RMU Definition of Communication Interface

Release 2.7

Document History:

Version	Release data	Replaces
1.0	5-23-1808	Internal test version
1.1	6-23-1808	Add UII for data operation
1.2	7-11-2008	Add the Access operation; change the operation of read/config parameter.
2.0	7-28-2008	The first release version
2.1	10-7-2008	Add the command to read RMU information
2.2	11-24-2008	Test version
2.3	12-2-2008	Add the operation of ‘read the frequency config and of setting frequency’ (For RMU900E family)
2.4	1-9-2009	Add the operation of read the antenna config and of setting antenna (For RMU900E family)
2.5	3-8-2009	Add the operation of identy a tag (single inventory round)
2.6	3-17-2009	Add the ‘Wiegand identify’
2.7	4-15-2009	Add the ‘read the tag data’(no indicator UII) and ‘write the tag data’(no indicator UII)

1. Communication interface

The RMU family of UHF RFID module is communicated with host computer by UART interface. The Host computer (PC or MCU) can receive the information that is return back from RMU according to the specific format. We provided mass API functions for perform RMU base on Window flat, with which user can easily do secondary development work. The API function is written by C++ and is compiled to Windows Standard DLL.

Table 1-1 The UART parameters is supported by RMU

Bits per second	57600
Number of bits	8
Odd/even	none
Stop bit	1

2. Frame Format

The frame that is sending from host computer to RMU is called ‘command’, and the frame that is sending from RMU to host computer is called ‘response’. All of the following data segments length is in bytes. The normal frame format refers to table 2-1 and table 2-2.

Table 2-1 Frame format of command

Field	SOF	LENGTH	CMD	PAYLOAD	*CRC-16	EOF
bits	1	1	1	< 254	2	1

Table 2-2 Frame format of response

Field	SOF	LENGTH	CMD	STATUS	PAYLOAD	*CRC-16	EOF
bits	1	1	1	1	< 253	2	1

Note: the * is optional, hereinafter the same.

2.1 SOF (Start of Frame)

SOF is a constant byte (SOF=0xAA), and is the beginning of frame.

2.2 LENGTH

LENGTH is bytes from SOF to EOF. (include <LENGTH>、<CMD>、<STATUS>、<PAYLOAD>、<CRC-16>).

2.3 CMD

The definition of CMD Frame is referring to table 2-3.

Table 2-3 Definition of CMD Frame

Bit	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Description	Control bits of CRC	RMU Command						
Function	0 = no CRC-16 in frame 1 = CRC-16 in frame	Refer to table 2-6						

The bit 7 is used to check function of CRC-16, user can select it or not. And the CRC-16 status that is return back from RMU must be complied with the CRC-16 status of host computer. The commands of read/write refer to table 2-5.

2.4 STATUS

Status is the perform status of RMU response to Host computer command. Status only occurred in RMU response, there is no status in Host computer. Four MSBs of Status is the common indicator; Four LSBs of Status is the specific status of individual command. The definition of common indicator Status refers to table 2-4, the definition of LSB refers to the definition of individual command.

Table 2-4 the definition of Common Indicator STATUS

Bits	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3 ~ 0
Description	1 = failed to perform command 0 = success to perform command	1 = failed to verify the CRC 0 = success to verify the CRC	reserved	reserved	Refer to definitions of command

2.5 PAYLOAD

Payload is the data that will be tranmitted. Except for the valid bytes that been defined in all commands, the Payload Length can be extended as needed in range of LENGTH, RMU do not operate point to that.

2.6 CRC-16

CRC-16 is a calculated from <LENGTH>、<CMD>、<STATUS>(response) and <PAYLOAD>. CRC-16 can be selected by CMD bit 7.

When the Host Computer fails to verify the CRC-16, it will return a specific format response. Please refer to table 2-5. The bytes of Status are 0xC0.

Table 2-5 response of failing to verify CRC

Field	SOF	LEN	CMD	STATUS	*CRC	EOF
Bits	1	1	1	1	2	1

2.7 EOF (End of Frame)

EOF is one byte constant (EOF=0x55), and is the ending of frame

2.8 RMU command list

Table 2-6 RMU command list

Command	**Value(hex)**	**Function**	**Response waiting time(ms)**
RMU_GET_STATUS	00	Acquire status	500
RMU_GET_POWER	01	Get the output power setting	500
RMU_SET_POWER	02	Set the output power	500
RMU_OPEN_POWER	03	Open the Amplifier	500
RMU_CLOSE_POWER	04	Close the Amplifier	500
RMU_GET_FRE	05	Get the frequency setting	500
RMU_SET_FRE	06	Set frequency	500
RMU_GET_VERSION	07	Get the RMU information	500

RMU_GET_ANT	08	Get the Antenna setting	500
RMU_SET_ANT	09	Set antenna	500
RMU_INVENTORY	10	Get tag UII(single inventory round)	500
RMU_INVENTORY_ANTI	11	Get tag UII(Anti-collision inventory)	500
RMU_STOP_GET	12	Stop Reading tags	500
RMU_READ_DATA	13	Read tag data	500
RMU_WRITE_DATA	14	Write tag data	800
RMU_ERASE_DATA	15	Erase tag data	800
RMU_LOCK_MEM	16	Lock tag	800
RMU_KILL_TAG	17	Kill tag	800
RMU_INVENTORY_SINGLE	18	Get tag UII(single inventory round)	500
RMU_WIEGAND_INVENTORY	19	Inventory with Wiegand output	500
RMU_SINGLE_READ_DATA	20	Read tag data(without seting UII)	500
RMU_SINGLE_WRITE_DATA	21	Write tag data(without seting UII)	500

2.9 Insert Byte

We ensure the unique of SOF and EOF by inserting a byte in the communication and to avoid the SOF and EOF bytes occurred in data. At once the data with SOF and EOF have the 0xAA, 0x55 and 0xFF bytes, the Sender should insert a 0xFF byte at the beginning of frame. And the receiver should delete the inserted byte after he received it. The inserted bytes should not be calculated in the LENGTH. The example refers to appendix E.

2.10 Response time of RMU

IF the RMU could not response to it after the host computer send commands to RMU in some time, we think the command format is incorrect or RMU command must be encountered unpredictable error in the implementation. Then the PC can send the command again. The response time of command refers to table 2-6.

3. Command Definition

3.1. Query status

3.1.1 Function introduction

This command queries the RMU’s current status. When RMU correctly received the command, it will return the Power Amplifier’s status. User can use this command conducts two operation:

1) When operating the RMU, user can use this command to check if the Power Amplifier has been set on and conduct the related operation.

