

云知声蜂鸟 UART 通信协议接口文档

V2.0

2019 年 12 月

修改记录

版本	修订日期	修订人员	修订纪要
V2.0	2019.12.30	尚金龙	Release 2.0

Unisound

文档目录

1. 概述.....	4
1.1. 目的与受众.....	4
1.2. 简介.....	4
2. 接口定义.....	5
2.1. UART 底层 API.....	5
2.1.1. UartInitialize.....	5
2.1.2. UartFinalize.....	6
2.1.3. UartWrite.....	6
2.2. UART protocol API.....	7
2.2.1. CommProtocolInit.....	7
2.2.2. CommProtocolFinal.....	7
2.2.3. CommProtocolPacketAssembleAndSend.....	7
2.2.4. CommProtocolReceiveUartData.....	8
3. Protocol 帧结构.....	9
3.1.1. Protocol layout.....	9
3.1.2. Protocol struct.....	9
3.1.3. Protocol ACK.....	9
3.1.4. Protocol 重要参数指标.....	10
4. 接口调用流程.....	10
4.1. 已有自定义协议调用过程.....	10
4.2. 蜂鸟 UART 协议调用过程.....	11
5. 附录.....	13
5.1. 错误码.....	13

1. 概述

1.1. 目的与受众

本文系云知声蜂鸟芯片 UART 应用层通信串口协议，供内部开发人员和获授权客户开发使用，旨在为客户提供简单、通用的 UART 通信协议规范。

1.2. 简介

目标客户：

一、已有自定义 UART 通信协议客户，对此类型客户，开放 UART 读写 API，客户可自行定制基于 UART 的各类传输特性。

二、尚未自定义 UART 通信协议客户，对此类客户，开放一套完整的 UART 通信协议，提供 API，实现串口通信。

UART 协议特性：

- 一、支持可配置 ack 的可靠性传输。
- 二、自动组装数据帧，自动拆解数据帧。
- 三、支持误码率存在下，物理错误发生时协议可容错复位。
- 四、支持最大帧长检测设置。
- 五、支持控制命令类型定制，命令参数定制。
- 六、支持极低内存限制下，自适应内存垃圾回收功能。
- 七、crc16 校验和。

2. 接口定义

2.1. UART 底层 API

用于 uart 底层驱动初始化，数据发送、接收

2.1.1. UartInitialize

```
typedef enum {  
    UNI_UART1,  
    UNI_UART2,  
    UNI_UART3,  
} UartDeviceName;  
typedef enum {  
    UNI_B_1200,  
    UNI_B_2400,  
    UNI_B_4800,  
    UNI_B_9600,  
    UNI_B_14400,  
    UNI_B_19200,  
    UNI_B_38400,  
    UNI_B_57600,  
    UNI_B_115200,  
} UartSpeed;  
typedef enum {  
    UNI_PARITY_ODD,  
    UNI_PARITY_EVEN,  
    UNI_PARITY_NONE,  
    UNI_PARITY_MARK,  
    UNI_PARITY_SPACE,  
} UartParity;  
typedef enum {  
    UNI_ONE_STOP_BIT,  
    UNI_ONE_5_STOP_BIT,  
    UNI_TWO_STOP_BIT,  
} UartStop;
```

```
/**
 * uart init configure parameter
 */
typedef struct {
    UartDeviceName device; /* device name */
    UartSpeed speed; /* baudrate */
    UartParity parity; /* parity check */
    UartStop stop; /* stop bit */
    int data_bit; /* data bit */
} UartConfig;
typedef void (*RecvUartDataHandler)(char *buf, int len);
/**
 * @brief uart init
 * @param config uart configure parameter
 * @param handler handle uart receive data hook
 * @return 0 means success, -1 means failed
 */
int UartInitialize(UartConfig *config, RecvUartDataHandler handler);
```

2.1.2.UartFinalize

```
/**
 * @brief uart finalize
 * @param void
 * @return void
 */
void UartFinalize(void);
```

2.1.3.UartWrite

```
/**
 * @brief write data by UART, multi-thread unsafe, please write in sync mode
 * @param buf the data buffer to write
 * @param len the data length
 * @return the actual write length by UART
 */
int UartWrite(char *buf, int len);
```

