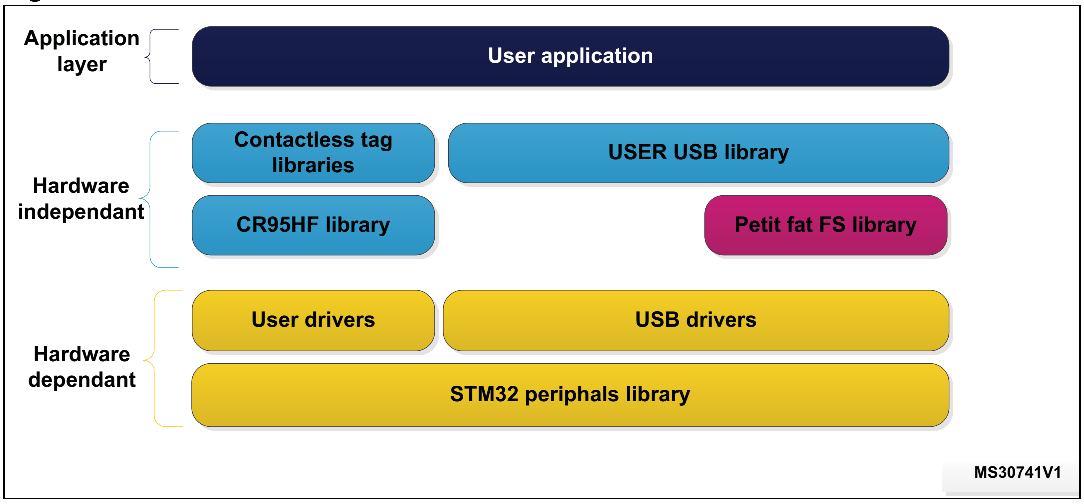
2.3 Application firmware architecture

This layers architecture improves the code reusability by splitting the application programming interface code (portable and reusable) provided by the libraries layer from the hardware abstraction layer code (hardware-dependent and written in the STM32F10xxx libraries).

The application layer also includes the third party library Petit FatFS. Petit FatFs is a sub-set of the FatFs module; It has been written in compliance with ANSI C and is completely separated from the disk I/O layer. It can be embedded into the microcontroller with little memory, even with a RAM size lower than the sector size.

For more details, refer to the internet web site: http://elm-chan.org/fsw/ff/00index_p.html.

Figure 7. Firmware architecture



3 HID commands dedicated to the CR95HF

This section describes the commands dedicated to the CR95HF device. The MCU changes the USB command sent by the PC software to a SPI or UART command. In the same way, the contactless tag response is retrieved by the CR95HF device and converted to a USB frame sent to the PC software.

The MCU doesn't check or change the data provided by the PC software.

Table 2. Function description format

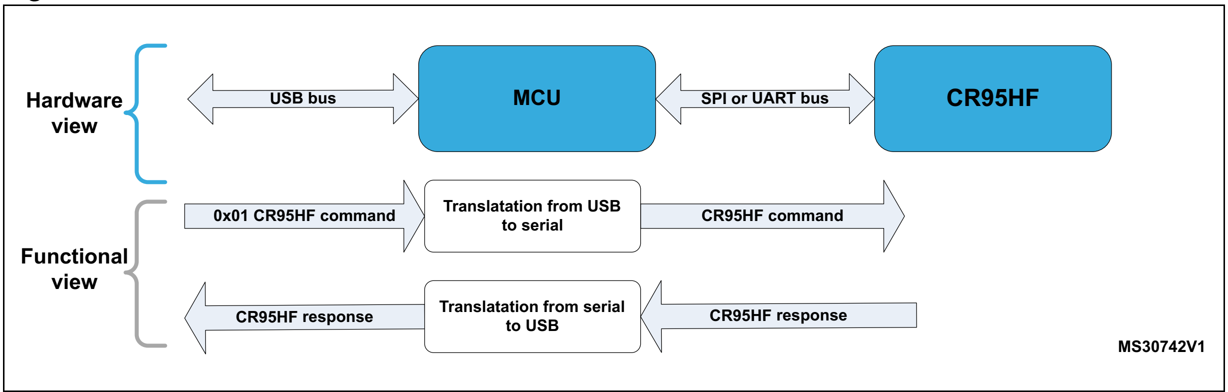
Function	Description
Function name	Name of the peripheral function
Function prototype	Prototype declaration
Input parameter	Description of the input parameters
Output parameter	Description of the output parameters
Return value	Value returned by the function

The user can send these commands thanks to the CR95HF development software.

The format of an HID command is detailed in *Table 2*. Two families of HID commands are available. The first family of HID commands is dedicated to the CR95HF device.

Figure 8 shows this transaction.

Figure 8. Hardware and functional view of an HID command



The user can transmit one command at a time. The MCU does not check the command and will not change it. The command dedicated to the CR95HF starts with byte 0x01. The next bytes of the CR95HF command are defined in the datasheet.

Table 3. HID command description format

HID command format			
0x01 or 0x02	**1 byte**	**1 byte**	**Zero or more bytes**
0x01: CR95HF command 0x02: Advanced command			
Command code			
Number of bytes of data			
Data			

