



南京凌鸥创芯电子有限公司

LKS08X CAN 应用笔记

@ 2021, 版权归凌鸥创芯所有

机密文件，未经许可不得扩散

目录

1. 概述.....	3
2. 硬件描述.....	3
3. 软件使用.....	3
3.1. 时钟开启，GPIO 复用.....	3
3.2. CAN 模块初始化：.....	3
3.3. Id 滤波设置.....	3
3.4. 发送数据.....	4
3.5. 接收数据.....	4
4. 例程演示：.....	4
5. CAN_DMA 传输.....	6
5.1. CAN_DMA 软件配置：.....	6
5.2. 硬件连接：.....	7
5.3. 测试结果：.....	8

1. 概述

CAN 是汽车网络的标准协议，CAN 的高性能和可靠性已被认同。

CAN 控制器根据两根线上的电位差来判断总线电平，总线电平分为显性电平和隐性电平。发送方通过使总线电平发生变化，将消息发送给接收方。

2. 硬件描述

Lks08x 提供 can 协议的 TTL 接口，分别为 RX,TX 全双工接口。与节点设备需要通过硬件转换芯片进行总线连接(注意 CAN 测试时需连接 CAN 收发器且 CAN 收发器供电使用 5V 供电，或者将芯片 CNA_TX 与 CAN_RX 进行自测)。如图 1，

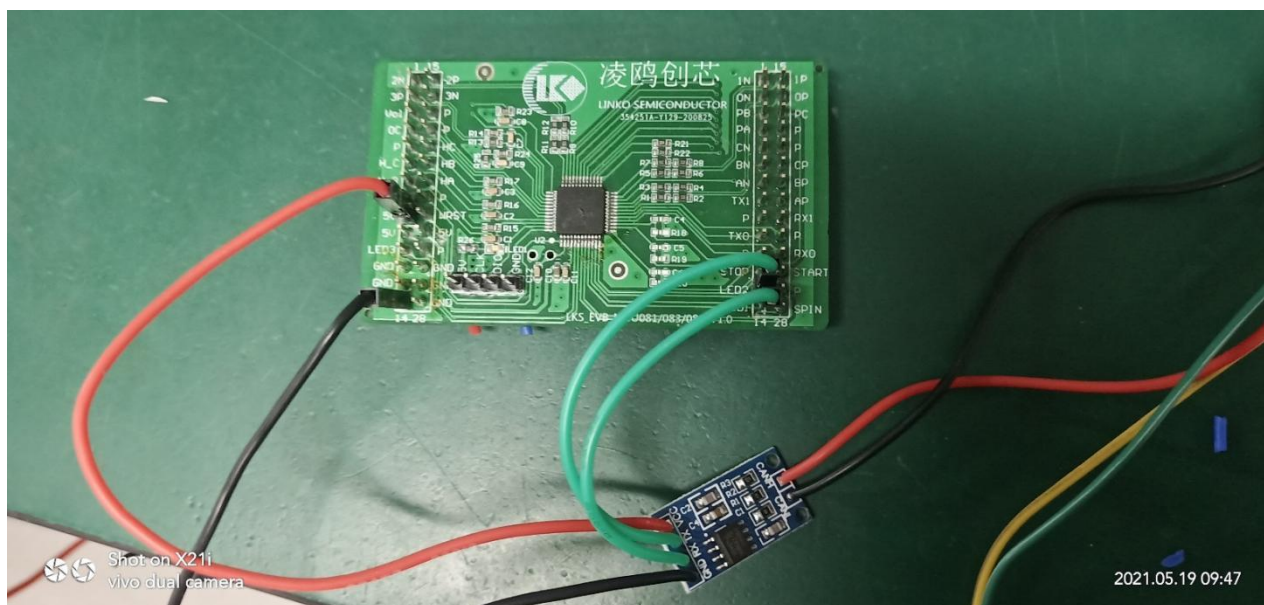


图 1

3. 软件使用

3.1. 时钟开启，GPIO 复用

3.2. CAN 模块初始化：

```
void CAN_Init(u8 btr0,u8 btr1);
```

通过 btr0, btr1 配置波特率。在 lks08x 用户手册中给出不同波特率下的推荐 btr0 和 btr1, 经过验证可以可靠运行。该函数同时开启的发送完成中断，接收完成中断，数据溢出中断，仲裁丢失中断和总线错误中断。

