零死角玩转STM32

与野火同行 乐意惬无边

1 原创教程，完全开源。

2 通俗易懂，详尽解读。

3 由浅入深，结合实操。

4 配套板子，全面玩转。

5 强强联合，不断更新。

野火团队 Wild Fire Team

0、友情提示

《零死角玩转 STM32》系列教程由初级篇、中级篇、高级篇、系统篇、四个部分组成，根据野火 STM32 开发板旧版教程升级而来，且经过重新深入编写，重新排版，更适合初学者，步步为营，从入门到精通，从裸奔到系统，让您零死角玩转 STM32。M3 的世界，与野火同行，乐意惬无边。

另外，野火团队历时一年精心打造的《STM32 库开发实战指南》将于今年 10 月份由机械工业出版社出版，该书的排版更适于纸质书本阅读以及更有利于查阅资料。内容上会给你带来更多的惊喜。是一本学习 STM32 必备的工具书。敬请期待！

8、CAN（Looback）

8.1 实验描述及工程文件清单

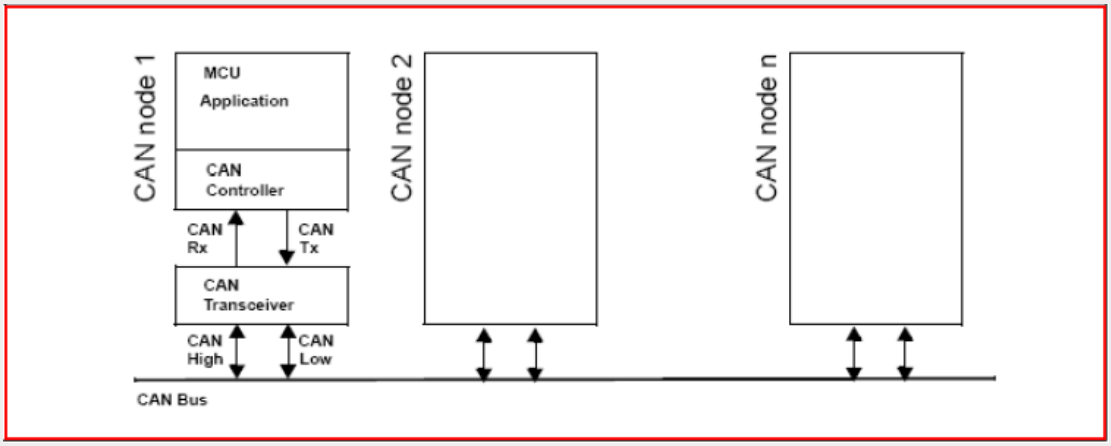
实验描述	can 测试实验(中断模式和回环)，并将测试信息通过 USART1 在超级终端中打印出来。
硬件连接	PB8-CAN-RX PB9-CAN-TX
用到的库文件	startup/start_stm32f10x_hd.c CMSIS/core_cm3.c CMSIS/system_stm32f10x.c FWlib/stm32f10x_gpio.c FWlib/stm32f10x_rcc.c FWlib/stm32f10x_usart.c FWlib/stm32f10x_can.c FWlib/misc.c
用户编写的文件	USER/main.c USER/stm32f10x_it.c USER/led.c USER/usart.c USER/can.c

8.2 CAN 简介

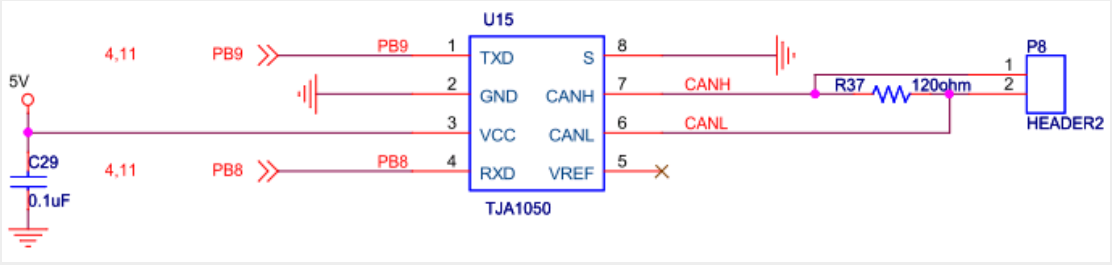
CAN 是控制器局域网络(Controller Area Network, CAN)的简称，是由研发和生产汽车电子产品著称的德国 BOSCH 公司开发的，并最终成为国际标准（ISO118?8）。是国际上应用最广泛的现场总线之一。 在北美和西欧，CAN 总线协议已经成为汽车计算机控制系统和嵌入式工业控制局域网的标准总线，并且拥有以 CAN 为底层协议专为大型货车和重工机械车辆设计的 J1939 协议。

