技　术　文　件

技术文件名称：LT-Easy2812 开发板 eCAN 和 ADC 实验手册

技术文件编号：<V1.0>

版　　本：<V1.0>

拟　制　叶红渝

审　核　张　勇

批　准

力天电子 www.LT430.com

修改记录

文件编号	版本号	拟制人/修改人	拟制/修改日期	更改理由	主要更改内容（写要点即可）
	1.0	叶红渝	2011-6-02		

1、eCAN 发送数据实验

实验目的：初步了解 DSP2812 内部 eCAN 发送数据的操作。

实验现象：上位机软件 CAN 分析仪将接收的数据格式显示出来。

实验原理：通过对 eCAN 相关寄存器的配置，启动 eCAN 的传输操作，数据就会通过 CAN 接口传输出来，通信的波特率为 1Mbps，帧格式采用扩展帧，邮箱通过邮箱 0 作为发送邮箱，第一次发送的数据为 0x0123456789ABCDEF,之后循环发送字符串“LiTian”,通过 CAN 模块的解析之后，上传到上位机软件，由软件将相关数据显示在界面中。

注意事项：本实验需要在开发板外围接 CAN 模块，同时需要上位机软件才能实现最终显示的结果

实验代码：

主程序代码如下：

```
void main(void)
{
    unsigned int i,j,k;
    unsigned char senddata[]="LiTian";
    InitSysCtrl();  // 系统初始化子程序，在 DSP281x_sysctrl.c 中

        /*初始化 ecan 寄存器*/
     InitECan();

     MessageSendCount = 0;
    MessageReceiveCount = 0;
     i=0;

    for(;;)
    {
        ECanaShadow.CANTRS.all=ECanaRegs.CANTRS.all;
        ECanaShadow.CANTRS.all=0;
          ECanaShadow.CANTRS.bit.TRS0=1;
          ECanaRegs.CANTRS.all=ECanaShadow.CANTRS.all;
```

```
            do
            {
              ECanaShadow.CANTA.all=ECanaRegs.CANTA.all;
            }while(ECanaShadow.CANTA.bit.TA0!=1);

              ECanaShadow.CANTA.bit.TA0=1;
              ECanaRegs.CANTA.all=ECanaShadow.CANTA.all;


          MessageSendCount++;                    //在这里设断点，观察

          ECanaMboxes.MBOX0.MDL.all =senddata[i];
           ECanaMboxes.MBOX0.MDH.all =senddata[i+1];

          i=i+2;

            if(i>5)
          i=0;
          for(k=0;k<100;k++)
              for(j=0;j<10000;j++);
      }

}
```

eCAN 发送数据初始化代码如下：

```
void InitECan(void)
{
    struct ECAN_REGS ECanaShadow;
    EALLOW;
```

```
//配置 GPIO 引脚工作在 eCAN 功能
GpioMuxRegs.GPFMUX.bit.CANRXA_GPIOF7=1;
GpioMuxRegs.GPFMUX.bit.CANTXA_GPIOF6=1;

//配置 eCAN 的 RX 和 TX 分别为 eCAN 的接收和发送引脚
ECanaShadow.CANTIOC.all = ECanaRegs.CANTIOC.all;
ECanaShadow.CANTIOC.bit.TXFUNC = 1;
ECanaRegs.CANTIOC.all = ECanaShadow.CANTIOC.all;

ECanaShadow.CANRIOC.all = ECanaRegs.CANRIOC.all;
ECanaShadow.CANRIOC.bit.RXFUNC = 1;
ECanaRegs.CANRIOC.all = ECanaShadow.CANRIOC.all;
EDIS;

EALLOW;
ECanaShadow.CANMC.all = ECanaRegs.CANMC.all;
//工作在正常模式
ECanaShadow.CANMC.bit.STM = 0;
//工作在 eCAN 模式
ECanaShadow.CANMC.bit.SCB = 1;
ECanaRegs.CANMC.all = ECanaShadow.CANMC.all;
EDIS;

//初始化所有主设备控制区域为 0，控制区域所有的位都初始化为 0
ECanaMboxes.MBOX0.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX1.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX2.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX3.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX4.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX5.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX6.MSGCTRL.all = 0x00000000;
```