Table 4. HID response description format

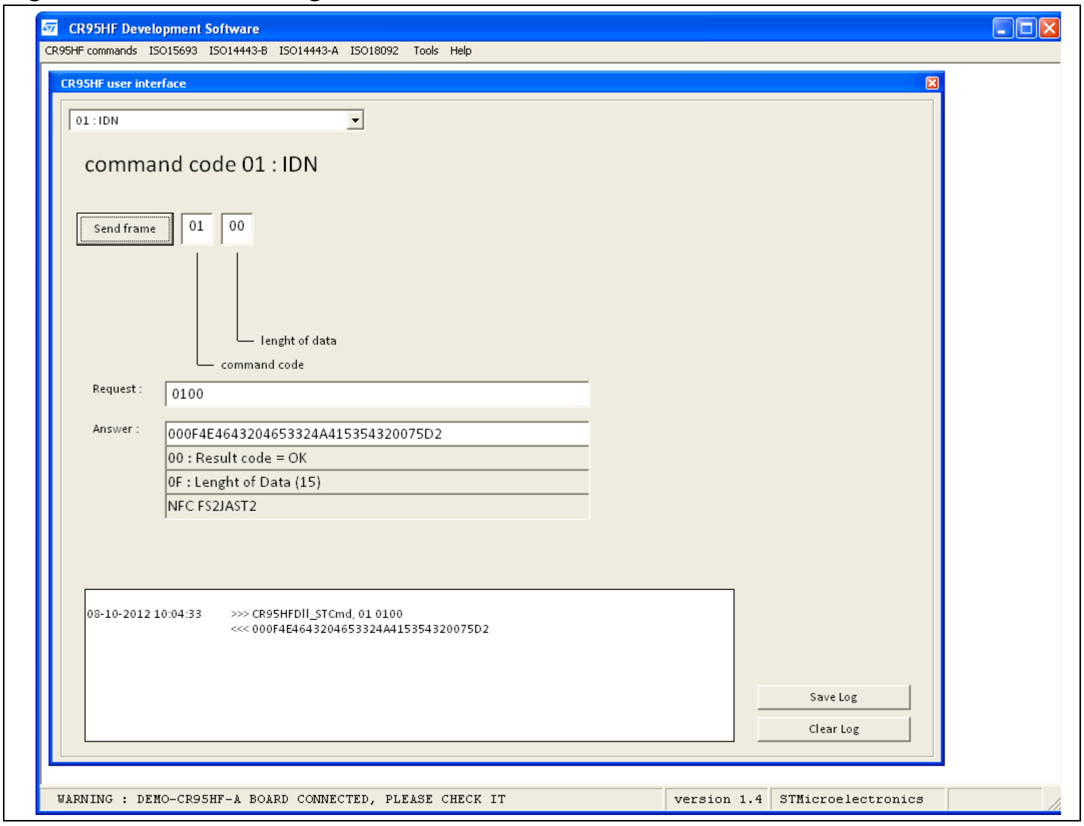
HID response format		
1 byte	**1 byte**	**Zero or more bytes**
Response code		
Number of bytes of data		
Data		

3.1 Example of an HID command dedicated to the CR95HF

The CR95HF development software can send the HID command dedicated to the CR95HF transceiver.

The capture below show the way to send an HID command dedicated to the CR95HF.

Figure 9. User sending an HID command to the CR95HF



The command is an IDN command and the frame is displayed in the Log window:

>>> CR95HFDll_STCmd, 01 0100

<<< 000F4E4643204653324A415354320075D2

The HID command can be split as follows:

Table 5. HID command of the IDN command

HID command format: 01 0100			
0x01	**0x01**	**0x00**	**-**
CR95HF command			
IDN command code			
Number of bytes of data			
No data			

The HID response can be split as follows:

Table 6. HID response of the IDN command

HID response format: 000F4E4643204653324A415354320075D2		
0x00	**0x0F**	**4E4643204653324A415354320075D2**
Response code		
Number of bytes of data		
Data		

3.1.1 Function of the firmware

When the MCU receives an HID command with the first byte to 0x01, the firmware will call the SPIUART_SendReceive function.

Table 7. Calling the SPIUART_SendReceive function

Function	Description
Function name	SPIUART_SendReceive
Function prototype	int8_t SPIUART_SendReceive (uc8 *pCommand, uint8_t *pResponse)
Input parameter	pCommand: pointer on the command to send to the CR95HF
Output parameter	pResponse pointer on the CR95HF response
Return value	RESULTOK: successful code CR95HF_ERRORCODE_DEFAULT: an error occurred CR95HF_POLLING_CR95HF: timeout error

3.2 Error code

Table 8 lists the error code that the MCU can return.

Table 8. Error code returned by the MCU

Error code name	value	description
CR95HF_ERRORCODE_DEFAULT	0xFE	Unidentified error code
CR95HF_ERRORCODE_TIMEOUT	0xFD	Timeout error. The CR95HF did not return a response
CR95HF_ERRORCODE_FILENOTFOUND	0xFA	The file was not found
CR95HF_ERRORCODE_READALLMEMORY	0xF9	The ReadAllMemory function returned an error
CR95HF_ERRORCODE_TAGDETECTINGCALIBRATION	0xF8	The tag calibration was not successful
CR95HF_ERRORCODE_CUSTOMCOMMANDUNKNOWN	0xF7	The command is unknown
CR95HF_ERRORCODE_TAGDETECTING	0xF5	The tag detecting function returned an error code
CR95HF_ERRORCODE_NOTAGFOUND	0xF4	No tag was detected in the RF field

3.2.1 Example of an HID transaction with an MCU error code

The following transaction is an Idle command. This command switches the CR95HF into the hibernate state and therefore does not reply to the command. An MCU timeout error occurs:

>>> CR95HFDll_STCmd, 01 070E0801003800180000600000000000

<<< FD00

4 Commands dedicated to the MCU

This family of commands is dedicated to the MCU. The firmware will decode and launch a specific operation, for example the management of an anticollision process. In this case, the MCU will manage the communication with the CR95HF transceiver. The MCU can transmit one or more command and analyze the CR95HF response.

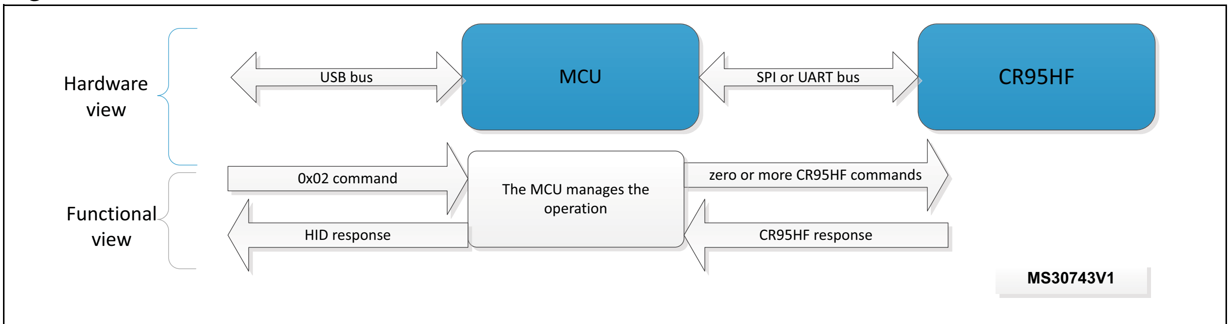
Table 9. Function description format

Function	Description
Function name	The name of the peripheral function
Function prototype	Prototype declaration
Input parameter	Description of the input parameters
Output parameter	Description of the output parameters
Return value	Value returned by the function
Required preconditions	Requirements before calling the function

This section describes the dedicated functions available and explains how to use them.

Figure 10 shows the hardware and functional view of a HID command dedicated to the MCU.

Figure 10. Hardware and functional view of an HID command dedicated to the MCU



The tables below detail the HID command and response.

