AN3985
Application note

Using the CR95HF library with STM8L microcontrollers

1 Introduction

This document describes the CR95HF library allowing an STM8L microcontroller to drive the CR95HF 13.56 MHz multiprotocol contactless transceiver using an SPI or UART interface, in order to perform wireless communications with ISO/IEC15693 contactless tag.

The library was developed to speed up the development of applications using the CR95HF.

The CR95HF library is composed of three layers:

- Low level CR95HF layer
- Standard ISO/IEC 15693 protocol layer
- Product layer (LRIxK and Dual Interface EEPROM contactless tag)

The library code has been developed in ANSI C language, and validated on an STM8L evaluation board.

The firmware runs either on the STM8L1528-EVAL evaluation board or the STM8L-DISCOVERY board. The MCU can be either the STM8L152M8 (on the STM8L1528-Eval board) or the STM8L152C6 (STM8L-DISCOVERY).

1.1 Reference documents

- ISO/IEC 15693 specification
- LRIxK ISO/IEC 15693 contactless memory with 64-bit UID, AFI, DSFID, anti-collision and kill function datasheet
- CR95HF 13.56-MHz multi-protocol contactless transceiver IC with SPI and UART serial access datasheet
- M24LRxx dual interface EEPROM with password protection datasheet
- UM1037: STM8L1528-EVAL evaluation board user manual
- UM0970: STM8L-DISCOVERY evaluation board user manual

Contents

1 **Introduction** 1

1.1 Reference documents 1

2 **Acronyms and notational conventions** 9

2.1 List of terms 9

2.2 Notational conventions 10

2.2.1 Binary number representation 10

2.2.2 Hexadecimal number representation 10

2.2.3 Decimal number representation 10

3 **Overview** 11

3.1 CR95HF overview 11

3.2 Library overview 11

3.2.1 Example of an application architecture 11

3.2.2 Types 12

4 **CR95HF low level layer** 13

4.1 Overview 13

4.2 Structures 13

4.2.1 Command format 13

4.2.2 Response format 14

4.2.3 Protocol selection structure 14

4.2.4 Idle structure 15

4.3 CR95HF layer functions 16

4.3.1 Command functions 17

4.3.2 Additional functions 22

4.3.3 Low power mode functions 22

4.3.4 Is functions 24

4.3.5 Advanced functions 26

4.3.6 Application example: protocol selection and communication 28

5 **ISO/IEC 15693 layer** 29

5.1 Overview 29

5.2 Structures 29

5.2.1 Structures of command and response of an ISO/IEC 15693 tag 29
5.3 ISO/IEC 15693 command format 30
5.3.1 SOF and EOF 30
5.3.2 Request flag management 30
5.3.3 Command code and Data management 32
5.3.4 CRC16 management 32
5.4 ISO/IEC 15693 layer functions 33
5.4.1 Compute parameter byte functions 36
5.4.2 ISO/IEC 15693 command functions 38
5.4.3 Build up functions 47
5.4.4 Is Functions 48
5.4.5 Get functions 52
5.4.6 CRC16 functions 57
5.4.7 Fill In functions 58

6 LRIxK layer 60
6.1 Overview 60
6.2 Command format 60
6.2.1 CRC16 management 60
6.2.2 Request flag management 60
6.2.3 Request flags and CR95HF_ProtocolSelect functions 61
6.3 LRIxK layer commands 62
6.3.1 LRIxK command functions 63

7 M24LRxx layer 66
7.1 Overview 66
7.2 Command format 66
7.2.1 CRC16 management 66
7.2.2 Request flag management 66
7.2.3 Request flags and CR95HF_ProtocolSelect functions 68
7.3 M24LRxx commands 68
7.3.1 M24LRxx command functions 69
7.3.2 M24LRxx Energy Harvesting functions 81

8 Application example 86
8.1 Main functions 87

8.1.1 Board initialization . . . 88
8.1.2 Display initialization . . . 88
8.1.3 Test configuration . . . 88
8.1.4 ISO/IEC 15693 Protocol selection . . . 89
8.1.5 Tag hunting . . . 89
8.1.6 Display . . . 90
8.1.7 User application . . . 90
8.1.8 Low power modes . . . 90
8.1.9 Communication test . . . 94
8.2 Hardware . . . 96
8.2.1 STM8L discovery board . . . 96
8.2.2 STM8L evaluation board . . . 96
8.2.3 CR95HF plug board . . . 96
8.3 Software . . . 97
8.3.1 ST Visual Develop . . . 97
8.3.2 Cosmic compiler . . . 97
8.4 Project . . . 97
8.4.1 Opening the Project: . . . 97
8.4.2 Compilation / Debug . . . 98
8.5 Compilation management . . . 99
8.5.1 Conditional compilation . . . 99
8.5.2 Polling method . . . 100
8.6 Hardware layout and configuration . . . 100
8.7 Pinout description . . . 101
8.7.1 PLUG-CR95HF-B Board pin . . . 103
8.8 Switching between STM8L 1528-Eval and STM8L Discovery boards . . 103

9 Revision history . . . 104

List of tables

Table 1. List of terms . . . 9
Table 2. Command fields formats . . . 13
Table 3. Response field formats . . . 14
Table 4. Protocol parameter formats . . . 14
Table 5. Protocol values . . . 15
Table 6. Idle structure parameters . . . 15
Table 7. CR95HF layer functions based on CR95HF commands . . . 16
Table 8. CR95HF layer additional functions . . . 16
Table 9. Low Power mode functions . . . 16
Table 10. CR95HF layer IS functions . . . 16
Table 11. CR95HF layer advanced functions . . . 17
Table 12. CR95HF_IDN function description . . . 17
Table 13. CR95HF_Echo function description . . . 17
Table 14. CR95HF_ProtocolSelect function description . . . 18
Table 15. Input parameters settings for ISO/IEC 15693 protocol . . . 18
Table 16. CR95HF_SendRecv function description . . . 19
Table 17. SendRecv correct response . . . 19
Table 18. SendRecv error response . . . 19
Table 19. CR95HF_Idle function description . . . 20
Table 20. CR95HF_Rd_Wake_up_Reg function description . . . 20
Table 21. CR95HF_Rd_Analog_Register_Config_B function description . . . 21
Table 22. CR95HF_Baud_Rate function description . . . 21
Table 23. CR95HF_SendEOF function description . . . 22
Table 24. CR95HF_FieldOff function description . . . 22
Table 25. CR95HF_Hibernate function description . . . 22
Table 26. CR95HF_Sleep function description . . . 23
Table 27. CR95HF_TagDetecting function description . . . 23
Table 28. CR95HF_IdlebyTimer function description . . . 24
Table 29. CR95HF_IsReaderResultOK function description . . . 24
Table 30. Is_IRQ_in_Interupt_Wake_up_Condition function description . . . 24
Table 31. Is_NSS_Interrupt_Wake_up_Condition function description . . . 25
Table 32. Is_Tag_Detected_Wake_up_Condition function description . . . 25
Table 33. Is_Tag_Detected_Wake_up_Condition function description . . . 25
Table 34. CR95HF_Modify_Baud_Rate function description . . . 26
Table 35. CR95HF_Idle_Detector_Calibration function description . . . 26
Table 36. CR95HF_Wait_Wake_up_From_Idle function description . . . 27
Table 37. CR95HF_Wake_up_CR95HF function description . . . 27
Table 38. ISO15693_tag structure description . . . 29
Table 39. Request flag bits description . . . 31
Table 40. Request flag bits values . . . 31
Table 41. Bit for request flag with inventory flag reset . . . 31
Table 42. Bit for request flag with inventory flag reset . . . 32
Table 43. Functions to compute parameter byte . . . 33
Table 44. ISO15693 library command based on ISO/IEC 15693 specification . . . 33
Table 45. Functions to assemble ISO/IEC 15693 command . . . 34
Table 46. Get functions . . . 35
Table 47. CRC16 functions . . . 35
Table 48. Structure Fill In functions . . . 36
Table 49. ISO15693_ComputeParameterByte_ProtocolSelect function description . . . 36
Table 50. ComputeByte function description . . . 37
Table 51. ISO15693_Inventory function description . . . 38

Table 52. ISO15693_StayQuiet function description . . . 38
Table 53. ISO15693_Read_Single_Block function description . . . 39
Table 54. ISO15693_Write_Single_Block function description . . . 39
Table 55. ISO15693_Write_Single_Block results depending on Option flag . . . 40
Table 56. ISO15693_Lock_Block function description . . . 40
Table 57. ISO15693_Read_Multiple_Block function description . . . 41
Table 58. ISO15693_Write_Multiple_Block function description . . . 41
Table 59. ISO15693_Select function description . . . 42
Table 60. ISO15693_Reset_to_Ready function description . . . 42
Table 61. ISO15693_Write_AFI function description . . . 43
Table 62. ISO15693_ Lock_AFI function description . . . 43
Table 63. ISO15693_ WriteDSFID function description . . . 44
Table 64. ISO15693_Lock_DSFID function description . . . 44
Table 65. ISO15693_Get_System_Info function description . . . 45
Table 66. ISO15693_Get_Multiple_Blocks_Security_Status function description . . . 45
Table 67. ISO15693_Custom_Commands function description . . . 46
Table 68. ISO15693_SendEOF function description . . . 46
Table 69. ISO15693_Add_UID_to_command description . . . 47
Table 70. ISO15693_Add_Mask_to_command function description . . . 47
Table 71. ISO15693_Add_Request_Flags_Command_code_to_command function description . . . 48
Table 72. ISO15693_Is_ResponseFlags_noError_Detected function description . . . 48
Table 73. ISO15693_Is_Request_Flags_ok function description . . . 48
Table 74. ISO15693_Is_Collision_Flag_Set function description . . . 49
Table 75. ISO15693_IsCRC_Flag_Set function description . . . 49
Table 76. ISO15693_Is_CorrectCRC16_Check function description . . . 49
Table 77. ISO15693_Is_TagConstructor_STM function description . . . 50
Table 78. ISO15693_Is_Memory_programming function description . . . 50
Table 79. ISO15693_Is_DSFID_Present_Infoflag function description . . . 50
Table 80. ISO15693_Is_AFI_Present_Infoflag function description . . . 51
Table 81. ISO15693_Is_MemSize_Present_Infoflag function description . . . 51
Table 82. ISO15693_Is_ICRef_Present_Infoflag function description . . . 51
Table 83. ISO15693_Get_Subcarrier_Flag function description . . . 52
Table 84. ISO15693_Get_DataRate_Flag function description . . . 52
Table 85. ISO15693_Get_Inventory_Flag function description . . . 52
Table 86. ISO15693_Get_ProtocolExtension_Flag function description . . . 53
Table 87. ISO15693_Get_Select_Flag function description . . . 53
Table 88. ISO15693_Get_Address_Flag function description . . . 53
Table 89. ISO15693_Get_AFI_Flag function description . . . 54
Table 90. ISO15693_Get_Slots_Flag function description . . . 54
Table 91. ISO15693_Get_Option_Flag function description . . . 54
Table 92. ISO15693_Get_RFU_Flag function description . . . 55
Table 93. ISO15693_Get_NumberofByte_MemSize_using_ICREF function description . . . 55
Table 94. ISO15693_Get_Size_of_Block_using_ICRef function description . . . 55
Table 95. ISO15693_Get_Size_of_Block_using_MemSize function description . . . 56
Table 96. ISO15693_Get_MemSize_kbits function . . . 56
Table 97. ISO15693_Get_MemSize_bits function description . . . 56
Table 98. ISO15693_Get_UID_from_EEPROM function description . . . 57
Table 99. ISO15693_CRC16 function description . . . 57
Table 100. ISO15693_IsCorrectCRC16Residue function description . . . 57
Table 101. ISO15693_CRC16_EEPROM function description . . . 58
Table 102. ISO15693_IsCorrectCRC16Residue_EEPROM function description . . . 58
Table 103. ISO156693_Check_Data_Received_Start_Process function description . . . 58
Table 104. ISO15693_Retrieve_UID function description . . . 59
Table 105. ISO15693_Retrieve_ICRef function . . . 59

Table 106. Request flag management description . . . 61
Table 107. LRIxK layer commands description . . . 62
Table 108. LRIxK_Kill function description . . . 63
Table 109. LRIxK_Write_Kill function description . . . 63
Table 110. LRIxK_Lock_Kill function description . . . 64
Table 111. LRIxK_Inventory_Initiated function description . . . 64
Table 112. LRIxK_Initiate function description . . . 65
Table 113. ISO15693_Fast_Read_Single_Block function description . . . 65
Table 114. M24LR* layer forced request flags . . . 67
Table 115. Relationship between M24LRxx and ISO/IEC 15693 layer . . . 68
Table 116. M24LRxx_Read_Single_Block function description . . . 70
Table 117. M24LRxx_Write_Single_Block function description . . . 70
Table 118. M24LRxx_Read_Multiple_Block function description . . . 71
Table 119. M24LRxx_Get_System_Info function description . . . 71
Table 120. M24LRxx_Get_Multiple_Blocks_Security_Status function description . . . 72
Table 121. M24LRxx_Write_Sector_Password function description . . . 72
Table 122. M24LRxx_Lock_Sector_Password function description . . . 73
Table 123. M24LRxx_Present_Sector_Password function description . . . 73
Table 124. M24LRxx_Fast_Read_Single_Block function description . . . 74
Table 125. M24LRxx_Fast_Read_Multiple_Block function description . . . 74
Table 126. M24LRxx_Inventory_Initiated function description . . . 75
Table 127. M24LRxx_Fast_Inventory_Initiated function description . . . 75
Table 128. M24LRxx_Get_System_Info function description . . . 76
Table 129. M24LRxx_Get_Multiple_Blocks_Security_Status function description . . . 76
Table 130. M24LRxx_Write_Sector_Password function description . . . 77
Table 131. M24LRxx_Lock_Sector_Password function description . . . 77
Table 132. M24LRxx_Present_Sector_Password function description . . . 78
Table 133. M24LRxx_Fast_Read_Single_Block function description . . . 78
Table 134. M24LRxx_Fast_Read_Multiple_Block function description . . . 79
Table 135. M24LRxx_Inventory_Initiated function description . . . 79
Table 136. M24LRxx_Fast_Inventory_Initiated function description . . . 80
Table 137. M24LRxx_Initiate function description . . . 80
Table 138. M24LRxx_Fast_Initiate function description . . . 81
Table 139. M24LRxx_ReadCfg function description . . . 81
Table 140. M24LRxx_Write_EH_Cfg function description . . . 82
Table 141. M24LRxx_SetRst_EH_en function description . . . 83
Table 142. M24LRxx_Check_EH_En function description . . . 83
Table 143. M24LRxx_Get_Energy_Harvesting_Range function description . . . 84
Table 144. M24LRxx_Get_RF_BUSY_WIP function description . . . 84
Table 145. M24LRxx_Get_EH_mode_Configuration_Byte function description . . . 85
Table 146. M24LRxx_Get_EH_mode_Control_Register function description . . . 85
Table 147. Test config procedure . . . 88
Table 148. Low power modes descriptions . . . 91
Table 149. Functionalities description . . . 99
Table 150. Communication with CR95HF I/Os . . . 101
Table 151. Board STM8L-1528Eval Specific I/Os . . . 102
Table 152. Board STM8L-Discovery Board Specific I/Os . . . 102
Table 153. Switching procedure . . . 103
Table 154. Document revision history . . . 104

List of figures

Figure 1. Typical application block diagram . . . 11
Figure 2. Interaction between typical user application and CR95HF library layers . . . 12
Figure 3. Function flowchart example . . . 28
Figure 4. Application example main functions . . . 87
Figure 5. Test config procedure flowchart . . . 89
Figure 6. Joystick utilization . . . 90
Figure 7. Wait for interrupt mode schematic . . . 91
Figure 8. Wait for interrupt mode flowchart . . . 92
Figure 9. Halt mode schematic . . . 92
Figure 10. Halt mode flowchart . . . 93
Figure 11. Halt CR95HF Timer mode schematic . . . 93
Figure 12. Halt CR95HF Timer mode flowchart . . . 94
Figure 13. Communication test flowchart . . . 95
Figure 14. Project tree structure . . . 97
Figure 15. Debug Instruments Settings dialog box . . . 98
Figure 16. STM8L1528-EVAL hardware block diagram . . . 101
Figure 17. PLUG-CR95HF-B Board I/Os . . . 103

2 Acronyms and notational conventions

2.1 List of terms

Table 1. List of terms

Acronyms	Definitions
AFI	Application Family Identifier
CRC	Cyclic Redundancy Check
CPU	Central Processing Unit
DAC	Digital to Analog Converter
DSFID	Data Storage Format IDentifier
EEPROM	Electrically Erasable Programmable Read-Only Memory
EOF	End Of Frame
HFO	High Frequency Oscillator
IC	Integrated Circuit
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
FIFO	First In First Out
LRI	Long Range Interface
LSB	Least Significant Bit
M24LR64-R	Dual interface EEPROM (I2C & RF) with 64-kbit memory size
M24LR16-E	Dual interface EEPROM (I2C & RF) with 16-kbit memory size and energy harvesting functionality
MCU	Microcontroller Unit
MSB	Most Significant Bit
NFC	Near Field Communication
POR	Power On Reset
RTC	Real Time Clock
RF	Radio Frequency
RFU	Reserved for Future Use
RFID	Radio Frequency Identification
STVD	ST Visual Develop
UID	Unique Identifier
Write_DO_Cfg	Write Digital Output Configuration (M24LRxx-E command)
Write_EH_Cfg	Write Energy Harvesting Configuration (M24LRxx-E command)
XOR	eXclusive OR

2.2 Notational conventions

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

2.2.1 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 “0b” added at the beginning.

For example: 0b11110101

2.2.2 Hexadecimal number representation

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

For example: 0xF5

2.2.3 Decimal number representation

Decimal numbers are represented as is, without any trailing character.

For example: 245

3 Overview

3.1 CR95HF overview

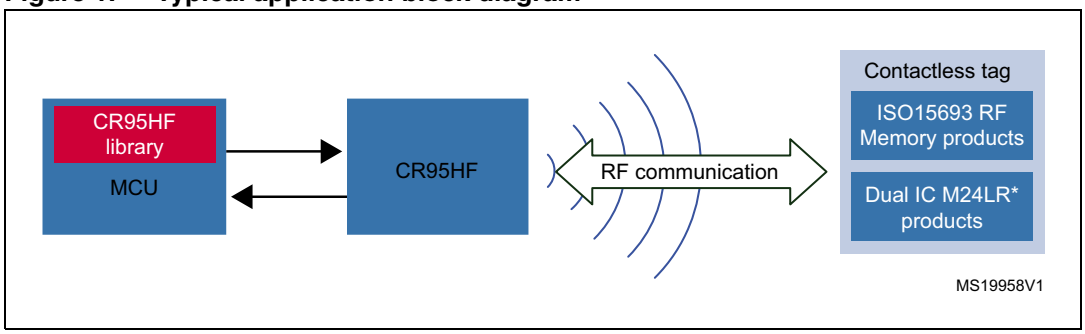
The CR95HF is a Radio Frequency (RF) transceiver Integrated Circuit (IC) for 13.56 MHz contactless tags, which includes ISO/IEC 14443, ISO/IEC 15693 and ISO/IEC 18092 protocols. It manages the RF communication with Radio Frequency Identification (RFID) or Near Field Communication (NFC) tags. It includes frame coding, RF modulation and contactless tag response decoding.

The CR95HF is a slave device and must be controlled by a host (Microcontroller Unit). This library is an interface between user application function and standard peripheral driver.

The library was written in compliance with ANSI C standards.

Figure 1 describes a typical application block diagram.

Figure 1. Typical application block diagram



For more details concerning the CR95HF, please refer to CR95HF datasheet.

3.2 Library overview

The library described in this application note is composed of three layers:

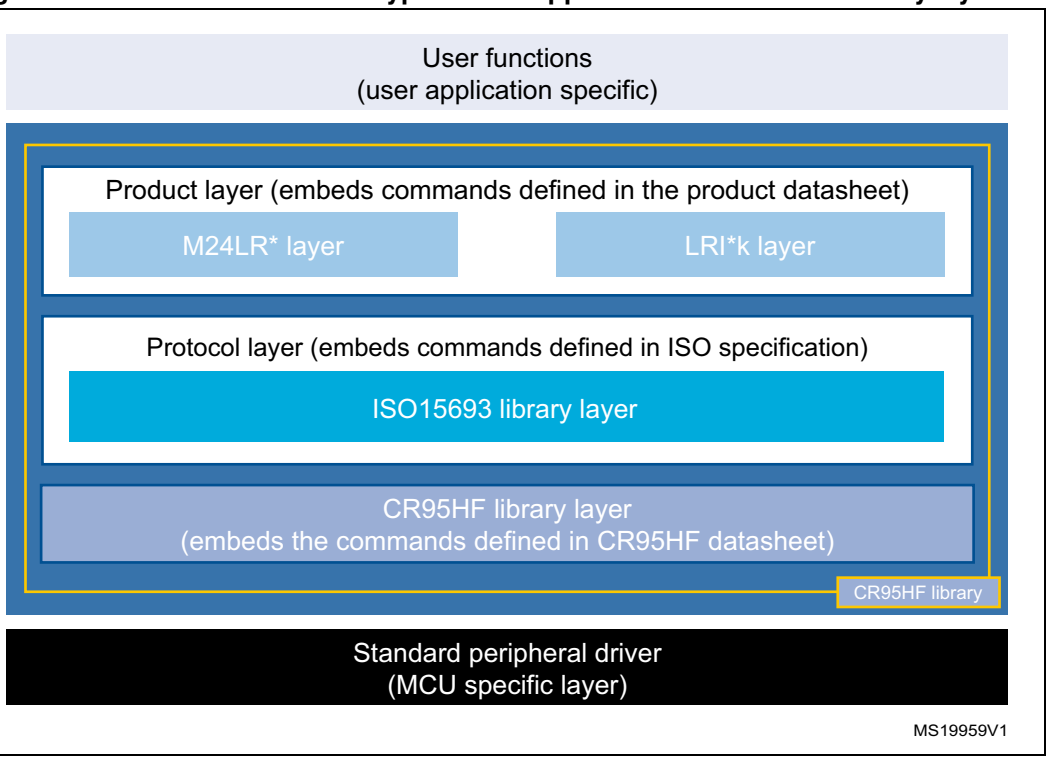
- A low level layer supporting the commands described in the CR95HF datasheet. This level is fully supported by the CR95HF library.
- An intermediate layer based on ISO/IEC 15693 specification
- A product layer supporting the commands described in the LRIxK and M24LRxx datasheets

3.2.1 Example of an application architecture

The library can be downloaded from http://www.st.com.

Figure 2 shows the interaction between a typical user application and the CR95HF library layers.

Figure 2. Interaction between typical user application and CR95HF library layers



3.2.2 Types

The CR95HF library functions use the following ANSI C compliant types defined in the *stm8l15x.h* file.

Signed integer types

```
typedef       signed char         int8_t;
typedef       signed short        int16_t;
typedef       signed long         int32_t;
typedef       int32_t             s32;
typedef       int16_t             s16;
typedef       int8_t              s8;
```

Unsigned integer types

```
typedef       unsigned char       uint8_t;
typedef       unsigned short      uint16_t;
typedef       unsigned long       uint32_t;
typedef       uint32_t            u32;
typedef       uint16_t            u16;
typedef       uint8_t             u8;
typedef       unsigned char       uint8_t;
typedef       signed char         int8_t;
typedef       const uint8_t       uc8;
typedef       signed short int    int16_t;
```

4 CR95HF low level layer

This layer is composed of:

- the *lib_CR95HF.c* source file
- the *lib_CR95HF.h* include file

4.1 Overview

This layer supports all the commands defined in the CR95HF datasheet. Each CR95HF command corresponds to a specific function. As an example, the function calling the ECHO command is `u8 CR95HF_Echo (void)`.

Additional functions are described in *Table 8: CR95HF layer additional functions*.

Application developers can use the functions described in *Table 7: CR95HF layer functions based on CR95HF commands* to create their own higher level functions.

Refer to the CR95HF datasheet for more details on the commands.

4.2 Structures

4.2.1 Command format

A command from MCU to CR95HF is composed of three fields:

- command code
- length
- data

The dedicated structure is `CR95HF_CommandSending`.

Table 2 lists the fields of a generic command to CR95HF, the number of bytes and the name of the dedicated structure member.

Table 2. Command fields formats

Field	Length	Dedicated structure
Command code	1 byte	`cmd_code`
Length	1 byte	`data_lenght`
Data	Up to 255 bytes	`data`

The structure definition of a command to CR95HF is:

```
typedef struct
{
u8 cmd_code;
u8 data_length;
u8 data[MAX_DATASENT_LENGTH];
}CR95HF_CommandSending;
```

4.2.2 Response format

A response from CR95HF to MCU is composed of three fields:

- response code
- length
- data

The dedicated structure is `CR95HF_ResponseReceiving`.

Table 3 lists the fields of a generic response from CR95HF, the number of bytes and the name of the dedicated structure member.

Table 3. Response field formats

Field	Length	Dedicated structure
RF protocol	1 byte	`resp_code`
Length	1 byte	`data_lenght`
Data	Up to 255 bytes	`data`

The structure definition of a response from CR95HF is:

```
typedef struct
{
u8 resp_code;
u8 data_length;
u8 data[MAX_DATARECEIVED_LENGTH];
}CR95HF_CommandSending;
```

4.2.3 Protocol selection structure

The CR95HF can use different RF protocols:

- ISO/IEC 14443 type A or type B
- ISO/IEC 15693
- ISO/IEC 18092

Each protocol has its own parameters.

The `Protocol_Config` structure displays the RF protocol parameters.

Table 4 lists the protocol parameters.

Table 4. Protocol parameter formats

Field	Length	Dedicated structure
Response code	1 byte	`protocol_in_use`
Length	1 byte	`parameters_lenght`
Data	Up to 3 bytes	`parameters`

The structure definition of a protocol selection structure is:

```
typedef struct
{
```

```
u8 protocol_in_use;
u8 parameters_lenght;
u8 paramaters[MAX_PARAMAETERS_LENGTH];
}Protocol_Config;
```

Table 5 lists the available values for `protocol_in_use`.

Table 5. Protocol values

RF protocol	Values
ISO/IEC 15693	1
ISO/IEC 14443 type A	2
ISO/IEC 14443 type B	3
ISO/IEC 18092	4

Note: These values are defined in CR95HF_command.h.

4.2.4 Idle structure

The idle structure contains the last parameters of the idle command sent to CR95HF.

Table 6 lists the parameters contained in an idle structure.

Table 6. Idle structure parameters

Parameter	Length	Dedicated structure
Wake-up flags	1 byte	`wuFlags`
Low threshold for tag detector mode	1 byte	`dacDataL`
High threshold for tag detector mode	1 byte	`dacDataH`
Cause of last wake-up exit	1 byte	`last_reason_of_wakeup`

The structure definition of an idle structure is:

```
typedef struct
{
u8 wuFlags;
u8 dacDataL;
u8 dacDataH;
u8 last_reason_of_wakeup;
}CR95HF_Idle_Config;
```

Note: This structure is defined in `CR95HF_structure.h file`.

For more details about idle command, refer to CR95HF datasheet.

4.3 CR95HF layer functions

The tables below list the functions available in a CR95HF layer.

Table 7. CR95HF layer functions based on CR95HF commands

Function name	Description
CR95HF_IDN	Sends an Idn command.
CR95HF_Echo	Sends an Echo command.
CR95HF_ProtocolSelect	Sends a ProtocolSelect command.
CR95HF_SendRecv	Sends a SendRecv command.
CR95HF_Idle	Sends an Idle command.
CR95HF_Rd_Wake_up_Reg	Reads the wake-up Register command.
CR95HF_Rd_Analog_Register_Config_B	Reads the analog register configuration B command.
CR95HF_BaudRate	Sends a BaudRate command.

Table 8. CR95HF layer additional functions

Function name	Description
CR95HF_SendEOF	Sends an EOF pulse.
CR95HF_FieldOff	Switches off the RF field.

Table 9. Low Power mode functions

Function name	Description
CR95HF_Hibernate	Sends an hibernate command to CR95HF.
CR95HF_Sleep	Sends a Sleep command to CR95HF.
CR95HF_IdlebyTimerfunction	Sends a Idle command to CR95HF. The wake-up source is its internal timer.
CR95HF_TagDetecting	Sends a Tag detecting command to CR95HF.

Table 10. CR95HF layer IS functions

Function name	Description
CR95HF_IsReaderResultOK	Checks if the returned code is successful.
CR95HF_Is_IRQ_in_Interrupt_Wake_up_Condition	Checks if a pulse on $\overline{\text{IRQ_IN}}$ pad is a wake-up condition.
CR95HF_Is_NSS_Interrupt_Wake_up_Condition	Checks if a pulse on NSS pad is a wake-up condition.
CR95HF_Is_Tag_Detected_Wake_up_Condition	Checks if a tag detector state is a wake-up condition.
CR95HF_Is_Timeout_Wake_up_Condition	Checks if an internal timer is a wake-up condition.

Table 11. CR95HF layer advanced functions

Function name	Description
`CR95HF_Modify_Baud_Rate`	Changes the UART baud rate of MCU and CR95HF.
`CR95HF_Idle_Detector_Calibration`	Carries out a calibration for tag detector mode.
`CR95HF_Wait_Wake_up_From_Idle`	Waits for CR95HF to exit the idle state and retrieve what caused the wake-up.
`CR95HF_Wake_up_CR95HF`	Wakes up CR95HF when exists idle mode.

4.3.1 Command functions

CR95HF_IDN function

This function sends an IDN command to the CR95HF device. It returns its version number.

Table 12. CR95HF_IDN function description

Prototype	`u8 CR95HF_IDN (CR95HF_ResponseReceiving* Response);`
Input parameter	None
Output parameter	Response: pointer on the response structure
Return parameter	**CR95HF_SUCCESS_CODE**: The command is successful. **CR95HF_ERROR_CODE**: The command failed.

CR95HF_Echo function

This function sends an EchoCode command to the CR95HF which returns an Echo code response (0x55). The Echo function checks that communications can be started between the MCU and the CR95HF.

Table 13. CR95HF_Echo function description

Prototype	`u8 CR95HF_Echo (void);`
Input parameter	None
Output parameter	None
Return parameter	**CR95HF_SUCCESS_CODE**: The command is successful. **CR95HF_ERROR_CODE**: The command failed.

CR95HF_ProtocolSelect function

This function sends a Protocol Select command to the CR95HF. It selects the RF communication protocol, configures RF parameters, and switches the RF field on.To set up communications with a contactless tag, the Protocol Select command shall be sent to the CR95HF before the SendRecv command.

Table 14. CR95HF_ProtocolSelect function description

Prototype	`u8 CR95HF_ProtocolSelect (const u8 Protocol, const u8 ParametersLength,const u8* Parameters, CR95HF_Protocol_Config* Protocol_Settings);`
Input parameter(1)	**Protocol**: RF protocol selected **ParametersLength**: length of any parameters attached **Parameters**: any parameters to be attached for different protocols
Output parameter	**Protocol_Settings**: structure at the end of the command containing the last protocol selected and its parameters
Return parameter	**CR95HF_SUCCESS_CODE**: The command is successful. **CR95HF_ERROR_INVALID_PROTOCOL**: The protocol selected is invalid. **CR95HF_ERROR_INVALID_CMD_LENGTH**: The command length is invalid. **CR95HF_ERROR_CODE**: The command failed.

1. The input parameter settings depend on the selected protocol. This application note applies to ISO/IEC 15693 products.

Table 15. Input parameters settings for ISO/IEC 15693 protocol

Parameter name	Byte	Bit	Value	Example
Command code	0		0x02	02020126
Length	1		0x02	
Protocol	2		0x01: ISO/IEC 15693 protocol	
Parameter	3	7:6	RFU	
		5:4	0b00: 26 kbps	
			0b01: 52 kbps	
			0b10: 6 kbps	
			0b11: RFU	
		3	0b0: ensure a 312 µs delay	
			0b1: wait for SOF	
		2	0b0: 100% modulation	
			0b1: 10% modulation	
		1	0b0: single subcarrier	
			0b1: single subcarrier	
		0	0b0: don't append CRC	
			0b1: append CRC (1)	

1. It is recommended to set the append CRC bit (see *Chapter 5.3.4: CRC16 management*)

For more detail concerning this parameter, please refer to the CR95HF datasheet.

CR95HF_SendRecv function

This function sends a SendRecv command. The parameter passed to the command is the frame, which is coded according to the protocol previously selected by issuing a Protocol Select command. The CR95HF encodes the frame, transmits it at 13.56 MHz, and decodes the contactless tag response.

Table 16. CR95HF_SendRecv function description

Prototype	`u8 CR95HF_SendRecv(CR95HF_ResponseReceiving* Response,const CR95HF_CommandSending* Command);`
Input parameter	**Command**: the command structure to send to a contactless tag
Output parameter	**Response**: pointer on the response structure
Return parameter	**CR95HF_SUCCESS_CODE**: The command is successful. **CR95HF_ERROR_CODE**: The command failed. `Response` contains the right error code.

The Parameters of input parameter depend on the selected protocol. *Table 15* gives ISO/IEC 15693 protocol parameters.

In case of a correct response, the SendRecv response sent to MCU is as follows:

Table 17. SendRecv correct response

Parameter name	Byte	Bit	Value	Example
Result Code	0	7:0	0x80	0x800D0000FA20563C172202E0746800 with: 80: success code 0D: number of bytes 0000FA20563C172202E0: contactless tag data 7468: CRC 00: control byte
Length	1	7:0	0xXX	
Data	2:X	7:0	0xXX	
CRC	X+2:X+3	7:0	0xXX	
Control byte	X+4	7:2	RFU	
		1	CRC error if set	
		0	Collision detected if set	

In case of an error response, the SendRecv response sent to MCU is as follows:

Table 18. SendRecv error response

Parameter name	Byte	Bit	Value	Example
Result Code	0	7:0	0x8X	0x 87 00: no tag in the field
Length	1	7:0	0x00	

For more details, please refer to the CR95HF datasheet.

CR95HF_Idle function

This function sends an Idle command to the CR95HF to switch the CR95HF device into low consumption mode.

Table 19. CR95HF_Idle function description

Prototype	`u8 CR95HF_Idle (const u8 WakeUpFlags,const u8 EnterCtrlL,const u8 EnterCtrlH, const u8 WUCtrlL,const u8 WUCtrlH,const u8 LeaveCtrlL,const u8 LeaveCtrlH ,const u8 WUPeriod,const u8 OscStart,const u8 DacStart,const u8 DacDataL,const u8 DacDataH,const u8 SwingsCnt,const u8 MaxSleep, CR95HF_Idle_Config* Idle_Config);`
Input parameter	**WakeUpFlags**: Specifies wake-up condition. **EnterCtrlL**: first byte of setting to enter to Idle mode **EnterCtrlH**: second byte of setting to enter to Idle mode **WUCtrlL**: first byte of setting to wake-up from Idle mode **WUCtrlH**: second byte of setting to wake-up from Idle mode **LeaveCtrlL**: first byte of setting to leave Idle mode **LeaveCtrlH**: second byte of setting to leave Idle mode **WUPreiod**: period of time between two tags detection **OscStart**: waiting time to stabilize HFO **DacStart**: waiting time to stabilize DAC **DacDataL**: lower compare value for tag detection **DacDataH**: higher compare value for tag detection **SwingsCnt**: number of HF swings during tag detection **MaxSleep**: maximum number of tag detection trials before timeout
Output parameter	**Idle_Config**: pointer on the Idle structure
Return parameter	**CR95HF_SUCCESS_CODE**: The command is successful.

Note: Some Low power mode commands using the Idle command are defined in Chapter 4.3.3: Low power mode functions

CR95HF_Rd_Wake_up_Reg function

This function reads the wake-up register of the CR95HF to determine what has caused the awakening of the CR95HF after an Idle command.

Table 20. CR95HF_Rd_Wake_up_Reg function description

Prototype	`u8 CR95HF_Rd_Wake_up_Reg(CR95HF_ResponseReceiving* Response,CR95HF_Idle_Config* Idle_Config);`
Input parameter	None
Output parameter	**Response**: pointer on the structure to fill in with the data retrieved **Idle_Config**: pointer on the structure to update
Return parameter	**CR95HF_SUCCESS_CODE**: The command is successful. **CR95HF_ERROR_INVALID_CMD_LENGTH**:The command failed, invalid length. **CR95HF_ERROR_CODE**: Command failed, `response` contains the error code.

CR95HF_Rd_Analog_Register_Config_B function

This function reads the analog register configuration B of the CR95HF to check the analog configuration Register containing the modulation depth and reader chain gain.

Note: A `CR95HF_Set_Analog_Register_Config_B_Index` command must be issued before reading the register.

Table 21. CR95HF_Rd_Analog_Register_Config_B function description

Prototype	`u8 CR95HF_Rd_Wake_up_Reg(CR95HF_ResponseReceiving* Response,CR95HF_Idle_Config* Idle_Config);`
Input parameter	None
Output parameter	**Response**: pointer on the structure to fill in with the data retrieved **Idle_Config**: pointer on the structure to update
Return parameter	**CR95HF_SUCCESS_CODE**: The command is successful. **CR95HF_ERROR_INVALID_CMD_LENGTH**: The command failed, invalid length **CR95HF_ERROR_CODE**: Unsuccessful command, `Response` contains the error code.

CR95HF_Baud_Rate function

This function sends a BaudRate command to the CR95HF. It allows to configure the UART baudrate.

Table 22. CR95HF_Baud_Rate function description

Prototype	`u8 CR95HF_Baud_Rate(u8 New_Baud_Rate);`
Input parameter	**New_Baud_Rate**: new baud rate = 13.56/(2*BaudRate+2) Mbps
Output parameter	**pResponse**: pointer on the CR95HF response
Return parameter	**CR95HF_SUCCESS_CODE**: The command is successful. **CR95HF_ERROR_CODE**: Unsuccessful command, UART communication may be lost.

4.3.2 Additional functions

CR95HF_SendEOF function

This function sends an EOF. This can be used for instance in an ISO/IEC 15693 inventory or write command. Right after the EOF, the tag could answer a command (Write or Inventory), so the response is received and written into the provided structure.

Table 23. CR95HF_SendEOF function description

Prototype	`u8 CR95HF_SendEOF(CR95HF_ResponseReceiving* Response);`
Input parameter	None
Output parameter	**Response**: pointer on the structure which will contain the Tag answer
Return parameter	**CR95HF_SUCCESS_CODE**: The command is successful. **CR95HF_ERROR_CODE**: The command failed.

CR95HF_FieldOff function

This function switches off the RF Field using the protocol select command.

Table 24. CR95HF_FieldOff function description

Prototype	`u8 CR95HF_FieldOff(CR95HF_Protocol_Config* Protocol_Settings);`
Input parameter	**Protocol_Settings**: structure to update
Output parameter	None
Return parameter	**CR95HF_SUCCESS_CODE**: The field was successfully turned off **CR95HF_ERROR_INVALID_PROTOCOL**: The selected protocol is wrong. **CR95HF_ERROR_INVALID_CMD_LENGTH**: The command length is invalid. **CR95HF_ERROR_CODE**: The command failed

4.3.3 Low power mode functions

These functions send an idle command with some parameters forced into the function.

CR95HF_Hibernate function

This function sends an Hibernate command to CR95HF and turns the hibernate state on. This is the lowest consumption state of CR95HF. Only a pulse on $\overline{\text{IRQ_IN}}$ pin can wake-up the CR95HF.

Table 25. CR95HF_Hibernate function description

Prototype	`u8 CR95HF_Hibernate (CR95HF_Idle_Config* Idle_Config);`
Input parameter	None
Output parameter	**Idle_Config**: pointer on the structure which contains data about some idle parameters to update.
Return parameter	**CR95HF_SUCCESS_CODE**: The command is successful.

CR95HF_Sleep function

This function sends a Sleep command to CR95HF and turns the sleep state on.

Table 26. CR95HF_Sleep function description

Prototype	`u8 CR95HF_Sleep (CR95HF_Idle_Config* Idle_Config);`
Input parameter	**WakeUpSource**: the source of wake-up for CR95HF(1)
Output parameter	**Idle_Config**: pointer on the structure which contains data about some idle parameters to update
Return parameter	**CR95HF_SUCCESS_CODE**: The command is successful.

1. e.g.: for a wake-up by a pulse on SPI_NSS pad, use CR95HF_IDLE_WAKEUP_SPIN_SS value. For a wake- up by a pulse on IRQ_IN pad, use CR95HF_IDLE_WAKEUP_IRQ_in value.

CR95HF_TagDetecting function

This function sends a Tag detecting command to CR95HF. The CR95HF turns in a low power mode and periodically checks if a contactless tag is in the operating value. In this last case, the CR95HF will wake-up by itself.

Table 27. CR95HF_TagDetecting function description

Prototype	`u8 CR95HF_TagDetecting (const u8 WakeUpSource, const u8 WUperiod,const u8 DACdataL, const u8 DACdataH, CR95HF_Idle_Config* Idle_Config);`
Input parameter	**WakeUpSource**: other source of wake-up for CR95HF than tag detecting (1) **LFOfreq**: LFO frequency (2bits) (00 : 32 kHz 01 : 16 kHz 10 : 8 kHz 11 : 4 kHz) **WUperiod**: $t_{inactive}$ = time between two bursts = (Wuperiod +1)* 256 / LFOfreq **DACdataL**: low threshold of tag detecting **DACdataH**: high threshold of tag detecting
Output parameter	**Idle_Config**: pointer on the structure which contains data about some idle parameters to update
Return parameter	**CR95HF_SUCCESS_CODE**: The command is successful.

1. e.g.: for a wake-up by a pulse on SPI_NSS pad, use CR95HF_IDLE_WAKEUP_SPINSS value. For a wake-up by a pulse on IRQ_IN pad, use CR95HF_IDLE_WAKEUP_IRQ_in value.

CR95HF_IdlebyTimer function

This function sends an Idle command to CR95HF. The wake-up source is its internal timer.

Table 28. CR95HF_IdlebyTimer function description

Prototype	`u8 CR95HF_IdlebyTimer (const u8 WakeUpSource,const u8 LFOfreq, const u8 WUperiod,const u8 MaxSleep, CR95HF_Idle_Config* Idle_Config);`
Input parameter	**WakeUpSource**: other source of wake-up for CR95HF than tag detecting (1) **LFOfreq**: LFO frequency (2bits) (00: 32 kHz, 01: 16 kHz, 10: 8 kHz, 11: 4 kHz) **WUperiod**: timeout = (Wuperiod +1)* MaxSleep * 256 / LFOfreq **MaxSleep**: timeout = (Wuperiod +1)* MaxSleep * 256 / LFOfreq
Output parameter	**Idle_Config**: pointer on the structure contains the data about some idle parameters to update
Return parameter	**CR95HF_SUCCESS_CODE**: The command is successful.

1. For example, for a wake-up by a pulse on SPI_NSS pad, use CR95HF_IDLE_WAKEUP_SPINSS value. For a wake-up by a pulse on IRQ_IN pad, use CR95HF_IDLE_WAKEUP_IRQ_in value.

4.3.4 Is functions

These functions check a parameter and return either TRUE_CODE or FALSE_CODE.

CR95HF_IsReaderResultOK function

This function checks if the CR95HF has answered a successful code.

Table 29. CR95HF_IsReaderResultOK function description

Prototype	`u8 CR95HF_IsReaderResultOK(CR95HF_ResponseReceiving* Response,const u8 Code_OK);`
Input parameter	CR95HF_ResponseReceiving* Response,
Output parameter	None
Return parameter	**TRUE_CODE**: CR95HF returned a successful code. **FALSE_CODE**: CR95HF did not return a successful code.

Is_IRQ_in_Interupt_Wake_up_Condition function

This function returns whether an interrupt on $\overline{\text{IRQ_IN}}$ pin is a wake-up condition or not.

Table 30. Is_IRQ_in_Interupt_Wake_up_Condition function description

Prototype	`u8 CR95HF_Is_IRQ_in_Interrupt_Wake_up_Condition(u8 WUFlags);`
Input parameter	**WUFlags**: the byte containing the expected data
Output parameter	None
Return parameter	**TRUE_CODE**: An interrupt on $\overline{\text{IRQ_IN}}$ pin is a wake-up condition. **FALSE_CODE**: An interrupt on $\overline{\text{IRQ_IN}}$ pin is not a wake-up condition.

Is_NSS_Interrupt_Wake_up_Condition function

This function returns whether an interrupt on NSS is a wake-up condition or not.

Table 31. Is_NSS_Interrupt_Wake_up_Condition function description

Prototype	`u8 CR95HF_Is_NSS_Interrupt_Wake_up_Condition(u8 WUFlags);`
Input parameter	**WUFlags**: the byte containing the expected data
Output parameter	None
Return parameter	**TRUE_CODE**: An interrupt on $\overline{\text{IRQ_IN}}$ is a wake-up condition. **FALSE_CODE**: An interrupt on $\overline{\text{IRQ_IN}}$ is not a wake-up condition.

Is_Tag_Detected_Wake_up_Condition function

This function returns whether a tag detection is a wake-up condition or not.

Table 32. Is_Tag_Detected_Wake_up_Condition function description

Prototype	`u8 CR95HF_Is_Tag_Detected_Wake_up_Condition_(u8 WUFlags);`
Input parameter	**WUFlags**: the byte containing the expected data
Output parameter	None
Return parameter	**TRUE_CODE**: A tag detection is a wake-up condition. **FALSE_CODE**: A tag detection is not a wake-up condition.

Is_Timeout_Wake_up_Condition function

This function returns whether the internal timer is a wake-up condition.

Table 33. Is_Tag_Detected_Wake_up_Condition function description

Prototype	`u8 CR95HF_Is_Timeout_Wake_up_Condition(u8 WUFlags);`
Input parameter	**WUFlags**: the byte containing the expected data
Output parameter	None
Return parameter	**TRUE_CODE**: An interrupt on $\overline{\text{IRQ_IN}}$ is a wake-up condition. **FALSE_CODE**: An interrupt on $\overline{\text{IRQ_IN}}$ is not a wake-up condition.

4.3.5 Advanced functions

CR95HF_Modify_Baud_Rate function

This function is a procedure to change baud rates in UART communication. The function calls the CR95HF_Baud_Rate function and changes the MCU internal baud rate. In case of a failure, the function tries to recover the communication by returning into the previous baud rate or into the default baud rate.

Table 34. CR95HF_Modify_Baud_Rate function description

Prototype	`u8 CR95HF_Modify_Baud_Rate(const u8 New_Baud_Rate);`
Input parameter	`u8 New_Baud_Rate`
Output parameter	None
Return parameter	**CR95HF_SUCCESS_CODE**: The command is successful. **CR95HF_BAUd_RATE_FAILED_PREVIOUS_VALUE_ON**: The command failed but the transmission with the previous baud rate is kept. **CR95HF_BAUD_RATE_FAILED_DEFAULT_VALUE_ON**: The command failed and did not manage to keep the previous baud rate. Default baud rate is on. **CR95HF_ERROR_CODE**: The command failed. UART communication may be lost.

CR95HF_Idle_Detector_Calibration function

The purpose of this function is to perform the detector calibration according to the AN3433 Application Note.

Table 35. CR95HF_Idle_Detector_Calibration function description

Prototype	`u8 CR95HF_Idle_Detector_Calibration (`
	`CR95HF_Protocol_Config* Protocol_Settings ,`
	`CR95HF_Idle_Config* Idle_Config);`

Input parameter	**Protocol_Settings**: pointer on the structure to update the protocol select. (Turn RF field off)
Output parameter	**Idle_Config**: pointer on the structure to write the high compare value and low compare value for tag detection and the last wake-up reason.
Return parameter	**CR95HF_SUCCESS_CODE**: The command is successful. **CR95HF_ERROR_CODE**: The command failed.

CR95HF_Wait_Wake_up_From_Idle function

The purpose of this function is to wait for CR95HF to exit the idle state and retrieve what caused the wake-up. The method proposed is to wait for the CR95HF answer and then to send several echo commands in order to resume communication and clear the CR95HF response FIFO stack.

Table 36. CR95HF_Wait_Wake_up_From_Idle function description

Prototype	`u8 CR95HF_Wait_Wake_up_from_Idle(CR95HF_ResponseReceiving* CR95HF_Response_Idle,CR95HF_Idle_Config* Idle_Config, u8 Number_of_DELAY_BEFORE_TIMEOUT_250MS);`
Input parameter	**Idle_Config**: pointer on the structure to update with the new last reason of wake-up **Number_of_DELAY_BEFORE_TIMEOUT_250MS**: number of periods of 250ms duration to wait before aborting the procedure
Output parameter	**Idle_Config**: pointer on the structure to write the High compare value and low compare value for tag detection and the last wake-up reason
Return parameter	**EXIT_COMMUNICATION_OK** (`CR95HF_SUCCESS_CODE`): communication is possible. **EXIT_COMMUNICATION_TIMEOUT**: communication is not possible.

This function waits for CR95HF to wake-up on its own (internal timer, tag detection).

After this, the CR95HF_Wake_up_CR95HF function may be called to wake-up the CR95HF with NNS or $\overline{\text{IRQ_IN}}$ interrupt.

CR95HF_Wake_up_CR95HF function

This function wakes up the CR95HF device from idle mode. The method proposed is to send several echo commands in order to resume communication and clear the CR95HF response FIFO stack.

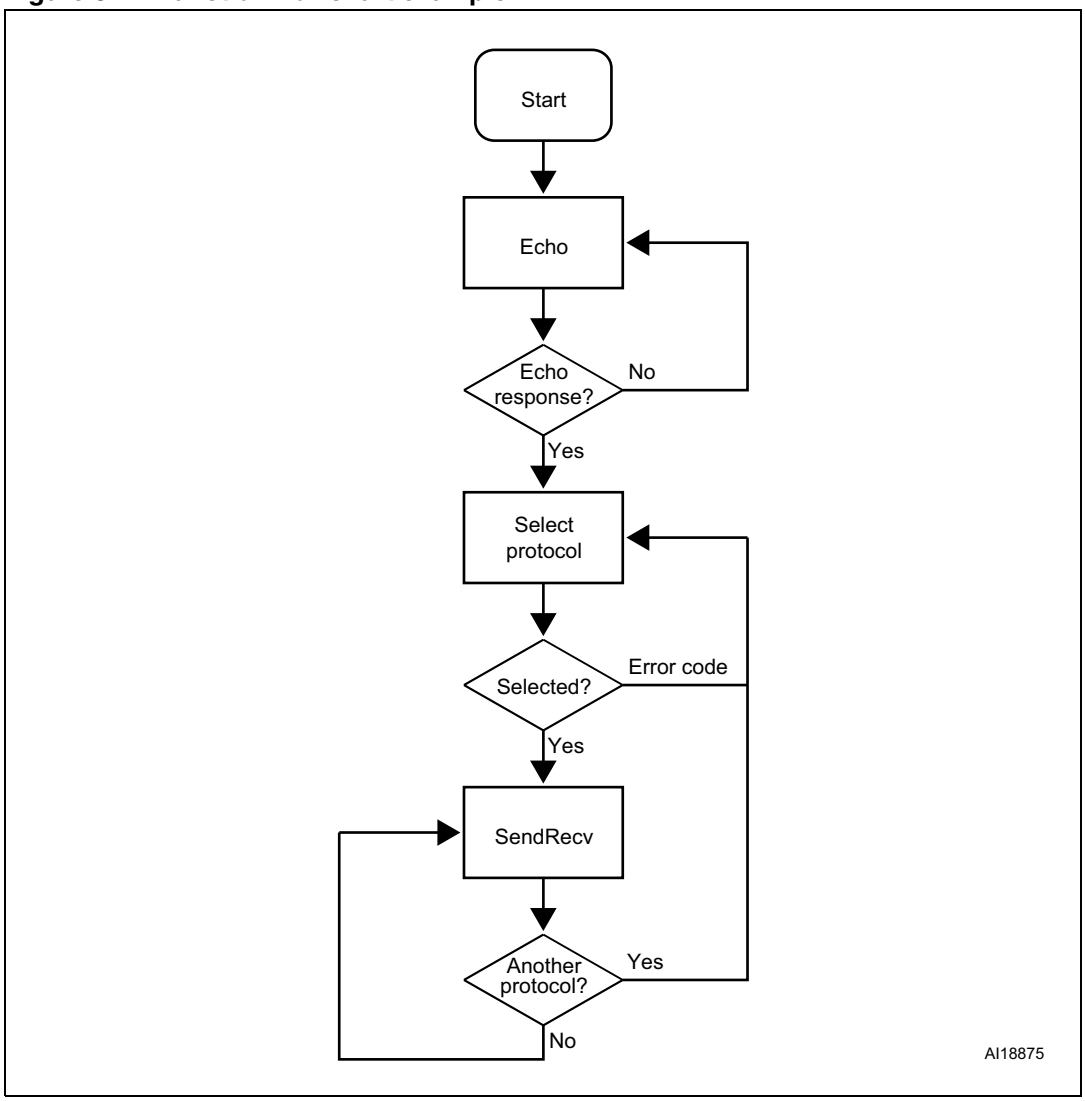
Table 37. CR95HF_Wake_up_CR95HF function description

Prototype	`u8 CR95HF_Wake_up_CR95HF(CR95HF_Idle_Config* Idle_Config);`
Input parameter	**Idle_Config**: contains the wake-up condition
Output parameter	None
Return parameter	**CR95HF_SUCCESS_CODE**: the CR95HF woke up, communication is possible. **CR95HF_ERROR_CODE**: fail to wake-up the CR95HF, either the CR95HF waits for another event (tag detection, timer) or communication cannot be resumed.

4.3.6 Application example: protocol selection and communication

To communicate with a contactless tag, the application must first select the RF protocol by sending a ProtocolSelect command. Then the application can use SendRecv commands to send data to a contactless tag. The user can select another protocol or change RF parameters (e.g. choose another data rate) by issuing again a ProtocolSelect command (refer to *Figure 3*).

Figure 3. Function flowchart example



5 ISO/IEC 15693 layer

This layer is composed of:

- the source file: *iso15693_command.c*
- the included file: *iso15693_command.h*

5.1 Overview

The ISO/IEC 15693 specification defines a set of commands to handle one or more contactless tags, and some commands of a higher level such as read or write commands.

This layer supports all the commands defined in the ISO/IEC 15693 specification and integrates some functions to facilitate the use of the ISO/IEC 15693 commands.

5.2 Structures

This layer uses a structure to store the information of an ISO/IEC contactless tag.

5.2.1 Structures of command and response of an ISO/IEC 15693 tag

The ISO15693_tag structure contains the different fields associated to an ISO/IEC15693 contactless tag.

Table 38. ISO15693_tag structure description

Name	Number of bytes	Comment
command_ok	1	The command and the response are ok, the structure is ok.
error_code	1	Error code of a contactless tag or CRC error
response_flags	1	Response flag provided by the tag
dataField	ISO15693_DATA_RECEIVED_MAX_SIZE	Whole data transmitted by the tag and CR95HF control byte
receivedDataLength	1	Length of data transmitted by the Tag
UID_Tag	ISO15693_UID_SIZE	UID of the contactless tag
AFI_Tag	1	AFI of the contactless tag
DSFID_Tag	1	DSFID of the contactless tag
ICRef	1	ICRef of the Tag (only for those manufactured by STMicroelectronics).
memSize	ISO15693_MEMSIZE_MAX_WORD_BYTE_SIZE	Memfield of getsystem info command

This structure is defined in *ISO15693_structure.h*.

```
typedef struct
{
u8 command_ok;
u8 error_code;
u8 response_flags;
u8 dataField[ISO15693_DATA_RECEIVED_MAX_SIZE];
u8 receivedDataLength;
u8 UID_Tag[ISO15693_UID_SIZE];
u8 AFI_Tag;
u8 DSFID_Tag;
u8 ICRef;
u8 memSize[ISO15693_MEMSIZE_MAX_WORD_BYTE_SIZE];
}ISO15693_Tag;
```

5.3 ISO/IEC 15693 command format

The commands defined by ISO/IEC 15693 specification have the following structure:

SOF	Request flag	Command code	Data	CRC	EOF

With:

- SOF: start of frame
- Request flag: 1 byte
- Command code: 1 byte
- Data: 1 or more byte
- CRC: 2 bytes
- EOF: end of frame

5.3.1 SOF and EOF

The SOF and EOF are managed by CR95HF device.

5.3.2 Request flag management

The request flags byte is managed by the user application. Each bit or flag specifies the actions to be carried out by the contactless tag and whether the corresponding fields are present or not.

The request flags is a byte integrated on all ISO/IEC 15693 commands, which specifies the actions to be performed by the contactless tag.

The meaning of bit 1, 2 and 4 is the same for all ISO/IEC 15693 commands.

Bit 3 (Inventory_flag) of the request flag defines the contents of the 4 MSBs (bits 5 to 8). When bit 3 is reset (0), bits 5 to 8 contain the contactless tag selection criteria. When bit 3 is set (1), bits 5 to 8 define the contactless tag Inventory parameters.

Table 39. Request flag bits description

Request flags	b1	b2	b3	b4	b5	b6	b7	b8
Inventory command	Subcarrier	Data rate	Inventory = 1	Protocol Extension	AFI	Address	Option	RFU
Other command			Inventory = 0		Select	Nb slots	Option	RFU

Table 40. Request flag bits values

Bit	Value	Description
Subcarrier	0	A single subcarrier frequency is used.
	1	Two subcarriers are used.
Data rate	0	Low data rate is used.
	1	High data rate is used.
Inventory	0	Other command.
	1	Inventory command.
Protocol Extension	0	RFU

Table 41 describes the bit for Request flag if the inventory flag is reset (other command than inventory).

Table 41. Bit for request flag with inventory flag reset

Bit	Value	Description
Select	0	Request is executed in other state than Selected.
	1	Request is executed only by the contactless tag in Selected state.
Data rate	0	Request is not addressed. UID field is not present.
	1	Request is addressed. UID field is present. The request is executed only by the contactless tag whose UID matches the UID specified in the request.
Option	-	Depends on the command.
RFU	0	RFU

Table 42 describes the bit for Request flag if the inventory flag is set (inventory commands)

Table 42. Bit for request flag with inventory flag reset

Bit	Value	Description
AFI	0	AFI field is not present.
	1	AFI field is present.
Data rate	0	16 slots
	1	1 slot
Option	0	RFU
RFU	0	RFU

For more information, please refer to ISO/IEC 15693 specification or ISO/IEC 15693 STMicroelectronics product datasheet.

Request flags and CR95HF_ProtocolSelect functions

The `CR95HF_ProtocolSelect` function (see *Chapter 4.2.3: Protocol selection structure*) selects the RF protocol and defines the CR95HF datarate and contactless tag response format (single or double subcarrier). Once the protocol and the RF parameters are configured, the CR95HF can decode only contactless tag responses of same format and datarate.

The datarate and contactless tag response format are also defined in the request flag byte of each ISO/IEC 15693 command. The user application must ensure that the data rate and subcarrier flags match the `CR95HF_ProtocolSelect` function parameters. This check is performed by the functions of the ISO/IEC 15693 layer. If the parameters do not match, the functions are not executed and the corresponding commands are not sent to the CR95HF.

5.3.3 Command code and Data management

The command and data shall be managed by user application.

5.3.4 CRC16 management

The ISO/IEC 15693 specification defines a two bytes CRC. It is appended to ISO/IEC 15693 command in order to check the data transmission between CR95HF and a contactless tag.

Bit 0 of parameter (AppendCRC) of ProtocolSelect command allows to append CRC to all RF commands. The `ISO15693_ComputeParameterByte_ProtocolSelect` function computes the parameter byte for protocol select according to the parameter provided and sets the append CRC bit.

This bit 0 has to be set because the function in ISO/IEC 15693 and products layers will not manage the CRC command. And the contactless tag will not answer to RF command.

5.4 ISO/IEC 15693 layer functions

The tables below summarize the available functions in CR95HF layer.

The functions send, through CR95HF, the command of ISO/IEC 15693 specification (refer to *Table 44*).

Table 43. Functions to compute parameter byte

Function name	Brief description
ISO15693_ComputeParameterByte_ProtocolSelect	Computes the Parameter byte for ProtocolSelect function.
ComputeByte	Computes a byte. It can be used to create Request flags byte.

Table 44. ISO15693 library command based on ISO/IEC 15693 specification

Function name	Brief description
ISO15693_Inventory	Emits an inventory command.
ISO15693_Stay_Quiet	Emits a stay quiet command.
ISO15693_Read_Single_Block	Emits a read single block command.
ISO15693_Write_Single_Block	Emits a write single block command.
ISO15693_Lock_Block	Emits a lock single block command.
ISO15693_Read_Multiple_Block	Emits a read multiple block command.
ISO15693_Write_Multiple_Block	Emits a write multiple block command.
ISO15693_Select	Emits a select command.
ISO15693_Reset_to_Ready	Emits a reset to ready command.
ISO15693_Write_AFI	Emits a write AFI command.
ISO15693_Lock_AFI	Emits a lock AFI command.
ISO15693_Write_DSFID	Emits a write DSFID command.
ISO15693_Lock_DSFID	Emits a lock DSFID command.
ISO15693_Get_System_Info	Emits a get system info command.
ISO15693_Get_Multiple_Blocks_Security_Status	Emits a Get multiple block security status command.
ISO15693_SendEOF	Emits an EOF pulse.
ISO15693_Custom_Commands	Emits a customs command.

Table 45. Functions to assemble ISO/IEC 15693 command

Function name	Brief description
ISO15693_Add_UID_to_command	Adds UID into the command's data.
ISO15693_Add_Mask_to_command	Adds mask field to command.
ISO15693_Add_Request_Flags_Command_code_to_command	Adds Request flags to command.
ISO15693_Is_ResponseFlags_noError_Detected	Checks if the response flag is set or not.
ISO15693_Is_Request_Flags_ok	Checks the request flag is correct.
ISO15693_Is_Collision_Flag_Set	Checks if the check collision flag added by the CR95HF is set.
ISO15693_IsCRC_Flag_Set	Checks if the check CRC16 flag added by the CR95HF is set.
ISO15693_IsCorrectCRC16_Check	Checks if the check CRC16 flag added by the CR95HF is set.
ISO15693_Is_TagConstructor_STM	Checks if the tag has been made by STMicroelectronics.
ISO15693_Is_Memory_programming_function	Returns if the command performs a memory programming.
ISO15693_Is_DSFID_Present_Infoflag	Returns if the DSFID is present.
ISO15693_Is_AFI_Present_Infoflag	Returns if the AFI is present.
ISO15693_Is_MemSize_Present_Infoflag	Returns if the Memsize is present.
ISO15693_Is_ICRef_Present_Infoflag	Returns if the ICREF is present.

Table 46. Get functions

Function name	Brief description
ISO15693_Get_Subcarrier_Flag	Returns if the tag's response is made with one or two subcarriers.
ISO15693_Get_DataRate_Flag	Returns if the tag's response is made in high or low datarate.
ISO15693_Get_Inventory_Flag	Returns if the inventory flag is set or not.
ISO15693_Get_ProtocolExtension_Flag	Returns if the protocol extension flag is set or not.
ISO15693_Get_Select_Flag	Returns if the select flag is set or not.
ISO15693_Get_Address_Flag	Returns if the address flag is set or not.
ISO15693_Get_AFI_Flag	Returns if the AFI flag is set.
ISO15693_Get_Slots_Flag	Returns the number of slots flag.
ISO15693_Get_Option_Flag	Returns if the option flag is set or not.
ISO15693_Get_RFU_Flag	Returns if the RFU is set or not.
ISO15693_Get_NumberofByte_MemSize_using_ICREF	Returns the number of bytes used to describe the memory size of the tag.
ISO15693_Get_Size_of_Block_using_ICRef	Returns the number of bytes of a EEPROM block.
ISO15693_Get_Size_of_Block_using_MemSize	Returns the number of bytes of a EEPROM block.
ISO15693_Get_MemSize_kbits	Returns the memory size of the tag in kbits.
ISO15693_Get_MemSize_bits	Returns the memory size of the tag in bits.
ISO15693_Get_UID_from_EEPROM	Retrieves the UID stored into the EEPROM by the inventory 16 slots commands.

Table 47. CRC16 functions

Function name	Brief description
ISO15693_CRC16	Computes CRC16 according to ISO/IEC 15693 specification.
ISO15693_IsCorrectCRC16Residue	Checks CRC16 residue according to ISO/IEC 15693 specification.
ISO15693_CRC16_EEPROM	Computes the CRC16 of data stored into the internal EEPROM.
ISO15693_IsCorrectCRC16Residue_EEPROM	Checks CRC16 residue of data stored into the internal EEPROM.

Table 48. Structure Fill In functions

Function name	Brief description
`ISO156693_Check_Data_Received_Start_Process`	Checks if the CRC is correct and fills the structure with the data common for all tag responses.
`ISO15693_Retrieve_UID`	Fills the Tag structure with the UID.
`ISO15693_Retrieve_ICRef`	Fills the Tag structure with the ICRef using the UID.

5.4.1 Compute parameter byte functions

These functions compute a parameter field which will be used in other functions.

ISO15693_ComputeParameterByte_ProtocolSelect function

This function computes the parameter byte for ISO/IEC 15693 ProtocolSelect command, bit0 is set to 1 to append CRC (see *Chapter 5.3.4: CRC16 management*).

Table 49. ISO15693_ComputeParameterByte_ProtocolSelect function description

Prototype	`u8 CR95HF_ISO15693_ComputeParameterByte_ProtocolSelect (u8 Bit7_6,u8 Bit5_4,u8 Bit3,u8 Bit2,u8 Bit1);`
Input parameter	**Bit7_6**: RFU **Bit5_4**: DataRate **Bit3**: ensure 312µs delay or wait for SOF **Bit2**: modulation **Bit1**: single or dual carrier
Output parameter	None
Return parameter	Parameter byte for ProtocolSelect command

ComputeByte

The ComputeByte function can be used to create the request flag required for all ISO/IEC 15693 commands.

Table 50. ComputeByte function description

Prototype	`u8 ComputeByte(const u8 Bit7,const u8 Bit6,const u8 Bit5,const u8 Bit4,const u8 Bit3, const u8 Bit2, const u8 Bit1,const u8 Bit0)`
Input parameter	**Bit7**: the parameter to place at the MSB bit **Bit6**: the parameter of bit number 6 **Bit5**: the parameter of bit number 5 **Bit4**: the parameter of bit number 4 **Bit3**: the parameter of bit number 3 **Bit2**: the parameter of bit number 2 **Bit1**: the parameter of bit number 1 **Bit0**: the LSB bit
Output parameter	None
Return parameter	Request flag byte

For instance, the following function creates a request flags for an Inventory command.

```
Request_Flags = ComputeByte(
ISO15693_BIT3SET_BIT8_RFU,
ISO15693_BIT3SET_BIT7_OPTIONFLAG_RESET,
ISO15693_BIT3SET_BIT6_1_SLOT,
ISO15693_BIT3SET_BIT5_AFI_NOT_PRESENT,
ISO15693_BIT4_NO_PROTOCOL_EXTENSION,
ISO15693_BIT3_INVENTORY_FLAG_SET,
ISO15693_BIT2_LOW_DATARATE,
ISO15693_BIT1_SINGLE_SUBCARRIER
);
```

5.4.2 ISO/IEC 15693 command functions

The functions described in this chapter send an ISO/IEC 15693 command.

ISO15693_Inventory function

This command emits an inventory command to contactless tag.

Table 51. ISO15693_Inventory function description

Prototype	`u8 ISO15693_Inventory(ISO15693_Tag* MyTag, const u8 Request_flags, const u8 Tag_AFI, const u8 MaskLength,const u8* Mask, u8* Inventory_16_slots_Nb_UID_Retrieved);`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not (1). **Tag_AFI**: AFI field to select a contactless tag family (2)(optional) **MaskLength**: length of the mask for transmitting (3) **Mask**: the mask to transmit **Inventory_16_slots_Nb_UID_Retrieved**: pointer to retrieve the number of tags seen during the inventory 16 slots
Output parameter	**MyTag**: pointer on the structure
Return parameter	**SO15693_ERROR_CODE**: The command failed. Either there is no tag or the tag did not manage to perform the request. **ISO15693_ERROR_PARAMETERS_REQUEST_FLAGS**: There is an error within the request flags parameters. **ISO15693_COMMAND_SUCCESS_CODE**: Tag answered the command.

1. Inventory shall be set.
2. AFI field is added if AFI flag of request flags is set
3. For example, to transmit a part of the UID of one tag. Masklength represents the numbers of significant bits. The user should ensure that other bits are reset for padding.

Note: If the inventory flag is not set, the function returns an error code.

ISO15693_StayQuiet function

This function emits a StayQuiet command to Contactless tag.

Table 52. ISO15693_StayQuiet function description

Prototype	`u8 ISO15693_Stay_Quiet(const u8 Request_flags, const u8* Tag_UID)`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not (1). **Tag_UID**: pointer on the UID (optional)
Output parameter	None
Return parameter	**ISO15693_ERROR_CODE**: The command failed. **ISO15693_ERROR_PARAMETERS_REQUEST_FLAGS**: There is an error within the request flags parameters. **ISO15693_SUCCESS_CODE**: The contactless tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).

ISO15693_Read_Single_Block function

This function sends, through the CR95HF, a read single block command to the contactless tag in the field.

Table 53. ISO15693_Read_Single_Block function description

Prototype	`u8 ISO15693_Read_Single_Block(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID, const u8 Block_number)`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not (1). **Tag_UID**: pointer on the UID (optional) **block_number**: address of the block to read
Output parameter	**MyTag**: pointer on the structure that stores the data read from the tag
Return parameter	**ISO15693_COMMAND_ERROR_CODE**:The command failed. Either there is no tag or the tag did not manage to perform the request. **ISO15693_ERROR_REQUEST_FLAGS_CODE**: The inventory flag is set. **ISO15693_COMMAND_SUCCESS_CODE**: Tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).

ISO15693_Write_Single_Block function

This function sends, through the CR95HF, a write single block command to the contactless tag in the field.

Table 54. ISO15693_Write_Single_Block function description

Prototype	`u8 ISO15693_Write_Single_Block(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID, const u8 Block_number,const u8 Block_Length, const u8* Data)`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not (1). **Tag_UID**: pointer on the UID (optional) **block_number**: address of the block to read **Block_Length**: number of bytes in a block **Data**: pointer to data to write into contactless tag memory
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**ISO15693_ERROR_CODE**: The command failed. **ISO15693_ERROR_PARAMETERS_REQUEST_FLAGS**: There is an error within the request flags parameters. **ISO15693_SUCCESS_CODE**: The contactless tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).

Note: According to ISO/IEC 15693 specification, the WriteSingleBlock command supports the Option Flag.

If the Option_flag is not set, the contactless tag shall return its response when it has completed the write operation.

If it is set, the contactless tag waits for an EOF pulse from the CR95HF.

The CR95HF supports these two cases but it has to be configured before sending an Write command.

This is the meaning of the Flag#3 of parameter of ProtocolSelect command (0: Ensure a 312 µs delay OR 1: Wait for SOF).

Table 55 shows the relationship between option flag of request_flag and:

- 312 µs delay; or,
- wait for SOF

Table 55. ISO15693_Write_Single_Block results depending on Option flag

Option flag of RequestFlag (312 µs or WaitSOF)	**0 (set)**	**1 (reset)**
0 (312 µs)	Ko (write worked out but CR95HF does not detect the response tag)	Ok (but required to emit an EOF pulse)
1 (Wait SOF)	Ok	Ok (but required to emit an EOF pulse)

The `ISO15693_Write_Single_Block` function returns an error code if the Option Flag is reset and 312 µs and SOF flag is reset.

If the option flag is set, the EOF pulse is managed by the `ISO15693_Write_Single_Block` function.

ISO15693_Lock_Block function

This function sends, through the CR95HF, a lock block command to the contactless tag in the field or to an unique contactless tag designated by its UID.

Table 56. ISO15693_Lock_Block function description

Prototype	`u8 ISO15693_Lock_Block(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID, const u8 Block_number)`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not (1). **Tag_UID**: pointer on the UID (optional) **block_number**: address of the block to read
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**ISO15693_ERROR_CODE**: The command failed. **ISO15693_ERROR_PARAMETERS_REQUEST_FLAGS**: There is an error within the request flags parameters. **ISO15693_SUCCESS_CODE**: The contactless tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).

ISO15693_Read_Multiple_Block function

This function sends, through the CR95HF, a read multiple block command to the contactless tag in the field.

Table 57. ISO15693_Read_Multiple_Block function description

Prototype	`u8 ISO15693_Read_Multiple_Block(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID, const u8 First_Block_number, const u8 Number_of_Blocks)`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not (1). **Tag_UID**: pointer on the UID (optional) **First_Block_number**: address of the first block to read **Number_of_Blocks**: number of blocks to read
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**ISO15693_ERROR_CODE**: The command failed. **ISO15693_ERROR_PARAMETERS_REQUEST_FLAGS**: There is an error within the request flags parameters. **ISO15693_SUCCESS_CODE**: The contactless tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).

Note: The data received is stored into the internal EEPROM of the MCU.

ISO15693_Write_Multiple_Block function

This function sends, through the CR95HF, a write multiple block command to the contactless tag in the field or to a unique contactless tag designated by its UID.

Table 58. ISO15693_Write_Multiple_Block function description

Prototype	`u8 ISO15693_Write_Multiple_Block(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID, const u8 First_Block_number, const u8 Number_of_Blocks,const u8 Block_Length)`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not (1). **Tag_UID**: pointer on the UID (optional) **First_Block_number**: address of the first block to read **Number_of_Blocks**: number of blocks to read **Block_Length**: number of bytes in a block
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**ISO15693_ERROR_CODE**: The command failed. **ISO15693_ERROR_PARAMETERS_REQUEST_FLAGS**: There is an error within the request flags parameters. **ISO15693_SUCCESS_CODE**: The contactless tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).

Note: Data to send to the contactless tag is retrieved from the internal MCU EEPROM.

ISO15693_Select function

This function sends, through the CR95HF, a reset to ready state command to a contactless tag.

Table 59. ISO15693_Select function description

Prototype	`u8 ISO15693_Select(ISO15693_Tag* MyTag,const u8 Request_flags, const u8* Tag_UID)`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not (1). **Tag_UID**: pointer on the UID (optional)
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**ISO15693_ERROR_CODE**: The command failed. **ISO15693_ERROR_PARAMETERS_REQUEST_FLAGS**: There is an error within the request flags parameters. **ISO15693_SUCCESS_CODE**: The contactless tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode)

ISO15693_Reset_to_Ready function

This function emits a ResetToReady command to Contactless tag in the field.

Table 60. ISO15693_Reset_to_Ready function description

Prototype	`u8 ISO15693_Reset_To_Ready(ISO15693_Tag* MyTag,u8 Request_flags, u8* Tag_UID)`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not (1). **Tag_UID**: pointer on the UID (optional)
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**ISO15693_ERROR_CODE**: The command failed. **ISO15693_ERROR_PARAMETERS_REQUEST_FLAGS**: There is an error within the request flags parameters. **ISO15693_SUCCESS_CODE**: The contactless tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).

ISO15693_Write_AFI function

This function sends, through the CR95HF, a write AFI command to the contactless tag in the field.

Table 61. ISO15693_Write_AFI function description

Prototype	`u8 ISO15693_Write_AFI(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID, const u8 Tag_AFI)`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not (1). **Tag_UID**: pointer on the UID (optional) **Tag_AFI**: AFI of the contactless tag, refer to datasheet for further information.
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**ISO15693_ERROR_CODE**: The command failed. **ISO15693_ERROR_PARAMETERS_REQUEST_FLAGS**: There is an error within the request flags parameters. **ISO15693_SUCCESS_CODE**: The contactless tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).

ISO15693_ Lock_AFI function

This function sends, through the CR95HF, a Lock AFI command to the contactless tag in the field.

Table 62. ISO15693_ Lock_AFI function description

Prototype	`u8 ISO15693_Lock_AFI (ISO15693_Tag* MyTag, u8 Request_flags, u8* Uid)`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not (1). **Tag_UID**: pointer on the UID (optional)
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**ISO15693_ERROR_CODE**: The command failed. **ISO15693_ERROR_PARAMETERS_REQUEST_FLAGS**: There is an error within the request flags parameters. **ISO15693_SUCCESS_CODE**: The contactless tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).

ISO15693_ WriteDSFID function

This function sends, through the CR95HF, a write DSFID command to the contactless tag in the field.

Table 63. ISO15693_ WriteDSFID function description

Prototype	`u8 ISO15693_Write_DSFID(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID, const u8 Tag_DSFID)`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not (1). **Tag_UID**: pointer on the UID (optional) **Tag_DSFID**: DSFID to write into Contactless tag memory
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**ISO15693_ERROR_CODE**: The command failed. **ISO15693_ERROR_PARAMETERS_REQUEST_FLAGS**: There is an error within the request flags parameters. **ISO15693_SUCCESS_CODE**: The contactless tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).

ISO15693_Lock_DSFID function

This function sends, through the CR95HF, a Lock DSFID command to the contactless tag in the field.

Table 64. ISO15693_Lock_DSFID function description

Prototype	`u8 ISO15693_Lock_DSFID(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID)`
Input parameter	**Request_flags**: option flag to indicate the parameters to use for the flag response and if UID is present (addressed) **Tag_UID**: pointer on the UID (optional)
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**ISO15693_ERROR_CODE**: The command failed. **ISO15693_ERROR_PARAMETERS_REQUEST_FLAGS**: There is an error within the request flags parameters. **ISO15693_SUCCESS_CODE**: The contactless tag answered the command.

ISO15693_Get_System_Info function

This function sends, through the CR95HF, an get System Information command to the contactless tag in the field.

Table 65. ISO15693_Get_System_Info function description

Prototype	`u8 ISO15693_Get_System_Info(ISO15693_Tag* MyTag, const u8 Request_flags,const u8* Tag_UID)`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not (1). **Tag_UID**: pointer on the UID (optional)
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**ISO15693_ERROR_CODE**: The command failed. **ISO15693_ERROR_PARAMETERS_REQUEST_FLAGS**: There is an error within the request flags parameters. **ISO15693_SUCCESS_CODE**: The contactless tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).

ISO15693_Get_Multiple_Blocks_Security_Status function

This function sends, through the CR95HF, a get multiple blocks security status command to the contactless tag in the field.

Table 66. ISO15693_Get_Multiple_Blocks_Security_Status function description

Prototype	`u8 ISO15693_Get_Multiple_Blocks_Security_Status(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID, const u8 First_Block_number, const u8 Number_of_Blocks)`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not (1). **Tag_UID**: pointer on the UID (optional). **First_Block_number**: address of the first block to read. **Number_of_Blocks**: number of blocks to read after the first one.
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**ISO15693_ERROR_CODE**: The command failed. **ISO15693_ERROR_PARAMETERS_REQUEST_FLAGS**: There is an error within the request flags parameters. **ISO15693_SUCCESS_CODE**: The contactless tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).

ISO15693_Custom_Commands function

This function sends, through the CR95HF, a custom command to the contactless tag in the field or to an unique contactless tag designated by its UID. The user should add in parameters all the data needed. Command size is limited to `MAX_DATASENT_LENGTH`.

Table 67. ISO15693_Custom_Commands function description

Prototype	`u8 ISO15693_Custom_Commands(ISO15693_Tag* MyTag, const u8 Request_flags, const u8 Custom, u8 IC_Mfg_Code, const u8 Parameters_Length, const u8* Parameters)`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not (1). **Tag_UID**: pointer on the UID (optional) **Custom**: the custom command code **IC_Mfg_Code**: the IC manufacturer code designating the manufacturer of the contactless tag **Parameters_Length**: length of parameters **Parameters**: data to write
Output parameter	**MyTag**: pointer on the structure that contains the CR95HF response
Return parameter	**ISO15693_ERROR_CODE**: The command failed. **ISO15693_ERROR_PARAMETERS_REQUEST_FLAGS**: There is an error within the request flags parameters. **ISO15693_SUCCESS_CODE**: The contactless tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).

ISO15693_SendEOF function

This function sends through the CR95HF an EOF pulse. This is used for Write Lock and Inventory 16 slots commands to inform the contactless tag that it can answer now (Write and Lock commands) or that a new slot is beginning (inventory 16 slots).

Table 68. ISO15693_SendEOF function description

Prototype	`u8 ISO15693_SendEOF(CR95HF_ResponseReceiving* Response);`
Input parameter	None
Output parameter	**pResponse**: pointer to the CR95HF response
Return parameter	**ISO15693_ERROR_CODE**: The command failed. **ISO15693_ERROR_PARAMETERS_REQUEST_FLAGS**: There is an error within the request flags parameters. **ISO15693_SUCCESS_CODE**: The contactless tag answered the command.

5.4.3 Build up functions

ISO15693_Add_UID_to_command

This function adds the UID into the command's data and increments the counter of data if the command is addressed.

Table 69. ISO15693_Add_UID_to_command description

Prototype	`void ISO15693_Add_UID_to_command ( CR95HF_CommandSending* Cmd, u8* Cptr_ToIncrement, u8* Uid ,u8 Request_flags);`
Input parameter	**Uid**: pointer on the UID **Request_flags**: to know if the command is addressed
Output parameter	**Cmd**: pointer on the command to fill in **cptr_ToIncrement**: pointer on the counter
Return parameter	None

ISO15693_Add_Mask_to_command function

This function adds the mask into the command's data and increments the counter of data.

Table 70. ISO15693_Add_Mask_to_command function description

Prototype	`void ISO15693_Add_Mask_to_command(CR95HF_CommandSending* Cmd, u8* Cptr_ToIncrement,const u8 MaskLength, const u8* Mask);`
Input parameter	**Cmd**: pointer on the command to fill in **cptr_ToIncrement**: pointer on the counter **MaskLength**: length of the Mask in bits **Mask**: pointer on the mask to add
Output parameter	None
Return parameter	None

ISO15693_Add_Request_Flags_Command_code_to_command function

This function adds the Request flags into the command's data and increments the counter of data.

Table 71. ISO15693_Add_Request_Flags_Command_code_to_command function description

Prototype	`Void ISO15693_Add_Request_Flags_Command_code_to_command ( CR95HF_CommandSending* Cmd, u8* Cptr_ToIncrement, const u8 Request_flags, const u8 Command_Code);`
Input parameter	**Cmd**: pointer on the command to fill in **cptr_ToIncrement**: pointer on the counter **Request_flags**: request flag bytes to append
Output parameter	None
Return parameter	None

5.4.4 Is Functions

ISO15693_Is_ResponseFlags_noError_Detected function

This function checks if the response flag is set or not. If it is set, the contactless tag encountered an error while performing the command.

Table 72. ISO15693_Is_ResponseFlags_noError_Detected function description

Prototype	`u8 ISO15693_Is_ResponseFlags_noError_Detected(const u8 ISO15693_Response_flags);`
Input parameter	**ISO15693_Response_flags**: response flag provided by the tag
Output parameter	None
Return parameter	**TRUE_CODE**: The contactless tag handled the request. **FALSE_CODE**: The contactless tag answered but the frame is an error message.

ISO15693_Is_Request_Flags_ok function

This function performs several checks to verify that the request flag is not wrong. This is not a guarantee that the request flag is right (depends on the contactless tag and on some command particularities).

Table 73. ISO15693_Is_Request_Flags_ok function description

Prototype	`u8 ISO15693_Is_Request_Flags_ok(u8 Request_flags, u8 ISO15693_cmd_code);`
Input parameter	**ISO15693_Response_flags**: response flag provided by the tag **ISO15693_cmd_code**: command code emitted to the contactless tag
Output parameter	None
Return parameter	**FALSE_CODE**: An error has been found out within the request flags. **TRUE_CODE**: There is no error seen within the request flags.

ISO15693_Is_Collision_Flag_Set function

This function checks if the check collision flag added by the CR95HF is set or not. If it is, it means that the CRC95HF has detected a collision between two contactless tag answers.

Table 74. ISO15693_Is_Collision_Flag_Set function description

Prototype	`u8 ISO15693_Is_Collision_Flag_Set(CR95HF_ResponseReceiving* Response);`
Input parameter	**ISO15693_Response_flags**: response flag provided by the tag **ISO15693_cmd_code**: command code emitted to contactless tag
Output parameter	None
Return parameter	**TRUE_CODE**: The collision flag is set. **FALSE_CODE**: The collision flag is not set.

ISO15693_IsCRC_Flag_Set function

This function checks if the check CRC flag added by the CR95HF is set or not. If it is, it means that the CRC95HF has detected an error during the reception.

Table 75. ISO15693_IsCRC_Flag_Set function description

Prototype	`u8 ISO15693_IsCRC_Flag_Set(CR95HF_ResponseReceiving Response);`
Input parameter	**Response**: response returned by CR95HF containing the data
Output parameter	None
Return parameter	**TRUE_CODE**: The CRT flag is set. **FALSE_CODE**: The CRT flag is not set.

ISO15693_Is_CorrectCRC16_Check function

This function checks the check CRC flag and the residue of the received command. If there is any problem (mismatch or error), the function returns an error value.

Table 76. ISO15693_Is_CorrectCRC16_Check function description

Prototype	`u8 ISO15693_Is_CorrectCRC16_Check(CR95HF_ResponseReceiving* Response);`
Input parameter	**Response**: response returned by CR95HF containing the data.
Output parameter	None
Return parameter	**TRUE_CODE**: The CRC residue is correct. **FALSE_CODE**: There is mismatch or error in CRC residue.

ISO15693_Is_TagConstructor_STM function

This function returns `TRUE_CODE` if the tag has been made by STMicroelectronics; the UID of the tag must be known.

Table 77. ISO15693_Is_TagConstructor_STM function description

Prototype	`u8 ISO15693_Is_TagConstructor_STM ( ISO15693_Tag* MyTag);`
Input parameter	**MyTag**: pointer on the structure containing the UID.
Output parameter	None
Return parameter	**TRUE_CODE**: STMicroelectronics is the manufacturer. **FALSE_CODE**: STMicroelectronics is not the manufacturer.

ISO15693_Is_Memory_programming function

This function returns if the contactless tag performs a memory programming by processing the command (write, lock).

Table 78. ISO15693_Is_Memory_programming function description

Prototype	`u8 ISO15693_Is_Memory_programming_function (const u8 ISO15693_cmd_code);`
Input parameter	**ISO15693_cmd_code**: command code
Output parameter	None
Return parameter	**TRUE_CODE**: The command is a memory programming command. **FALSE_CODE**: The command does not perform a memory programming.

ISO15693_Is_DSFID_Present_Infoflag function

This function returns if the DSFID field is present within the contactless tag answer to a get system information command.

Table 79. ISO15693_Is_DSFID_Present_Infoflag function description

Prototype	`u8 ISO15693_Is_DSFID_Present_Infoflag(const u8 Information_flags);`
Input parameter	**Information_Flags**: first byte transmitted by the contactless tag containing the information
Output parameter	None
Return parameter	**TRUE_CODE**: DSFID is supported. **FALSE_CODE**: DSFID is not supported.

ISO15693_Is_AFI_Present_Infoflag function

This function returns if the AFI field is present within the contactless tag answer to a get system info command.

Table 80. ISO15693_Is_AFI_Present_Infoflag function description

Prototype	`u8 ISO15693_Is_AFI_Present_Infoflag(const u8 Information_flags);`
Input parameter	**Information_Flags**: first byte transmitted by the contactless tag containing the information
Output parameter	None
Return parameter	**TRUE_CODE**: AFI is supported. **FALSE_CODE**: AFI is not supported.

ISO15693_Is_MemSize_Present_Infoflag function

This function returns if the Memsize is present within the contactless tag answer to a get system info command.

Table 81. ISO15693_Is_MemSize_Present_Infoflag function description

Prototype	`u8 ISO15693_Is_MemSize_Present_Infoflag(const u8 Information_flags);`
Input parameter	**Information_Flags**: first byte transmitted by the contactless tag containing the information
Output parameter	None
Return parameter	**TRUE_CODE**: Memsize is supported. **FALSE_CODE**: Memsize is not supported.

ISO15693_Is_ICRef_Present_Infoflag function

This function returns if the ICREF is present within the contactless tag answer to a get system info command.

Table 82. ISO15693_Is_ICRef_Present_Infoflag function description

Prototype	`u8 ISO15693_Is_ICRef_Present_Infoflag(const u8 Information_flags);`
Input parameter	**Information_Flags**: first byte transmitted by the contactless tag containing the information
Output parameter	None
Return parameter	**TRUE_CODE**: ICREF is supported. **FALSE_CODE**: ICREF is not supported.

5.4.5 Get functions

ISO15693_Get_Subcarrier_Flag function

This function returns the subcarrier flag determining if the contactless tag's response is made with one or two subcarriers.

Table 83. ISO15693_Get_Subcarrier_Flag function description

Prototype	`u8 ISO15693_Get_Subcarrier_Flag(const u8 Request_flags);`
Input parameter	**Request_flags**: information flags containing the data.
Output parameter	None
Return parameter	**ISO15693_SINGLE_SUBCARRIER**: The tag must answer with a single subcarrier. **ISO15693_TWO_SUBCARRIER**: The tag must answer with two subcarriers.

ISO15693_Get_DataRate_Flag function

This function returns the datarate flag determining if the contactless tag's response is made at a high or low datarate.

Table 84. ISO15693_Get_DataRate_Flag function description

Prototype	`u8 ISO15693_Get_DataRate_Flag(const u8 Request_flags);`
Input parameter	**Request_flags**: information flags containing the data
Output parameter	None
Return parameter	**ISO15693_LOW_DATARATE**: Low data rate shall be used. **ISO15693_HIGH_DATARATE**: High data rate shall be used.

ISO15693_Get_Inventory_Flag function

This function returns the inventory flag to determine if the contactless tag should run the anti-collision sequence (inventory command).

Table 85. ISO15693_Get_Inventory_Flag function description

Prototype	`u8 ISO15693_Get_Inventory_Flag(const u8 Request_flags);`
Input parameter	**Request_flags**: information flags containing the data
Output parameter	None
Return parameter	**ISO15693_SELECT_FLAG_RESET**: This request flag is not dedicated to the inventory command. **ISO15693_INVENTORY_FLAG_SET**: This request flag is dedicated to the inventory command.

ISO15693_Get_ProtocolExtension_Flag function

This function returns Protocol extension flag from Request Flags.

Table 86. ISO15693_Get_ProtocolExtension_Flag function description

Prototype	`u8 ISO15693_Get_ProtocolExtension_Flag(const u8 Request_flags);`
Input parameter	**Request_flags**: information flags containing the data
Output parameter	None
Return parameter	**ISO15693_NO_PROTOCOL_EXTENSION**: There is no protocol format extension. **ISO15693_PROTOCOL_EXTENSION**: Protocol format is extended.

ISO15693_Get_Select_Flag function

This function returns the select flag to determine if the command is designated to the contactless tag in the selected mode.

Table 87. ISO15693_Get_Select_Flag function description

Prototype	`u8 ISO15693_Get_Select_Flag(const u8 Request_flags);`
Input parameter	**Request_flags**: information flags containing the data
Output parameter	None
Return parameter	**ISO15693_SELECT_FLAG_SET**: The select flag is set. **ISO15693_SELECT_FLAG_RESET**: The select flag is reset. **ISO15693_ERROR_INVALID_FLAG**: Bit 3 is set so the parameter does not exist.

Note: a check is made to ensure that the inventory flag is reset.

ISO15693_Get_Address_Flag function

This function returns the address flag to determine if the command is addressed to the contactless tag designed by its UID.

Table 88. ISO15693_Get_Address_Flag function description

Prototype	`u8 ISO15693_Get_Address_Flag(const u8 Request_flags);`
Input parameter	**Request_flags**: information flags containing the data
Output parameter	None
Return parameter	**ISO15693_ADDRESS_FLAG_SET**: The address flag is set. **ISO15693_ADDRESS_FLAG_RESET**: The address flag is reset. **ISO15693_ERROR_INVALID_FLAG**: Bit 3 is set so the parameter does not exist.

Note: A check is made to ensure that the inventory flag is reset.

ISO15693_Get_AFI_Flag function

This function returns the AFI flag to determine if the AFI is present within the command (inventory).

Table 89. ISO15693_Get_AFI_Flag function description

Prototype	`u8 ISO15693_Get_AFI_Flag(const u8 Request_flags);`
Input parameter	**Request_flags**: first byte of RF command sent to the contactless tag
Output parameter	None
Return parameter	**ISO15693_AFI_PRESENT**: The AFI flag is set. **ISO15693_AFI_NOT_PRESENT**: The AFI flag is reset. **ISO15693_ERROR_INVALID_FLAG**: Bit 3 is reset so the parameter does not exist.

Note: A check is made to ensure that the inventory flag is set.

ISO15693_Get_Slots_Flag function

This function returns the number of slots flag (1 or 16) that the contactless tag has to answer to an inventory command.

Table 90. ISO15693_Get_Slots_Flag function description

Prototype	`u8 ISO15693_Get_Slots_Flag(const u8 RequestFlags);`
Input parameter	**Request_flags**: information flags containing the data
Output parameter	None
Return parameter	**ISO15693_1_SLOT**: The contactless tag has one slot to answer. **ISO15693_16_SLOTS**: The contactless tag has 16 slots to answer. **ISO15693_ERROR_INVALID_FLAG**: Bit 3 is set so the parameter does not exist.

Note: A check is made to ensure that the inventory flag is set.

ISO15693_Get_Option_Flag function

This function returns the option flag. Note: this function assumes that the option flag has the same meaning whatever the inventory flag value.

Table 91. ISO15693_Get_Option_Flag function description

Prototype	`u8 ISO15693_Get_Option_Flag(const u8 Request_flags);`
Input parameter	**Request_flags**: information flags containing the data
Output parameter	None
Return parameter	**ISO15693_OPTION_FLAG_SET**: The option flag is set. **ISO15693_OPTION_FLAG_RESET**: The option flag is reset.

ISO15693_Get_RFU_Flag function

This function returns the RFU flag. The function may need to be tailored when RFU will be defined.

Table 92. ISO15693_Get_RFU_Flag function description

Prototype	`u8 ISO15693_Get_RFU_Flag(const u8 Request_flags);`
Input parameter	**Request_flags**: information flags containing the data
Output parameter	None
Return parameter	**ISO15693_BIT8_RFU_SET**: The RFU flag is set. **ISO15693_BIT8_RFU_RESET**: The RFU flag is reset.

ISO15693_Get_NumberofByte_MemSize_using_ICREF function

This function returns the number of bytes used to describe the memory size of the contactless tag (STMicroelectronics manufactured tag).

Table 93. ISO15693_Get_NumberofByte_MemSize_using_ICREF function description

Prototype	`u8 ISO15693_Get_NumberofByte_MemSize_using_ICREF(const u8 ICRef);`
Input parameter	**ICRef**: the ICref to compare with the one referenced in the miscellaneous.h file (those available at the date of creation of this file; as a matter of fact, an update should be necessary)
Output parameter	None
Return parameter	**Number_of_Byte**: the number of bytes composing the word **NO_ICREF_MATCHING**: No Icref matches, 0 is returned.

ISO15693_Get_Size_of_Block_using_ICRef function

This function returns the size of a block using the UID of the Tag (STMicroelectronics' manufactured contactless tag).

Table 94. ISO15693_Get_Size_of_Block_using_ICRef function description

Prototype	`u8 ISO15693_Get_Size_of_Block_using_ICRef(const u8 ICRef);`
Input parameter	**ICRef**: the ICref to compare with the one referenced in the miscellaneous.h file (those available at the date of creation of this file; as a matter of fact, an update should be necessary)
Output parameter	None
Return parameter	**Number_of_Byte**: the number of bytes composing a block **ISO15693_BLOCK_SIZE_ERROR**: No Icref matches, 0 is returned

ISO15693_Get_Size_of_Block_using_MemSize function

This functions returns the size of a block using the MemSize field of the tag structure. A get system info command should have been performed before calling this function.

Table 95. ISO15693_Get_Size_of_Block_using_MemSize function description

Prototype	`u8 ISO15693_Get_Size_of_Block_using_MemSize(const ISO15693_Tag* MyTag);`
Input parameter	**MyTag**: the structure containing the data describing the contactless tag
Output parameter	None
Return parameter	**Size_of_Block**: the number of bytes composing the block **ISO15693_BLOCK_SIZE_ERROR**: Memsize contains no valid value, 0 is returned.

ISO15693_Get_MemSize_kbits function

This function returns the memory size of the tag in kbits.

Table 96. ISO15693_Get_MemSize_kbits function

Prototype	`u8 ISO15693_Get_MemSize_kbits(u8* MemSize, u8 ICRef);`
Input parameter	**MemSize**: pointer on the bytes describing the memory size **ICRef**: ICRef to retrieve the number of bytes if necessary
Output parameter	None
Return parameter	**size**: the size of the memory in kbits

ISO15693_Get_MemSize_bits function

This function returns the memory size of the contactless tag in bits. This function should be called if the size of the memory is not an integer number of kbits.

Table 97. ISO15693_Get_MemSize_bits function description

Prototype	`u32 ISO15693_Get_MemSize_bits(u8* MemSize);`
Input parameter	**MemSize**: pointer on the bytes describing the memory size
Output parameter	None
Return parameter	**size**: the size of the memory in bits (max 1023 bits)

ISO15693_Get_UID_from_EEPROM function

This function retrieves the UID stored into the EEPROM by the inventory 16 slots commands.

Table 98. ISO15693_Get_UID_from_EEPROM function description

Prototype	`u8 ISO15693_Get_UID_from_EEPROM(u8* UID_Tag, u8 UID_number);`
Input parameter	**MemSize**: pointer on the bytes describing the memory size
Output parameter	None
Return parameter	**ISO15693_ERROR_CODE**: The UID read from EEPROM contains an error. **ISO15693_SUCCESS_CODE**

5.4.6 CRC16 functions

ISO15693_CRC16 function

This function computes the CRC16 as defined by CRC ISO/IEC 13239

Table 99. ISO15693_CRC16 function description

Prototype	`int16_t ISO15693_CRC16 (u8* DataIn,u8 NbByte);`
Input parameter	**DataIn**: input data **Length**: number of bytes of DataIn
Output parameter	None
Return parameter	**ResCrc**: CRC16 computed.

ISO15693_IsCorrectCRC16Residue function

This function computes the CRC16 residue as defined by CRC ISO/IEC 13239 and returns ISO15693_RESULTOKCRC16 if the residue is compliant the ISO specification.

Table 100. ISO15693_IsCorrectCRC16Residue function description

Prototype	`ISO15693_IsCorrectCRC16Residue (u8 *DataIn,u8 Length);`
Input parameter	**DataIn**: input data **Length**: number of bytes of DataIn
Output parameter	None
Return parameter	**ISO15693_RESULTOKCRC16**: CRC16 residue is correct. **ISO15693_ERRORCODE_GENERICCRC16**: CRC16 residue is false.

ISO15693_CRC16_EEPROM function

This function computes the CRC16 residue as defined by CRC ISO/IEC 13239 and returns ISO15693_RESULTOKCRC16 if the residue is compliant with the ISO specification.

Table 101. ISO15693_CRC16_EEPROM function description

Prototype	`int16_t ISO15693_CRC16_EEPROM (const u8 NbByte);`
Input parameter	**NbByte**: number of bytes to check
Output parameter	None
Return parameter	**ResCrc**: CRC16 computed.

ISO15693_IsCorrectCRC16Residue_EEPROM function

This function computes the CRC16 residue as defined by CRC ISO/IEC 13239, using data stored into the internal EEPROM of the MCU.

Table 102. ISO15693_IsCorrectCRC16Residue_EEPROM function description

Prototype	`u8 ISO15693_IsCorrectCRC16Residue_EEPROM (const u8 Length);`
Input parameter	**Length**: number of bytes of DataIn
Output parameter	None
Return parameter	**ISO15693_RESULTOKCRC16**: CRC16 residue is correct. **ISO15693_ERRORCODE_GENERICCRC16**: CRC16 residue is false.

5.4.7 Fill In functions

ISO156693_Check_Data_Received_Start_Process function

This function checks the data integrity (the data is error-free and filled in basic data common for all responses: copy the whole data, write response_flag and data_length into the structure).

Table 103. ISO156693_Check_Data_Received_Start_Process function description

Prototype	`u8 ISO156693_Check_Data_Received_Start_Process(ISO15693_Tag* MyTag,CR95HF_ResponseReceiving Response, u8* Cursor_in_response_data);`
Input parameter	**Response**: the response containing the data
Output parameter	None
Return parameter	**ISO15693_RESULTOKCRC16**: Data integrity is ok. **ISO15693_ERROR_CRC**: Data is corrupted, errors are set in the structure.

ISO15693_Retrieve_UID function

The function fills the Tag structure with the UID.

Table 104. ISO15693_Retrieve_UID function description

Prototype	`void ISO15693_Retrieve_UID(ISO15693_Tag* MyTag,CR95HF_ResponseReceiving Response, u8* Cursor_in_response_data);`
Input parameter	**Response**: the response containing the data
Output parameter	**MyTag**: pointer on the structure to complete **Cursor_in_response_data**: actual position of the cursor into data and to increment
Return parameter	None

ISO15693_Retrieve_ICRef function

The function fills the Tag structure with the ICRef using the UID (only available for tag manufactured by STMicroelectronics).

Table 105. ISO15693_Retrieve_ICRef function

Prototype	`void ISO15693_Retrieve_ICRef(ISO15693_Tag* MyTag);`
Input parameter	None
Output parameter	**MyTag**: pointer on the structure to complete
Return parameter	None

6 LRIxK layer

6.1 Overview

LRIxK devices are compliant with ISO/IEC 15693 specifications. Several functions of the LRIxK-R layer are identical to ISO/IEC 15693 functions.

This layer includes all the commands defined in the LRIxK datasheets.

The LRIX layer applied to LRI1k, LRIS2k and LRI2k and is composed of:

- LRIxK_command.c
- LRIxK_command.h

6.2 Command format

6.2.1 CRC16 management

The LRIxK datasheet defines a two bytes CRC. It is appended to the RF command in order to check the data transmission between CR95HF and a contactless tag.

Bit 0 of parameter (AppendCRC) of ProtocolSelect command is set in the ProtocolSelect function. Thus the CR95HF manages the CRC of the RF command.

6.2.2 Request flag management

The request flag is detailed in *Chapter 5.3.2: Request flag management*.

The request flag byte is managed by the user application. Each bit or flag specifies the actions to be performed by the contactless tag and whether the corresponding fields are present or not. Bit 3 (Inventory_flag) of the request flag defines the contents of the 4 MSBs (bits 5 to 8). When bit 3 is reset (0), bits 5 to 8 define the contactless tag selection criteria.

When bit 3 is set (1), bits 5 to 8 define the contactless tag Inventory parameters.

Some commands of LRIxK datasheet required specific request flags. Those Request flags are forced inside the LRIxK layer function.

Table 106 lists the request flags witch are forced inside the M24LRxx layer functions:

- 0 means the flag is reset.
- 1 means the flag is set.
- - means the user application manages the flag.

Table 106. Request flag management description

Command	Inventory Flag	Select Flag	Option Flag	Protocol Extension flag	Address flag	RFU flag
Inventory	1	-	0	0	-	-
Stay Quiet	0	0	0	0	1	-
Read Single block	0	-	-	0	-	-
Write Single Block	0	-	-	0	-	-
Lock Block	0	-	0	0	-	-
Read Multiple Blocks	0	-	-	0	-	-
Select	0	0	0	0	1	-
Reset to ready	0	-	0	0	-	-
Write AFI	0	-	-	0	-	-
Lock AFI	0	-	-	0	-	-
Write DSFID	0	-	-	0	-	-
Lock DSFID	0	-	-	0	-	-
Get system Info	0	-	0	0	-	-
Get multiple blocks status	0	-	0	0	-	-
Kill	0	-	-	0	-	-
Write Kill	0	-	-	0	-	-
Lock Kill	-	-	-	0	-	1
Fast Read Single Block	0	-	-	0	-	-
Fast Inventory Initiated	0	-	0	0	-	-
Fast Initiate	0	0	0	0	0	-
Fast Read multiple blocks	0	-	-	0	-	-
Inventory Initiated	1	-	0	0	-	-
Initiate	0	0	0	0	0	-

For more information, please refer to the ISO/IEC 15693 specification and to the LRIxK datasheets.

6.2.3 Request flags and CR95HF_ProtocolSelect functions

The CR95HF_ProtocolSelect function (defined into the CR95HF layer) selects the RF protocol and defines, for the CR95HF device, the datarate and contactless tag response format (single or double subcarrier). Once the protocol and the RF parameters are configured, CR95HF is able to decode only contactless tag responses with the same format and datarate.

The datarate and contactless tag response format are also defined in the request flag byte of each command. The user application must ensure that the datarate and subcarrier flags match the CR95HF_ProtocolSelect function parameters. This check is performed by the ISO/IEC 15693 layer functions. If the parameters do not match, the functions are not executed and the corresponding commands are not sent to the CR95HF.

6.3 LRIxK layer commands

LRIxK devices are compliant with ISO/IEC 15693 specification. Thus the LRIxK functions call the ISO/IEC 15693 functions. Furthermore the LRIxK functions carry out some additional check and modifies some parameter as defined in the LRIxK datasheet.

The table below shows the relationship between LRIxK and ISO/IEC 15693 layers. The "function name" column presents the LRIxK functions.

The next column gives the functions called in the ISO15593 layer. If the function does not exist in the ISO/IEC 15693 layer, the cell is empty.

Table 107. LRIxK layer commands description

Function name	New or equivalent function	Brief description
LRIxK_Inventory	ISO15693_Inventory	
LRIxK_Stay_Quiet	ISO15693_Stay_Quiet	
LRIxK_Read_Single_Block	ISO15693_Read_Single_Block	
LRIxK_Write_Single_Block	ISO15693_Write_Single_Block	
LRIxK_Lock_Block	ISO15693_Lock_Block	
LRIxK_Read_Multiple_Block	ISO15693_Read_Multiple_Block	
LRIxK_Select	ISO15693_Select	
LRIxK_Reset_to_Ready	ISO15693_Reset_to_Ready	
LRIxK_Write_AFI	ISO15693_Write_AFI	
LRIxK_Lock_AFI	ISO15693_Lock_AFI	
LRIxK_Write_DSFID	ISO15693_Write_DSFID	
LRIxK_Lock_DSFID	ISO15693_Lock_DSFID	
LRIxK_Get_System_Info	ISO15693_Get_System_Info	
LRIxK_Get_Multiple_Block_ Security_Status	ISO15693_Get_Multiple_Block_ Security_Status	
LRIxK_Kill		Sends a kill command.
LRIxK_Write_Kill		Sends a Write Kill password.
LRIxK_Lock_Kill		Sends a lock Kill password.
LRIxK_Inventory_Initiated		Sends an Inventory Initiated command.
LRIxK_Initiate		Sends an Initiate command.
LRIxK_Fast_Read_Single_Block		Sends an Fast Read Single Block command.

6.3.1 LRIxK command functions

This chapter describes the specific LRIxK layer functions.

LRIxK_Kill function

This function sends, through the CR95HF, a kill command to the unique contactless tag designated by its UID and belonging to the LRIxK family. On receiving the kill command, the contactless tag compares the kill code added to the command and checks if it matches with the kill code previously written into the code. The contactless tag answers to the command before being deactivated.

Table 108. LRIxK_Kill function description

Prototype	`u8 LRIXK_Kill(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID, const u8* Kill_Code);`
Input parameter	**Request_flags**: option flag to indicate the parameters to use for the flag response and if UID is present (addressed). **Tag_UID**: pointer on the UID of the tag. **Kill_Code**: kill code to compare with the one inside the contactless tag in order to perform the kill command.
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**LRIxK_ERROR_CODE**: The command failed. Either there is no tag or the tag did not manage to perform the request. **LRIxK_ERROR_REQUEST_FLAGS_CODE**: The inventory flag is set. **LRIxK_COMMAND_SUCCESS_CODE**: The tag answered the command.

LRIxK_Write_Kill function

This function sends, through the CR95HF, a write kill command to an LRIxK contactless tag present in the RF field or designated by its UID. The kill code provided is transmitted to the contactless tag and will be required to deactivate the LRIxK product.

Table 109. LRIxK_Write_Kill function description

Prototype	`u8 LRIxK_Write_Kill(ISO15693_Tag* MyTag, u8 Request_flags, u8* Tag_UID, u8* Kill_Code);`
Input parameter	**Request_flags**: option flag to indicate the parameters to use for the flag response and if UID is present (addressed). **Tag_UID**: pointer on the UID of the tag. **Kill_Code**: kill code to write into the contactless tag memory.
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**LRIxK_ERROR_CODE**: The command failed. Either there is no tag or the tag did not manage to perform the request. **LRIxK_ERROR_REQUEST_FLAGS_CODE**: The inventory flag is set. **LRIxK_COMMAND_SUCCESS_CODE**: The tag answered the command.

Note: A lock kill command is necessary after this command to protect the kill code.

LRIxK_Lock_Kill function

This function sends, through the CR95HF, a lock kill command to an LRIxK contactless tag present in the RF field or designated by its UID.

Table 110. LRIxK_Lock_Kill function description

Prototype	`u8 LRIxK_Lock_Kill(ISO15693_Tag* MyTag, u8 Request_flags, u8* Tag_UID);`
Input parameter	**Request_flags**: option flag to indicate the parameters to use for the flag response and if UID is present (addressed) **Tag_UID**: pointer on the UID of the tag
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**LRIxK_ERROR_CODE**: The command failed. Either there is no tag or the tag did not manage to perform the request. **LRIxK_ERROR_REQUEST_FLAGS_CODE**: The inventory flag is set. **LRIxK_COMMAND_SUCCESS_CODE**: The tag answered the command.

Note: The kill code written previously is locked by the LRIxK contactless tag and, once done, the kill code cannot be changed.

LRIxK_Inventory_Initiated function

This function sends, through the CR95HF, an initiate command to an LRIxK contactless tag present in the field or designated by its UID. On receiving the initiate command, the LRIxK contactless tag sets the internal initiate flag (to send an inventory initiated command) and returns its UID and its DSFID.

Table 111. LRIxK_Inventory_Initiated function description

Prototype	`u8 LRIxK_Inventory_Initiated(ISO15693_Tag* MyTag, const u8 Request_flags, const u8 Tag_AFI, const u8 MaskLength, const u8* Mask, u8* Inventory_16_slots_Nb_UID_Retrieved);`
Input parameter	**Request_flags**: option flag to indicate the parameters to use for the flag response (1) **Tag_AFI**: the optional AFI to select a tag family **MaskLength**: length of the mask for transmitting(2) **Mask**: the mask to transmit **Inventory_16_slots_Nb_UID_Retrieved**: pointer to retrieve the number of tags seen during the inventory 16 slots
Output parameter	**MyTag**: pointer on the structure
Return parameter	**LRIxK_ERROR_CODE**: The command failed. Either there is no tag or the tag did not manage to perform the request. **LRIxK_ERROR_REQUEST_FLAGS_CODE**: The inventory flag is not set. **LRIxK_COMMAND_SUCCESS_CODE**: The tag answered the command.

1. Masklength represents the number of significant bits. The user should ensure that other bits are reset for padding.
2. The inventory flag is reset.

LRIxK_Initiate function

This function sends, through the CR95HF, an initiate command to an LRIxK contactless tag present in the RF field or designated by its UID. On receiving the initiate command, the LRIxK contactless tag sets the internal initiate flag (to send an inventory initiated command) and returns its UID and its DSFID.

Table 112. LRIxK_Initiate function description

Prototype	`u8 LRIXK_Initiate(ISO15693_Tag* MyTag, const u8 Request_flags);`
Input parameter	**Request_flags**: option flag to indicate the parameters to use for the flag response (1)
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**LRIxK_ERROR_CODE**: The command failed. Either there is no tag or the tag did not manage to perform the request. **LRIxK_ERROR_REQUEST_FLAGS_CODE**: The inventory flag is not set. **LRIxK_COMMAND_SUCCESS_CODE**: The tag answered the command.

1. Masklength represents the numbers of significant bits. The user should ensure that others bits are reset for padding.

Note: The command should not be in addressed or selected mode.

ISO15693_Fast_Read_Single_Block function

This function sends, through the CR95HF, a read single block command to the LRIxK contactless tag in the field. The particularity of this command is the data rate used by the contactless tag to answer: 53kbits/s.

Table 113. ISO15693_Fast_Read_Single_Block function description

Prototype	`u8 LRIxK_Fast_Read_Single_Block(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID, const u8 Block_number);`
Input parameter	**Request_flags**: option flag to indicate the parameters to use for the flag response and if UID is present (addressed) **Tag_UID**: pointer on the UID of the tag **Block_number**: address of the block to read
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	None

7 M24LRxx layer

The M24LRxx layer is composed of:

- M24LRxx_command.c
- M24LRxx_command.h

7.1 Overview

The library includes all the commands defined in the M24LRxx datasheets.

Since M24LRxx-R devices are compliant with ISO/IEC 15693 specifications, several M24LRxx layer functions are identical to ISO/IEC 15693 functions.

7.2 Command format

7.2.1 CRC16 management

The M24LRxx datasheet defines a two bytes CRC. It is appended to the RF command in order to check the data transmission between CR95HF and a contactless tag.

Bits 0 of parameter (AppendCRC) of ProtocolSelect command allow to append CRC to all RF commands.

The `ISO15693_ComputeParameterByte_ProtocolSelect` function computes the parameter byte for protocol select according to the parameter provided and sets the append CRC bit.

Bit 0 has to be set because the function in ISO/IEC 15693 and product layers will not manage the CRC command. And the contactless tag will not answer to RF command.

7.2.2 Request flag management

The request flag byte is detailed in *Chapter 5.3.2: Request flag management*.

Some commands of M24LRxx datasheet require specific request flags.

Table 114 lists the request flags witch are forced inside the M24LRxx layer functions:

- 0 means the flag is reset.
- 1 means the flag is set.
- - means the user application manages the flag.

Table 114. M24LR* layer forced request flags

Function	Inventory Flag	Protocol Extension flag	Option Flag	Address flag	Select flag
Inventory	1	0	0	-	-
Stay Quiet	0	0	0	1	0
Read Single block	0	-	-	-	-
Write Single Block	0	-	-	-	-
Read Multiple Blocks	0	-	-	-	-
Select	0	0	0	1	0
Reset to ready	0	0	0	-	-
Write AFI	0	0	-	-	-
Lock AFI	0	0	-	-	-
Write DSFID	0	0	-	-	-
Lock DSFID	0	0	-	-	-
Get system Info	0	1	0	-	-
Get multiple blocks status	0	1	0	-	-
Write Sector Password	0	0	-	-	-
Present Sector Password	0	0	-	-	-
Fast Read Single Block	0	-	-	-	-
Fast Inventory Initiated	1	0	0	-	-
Fast Initiate	0	0	0	0	0
Fast Read multiple blocks	0	-	-	-	-
Inventory Initiated	1	0	0	-	-
Initiate	0	0	0	0	0
ReadCfg	0	0	0	-	-
WriteEHCfg	0	0	-	-	-
SetRstEHEn	0	0	0	-	-
CheckEHEn	0	0	0	-	-
WriteDOCfg	0	0	-	-	-

For more information, please refer to ISO/IEC 15693 specification and to the LRIxK datasheets.

7.2.3 Request flags and CR95HF_ProtocolSelect functions

The CR95HF_ProtocolSelect function (defined into CR95HF layer) selects the RF protocol and defines, for a CR95HF device, the datarate and contactless tag response format (single or double subcarrier). Once the protocol and the RF parameters are configured, the CR95HF is able to decode only contactless tag responses with same format and datarate.

The datarate and contactless tag response format are also defined in the request flag byte of each command. The user application must ensure that the datarate and subcarrier flags match the CR95HF_ProtocolSelect function parameters. This check is performed by the ISO/IEC 15693 layer functions. If the parameters do not match, the functions are not executed and the corresponding commands are not sent to the CR95HF.

7.3 M24LRxx commands

M24LRxx devices are based on ISO/IEC 15693 specification. Thus the M24LRxx functions call the ISO/IEC 15693 functions. Furthermore the M24LRxx functions carry out some additional checks and modify some parameter as defined in the M24LRxx datasheet.

Table 115 shows the relationship between M24LRxx and ISO/IEC 15693 layer. The "function name" column presents the M24LRxx functions.

The next column gives the ISO15593 layer functions called. If the functions does not exist in the ISO/IEC 15693 layer, the "New function" idiom is display.

Table 115. Relationship between M24LRxx and ISO/IEC 15693 layer

Function name	New or equivalent function	Brief description
M24LRxx_Inventory	ISO15693_Inventory	
M24LRxx_Stay_Quiet	ISO15693_Stay_Quiet	
M24LRxx_Read_Single_Block	New function	Sends a ReadSingleBlock command
M24LRxx_Write_Single_Block	New function	Sends a WriteSingleBlock command
M24LRxx_Read_Multiple_Block	New function	Sends a ReadMultipleBlock command
M24LRxx_Select	ISO15693_Select	
M24LRxx_Reset_to_Ready	ISO15693_Reset_to_Ready	
M24LRxx_Write_AFI	ISO15693_Write_AFI	
M24LRxx_Lock_AFI	ISO15693_Lock_AFI	
M24LRxx_Write_DSFID	ISO15693_Write_DSFID	
M24LRxx_Lock_DSFID	ISO15693_Lock_DSFID	
M24LRxx_Get_System_Info	New functions	Sends a GetSystemInfo command

Table 115. Relationship between M24LRxx and ISO/IEC 15693 layer

Function name	New or equivalent function	Brief description
M24LRxx_Get_Multiple_Blocks_Security_Status	New functions	Sends a GetMultiple BlocksSecurityStatus command
M24LRxx_Write_Sector_Password	New functions	Sends A WriteSectorPassword command
M24LRxx_Lock_Sector_Password	New functions	Sends a LockSectorPassword command
M24LRxx_Present_Sector_Password	New functions	Sends a PresentSectorPassword command
M24LRxx_Fast_Read_Single_Block	New functions	Sends a FastReadSingleBlock command
M24LRxx_Fast_Inventory_Initiated	New function	Sends a FastInventoryInitiated
M24LRxx_Fast_Initiate	New function	Sends a FastInitiate command
M24LRxx_Fast_Read_Multiple_Block	New function	Sends a FastReadMultipleBlock command
M24LRxx_Inventory_Initiated	New function	Sends a InventoryInitiated command
M24LRxx_Initiate	New function	Sends a Initiate command
M24LRxx_ReadCfg	New function (1)	Sends a ReadCfg command
M24LRxx_WriteEHCfg		Sends a WriteEHCfg command
M24LRxx_SetRstEHEn		Sends a SetRstEHen command
M24LRxx_CheckEHEn		Sends a CheckEHEn command
M24LRxx_WriteDOCfg		Sends a WriteDOCfg command

1. Commands specific to M24LRxx-E device. These commands manage the energy harvesting feature.

7.3.1 M24LRxx command functions

This chapter describes the specific M24LRxx layer functions.

M24LRxx_Read_Single_Block function

This function sends, through the CR95HF, a read single block command to a contactless tag in the field.

Table 116. M24LRxx_Read_Single_Block function description

Prototype	`u8 M24LRxx_Read_Single_Block(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID, const u16 Block_number);`
Input parameter	**Request_flags**: specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) (2) **Tag_UID**: pointer on the UID (optional) **block_number**: address of the block to read (3)
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**ISO15693_COMMAND_ERROR_CODE**: The command failed. **ISO15693_ERROR_REQUEST_FLAGS_CODE**: The inventory flag is set. **ISO15693_COMMAND_SUCCESS_CODE**: The tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).
2. If the memory size is more than 8 kbit, the protocol extension flag of request flags should be set.
3. If the memory size is more than 8 kbit, the block number is composed by two bytes.

M24LRxx_Write_Single_Block function

This function sends, through the CR95HF, a write single block command to a contactless tag in the field.

Table 117. M24LRxx_Write_Single_Block function description

Prototype	`u8 M24LRxx_Write_Single_Block(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID, const u16 Block_number, const u8* Data);`
Input parameter	**Request_flags**: specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) (2) **Tag_UID**: pointer on the UID (optional) **block_number**: address of the block to read (3) **Data**: data to write into the contactless tag memory
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).
2. If the memory size is more than 8 kbit, the protocol extension flag of request flags should be set.
3. If the memory size is more than 8 kbit, the block number is composed by two bytes.

M24LRxx_Read_Multiple_Block function

This function sends, through the CR95HF, a read multiple block command to a contactless tag in the field.

Table 118. M24LRxx_Read_Multiple_Block function description

Prototype	`u8 M24LRxx_Read_Multiple_Block(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID, const u16 First_Block_number, const u8 Number_of_Blocks);`
Input parameter	**Request_flags**: specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) (2) **Tag_UID**: pointer on the UID (optional) **First_Block_number**: address of the first block to read (3) **Number_of_Blocks**: number of blocks to read after the first one
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).
2. If the memory size is more than 8 kbit, the protocol extension flag of request flags should be set.
3. If the memory size is more than 8 kbit, the block number is composed by two bytes.

Note: Received data is stored into the internal EEPROM of the MCU.

M24LRxx_Get_System_Info function

This function sends, through the CR95HF, a Get System Information command to a contactless tag in the field.

Table 119. M24LRxx_Get_System_Info function description

Prototype	`u8 M24LRxx_Get_System_Info(ISO15693_Tag* MyTag, const u8 Request_flags,const u8* Tag_UID);`
Input parameter	**Request_flags**: specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) (2) **Tag_UID**: pointer on the UID (optional)
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).
2. If the memory size is more than 8 kbit, the protocol extension flag of request flags should be set.

M24LRxx_Get_Multiple_Blocks_Security_Status function

This function sends, through the CR95HF, a get multiple blocks security status command to a contactless tag in the field.

Table 120. M24LRxx_Get_Multiple_Blocks_Security_Status function description

Prototype	`u8 M24LRxx_Get_Multiple_Blocks_Security_Status(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID, const u16 First_Block_number, const u8 Number_of_Blocks);`
Input parameter	**Request_flags**: specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) (2) **Tag_UID**: pointer on the UID (optional) **block_number**: address of the block to read (3) **Number_of_Blocks**: number of blocks to read after the first one
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).
2. If the memory size is more than 8 kbit, the protocol extension flag of request flags should be set.
3. If the memory size is more than 8 kbit, the block number is composed by two bytes.

Note: Received data is stored into the internal EEPROM of the MCU.

M24LRxx_Write_Sector_Password function

This function sends, through the CR95HF, a write sector password command to a contactless tag in the field.

Table 121. M24LRxx_Write_Sector_Password function description

Prototype	`u8 M24LRxx_Write_Sector_Password(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID,const u8 Password_Number ,const u8* Password);`
Input parameter	**Request_flags**: specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) **Tag_UID**: pointer on the UID (optional) **Password_Number**: number identifying the password (between 1 and 3) **Password**: password to write
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).

M24LRxx_Lock_Sector_Password function

This function sends, through the CR95HF, a lock sector password command to a contactless tag in the field.

Table 122. M24LRxx_Lock_Sector_Password function description

Prototype	`u8 M24LRxx_Lock_Sector_Password(ISO15693_Tag* MyTag, u8 Request_flags, const u8* Tag_UID, u16 Sector_Number, u8 Sector_Security_Status) ;`
Input parameter	**Request_flags**: specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) **Tag_UID**: pointer on the UID (optional) **Sector_Number**: one of the block addresses contained into the sector **Sector_Security_Status**: defines the read write protection and the number of the password protecting the sector.
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).

M24LRxx_Present_Sector_Password function

This function sends, through the CR95HF, a present sector password command to a contactless tag in the field.

Table 123. M24LRxx_Present_Sector_Password function description

Prototype	`u8 M24LRxx_Present_Sector_Password(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID,const u8 Password_Number ,const u8* Password);`
Input parameter	**Request_flags**: specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) **Tag_UID**: pointer on the UID (optional) **Password_Number**: number identifying the password (between 1 and 3) **Password**: password to compare
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).

M24LRxx_Fast_Read_Single_Block function

This function sends, through the CR95HF, a fast read single block command to a contactless tag in the field. The datarate of the response is multiplied by 2 (52 kbit/s in this case).

Table 124. M24LRxx_Fast_Read_Single_Block function description

Prototype	`u8 M24LRxx_Fast_Read_Single_Block(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID, const u16 Block_number);`
Input parameter	**Request_flags**: specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) (2) **Tag_UID**: pointer on the UID (optional) **Block_number**: address of the block to read (3)
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).
2. If the memory size is more than 8 kbit, the protocol extension flag of request flags should be set.
3. If the memory size is more than 8 kbit, the block number is composed by two bytes.

M24LRxx_Fast_Read_Multiple_Block function

This function sends, through the CR95HF, a read multiple block command to a contactless tag in the field. The datarate of the response is multiplied by 2 (52 kbit/s in this case).

Table 125. M24LRxx_Fast_Read_Multiple_Block function description

Prototype	`u8 M24LRxx_Fast_Read_Multiple_Block(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID, const u8 First_Block_number, const u8 Number_of_Blocks);`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) (2) **Tag_UID**: pointer on the UID (optional) **First_Block_number**: address of the first block to read (3) **Number_of_Blocks**: number of blocks to read after the first one
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).
2. If the memory size is more than 8 kbit, the protocol extension flag of request flags should be set.
3. If the memory size is more than 8 kbit, the block number is composed by two bytes.

Note: Received data is stored into the internal EEPROM of the MCU.

M24LRxx_Inventory_Initiated function

This function sends, through the CR95HF, a fast inventory command to a contactless tag in the field.

Table 126. M24LRxx_Inventory_Initiated function description

Prototype	`u8 M24LRxx_Inventory_Initiated(ISO15693_Tag* MyTag, const u8 Request_flags, const u8 Tag_AFI, const u8 MaskLength, const u8* Mask, u8* Inventory_16_slots_Nb_UID_Retrieved);`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. **Tag_AFI**: AFI field to select a contactless tag family (optional) (1) **MaskLength**: length of the mask for transmitting (2) **Mask**: the mask to transmit **Inventory_16_slots_Nb_UID_Retrieved**: pointer to retrieve the number of tags seen during the inventory 16 slots
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. AFI field is added if AFI flag of requests flags is set.
2. For example, to transmit a part of the UID of one tag. Masklength represents the number of significant bits. The user should ensure that other bits are reset for padding.

M24LRxx_Fast_Inventory_Initiated function

This function sends, through the CR95HF, a fast inventory command to a contactless tag in the field. The datarate of the response is multiplied by 2 (52 kbit/s in this case).

Table 127. M24LRxx_Fast_Inventory_Initiated function description

Prototype	`u8 M24LRxx_Fast_Inventory_Initiated(ISO15693_Tag* MyTag,const u8 Request_flags, const u8 Tag_AFI, const u8 MaskLength, const u8* Mask, u8* Inventory_16_slots_Nb_UID_Retrieved);`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. **Tag_AFI**: AFI field to select a contactless tag family (optional) (1) **MaskLength**: Length of the mask for transmitting. (2) **Mask**: The mask to transmit. **Inventory_16_slots_Nb_UID_Retrieved**: Pointer to retrieve the number of tags seen during the inventory 16 slots
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. AFI field is added if AFI flag of requests flags is set
2. For example to transmit a part of the UID of one tag. Masklength represents the number of significant bits. The user should ensure that other bits are reset for padding.

Note: Received data is stored into the internal EEPROM of the MCU.

M24LRxx_Get_System_Info function

This function sends, through the CR95HF, a Get System Information command to a contactless tag in the field.

Table 128. M24LRxx_Get_System_Info function description

Prototype	`u8 M24LRxx_Get_System_Info(ISO15693_Tag* MyTag, const u8 Request_flags,const u8* Tag_UID);`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) (2) **Tag_UID**: pointer on the UID (optional)
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).
2. If the memory size is larger than 8 kbit, the protocol extension flag of request flags should be set.

M24LRxx_Get_Multiple_Blocks_Security_Status function

This function sends, through the CR95HF, a get multiple blocks security status command to a contactless tag in the field.

Table 129. M24LRxx_Get_Multiple_Blocks_Security_Status function description

Prototype	`u8 M24LRxx_Get_Multiple_Blocks_Security_Status(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID, const u16 First_Block_number, const u8 Number_of_Blocks);`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) (2) **Tag_UID**: pointer on the UID (optional) **First_Block_number**: address of the first block to read (3) **Number_of_Blocks**: number of blocks to read after the first one
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).
2. If the memory size is more than 8 kbit, the protocol extension flag of request flags should be set.
3. If the memory size is more than 8 kbit, the block number is composed by two bytes.

Note: Received data is stored into the internal EEPROM of the MCU.

M24LRxx_Write_Sector_Password function

This function sends, through the CR95HF, a write sector password command to a contactless tag in the field.

Table 130. M24LRxx_Write_Sector_Password function description

Prototype	`u8 M24LRxx_Write_Sector_Password(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID,const u8 Password_Number ,const u8* Password);`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) **Tag_UID**: pointer on the UID (optional) **Password_Number**: number identifying the password (between 1 and 3) **Password**: password to write
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).

M24LRxx_Lock_Sector_Password function

This function sends, through the CR95HF, a lock sector password command to a contactless tag in the field.

Table 131. M24LRxx_Lock_Sector_Password function description

Prototype	`u8 M24LRxx_Lock_Sector_Password(ISO15693_Tag* MyTag, u8 Request_flags, const u8* Tag_UID, u16 Sector_Number, u8 Sector_Security_Status);`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) **Tag_UID**: pointer on the UID (optional) **Sector_Number**: one of the block addresses contained into the sector **Sector_Security_Status**: Defines the read write protection and the number of password protecting the sector.
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).

M24LRxx_Present_Sector_Password function

This function sends, through the CR95HF, a present sector password command to a contactless tag in the field.

Table 132. M24LRxx_Present_Sector_Password function description

Prototype	`u8 M24LRxx_Present_Sector_Password(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID,const u8 Password_Number ,const u8* Password);`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) **Tag_UID**: pointer on the UID (optional) **Password_Number**: number identifying the password (between 1 and 3) **Password**: password to compare
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: Tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).

M24LRxx_Fast_Read_Single_Block function

This function sends, through the CR95HF, a fast read single block command to a contactless tag in the field. The datarate of the response is multiplied by 2 (52 kbit/s in this case)

Table 133. M24LRxx_Fast_Read_Single_Block function description

Prototype	`u8 M24LRxx_Fast_Read_Single_Block(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID, const u16 Block_number);`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) (2) **Tag_UID**: pointer on the UID (optional) **Block_number**: address of the block to read (3)
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).
2. If the memory size is more than 8 kbit, the protocol extension flag of request flags should be set.
3. If the memory size is more than 8 kbit, the block number is on two bytes.

M24LRxx_Fast_Read_Multiple_Block function

This function sends, through the CR95HF, a fast read multiple block command to a contactless tag in the field. The datarate of the response is multiplied by 2 (52 kbit/s in this case).

Table 134. M24LRxx_Fast_Read_Multiple_Block function description

Prototype	`u8 M24LRxx_Fast_Read_Multiple_Block(ISO15693_Tag* MyTag, const u8 Request_flags, const u8* Tag_UID, const u8 First_Block_number, const u8 Number_of_Blocks);`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) (2) **Tag_UID**: pointer on the UID (optional) **First_Block_number**: address of the first block to read (3) **Number_of_Blocks**: number of blocks to read after the first one
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).
2. If the memory size is more than 8 kbit, the protocol extension flag of request flags should be set.
3. If the memory size is more than 8 kbit, the block number is on two bytes.

Received data is stored into the internal EEPROM of the MCU.

M24LRxx_Inventory_Initiated function

This function sends, through the CR95HF, a fast inventory command to a contactless tag in the field.

Table 135. M24LRxx_Inventory_Initiated function description

Prototype	`u8 M24LRxx_Inventory_Initiated(ISO15693_Tag* MyTag, const u8 Request_flags, const u8 Tag_AFI, const u8 MaskLength, const u8* Mask, u8* Inventory_16_slots_Nb_UID_Retrieved);`
Input parameter	**Request_flags**: specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. **Tag_AFI**: AFI field to select a contactless tag family (optional) (1) **MaskLength**: length of the mask for transmitting (2) **Mask**: the mask to transmit **Inventory_16_slots_Nb_UID_Retrieved**: pointer to retrieve the number of tags seen during the inventory 16 slots.
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. AFI field is added if AFI flag of requests flags is
2. For example, to transmit a part of the UID of one tag. Masklength represents the numbers of significant bits. The user should ensure that others bits are reset for padding.

M24LRxx_Fast_Inventory_Initiated function

This function sends, through the CR95HF, a fast inventory command to a contactless tag in the field. The datarate of the response is multiplied by 2 (52 kbit/s in this case).

Table 136. M24LRxx_Fast_Inventory_Initiated function description

Prototype	`u8 M24LRxx_Fast_Inventory_Initiated(ISO15693_Tag* MyTag,const u8 Request_flags, const u8 Tag_AFI, const u8 MaskLength, const u8* Mask, u8* Inventory_16_slots_Nb_UID_Retrieved);`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. **Tag_AFI**: AFI field to select a contactless tag family (optional) (1) **MaskLength**: length of the mask for transmitting (2) **Mask**: the mask to transmit **Inventory_16_slots_Nb_UID_Retrieved**: pointer to retrieve the number of tags seen during the inventory 16 slots.
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. AFI field is added if AFI flag of requests flags is
2. For example, to transmit a part of the UID of one tag. Masklength represents the numbers of significant bits. The user should ensure that others bits are reset for padding.

M24LRxx_Initiate function

This function sends, through the CR95HF, an initiate command to a contactless tag in the field.

Table 137. M24LRxx_Initiate function description

Prototype	`u8 M24LRxx_Initiate(ISO15693_Tag* MyTag, const u8 Request_flags);`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) **Tag_UID**: pointer on the UID (optional)
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).

M24LRxx_Fast_Initiate function

This function sends, through the CR95HF, a fast initiate command to a contactless tag in the field. The datarate of the response is multiplied by 2 (52 kbit/s in this case).

Table 138. M24LRxx_Fast_Initiate function description

Prototype	`u8 M24LRxx_Fast_Initiate(ISO15693_Tag* MyTag, const u8 Request_flags);`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) **Tag_UID**: pointer on the UID (optional)
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: Tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).

7.3.2 M24LRxx Energy Harvesting functions

These functions are specific to energy harvesting M24LRxxE-R devices.

Compilation management

The `USE_ENERGY_HARVESTING_COMMANDS` constant allows to compile or not these commands.

M24LRxx_ReadCfg function

This function sends, through the CR95HF, read Config command to a contactless in the field. On receiving the read Config command the M24LRxxE-R contactless tag reads the configuration byte and sends back its value.

Table 139. M24LRxx_ReadCfg function description

Prototype	`u8 M24LRxx_ReadCfg(ISO15693_Tag* MyTag, const u8 Request_flags,const u8* Tag_UID);`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) **Tag_UID**: pointer on the UID (optional)
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).

M24LRxx_Write_EH_Cfg function

This function sends, through the CR95HF, Write_EH_Cfg command to a contactless tag in the field. On receiving the write Energy Harvesting Configuration command, the M24LRxxE-R contactless tag writes the data provided to the configuration byte and reports the status of the command.

Table 140. M24LRxx_Write_EH_Cfg function description

Prototype	`u8 Write_EH_Cfg(ISO15693_Tag* MyTag, const u8 Request_flags,const u8* Tag_UID, u8 Configuration_Byte);`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) **Tag_UID**: pointer on the UID (optional) **Configuration_Byte**: the byte to write to the configuration byte (2)
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).
2. Only bits 0 to 2 are written, bit 3 is ignored.

M24LRxx_Write_DO_Cfg function

This function sends, through the CR95HF, Write_DO_Cfg command to a contactless tag in the field. On receiving the Write_DO_Cfg command the M24LRxxE-R contactless tag writes the data provided to the configuration byte and reports the status of the command.

Caution: M24LRxx_Write_DO_Cfg function description

Prototype	`u8 M24LRxx_Write_DO_Cfg (ISO15693_Tag* MyTag, const u8 Request_flags,const u8* Tag_UID, u8 Configuration_Byte);`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) **Tag_UID**: pointer on the UID (optional) **Configuration_Byte**: the byte to write to the configuration byte (2)
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).
2. Only bits 0 to 2 are written, bit 3 is ignored.

M24LRxx_SetRst_EH_en function

This function sends, through the CR95HF, a SetRst_EH_en command to a contactless tag in the field. On receiving the set reset Energy Harvesting Enable command, the M24LRxxE-R contactless tag set or reset the EHenable bit, within the volatile control register.

Table 141. M24LRxx_SetRst_EH_en function description

Prototype	`u8 M24LRxx_SetRst_EH_en (ISO15693_Tag* MyTag, const u8 Request_flags,const u8* Tag_UID, u8 Set_Reset);`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) **Tag_UID**: pointer on the UID (optional) **Set_Reset**: Enable or disable the energy harvesting. (2)
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).
2. Value to enable the energy harvesting: `M24LRxx_CONTROL_BYTE_EH_ENABLE`

Value to disable the energy harvesting: `M24LRxx_CONTROL_BYTE_EH_DISABLE`

M24LRxx_Check_EH_En function

This function sends, through the CR95HF, a Check_EH_En command to a contactless tag in the field. On receiving the Check Energy Harvesting enabled command, the M24LRxxE-R contactless tag reads the control register and sends back its value.

Table 142. M24LRxx_Check_EH_En function description

Prototype	`u8 M24LRxx_Check_EH_En (ISO15693_Tag* MyTag, const u8 Request_flags,const u8* Tag_UID);`
Input parameter	**Request_flags**: Specifies the actions to be performed by the contactless tag and whether corresponding fields are present or not. (1) **Tag_UID**: pointer on the UID (optional)
Output parameter	**MyTag**: pointer on the structure that contains the tag response
Return parameter	**M24LRxx_ERROR_CODE**: The command failed. **M24LRxx_ERROR_PARAMETERS_REQUEST_FLAGS**: The inventory flag is set. **M24LRxx_SUCCESS_CODE**: The tag answered the command.

1. The Address flag indicates if Tag_UID field will be added to the command (addressed mode).

M24LRxx_Get_Energy_Harvesting_Range function

This function returns the range of energy harvesting using the Configuration byte retrieved with a Read Cfg command.

Table 143. M24LRxx_Get_Energy_Harvesting_Range function description

Prototype	`u8 M24LRxx_Get_Energy_Harvesting_Range( u8 Configuration_Byte);`
Input parameter	**Configuration_Byte**: the byte containing the data available in the tag structure at dataField [`M24LRxx_CONFIGURATION_BYTE_CURSOR`]
Output parameter	None
Return parameter	**M24LRxx_CONFIGURATION_BYTE_RANGE_6_MA**: The energy harvesting is activated when at least 6 mA can be retrieved. **M24LRxx_CONFIGURATION_BYTE_RANGE_3_MA**: The energy harvesting is activated when at least 3 mA can be retrieved. **M24LRxx_CONFIGURATION_BYTE_RANGE_1_MA**: The energy harvesting is activated when at least 1 mA can be retrieved. **M24LRxx_CONFIGURATION_BYTE_RANGE_300_MA**: The energy harvesting is activated when at least 300 µA can be retrieved.

M24LRxx_Get_RF_BUSY_WIP function

This function returns the RF busy or Write in progress bit using the Configuration byte retrieved with a ReadCfg command.

Table 144. M24LRxx_Get_RF_BUSY_WIP function description

Prototype	`u8 M24LRXX_Get_RF_BUSY_WIP(u8 Configuration_Byte);`
Input parameter	**Configuration_Byte**: the byte containing the data available in the tag structure at dataField[`M24LRxx_CONFIGURATION_BYTE_CURSOR`]
Output parameter	None
Return parameter	**M24LRxx_CONFIGURATION_BYTE_RF_BUSY**: Indicate to the I2C bus when communication in RF is happening. **M24LRxx_CONFIGURATION_BYTE_RF_WIP**: Indicate to the I2C bus when data has been changed by the RF.

M24LRxx_Get_EH_mode_Configuration_Byte function

This function returns the Energy Harvesting state bit using the Configuration byte retrieved with a ReadCfg command.

Table 145. M24LRxx_Get_EH_mode_Configuration_Byte function description

Prototype	`u8 M24LRxx_Get_EH_mode_Configuration_Byte (u8 Configuration_Byte);`
Input parameter	**Configuration_Byte**: the byte containing the data available in the tag structure at dataField[`M24LRxx_CONFIGURATION_BYTE_CURSOR`]
Output parameter	None
Return parameter	**M24LRxx_CONFIGURATION_BYTE_EH_ENABLED**: The energy harvesting is enabled. **M24LRxx_CONFIGURATION_BYTE_EH_DISABLED**: The energy harvesting is disabled.

M24LRxx_Get_EH_mode_Control_Register function

This function returns the Energy Harvesting state bit using the Control Register retrieved with a Check EH En command.

Table 146. M24LRxx_Get_EH_mode_Control_Register function description

Prototype	`u8 M24LRxx_Get_EH_mode_Control_Register (u8 Control_Register);`
Input parameter	**Configuration_Byte**: the byte containing the data available in the tag structure at dataField[`M24LRxx_CONFIGURATION_BYTE_CURSOR`]
Output parameter	None
Return parameter	**M24LRxx_CONFIGURATION_BYTE_EH_ENABLED**: The energy harvesting is enabled. **M24LRxx_CONFIGURATION_BYTE_EH_DISABLED**: The energy harvesting is disabled.

8 Application example

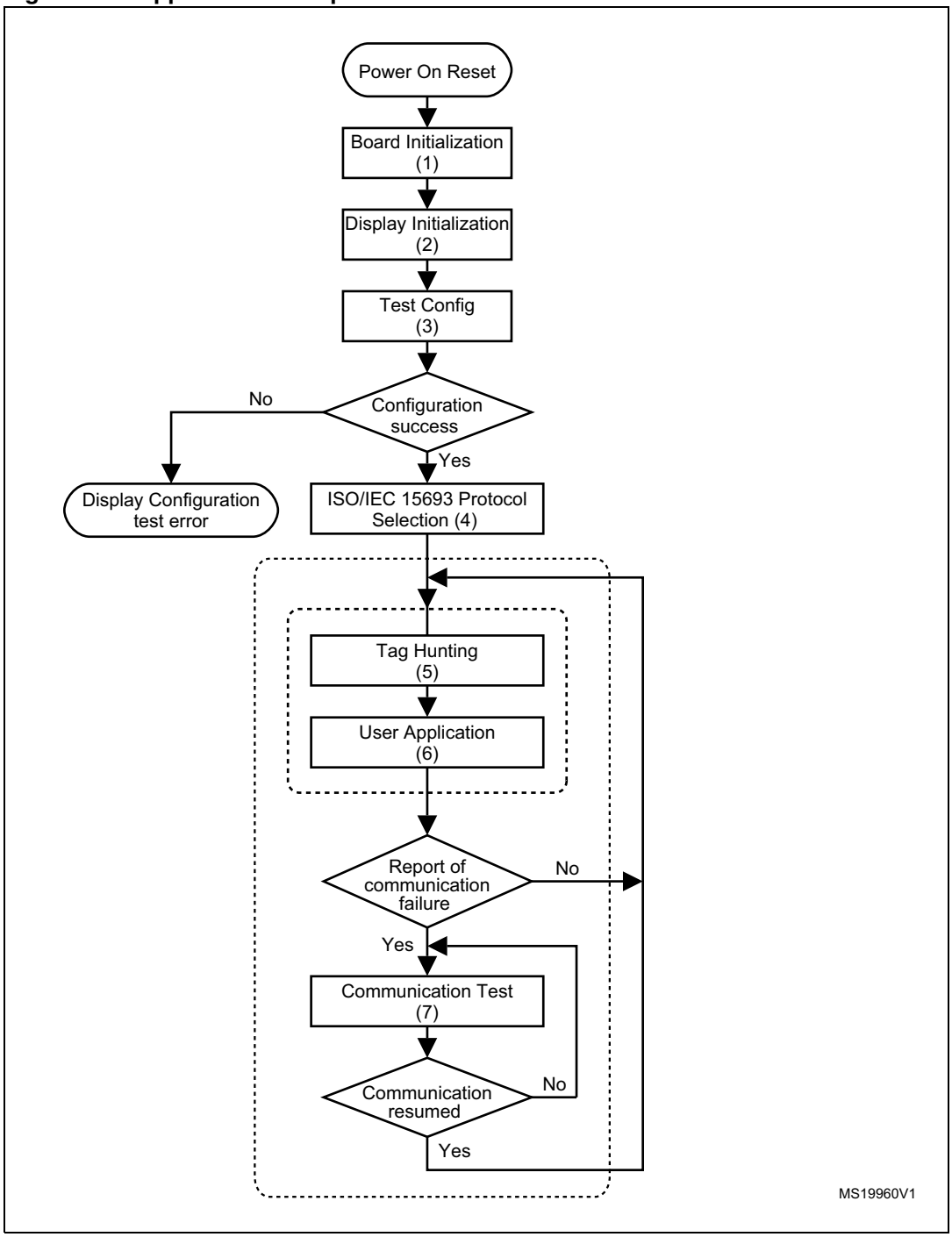
The application example provided with this firmware illustrates the library functions within an application environment. The application example focuses on the management of an inventory command and on the low power state of STM8L and CR95HF devices. Different low power states can be issued by using the joystick of STM8L Eval board.

This application example configures the STM8L and CR95HF in order to illustrate some CR95HF and STM8L functionalities.

8.1 Main functions

Figure 4 presents the main function. Some details are given below in the next chapter.

Figure 4. Application example main functions



The application example is located in the main.c file and user_application.c files.

8.1.1 Board initialization

This function configures the STM8L resources needed:

- Interface bus to communicate with CR95HF device (through SPI or UART bus)
- Timers
- LCD
- I2C
- GPIO (LED & joystick)

Delay management

The delays are handled by two timers. Timer #4 allows to introduce a delay between two instructions, whereas timer #2 is used with interrupts to control the execution time of a group of instructions.

8.1.2 Display initialization

The LCD screen is initialized with the Logo of the STMicroelectronics company and the CR95HF product name.

8.1.3 Test configuration

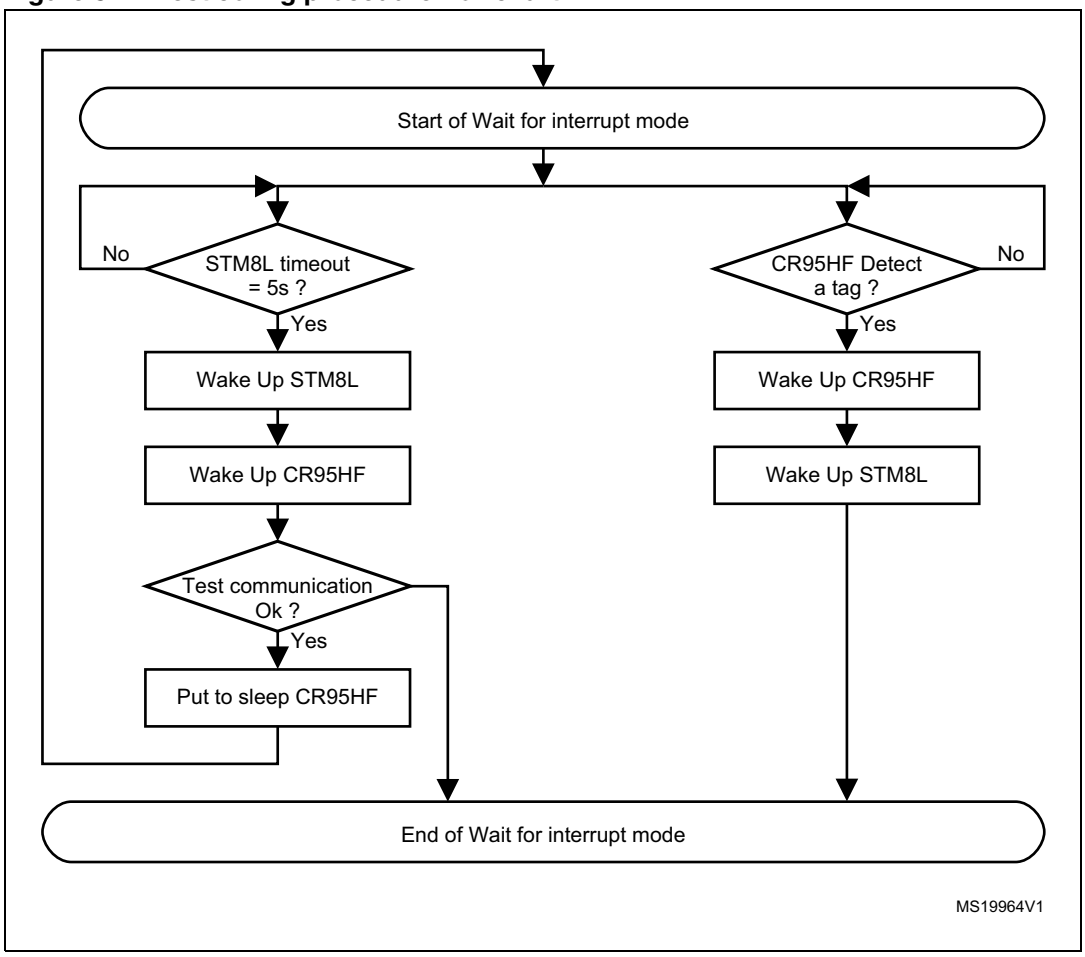
This function configures the CR95HF for the application example.

Table 147. Test config procedure

Step	Action
1	The first step is to display the interface bus selected to communicate with CR95HF device.
2	A pulse is sent to CR95HF to wake it up.
3	An ECHO command is sent while there is no valid answer. If the echo fails, a message is displayed on the screen and the following procedure is repeated while the echo command fails: – Send a pulse to CR95HF – Echo command – SPI Reset At this point, STM8L and CR95HF can communicate which each other.
4	The internal EEPROM of the STM8L is unlocked.
5	A protocol select command is issued.
6	A calibration of the tag detection feature is launched.
7	A success message is displayed.

Figure 5 details these different steps.

Figure 5. Test config procedure flowchart



8.1.4 ISO/IEC 15693 Protocol selection

Before beginning to communicate with a contactless tag, the RF protocol and its parameters should be set. This step ensures that it is done before entering the main loop.

8.1.5 Tag hunting

The Tag Hunting function sends an inventory and a Get system Info command. The contactless tag responses are stored into the MyTag15693 structure.

The inventory sent is an inventory 16 slots, therefore if several contactless tags answer the command, then only one UID will be retrieved or potentially 16 structures will be needed. The solution proposed and used in the inventory 16 slots is to store the UIDs retrieved into the internal EEPROM of the STM8L. When the inventory is finished, the number of contactless tags seen is available thanks to the ISO15693_Inventory_16slots_Nb_UID_Retrieved variable.

Commands may be addressed to each tag seen individually. The procedure is exposed using the Get System Info Command. If only one tag answers the inventory, its UID is stored in the STM8L EEPROM, and on top of that the UID is written in the Tag structure.

For one slot inventory, only UID can be retrieved or there will be a collision. To prevent a collision (with 1 or 16 slots), a mask can be added to the inventory command. The mask corresponds to the LSB bits of the UID, the contactless tag compares those with its UID and answers only if they match. The length of the mask is in bits, unused bits in the byte must be reset in order to transmit whole bytes.

8.1.6 Display

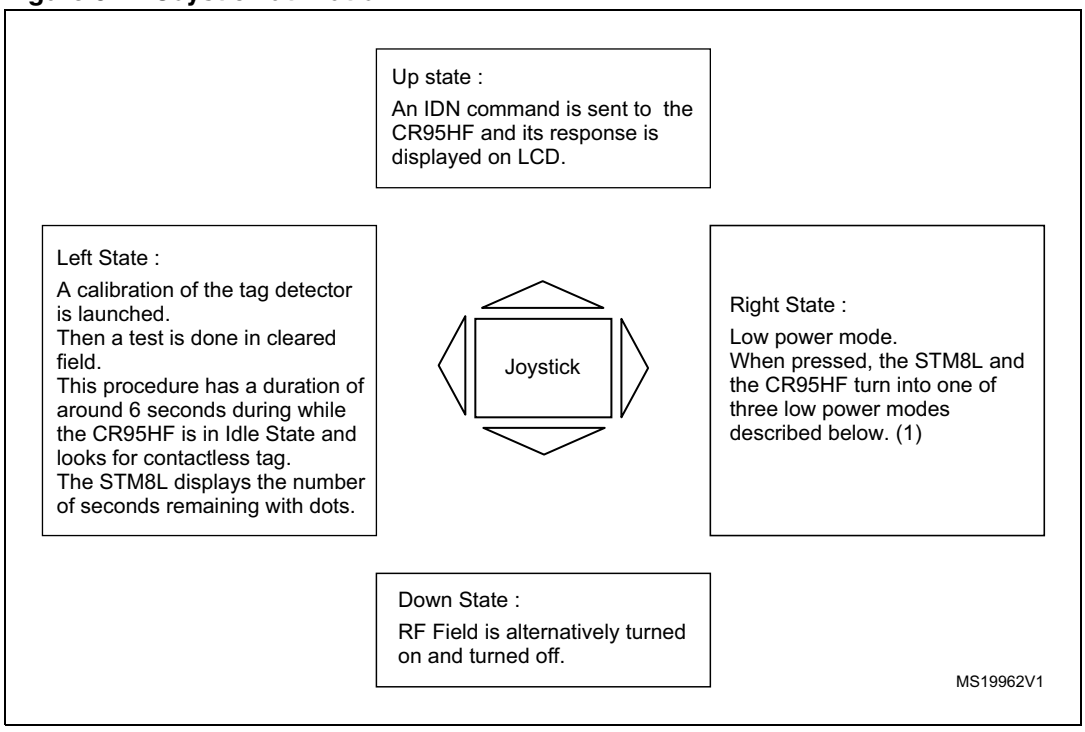
If the inventory and Get System Info commands passed, then the UID and the ICRef of the tag are displayed. Note that as the MSB byte is common to all contactless tags, it is not displayed.

For STMicroelectronics manufactured contactless tags, a database containing the ICref is present in the miscellaneous files.

8.1.7 User application

User application is based on the utilization of the joystick present on the STM8L 1528-Eval board. This enables the user to perform the actions described in *Figure 6*:

Figure 6. Joystick utilization



1. A detection of a contactless tag or pressing the Reset button can exit this mode.

8.1.8 Low power modes

The CR95HF embeds a low power state named "tag detector state". In this state, the CR95HF device is turned in Idle state and will periodically check if a tag is present in the volume operation. If a tag is detected, the CR95HF devices will wake-up.

The STM8L integrates different low power modes. Once STM8L is turned in low power mode, it can be waken up by itself using an internal timer or by an external interruption.

Table 148 sums up the three low power modes available in this application example.

Table 148. Low power modes descriptions

Mode	Periodical wake-up	STML8		CR95HF	
		Low power mode	Wake-up source	Low power mode	Wake-up source
1	STM8L	Wait For Interrupt	– Internal RTC clock – CR95HF	Tag Detection	– Tag Detection – STM8L
2	CR95HF	Halt	CR95HF	Tag Detection	Tag Detection
3	CR95HF	Halt	CR95HF	– Tag Detection – Timer	– Tag Detection – Internal Timer

In each case, both CR95HF and STM8L devices are turned into low power state and the Tag detection states are activated. The difference between the three states is the second source of the system wake-up. It can be issued either by CR95HF or by STM8L.

Mode#1: Wait For Interrupt mode

In this state, the CPU of the STM8L is deactivated. An RTC interrupt is sent every 5 seconds to STM8L. If the CR95HF has not seen a tag during 5 seconds, the STM8L's RTC clock wakes up the MCU which sends an Interrupt to wake up the CR95HF. Then the communication between CR95HF and STM8L is tested and the CR95HF is turned into Idle state again before the STM8L returns into Wait For Interrupt mode. *Figure 7* schematizes these actions.

Figure 7. Wait for interrupt mode schematic

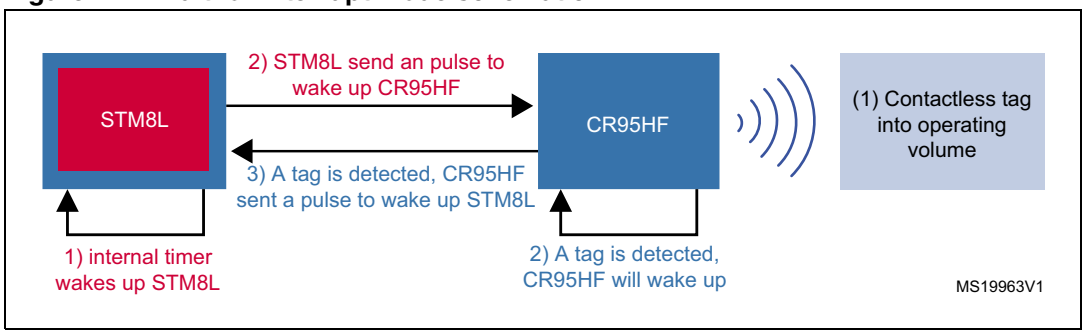
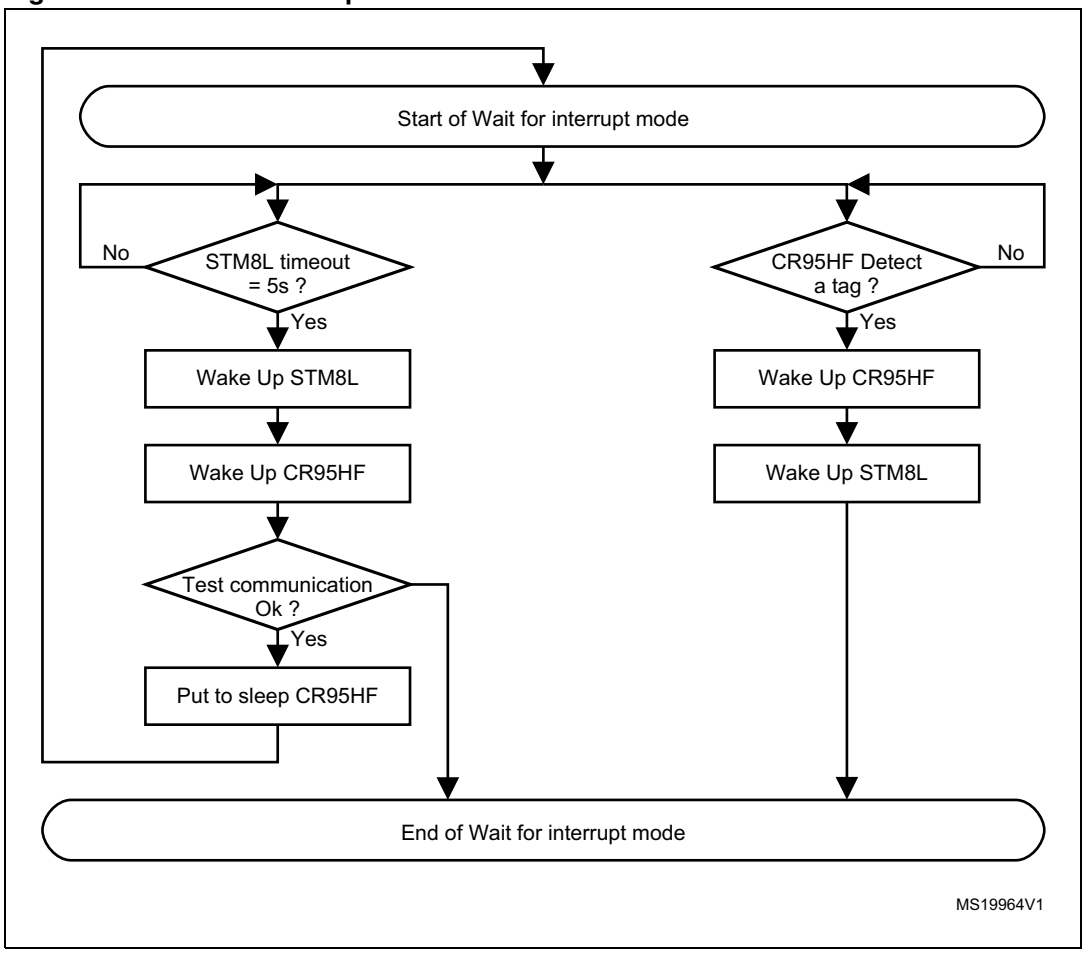


Figure 8. Wait for interrupt mode flowchart



Mode#2: Halt Mode

The Halt State is the low power consumption state of the STM8L. The main clock and all peripherals are switched off. Only an interrupt on I/O pin or reset can wake up the STM8L. In this state, only tag detection or a pressure on the reset button can wake up the system.

Figure 9. Halt mode schematic

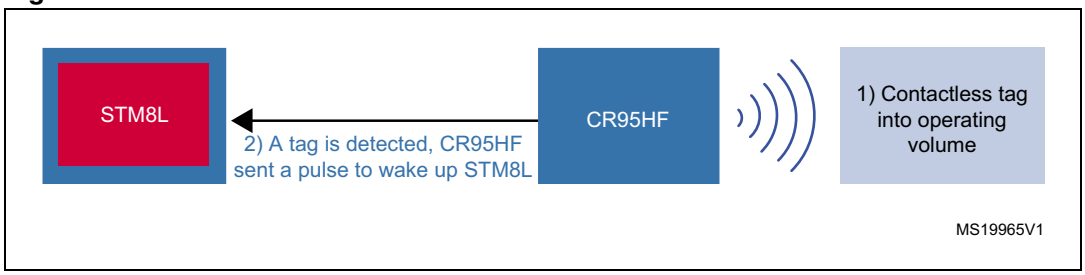
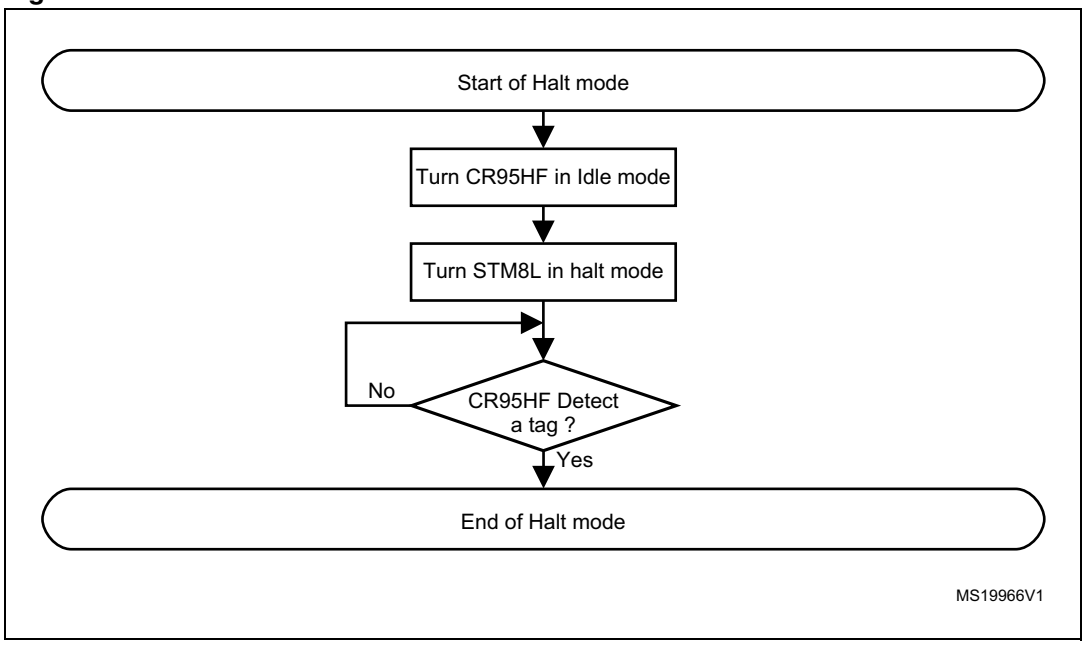


Figure 10. Halt mode flowchart



Mode#3: Halt CR95HF Timer mode

In this state, the STM8L is turned into halt state and the CR95HF wakes up periodically around every 5 seconds thanks to its internal Timer. So when the CR95HF wakes up, it does so with the STM8L which determines what caused the waking up. If it is the timer, then the STM8L sends again the Idle command. If a tag has been detected, the field is turned on in ISO/IEC 15693 protocol to communicate with the contactless tag.

Note: An $\overline{IRQ_IN}$ interrupt can wake up the CR95HF in all modes. Indeed, it is warmly recommended to keep an external interrupt as a way of waking up the CR95HF.

Figure 11. Halt CR95HF Timer mode schematic

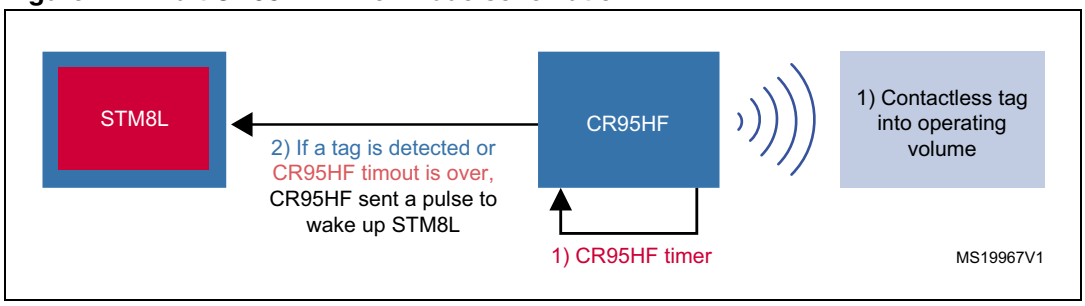
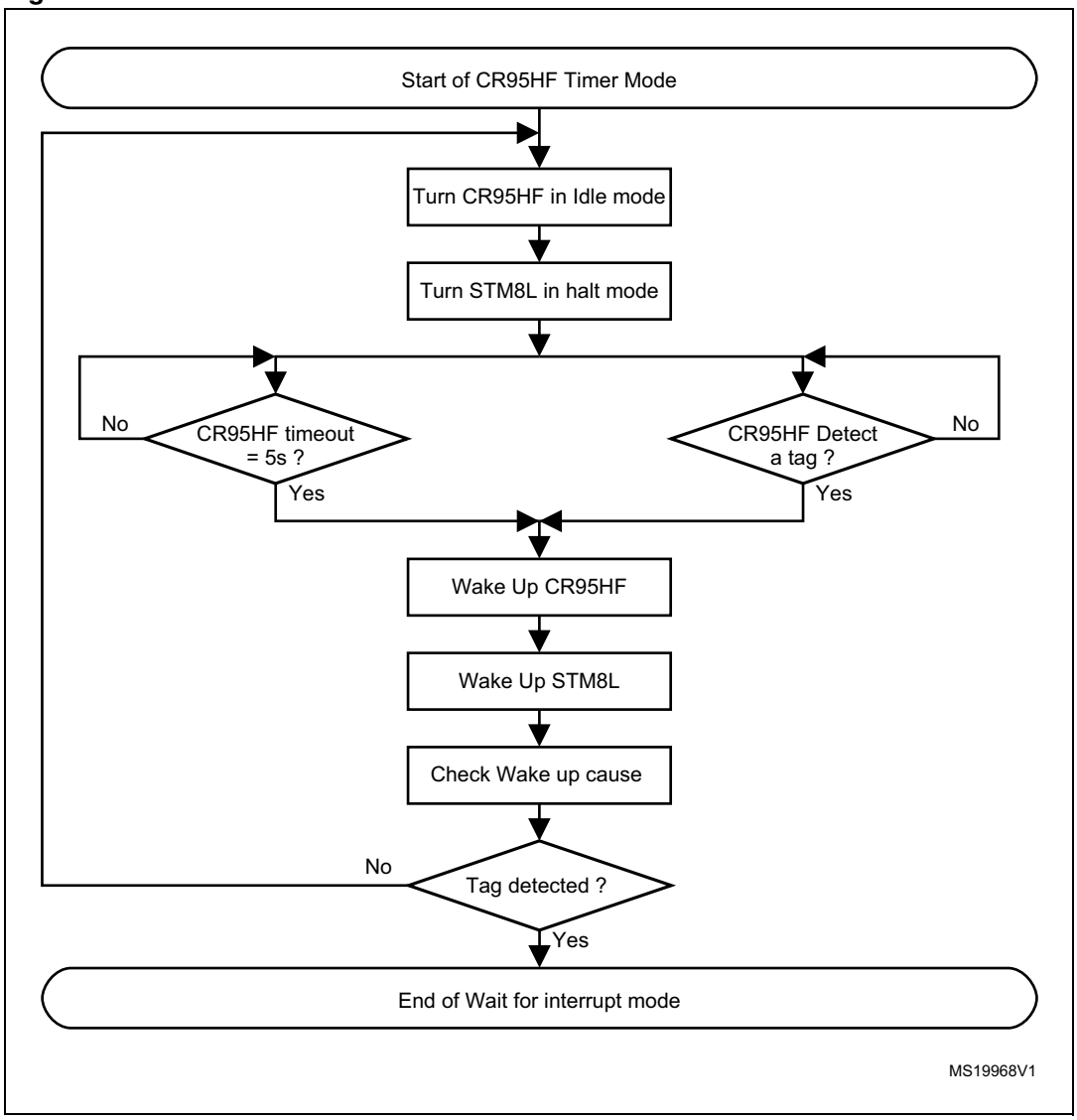


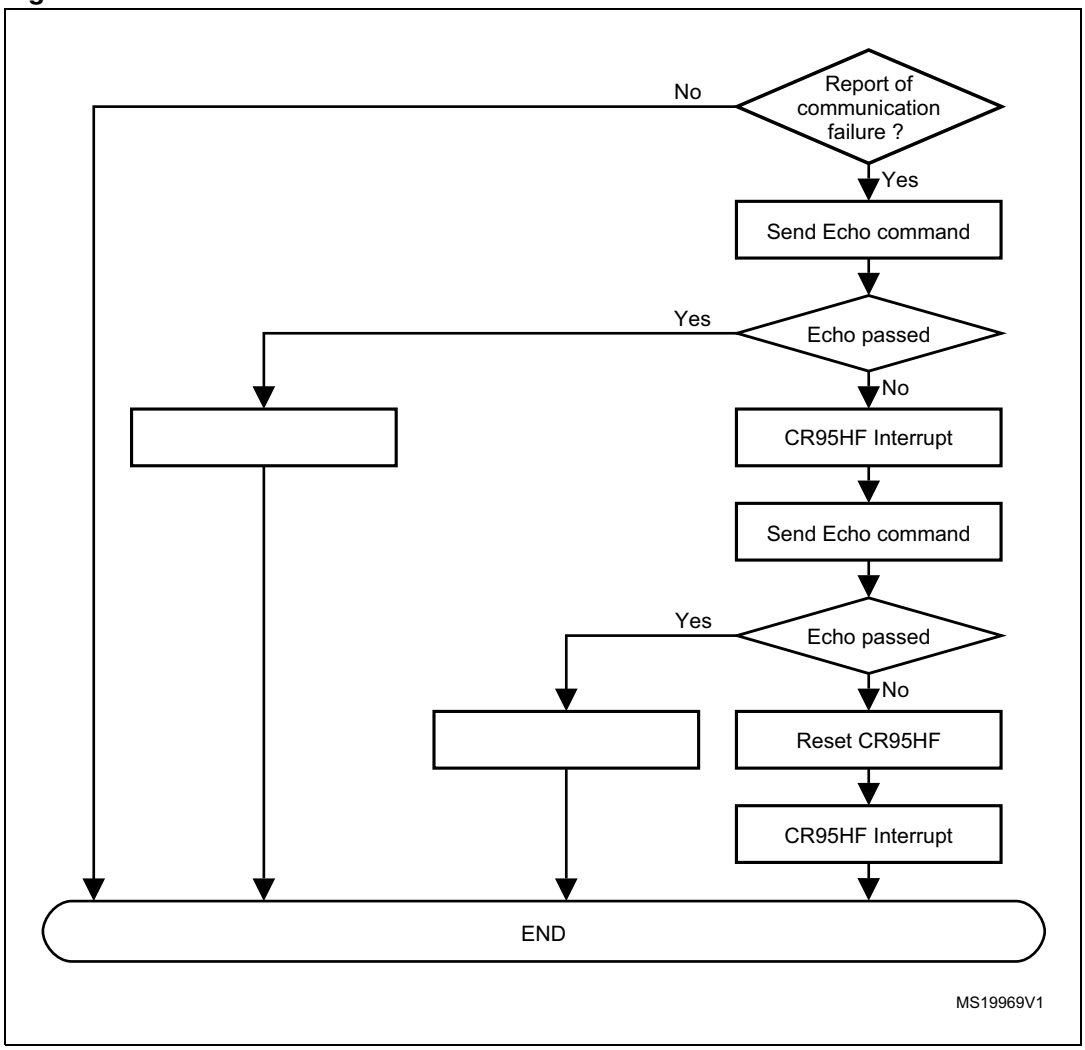
Figure 12. Halt CR95HF Timer mode flowchart



8.1.9 Communication test

While waiting for an answer from the CR95HF, a timer is launched and the polling procedure is active during a limited period. If, at the end of this period, no answer is ready, the STM8L assumes that no answer will come and it increments a global counter called communication_watchdog. At the end of the secondary loop, this counter is checked and compared with a maximum value of tolerance. If it overtakes this value, a procedure is launched in order to resume the communication possibly lost.

Figure 13. Communication test flowchart



Note: While the `communication_watchdog` variable is not reset, this function will be called within the main loop.

8.2 Hardware

Two STM8L boards can be used with this firmware:

- The STM8L discovery board. It can be used as a demonstration board.
- The STM8L evaluation board. It can be used as a development board.

8.2.1 STM8L discovery board

The STM8L Discovery board helps you to discover the STM8L ultra low power features and to develop and share your applications. It is based on an STM8L152C6T6 and includes an ST-Link embedded debug tool interface, an LCD, LEDs and push buttons.

The LCD of the discovery board used the most of available GPIO and the available GPIO is not enough to communicate with the CR95HF devices. This is the reason why the LCD of STM8L discovery board is deactivated.

The LEDs are the only information source for the user application.

The STM8L discovery board documentation is available on STM website.

8.2.2 STM8L evaluation board

The STM8L1528-EVAL evaluation board is designed as a complete demonstration and development platform for the STM8 core based STM8L152M8T6 microcontroller with I2C, two SPI channels, 3 USART channels, 12-bit ADC, two 12-bit DACs, an LCD driver, internal SRAM, data EEPROM and Flash program memory as well as SWIM debugging support.

The full range of hardware features on the board is provided to help you evaluate all the MCU peripherals (motor control, USART, microphone, audio DAC, LCD, IR LED, IrDA, SPI Flash, MicroSD card, temperature sensor, EEPROM… etc.) and develop your own applications. Extension headers make it possible to easily connect a daughter board or wrapping board for your specific application.

An ST-LINK V2 is integrated on the board as an embedded in-circuit debugger and programmer for the STM8 MCU.

The STM8L evaluation documentation is available on STM website.

8.2.3 CR95HF plug board

The PLUG-CR95HF-B is a board which includes a CR95HF device and a matched antenna. A host configured as a master can communicate with CR95HF through the SPI bus.

The PLUG-CR95HF-B is powered through the Vps pin 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.

8.3 Software

8.3.1 ST Visual Develop

ST Visual Develop (STVD) provides an easy-to-use, efficient environment for start-to-finish control of application development - from building and debugging the application code to programming the microcontroller.

STVD is available on STM web site at
http://www.st.com/internet/evalboard/product/210567.jsp

8.3.2 Cosmic compiler

Cosmic is the compiler toolchain used by ST Visual Develop. There is a 1 year free license limited to 32 Kbytes of code and data which requires registration.

For further information about the license, see the Cosmic Software website.

8.4 Project

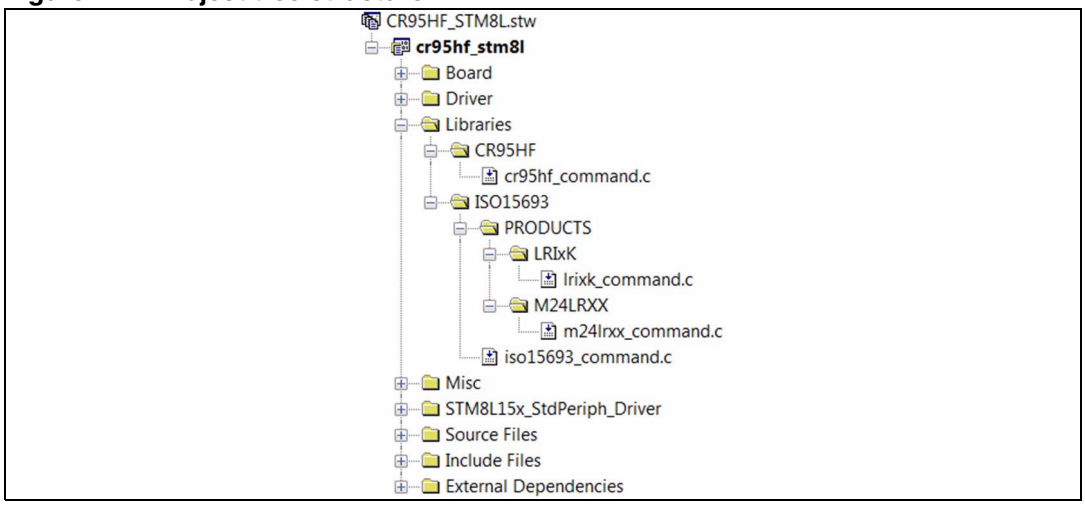
The project was built using the MCD standard library which is included in the STM8L15x_StPeriph_Driver folder. In order to update the library, a copy of the stm8l15x_it.h, stm8l15x_it.c and the stm8l15x_conf.h is required to store the modifications made within those files. Then the whole folder can be updated with the latest version before replacing the three files previously copied.

8.4.1 Opening the Project:

First step: Launch ST Visual Develop

Second Step: In the File menu, open Workspace, browse to the project folder and select *CR95HF_STM8L.stw*

Figure 14. Project tree structure



8.4.2 Compilation / Debug

In order to compile and debug the project, there are few steps to perform.

1) First step: Build

On menu bar, click on Build menu, then click on Build; the project will be built and any errors will be displayed on the lower part of the screen.

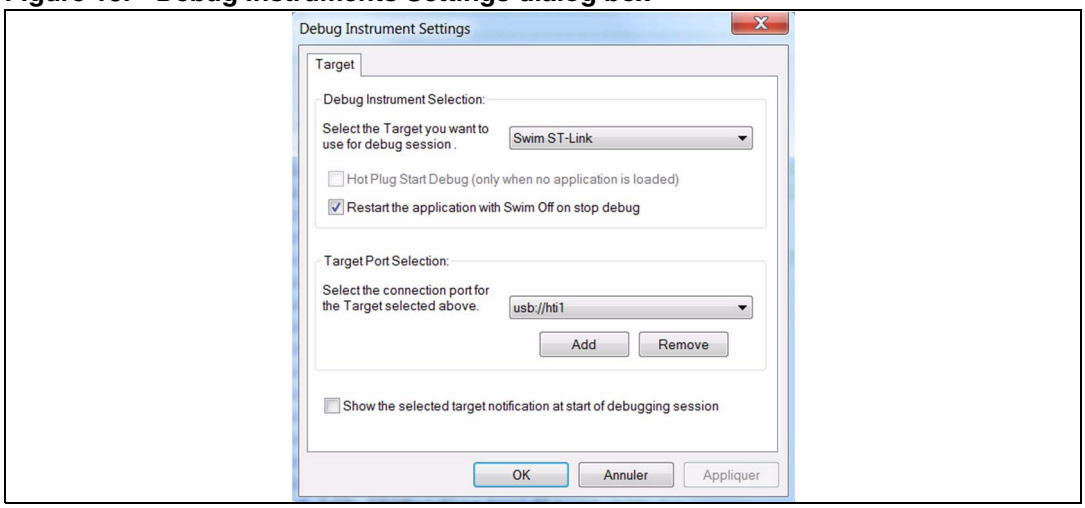
2) Second Step: Configure the Debug Instruments

Connect a USB cable or an ST-Link between the board and the USB port of the computer. Note that using the Swim, instead of ST-Link embedded, requires a power supply device.

See the board datasheet and User Manual for further information.

->DebugInstruments Settings, Select Swim ST-Link, the target Port Selection should be updated automatically. If not, select Add and select the USB port or let the software detect the port automatically.

Figure 15. Debug Instruments Settings dialog box



3) Third Step: Launch the debugger

->Debug-> Start Debugging or press the button

Once the debugger is launched, the following menu allows to run the application, run it step by step, by function, etc… and to observe the behavior of the code and the values taken by the different variables.



Refer to ST Visual Develop Information Manuals for further information about the debugging mode.

8.5 Compilation management

This firmware embeds all the commands of CR95HF devices, LRIxK and M24LRxx contactless tag. Furthermore, some advanced functionalities are available such as the tag detection state or the read/write multiple block process. All these features consume memory space and can be removed to minimize the code size.

8.5.1 Conditional compilation

In order to save STM8L memory space, several actions have been performed. The main one is the conditional compilation. This allows to determine what is needed. There are three levels of conditional compilation.

Level 1: STM8L15x_StPeriph_Driver

All the functions that are not called within the library are not compiled. In order to use those functions, there are two solutions:

Solution 1: Locate in the .h file of the peripheral concerned the following command (right after the includes) //`#define USE_FULL_"peripheral concerned"_ Board` [(a)] and uncomment the line. As a consequence, all the functions will become available.

Solution 2: Locate the function to release and erase the `#ifdef USE_FULL_"peripheralconcerned"_ Board` and `#endif` commands. If the function is in a group of functions, insert a #endif before the function and a `#ifdef USE_FULL_"peripheralconcerned"_ Board` right after. This procedure should be done in both the .c and the .h files.

Level 2: Board Specificities

This level is completely transparent because it is related to boards. The specific I/O configurations linked to one board (STM8L 1528-Eval for example) are not compiled on the other board (STM8L Discovery for example).

Level 3: Functionalities

Several functionalities are available within the libraries. *Table 149* lists the functionalities and the commands to uncomment to use them.

Table 149. Functionalities description

Name of the functionality (*brief*)	Description	Code line to uncomment	Location
STM8L-1528 Eval	To use the STM8L-1528 Eval board	`#define BOARD_SELECTED_EVALBOARD`	Board_config.h
STM8L-Discovery Board	To use the STM8L-Discovery board	`#define BOARD_SELECTED_DISCOVERYBOARD`	Board_config.h
RTC clock function	Used in Low Power Mode, allows to generate an interruption at programmed time	`#define USE_RTC_CLOCK_FUNCTION`	Board_config.h

a. Where "peripheral concerned" is the name of the peripheral as written in its file.

Table 149. Functionalities description (continued)

Name of the functionality (*brief*)	Description	Code line to uncomment	Location
External EEPROM	Allows to Read the memory of a contactless tag and then to store it into the external EEPROM of the STM8L	`#define USE_EXTERN_EEPROM_FUNCTION` (STM8L-1528 Eval Functionality should be used)	Board_config.h
Full ISO15693 commands	Release the Write Multiple Block command	`#define USE_FULL_ISO15693_COMMANDS`	ISO15693_command.h
Energy Harvesting commands	Release the Energy Harvesting commands related to M24LRX-E devices	`#define USE_ENERGY_HARVESTING_COMMANDS`	M24LRxx_command.h
User Application	Demonstration of some commands	`#define USE_USER_APPLICATION`	Main.c

8.5.2 Polling method

In SPI communication, the application must ensure that the CR95HF is ready to transmit data before trying to receive a response. This could be performed by two ways:

- Polling byte: the STM8L transmits a control byte (0b11) to the CR95HF, and checks the byte received in response.
- Polling $\overline{\text{IRQ_OUT}}$ (CR95HF): when the CR95HF is ready to transmit data, a transition on its $\overline{\text{IRQ_OUT}}$ pin can be seen.

The software lets the user chose its method by changing the value of the constant below in the CR95HF_driver.h

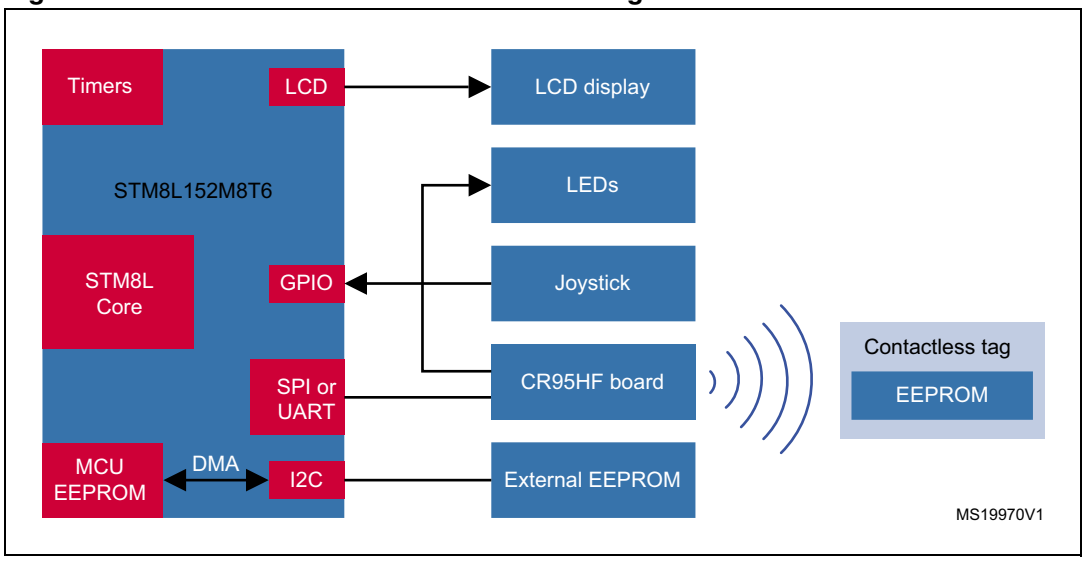
```
#define WAITING_CR95HF_RESPONSE_METHOD      WAITING_USING_IRQ
```

The two choices are `WAITING_USING_IRQ` and `WAITING_USING_POLLING`

8.6 Hardware layout and configuration

The STM8L1528-EVAL evaluation board is designed around the STM8L152M8T6 (80-pin LQFP package). *Figure 16* illustrates the connection between STM8L152M8T6 and peripherals (CR95HF device, LCD screen, EEPROM, USART, and embedded STLINK).

Figure 16. STM8L1528-EVAL hardware block diagram



For more details about the STM8L1528-EVAL board, please refer to the UM1037 user manual.

8.7 Pinout description

The tables below present the I/Os used and their configurations for the two boards.

Table 150. Communication with CR95HF I/Os

STM8L board Pin (1)	SPI				UART			
	Name	Direction	Configuration		Name	Direction	Configuration	
		DDR (2)	CR1 (3)	CR2 (4)		DDR	CR1	CR2
PB7	MISO	Input	Floating	No interrupt				
PB6	MOSI	Output	Push-Pull	10 MHz				
PB5	SCK	Output	Push-Pull	10 MHz				
PB4	NSS	Output	Open Drain	2 MHz	NSS	Output	Open Drain	2 MHz
PB3	SSIO	Input	Floating	No interrupt	SSIO	Input	Floating	No interrupt
PC2	$\overline{\text{IRQ_IN}}$	Input	Pull-up	No interrupt	UART_RX	Input	Pull-up	No interrupt
PC3	$\overline{\text{IRQ_OUT}}$	Output	Open Drain	2 MHz	UART_RX	Output	Push-Pull	10 MHz

1. For both STM8L discovery and evaluation board.
2. Direction data register
3. Control register 1
4. Control register 2

Note: In low Power state, PC2 is set with Interrupt. All pins are named from STM8L point of view.

Table 151. Board STM8L-1528Eval Specific I/Os

Pin	Name	Direction	Configuration	
		DDR	CR1	CR2
PI3	LCD MISO	Input	Floating	No interrupt
PI2	LCD MOSI	Output	Push-Pull	10 MHz
PI1	LCD SCK	Output	Push-Pull	10 MHz
PF2	LCD NSS	Output	Open Drain	2 MHz
PG0 - PG4	Joystick	Input	Floating	No interrupt
PH0 - PH3	LED	Output	Push-Pull	2 MHz
PC1	I2C SDA	Input	Pull-up	No interrupt
PC0	I2C SCL	Input	Pull-up	No interrupt

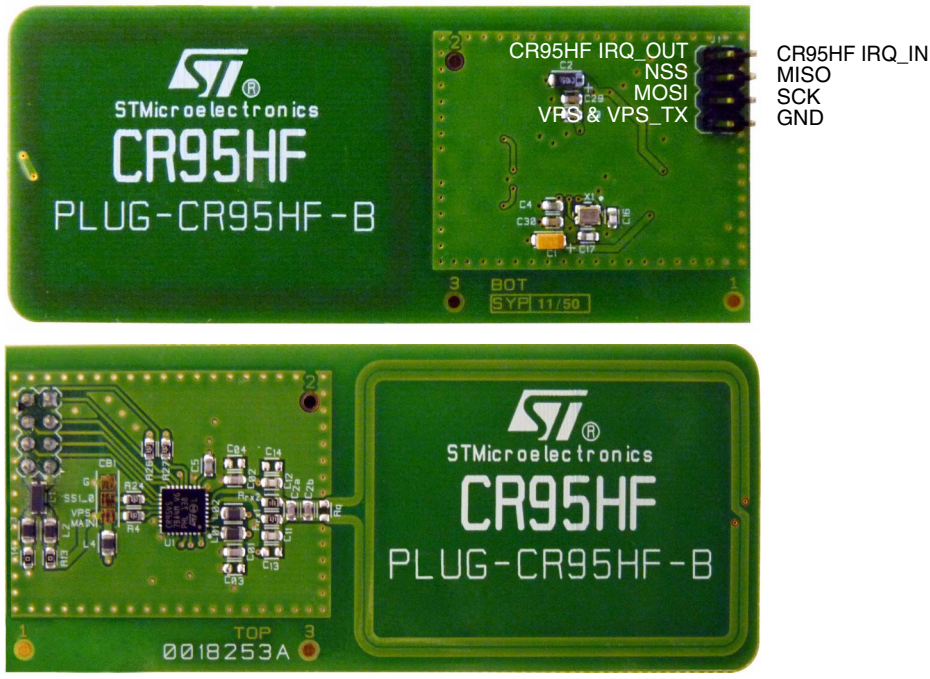
Table 152. Board STM8L-Discovery Board Specific I/Os

Pin	Name	Direction	Configuration	
		DDR	CR1	CR2
PC7	Blue LED	Output	Push-Pull	2 MHz
PE7	Green LED	Output	Push-Pull	2 MHz

8.7.1 PLUG-CR95HF-B Board pin

Figure 17 presents the eight I/Os of a PLUG CR95HF-B board, as an example.

Figure 17. PLUG-CR95HF-B Board I/Os



8.8 Switching between STM8L 1528-Eval and STM8L Discovery boards

Figure 153 shows, in three steps, how the user can switch from the STM8L 1528-Eval to STM8L Discovery board. To proceed the other way, perform the reverse actions.

Table 153. Switching procedure

Step	Name	Actions
1	Board Selection	In order to change the board selected, in the board_config.h file: – uncomment the command `#define BOARD_SELECTED_DISCOVERYBOARD` – comment the command `#define BOARD_SELECTED_EVALBOARD`
2	Functionalities Cut	To disable the external EEPROM functionality, in the board_config.h file: comment the command `#define USE_EXTERN_EEPROM_FUNCTION`
3	Display Commands	In the main.c file, comment all functions: – Beginning by `LCD_` – `Wait_Display` – `Board_Init_Display` – `Display_Motion` – `ISO15693_Display_Tag_Answer_Inventory_GetSys_Info`

9 Revision history

Table 154. Document revision history

Date	Revision	Changes
19-Dec-2011	1	Initial release.
07-Feb-2012	2	Updated *Figure 17: PLUG-CR95HF-B Board I/Os on page 103*.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY TWO AUTHORIZED ST REPRESENTATIVES, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2012 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com