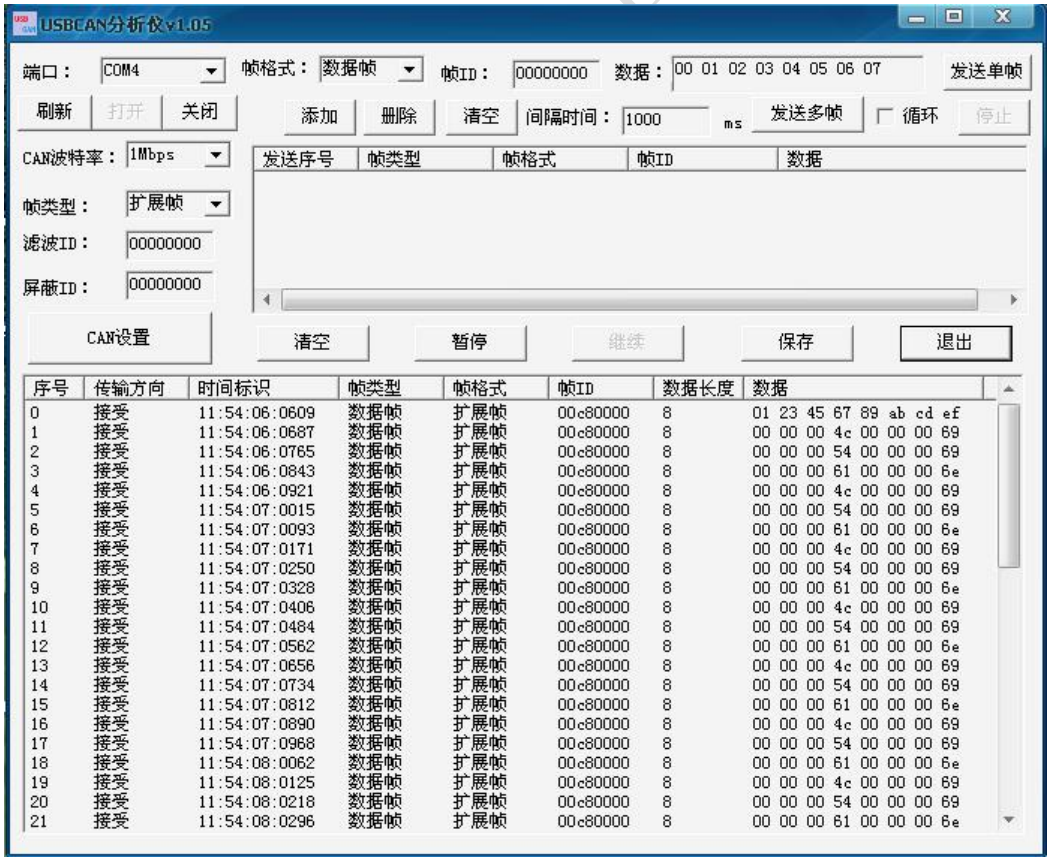
```
ECanaMboxes.MBOX7.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX8.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX9.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX10.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX11.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX12.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX13.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX14.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX15.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX16.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX17.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX18.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX19.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX20.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX21.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX22.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX23.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX24.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX25.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX26.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX27.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX28.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX29.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX30.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX31.MSGCTRL.all = 0x00000000;

//清除所有的 TA 位
ECanaRegs.CANTA.all = 0xFFFFFFFF;

//清除所有的 RMP 位
 ECanaRegs.CANRMP.all = 0xFFFFFFFF;
```

```
    ECanaRegs.CANAA.all= 0xFFFFFFFF;
    ECanaRegs.CANGIF0.all=0xFFFFFFFF;

    //配置时钟参数
    EALLOW;
    ECanaShadow.CANMC.all = ECanaRegs.CANMC.all;
    ECanaShadow.CANMC.bit.CCR = 1;
    ECanaRegs.CANMC.all = ECanaShadow.CANMC.all;
    EDIS;

    //CPU 请求向 CANBTC 和 CANGAM 写配置信息，该位置 1 后必须等到 CANED.CCE 为 1，才能
    //对 CANBTC 进行操作。
    do
    {
        ECanaShadow.CANES.all = ECanaRegs.CANES.all;
    }
    while(ECanaShadow.CANES.bit.CCE != 1);

    EALLOW;
    //(BRPREG+1)=10 ， CAN 时钟为 15MHz
    ECanaShadow.CANBTC.bit.BRPREG = 9;     //15
     //CAN 通信的波特率为 1MHz
    ECanaShadow.CANBTC.bit.TSEG2REG = 2;
    ECanaShadow.CANBTC.bit.TSEG1REG = 10;
    ECanaRegs.CANBTC.all = ECanaShadow.CANBTC.all;

     //CPU 请求正常操作
    ECanaShadow.CANMC.all = ECanaRegs.CANMC.all;
    ECanaShadow.CANMC.bit.CCR = 0;
```

```
ECanaRegs.CANMC.all = ECanaShadow.CANMC.all;
EDIS;

do
{
 ECanaShadow.CANES.all = ECanaRegs.CANES.all;
}
while(ECanaShadow.CANES.bit.CCE != 0);          //等待 CCE 位清零

//屏蔽所有邮箱，在写 MSGID 之前要完成该操作
ECanaRegs.CANME.all = 0;



 ECanaShadow.CANME.all=ECanaRegs.CANME.all;
 ECanaShadow.CANME.bit.ME0=0;
 ECanaRegs.CANME.all=ECanaShadow.CANME.all;

//设置发送邮箱的 ID 号，扩展帧
ECanaMboxes.MBOX0.MSGID.all = 0x80C80000;

//邮箱 0 为 TX
ECanaShadow.CANMD.all = ECanaRegs.CANMD.all;
ECanaShadow.CANMD.bit.MD0 =0;
ECanaRegs.CANMD.all = ECanaShadow.CANMD.all;

//数据长度  8 个 BYTE
ECanaMboxes.MBOX0.MSGCTRL.bit.DLC = 8;

//设置发送优先级
//  ECanaMboxes.MBOX0.MSGCTRL.bit.TPL = 0;
```

```
    //没有远方应答帧被请求
    ECanaMboxes.MBOX0.MSGCTRL.bit.RTR = 0;

    //向邮箱 RAM 区写数据
    ECanaMboxes.MBOX0.MDL.all = 0x01234567;
    ECanaMboxes.MBOX0.MDH.all = 0x89ABCDEF;

     //邮箱使能 Mailbox0
    ECanaShadow.CANME.all = ECanaRegs.CANME.all;
    ECanaShadow.CANME.bit.ME0 =1;
    ECanaRegs.CANME.all = ECanaShadow.CANME.all;
}
```