Table 10. HID command description format

<table>
<tr><th colspan="4">HID command format dedicated to the MCU</th></tr>
<tr><th>0x02</th><th>1 byte</th><th>1 byte</th><th>Zero or more bytes</th></tr>
<tr><td>Advanced command</td><td></td><td></td><td></td></tr>
<tr><td colspan="2">Command code</td><td></td><td></td></tr>
<tr><td colspan="3">Number of bytes of data</td><td></td></tr>
<tr><td colspan="4">Data</td></tr>
</table>

Table 11. HID response description format

HID response format		
1 byte	**1 byte**	**Zero or more bytes**
Response code		
Number of bytes of data		
Data		

4.1 Example of an HID command dedicated to the MCU

This example is the simplest HID command dedicated to the MCU. The firmware returns its revision number.

4.1.1 Example of a GetFirmwareVersion transaction

This transaction is an HID command dedicated to the MCU. The MCU returns its firmware version number:

>>> CR95HFDLL_STCMD, 02 BC00

<<< 0003020002

The HID command can be split as follows:

Table 12. HID command of the GetFirmwareVersion transaction

HID command format dedicated to the MCU: 02 BC00			
0x02	**0xBC**	**0x00**	**-**
Customs command			
Get the firmware revision			
Number of bytes of data			
No data			

The HID response can be split as follows:

Table 13. HID response of the GetFirmwareVersion transaction

HID response format: 0003020002		
0x00	**0x03**	**020002**
Successful code		
Number of bytes of data		
Version 2.0.2		

4.2 List of the customs functions

Table 14 list the customs functions.

Table 14. Customs functions

Name	Operating code	Brief description
ISO/IEC 15693 anticollision	0xA0	Runs a anti-collision process
inventory 16 slots	0xA1	transmits an inventory 16 slots
GotoTagdetectingState	0xA3	Goes to the tag detecting state
CR95HF_IsWakeUp	0xA2	Checks if the CR95HF is waked up
CalibrateTagdetecting	0xA4	Carries out the tag detecting calibration
PulseSPINSS	0xB9	Sends a pulse on SPI_NSS pin
PulseIRQin	0xBE	Sends a pulse on IRQin pin
SendSPIreset	0xBD	Sends an SPI reset sequence to reset the CR95HF device
ActivateTheTagTacking	0xC4	Activates the tag tracking
CustomReadTagMemory	0xB0	Reads a part of or the whole memory of an ISO/IEC 15693 contactless tag
GetFirmwareVersion	0xBC	Returns the firmware version
GetHardwareRevision	0xB2	Returns the hardware version

4.3 ISO/IEC 15693 anticollision function

This customs command runs an anti-collision function for the ISO/IEC 15693 protocol.

4.3.1 HID command format

Table 15. HID command of the ISO/IEC 15693 anti-collision function

HID command format dedicated to the MCU: 02 A0 01 26			
0x02	**0xA0**	**0x01**	**26**
MCU command			
Get the firmware revision			
Number of bytes of data			
Request flags (1)			

1. The request flags is the first byte of the ISO/IEC 15693 RF commands.

4.3.2 HID response format

Table 16. HID response of the ISO/IEC 15693 anti-collision function

HID command format dedicated to the MCU: 02 A0 01 26			
0x80	**1 byte**	**1 byte**	**Multiples of 9 bytes**
MCU command			
Get the number of byte data			
Number of tags inventoried			
DSFID bye and the 8 UID bytes (1)			

1. The response can contains up to 6 DSFID and UID fields.

4.3.3 Example

This example shows the transaction of an ISO/IEC 15693 anti-collision process:

>>> CR95HFDLL_STCMD, 02A00126

<<< 801C0300EC19563C172202E0006E8E1938422002E0003125563C172202E0

Table 17. HID response of the ISO/IEC 15693 anti-collision function

HID command format dedicated to the MCU: 801C0300EC19563C172202E0006E8E1938422002E0003125563C172202E0			
0x80	**0x1C**	**03**	**00 EC19563C172202E0 00 6E8E1938422002E0 00 03125563C172202E0**
MCU command			
Number of bytes			
3 tags have been inventoried			
DSFID byte and the 8 UID bytes of the 3 tags			

Table 18. Function description format

Function	Description
Function name	ISO15693_RunAntiCollision
Function prototype	int8_t ISO15693_RunAntiCollision (uc8 Flags, uc8 AFI,uint8_t *NbTag,uint8_t *pUIDout)
Input parameter	Flags: Request Flags of the ISO/IEC 15693 RF command AFI:AFI byte of the ISO/IEC 15693 RF command
Output parameter	NbTag: Number of tags inventoried pUIDout: pointer on DSFID and UID fields

Table 18. Function description format (continued)

Function	Description
Return value	RESULTOK: function successful ERRORCODE_GENERIC: an error occurred
Required preconditions	A protocol select function shall be transmitted before in order to switch the RF field On and configure the CR95HF transceiver to use the ISO/IEC 15693 protocol.

4.4 ISO/IEC 15693 inventory 16 slots function

This customs command sends an inventory 16 slots function and manages the transmission of the slots.

4.4.1 HID command format

Table 19. HID command of the ISO/IEC 15693 inventory 16 slots function

HID command format			
0x02	**0xA1**	**0x01**	**0x26**
Customs command			
Get the command code			
Number of bytes of data			
Request flags (1)			

1. The request flags is the first byte of the ISO/IEC 15693 RF commands.

4.4.2 HID response format

Table 20. HID response of the ISO/IEC 15693 inventory 16 slots function

HID command format			
0x80	**1 byte**	**1 byte**	**Multiples of 9 bytes**
Customs command			
Get the firmware revision			
Number of tags inventoried			
DSFID bye and the 8 UID bytes (1)			

1. The response can contains up to 6 DSFID and UID fields.

4.4.3 Example

This example shows the transaction of an ISO/IEC 15693 inventory 16 slots function:

>>> CR95HFDLL_STCMD, 02A10126

<<< 801C0300EC19563C172202E0006E8E1938422002E0003125563C172202E0

Table: HID response of the ISO/IEC 15693 anti-collision function

Table 21. HID response of the ISO/IEC 15693 inventory 16 slots function

HID command format dedicated to the MCU: 801C0300EC19563C172202E0006E8E1938422002E0003125563C172202E0			
0x80	0x1C	03	00 EC19563C172202E0 00 6E8E1938422002E0 00 03125563C172202E0
MCU command			
Number of bytes			
3 tags have been inventoried			
DSFID byte and the 8 UID bytes of the 3 tags			

Table 22. Function description format

Function	Description
Function name	ISO15693_RunInventory16slots
Function prototype	int8_t ISO15693_RunInventory16slots (uc8 Flags, uc8 AFI,uint8_t *NbTag,uint8_t *pUIDout)
Input parameter	Flags: Request Flags of the ISO/IEC 15693 RF command AFI:AFI byte of the ISO/IEC 15693 RF command
Output parameter	NbTag: Number of tag inventoried pUIDout: pointer on DSFID and UID fields
Return value	RESULTOK: function successful ERRORCODE_GENERIC: an error occurred
Required preconditions	A protocol select function shall be transmitted before in order to switch the RF field On and configure the CR95HF transceiver to use the ISO/IEC 15693 protocol.