2) User can use this command to check if the RMU has been connected correctly. If the RMU has been connected, then RMU will response within the certain time. Otherwise it means that the RMU is not reachable.

3.1.2 Command Format

Table 3-1-1 Command format (query status)

Field	SOF	LEN	CMD	*CRC	EOF
Length(bit)	1	1	1	2	1

Table 3-1-2 Response format of query status

Field	SOF	LEN	CMD	STATUS	*CRC	EOF
Length(bit)	1	1	1	1	2	1

3.1.3 Return status definition

Table 3-1-3 STATUS

Bits	Bit 7 ~ 4	Bit 3 ~ 1	Bit 0
Function	Common bits	Reserved	1 = the Power Amplifier has been set on 0 = the Power Amplifier has been set off

Notes: Bit0 is valid only when Bit 7 has been set 0.

3.1.4 Related API Function

Table 3-1-4 Related API Function

Function name	Description
RmuOpenAndConnect()	Open COM terminal and connected with RMU
RmuGetPaStatus()	Query Power Amplifier status

3.1.5 命令示例

Send Commands(hex)	Returned Value(hex)
aa 02 00 55	Success:aa 03 00 00 55
	Fail:aa 03 00 80 55

3.2. Get Power Settings

3.2.1 Command introduction

This command reads the RMU’s power settings. Before conduct any other operation, user can use this command reads out RMU’s power settings. This command has two kinds of response format: Success (Table3-2-2) and fail (Table3-2-3).

3.2.2 Command format

Table 3-2-1 Command Format

Field	SOF	LEN	CMD	*CRC	EOF
Length(bit)	1	1	1	2	1

Table 3-2-2 Response format (Success)

Field	SOF	LEN	CMD	STATUS	POWER	*CRC	EOF
Length(bit)	1	1	1	1	1	2	1

Table 3-2-3 Response format (Fail)

Field	SOF	LEN	CMD	STATUS	*CRC	EOF
Length(bit)	1	1	1	1	2	1

Table 3-2-4 POWERField format

POWER	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Description	1	Output Power (dBm)						

3.2.3 Return status definition

This Command only supports common status bits.

3.2.4 Related API Function

Table 3-2-5 Related API Function

Functions Name	Description
RmuGetPower()	Read the RMU’s power settings

3.2.5 Command example

Send Commands(hex)	Returned Value(hex)
aa 02 01 55	Success:aa 04 01 00 9a 55
	Fail:aa 03 01 80 55

3.3. Setting Power

3.3.1 Command introduction

This command sets the output power of the power amplifier. Users should use this command to set the output power before conducting any operation with the tag. If user doesn't set the output power of the RMU, the default settings will be used.

3.3.2 Command format

Table 3-3-1 Command format

Field	SOF	LEN	CMD	OPTION	POWER	*CRC	EOF
Length(bit)	1	1	1	1	1	2	1

Table 3-3-2 Response format

Field	SOF	LEN	CMD	STATUS	*CRC	EOF
Length(bit)	1	1	1	1	2	1

Table 3-3-3 OPTIONField format

OPTION	Bit 7 ~ 1	Bit 0
Description	Reserved	Output Power settings
Function	Reserved	1: POWER's Bit6~0 valid 0: POWER's Bit6~0 non-valid

Notes: POWER Field definition is in Table3-2-3.

3.3.3 Return status definition

This Command only supports general status bits.

3.3.4 Related API Function

Table 3-3-4 Related API Function

Functions Name	Description
RmuSetPower()	sets RMU 's output power

3.3.5 Command example

Send Commands(hex)	Returned Value(hex)
aa 02 02 03 9a 55	Success:aa 03 02 00 55
	Fail:aa 03 02 80 55

3.4. Read Frequency Settings

(TBD)

The function is not available now.

3.5. Setting the Frequency

(TBD)

The function is not available now. The RMU Frequency has been setted by the MCU inside.

3.6. Tag Inventory (Single Tag Inventory)

3.6.1 Command introduction

This command starts the inventory round of the tag. The command should only be used to inventory the single tag. The command has two kinds of response format. The RMU returns the Inventory Tag Response (Table3-6-2) when received this command identifying if the inventory round has been started successfully or not. After the inventory round has been started successfully, the RMU will return the Tags ID Response (Table3-6-3), until to the inventory round is stopped. Each Tags ID Response only contains one tag's UII.

3.6.2 Command format

Table 3-6-1 Command format (Single Tag Inventory)

Field	SOF	LEN	CMD	*CRC	EOF
Length(bit)	1	1	1	2	1

Table 3-6-2 Inventory Tag Response format

Field	SOF	LEN	CMD	STATUS	*CRC	EOF
Length(bit)	1	1	1	1	2	1

Table 3-6-3 Get Tag ID Response format

Field	SOF	LEN	CMD	STATUS	UII	*CRC	EOF
Length(bit)	1	1	1	1		2	1

Notes: UII used in this document includes PC bits. That is PC+UII. UII's format is in Appendix A.

3.6.3 Return status definition

Table 3-6-4 STATUS

Bits	Bit 7 ~ 4	Bit 3 ~ 1	Bit 0
Function	Common Bits	Reserved	1 =Response of query tags (NOT include UII) 0 = Response of acquire tags ID(include UII)

Notes: STATUS Bit0 is valid only when Bit 7 is 0.

3.6.4 Related API Function

Table 3-6-5 Related API Function

Functions Name	Description
RmuInventory()	Acquire UII of the tag

3.6.5 Command example

Send Commands(hex)	Returned Value(hex)
aa 02 10 55	Success:aa 05 10 00 08 00 00 01 55
	Fail:aa 03 10 01 55

3.7. Tag Inventory (ANTI-Collision Inventory)

3.7.1 Command introduction

This command will make the Reader into inventory tags cycling. Users should use this command when inventorying more than one tag. When sending this command to inventory tags, users must select the Q value. If Q is 0, the Reader will use default Q value. This command response modeis same with inventory tag for single command.

3.7.2 Command format

Table 3-7-1 Command format (ANTI-Collision Inventory)

Field	SOF	LEN	CMD	Q	*CRC	EOF
Length(bits)	1	1	1	1	2	1

Table 3-7-2 Command format of Q

Q	Bit 7 ~ Bit 4	Bit 3 ~ Bit 0
Description	Reserved	Q value

Table 3-7-3 Command format of Reader return

Field	SOF	LEN	CMD	STATUS	*CRC	EOF
Length(bits)	1	1	1	1	2	1

Table 3-7-4 Get Tag ID Response format

Field	SOF	LEN	CMD	STATUS	UII	*CRC	EOF
Length(bits)	1	1	1	1		2	1

3.7.3 Return status definition

Table 3-7-5 STATUS

Bit	Bit 7 ~ 4	Bit 3 ~ 1	Bit 0
Function	Common Bits	Reserved	1 =Response of query tags (NOT includeUII) 0 =Response of acquire tags ID(include UII)

Notes: STATUS Bit0 is valid only when Bit 7 is 0.

3.7.4 Related API Function

Table 3-7-6 Related API Function

Function name	Description
RmuInventory()	Acquire UII of the tag(ANTI-Collision Inventory)

3.7.5 Command example

Send Commands(hex)	Returned Value(hex)
aa 02 11 55	Success:aa 05 11 00 08 00 00 01 55
	Fail:aa 03 11 01 55

3.8. Tag Inventory (single inventory round)

3.8.1 Command introduction

This command will let the system to inventory tags in single inventory round.This command could inventory more than one tag.The differece from Identify Tags(ANTI-Collision) is when the Reader identify tags succeed, it will have a return,when identify tags failed, the Reader will not have a return.

3.8.2 Command format

Tabel 3-8-1 Command format (Single Tag Inventory)

Field	SOF	LEN	CMD	*CRC	EOF
Length(bits)	1	1	1	2	1

Table 3-8-2 Inventory Tag Response format

Field	SOF	LEN	CMD	STATUS	UII	*CRC	EOF
Length(bits)	1	1	1	1		2	1

3.8.3 Return status definition

This command only supports common STATUS.

3.8.4 Related API Function

Table 3-8-3 Related API Function

Function name	Description
RmuInventorySingle()	Acquire UII of the tag

3.8.5 Command example

Send Commands(hex)	Returned Value(hex)
aa 02 18 55	Success:aa 05 18 00 08 00 55
	Fail:aa 03 18 01 55

3.9. Tag Inventory (with wiegand output)

3.9.1 Command introduction

Users use this command to inventory one tag with Wiegand output.The command is complied with Wiegand-26 protocol.This command will not start ANTI-Collision function.

[Note]: When using Wiegand Port to transmit data, there must be a +12V external power which between +12V pin and GND pin.

When inventory tag succeeded, the Reader will have a return. When inventory tag failed, the Reader will not have a return.

3.9.2 Command format

Table 3-9-1 Command format (with WIEGAND output)

Field	SOF	LEN	CMD	*CRC	EOF
Length(bits)	1	1	1	2	1

Table 3-9-2 Inventory Tag Response format

Field	SOF	LEN	CMD	STATUS	*CRC	EOF
Length(bits)	1	1	1	1	2	1

Table 3-9-3 Get Tag ID Response format

Field	SOF	LEN	CMD	STATUS	UII	*CRC	EOF
Length(bits)	1	1	1	1		2	1

3.9.3 Return status definition

Table 3-9-4 STATUS

Bits	Bit 7 ~ 4	Bit 3 ~ 1	Bit 0
Function	Common bits	Reserved	1 =Response of query tags (NOT includeUII) 0 =Response of acquire tags ID(include UII)

Notes: STATUS Bit0 is valid only when Bit 7 is 0.

3.9.4 Related API Function

Table 3-9-5 Related API Function

Function name	Description
RmuWiegandInventory ()	Acquire UII of the tag (with Wiegand output)

3.9.5 Command example

Send Commands(hex)	Returned Value(hex)
aa 02 19 55	Success:aa 05 19 00 08 00 55
	Fail:aa 03 19 01 55

3.10. Stop inventory command

3.10.1 Command introduction

This command is used to stop inventorying tags.

3.10.2 Command format

Table 3-10-1 Command format of Stop inventorying command

Field	SOF	LEN	CMD	*CRC	EOF
Length(bits)	1	1	1	2	1

Table 3-10-2 Response format

Field	SOF	LEN	CMD	STATUS	*CRC	EOF
Length(bits)	1	1	1	1	2	1

3.10.3 Return status definition

This command only supports common STATUS.

3.10.4 Related API Function

Table 3-10-3 Related API Function

Function name	Description
RmuStopGet()	Stop inventorying tags.

3.10.5 Command example

Send Commands(hex)	Returned Value(hex)
aa 02 12 55	Success:aa 03 12 00 55
	Fail:aa 03 12 80 55

3.11. Read the tag data (Include UII mode)

3.11.1 Command introduction

This command is used to read data from tags.When using this command to get tag’s data, you must get the tag’s ID firstly. This command has two different data formats between success and Fail (Table 3-11-1 and Table 3-11-2) when reading data.

There are ACCESS password (APWD Field) in the Data format of Read, Write, Erase and Lock commands. When the APWD is not all 0, The Reader must use ACCESS command to ensure tags in the SECURED state firstly.

The Reader will be possible for returning Error Code when operating tag’s data (Read, Write, Erase, Lock and Kill). AT the same time, the response data format includes 1 Byte of error codes (Optional). The Definitions of Error codes are in Appendix C.

3.11.2 Command format

Table 3-11-1 Command format

Field	SOF	LEN	CMD	APWD	BANK	PTR	CNT	UII	*CRC	EOF
Length(bits)	1	1	1	4	1	EBV	1		2	1

Note 1: APWD Field means ACCESS PASSWORD.

Note 2: PTR Field is EBV format.The Command format of EBV are at Appendix B.

Note 3: CNTFieldis is the WORD (2 bytes) for the length of data read.

Table 3-11-2 Read Tag Data Response format (Success)

Field	SOF	LEN	CMD	STATUS	DATA	*CRC	EOF
Length(bits)	1	1	1	1	CNT*2	2	1

Table 3-11-3 Read Tag Data Response format (Fail)

Field	SOF	LEN	CMD	STATUS	*ECODE	*CRC	EOF
Length(bits)	1	1	1	1	1	2	1

Note: ECODE (Error Code) Field is Optional.

3.11.3 Return status definition

Table 3-11-4 STATUS

Bits	Bit 7 ~ 4	Bit 3 ~ 1	Bit 0
Function	Common Bits	Reserved	1= Response of query tags (include ECODE Field) 0= Response of query tags (not include ECODEField)

Note: The Bit0 of Status will be valid when the Bit7 is 0.

3.11.4 Related API Function

Table 3-11-5 Related API Function (Include UII mode)

Function name	Description
RmuReadData()	Read tag data

3.11.5 Command example

Send Commands(hex)	Returned Value(hex)
aa 13 13 00 00 00 00 01 01 01 08 00 00 01 55	Success:aa 05 13 00 08 00 55
	Fail:aa 04 13 81 04 55

3.12. Read the tag data (not Include UII mode)

3.12.1 Command introduction

This command is used to read data from tags. The command does not need to get the tag’s ID before using this command to read tag data.

There are ACCESS password (APWD Field) in the Data format of Read, Write, Erase and Lock commands. When the APWD is not all 0, the reader must use ACCESS command to ensure tags in the SECURED state firstly.

The Reader will be possible for returning Error Code when operating tag’s data (Read, Write, Erase, Lock and Kill). AT the same time, the response data format includes 1 Byte of error codes (Optional). The Definitions of Error codes are in Appendix C.

3.12.2 Command format

Table 3-12-1 Command format of Read the tag data (No UII mode)

Field	SOF	LEN	CMD	APWD	BANK	PTR	CNT	*CRC	EOF
Length(bits)	1	1	1	4	1	EBV	1	2	1

Note 1: APWDField means ACCESS PASSWORD.

Note 2: PTRField is EBV format.The Data format of EBV are at Appendix B.

Note 3: CNTFieldis is the WORD (2 bytes) for the length of data read.

Table 3-12-2 Read Tag Data Response format (Succeed)

Field	SOF	LEN	CMD	STATUS	DATA	UII	*CRC	EOF
Length(bits)	1	1	1	1	CNT*2		2	1

Table 3-12-3 Read Tag Data Response format (Fail)

Field	SOF	LEN	CMD	STATUS	*ECODE	*CRC	EOF
Length(bits)	1	1	1	1	1	2	1

Note: ECODE (Error Code) Field is Optional.

3.12.3 Return status definition

Table 3-12-4 STATUS

Bits	Bit 7 ~ 4	Bit 3 ~ 1	Bit 0
Function	Common bits	Reserved	1= Response of query tags (include ECODE Field) 0= Response of query tags (not include ECODE Field)

Note: The Bit0 of Status will be valid when the Bit7 is 0.

3.12.4 Related API Function

Table 3-12-5 Related API Function (No UII mode)

Function name	Description
RmuReadDataSingle()	Read tag data(No UII mode)

3.12.5 Command example

Send Commands(hex)	Returned Value(hex)
aa 09 20 00 00 00 00 01 01 01 55	Success:aa 09 20 00 08 00 08 00 00 01 55

	Fail:aa 04 20 81 04 55

3.13. Write the tag data (Include UII mode)

3.13.1 Command introduction

This command is used to write data to tags. In this mode, The Write command format must include tag’s UII before writing data to tags.

3.13.2 Command format

Table 3-13-1 Command format of Write the tag data (Include UII)

Field	SOF	LEN	CMD	APWD	BANK	PTR	CNT	DATA	UII	*CRC	EOF
Length(bits)	1	1	1	4	1	EBV	1	CNT*2		2	1

Note: CNT Field is the WORD (2 bytes) for the length of data write. Only CNT1 is supported now.

Table 3-13-2 Response format (Include UII)

Field	SOF	LEN	CMD	STATUS	*ECODE	*CRC	EOF
Length(bits)	1	1	1	1	1	2	1

3.13.3 Return status definition

Table 3-13-3 STATUS

Bits	Bit 7 ~ 4	Bit 3 ~ 1	Bit 0
Function	Common bits	Reserved	1= Response of query tags (include ECODE Field) 0= Response of query tags (not include ECODE Field)

Note:The Bit0 of Status will be valid when the Bit7 is 0.

3.13.4 Related API Function

Table 3-13-4 Related API Function (Include UII)

Function name	Description
RmuWriteData()	Write data to tags

3.13.5 Command example

Send Commands(hex)	Returned Value(hex)
aa 15 14 00 00 00 00 01 01 01 10 00 08 00 00 01 55	Success:aa 03 14 00 55
	Fail:aa 04 14 81 04 55

3.14. Write the tag data (not Include UII mode)

3.14.1 Command introduction

This command is used to write data to tags. In this mode, The Write command format does not need to include tag’s UII before writing data to tags.

3.14.2 Command format

Table 3-14-1 Command format of Write the tag data (No UII mode)

Field	SOF	LEN	CMD	APWD	BANK	PTR	CNT	DATA	*CRC	EOF
Length(bits)	1	1	1	4	1	EBV	1	CNT*2	2	1

Note: CNTField is the WORD (2 bytes) for the length of data write. Only CNT1 is supported now.

Table 3-14-2 Response format (No UII mode)

Field	SOF	LEN	CMD	STATUS	UII	*ECODE	*CRC	EOF
Length(bits)	1	1	1	1		1	2	1

3.14.3 Return status definition

Table 3-14-3 STATUS

Bits	Bit 7 ~ 4	Bit 3 ~ 1	Bit 0
Function	Common bits	Reserved	1= Response of query tags (include ECODE Field) 0= Response of query tags (not include ECODE Field)

Notes: STATUS Bit0 is valid only when Bit 7 is 0.

3.14.4 Related API Function

Table 3-14-4 Related API Function

Function name	Description
RmuWriteDataSingle()	Write data to tags(No UII mode)

3.14.5 Command example

Send Commands(hex)	Returned Value(hex)
aa 11 21 00 00 00 00 01 01 01 10 00 55	Success:aa 07 21 00 08 00 00 01 55
	Fail:aa 04 21 81 04 55

3.15. Erase Tag Data

3.15.1 Command introduction

Erase the specified data segment in the specified tag. This command can be used to erase data in the tag after the user access to the tag ID. It is only effective to the tags that support the command ‘BlockErase’.

3.15.2 Command format

Table 3-15-1 Command format (Erase Tag Data)

Field	SOF	LEN	CMD	APWD	BANK	PTR	CNT	UII	*CRC	EOF
Length(bit)	1	1	1	4	1	EBV	1		2	1

Note: CNT is the Field that needs to be erased in the unit of word.

Table 3-15-2 Response format of Erase Tag Data

Field	SOF	LEN	CMD	STATUS	*ECODE	*CRC	EOF
Length(bit)	1	1	1	1	1	2	1

3.15.3 Return status definition

Table 3-15-3 Erase Tag data status

Bits	Bit 7 ~ 4	Bit 3 ~ 1	Bit 0
Function	Common Bits	Reserved	1 = ECODE data segment included in the response 0 = ECODE data segment not included in the response

Notes: STATUS Bit0 is valid only when Bit 7 is 0.

3.15.4 Related API functions

Table 3-15-4 Related API Function

Function	Description
RmuEraseData()	Erase tag data

3.15.5 Command example

Send Commands(hex)	Returned Value(hex)
aa 13 15 00 00 00 00 11 01 01 08 00 00 01 55	Success:aa 03 15 00 55
	Fail:aa 04 15 81 04 55

3.16. Lock Tag

3.16.1 Command introduction

Lock the specific data segments of the tag. Lock operation can be applied with this command after getting the tag ID.

3.16.2 Command format

Table 3-16-1 Command format (Lock Tag)

Field	SOF	LEN	CMD	APWD	LOCKDATA	UII	*CRC	EOF
Length(bit)	1	1	1	4	3		2	1

Note: The high-four-bit of data segment LOCKDATA are reserved bits, and the low-twenty-bit is Lock-Command Payload. Refer to appendix D for details.

Table 3-16-2 Response format of Lock Tag

Field	SOF	LEN	CMD	STATUS	*ECODE	*CRC	EOF
Length(bit)	1	1	1	1	1	2	1

3.16.3 Return status definition

Table 3-16-3 STATUS

Bits	Bit 7 ~ 4	Bit 3 ~ 1	Bit 0
Function	Common Bits	Reserved	1 = ECODE data segment included in the response 0 = ECODE data segment not included in the response

Note: STATUS Bit0 only effective when Bit 7=1.

3.16.4 Related API functions

Table 3-16-4 Related API Function

Function	Description
RmuLockMem()	Lock tag

3.16.5 Command example

Send Commands(hex)	Returned Value(hex)
aa 13 16 00 00 00 00 10 04 08 00 00 01 55	Success:aa 03 16 00 55
	Fail:aa 04 16 81 04 55

3.17. Destroy Tag

3.17.1 Command introduction

Destroy the specific tags. This command can be used to destroy the specific tag after getting the tag ID.

3.17.2 Command format

Table 3-17-1 Command format (Destroy Tag)

Field	SOF	LEN	CMD	KILLPWD	UII	*CRC	EOF
Length(bit)	1	1	1	4		2	1

Note: Data segment KILLPWD is 4-byte Kill Password.

Table 3-17-2 Response format of Destroy Tag

Field	SOF	LEN	CMD	STATUS	*ECODE	*CRC	EOF
Length(bit)	1	1	1	1	1	2	1

3.17.3 Return status definition

Table 3-17-3 STATUS

Bits	Bit 7 ~ 4	Bit 3 ~ 1	Bit 0
Function	Common Bits	Reserved	1 = ECODE data segment included in the response. 0 = ECODE data segment not included in the response.

Notes: STATUS Bit0 is valid only when Bit 7 is 0.

3.17.4 Related API functions

Table 3-17-4 Related API Function

Function	Description
RmuKillTag()	Destroy tags

3.17.5 Command Example

Send Commands(hex)	Returned Value(hex)
aa 10 17 00 00 00 00 08 00 00 01 55	Success:aa 03 17 00 55
	Fail:aa 04 17 81 04 55

3.18. Get RMU Information

3.18.1 Command introduction

Get the serial number of the hardware and software version. RMU hardware serial number is a 6-byte hexadecimal number and software version number is a byte. The first four-bit of the software version byte is the main version number, and the latter four-bit is the secondary version number.

3.18.2 Command format

Table 3-18-1 Command format (Read RMU Information)

Field	SOF	LEN	CMD	*CRC	EOF
Length(bit)	1	1	1	2	1

Table 3-18-2 Response format (Read RMU Information)

Field	SOF	LEN	CMD	STATUS	SERIAL	VERSION	*CRC	EOF
Length(bit)	1	1	1	1	6	1	2	1

3.18.3 Return status definition

Table 3-18-3 STATUS

Bits	Bit 7 ~ 4	Bit 3 ~ 1	Bit 0
Function	Common Bits	Reserved	1 = ECODE data segment included in the response. 0 = ECODE data segment not included in the response

Notes: STATUS Bit0 is valid only when Bit 7 is 0.

3.18.4 Related API functions

Table 3-18-4 Related API functions

Function	Description
RmuGetVersion()	Read RMU Information

3.18.5 Command Example

Send Commands(hex)	Returned Value(hex)
aa 02 07 55	Success:aa 0a 07 01 ff ff ff ff ff ff ff ff ff ff ff ff ff 55

	Fail:aa 03 07 81 55

4. API Definition

4.1 RmuOpenAndConnect ()

4.1.1 Function description

This function opens the COM port and connects with RMU.

4.1.2 Prototype

int WINAPI RmuOpenAndConnect (HANDLE &hCom, char* cPort, UCHAR flagCrc);

4.1.3 Return value

1: Open COM and connect to RMU succeed.

Other: fail to open COM port or connect to RMU.

4.1.4 Function parameters

HANDLE &hCom: The handle of COM port, init as NULL.

char* cPort:The COM port, e.g. COM1, COM2.

UCHAR flagCrc: If using the CRC16 verification. 1: use CRC16; 0: no CRC16.

4.2 RmuCloseAndDisconnect ()

4.2.1 Function description

This function closes RMU PA and disconnects from RMU.

4.2.2 Prototype

int WINAPI RmuCloseAndDisconnect (HANDLE &hCom, UCHAR flagCrc);

4.2.3 Return value

1: Close RMU PA and disconnect from RMU succeed.

Other: Fail to close RMU PA or disconnect from RMU.

4.2.4 Function parameters

HANDLE &hCom: The handle of COM port.

UCHAR flagCrc: If using the CRC16 verification. 1: use CRC16; 0: no CRC16.

4.3 RmuGetPaStatus ()

4.3.1 Function description

This function queries RMU PA status.

4.3.2 Prototype

int WINAPI RmuGetPaStatus (HANDLE hCom, UCHAR* uStatus, UCHAR flagCrc);

4.3.3 Return value

1: Succeed to Query RMU PA status.

Other: Fail to query RMU PA status.

4.3.4 Function parameters

HANDLE hCom: the handle of COM port

UCHAR* uStatus: RMU PA status: 1: PA is opened; 0: PA is closed.

UCHAR flagCrc: If using the CRC16 verification. 1: use CRC16; 0: no CRC16.

4.4 RmuGetPower ()

4.4.1 Function description

This function gets RMU power setting.

4.4.2 Prototype

int WINAPI RmuGetPower (HANDLE hCom, UCHAR* uPower, UCHAR flagCrc);

4.4.3 Return value

1: Succeed to get RMU power setting.

Other: Fail to get RMU power setting.

4.4.4 Function parameters

HANDLE hCom: the handle of COM port.

UCHAR* uPower: The POWER byte returned by RMU.

UCHAR flagCrc: If using the CRC16 verification. 1: use CRC16; 0: no CRC16.

4.5 RmuSetPower ()

4.5.1 Function description

This function sets RMU power.

4.5.2 Prototype

int WINAPI RmuSetPower (HANDLE hCom, UCHAR uOption, UCHAR uPower, UCHAR flagCrc);

4.5.3 Return value

1: Set RMU power succeeds.

Other: Fail to set RMU power.

4.5.4 Function parameters

HANDLE hCom: The handle of COM port.

UCHAR uOption: The OPTION field in the command of RMU_SET_POWER.

UCHAR uPower: The POWER field in the command of RMU_SET_POWER.

UCHAR flagCrc: If using the CRC16 verification. 1: use CRC16; 0: no CRC16.

4.6 RmuGetFrequency ()

4.6.1 Function description

This function gets RMU frequency setting.

4.6.2 Prototype

int WINAPI RmuGetFrequency (HANDLE hCom, UCHAR* uBaseFre, UCHAR* uChannNum, UCHAR* uChannSpc, UCHAR flagCrc);

4.6.3 Return value

1: Get RMU frequency setting succeeds.

Other: Fail to get RMU frequency setting.

4.6.4 Function parameters

HANDLE hCom: The handle of COM port.

UCHAR* uBaseFre: The BF (Base Frequency) filed returned by RMU (2 bytes).

UCHAR* uChannNum: The CN (Channel) filed returned by RMU (1 byte).

UCHAR* uChannSpc: The SPC (Space) filed returned by RMU (1 byte).

UCHAR flagCrc: If using the CRC16 verification. 1: use CRC16; 0: no CRC16.

4.7 RmuSetFrequency ()

4.7.1 Function description

This function set RMU frequency.

4.7.2 Prototype

int WINAPI RmuSetFrequency (HANDLE hCom, UCHAR* uBaseFre, UCHAR uChannNum, UCHAR uChannSpc, UCHAR flagCrc);

4.7.3 Return value

1: Set RMU frequency succeeds.

Other: Fail to set RMU frequency.

4.7.4 Function parameters

HANDLE hCom: The handle of COM port.

UCHAR* uBaseFre: The BF (Base Frequency) filed in the command of RMU_SET_FRE (2 bytes).

UCHAR* uChannNum: The CN (Channel) filed in the command of RMU_SET_FRE (1 byte).

UCHAR* uChannSpc: The SPC (Space) filed in the command of RMU_SET_FRE (1 byte).

UCHAR flagCrc: If using the CRC16 verification. 1: use CRC16; 0: no CRC16.

4.8 RmuInventory ()

4.8.1 Function description

This function starts the RMU inventory loop.

4.8.2 Prototype

int WINAPI RmuInventory (HANDLE hCom, UCHAR flagAnti, UCHAR initQ, UCHAR flagCrc);

4.8.3 Return value

1: Starting RMU inventory loop succeeds.

Other: Fail to start RMU inventory loop.

4.8.4 Function parameters

HANDLE hCom: The handle of COM port.

UCHAR flagAnti: if using anti-collision inventory. 1: use anti-collision; 0: no anti-collision.

UCHAR initQ: The initial value of Q, valid when flagAnti set to 1.

UCHAR flagCrc: If using the CRC16 verification. 1: use CRC16; 0: no CRC16.

4.9 RmuInventorySingle ()

4.9.1 Function description

This function runs single inventory operation.

4.9.2 Prototype

int WINAPI RmuInventorySingle (HANDLE hCom, UCHAR* uLenUii, UCHAR* uUii , UCHAR flagCrc);

4.9.3 Return value

1: Get a tag UII succeed.

Other: Fail to get a tag UII.

4.9.4 Function parameters

HANDLE hCom: The handle of COM port.

UCHAR* uLenUii: The length of UII, 1byte.

UCHAR* uUii: Tag UII.

UCHAR flagCrc: If using the CRC16 verification. 1: use CRC16; 0: no CRC16.

4.10 RmuWiegandInventory ()

4.10.1 Function description

This function start RMU inventory loop with Wiegand communication enabled.

4.10.2 Prototype

int WINAPI **RmuWiegandInventory** (HANDLE hCom, UCHAR flagAnti, UCHAR initQ, UCHAR flagCrc);

4.10.3 Return value

1: Start RMU inventory loop succeed.

Other: Fail to start RMU inventory loop.

4.10.4 Function parameters

HANDLE hCom: The handle of COM port.

UCHAR flagAnti: if using anti-collision inventory. 1: use anti-collision; 0: no anti-collision.

UCHAR initQ: The initial value of Q, valid when flagAnti set to 1.

UCHAR flagCrc: If using the CRC16 verification. 1: use CRC16; 0: no CRC16.

4.11 RmuGetReceived ()

4.11.1 Function description

This function gets a tag UII inventoried by RMU.

4.11.2 Prototype

int WINAPI RmuGetReceived (HANDLE hCom, UCHAR* uLenUii, UCHAR* uUii);

4.11.3 Return value

1: Get a tag UII succeed.

Other: Fail to get a tag UII.

4.11.4 Function parameters

HANDLE hCom: The handle of COM port.

UCHAR* uLenUii: The length of UII, 1byte.

UCHAR* uUii: Tag UII.

4.12 RmuStopGet ()

4.12.1 Function description

This function stops RMU inventory loop.

4.12.2 Prototype

int WINAPI RmuStopGet (HANDLE hCom, UCHAR flagCrc);

4.12.3 Return value

1: Stop RMU inventory loop succeed.

Other: Fail to stop RMU inventory loop.

4.12.4 Function parameters

HANDLE hCom: The handle of COM port.

UCHAR flagCrc: If using the CRC16 verification. 1: use CRC16; 0: no CRC16.

4.13 RmuReadData ()

4.13.1 Function description

This function reads data from tag memory.

4.13.2 Prototype

int WINAPI RmuReadData (HANDLE hCom, UCHAR* uAccessPwd, UCHAR uBank, UCHAR* uPtr, UCHAR uCnt, UCHAR* uUii, UCHAR* uReadData, UCHAR* uErrorCode, UCHAR flagCrc);

4.13.3 Return value

1: Read data from tag succeed.

Other: Fail to read data from tag.

4.13.4 Function parameters

HANDLE hCom: The handle of COM port.

UCHAR* uAccessPwd: The Access Password of tag.

UCHAR uBank: The memory bank.

UCHAR* uPtr: The pointer of start address.

UCHAR uCnt: The number of words (2 bytes) to read.

UCHAR* uUii: Tag UII.

UCHAR* uReadData: The data read from tag (uCnt number of words).

UCHAR* uErrorCode: The Error Code returned from tag, valid when function returns 0 and uErrorCode NOT equal to 0xFF.

UCHAR flagCrc: If using the CRC16 verification. 1: use CRC16; 0: no CRC16.

4.14 RmuReadDataSingle ()

4.14.1 Function description

This function reads data from tag memory without specifying the tag UII.

4.14.2 Prototype

int WINAPI RmuReadData (HANDLE hCom, UCHAR* uAccessPwd, UCHAR uBank, UCHAR* uPtr, UCHAR uCnt, UCHAR* uReadData, UCHAR* uUii, UCHAR* uLenUii, UCHAR* uErrorCode, UCHAR flagCrc);

4.14.3 Return value

1: Read data from tag succeed.

Other: Fail to read data from tag.

4.14.4 Function parameters

HANDLE hCom: The handle of COM port.

UCHAR* uAccessPwd: The Access Password of tag.

UCHAR uBank: The memory bank.

UCHAR* uPtr: The pointer of start address.

UCHAR uCnt: The number of words (2 bytes) to read.

UCHAR* uUii: Tag UII returned.

UCHAR* uLenUii: The length of tag UII returned.

UCHAR* uReadData: The data read from tag (uCnt number of words).

UCHAR* uErrorCode: The Error Code returned from tag, valid when function returns 0 and uErrorCode NOT equal to 0xFF.

UCHAR flagCrc: If using the CRC16 verification. 1: use CRC16; 0: no CRC16.

4.15 RmuWriteData ()

4.15.1 Function description

This function writes data into tag memory.

4.15.2 Prototype

int WINAPI RmuWriteData (HANDLE hCom, UCHAR* uAccessPwd, UCHAR uBank, UCHAR* uPtr, UCHAR uCnt, UCHAR* uUii, UCHAR* uWriteData, UCHAR* uErrorCode, UCHAR flagCrc);

4.15.3 Return value

1: Write data into tag memory succeed.

Other: Fail to write data into tag memory.

4.15.4 Function parameters

HANDLE hCom: The handle of COM port.

UCHAR* uAccessPwd: The Access Password of tag.

UCHAR uBank: The memory bank.

UCHAR* uPtr: The pointer of start address.

UCHAR uCnt: The number of words to write.

UCHAR* uUii: Tag UII.

UCHAR* uWriteData: The data to write.

UCHAR* uErrorCode: The Error Code returned from tag, valid when function returns 0 and uErrorCode NOT equal to 0xFF.

UCHAR flagCrc: If using the CRC16 verification. 1: use CRC16; 0: no CRC16.

4.16 RmuWriteDataSingle ()

4.16.1 Function description

This function writes data into tag memory without specifying tag UII.

4.16.2 Prototype

int WINAPI RmuWriteData (HANDLE hCom, UCHAR* uAccessPwd, UCHAR uBank, UCHAR* uPtr, UCHAR uCnt, UCHAR* uWriteData, UCHAR* uUii, UCHAR* uLenUii, UCHAR* uErrorCode, UCHAR flagCrc);

4.16.3 Return value

1: Write data into tag memory succeed.

Other: Fail to write data into tag memory.

4.16.4 Function parameters

HANDLE hCom: The handle of COM port.

UCHAR* uAccessPwd: The Access Password of tag.

UCHAR uBank: The memory bank.

UCHAR* uPtr: The pointer of start address.

UCHAR uCnt: The number of words to write.

UCHAR* uWriteData: The data to write.

UCHAR* uUii: The tag UII returned.

UCHAR* uLenUii: Length of tag UII returned.

UCHAR* uErrorCode: The Error Code returned from tag, valid when function returns 0 and uErrorCode NOT equal to 0xFF.

UCHAR flagCrc: If using the CRC16 verification. 1: use CRC16; 0: no CRC16.

4.17 RmuEraseData ()

4.17.1 Function description

This function erases data in tag memory.

4.17.2 Prototype

int WINAPI RmuEraseData (HANDLE hCom, UCHAR* uAccessPwd, UCHAR uBank, UCHAR* uPtr, UCHAR uCnt, UCHAR* uUii, UCHAR* uErrorCode, UCHAR flagCrc);

4.17.3 Return value

1: Erase data in tag memory succeed.

Other: Fail to erase data in tag memory.

4.17.4 Function parameters

HANDLE hCom: The handle of COM port.

UCHAR* uAccessPwd: The Access Password of tag.

UCHAR uBank: The memory bank.

UCHAR* uPtr: The pointer of start address.

UCHAR uCnt: The number of words to erase.

UCHAR* uUii: Tag UII.

UCHAR* uErrorCode: The Error Code returned from tag, valid when function returns 0 and uErrorCode NOT equal to 0xFF.

UCHAR flagCrc: If using the CRC16 verification. 1: use CRC16; 0: no CRC16.

4.18 RmuLockMem ()

4.18.1 Function description

This function locks or unlocks tag memory bank.

4.18.2 Prototype

int WINAPI RmuLockMem (HANDLE hCom, UCHAR* uAccessPwd, UCHAR* uLockData, UCHAR* uUii, UCHAR* uErrorCode, UCHAR flagCrc);

4.18.3 Return value

1: Lock or unlock tag memory bank succeed.

Other: Fail to lock or unlock tag memory bank.

4.18.4 Function parameters

HANDLE hCom: The handle of COM port.

UCHAR* uAccessPwd: The Access Password of tag.

UCHAR* uLockData: The LOCKDATA field in the command of RMU_LOCK_MEM.

UCHAR* uUii: Tag UII.

UCHAR* uErrorCode: The Error Code returned from tag, valid when function returns 0 and uErrorCode NOT equal to 0xFF.

UCHAR flagCrc: If using the CRC16 verification. 1: use CRC16; 0: no CRC16.

4.19 RmuKillTag ()

4.19.1 Function description

This function kills a tag.

4.19.2 Prototype

int WINAPI RmuKillTag (HANDLE hCom, UCHAR* uKillPwd, UCHAR* uUii, UCHAR* uErrorCode, UCHAR flagCrc);

4.19.3 Return value

1: Kill a tag succeed.

Other: Fail to kill a tag.

4.19.4 Function parameters

HANDLE hCom: The handle of COM port.

UCHAR* uKillPwd: The Kill Password of tag.

UCHAR* uUii: Tag UII.

UCHAR* uErrorCode: The Error Code returned from tag, valid when function returns 0 and uErrorCode NOT equal to 0xFF.

UCHAR flagCrc: If using the CRC16 verification. 1: use CRC16; 0: no CRC16.

4.20 RmuGetVersion ()

4.20.1 Function description

This function gets hardware serial number and software version of RMU.

4.20.2 Prototype

int WINAPI RmuGetVersion (HANDLE hCom, UCHAR* uSerial, UCHAR* uVersion, UCHAR flagCrc);

4.20.3 Return value

1: Get hardware serial number and software version of RMU succeed.

Other: Fail to get hardware serial number and software version of RMU.

4.20.4 Function parameters

HANDLE hCom: The handle of COM port.

UCHAR* uSerial: The hardware serial number returned. (6 bytes)

UCHAR* uVersion: The software version returned. (1 byte)

UCHAR flagCrc: If using the CRC16 verification. 1: use CRC16; 0: no CRC16.

4.21 RmuGetAntenna ()

4.21.1 Function description

This function gets RMU antenna setting.

4.21.2 Prototype

int WINAPI RmuGetAntenna (HANDLE hCom, UCHAR* uAnt, UCHAR flagCrc);

4.21.3 Return value

1: Get RMU antenna setting succeeds.

Other: Fail to get RMU antenna setting.

4.21.4 Function parameters

HANDLE hCom: The handle of COM port.

UCHAR* uAnt: The ANT field returned by RMU.

UCHAR flagCrc: If using the CRC16 verification. 1: use CRC16; 0: no CRC16.

4.22 RmuSetAntenna ()

4.22.1 Function description

This function set RMU antenna.

4.22.2 Prototype

int WINAPI RmuSetAntenna (HANDLE hCom, UCHAR uAnt, UCHAR flagCrc);

4.22.3 Return value

1: Set RMU antenna succeeds.

Other: Fail to set RMU antenna.

4.22.4 Function parameters

HANDLE hCom: The handle of COM port.

UCHAR* uAnt: The ANT field in the command of RMU_SET_ANT.

UCHAR flagCrc: If using the CRC16 verification. 1: use CRC16; 0: no CRC16.

Appendix A: UII Format

The UII mentioned in this document contains PC bits. The first two bytes are PC (Protocol-control) bits, whose format is shown in table A-1.

Table A-1 PC bits Format

Bits 0 ~ 4	**Bits 5 ~ 6**	**Bits 7 ~ 15**
Total length of PC and UII in the unit of word (2 bytes)	Undefined	NSI(Unused)

Note: UII should transmit from the lower bit.

The five MSBs of PC indicates the total length of PC and UII.

For example:

PC bits 0~4 (bin)	PC+UII length (byte)
00000	2
00001	4
00010	6
…	…

Appendix B: Extensible bit Vectors (EBV)

EBV (Extensible Bit Vector) is a kind of data format which can denote the extensible data. The EBV mentioned in this document is arrays in the unit of byte. The highest bit of every byte in the array is the extensible bit. ‘0’ in the extensible bit means this byte is the last one, while ‘1’ in the extensible bit indicates there will be valid bytes in the subsequence. The valid data in EBV format are bits stream from left to right, neglecting the extensible bits.

RMU only supports EVB data of one or two bytes, and the format is as follows.

0	X X X X X X X		
1	X X X X X X X	0	X X X X X X X

The highest bit in each byte is extensible bit. If the denoted number is less than 128, EBV of one byte is sufficient. When the denoted number is more than 127 but less than 16384, it requires EBV have two bytes.

For example:

12: 0 0 0 0 1 1 0 0

130: 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0

Appendix C: Error Codes

When RMU operates with tags (reading data from tag, writing data to tag, erasing data, locking tag, or killing tag), if any error occurs in the tag, it will return tag error codes.

Table C-1 Tag error codes

Error Code Support	Error Code value (bin)	Error Code Name	Error Description
Error-specific	00000000	Other errors	Errors which are not defined in other error codes
	00000011	Memory space overflow or unsupported PC value	Pointed memory space does not exist or pointed PC value is not supported by the tag
	00000100	Memory space locked	Pointed memory space is locked, reading or writing operation is forbidden
	00001011	Power shortage	Can not write the tag due to power shortage
Non-specific	00001111	Unknown error	Tag does not support the Error-specific code

Appendix D: Lock-Command Payload

Lock-Command Payload is a 20-bit data whose higher 10 bits are Mask and lower 10 bits are Action. Table D-1 shows the format of Lock-Command Payload. When Mask is set to 1, the corresponding Action bits are effective. Table D-2 shows descriptions of Action bits.

Table D-1 Data format of Lock-Command Payload

Kill password		**Access password**		**UII memory**		**TID memory**		**User memory**	
19	18	17	16	15	14	13	12	11	10
Skip/ Write	Skip/ Write	Skip/ Write	Skip/ Write	Skip/ Write	Skip/ Write	Skip/ Write	Skip/ Write	Skip/ Write	Skip/ Write
9	8	7	6	5	4	3	2	1	0
Pwd read/ write	Perma lock	Pwd read/ write	Perma lock	Pwd write	Perma lock	Pwd write	Perma lock	Pwd write	Perma lock

Table D-2 Lock Action bits

Pwd-write	**Permalock**	**Description**
0	0	The corresponding data segment is writable when the state is OPEN or SECURED
0	1	The corresponding data segment is writable permanently when the state is OPEN or SECURED, and it can not be locked
1	0	The corresponding data segment is writable when the state is SECURED, but it is unwritable when the state is OPEN.
1	1	The corresponding data segment is unwritable in any state.
Pwd-read/write	**Permalock**	**Description**
0	0	The corresponding data segment is writable and readable when the state is OPEN or SECURED
0	1	The corresponding data segment is writable and readable permanently when the state is OPEN or SECURED, and it can not be locked
1	0	The corresponding data segment is writable and readable when the state is SECURED, but it is unwritable and unreadable when the state is OPEN.
1	1	Corresponding data segment is unwritable and unreadable in any state.

Appendix E: Examples for Inserting Bytes

In the real communication process, when 0xAA, or 0x55, or 0xFF exist between SOF and EOF in the sending packets, a 0xFF byte should be inserted before these bytes by the sender. When the data with inserted bytes is received, the acceptor should remove the inserted bytes and extract the valid data. The inserted bytes shall not be counted in LENGTH. For example:

Data packet needed to send (hex): AA 04 55 00 01 55
Data packet sent actually (hex): AA 04 FF 55 00 01 55
Data packet needed to send (hex): AA 05 00 00 01 AA 55
Data packet sent actually (hex): AA 05 00 00 01 FF AA 55
Data packet needed to send (hex): AA 06 00 00 01 AA FF 55
Data packet sent actually (hex): AA 06 00 00 01 FF AA FF FF 55