上位机软件 CAN 分析仪显示的结果如下图所示



从图中可以看出，第一帧接收到的数据是初始值 0x0123456789ABCDEF；第二帧为 0x0000004c00000069,其中 0x4c 是字母 L 的 ASCII 码，0x69 是字母 i 的 ASCII

码；第三帧为 0x0000005400000069，其中 0x54 是字母 T 的 ASCII 码，0x69 是字母 i 的 ASCII 码；第四帧为 0x000000610000006e，其中 0x61 是字母 a 的 ASCII 码，0x6e 是字母 n 的 ASCII 码；2~4 帧连接起来就是字符串“LiTian”，之后的数据帧就是 2~4 帧的循环出现，结果完全和实验要求一致。

2、eCAN 接收数据实验

实验目的：初步了解 DSP2812 内部 eCAN 接收数据的操作。

实验现象：在 CCS 的数据窗口显示 eCAN 接收的数据。

实验原理：通过对 eCAN 相关寄存器的配置，启动 eCAN 的中断。通信的波特率为 1Mbps，帧格式采用扩展帧，邮箱采用 16 号邮箱作为接收邮箱，并采用中断的方式来接收数据，中断使用 ECAN0INT 中断线。CAN 调试器给 DSP 发送的数据是“00 01 02 03 04 A5 B6 C7”，然后在中断服务子程序出设置断点，通过 CCS 的数据窗口观察邮箱 16 所接收到的数据。

注意事项：本实验需要在开发板外围接 CAN 模块，同时需要上位机软件才能实现实验描述的要求。

实验代码：

主程序代码如下：

```
void main(void)
{
    /*初始化系统*/
    InitSysCtrl();

    /*初始化 ecan 寄存器*/
     InitECan();

     //使能 PIE 中断
    PieCtrlRegs.PIEIER9.bit.INTx5 = 1;

    //使能 CPU 中断
    IER |= M_INT9;

    EINT;    //开全局中断
    ERTM;  //开实时中断

     for(;;)
    {
```

```
    }

}
```

eCAN 接收数据初始化代码如下：

```
void InitECan(void)
{
    struct ECAN_REGS ECanaShadow;

    //配置 GPIO 引脚工作在 eCAN 功能
    EALLOW;
    GpioMuxRegs.GPFMUX.bit.CANRXA_GPIOF7=1;
    GpioMuxRegs.GPFMUX.bit.CANTXA_GPIOF6=1;

    //配置 eCAN 的 RX 和 TX 分别为 eCAN 的接收和发送引脚
    ECanaShadow.CANTIOC.all = ECanaRegs.CANTIOC.all;
    ECanaShadow.CANTIOC.bit.TXFUNC = 1;
    ECanaRegs.CANTIOC.all = ECanaShadow.CANTIOC.all;

    ECanaShadow.CANRIOC.all = ECanaRegs.CANRIOC.all;
    ECanaShadow.CANRIOC.bit.RXFUNC = 1;
    ECanaRegs.CANRIOC.all = ECanaShadow.CANRIOC.all;
    EDIS;


    EALLOW;
    ECanaShadow.CANMC.all = ECanaRegs.CANMC.all;
    //工作在非测试模式
    ECanaShadow.CANMC.bit.STM = 0;
    //工作在 ecan 模式
    ECanaShadow.CANMC.bit.SCB = 1;
    ECanaRegs.CANMC.all = ECanaShadow.CANMC.all;
    EDIS;
```

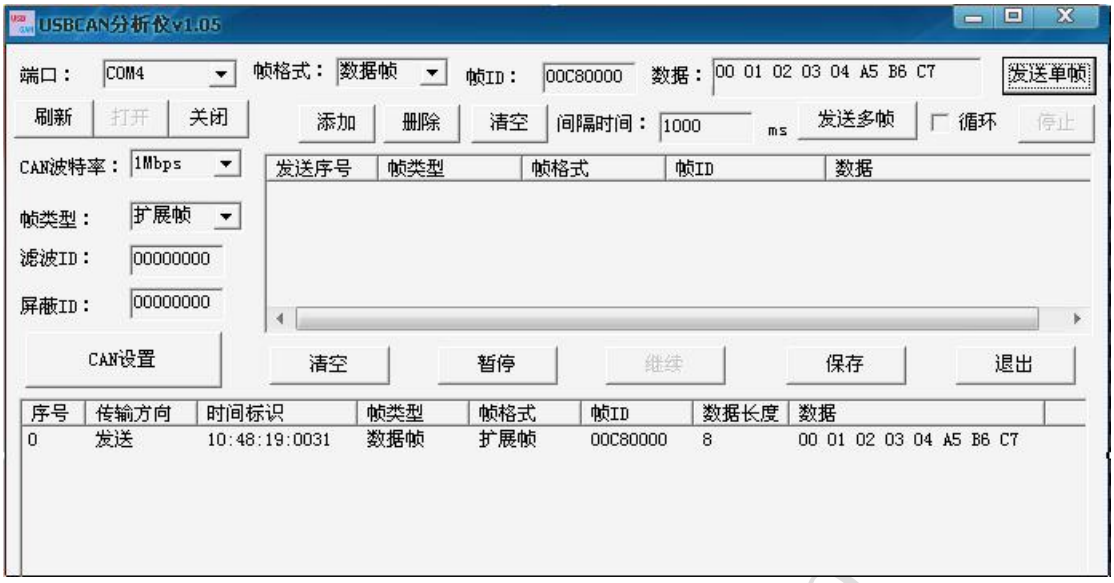
```
//初始化所有主设备控制区域为 0，控制区域所有的位都初始化为 0
ECanaMboxes.MBOX0.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX1.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX2.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX3.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX4.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX5.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX6.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX7.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX8.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX9.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX10.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX11.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX12.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX13.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX14.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX15.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX16.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX17.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX18.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX19.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX20.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX21.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX22.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX23.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX24.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX25.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX26.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX27.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX28.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX29.MSGCTRL.all = 0x00000000;
```

```
ECanaMboxes.MBOX30.MSGCTRL.all = 0x00000000;
ECanaMboxes.MBOX31.MSGCTRL.all = 0x00000000;

//清除所有的 TA 位
ECanaRegs.CANTA.all = 0xFFFFFFFF;

    //清除所有的 RMP 位
ECanaRegs.CANRMP.all = 0xFFFFFFFF;

//清除所有的中断标志位
ECanaRegs.CANGIF0.all = 0xFFFFFFFF;
ECanaRegs.CANGIF1.all = 0xFFFFFFFF;

//配置时钟参数
EALLOW;
ECanaShadow.CANMC.all = ECanaRegs.CANMC.all;
ECanaShadow.CANMC.bit.CCR = 1;
ECanaRegs.CANMC.all = ECanaShadow.CANMC.all;
EDIS;
 //CPU 请求向 CANBTC 和 CANGAM 写配置信息，该位置 1 后必须等到 CANED.CCE 为 1，才能
//对 CANBTC 进行操作。
do
{
  ECanaShadow.CANES.all = ECanaRegs.CANES.all;
}
while(ECanaShadow.CANES.bit.CCE != 1);

EALLOW;
//(BRPREG+1)=10 ， CAN 时钟为 15MHz
ECanaShadow.CANBTC.bit.BRPREG = 9;
```

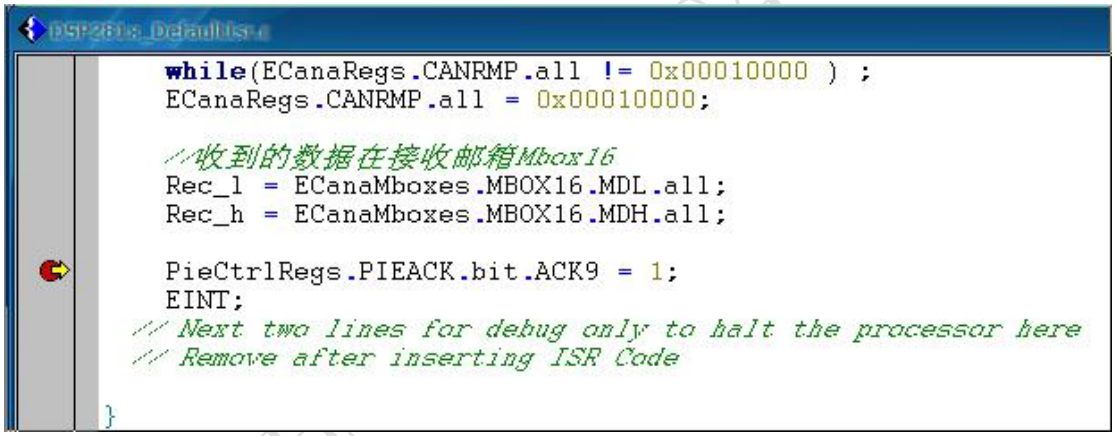
```
//CAN 通信的波特率为 1MHz
ECanaShadow.CANBTC.bit.TSEG2REG = 2;
ECanaShadow.CANBTC.bit.TSEG1REG = 10;
ECanaRegs.CANBTC.all = ECanaShadow.CANBTC.all;

//CPU 请求正常操作
ECanaShadow.CANMC.all = ECanaRegs.CANMC.all;
ECanaShadow.CANMC.bit.CCR = 0;
ECanaRegs.CANMC.all = ECanaShadow.CANMC.all;
EDIS;
do
{
   ECanaShadow.CANES.all = ECanaRegs.CANES.all;
}
while(ECanaShadow.CANES.bit.CCE != 0);

//屏蔽所有邮箱，在写 MSGID 之前要完成该操作
ECanaRegs.CANME.all = 0;

 //设置接收邮箱的 ID，扩展帧
ECanaMboxes.MBOX16.MSGID.all =    0x80C80000;

//设置邮箱 16 为接收邮箱
ECanaShadow.CANMD.all = ECanaRegs.CANMD.all;
 ECanaShadow.CANMD.bit.MD16 =1;
ECanaRegs.CANMD.all = ECanaShadow.CANMD.all;

//数据长度  8 个 BYTE
ECanaMboxes.MBOX16.MSGCTRL.bit.DLC = 8;

//没有远方应答帧被请求*/
```

```
    ECanaMboxes.MBOX16.MSGCTRL.bit.RTR = 0;

    //邮箱使能
    ECanaShadow.CANME.all = ECanaRegs.CANME.all;
    ECanaShadow.CANME.bit.ME16 =1;
    ECanaRegs.CANME.all = ECanaShadow.CANME.all;

    //邮箱中断使能
     EALLOW;
     ECanaRegs.CANMIM.all = 0xFFFFFFFF;
     //邮箱中断将产生在 ECAN0INT
     ECanaRegs.CANMIL.all = 0;
     ECanaRegs.CANGIF0.all = 0xFFFFFFFF;
     //ECAN0INT 中断请求线被使能
     ECanaRegs.CANGIM.bit.I0EN = 1;
     EDIS;


}
```

将 DSP2812 上的 CAN 接口同 CAN 调试器的接口通过导线连接好，然后运行此程序，在中断服务子程序的最后一行代码处设置断点，然后设置 CAN 调试工具，其软件的参数设置如图下图，然后点击“发送”按钮。

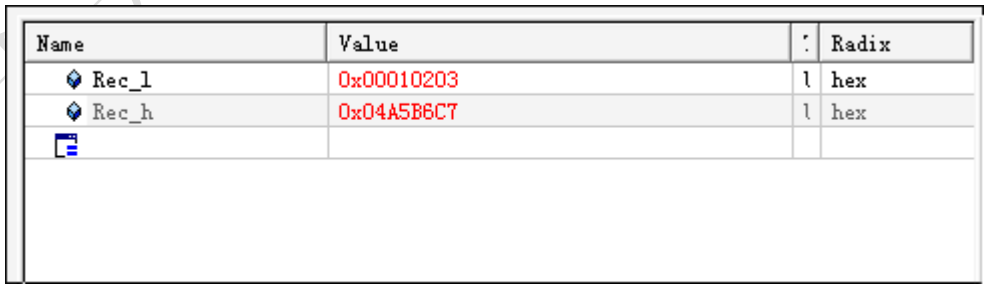


CAN 调试工具将数据发出以后，DSP 的程序将在中断服务子程序的断点处暂停下来，说明 DSP 接收到了数据。



```
    while(ECanaRegs.CANRMP.all != 0x00010000 ) ;
    ECanaRegs.CANRMP.all = 0x00010000;

    //收到的数据在接收邮箱Mbox16
    Rec_l = ECanaMboxes.MBOX16.MDL.all;
    Rec_h = ECanaMboxes.MBOX16.MDH.all;

    PieCtrlRegs.PIEACK.bit.ACK9 = 1;
    EINT;
  // Next two lines for debug only to halt the processor here
  // Remove after inserting ISR Code

}
```

然后将变量 Rec_l 和 Rec_h 添加到 Watch Window 中，观察 DSP 所接收到的数据，结果如下图所示。



Name	Value	Radix
Rec_l	0x00010203	hex
Rec_h	0x04A5B6C7	hex

从图中可以看出，变量 Rec_l 的值是 0x00010203，变量 Rec_h 的值是 0x04A5B6C7，结果完全正确。

3、ADC 数据采集实验

实验目的：初步了解 DSP2812 内部 ADC 采集信号的操作。

实验现象：将 AD 转换的结果以二进制的形式通过 8 位 LED 显示出来。

实验原理：采用顺序采样的模式对 ADCINA0 和 ADCINB0 接口的模拟信号进行采样，采用软件置位的方式启动采样。在采样中断函数中读出采样的结果，同时将 ADCINA0 的采样结果高 8 位的二进制形式显示在 8 位 LED 上。

注意事项：本实验中采集的信号是直接由 LT-Easy2812 开发板提供的，开关 SW0，SW1 默认是拨到右侧，此时 SW2 如果拨到右侧，则 ADCINA0 和 ADCINB0 接口输入的是一个固定信号，分别为 2.2v 和 1.1v，SW2 如果拨到左侧，则 ADCINA0 和 ADCINB0 接口输入信号由电位器 RP1 调节控制。

实验代码：

主程序：

```
void main(void)
{
    InitSysCtrl();  // 系统初始化子程序，在 DSP281x_sysctrl.c 中

    Init_LED();
    InitAdc(); //初始化 AD 模块

    PieCtrlRegs.PIEIER1.bit.INTx6 =1; //使能 PIE 模块中的 AD 采样中断

    IER|=M_INT1;  //开 CPU 中断

    EINT;  //使能全局中断
    ERTM;  //使能实时中断

    while(1);

}
```

中断处理程序：

```
    interrupt void  ADCINT_ISR(void)        // ADC
{
    //读取转换结果
    ADC_Result[0]=((float)AdcRegs.ADCRESULT0)*3.0/65520.0;          // 保存 ADCINA0 的结果
    ADC_Result[1]=((float)AdcRegs.ADCRESULT1)*3.0/65520.0;          // 保存 ADCINA1 的结果
/*  ADC_Result[2]=((float)AdcRegs.ADCRESULT2)*3.0/65520.0;          // 保存 ADCINA2 的结果
    ADC_Result[3]=((float)AdcRegs.ADCRESULT3)*3.0/65520.0;          // 保存 ADCUNA3 的结果
    ADC_Result[4]=((float)AdcRegs.ADCRESULT4)*3.0/65520.0;          // 保存 ADCINA4 的结果
    ADC_Result[5]=((float)AdcRegs.ADCRESULT5)*3.0/65520.0;          // 保存 ADCINA5 的结果
    ADC_Result[6]=((float)AdcRegs.ADCRESULT6)*3.0/65520.0;          // 保存 ADCINA6 的结果
    ADC_Result[7]=((float)AdcRegs.ADCRESULT7)*3.0/65520.0;          // 保存 ADCINA7 的结果
    ADC_Result[8]=((float)AdcRegs.ADCRESULT8)*3.0/65520.0;          // 保存 ADCINB0 的结果
    ADC_Result[9]=((float)AdcRegs.ADCRESULT9)*3.0/65520.0;          // 保存 ADCINB1 的结果
    ADC_Result[10]=((float)AdcRegs.ADCRESULT10)*3.0/65520.0;    // 保存 ADCINB2 的结果
    ADC_Result[11]=((float)AdcRegs.ADCRESULT11)*3.0/65520.0;    // 保存 ADCINB3 的结果
    ADC_Result[12]=((float)AdcRegs.ADCRESULT12)*3.0/65520.0;    // 保存 ADCINB4 的结果
    ADC_Result[13]=((float)AdcRegs.ADCRESULT13)*3.0/65520.0;    // 保存
```

ADCINB5 的结果

ADC_Result[14]=((float)AdcRegs.ADCRESULT14)*3.0/65520.0; // 保 存 ADCINB6 的结果

ADC_Result[15]=((float)AdcRegs.ADCRESULT15)*3.0/65520.0; // 保 存 ADCINB7 的结果

*/

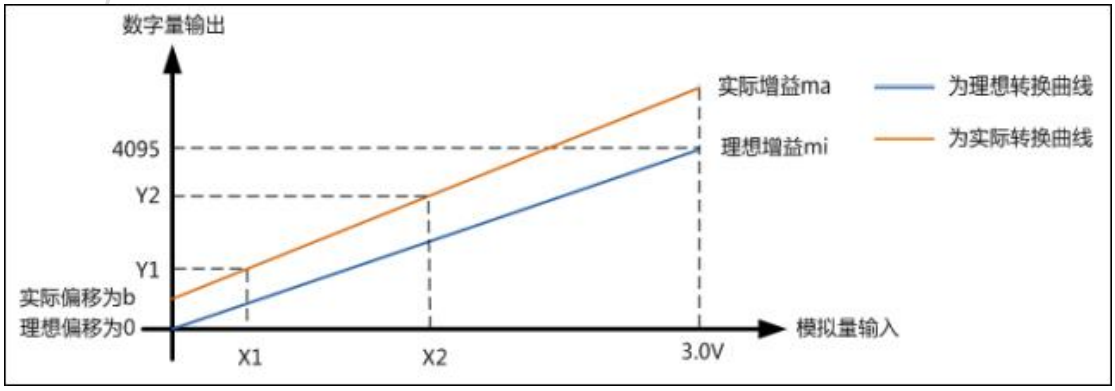
```
ADC_LED(AdcRegs.ADCRESULT0);
ADC_LED(AdcRegs.ADCRESULT0);
PieCtrlRegs.PIEACK.bit.ACK1=1;  //响应 PIE 同组中断
AdcRegs.ADCST.bit.INT_SEQ1_CLR=1; //清除 AD 中断的标志位
AdcRegs.ADCTRL2.bit.SOC_SEQ1=1;  //立即启动下一次转换
EINT;  //使能全局中断
}
```

4、ADC 校正实验

实验目的：初步了解 DSP2812 内部 ADC 采集校正的操作。

实验现象：在 CCS 的数据窗口中可以看到计算出来的校正的增益值和偏移值。

实验原理：通过 DSP2812 内部 12 位的 ADC 模块，在实际的应用过程中，由于外界干扰或者噪声等因素，采样值和实际值之间存在一定的误差。为了减小误差，保证采样结果结果的精确，采用的校正的算法可以比较好的达到要求。AD 转换的曲线如下图所示：



从上图可以看出，理想的 12 位 ADC 模拟量输入 X 和数字量 Y 之间的关系应该是：

$$Y=mi*X$$

理想曲线上有有一个固定点坐标是（3.0,4095），因此可以理想情况下增益为：

$$mi=4095/3.0=1365$$

实际应用中是存在增益误差和偏移误差的，假设实际增益为 ma,实际偏移量为 b，则模拟量与数字量之间的关系为：

$$Y=ma*X+b$$

这个式子是二元一次方程，要得出未知量 ma,b 的值，只需要两个这样的方程组成一个二元一次方程组，就可以求解

$$Y1=ma*X1+b$$

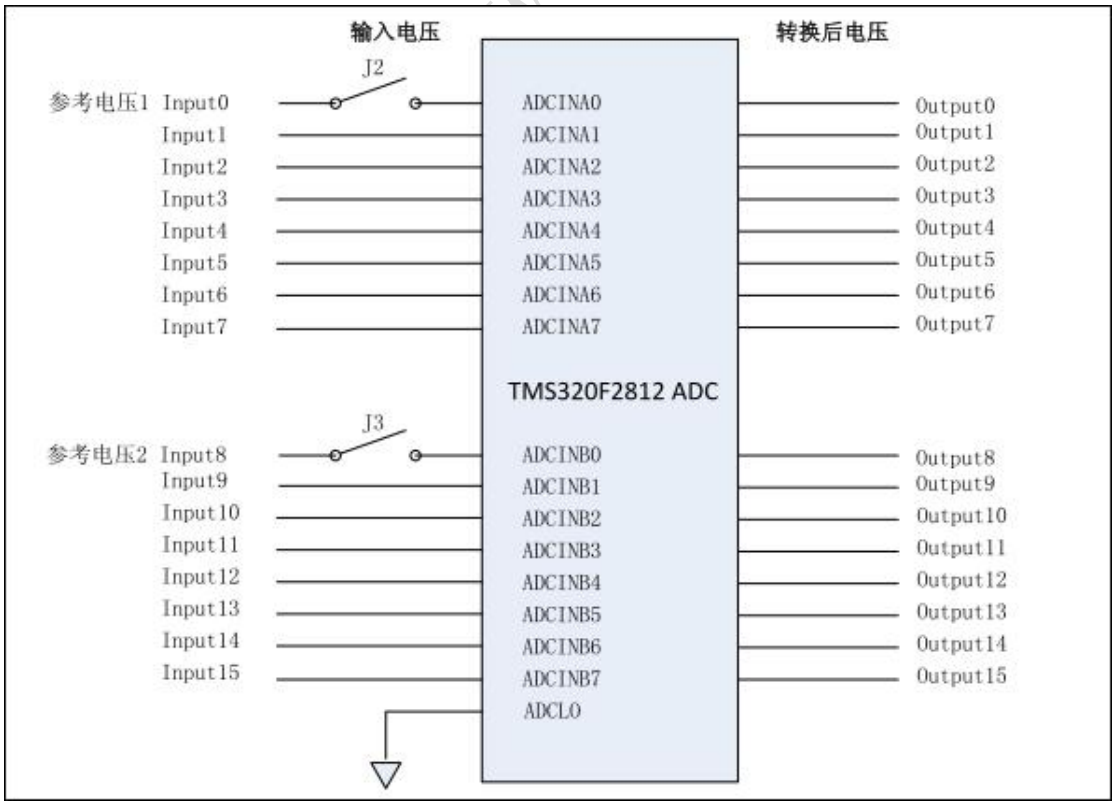
$$Y2=ma*X2+b$$

其中 X1，X2 为已知的输入模拟量，Y1，Y2 输出的数字量，有这个方程组可以推导出

$$ma= (Y2 -Y1)/ (X2 -X1)$$

$$b=(X2Y1 - X1Y2)/ (X2 -X1)$$

本实验中，在 ADCINA0，ADCINB 接入固定的模拟量分别为 X2，X1，启动 ADC,就能得到数字量 Y2，Y1，代如上面的公式，就可以得到增益 ma 和偏移 b



在 LT-Easy2812 开发板中 J2，J3 所对应的开关丝印表示为 SW0，SW1。当

开关 SW0，SW1 闭合时，参考电压就加在了模拟输入接口上，此时校正功能使能，其他的 14 路通道可通过程序自由配置；当开关断开时，校正功能禁止，此时 16 路通道可通过程序自由配置。