4.5 GotoTagdetectingState function

This customs commands sends an Idle command to the CR95HF. The Wake up sources are both IRQ_in or the tag detecting state. This command returns a successful code. After sending this command, the PC shall use the CR95HF_IsWakeUp function to know if the CR95HF is in the ready state and ready to execute some new commands.

The main function of the firmware scans the state of the CR95HF device to know if a tag was detected.

4.5.1 HID command format

Table 23. HID command of the GoToTagDetectingState function

HID command format			
0x02	**0xA3**	**0x02**	**2 bytes**
Custom command			
Get the command code			
Number of bytes of data			
DACdataL & DACdataH values (1)			

1. For more details about these values, refer to the CR895Hf datasheet.

4.5.2 HID response format

Table 24. HID response of the GoToTagDetectingState function

HID command format	
0x80	**0x00**
Response code	
Number of bytes	

4.6 CR95HF_IsWakeUp function

This customs commands returns the state of the IRQout pin. When the CR95HF wakes up from an Idle state, the IRQout goes to the low state.

4.6.1 HID command format

Table 25. HID command format

HID command format			
0x02	**0xA2**	**0x00**	**-**
Custom command			
Command code			
Number of bytes of data			
No data			

4.6.2 HID response format

Table 26. HID response of the CR95HF_IsWakeUp function

HID command format		
0x80	**0x01**	**1 byte**
MCU command		
Number of bytes		
0x01: IRQout pin is to the High state 0x00: IRQout pin is to the Low state		

4.7 CR95HF_CalibrateTagDetecting function

This customs command carries out the calibration of the tag detection as described in the AN3433 "Optimizing wakeup time and power consumption in CR95HF and STRFNFCA devices".

4.7.1 HID command format

Table 27. HID command of the CalibrateTagDetecting

HID command format		
0x02	**0xA4**	**0x00**
Custom command		
Command code		
Number of bytes of data		

4.7.2 HID response format

Table 28. HID response of the CalibrateTagDetecting

HID response format			
0x80	**0x02**	**1 byte**	**1 byte**
response code			
Number of bytes of data			
DACdataL[(1)]			
DACdataH [(1)]			

1. The DACdataL and DACdataH values are used by the Idle command to configure the CR95HF into the tag detecting state.

Table 29. Function description format

Function	Description
Function name	CR95HF_ GetTagDetectionRefValue
Function prototype	int8_t CR95HF_GetTagDetectionRefValue (uint8_t *DacDataRef)
Input parameter	DacDataRef
Output parameter	none
Return value	CR95HF_SUCCESS_CODE: the function is successful CR95HF_ERRORCODE_TAGDETECTINGCALIBRATION: the function is not successful
Required preconditions	none

4.7.3 Example of a tag detection application

1. To activate the tag detecting state, the first step should be the calibration of the tag detecting. This process allows to find the DACdataL and DacDataH values.
   The log script below corresponds to the calibration of a tag detecting state:

   >>> CR95HFDLL_STCMD, 02 A400
   <<< 80026C74

Table 30. HID command format

HID command format: 80026C74			
0x80	**0x02**	**0x6C**	**0x74**
Response code			
Number of bytes of data			
DACdataL[(1)]			
DACdataH [(1)]			

1. The DACdataL and DACdataH values are used by the Idle command to configure the CR95HF into the tag detecting state.

2. The second step is to configure the CR95HF into the tag detecting state.

   >>> CR95HFDLL_STCMD, 02 A3026C74
   <<< 8000

   At this moment, the CR95HF is in a Tag detecting state and will not reply to the serial interface command. For example:

   >>> CR95HFDll_STCmd, 01 55
   <<< FD00

   The MCU returns the timeout error code because the CR95HF is in the tag detection mode and did not reply to the ECHO command.
3. The third step is to pool the MCU to know if a tag has gone to the RF field.

   >>> CR95HFDLL_STCMD, 02 A200
   <<< 800101
   >>> CR95HFDLL_STCMD, 02 A200
   <<< 800101

```
>>> CR95HFDLL_STCMD, 02 A200
<<< 800101
>>> CR95HFDLL_STCMD, 02 A200
<<< 800100
```

The last response, 80 01 00, indicates that a tag is in the RF field.

4.8 PulseSPINSS function

This customs commands sends a negative pulse on the SPI NSS pin. This pin can be configured as a wake-up source of the Idle mode.

4.8.1 HID command format

Table 31. HID command of the PulseSPINSS function

HID command format		
0x02	**0xB8**	**0x00**
Customs command		
Command code		
Number of bytes of data		

4.8.2 HID response format

Table 32. HID response of the PulseSPINSS function

HID response format	
0x80	**0x00**
Response code	
Number of bytes	

Table 33. Function description format

Function	Description
Function name	CR95HF_Send_SPINSS_NegativePulse
Function prototype	void CR95HF_Send_SPINSS_NegativePulse(void)
Input parameter	none
Output parameter	none
Return value	none
Required preconditions	none

4.9 PulseIRQin function

This customs commands sends a negative pulse on the IRQ_In pin. This pin can be configured as a wake-up source of the Idle mode.

4.9.1 HID command format

Table 34. HID command of the PulseIRQin function

HID command format		
0x02	0xBE	0x00
Custom command		
Command code		
Number of bytes of data		

4.9.2 HID response format

Table 35. HID response of the PulseIRQin function

HID response format	
0x80	0x00
Custom command	
Number of bytes	

Table 36. Function description format

Function	Description
Function name	CR95HF_Send_IRQIN_NegativePulse
Function prototype	void CR95HF_Send_IRQIN_NegativePulse (void)
Input parameter	none
Output parameter	none
Return value	none
Required preconditions	none

4.10 SendSPIReset function

This customs command sends an SPI reset sequence in order to reset the CR95HF device and the negative pulse on the IRQin pin to wake up the CR95HF. This reset can be carried out only when the CR95HF device uses the SPI serial interface.

