Application Note

ATK-MB025 CAN 模块使用说明

CAN 模块

使用说明

正点原子

广州市星翼电子科技有限公司

修订历史

版本	日期	原因
V1.0	2024/11/01	第一次发布

目 录

1，硬件连接

这里以正点原子 M48Z-M3 最小系统板 STM32F103 版为例，给大家介绍一下模块和板卡的连接方法。其它板卡与模块的硬件连接方法，请大家在“**ATK-MB025 CAN 模块\3，程序源码\相应板卡例程文件夹\readme.txt**”路径下查看。

ATK-MB025 CAN 模块可通过杜邦线与正点原子 M48Z-M3 最小系统板 STM32F103 版进行连接，具体的连接关系，如下表所示：

模块对应开发板	连接关系			
ATK-MB025 CAN 模块	VCC	GND	TX	RX
M48Z-M3 最小系统板 STM32F103 版	5V	GND	PA12	PA11

表 1.1 CAN 模块与 M48Z-M3 最小系统板 STM32F103 版连接关系

值得注意的是，要实现 CAN 普通模式的通信，则需两套相关的设备，例如：两块 STM32F103 最小系统板+两个 ATK-MB025 CAN 模块。通信的时候，两个 CAN 模块的连接关系如下表所示：

CAN 模块	连接关系		
模块 1	CANH	CANL	GND
模块 2	CANH	CANL	GND

表 1.2 两个 CAN 模块之间的连接关系

2，实验功能

2.1 ATK-MB025 CAN 模块测试实验

2.1.1 功能说明

在本实验中，串口会打印 CAN 模块发送或接收到的数据。需要查看这部分实验信息的用户，可用杜邦线将最小系统板 STM32F103 的 PA9 引脚和 GND 连接至外部的 USB 转串口设备，这样就可以通过 XCOM 上位机查看串口打印的信息了。

程序默认为 CAN 回环模式，仅可实现自发自收，用户按下 WK_UP 按键即可切换模式。当 CAN 模式为普通模式（Nnormal）时，即可与其他 CAN 设备进行通信。

当 KEY0 按键被按下时，CAN 模块会往外发送一次数据，数据量为 8 个字节，数据内容随机。当 CAN 模块接收到数据时，会自动将数据打印到串口。

用户可以通过拨动开关选择是否开启 120Ω终端电阻，ON 档即开启，OFF 档即关闭。

开发板的 LED0 闪烁，提示程序运行。

2.1.2 源码解读

打开本实验的工程文件夹，能够在./Drivers/BSP 目录下看到 ATK_CAN 文件夹，其包含了 ATK-MB025 CAN 模块的驱动文件，如下图所示：

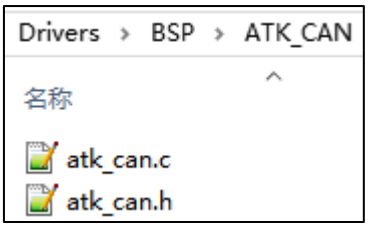

图 2.1.2.1 ATK-MB025 CAN 模块驱动代码

2.1.2.1 ATK-MB025 CAN 模块驱动

下面简要介绍 atk_can.c 中几个重要的 API 函数。

1. 函数 atk_can_init()