3.3. Id 滤波设置

单滤波 id 设置:

```
void ID_Filter(u32 acr,u32 amr,u8 ide);
```

acr 低 16 位为校对的 id 值, amr 低 16 位存放需要匹配的位掩码: 即 amr 中的第 BITn 为 1 时, 则接收到的 id 不需要与 ACR 的第 BITn 值相同, 若 amr 中的第 BITn 为 0 时, 则接收到的 id 的第 BITn 位需要与 ACR 的第 BITn 位相同。如 acr = 0X04d1 amr = 0xfffe, 则匹配的 id 的为 0Xxxxxxxx1。

ide: 1: id 为扩展帧长度格式 0: id 为标准帧长度格式

双 id 滤波设置:

```
void ID_Filter_Dual(u32 acr,u32 amr,u8 ide);
```

acr 低 16 位为校对的 id1 值, amr 低 16 位存放 id1 需要匹配的位掩码.

acr 高 16 位为校对的 id2 值, amr 高 16 位存放 id2 需要匹配的位掩码.

3.4. 发送数据

```
u8 My_CAN_Send_Msg(u32 id,u8 ide,u8 rtr, u8 *msg,u8 len);
```

id: 待发送的 id 值

ide: 0: 标准帧; 1: 扩展帧

rtr: 0: 数据帧; 1: 远程帧

msg: 待发送的数据区

len: 发送的数据长度

3.5. 接收数据

```
void CAN_Receive_Msg_IR(u32* id,u8* ide,u8* rtr,u8 *buf);
```

id: 获取接收到的 id 值

ide: 1: 接收到扩展帧; 0: 接收到标准帧

rtr: 1: 接收到远程帧; 0: 接收到数据帧

buf: 获取接收到的数据

4. 例程演示:

硬件设备: Lks08x 开发板和 CAN 分析盒。如图 2

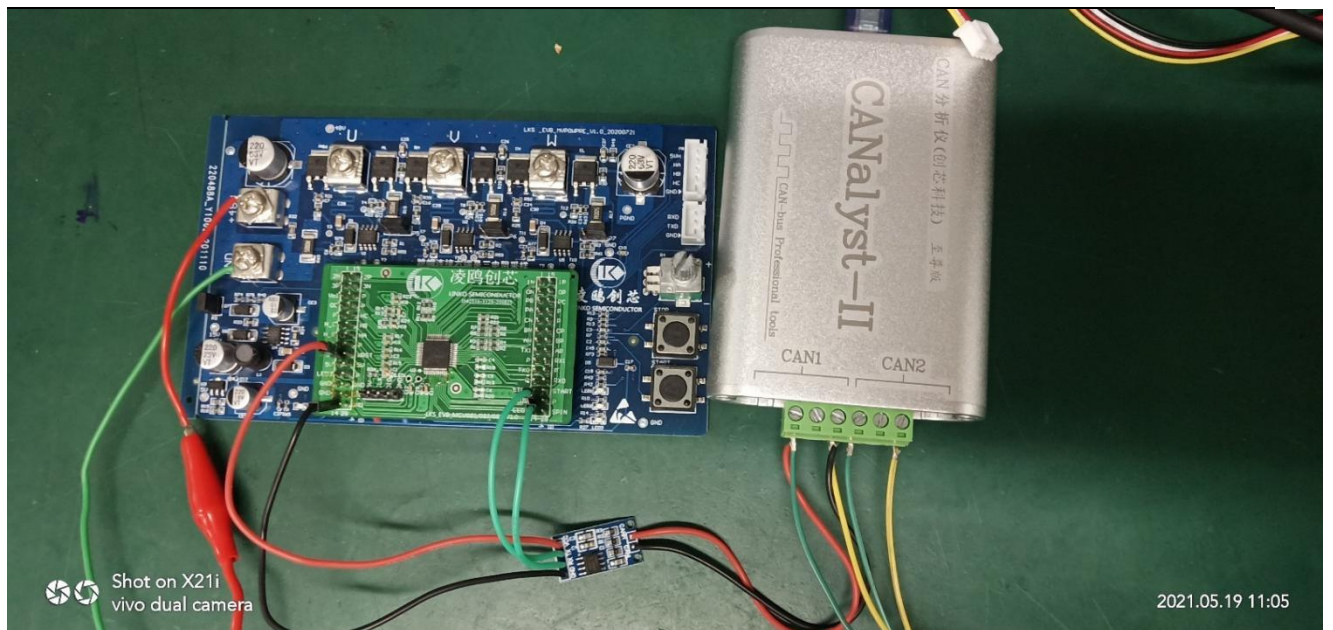


图 2

功能实现：

CAN 分析盒发送数据给开发板，开发板再将收到的数据回传。

波特率：500k，发送周期 10ms。

测试结果：可以正常收发。如图 3

CAN发送										
帧格式:	扩展帧	帧类型:	数据帧	帧ID:	00 00 04 d0	CAN通道:	2	发送总帧数:	10	<input checked="" type="checkbox"/> ID递增
数据:	11 00 11 00 00 00 00	发送消息						发送周期:	10	ms <input checked="" type="checkbox"/> 数据递增
CAN中继状态		接收滤波ID设置(直接ID号)		保存总帧数: 0		停止发送		发送文件		
<input checked="" type="radio"/> 使能		<input checked="" type="radio"/> 关闭		01 02		设置		<input checked="" type="checkbox"/> 打开CAN接收		清空
统计数据: 通道1		帧率R: 0		帧率T: 0	校验错误: 0	统计数据: 通道2		帧率R: 0	帧率T: 0	校验错误: 0
序号	系统时间	时间标识	CAN通道	传输方向	ID号	帧类型	帧格式	长度	数据	
00000	11:15:04.077	无	ch2	发送	0x000004D0	数据帧	扩展帧	0x08	x 11 00 11 00 00 00 00 00	
00001	11:15:04.086	无	ch2	发送	0x000004D1	数据帧	扩展帧	0x08	x 12 00 11 00 00 00 00 00	
00002	11:15:04.093	无	ch2	发送	0x000004D2	数据帧	扩展帧	0x08	x 13 00 11 00 00 00 00 00	
00003	11:15:04.101	0x2C31F8C	ch2	接收	0x04D0	数据帧	标准帧	0x08	x 11 00 11 00 00 00 00 00	
00004	11:15:04.101	0x2C31FE0	ch2	接收	0x04D1	数据帧	标准帧	0x08	x 12 00 11 00 00 00 00 00	
00005	11:15:04.101	0x2C3202C	ch2	接收	0x04D2	数据帧	标准帧	0x08	x 13 00 11 00 00 00 00 00	
00006	11:15:04.103	无	ch2	发送	0x000004D3	数据帧	扩展帧	0x08	x 14 00 11 00 00 00 00 00	
00007	11:15:04.113	无	ch2	发送	0x000004D4	数据帧	扩展帧	0x08	x 15 00 11 00 00 00 00 00	
00008	11:15:04.125	无	ch2	发送	0x000004D5	数据帧	扩展帧	0x08	x 16 00 11 00 00 00 00 00	
00009	11:15:04.132	0x2C32090	ch2	接收	0x04D3	数据帧	标准帧	0x08	x 14 00 11 00 00 00 00 00	
00010	11:15:04.132	0x2C320FC	ch2	接收	0x04D4	数据帧	标准帧	0x08	x 15 00 11 00 00 00 00 00	
00011	11:15:04.132	0x2C3216D	ch2	接收	0x04D5	数据帧	标准帧	0x08	x 16 00 11 00 00 00 00 00	
00012	11:15:04.135	无	ch2	发送	0x000004D6	数据帧	扩展帧	0x08	x 17 00 11 00 00 00 00 00	
00013	11:15:04.145	无	ch2	发送	0x000004D7	数据帧	扩展帧	0x08	x 18 00 11 00 00 00 00 00	
00014	11:15:04.153	无	ch2	发送	0x000004D8	数据帧	扩展帧	0x08	x 19 00 11 00 00 00 00 00	
00015	11:15:04.161	无	ch2	发送	0x000004D9	数据帧	扩展帧	0x08	x 1A 00 11 00 00 00 00 00	
00016	11:15:04.161	0x2C321C7	ch2	接收	0x04D6	数据帧	标准帧	0x08	x 17 00 11 00 00 00 00 00	
00017	11:15:04.161	0x2C32231	ch2	接收	0x04D7	数据帧	标准帧	0x08	x 18 00 11 00 00 00 00 00	
00018	11:15:04.161	0x2C32281	ch2	接收	0x04D8	数据帧	标准帧	0x08	x 19 00 11 00 00 00 00 00	
00019	11:15:04.161	0x2C322D2	ch2	接收	0x04D9	数据帧	标准帧	0x08	x 1A 00 11 00 00 00 00 00	

图 3

5. CAN_DMA 传输

5.1. CAN_DMA 软件配置:

- 1、初始化 DMA 模块，将本次发送的数据来源，接收的数据去向配置好，传输长度配置完毕。
- 2、初始化 GPIO 模块，将 CAN 复用的 GPIO 配置完毕。
- 3、初始化 CAN 接口，控制寄存器配置完毕。
- 4、触发 CAN 接口，进入发送状态

其实 CAN_DMA 与 CAN_MCU 传输的区别只是将 CAN_CMR 的 BIT5 位 DMA_EN 由 0 置 1，然后配置 DMA_CH3 即可。

19.2.3.2.2 CAN_CMR 命令寄存器

地址: 0x4001_3404

复位值: 0x0

表 19-9 命令寄存器 CAN_CMR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
											DMA_EN	RX_DUR_TX	CLR_OV	RELEASE_FIFO	INTR_TRANS	TRANS_REQ
											WO	WO	WO	WO	WO	WO
											0	0	0	0	0	0

位置	位名称	说明
----	-----	----



[31:6]		未使用
[5]	DMA_EN	写 1，使能 DMA 功能
[4]	RX_DUR_TX	写 1，发送数据的同时也将数据接收回来
[3]	CLR_OV	写 1，清除数据溢出标志位
[2]	RELEASE_FIFO	写 1，释放 RFIFO
[1]	INTR_TRANS	写 1，将中断取消未执行的发送传输
[0]	TRANS_REQ	写 1，产生 CAN 发送传输请求

图 4
6

本 CAN 模块设计的 DMA 不同于其它模块的 DMA 搬移操作，需要 MCU 介入部分搬移操作。假定当前配置 CAN 模块发送 N 帧数据，那么第一帧数据需要 MCU 搬移到 CAN 模块寄存器中，后续帧（N-1）的数据可由 DMA 实现搬移。

CAN_TX_DMA 例程中一次发送 5 帧数据，第一帧数据通过 My_CAN_Send_Msg()函数 MCU 发送给 CAN 模块，剩余四帧通过 DMA 搬移到 CAN 模块进行数据传输。其中需要注意，DMA 搬移时数据，数组中每组数据的前三个字节包含了地址、标准帧/数据帧、数据、数据长度信息，后面 8 个字节是要发送的数据，具体每帧数据的前 3 个字节如何赋值如下。（该例程采用 2.0B 协议,SFF 地址格式）

```
frame_inf |= ide << 7;           //ide 为 0 标准帧/1 扩展帧
frame_inf |= rtr << 6;          //rtr 为 0,数据帧;1,遥控帧
frame_inf |= len << 0;          //len 为发送数据长度
CAN_TX[0] = frame_inf;           //发送 TX 帧信息
CAN_TX[1] = id >> 3;             //TX ID0
CAN_TX[2] = (id & 0X07) << 5;   //TX ID1
```

如 CAN_TX_DMA 例程中发送地址 0x5555, 0 标准帧, 0 数据帧, 发 0x00, 0x01, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16 数据，数据长度 8。那一帧数据：

$CAN_TX[11] = \{0x08, 0xAA, 0xA0, 0x00, 0x01, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16\}$ 。

5.2. 硬件连接：

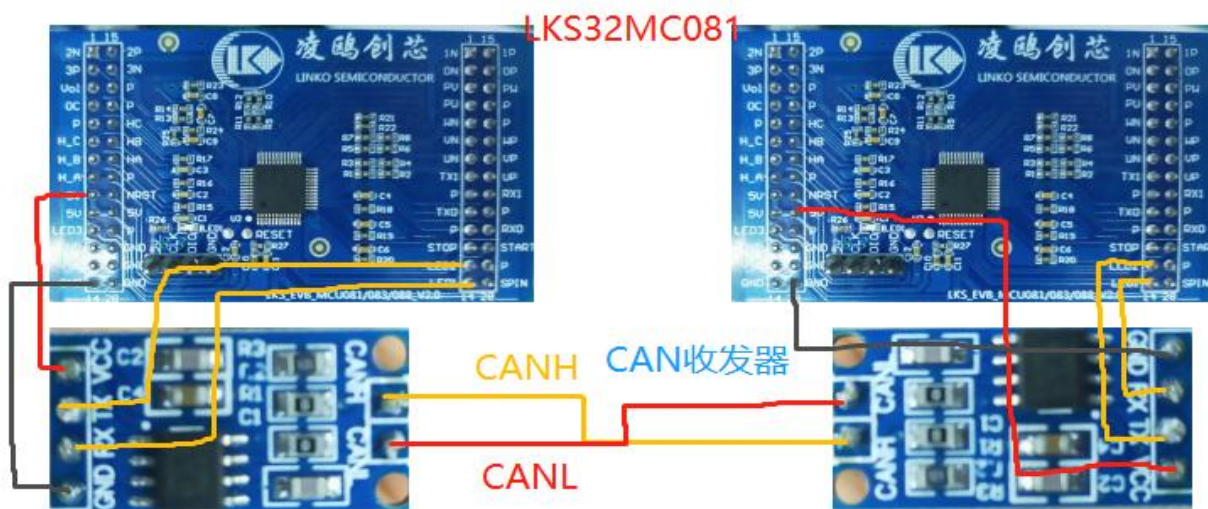


图 5

5.3. 测试结果:

CAN_TX 发送的数据:

```
#include "can.h"
u8 Can_tx[8] = {0x00, 0x01, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16}; 第一帧数据
u8 Can_TX[44] = {0x08, 0xAA, 0xA0, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 第二帧数据
                  0x08, 0xAA, 0xA0, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 第三帧数据
                  0x08, 0xAA, 0xA0, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 第四帧数据
                  0x08, 0xAA, 0xA0, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99 第五帧数据
                  };
```

CAN_RX 接收的数据:

Can_RX	0x20000028 Can_RX[] ...	unsigned char[5]
[0]	0x08	unsigned char
[1]	0xAA '?'	unsigned char
[2]	0xA0 '?'	unsigned char
[3]	0x00	unsigned char
[4]	0x01	unsigned char
[5]	0x11	unsigned char
[6]	0x12	unsigned char
[7]	0x13	unsigned char
[8]	0x14	unsigned char
[9]	0x15	unsigned char
[10]	0x16	unsigned char
[11]	0x08	unsigned char
[12]	0xAA '?'	unsigned char
[13]	0xA0 '?'	unsigned char
[14]	0x01	unsigned char
[15]	0x02	unsigned char
[16]	0x03	unsigned char
[17]	0x04	unsigned char
[18]	0x05	unsigned char
[19]	0x06	unsigned char
[20]	0x07	unsigned char
[21]	0x08	unsigned char
[22]	0x08	unsigned char
[23]	0xAA '?'	unsigned char
[24]	0xA0 '?'	unsigned char
[25]	0x11	unsigned char
[26]	0x12	unsigned char
[27]	0x13	unsigned char
[28]	0x14	unsigned char
[29]	0x15	unsigned char
[30]	0x16	unsigned char
[31]	0x17	unsigned char
[32]	0x18	unsigned char
[33]	0x18	unsigned char
[34]	0x08	unsigned char
[35]	0xAA '?'	unsigned char
[36]	0xA0 '?'	unsigned char
[37]	0x21 '!	unsigned char
[38]	0x22 '""	unsigned char
[39]	0x23 '#'	unsigned char
[40]	0x24 '\$'	unsigned char
[41]	0x25 '%'	unsigned char
[42]	0x26 '&'	unsigned char
[43]	0x27 ' "'	unsigned char
[44]	0x28 '('	unsigned char
[45]	0x08	unsigned char
[46]	0xAA '?'	unsigned char
[47]	0xA0 '?'	unsigned char
[48]	0x22 '""	unsigned char
[49]	0x33 '3'	unsigned char
[50]	0x44 'D'	unsigned char
[51]	0x55 'U'	unsigned char
[52]	0x66 'f'	unsigned char
[53]	0x77 'w'	unsigned char
[54]	0x88 '?'	unsigned char
[55]	0x99 '?'	unsigned char

图 6