GB系统CAN多帧传输通信协议

文件信息

|  |  |  |  |
| --- | --- | --- | --- |
| 文件状态： | [ √ ] 草稿文件  [ ] 正式文件  [ ] 更改正式文件 | 当前版本： | 0.0.2 |
| 作 者： | 武敏 |
| 审 核： |  |
| 完成日期： | 2018.1.22 |
| 文档编号： |  | 文档标题： |  |
| 文档类别： |  | 提交人员： |  |
| 文 件 名： |  | | |
| 文件摘要： |  | | |
| 项目名称： |  | | |
| 当前阶段： |  | | |
| 版权所有： |  | | |

修改历史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **日期** | **版本** | **作者** | **修改内容** | **审核** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目录

[1 设计规格 4](#_Toc504548911)

[2 数据框图 4](#_Toc504548912)

[3 发送数据帧定义 4](#_Toc504548913)

[3.1 帧类型定义 5](#_Toc504548914)

[3.1.1 0x0 单帧协议 5](#_Toc504548915)

[3.1.2 0x1 起始帧(多帧协议) 5](#_Toc504548916)

[3.1.3 0x2中间帧(多帧协议) 5](#_Toc504548917)

[3.1.4 0x3结束帧(多帧协议) 5](#_Toc504548918)

[3.2 多帧长度(多帧协议) 5](#_Toc504548919)

[3.3 当前帧ID(多帧协议) 5](#_Toc504548920)

[4 代码实现 6](#_Toc504548921)

[4.1 宏定义部分 6](#_Toc504548922)

[4.2 结构体定义 6](#_Toc504548923)

[4.3 实现的函数声明 6](#_Toc504548924)

[5 发送接收案例 7](#_Toc504548925)

[5.1 要实现CAN连续发送0x00, 0x01, 0x02 … 0x3F 这样的连续的64字节的数据 7](#_Toc504548926)

[5.1.1 起始帧(ID = 0) 7](#_Toc504548927)

[5.1.2 中间帧(ID = 1) 7](#_Toc504548928)

[5.1.3 中间帧(ID=2) 7](#_Toc504548929)

[5.1.4 中间帧(ID=3) 7](#_Toc504548930)

[5.1.5 中间帧(ID=4) 7](#_Toc504548931)

[5.1.6 中间帧(ID=5) 7](#_Toc504548932)

[5.1.7 中间帧(ID=6) 8](#_Toc504548933)

[5.1.8 结尾帧(ID=7) 8](#_Toc504548934)

[6 附录 8](#_Toc504548935)

# 设计规格

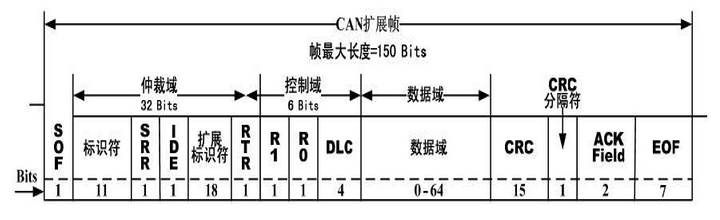
**CAN通讯局限于有效payload只有8字节, 这在很多场合中都不能适应现在的通讯环境, 为扩展发送变成数据, 又要兼顾通信的效率, 在扩展标识符中增加标识字段是业界行为有效的方法, 这篇文档介绍的是如何实现最大发送字节数为 2048 字节的一种通信协议.**

# 数据框图

|  |  |  |
| --- | --- | --- |
| CAN 多帧传输 | | |
| CAN单帧起始段 | CAN 单帧中间段 | CAN 单帧结束段 |

# 发送数据帧定义

CAN flex 协议的扩展帧格式为



我们的实现方式是在扩展标识符的 18bit 中加入识别码, 实现多帧传输

|  |  |  |
| --- | --- | --- |
| 17~16 bit | 15bit ~ 8bit | 7bit ~ 0bit |
| 帧类型定义 | 多帧长度 | 当前帧的ID |

## 帧类型定义

### 0x0 单帧协议

此表示为这是一个长度为1的应用层变长帧

### 0x1 起始帧(多帧协议)

标识此扩展帧是一个多帧协议的帧头

### 0x2中间帧(多帧协议)

标识此扩展帧是一个多帧协议的中间段

### 0x3结束帧(多帧协议)

标识此扩展帧是一个多帧协议的帧尾

## 多帧长度(多帧协议)

此表示段在多帧协议时起作用, 表示整个多帧包含多少个扩展帧

## 当前帧ID(多帧协议)

CAN 发送的有效字节是8字节, 此字段按照8字节的倍数进行组包, 这个 ID 从0开始计算, 最大256个单帧, 也就是最大发送 8\*256=2048 字节

# 代码实现

## 宏定义部分

#define HAL\_CAN\_SINGLE\_FRAME 0

#define HAL\_CAN\_MUL\_FRAME\_START 0x1

#define HAL\_CAN\_MUL\_FRAME\_MIDDLE 0x2

#define HAL\_CAN\_MUL\_FRAME\_END 0x3

## 结构体定义

typedef struct can\_multi\_frame\_info

{

unsigned char type;

unsigned char length;

unsigned char index;

}s\_can\_multi\_frame\_info;

typedef struct hal\_can\_frame

{

unsigned short int id;

s\_can\_multi\_frame\_info externed\_id;

unsigned char \*p\_data;

unsigned char trans\_status;

}s\_hal\_can\_frame;

## 实现的函数声明

int can\_send\_multi\_frame(int socket, s\_hal\_can\_frame \*p\_hal\_can\_frame);

int can\_recv\_multi\_frame(int socket, s\_hal\_can\_frame \*p\_hal\_can\_frame);

# 发送接收案例

## 要实现CAN连续发送0x00, 0x01, 0x02 … 0x3F 这样的连续的64字节的数据

要实现64 byte 数据发送, 要是连续发送 8 个扩展帧

### 起始帧(ID = 0)

扩展标识符: 0x10700

数据部分: 0x07, 0x06, 0x05, 0x04, 0x03, 0x02, 0x01, 0x00

### 中间帧(ID = 1)

扩展标识符: 0x20701

数据部分: 0x0F, 0x0E, 0x0D, 0x0C, 0x0B, 0x0A, 0x09, 0x08

### 中间帧(ID=2)

扩展标识符: 0x20702

数据部分: 0x17, 0x16, 0x15, 0x14, 0x13, 0x12, 0x11, 0x10

### 中间帧(ID=3)

扩展标识符: 0x20703

数据部分: 0x1F, 0x1E, 0x1D, 0x1C, 0x1B, 0x1A, 0x19, 0x18

### 中间帧(ID=4)

扩展标识符: 0x20704

数据部分: 0x27, 0x26, 0x25, 0x24, 0x23, 0x22, 0x21, 0x20

### 中间帧(ID=5)

扩展标识符: 0x20705

数据部分: 0x2F, 0x2E, 0x2D, 0x2C, 0x2B, 0x2A, 0x29, 0x28

### 中间帧(ID=6)

扩展标识符: 0x20706

数据部分: 0x37, 0x36, 0x35, 0x34, 0x33, 0x32, 0x31, 0x30

### 结尾帧(ID=7)

扩展标识符: 0x30707

数据部分: 0x3F, 0x3E, 0x3D, 0x3C, 0x3B, 0x3A, 0x39, 0x38

# 附录