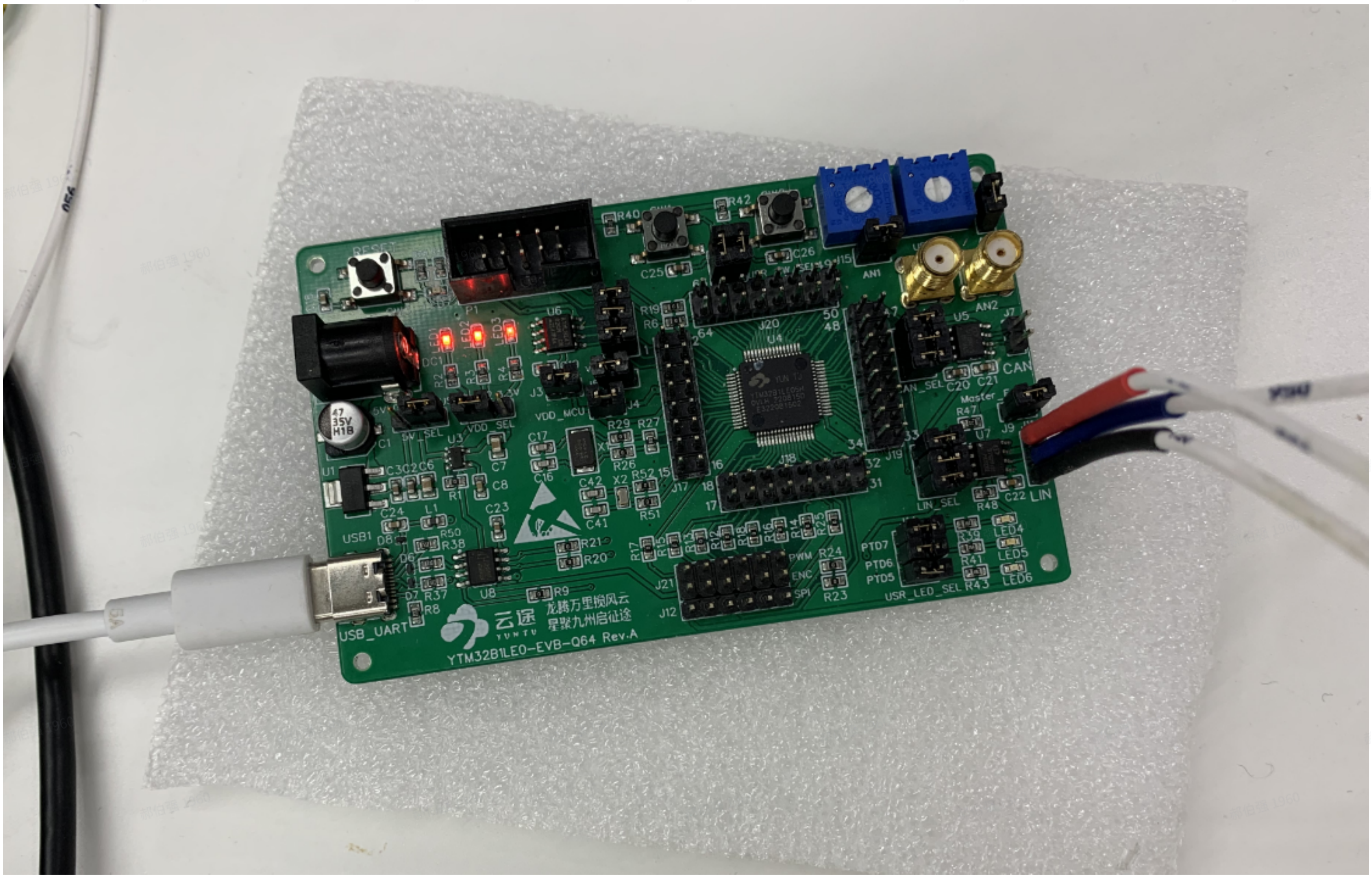
003-LE0: LIN slave

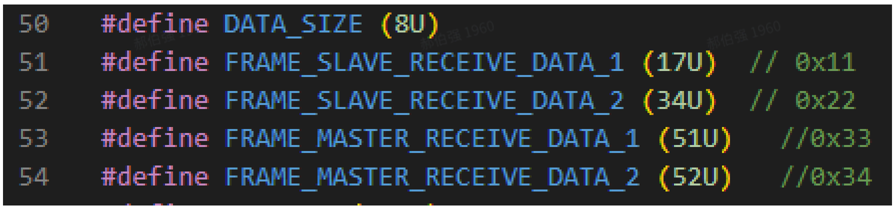
1. 环境搭建

LE0 EVB板5V供电，使用LIN工具盒给LIN模块提供12V电压



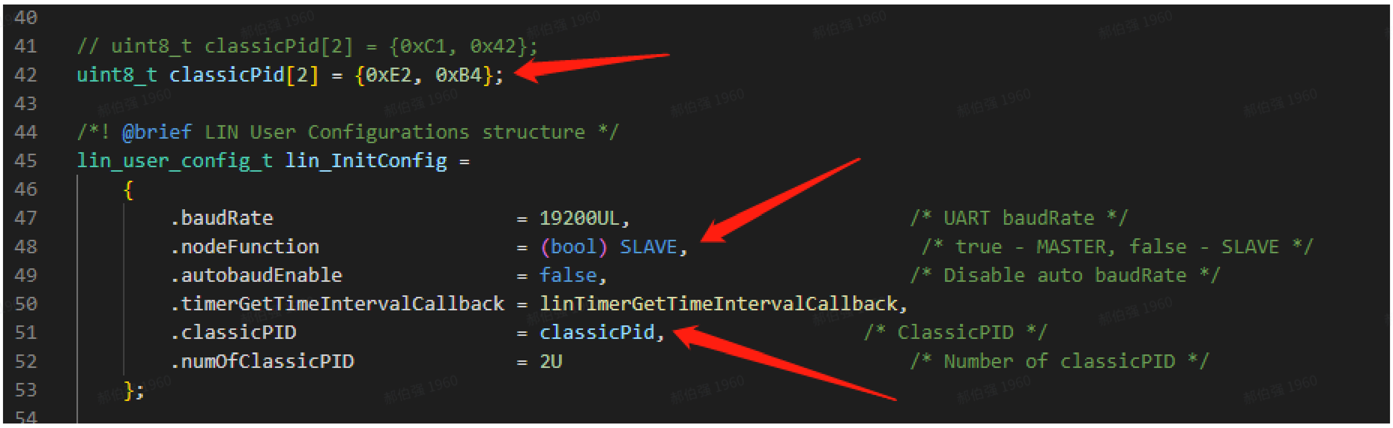
2. 初始化配置

   a. 定义接收和发送ID



```
#define DATA_SIZE (8U)
#define FRAME_SLAVE_RECEIVE_DATA_1 (17U)  // 0x11
#define FRAME_SLAVE_RECEIVE_DATA_2 (34U)  // 0x22
#define FRAME_MASTER_RECEIVE_DATA_1 (51U)   //0x33
#define FRAME_MASTER_RECEIVE_DATA_2 (52U)   //0x34
```

   b. demo板作为LIN slave，当主机以经典模式接收时配置classicPid数组，根据ID与PID对应表查得ID为0x22和0x34时对应的PID为0xB4和0xE2

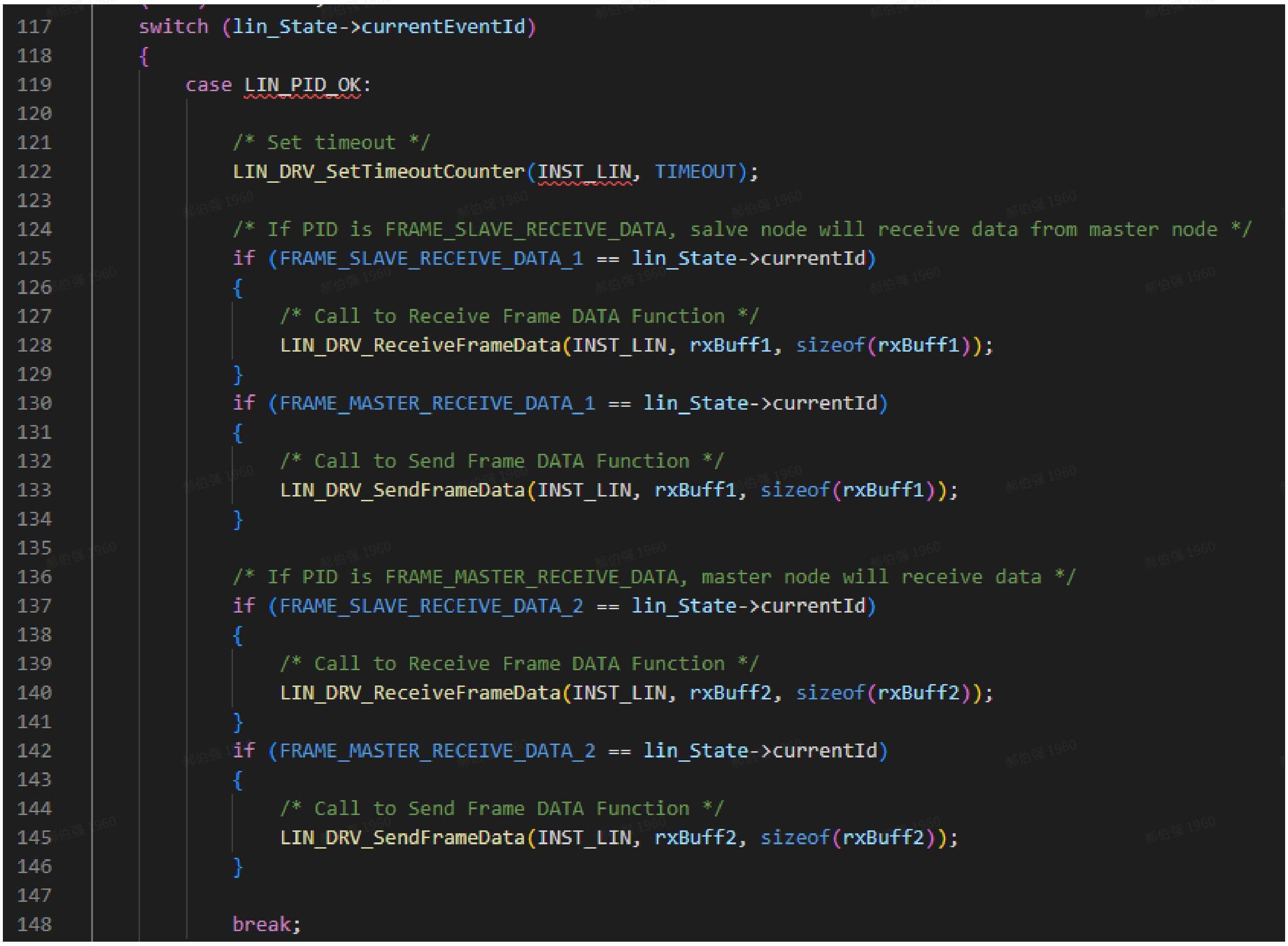


```
// uint8_t classicPid[2] = {0xC1, 0x42};
uint8_t classicPid[2] = {0xE2, 0xB4};

/*! @brief LIN User Configurations structure */
lin_user_config_t lin_InitConfig =
    {
        .baudRate                      = 19200UL,                       /* UART baudRate */
        .nodeFunction                  = (bool) SLAVE,                   /* true - MASTER, false - SLAVE */
        .autobaudEnable                = false,                         /* Disable auto baudRate */
        .timerGetTimeIntervalCallback = linTimerGetTimeIntervalCallback,
        .classicPID                    = classicPid,                 /* ClassicPID */
        .numOfClassicPID               = 2U                             /* Number of classicPID */
    };
```

c. 在中断回调函数里判断currentID符合哪种情况

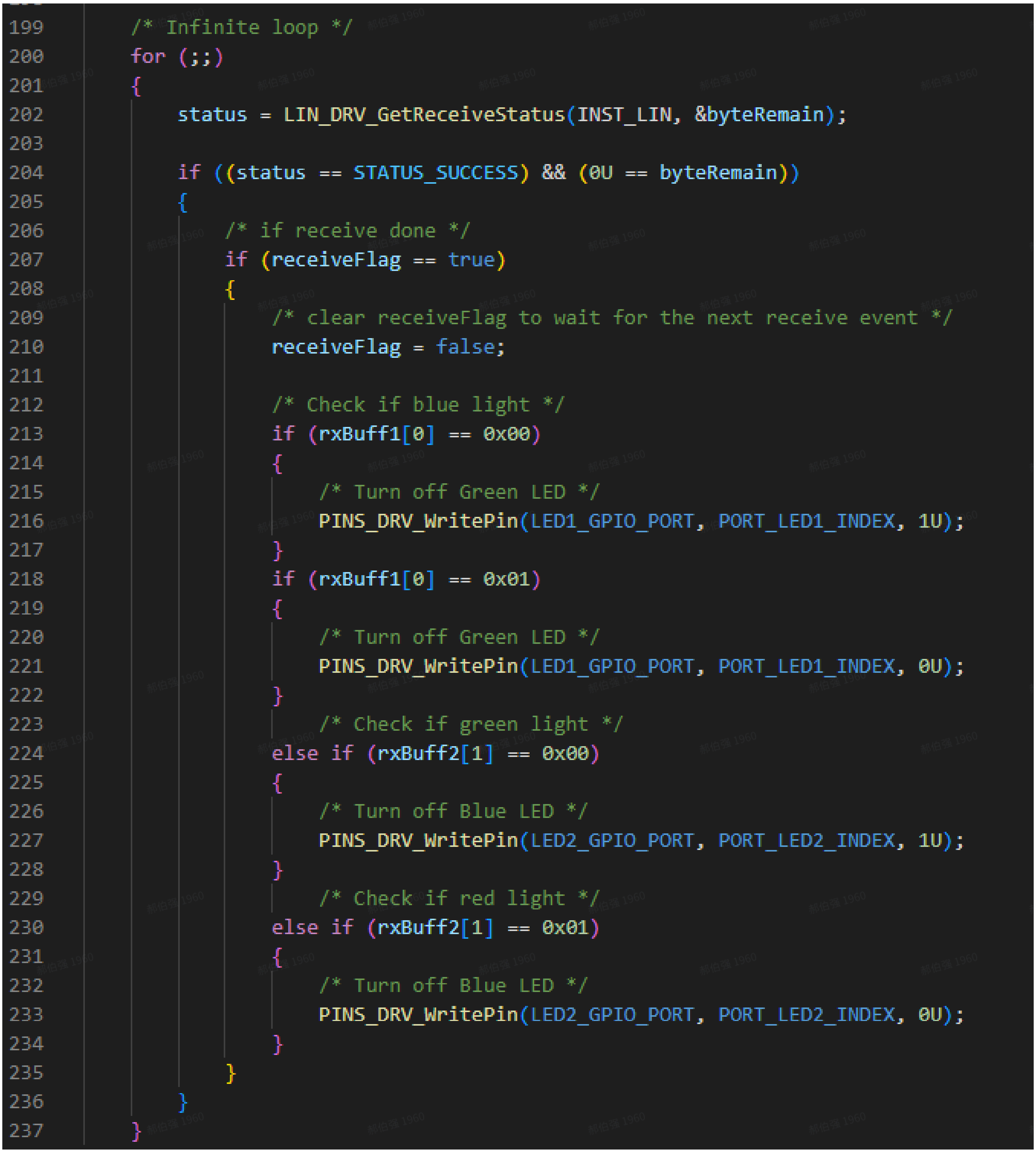
classicPid没有配置0x11和0x33，这两个ID为增强型

classicPic配置了0x22和0x34，这两个ID是标准型



```
        switch (lin_State->currentEventId)
        {
            case LIN_PID_OK:

                /* Set timeout */
                LIN_DRV_SetTimeoutCounter(INST_LIN, TIMEOUT);

                /* If PID is FRAME_SLAVE_RECEIVE_DATA, salve node will receive data from master node */
                if (FRAME_SLAVE_RECEIVE_DATA_1 == lin_State->currentId)
                {
                    /* Call to Receive Frame DATA Function */
                    LIN_DRV_ReceiveFrameData(INST_LIN, rxBuff1, sizeof(rxBuff1));
                }
                if (FRAME_MASTER_RECEIVE_DATA_1 == lin_State->currentId)
                {
                    /* Call to Send Frame DATA Function */
                    LIN_DRV_SendFrameData(INST_LIN, rxBuff1, sizeof(rxBuff1));
                }

                /* If PID is FRAME_MASTER_RECEIVE_DATA, master node will receive data */
                if (FRAME_SLAVE_RECEIVE_DATA_2 == lin_State->currentId)
                {
                    /* Call to Receive Frame DATA Function */
                    LIN_DRV_ReceiveFrameData(INST_LIN, rxBuff2, sizeof(rxBuff2));
                }
                if (FRAME_MASTER_RECEIVE_DATA_2 == lin_State->currentId)
                {
                    /* Call to Send Frame DATA Function */
                    LIN_DRV_SendFrameData(INST_LIN, rxBuff2, sizeof(rxBuff2));
                }

                break;
```

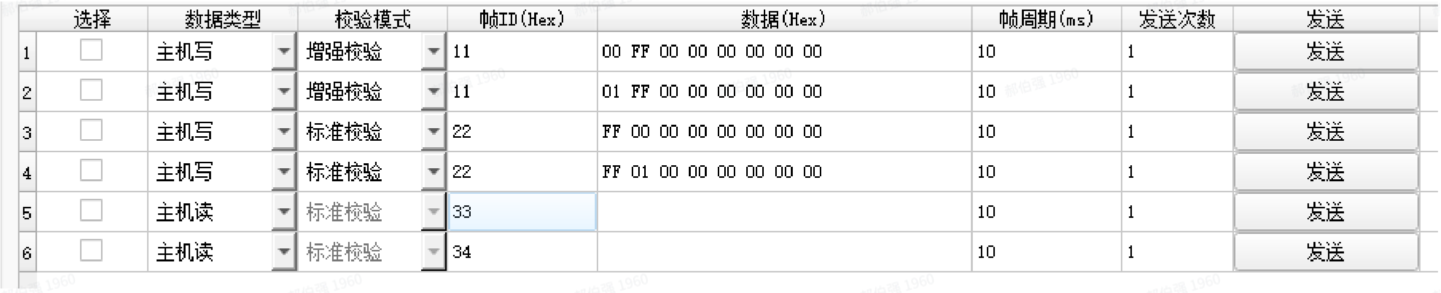
d. 在循环里判断data[]不同位来控制不同灯的开关



```c
    /* Infinite loop */
    for (;;)
    {
        status = LIN_DRV_GetReceiveStatus(INST_LIN, &byteRemain);

        if ((status == STATUS_SUCCESS) && (0U == byteRemain))
        {
            /* if receive done */
            if (receiveFlag == true)
            {
                /* clear receiveFlag to wait for the next receive event */
                receiveFlag = false;

                /* Check if blue light */
                if (rxBuff1[0] == 0x00)
                {
                    /* Turn off Green LED */
                    PINS_DRV_WritePin(LED1_GPIO_PORT, PORT_LED1_INDEX, 1U);
                }
                if (rxBuff1[0] == 0x01)
                {
                    /* Turn off Green LED */
                    PINS_DRV_WritePin(LED1_GPIO_PORT, PORT_LED1_INDEX, 0U);
                }
                    /* Check if green light */
                else if (rxBuff2[1] == 0x00)
                {
                    /* Turn off Blue LED */
                    PINS_DRV_WritePin(LED2_GPIO_PORT, PORT_LED2_INDEX, 1U);
                }
                    /* Check if red light */
                else if (rxBuff2[1] == 0x01)
                {
                    /* Turn off Blue LED */
                    PINS_DRV_WritePin(LED2_GPIO_PORT, PORT_LED2_INDEX, 0U);
                }
            }
        }
    }
```

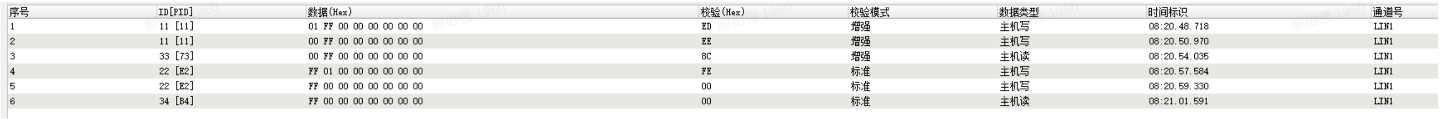
3. 测试结果

上位机控制指令



	选择	数据类型	校验模式	帧ID(Hex)	数据(Hex)	帧周期(ms)	发送次数	发送
1	□	主机写	增强校验	11	00 FF 00 00 00 00 00 00	10	1	发送
2	□	主机写	增强校验	11	01 FF 00 00 00 00 00 00	10	1	发送
3	□	主机写	标准校验	22	FF 00 00 00 00 00 00 00	10	1	发送
4	□	主机写	标准校验	22	FF 01 00 00 00 00 00 00	10	1	发送
5	□	主机读	标准校验	33		10	1	发送
6	□	主机读	标准校验	34		10	1	发送

上位机接收情况



序号	ID[PID]	数据(Hex)	校验(Hex)	校验模式	数据类型	时间标识	通道号
1	11 [11]	01 FF 00 00 00 00 00 00	ED	增强	主机写	08:20.48.718	LIN1
2	11 [11]	00 FF 00 00 00 00 00 00	EE	增强	主机写	08:20.50.970	LIN1
3	33 [73]	00 FF 00 00 00 00 00 00	8C	增强	主机读	08:20.54.035	LIN1
4	22 [E2]	FF 01 00 00 00 00 00 00	FE	标准	主机写	08:20.57.584	LIN1
5	22 [E2]	FF 00 00 00 00 00 00 00	00	标准	主机写	08:20.59.330	LIN1
6	34 [B4]	FF 00 00 00 00 00 00 00	00	标准	主机读	08:21.01.591	LIN1

lin_slave_l.zip

1.06MB