近年来，其所具有的高可靠性和良好的错误检测能力受到重视，被广泛应用于汽车计算机控制系统和环境温度恶劣、电磁辐射强和振动大的工业环境。

野火 STM32 开发板的 CPU(cpu 型号为：STM32F103VET6)自带了一个 CAN 控制器。具体 I/O 定义为 PB8-CAN-RX、PB9-CAN-TX、。板载的 CAN 外接了一个 TJA1050 CAN 收发器，外部的 CAN 设备可以作为一个设备节点挂接到板载的 CAN 收发器中，实现 CAN 通信，多个 CAN 节点通信图如下：



野火 STM32 开发板中 CAN 硬件原理图如下：

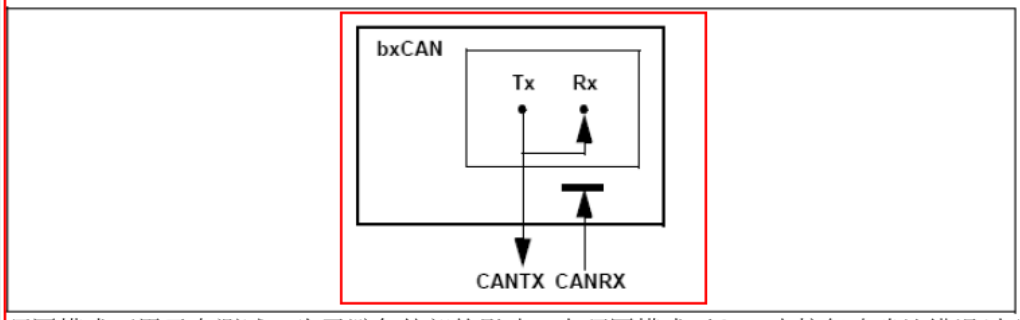


在本实验中并没有用得到双 CAN 通信，只是用了 CAN 的回环测试，这样我们就不需要挂接外部的 CAN 节点。有关双 CAN 通信的实验，大家可参考野火 STM32 光盘自带的例程**《16-野火 M3-CAN（Mutual）》**当我们用 CAN 的回环测试时，硬件会在内部将 TX 和 RX 连接起来，实现内部的收和发，从而达到测试的目的。

环回模式

通过对CAN_BTR寄存器的LBKM位置'1'，来选择环回模式。在环回模式下，bxCAN把发送的报文当作接收的报文并保存(如果可以通过接收过滤)在接收邮箱里。

图194　bxCAN工作在环回模式



环回模式可用于自测试。为了避免外部的影响，在环回模式下CAN内核忽略确认错误(在数据/远程帧的确认位时刻，不检测是否有显性位)。在环回模式下，bxCAN在内部把Tx输出回馈到Rx输入上，而完全忽略CANRX引脚的实际状态。发送的报文可以在CANTX引脚上检测到。

8.3 代码分析

首先在工程中添加需要用到的头文件：

```
FWlib/stm32f10x_gpio.c
FWlib/stm32f10x_rcc.c
 FWlib/stm32f10x_usart.c
 FWlib/stm32f10x_can.c
 FWlib/misc.c
```

还要将 c 文件对应的头文件添加进来，在库头文件 stm32f10x_conf.h 中实现：

```
1.  /* Includes ------------------------------------------------------------------*/
2.  /* Uncomment the line below to enable peripheral header file inclusion */
3.  /* #include "stm32f10x_adc.h" */
4.  /* #include "stm32f10x_bkp.h" */
5.  #include "stm32f10x_can.h"
6.  /* #include "stm32f10x_crc.h" */
7.  /* #include "stm32f10x_dac.h" */
8.  /* #include "stm32f10x_dbgmcu.h" */
9.  /* #include "stm32f10x_dma.h" */
10. /* #include "stm32f10x_exti.h" */
11. /*#include "stm32f10x_flash.h"*/
12. /* #include "stm32f10x_fsmc.h" */
13. #include "stm32f10x_gpio.h"
14. /* #include "stm32f10x_i2c.h" */
15. /* #include "stm32f10x_iwdg.h" */
16. /* #include "stm32f10x_pwr.h" */
17. #include "stm32f10x_rcc.h"
18. /* #include "stm32f10x_rtc.h" */
19. /* #include "stm32f10x_sdio.h" */
20. /* #include "stm32f10x_spi.h" */
21. /* #include "stm32f10x_tim.h" */
22. #include "stm32f10x_usart.h"
23. /* #include "stm32f10x_wwdg.h" */
24. #include "misc.h"  /* High level functions for NVIC and SysTick (add-
    on to CMSIS functions) */
```