该函数用于初始化 ATK-MB025 CAN 模块相关的 CAN 外设，具体的代码，如下所示：

```
/**
 * @brief       CAN 初始化
 * @param       tsjw    : 重新同步跳跃时间单元.范围: 1~3;
 * @param       tbs2    : 时间段 2 的时间单元.范围: 1~8;
 * @param       tbs1    : 时间段 1 的时间单元.范围: 1~16;
 * @param       brp     : 波特率分频器.范围: 1~1024;
 *   @note      以上 4 个参数, 在函数内部会减 1, 所以, 任何一个参数都不能等于 0
 *              CAN 挂在 APB1 上面, 其输入时钟频率为 Fpclk1 = PCLK1 = 36Mhz
 *              tq     = brp * tpclk1;
 *              波特率 = Fpclk1 / ((tbs1 + tbs2 + 1) * brp);
 *              我们设置 atk_can_init(1, 8, 9, 4, 1), 则 CAN 波特率为:
 *              36M / ((8 + 9 + 1) * 4) = 500Kbps
 *
```

```
 * @param       mode    : CAN_MODE_NORMAL,  正常模式;
                          CAN_MODE_LOOPBACK,回环模式;
 * @retval      0,  初始化成功; 其他, 初始化失败;
 */
uint8_t atk_can_init(uint32_t tsjw, uint32_t tbs2, uint32_t tbs1,
                     uint16_t brp, uint32_t mode)
{
    g_canx_handler.Instance = CAN1;
    g_canx_handler.Init.Prescaler = brp;    /* 分频系数(Fdiv)为brp+1 */
    g_canx_handler.Init.Mode = mode;        /* 模式设置 */
    /* 重新同步跳跃宽度(Tsjw)为tsjw+1个时间单位 CAN_SJW_1TQ~CAN_SJW_4TQ */
    g_canx_handler.Init.SyncJumpWidth = tsjw;
    /* tbs1范围CAN_BS1_1TQ~CAN_BS1_16TQ */
    g_canx_handler.Init.TimeSeg1 = tbs1;
    /* tbs2范围CAN_BS2_1TQ~CAN_BS2_8TQ */
    g_canx_handler.Init.TimeSeg2 = tbs2;
    g_canx_handler.Init.TimeTriggeredMode = DISABLE;    /* 非时间触发通信模式 */
    g_canx_handler.Init.AutoBusOff = DISABLE;           /* 软件自动离线管理 */
    /* 睡眠模式通过软件唤醒(清除CAN->MCR的SLEEP位) */
    g_canx_handler.Init.AutoWakeUp = DISABLE;
    g_canx_handler.Init.AutoRetransmission = ENABLE;    /* 禁止报文自动传送 */
    /* 报文不锁定,新的覆盖旧的 */
    g_canx_handler.Init.ReceiveFifoLocked = DISABLE;
    /* 优先级由报文标识符决定 */
    g_canx_handler.Init.TransmitFifoPriority = DISABLE;

    if (HAL_CAN_Init(&g_canx_handler) != HAL_OK)
    {
        return 1;
    }

    #if CAN_RX0_INT_ENABLE
    /* 使用中断接收 */
    /* FIFO0消息挂号中断允许 */
    __HAL_CAN_ENABLE_IT(&g_canx_handler, CAN_IT_RX_FIFO0_MSG_PENDING);
    /* 使能CAN中断 */
    HAL_NVIC_EnableIRQ(USB_LP_CAN1_RX0_IRQn);
    /* 抢占优先级1，子优先级0 */
    HAL_NVIC_SetPriority(USB_LP_CAN1_RX0_IRQn, 1, 0);
    #endif

    CAN_FilterTypeDef sFilterConfig;

    /* 配置CAN过滤器 */
```

```
    sFilterConfig.FilterBank = 0;                       /* 过滤器 0 */
    sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;   /* 标识符屏蔽位模式 */
    sFilterConfig.FilterScale = CAN_FILTERSCALE_32BIT; /* 长度 32 位位宽*/
    sFilterConfig.FilterIdHigh = 0x0000;                /* 32 位 ID */
    sFilterConfig.FilterIdLow = 0x0000;
    sFilterConfig.FilterMaskIdHigh = 0x0000;            /* 32 位 MASK */
    sFilterConfig.FilterMaskIdLow = 0x0000;
    /* 过滤器 0 关联到 FIFO0 */
    sFilterConfig.FilterFIFOAssignment = CAN_FILTER_FIFO0;
    sFilterConfig.FilterActivation = CAN_FILTER_ENABLE; /* 激活滤波器 0 */
    sFilterConfig.SlaveStartFilterBank = 14;

    /* 过滤器配置 */
    if (HAL_CAN_ConfigFilter(&g_canx_handler, &sFilterConfig) != HAL_OK)
    {
        return 2;
    }

    /* 启动 CAN 外围设备 */
    if (HAL_CAN_Start(&g_canx_handler) != HAL_OK)
    {
        return 3;
    }

    return 0;
}
```

atk_can_init()函数主要用于配置 CAN 的接收时钟、模式、过滤器等功能，并使能 CAN 以开始 CAN 控制器的工作。关于 CAN 的外设介绍及配置方法，请查看正点原子各个开发板对应的开发指南中 CAN 章节，或查找对应芯片的官方参考手册。

2. 函数 HAL_CAN_MspInit()

调用 HAL_CAN_Init 后会自动调用 HAL_CAN_MspInit，我们重定义这个函数，在函数中初始化用于控制 CAN 的收发引脚：

```
/**
 * @brief       CAN 底层驱动，引脚配置，时钟配置，中断配置
                此函数会被 HAL_CAN_Init()调用
 * @param       hcan:CAN 句柄
 * @retval      无
 */
void HAL_CAN_MspInit(CAN_HandleTypeDef *hcan)
{
    if (CAN1 == hcan->Instance)
    {
        CAN_RX_GPIO_CLK_ENABLE();       /* CAN_RX 脚时钟使能 */
        CAN_TX_GPIO_CLK_ENABLE();       /* CAN_TX 脚时钟使能 */
```

```
        __HAL_RCC_CAN1_CLK_ENABLE();    /* 使能 CAN1 时钟 */

        GPIO_InitTypeDef gpio_initure;

        gpio_initure.Pin = CAN_TX_GPIO_PIN;
        gpio_initure.Mode = GPIO_MODE_AF_PP;
        gpio_initure.Pull = GPIO_PULLUP;
        gpio_initure.Speed = GPIO_SPEED_FREQ_HIGH;
        HAL_GPIO_Init(CAN_TX_GPIO_PORT, &gpio_initure); /* CAN_TX 脚 模式设置 */

        gpio_initure.Pin = CAN_RX_GPIO_PIN;
        gpio_initure.Mode = GPIO_MODE_AF_INPUT;
        /* CAN_RX 脚 必须设置成输入模式 */
        HAL_GPIO_Init(CAN_RX_GPIO_PORT, &gpio_initure);
    }
}
```

这里使用的 CAN 为 CAN1，TX 引脚为 PA12，RX 引脚为 PA11。

3. 函数 atk_can_send_msg()

该函数用于 CAN 模块发送数据，具体代码，如下所示：

```
/**
 * @brief       CAN 发送一组数据
 *   @note      发送格式固定为: 标准 ID, 数据帧
 * @param       id      : 标准 ID(11 位)
 * @retval      发送状态 0, 成功; 1, 失败;
 */
uint8_t atk_can_send_msg(uint32_t id, uint8_t *msg, uint8_t len)
{
    uint32_t TxMailbox = CAN_TX_MAILBOX0;
    uint8_t waittime;

    g_canx_txheader.StdId = id;             /* 标准标识符 */
    g_canx_txheader.ExtId = id; /* 扩展标识符(29 位) 标准标识符情况下，该成员无效*/
    g_canx_txheader.IDE = CAN_ID_STD;       /* 使用标准标识符 */
    g_canx_txheader.RTR = CAN_RTR_DATA;     /* 数据帧 */
    g_canx_txheader.DLC = len;

    if (HAL_CAN_AddTxMessage(&g_canx_handler, &g_canx_txheader, msg,
                  &TxMailbox) != HAL_OK)    /* 发送消息 */
    {
        return 1;
    }

     /* 等待发送完成,所有邮箱为空 */
    while(HAL_CAN_GetTxMailboxesFreeLevel(&g_canx_handler) != 3)
```

```
    {
        waittime++;

        if( waittime > 30)
        {
            return 1;
        }
        delay_ms(100);
    }

    return 0;
}
```

CAN 发送报文是有 ID，所以我们在发送数据时需要设定 ID，具体的 ID 由函数入口参数决定。配置完相关参数之后，再调用 HAL_CAN_AddTxMessage()函数来发送数据，接着等待发送完成。如果数据发送超时，则返回 1，表示发送失败了。

4. 函数 atk_can_receive_msg()

该函数用于 CAN 模块接收数据，具体代码，如下所示：

```
/**
 * @brief       CAN 接收数据查询
 *   @note      接收数据格式固定为：标准 ID，数据帧
 * @param       id      : 要查询的 标准 ID(11 位)
 * @param       buf     : 数据缓存区
 * @retval      接收结果
 *   @arg       0   , 无数据被接收到;
 *   @arg       其他，接收的数据长度
 */
uint8_t atk_can_receive_msg(uint32_t id, uint8_t *buf)
{
    /* 没有接收到数据 */
    if (HAL_CAN_GetRxFifoFillLevel(&g_canx_handler, CAN_RX_FIFO0) == 0)
    {
        return 0;
    }

    if (HAL_CAN_GetRxMessage(&g_canx_handler, CAN_RX_FIFO0,
          &g_canx_rxheader, buf) != HAL_OK)  /* 读取数据 */
    {
        return 0;
    }

    /* 接收到的 ID 不对 / 不是标准帧 / 不是数据帧 */
    if (g_canx_rxheader.StdId!= id || g_canx_rxheader.IDE != CAN_ID_STD
          || g_canx_rxheader.RTR != CAN_RTR_DATA)
    {
```

```
        return 0;
    }

    return g_canx_rxheader.DLC;
}
```

我们在接收数据时需要设定 ID 和接收缓冲区，具体的 ID 和缓冲区地址由函数入口参数决定。进入到该函数之后，先判断有没有接收到数据，没有数据则直接返回 0；反之则开始读取数据。如果数据读取失败，则直接返回 0；反之则校验 ID 和帧数据，如果校验不通过，则返回 0。有效的数据会被保存到接收缓冲区中，函数会返回数据的长度。

2.1.2.3 实验测试代码

实验的测试代码在 demo.c 文件中，该文件在工程根目录下的 User 文件夹。测试代码的入口函数为 demo_run()，具体的代码，如下所示：

```
/**
 * @brief       例程演示入口函数
 * @param       无
 * @retval      无
 */
void demo_run(void)
{
    uint8_t key;
    uint8_t i = 0, t = 0, res;
    uint8_t cnt = 0;
    uint8_t rxlen = 0;
    uint8_t canbuf[8];
    uint8_t mode = 1;                /* CAN 工作模式: 0,正常模式; 1,回环模式 */

    key_init();                      /* 初始化按键 */

    /* CAN 初始化, 回环模式, 波特率 500Kbps */
    atk_can_init(CAN_SJW_1TQ, CAN_BS2_8TQ, CAN_BS1_9TQ, 4, CAN_MODE_LOOPBACK);

    while (1)
    {
        key = key_scan(0);
        if (key == KEY0_PRES)        /* KEY0 按下,发送一次数据 */
        {
            for (i = 0; i < 8; i++)
            {
                canbuf[i] = cnt + i;  /* 填充发送缓冲区 */
                printf("Send Data : %d \r\n", canbuf[i]);
            }
            res = atk_can_send_msg(0X12, canbuf, 8); /* ID = 0X12, 发送 8 个字节 */
            if(res)
            {
```

```
            printf("Send Data Failed! \r\n");
        }
        else
        {
            printf("Send Data OK! \r\n");
        }
        printf("\r\n");
    }
    else if (key == WKUP_PRES)   /* WK_UP 按下，改变 CAN 的工作模式 */
    {
        mode = !mode;
        if (mode == 0)           /* 正常模式，需要 2 个开发板 */
        {
            /* CAN 正常模式初始化，正常模式，波特率 500Kbps */
            atk_can_init(CAN_SJW_1TQ, CAN_BS2_8TQ, CAN_BS1_9TQ, 4,
                         CAN_MODE_NORMAL);
            printf("Nnormal Mode \r\n");
        }
        else                     /* 回环模式,一个开发板就可以测试了 */
        {
            /* CAN 回环模式初始化，回环模式，波特率 500Kbps */
            atk_can_init(CAN_SJW_1TQ, CAN_BS2_8TQ, CAN_BS1_9TQ, 4,
                         CAN_MODE_LOOPBACK);
            printf("LoopBack Mode \r\n");
        }
        printf("\r\n");
    }
    /* CAN ID = 0X12，接收数据查询 */
    rxlen = atk_can_receive_msg(0X12, canbuf);
    if (rxlen) /* 接收到有数据 */
    {
        for (i = 0; i < rxlen; i++)
        {
            printf("Receive Data : %d \r\n", canbuf[i]);
        }
        printf("Receive Data OK! \r\n");
        printf("\r\n");
    }

    t++;
    delay_ms(10);

    if (t == 20)
    {
```

```
            LED0_TOGGLE(); /* 提示系统正在运行 */
            t = 0;
            cnt++;
        }
    }
}
```

从上面代码可以看出，整个测试代码的逻辑相对简单。首先初始化 CAN 模块相关的 CAN 外设，默认为回环模式（LoopBack），波特率 500Kbps，接着在 while 循环中不断地扫描按键状态以及获取接收的数据。

当 WK_UP 按键被按下后，CAN 模式会进行切换，可切换为回环模式（LoopBack）或普通模式（Nnormal）。CAN 处于回环模式（LoopBack）时，仅可实现自发自收；CAN 处于普通模式（Nnormal）时，则可以与其他 CAN 设备进行通信。

当 KEY0 按键被按下后，CAN 模块将会发送 8 个字节的随机数据，与此同时，串口会打印这些已发送的数据；当 CAN 模块接收到数据后，也会通过串口进行打印。LED0 闪烁表示程序正常运行。

2.1.3 实验现象

将 ATK-MB025 CAN 模块按照第一节“硬件连接”中介绍的连接方式与开发板连接，并将实验代码编译烧录至开发板中。**值得注意的是，要实现 CAN 普通模式的通信，则需两套相关的设备，例如：两块 STM32F103 最小系统板+两个 ATK-MB025 CAN 模块。两块开发板都烧录本实验代码，两个 CAN 模块的接线端子 CANH 接 CANH，CANL 接 CANL，GND 接 GND。**

本实验使用串口输出调试信息，因此需将开发板的 PA9 连接至 DAP 虚拟串口（或 USB 转 TTL 模块）的 RX 引脚。完成连接后，可通过串口调试助手 XCOM 查看实验信息输出，如下图所示：