4.10.1 HID command format

Table 37. HID command of the SendSPIreset function

HID command format		
0x02	**0xBD**	**0x00**
Custom command		
Command code		
Number of bytes of data		

4.10.2 HID response format

Table 38. HID response of the SendSPIreset function

HID response format	
0x80	**0x00**
Custom command	
Number of bytes	

Table 39. Function description format

Function	Description
Function name	CR95HF_Send_SPI_ResetSequence
Function prototype	void CR95HF_Send_SPI_ResetSequence (void)
Input parameter	None
Output parameter	None
Return value	None
Required preconditions	None

4.11 GetFirmwareVersion function

This customs command returns the firmware version.

4.11.1 HID command format

Table 40. HID command of the GetFirmwareVersion function

HID command format		
0x02	**0xBD**	**0x00**
Custom command		
Command code		
Number of bytes of data		

4.11.2 HID response format

Table 41. HID response of the GetFirmwareVersion function

HID response format		
0x00	0x03	3 bytes
Response code		
Number of bytes		
Firmware version		

4.12 GetHardwareVersion function

This customs command returns the hardware version.

4.12.1 HID command format

Table 42. HID command of the GetHardwareVersion function

HID command format		
0x02	0xB2	0x00
Custom command		
Command code		
Number of bytes of data		

4.12.2 HID response format

Table 43. HID response of the GetHardwareVersion function

HID response format		
0x00	1 byte	Length bytes
Response code		
Number of bytes		
Hardware version (ASCII format)		

4.13 ActivateTagTracking function

This customs command activates the tag tracking function. After the board power-up, the tag tracking is activated and deactivated when the MCU receives an HID command.

4.13.1 HID command format

Table 44. HID command of the ActivateTagTracking function

HID command format		
0x02	**0xC4**	**0x00**
Custom command		
Command code		
Number of bytes of data		

4.13.2 HID response format

Table 45. HID response of the ActivateTagTracking function

HID response format	
0x80	**0x00**
Custom command	
Number of bytes	

4.14 CustomReadTagMemory function

This function reads a part of or the whole memory of the Low or High density ISO/IEC 15693 contactless tag. The data read from the contactless tag is saved in the MCU memory.

Table 46. HID command of the CustomReadTagMemory function

HID command format				
0x02	**0xB0**	**0x03**	**2 bytes**	**1 byte**
Custom command				
Command code				
Number of bytes				
Block address				
Number of blocks to read 0x00: read the whole memory				

4.14.1 HID response format

Table 47. HID response of the CustomReadTagMemory function

HID response format			
0x80	**0x02**	**1 byte**	**1 byte**
Response code			
Number of bytes			
Number of HID frames required to upload (1)			
Number of CRC errors occurred (1)			

1. See ReadMCUBuffer function.

4.15 ReadMCUBuffer function

This function returns 15 blocks of the MCU buffer. The size of a block is 4 bytes.

4.15.1 HID command format

Table 48. HID command of the ReadMCUBuffer function for low density products

HID command format			
0x02	**0xB1**	**0x01**	**1 byte**
Custom command			
Command code			
Number of bytes			
number of 60-byte fields to read			

Table 49. HID response of the ReadMCUBuffer function for low density products

HID response format			
0x80	**0x02**	**1 byte**	**1 byte**
Response code			
Number of bytes			
number of 60-byte fields to read			
Data			

4.16 Example of the CustomReadTagMemory and ReadMCUBuffer functions

In this example, the whole memory of an LRIS64k tag is read and saved in the MCU.

To read the whole memory of the LRIS64k contactless tag, use the CustomReadTagMemory with the following parameters:

Table 50. HID command

HID command format				
0x02	**0xB0**	**0x03**	**0x00 00**	**0x00**
Custom command				
Command code				
Number of bytes				
First block address				
0x00: read the whole memory				

>>> CR95HFDLL_STCMD, 02 B003000000

<<< 80028900

The response of the CustomReadTagMemory can be split as:

Table 51. HID response

HID response format: 80028900			
0x80	**0x02**	**0x89**	**0x00**
Response code			
Number of bytes			
Number of HID frames required to upload			
Number of CRC errors occurred			

At this point, the contents of the contactless tag has been saved to the STM32 RAM memory. In order to upload the STM32 buffer memory, use the ReadMCUBuffer function. The number of commands required to upload the whole memory is 0x89 (second byte of the CustomReadTagMemory response).

The LRIS64k has a 64-kbit memory or 8192 bytes. As the maximum number of bytes of the HID frame is 64 bytes, the response to a ReadMCUBuffer can contain only 60 bytes.

$$\mathrm{NumberofReadMCUBufferframe} = \frac{8192}{60} = 136,5 \sim 137 = 0x89$$

The example below shows the HID frame to read the first 3 fields of 60 bytes:

```
>>> CR95HFDLL_STCMD, 02 B10100
<<<
803D00E140FF030327D10123550173742E636F6D2F696E7465726E65742F6D6375
2F636C6173732F313736362E6A7370FE0000FFFFFFFFFFFFFFFFFFFFFFFF
>>> CR95HFDLL_STCMD, 02 B10101
<<<
803D01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FF
>>> CR95HFDLL_STCMD, 02 B10102
<<<
803D02FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FF
```

5 Tag Tracking feature

After the board power-up, the firmware launches the TagTracking feature. The MCU lets the CR95HF find the RFID or NFC tag present in the RF field. When a tag is found, a LED is On.

5.1 Algorithm of the tag tracking

The algorithm of the tag tracking is as follows:

Figure 11. Algorithm of the tag tracking

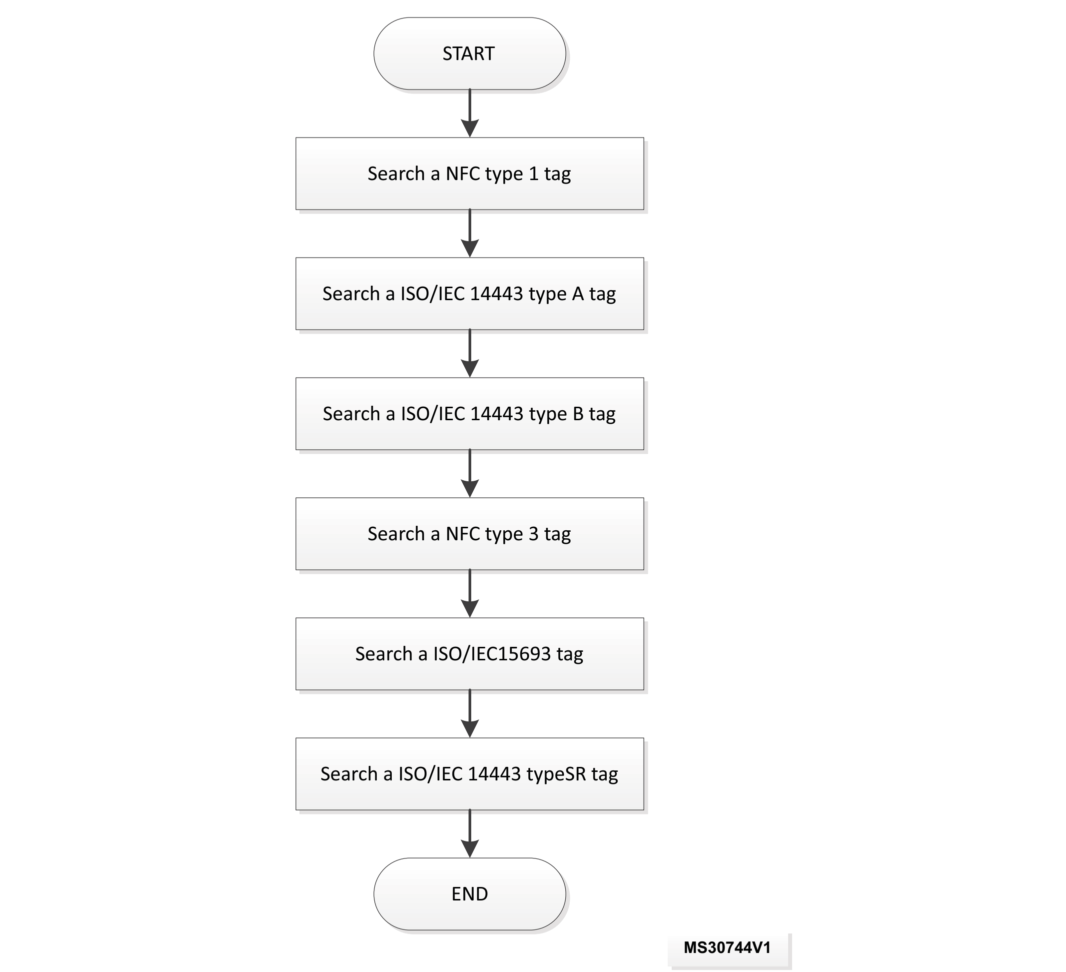


Table 52 lists the variables that activate the track of the different protocols.

Table 52. Variables to activate the tag tracking

Variable name	Description
CON_POLL_TOPAZ	Activates the track of the NFC type 1 tag
CON_POLL_A	Activates the track of the ISO/IEC 14443 type A tag

Table 52. Variables to activate the tag tracking

Variable name	Description
CON_POLL_B	Activates the track of the ISO/IEC 14443 type B tag
CON_POLL_F	Activates the track of the NFC type 3 tag
CON_POLL_15693	Activates the track of the ISO/IEC 15693 type A tag
CON_POLL_SR	Activates the track of the ISO/IEC 14443 type SR tag (1)

1. Proprietary ST tag based on the ISO/IEC 14443 type B specification*

5.2 Tag tracking management

The tag tracking feature is deactivated when the MCU received a USB HID frame. It can be activated using the ActivateTagTracking function (see ActivateTagTracking function).

6 USB mass storage feature

The USB mass storage device class, otherwise known as USB MSC or UMS, is a protocol that allows a Universal Serial Bus (USB) device to become accessible to a host computing device, in order to enable file transfers between the two host devices. The USB device is similar to an external hard drive, enabling drag-and-drop file transfers.

6.1 USB mass storage and transfer types

The interrupt and bulk transfers are USB protocols.

The DEMO_CR95HF_A board can use both transfer types.

The HID transfer uses an interrupt transfer and is useful to exchange a limited amount of data. The maximum number of bytes to be exchanged in a HID frame is 64 bytes. The HID transfer is adapted to the transfer of a small command and response.

The number of bytes of a bulk type frame adds up to 512 bytes of data. This is suitable for the transfer of the whole content of the contactless tag memory.

The table below shows the difference between the two protocols.

Table 53. Difference between HID and MSD protocols

USB transfer	Transfer type	Max. packet size
HID	Interrupt	64 bytes
MSD	bulk	512 bytes

6.2 Purpose of the USB mass storage

The purpose of the USB mass storage is twofold:

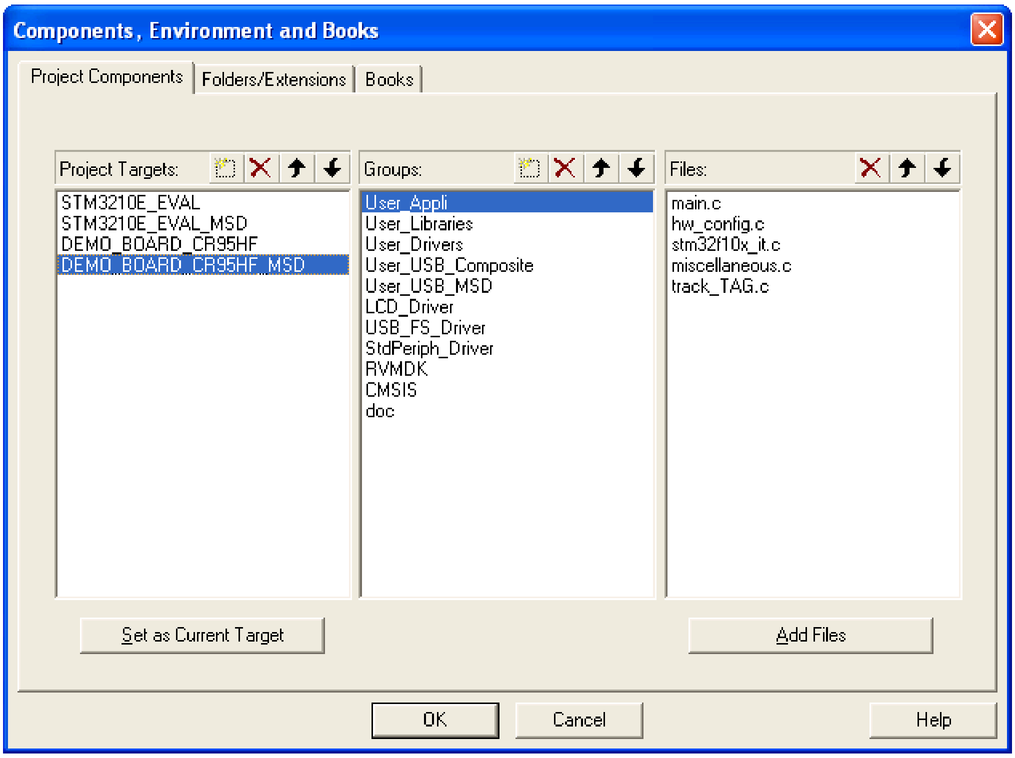
1. The first goal is to simplify the use of the RFID or NFC technology. The data of the tag is seen as a text file saved in a USB mass storage Key and can be easily transferred to a PC. The management of the RFID or NFC technology is done by the MCU.
2. The second goal is to improve the data transfer time between the PC and the contactless tag.