OK，库环境已经配置好，接下来我们就开始分析 main 函数吧：

```
/**
  * @brief  Main program.
  * @param  None
  * @retval : None
  */
int main(void)
{
    /* config the sysclock to 72M */
    SystemInit();

    /* USART1 config */
    USART1_Config();
    /* LED config */
    LED_GPIO_Config();
    printf( "\r\n 这个一个 CAN（回环模式和中断模式）测试程序..... \r\n" );
    USER_CAN_Init();
    printf( "\r\n CAN 回环测试初始化成功..... \r\n" );
    USER_CAN_Test();
    printf( "\r\n CAN 回环测试成功..... \r\n" );
    while (1)
    {
    }
}
```

首先我们调用库 SystemInit();函数将我们的系统时钟设置为 72MHZ，紧接着初始化串口 USART1_Config();和 LED LED_GPIO_Config();，因为在本实验中我们需要用到串口来打印一些调试信息，用 LED 来显示程序的运行状态。有关这三个函数的详细讲解，请参考前面章节的教程，这里不再详述，其中 USART1_Config();和 LED_GPIO_Config();是由用户实现的应用函数，并非库函数。

USER_CAN_Init(); 实现了 CAN I/O 端口的初始化和中断优先级的初始化(因为等下要用到 CAN 的中断模式)。在 can.c 中实现：

```
/*
 * 函数名：CAN_Init
 * 描述  ：CAN 初始化，包括端口初始化和中断优先级初始化
 * 输入  ：无
 * 输出  ：无
 * 调用  ：外部调用
 */
void USER_CAN_Init(void)
{
    CAN_NVIC_Configuration();
    CAN_GPIO_Config();
}
```

USER_CAN_Init();调用了两个内部函数 CAN_NVIC_Configuration();和 CAN_GPIO_Config();，见名知义就可知道这两个函数的功能是什么啦，他们也均在 can.c 中实现：

```
/*
```

```
2.   * 函数名：CAN_NVIC_Configuration
3.   * 描述   ：CAN RX0 中断优先级配置
4.   * 输入   ：无
5.   * 输出   ：无
6.   * 调用   ：内部调用
7.   */
8.  static void CAN_NVIC_Configuration(void)
9.  {
10.   NVIC_InitTypeDef NVIC_InitStructure;
11.
12.   /* Enable CAN1 RX0 interrupt IRQ channel */
13.   NVIC_InitStructure.NVIC_IRQChannel = USB_LP_CAN1_RX0_IRQn;
14.   NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;      // 主优先级为 0
15.   NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;             // 次优先级为 0
16.   NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
17.   NVIC_Init(&NVIC_InitStructure);
18. }
```

在 CAN_GPIO_Config(void)这个函数中我们要注意一点：CAN 的 RX 脚要设置为上拉输入，TX 脚要设置为复用推挽输出，其他就跟配置普通 I/O 口一样。

```
1.  /*
2.   * 函数名：CAN_GPIO_Config
3.   * 描述   ：CAN GPIO 和时钟配置
4.   * 输入   ：无
5.   * 输出   ：无
6.   * 调用   ：内部调用
7.   */
8.  static void CAN_GPIO_Config(void)
9.  {
10.   GPIO_InitTypeDef GPIO_InitStructure;
11.   RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO | RCC_APB2Periph_GPIOB, ENABLE);


12.   /* CAN1 Periph clock enable */
13.   RCC_APB1PeriphClockCmd(RCC_APB1Periph_CAN1, ENABLE);
14.
15.   /* Configure CAN pin: RX */                           // PB8
16.   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
```

```
17.    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;        // 上拉输入
18.    GPIO_Init(GPIOB, &GPIO_InitStructure);
19.
20.    /* Configure CAN pin: TX */                           // PB9
21.    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
22.    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;       // 复用推挽输出
23.    GPIO_Init(GPIOB, &GPIO_InitStructure);
24.
25.  //#define GPIO_Remap_CAN     GPIO_Remap1_CAN1 本实验没有用到重映射 I/O
26.    GPIO_PinRemapConfig(GPIO_Remap1_CAN1, ENABLE);
27. }
```

当我们将 CAN 初始化好之后，接下来就是重头戏了。我们调用 USER_CAN_Test();函数。这个函数里面包含了 CAN 的回环和中断两种测试，在 can.c 中实现：

```
1.  /*
2.   * 函数名: CAN_Test
3.   * 描述   : CAN 回环模式跟中断模式测试
4.   * 输入   : 无
5.   * 输出   : 无
6.   * 调用   : 外部调用
7.   */
8.  void USER_CAN_Test(void)
9.  {
10.     /* CAN transmit at 100Kb/s and receive by polling in loopback mode */
11.   TestRx = CAN_Polling();
12.   if (TestRx == FAILED)
13.   {
14.     LED1( OFF );     // LED1 OFF
15.   }
16.   else
17.   {
18.     LED1( ON );    // LED1 ON;
19.   }
20.   /* CAN transmit at 500Kb/s and receive by interrupt in loopback mode */
21.   TestRx = CAN_Interrupt();
22.   if (TestRx == FAILED)
23.   {
24.     LED2( OFF );     // LED2 OFF;
25.   }
26.   else
27.   {
28.     LED2( ON );    // LED2 ON;
29.   }
30. }
```

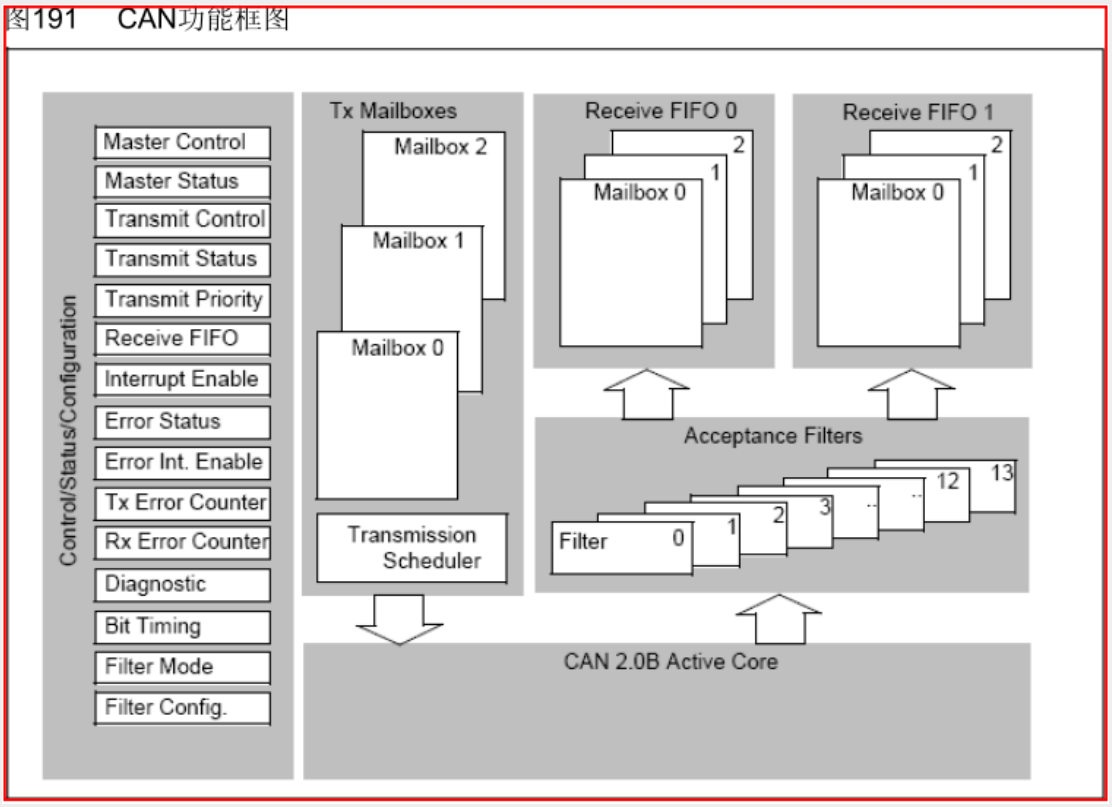
在这个函数中通过板载的 LED 的状态来显示回环和中断模式测试是否成功。其中回环模式测试调用了 CAN_Polling();这个函数，通信速率为 100Kb/s。中断模式测试调用了 CAN_Interrupt();这个函数，通信速率为 500Kb/s。

下面我们来重点分析下 CAN_Polling();和 CAN_Interrupt();这两个函数，这两个函数也在 can.c 中实现：

```c
/*
 * 函数名：CAN_Polling
 * 描述  ：配置 CAN 的工作模式为 回环模式
 * 输入  ：无
 * 输出  ：-PASSED   成功
 *         -FAILED   失败
 * 调用  ：内部调用
 */
TestStatus CAN_Polling(void)
{
  CAN_InitTypeDef        CAN_InitStructure;
  CAN_FilterInitTypeDef  CAN_FilterInitStructure;
  CanTxMsg TxMessage;
  CanRxMsg RxMessage;
  uint32_t i = 0;
  uint8_t TransmitMailbox = 0;

  /* CAN register init */
  CAN DeInit(CAN1);
  CAN StructInit(&CAN InitStructure);

  /* CAN cell init */
  CAN_InitStructure.CAN_TTCM=DISABLE;
  CAN_InitStructure.CAN_ABOM=DISABLE;
  CAN_InitStructure.CAN_AWUM=DISABLE;
  CAN InitStructure.CAN NART=DISABLE;
  CAN_InitStructure.CAN_RFLM=DISABLE;
  CAN_InitStructure.CAN_TXFP=DISABLE;
  CAN_InitStructure.CAN_Mode=CAN_Mode_LoopBack;  // 回环模式
  CAN InitStructure.CAN SJW=CAN SJW 1tq;
  CAN InitStructure.CAN BS1=CAN BS1 8tq;
  CAN_InitStructure.CAN_BS2=CAN_BS2_7tq;
  CAN_InitStructure.CAN_Prescaler=5;             // 分频系数为 5
  CAN_Init(CAN1, &CAN_InitStructure);            // 初始化 CAN

  /* CAN filter init */
  CAN_FilterInitStructure.CAN_FilterNumber=0;
  CAN_FilterInitStructure.CAN_FilterMode=CAN_FilterMode_IdMask;
  CAN FilterInitStructure.CAN FilterScale=CAN FilterScale 32bit;
  CAN FilterInitStructure.CAN FilterIdHigh=0x0000;
  CAN FilterInitStructure.CAN FilterIdLow=0x0000;
  CAN_FilterInitStructure.CAN_FilterMaskIdHigh=0x0000;
  CAN_FilterInitStructure.CAN_FilterMaskIdLow=0x0000;
  CAN_FilterInitStructure.CAN_FilterFIFOAssignment=0;
  CAN FilterInitStructure.CAN FilterActivation=ENABLE;
  CAN FilterInit(&CAN FilterInitStructure);

  /* transmit */
  TxMessage.StdId=0x11;             // 设定标准标识符（11 位，扩展的为 29 位）
  TxMessage.RTR=CAN_RTR_DATA;       // 传输消息的帧类型为数据帧（还有远程帧）
  TxMessage.IDE=CAN_ID_STD;         // 消息标志符实验标准标识符
  TxMessage.DLC=2;                  // 发送两帧，一帧 8 位
  TxMessage.Data[0]=0xCA;           // 第一帧数据
  TxMessage.Data[1]=0xFE;           // 第二帧数据

  TransmitMailbox=CAN_Transmit(CAN1, &TxMessage);
  i = 0;
    // 用于检查消息传输是否正常
  while((CAN_TransmitStatus(CAN1, TransmitMailbox) != CANTXOK) && (i != 0xFF))
  {
    i++;
  }

  i = 0;
  // 检查返回的挂号的信息数目
  while((CAN_MessagePending(CAN1, CAN_FIFO0) < 1) && (i != 0xFF))
  {
    i++;
  }
  /* receive */
  RxMessage.StdId=0x00;
  RxMessage.IDE=CAN_ID_STD;
  RxMessage.DLC=0;
  RxMessage.Data[0]=0x00;
  RxMessage.Data[1]=0x00;
  CAN Receive(CAN1, CAN FIFO0, &RxMessage);

  if (RxMessage.StdId!=0x11)
  {
    return FAILED;
  }
  if (RxMessage.IDE!=CAN_ID_STD)
  {
    return FAILED;
  }
```

```
86.   if (RxMessage.DLC!=2)
87.   {
88.     return FAILED;
89.   }
90.   if ((RxMessage.Data[0]<<8|RxMessage.Data[1])!=0xCAFE)
91.   {
92.     return FAILED;
93.   }
94.   //printf("receive data:0X%X,0X%X",RxMessage.Data[0], RxMessage.Data[1]);
95.   return PASSED; /* Test Passed */
96. }
```

在分析这两个函数之前，我们先来看下下面这幅图，截图来自《STM32 参考手册中文》。



上图中左边的 Control / Sdatus / Configuration 的具体细节我们先不管它，先放着。这个图中有三个专业名词，我们先解释下：

1-> Tx Mailboxes(发送邮箱)

STM32 的 CAN 中共有 3 个发送邮箱供软件来发送报文。发送调度器根据优先级决定哪个邮箱的报文先被发送。

2->Accepttance Filters(接收过滤器)

STM32 的 CAN 中共有 14 个位宽可变/可配置的标识符过滤器组，软件通过对它们编程，从而在引脚收到的报文中选择它需要的报文，而把其它报文丢弃掉。

3-> Receive FIFO(接收 FIFO)

STM32 的 CAN 中共有 2 个接收 FIFO，每个 FIFO 都可以存放 3 个完整的报文。它们完全由硬件来管理。在 CAN 回环测试实验中我们把要发送的数据放在 **Tx Mailboxes** 中，数据集经过 **Accepttance Filters** 发送到 **Receive FIFO**，再将接收到的数据存到相应的缓冲区中，通过比较接收和发送的数据是否相同来验证我们的实验是否正确。实验过程看起来很简单：发 送 -> 过滤 -> 接收 -> 比较。但只是宏观上把握罢了，这期间还需要做许多的工作。现在我们再回到 `CAN_Polling(void)` 这个函数中，看看这一过程我们到底做了些什么：

1-> CAN 寄存器初始化，全部初始化为默认值。

2->CAN cell 初始化。具体可参考源码，代码里面有详细注释。

3->CAN 过滤器初始化。

4->发送数据。这其中包括了设置设定标准标识符、传输消息的帧类型、要发送多少帧、邮箱中的帧数据是什么等。具体可参考源码，代码里面有详细注释。

5->接收数据。要初始化接收邮箱，用于接收数据，比较报文中设定的标准标识符是否相等等、最后比较发送的数据和接收的数据是否相等。具体可参考源码，代码里面有详细注释。

6->最后返回 TestStatus 信息，PASSED 表示成功，FAILED 表示失败。

在阅读这部分代码时，需要参考《STM32 参考手册中文》第 21 章<控制器局域网 bxcan>，这样对理解代码有很大的帮助。刚开始的时候我们也没太大必要完全去搞懂每一句代码是什么意思，先在宏观上把握他，看下这段代码是否真的工作了，一步一步去修改它，测试它，看看实验会发生什么效果。这样学习的时间长了，理解就自然深刻了，以前那些你觉得很纳闷的问题，那些瓶颈问题都会瞬间解决，一下子豁然开朗。其实这也是我的一种学习方法。量变了，质变还会远吗？

CAN 的中断模式跟回环模式是类似的，具体的大家自行阅读代码吧。唯一一个不同的是我们需要在 stm32f10x_it.c 中添加 CAN 的中断服务程序：

```
/*
 * 函数名：USB_LP_CAN1_RX0_IRQHandler
 * 描述  ：USB 中断和 CAN 接收中断服务程序，USB 跟 CAN 公用 I/O，这里只用到 CAN 的中断。
 * 输入  ：无
 * 输出  ：无
 * 调用  ：无
 */
void USB_LP_CAN1_RX0_IRQHandler(void)
{
  CanRxMsg RxMessage;

  RxMessage.StdId=0x00;
  RxMessage.ExtId=0x00;
  RxMessage.IDE=0;
  RxMessage.DLC=0;
  RxMessage.FMI=0;
  RxMessage.Data[0]=0x00;
  RxMessage.Data[1]=0x00;

  CAN_Receive(CAN1, CAN_FIFO0, &RxMessage);

  if((RxMessage.ExtId==0x1234) && (RxMessage.IDE==CAN_ID_EXT)
     && (RxMessage.DLC==2) && ((RxMessage.Data[1]|RxMessage.Data[0]<<8)==0xDECA))
  {
    ret = 1;
  }
  else
  {
    ret = 0;
  }
}
```

8.4 实验现象

将野火 STM32 开发板供电(DC5V)，插上 JLINK，插上串口线(两头都是母的交叉线)，打开超级终端，配置超级终端为 115200 8-N-1，将编译好的程序下载到开发板，可看到 LED1 和 LED2 全亮，超级终端打印出如下信息：