2.2. UART protocol API

用于串口数据封装、解包

2.2.1. CommProtocolInit

```
typedef struct {
    CommCmd cmd; /* command, such as power_on, power_off */
    CommPayloadLen payload_len; /* parameter length of command */
    char payload[0]; /* parameter of command */
} PACKED CommPacket;

typedef void (*CommRecvPacketHandler)(CommPacket *packet);

/**
 * @brief communication protocol init
 * @param write_handler the write handler, such as UartWrite int uni_uart.h
 * @param recv_handler when uart data disassemble as communication protocol
 * frame, the frame will be translate to struct CommPacket, then the CommPacket
 * will callback to user
 * @return 0 means success, -1 means failed
 */
int CommProtocolInit(CommWriteHandler write_handler,
                    CommRecvPacketHandler recv_handler);
```

2.2.2. CommProtocolFinal

```
/**
 * @brief communication protocol finalize
 * @param void
 * @return void
 */
void CommProtocolFinal(void);
```

2.2.3. CommProtocolPacketAssembleAndSend

```
/**
 * @brief send one packet(communication protocol frame format)
```

```
* @param type customer type, 0 means Unisound
* @param cmd command type, should define as enum (such as power_on, power_off)
* @param payload the payload of cmd, can set as NULL
* @param payload_len the payload length
* @param attribute the attribute for this packet, such as packet need ACK
* @return 0 means success, other means failed
*/
int CommProtocolPacketAssembleAndSend(CommType type,
                                     CommCmd cmd,
                                     char *payload,
                                     CommPayloadLen payload_len,
                                     CommAttribute *attribute);
```

2.2.4. CommProtocolReceiveUartData

```
/**
* @brief receive original uart data
* @param buf the uart data buffer pointer
* @param len the uart data length
* @return void
*/
void CommProtocolReceiveUartData(char *buf, int len);
```

3. Protocol 帧结构

3.1.1. Protocol layout

```
/*-----*/  
/*          layout of unisound communication app protocol          */  
/*-----*/  
/*-1byte-|-1byte-|---2byte---|---2byte---|---2byte---|---2byte---|---N byte---*/  
/* SYNC |seq-num| customer |  command |  checksum |payload len|  payload */  
/*-----*/  
  
/*-----ack frame-----*/  
/* 0xFF | seqNum|   0x0   |   0x0   |   crc16   |   0x0   |   NULL   */  
/*-----*/
```

3.1.2. Protocol struct

```
typedef unsigned short CommChecksum;  
typedef unsigned char CommSync;  
typedef unsigned char CommSequence;  
typedef struct {  
    CommSync sync; /* must be UNI_COMM_SYNC_VALUE */  
    CommSequence sequence; /* sequence number */  
    CommType type; /* customer type, 0 means Unisound*/  
    CommCmd cmd; /* command type, such as power on, power off etc */  
    CommChecksum checksum; /* checksum of packet, use crc16 */  
    CommPayloadLen payload_len; /* the length of payload */  
    char payload[0]; /* the payload */  
} PACKED CommProtocolPacket;
```

3.1.3. Protocol ACK

客户可通过 CommProtocolPacketAssembleAndSend 接口参数 attribute 设置该条串口数据是否需要接收方进行 ACK 确认，以实现可靠传输。

ACK 帧满足 protocol 结构，我们固定了 ack 帧的形式，即，sync=0xFF, type = 0, cmd = 0, payload_len = 0, payload = NULL, sequence 必须和收到的帧中的 sequence 相同（即，如果要对 sequence 为 100 的 frame 进行 ack 确认，则 ack 帧中的 sequence 必须设置为 100），checksum 为帧中所有数据计算 crc16 结果（计算前，请将帧中 checksum 置零）

3.1.4. Protocol 重要参数指标

- 1、最大 payload: 8182 字节。即超过 8182 字节将被拒绝发送、拒绝拆包解析。
- 2、自动纠错灵敏度: 100ms。即如果接收到 1 帧数据过程中 byte 与 byte 之间的延时超过 100ms 则判断为物理错误，以此来抵抗物理传输中的比特翻转，如果出现则丢弃该帧数据，接收解析器复位，等待下一帧数据到来。
- 3、自动内存回收阈值: 256 字节。由于接收到的数据长度不确定（即每一个 command 帧字节可不同），接收解析器会对 protocol buffer 自动扩容、收缩，以此来减少内存开销，protocol buffer 初始长度 16 字节，扩容策略为当前长度的 2 倍（即 16、32、64、128...）当解析出一帧有效数据后，会自动回收大于等于 256 字节的 heap 内存；小于 256 字节选择常驻内存，以此优化内存分配性能，直到调用 CommProtocolFinal 销毁。
- 4、ack 超时有效值: 50ms ~ 2000ms。不在该区间，将自动收敛到该区间。

4. 接口调用流程

4.1. 已有自定义协议调用过程

```
#include "uni_uart.h"
#include "uni_log.h"
#define MAIN_TAG "main"
static void __recv_uart_data(char *buf, int len) {
    LOGT(MAIN_TAG, "receive uart data");
    /**
     * 接收到 uart 数据，在此进行处理，数据可能是 byte by byte 形式，不能假设发送方发送了
     * 32 字节，本次回调出来就是 32 字节，只能保证多次回调后，最终 32 字节
     */
}
int main() {
    /**
     * step 1. 初始化 UART。
     * 初始化完成后，即可以实现串口数据的读写操作。
     */
    UartConfig config;
    config.device = UNI_UART1;
    config.parity = UNI_PARITY_NONE;
    config.speed = UNI_B_115200;
    config.stop = UNI_ONE_STOP_BIT;
    config.data_bit = 8;
```

```
UartInitialize(&config, __recv_uart_data);  
/**  
 * step 2. 写数据  
 * 向 UART 写数据, 请调用 UartWrite 接口  
 */  
char buf[32];  
UartWrite(buf, sizeof(buf));  
LOGT(MAIN_TAG, "write random data to UART");  
/**  
 * step 3. 反初始化 UART。  
 */  
UartFinalize();  
return 0;  
}
```

4.2. 蜂鸟 UART 协议调用过程

```
#include "uni_uart.h"  
#include "uni_communication.h"  
#include "uni_log.h"  
#define MAIN_TAG "main"  
typedef enum {  
    POWER_ON = 0,  
    POWER_FF,  
} CommandType;  
static void __recv_customer_packet(CommPacket *packet) {  
    LOGT(MAIN_TAG, "receive packet, cmd=%d, payload_len=%d",  
        packet->cmd, packet->payload_len);  
}  
int main() {  
    /**  
     * step 1. 初始化 UART。  
     * 初始化完成后, 即可以实现串口数据的读写操作。  
     */  
    UartConfig config;  
    config.device = UNI_UART1;  
    config.parity = UNI_PARITY_NONE;
```

```
config.speed = UNI_B_115200;
config.stop = UNI_ONE_STOP_BIT;
config.data_bit = 8;
UartInitialize(&config, CommProtocolReceiveUartData);
/**
 * step 2. 初始化 protocol
 * 向 protocol 注册 UART 写方法、接收拆包后结构信息回调
 */
CommProtocolInit(UartWrite, __recv_customer_packet);
/**
 * step 3. 写数据
 * 向 UART 写一帧满足协议格式的数据
 */
char payload[32];
for (int i = 0; i < sizeof(payload); i++) {
    payload[i] = (char)i;
}
CommAttribute attribute;
attribute.need_acked = true;
attribute.timeout_msec = 100;
/**
 * payload、attribute 设置为 NULL 则代表无 payload、不需要 ack
 * attribute.need_acked = true; 则表示接收方必须发送 ack、否则本次发送失败、
 * 返回 ack 超时错误码
 */
CommProtocolPacketAssembleAndSend(UNI_CUSTOMER_TYPE,
                                   POWER_ON,
                                   payload,
                                   sizeof(payload),
                                   &attribute);
/**
 * step 3. 反初始化。
 */
CommProtocolFinal();
UartFinalize();
return 0;
}
```

5. 附录

5.1. 错误码

结果码	含义	备注
(0)	成功	通用错误码
(-1)	错误	通用错误码
E_UNI_COMM_ALLOC_FAILED (-10001)	申请 heap 内存失败	内存不足，申请失败
E_UNI_COMM_PAYLOAD_TOO_LONG (-10000)	发送的 payload 太长	当前支持最大 payload 为 8182 字节
E_UNI_COMM_PAYLOAD_ACK_TIMEOUT (-9999)	设置接收数据端需 ack 属性时，接收数据端 ack 超时	目前超时支持[50ms, 2000ms]区间