6.3 Activation of the USB mass storage device

6.3.1 Selection of the project target and compilation

The Keil project contains four project targets, as shown on *Figure 12*.

Figure 12. The four project targets available



The STM3210E_EVAL_MSD and DEMO_BOARD_CR95HF_MSD projects embed the USB mass storage. Both projects run respectively on the STM3210E_EVAL and DEMO_CR95HF_B boards.

The USB mass storage is an add-on of the HID feature, that is why the HID commands described in the previous chapter are still working.

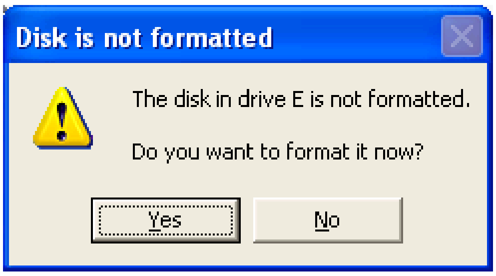
Once a target has been selected, all the target files shall be rebuilt.

6.3.2 Procedure at first use of the USB mass storage

Once the project has been compiled and the code loaded in the MCU, the DEMO_CR95HF_A is shown as a USB key device.

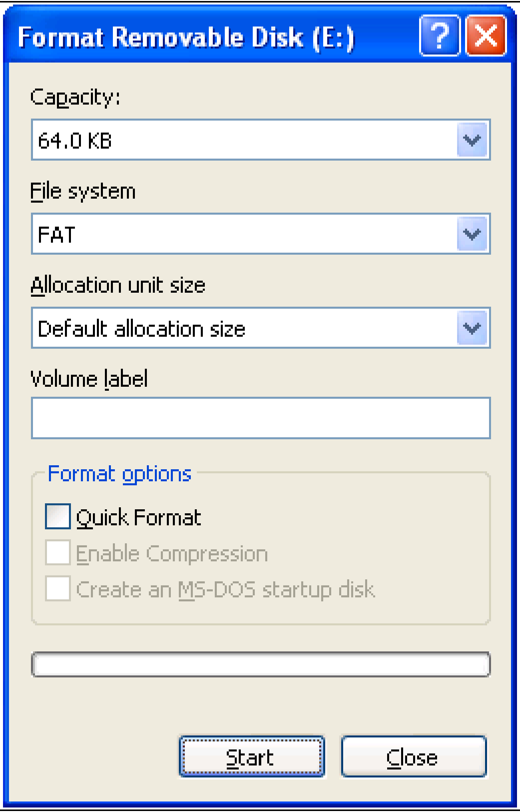
At the first use, the memory allocated to the USB mass storage device shall be formatted by the OS. When the pop-up window on *Figure 13* appears, click on Yes:

Figure 13. Disk is not formatted



In the new window that opens as on *Figure 14*, click on the start button:

Figure 14. Format Removable disk

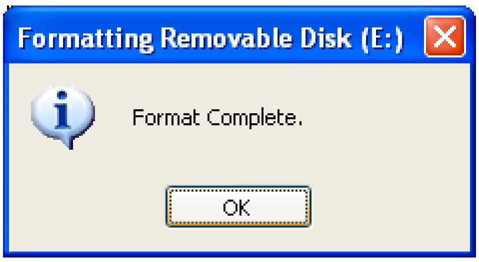


And on the next pop-up window, as on *Figure 15*, click the OK button:

Figure 15. Format removable disk warning

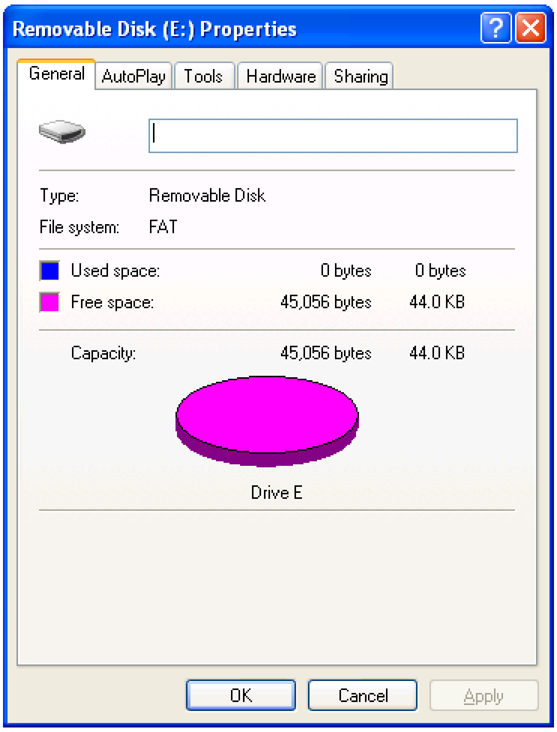
When the format has been completed, as on *Figure 16*, click the OK button:

Figure 16. Format complete



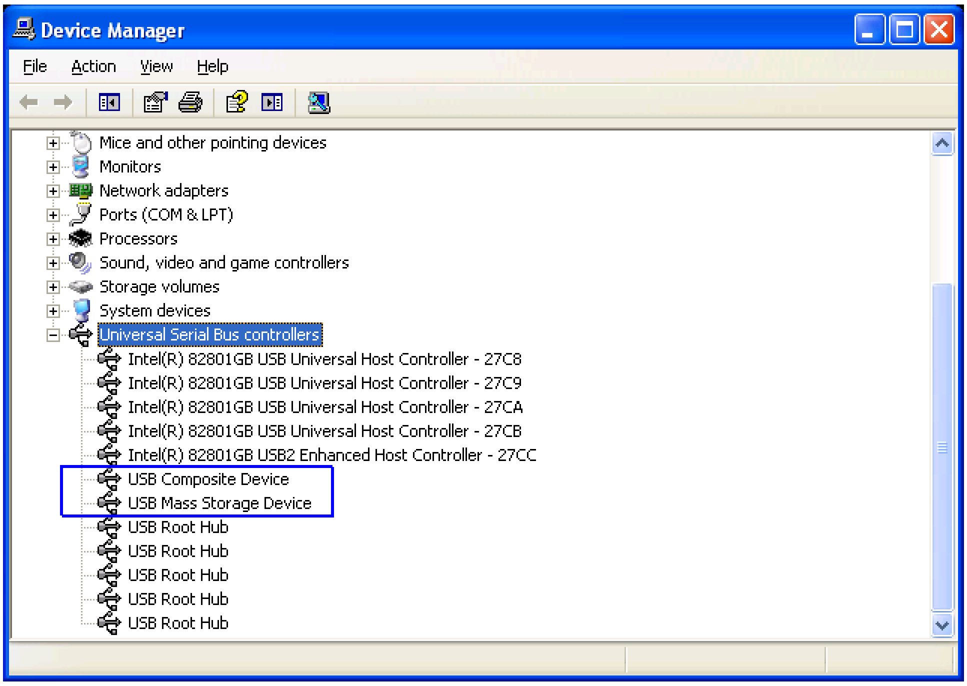
The USB key has now been detected, as can be seen on *Figure 17*:

Figure 17. USB key general properties



The DEMO_CR95HF_A board is now shown as a USB composite device and as a USB Mass storage device, as on *Figure 18*:

Figure 18. Device manager



6.4 USB mass storage limitation

The limitations of the USB mass storage are:

- The name of the file is limited to 5 characters and 3 extra characters for the extension, e.g. data1.txt is a correct file name.
- The memory used for the USB mass storage is the Flash memory of the MCU; its cycling endurance is limited to 10k times, and its size is limited.

7 USB mass storage functions

7.1 List of the USB mass storage functions

Table 54. USB mass storage functions

Name	Operating code	Brief description
CopyTagToFile	0xC1	Copies the memory of the ISO/IEC 15693 contactless tag to a bin file in the USB mass storage.
CopyFileTotag	0xC2	Copies a bin file to an ISO/IEC 15693 contactless tag in the USB mass storage.
SetUSBdisconnectPin	0xBB	Disconnects and reconnects the DEMO_CR95HF board from the USB bus.

7.1.1 DisconnectUSB function

This customs command disconnects and reconnects the DEMO_CR95HF_A board from the USB bus. The USB mass storage remains the power supply of the board.

When the MCU has created a new file in the USB mass storage, the MCU shall be disconnected and reconnected to update the USB key view of the PC.

7.1.2 HID command format

Table 55. HID command of the DisconnectUSB function

HID command format		
0x02	**0xBB**	**0x00**
Custom command		
Command code		

7.2 Copy the contactless tag memory to a bin file

This function reads the contactless tag memory and copies the data in a text file. This text file is stored in the MCU Flash memory and can be viewed by the PC.

At the end of this function, the MCU will disconnect and reconnect itself from the USB bus in order to force the PC to launch a new enumeration and find the new file.

7.3 HID command format

Table 56. HID command to copy the contactless tag memory to a bin file

HID command format				
0x02	**0xC1**	**0x04**	**2 bytes**	**2 bytes (1)**
Custom command				
Command code				
Number of bytes				
First block address to read				
Number of blocks to copy				

1. When this field is equal to 0x00 00, the whole contactless tag memory is copied

7.3.1 HID response format

The MCU does not return a response.

7.4 Copy the bin file to a contactless tag memory

This function copies a file of the MCU Flash memory to a contactless tag memory. The text file is stored in the MCU Flash memory and can be viewed by the PC.

7.4.1 HID command format

Table 57. HID command to copy the bin file to a contactless tag memory

HID command format				
0x02	**0xC2**	**0x04**	**2 bytes**	**2 bytes**
Custom command				
Command code				
Number of bytes				
First block address to read				
Number of blocks to copy				

7.4.2 HID response format

Table 58. HID response to copy the bin file to a contactless tag memory

HID response format	
0x80	**0x00**
Custom command	
Number of bytes	

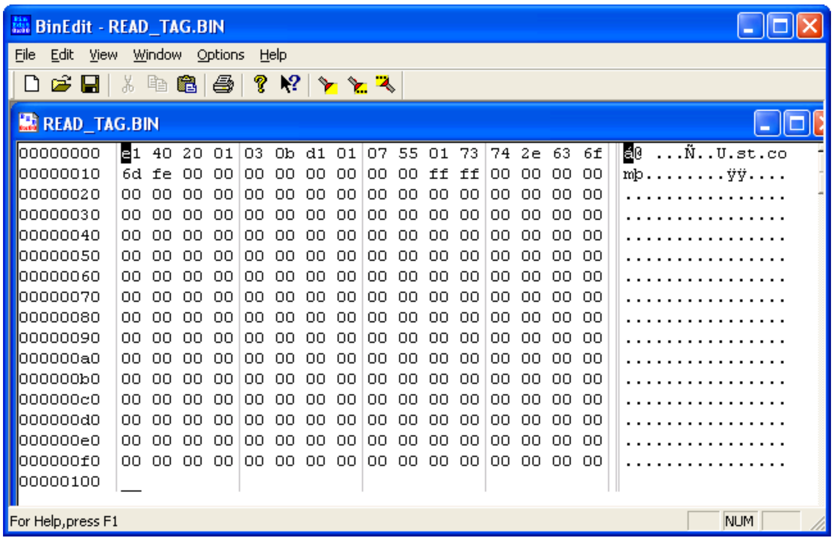
7.5 BinEdit software

The BinEdit software reads the bin files. This software is installed with either the CR95HF development software or the M24LRxx Application Software, as show on *Figure 19*:

Figure 19. CR95HF development software directory

The Read_Tag.bin files opened with the BinEdit File can be seen on *Figure 20*:

Figure 20. Read_Tag.bin files opened with the BinEdit File



Appendix A Acronym and notational conventions

A.1 Acronym

ISO: International Organization for Standardization

IEC: International Electrotechnical Commission

HID: Human interface device

MCU: Micro controller unit

MSD: Mass storage device

NFC: Near field communication

RF: Radio frequency

RFID: Radio Frequency Identification

USB: Universal Serial Bus

A.2 Representation of numbers

The following conventions and notations apply in this document unless otherwise stated.

A.2.1 Binary number representation

Binary numbers are represented by strings of digits 0 and 1 shown with the most significant bit (MSB) on the left, the least significant bit (LSB) on the right, and a "0b" added at the beginning.

Example: 0b11110101

A.2.2 Hexadecimal number representation

Hexadecimal numbers are represented by using the numbers 0 to 9 and the characters A - F, and adding an "0x" at the beginning. The Most Significant Byte (MSB) is shown on the left and the Least Significant Byte (LSB) on the right.

Example: 0xF5

A.2.3 Decimal number representation

Decimal numbers are represented as is without any trailing character.

Example: 245

Revision history

Table 59. Document revision history

Date	Revision	Changes
21-Nov-2012	1	Initial release.

Please Read Carefully: