OSAL\_STM\_W

数据通讯协议

V0.1

**修订历史**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 更新时间 | 原因 | 作者 |
| V0.1 | 2015-05-3 | 创建文档 | WSF |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 1通讯功能

## 1.1数据通讯协议

### 1.1.1通讯数据帧格式

表2-1串口数据帧格式

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 起始符(2B) | 设备地址 | 功能单元 | 功能码 | 操作码 | 数据长度 | | 数据 | 校验和 |
| 0x55EE | 2B | 2B | 2B | 1B | 长度校验 | 数据长度 | NB | 1B |
| 4bit | 12bit |

表2-2 Tcp数据帧格式

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 起始符(NB) | 设备地址 | 功能单元 | 功能码 | 操作码 | 数据长度 | | 数据 | 校验和 |
| TCP头 | 2B | 2B | 2B | 1B | 长度校验 | 数据长度 | NB | 1B |
| 4bit | 12bit |

表中的各项均按HEX方式解释及传输。数据均按MSB模式发送，即在数据发送过程中高字节在前，底字节在后。

数据收发还需要注意数据大小端模式。

### 1.1.2消息帧注解

1）起始符

起始符为消息帧起始标志位，表示一个新的消息帧的开始，占用2个字节。起始符固定为55EE H。TCP数据帧起始符则为TCP头。

2）通讯ID

即设备通讯地址，表示同一通讯链路上的唯一设备通讯ID。不同于设备唯一ID号。

3）功能单元

表示数据操作功能单元，如设备授权、数据通讯握手、数据读写等功能

4）功能码

表示某一功能单元的详细操作功能。

表2-3 功能描述

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **功能单元** | | | **功能码** | | |
| **定义** | **值** | **描述** | **定义** | **值** | **描述** |
| UPDATE\_UNIT | 01H | 升级单元 | START | 01H | 开始升级 |
| READY | 02H | 升级准备完成 |
| UPDATING | 03H | 升级中 |
| RESULT | 04H | 单个升级结果反馈 |
| LAST\_RESULT | 05H | 反馈最终升级结果 |
| END | 06H | 升级完成 |
|  |  |  |
| DEBUG\_UNIT | 02H | 调试单元 | GET\_BOOT\_VERSION | 11H | 获取Bootloader版本号 |
| GET\_APP\_VERSION | 12H | 获取应用程序版本号 |
| GET\_CHIP\_ID | 13H | 获取芯片ID |
|  |  |  |
| ROUTER\_UNIT | 03H | 通讯单元 | COMM\_HANDLE | 21H | 通讯握手心跳包 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

5）操作模式

操作模式表示数据操作功能，指示数据的读写。如表2-4所示

表2-4 操作码描述

|  |  |  |
| --- | --- | --- |
| **操作名称** | **功能码** | **描述** |
| STATUS | 00H | 状态，数据返回状态功能码 |
| SET | 01H | 写操作，读取内部数据及状态等 |
| GET | 02H | 读操作，不做数据反馈 |
| SETGET | 03H | 写操作并返回结果，有反馈数据 |
| ERROR | FFH | 错误状态，操作错误返回 |

6）数据长度

数据长度校验：高4位，其值为低4位按位取反。

CheckLen = ((~Len) << 12) | (Len&0x0fff)。

数据长度：低12位表示用户数据长度。

7）数据

表示通讯操作数据。读操作表示读取的数据，写操作则表示写入的数据。

8）校验和

校验和为消息帧校验和字段之前除起始符以外的所有数据异或模256所得的值。校验和不包括起始符。

如消息帧是“55 EE 01 08 00 01 08”，则最后一个字节（校验和）的计算方式是：01 ^ 08 ^ 00 ^ 01 = 08H，即为消息帧校验和。

### 1.1.3操作状态STATUS(0x00)

表示系统当前所处的某一状态，用于主控制单元或监控终端查询系统状态的信息反馈。此操作无需应答。



### 1.1.4设置SET(0x01)

表示系统要执行的某一指定的操作，比如设置某些状态或执行某些控制操作等。此操作无需应答。



### 1.1.5读取GET(0x02)

用于主控制单元或监控终端查询系统状态的操作，如读取芯片ID、系统版本号等操作。操作正常时反馈STATUS的操作状态，出错时反馈ERROR操作状态。此操作需有应答。



### 1.1.6设置并读取SET\_GET(0x03)

用于主控制单元或监控终端设置系统状态或执行某一操作，并要求反馈操作的结果。如执行音频、通讯端口的切换等操作。操作正常时反馈STATUS的操作状态，出错时反馈ERROR操作状态。此操作需有应答。



### 1.1.7出错ERROR(0xFF)

当系统执行操作出错时，用于反馈出错的状态。如：

 **或者** 

### 1.1.8数据解析流程



## 1.2系统升级单元（UPDATE\_UNIT）

系统升级单元专门负责系统的程序升级工作，系统升级主要分为系统开始升级、系统升级前的准备、系统升级、升级结果反馈和升级完成几个阶段。如表2.1所示。

**表2.1 系统功能操作ID定义**

| **功能单元** | **功能单元ID** | **功能号定义** | **功能ID** | **描述** |
| --- | --- | --- | --- | --- |
| 升级单元 | UPDATE\_UNIT  0x01 | START | 0x0001 | 开始升级 |
| READY | 0x0002 | 升级准备 |
| UPDATEING | 0x0003 | 升级中 |
| RESULT | 0x0004 | 单个升级结果反馈 |
| LAST\_RESULT | 0x0005 | 最终升级结果反馈 |
| END | 0x0006 | 升级完成 |

### 1.2.1开始升级 START(0x0001)

程序的开始升级阶段要完成系统升级指令触发和升级文件信息的传送两个任务，**此过程由监控终端触发**。

由系统监控终端或者系统升级控制终端向结点发送一个**START**指令，并传送需要升级的升级文件个数，系统接收到**START**指令之后进入系统升级准备状态并记录升级文件信息。

**START**指令消息结构：

| **功能单元** | **功能ID** | **操作类型** | **数据长度** | **用户数据** |
| --- | --- | --- | --- | --- |
| UpdateUnit | Start | Set\_Get (Status) | 2 | 2B（升级文件总数） |

监控终端 🡪 升级结点

UpdateUnit. Start. Set\_Get. (xx xx)

监控终端 🡨 升级结点

UpdateUnit. Start. Status. (xx xx)

其中xx xx表示升级文件的总数，数据类型为UINT16

### 1.2.2升级准备 READY(0x0002)

此阶段为系统升级准备阶段，主要任务是传输当前升级文件的基本信息，如当前升级文件的序号、文件大小及文件CRC校验值等基本信息，**此过程由监控终端触发**。

**READY**指令消息结构：

| **功能单元** | **功能ID** | **操作类型** | **数据长度** | **用户数据** | | |
| --- | --- | --- | --- | --- | --- | --- |
| UpdateUnit | Ready | Set\_Get (Status) | 8 | **文件序号** | **文件大小** | **CRC16** |
| 2B | 4B | 2B |

监控终端 🡪 升级结点

UpdateUnit. Ready. Set\_Get. (xx xx xx )

监控终端 🡨 升级结点

UpdateUnit. Ready. Status. (xx xx xx)

其中xx xx表示文件的序号、文件大小及CRC16校验

### 1.2.3升级中 UPDATING(0x0003)

程序的升级过程，由监控终端向升级结点传送系统升级程序的数据，升级结点接收数据并完成程序的升级，**此过程由监控终端触发**。

**UPDATING**指令消息结构：

| **功能单元** | **功能ID** | **操作类型** | **数据长度** | **用户数据** | |
| --- | --- | --- | --- | --- | --- |
| UpdateUnit | Updating | Set\_Get | N+4 | **偏移量** | **程序升级数据** |
| 4B | NB |
| UpdateUnit | Updating | Status | 5 | **偏移量** | **程序写入结果** |
| 4B | 1B |

监控终端 🡪 升级结点

UpdateUnit. Updating. Set\_Get.N+4. (**Offset** xx xx)

监控终端 🡨 升级结点

UpdateUnit. Updating. Status.5. (**Offset** $$)

其中xx xx表示升级程序所需的数据， $$为程序写入的结果（TRUE表示写入成功，FALSE表示写入失败）。

若监控终端接收到的状态为FALSE，说明数据写入未成功需重发数据直到写入成功或者超过重发次数为止。如果超过重发次数则自动终止当前的程序升级过程并提示程序升级失败。

### 1.2.4单个升级结果反馈 RESULT(0x0004)

当升级结点接收完所有数据并且成功写入之后，自动计算接收数据的CRC16校验结果，触发Result并将计算结果反馈到监控终端。如果接收完成所有文件则自动触发LAST\_RESULT。

**RESULT**指令消息结构：

| **功能单元** | **功能ID** | **操作类型** | **数据长度** | **用户数据** | |
| --- | --- | --- | --- | --- | --- |
| UpdateUnit | Result | Set\_Get | 0 |  | |
| UpdateUnit | Result | Status | 4 | **程序升级结果** | **接收数据CRC16** |
| 2B | 2B |

监控终端 🡪 升级结点

UpdateUnit. Result. Set\_Get.0

监控终端 🡨 升级结点

UpdateUnit. Result. Status.4. (xx xx $$ $$)

其中xx xx表示升级程序结果（详细在ErrorCode中介绍）， $$ $$为接收文件的CRC16校验值。

### 1.2.5最终升级结果反馈LAST\_RESULT(0x0005)

如果接收完成所有文件并接收到监控终端发送的最后一个Result状态，若接收到的状态正常将自动触发LAST\_RESULT。

**LAST\_RESULT**指令消息结构：

| **功能单元** | **功能ID** | **操作类型** | **数据长度** | | **用户数据** |
| --- | --- | --- | --- | --- | --- |
| UpdateUnit | LastResult | Set\_Get | 0 |  | |
| UpdateUnit | LastResult | Status | 2 | 程序升级结果 | |

监控终端 🡨 升级结点

UpdateUnit. LastResult. Set\_Get.0

监控终端 🡪 升级结点

UpdateUnit. LastResult. Status.2. (xx xx)

其中xx xx表示升级程序的最终结果，如果所有文件升级均无错误，则反馈结果为NOERROR，如果有错误则反馈出错的文件序号和错误状态。（错误信息详细在ErrorCode中介绍）。

### 1.2.6升级完成END(0x0006)

监控终端接收到LastResult指令后判断程序升级已经全部完成并且没有任何错误的情况下，触发End过程结束程序升级，同时告知升级结点升级的最终状态。

升级结点接收到End指令后，根据接收到的程序升级状态自动判断是否需要跳转到应用程序区执行。

**END**指令消息结构：

| **功能单元** | **功能ID** | **操作类型** | **数据长度** | **用户数据** |
| --- | --- | --- | --- | --- |
| UpdateUnit | End | Set\_Get (Status) | 1 | TRUE or FALSE |

监控终端 🡪 升级结点

UpdateUnit. End. Set\_Get.1.(TRUE or FALSE)

监控终端 🡨 升级结点

UpdateUnit. End. Status.1.(TRUE or FALSE)

## 1.3调试单元

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **功能单元** | **功能单元ID** | **功能号定义** | **功能ID** | **描述** |
| DEBUG\_UNIT | 02H | GET\_BOOT\_VERSION | 11H | 获取Bootloader版本号 |
| GET\_APP\_VERSION | 12H | 获取应用程序版本号 |
| GET\_CHIP\_ID | 13H | 获取芯片ID |
|  |  |  |

### 1.3.1获取Bootloader版本号 GET\_BOOT\_VERSION (0x0011)

**GET\_BOOT\_VERSION**指令消息结构：

| **功能单元** | **功能ID** | **操作类型** | **数据长度** | **用户数据** |
| --- | --- | --- | --- | --- |
| DebugUnit | GET\_BOOT\_VERSION | Get | 0 |  |
| Status | N | 版本信息 |

监控终端 🡪 设备结点

DebugUnit. GET\_BOOT\_VERSION. Get.0.

监控终端 🡨 设备结点

DebugUnit. GET\_BOOT\_VERSION. Status.N.(Bootloader Version String)

### 1.3.2获取App版本号 GET\_APP\_VERSION (0x0012)

**GET\_APP\_VERSION**指令消息结构：

| **功能单元** | **功能ID** | **操作类型** | **数据长度** | **用户数据** |
| --- | --- | --- | --- | --- |
| DebugUnit | GET\_APP\_VERSION | Get | 0 |  |
| Status | N | 版本信息 |

监控终端 🡪 设备结点

DebugUnit. GET\_APP\_VERSION. Get.0.