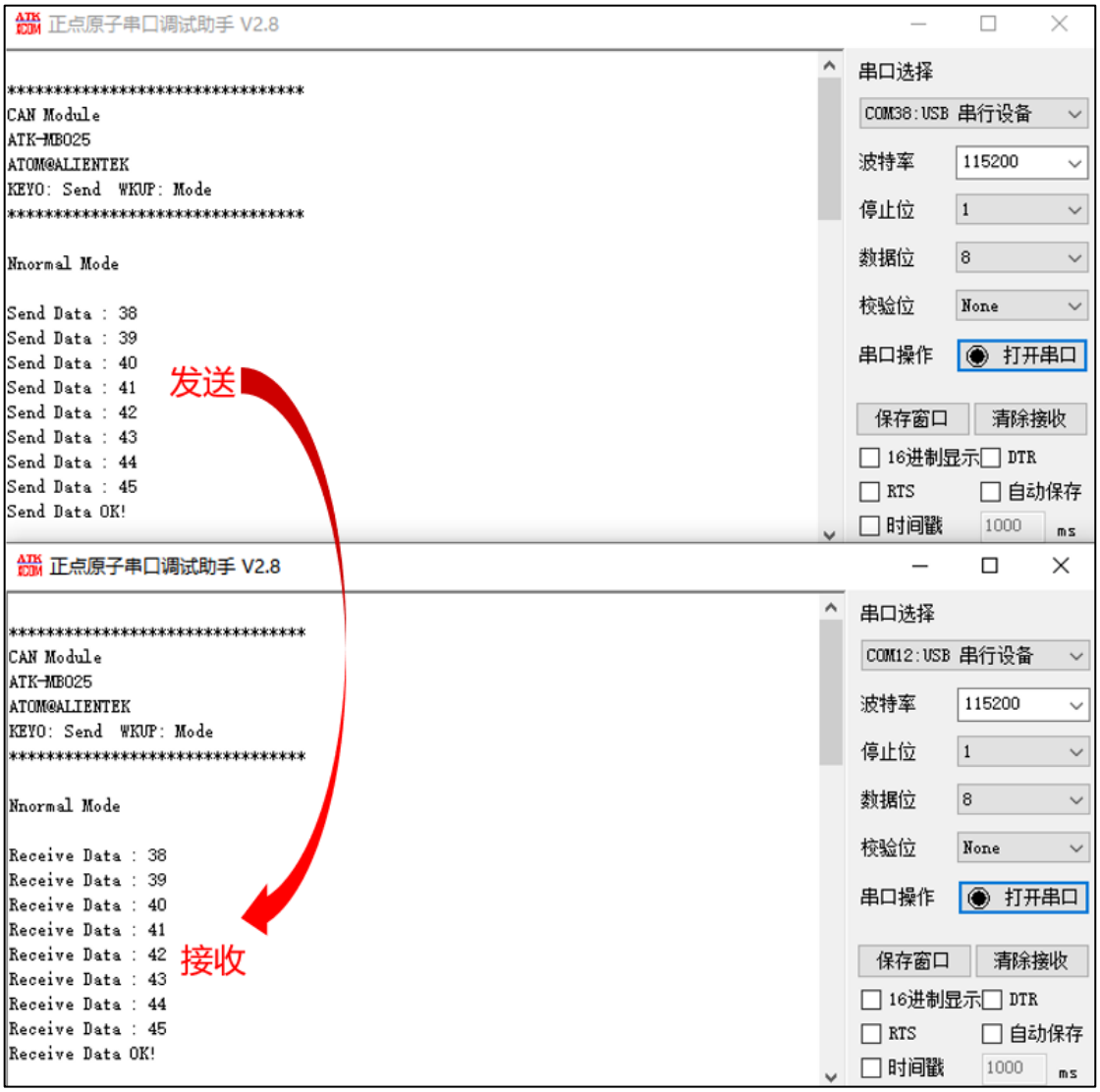

图 2.1.3.1 串口调试助手显示内容

注：其它现象请看 2.1.1 功能说明。

3，其他

1、 购买地址：

天猫：https://zhengdianyuanzi.tmall.com

淘宝：https://openedv.taobao.com

2、 资料下载

模块资料下载地址：http://www.openedv.com/docs/index.html

3、 技术支持

公司网址：www.alientek.com

技术论坛：http://www.openedv.com/forum.php

在线教学：www.yuanzige.com

B 站视频：https://space.bilibili.com/394620890

传真：020-36773971

电话：020-38271790

正点原子