监控终端 🡨 设备结点

DebugUnit. GET\_APP\_VERSION. Status.N.(App Version String)

### 1.3.3获取设备ID号GET\_CHIP\_ID (0x0013)

**GET\_CHIP\_ID**指令消息结构：

| **功能单元** | **功能ID** | **操作类型** | **数据长度** | **用户数据** |
| --- | --- | --- | --- | --- |
| DebugUnit | GET\_CHIP\_ID | Get | 0 |  |
| Status | 12B | 设备芯片唯一ID号 |

监控终端 🡪 设备结点

DebugUnit. GET\_CHIP\_ID. Get.0.

监控终端 🡨 设备结点

DebugUnit. GET\_CHIP\_ID. Status.12.(xx xx xx xx xx xx xx xx xx xx xx xx)

## 1.4通讯单元

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **功能单元** | **功能单元ID** | **功能号定义** | **功能ID** | **描述** |
| ROUTER\_UNIT | 03H | COMM\_HANDLE | 21H | 通讯握手心跳包 |
|  |  |  |
|  |  |  |
|  |  |  |

### 1.4.1握手心跳包 COMM\_HANDLE(0x0021)

**COMM\_HANDLE**指令消息结构：

| **功能单元** | **功能ID** | **操作类型** | **数据长度** | **用户数据** |
| --- | --- | --- | --- | --- |
| ROUTER\_UNIT | COMM\_HANDLE | Get | n |  |
| Status | n |  |

终端结点 🡪 主控单元

ROUTER\_UNIT. COMM\_HANDLE. Get. (xx xx)

终端结点 🡨 主控单元

ROUTER\_UNIT. COMM\_HANDLE. Status. (xx xx)

## 附录

### 附录A 数据大小端转换源码

#include "socket.h"

//大小端模式判断

static union **{** char c**[**4**];** unsigned long l**;** **}** endian\_test **=** **{** **{** 'l'**,** '?'**,** '?'**,** 'b' **}** **};**

#define ENDIANNESS ((char)endian\_test.l)

UINT16 bd\_sock\_ntohs**(**UINT16 netshort**)**

**{**

**if** **(**ENDIANNESS **==** 'l'**)**//little endian

**{**

UINT16 hostshort **=** 0**;**

hostshort **=** **(**netshort**)** **&** 0xff**;**

hostshort **=** **(**hostshort **<<** 8**)** **|** **((**netshort **>>** 8**)** **&** 0xff**);**

**return** hostshort**;**

**}**

**else** //big endian

**{**

**return** netshort**;**

**}**

**}**

UINT16 bd\_sock\_htons**(**UINT16 hostshort**)**

**{**

**if** **(**ENDIANNESS **==** 'l'**)**//little endian

**{**

UINT16 netshort **=** 0**;**

netshort **=** **(**hostshort**)** **&** 0xff**;**

netshort **=** **(**netshort **<<** 8**)** **|** **((**hostshort **>>** 8**)** **&** 0xff**);**

**return** netshort**;**

**}**

**else** //big endian

**{**

**return** hostshort**;**

**}**

**}**

UINT32 bd\_sock\_ntohl**(**UINT32 netlong**)**

**{**

**if** **(**ENDIANNESS **==** 'l'**)**//little endian

**{**

UINT32 hostlong **=** 0**;**

hostlong **=** **(**netlong**)** **&** 0xff**;**

hostlong **=** **(**hostlong **<<** 8**)** **|** **((**netlong **>>** 8**)** **&** 0xff**);**

hostlong **=** **(**hostlong **<<** 8**)** **|** **((**netlong **>>** 16**)** **&** 0xff**);**

hostlong **=** **(**hostlong **<<** 8**)** **|** **((**netlong **>>** 24**)** **&** 0xff**);**

**return** hostlong**;**

**}**

**else** //big endian

**{**

**return** netlong**;**

**}**

**}**

UINT32 bd\_sock\_htonl**(**UINT32 hostlong**)**

**{**

**if** **(**ENDIANNESS **==** 'l'**)**//little endian

**{**

UINT32 netlong **=** 0**;**

netlong **=** **(**hostlong**)** **&** 0xff**;**

netlong **=** **(**netlong **<<** 8**)** **|** **((**hostlong **>>** 8**)** **&** 0xff**);**

netlong **=** **(**netlong **<<** 8**)** **|** **((**hostlong **>>** 16**)** **&** 0xff**);**

netlong **=** **(**netlong **<<** 8**)** **|** **((**hostlong **>>** 24**)** **&** 0xff**);**

**return** netlong**;**

**}**

**else** //big endian

**{**

**return** hostlong**;**

**}**

**}**