本实验中，AD 采样频率为 10K，采样模式采用顺序采样。利用通用定时器 T1 的周期中断事件来启动 AD 转换。ADCINA0 和 ADCINB0 为参考电平，实际的电压值为 2.0 和 1.0，由于每个板子的具体特性稍有不同，请用万用表自行测量参考电压值。本程序对 ADCINA0、ADCINB02 个通道进行连续 10 次的采样，然后对各个通道的 10 个采样值进行排序、滤波，最后求取平均值。然后由 ADCINA0 和 ADCINB0 通道的值计算求得 CalGain 和 CalOffset。

注意事项：本实验中的校正参考电压是直接由 LT-Easy2812 开发板提供的，校正的过程中开关 SW0，SW1 ，SW2 必须都拨到右侧（默认是在右侧），此时 ADCINA0 和 ADCINB0 接口输入的是一个固定信号，分别为 2.2v 和 1.1v，作为校准的输入。

实验代码：

主程序：

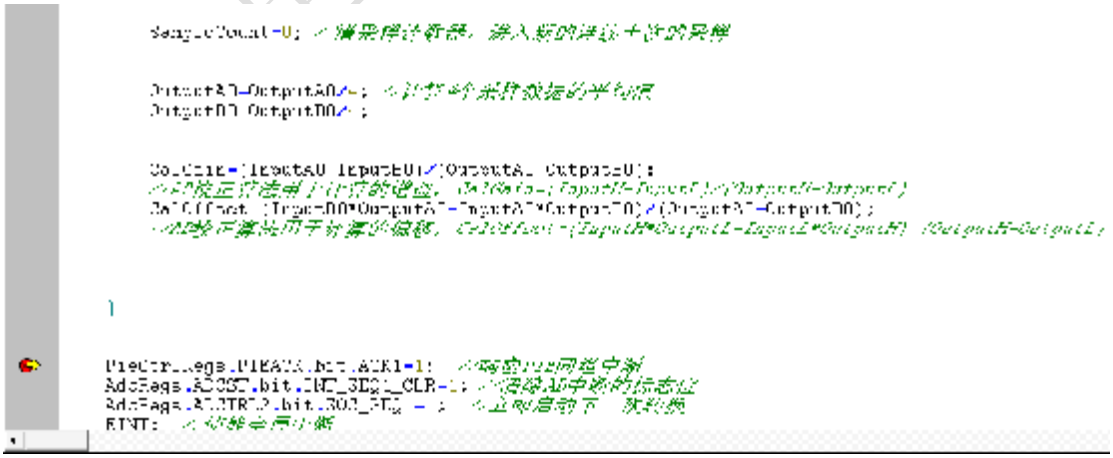
```
void main(void)
{
    int i;

    InitSysCtrl();  //初始化系统函数
     InitEv();
    InitAdc();

   InputA0=2.2;
   InputB0=1.1;

   OutputA0=0;
   OutputB0=0;


   for(i=0;i<10;i++)
```

```
    {
       ADC_ResultA0[i]=0;
       ADC_ResultB0[i]=0;

    }

    SampleCount=0;

    PieCtrlRegs.PIEIER1.bit.INTx6 =1;
    PieCtrlRegs.PIEIER2.bit.INTx4=1;

     IER|=M_INT1;   //开 CPU 中断
     IER|=M_INT2;

     EINT;
     ERTM;
     EvaRegs.T1CON.bit.TENABLE=1;    //启动 T1 计数

    for(;;)
    {

    }
}
     中断处理程序
     interrupt void   ADCINT_ISR(void)        // ADC
{

    //读取转换结果
    ADC_Result[0]=((float)AdcRegs.ADCRESULT0)*3.0/65520.0;          // 保 存
ADCINA0 的结果
```

```
ADC_Result[1]=((float)AdcRegs.ADCRESULT1)*3.0/65520.0;        // 保存ADCINA1 的结果

ADC_ResultA0[SampleCount]=ADC_Result[0];
ADC_ResultB0[SampleCount]=ADC_Result[1];

SampleCount++; //采样计数器计数

 if(SampleCount==10) //采样十次之后，需要进行滤波处理
{
    int i;
    i=0;

    OutputA0=0;
    OutputB0=0;


    sequence(ADC_ResultA0,10); //对采样十次得到的数据进行排序
    sequence(ADC_ResultB0,10);


    for(i=3;i<7;i++)
    {
        OutputA0=OutputA0+ADC_ResultA0[i]; //中值滤波法
        OutputB0=OutputB0+ADC_ResultB0[i]; //取中间的 4 个数据进行求和
    }

    SampleCount=0; //清采样计数器，进入新的连续十次的采样
```

```
        OutputA0=OutputA0/4; //计算 4 个采样数据的平均值
        OutputB0=OutputB0/4;


        CalGain=(InputA0-InputB0)/(OutputA0-OutputB0);
        //AD 校正算法用于计算的增益，
        CalGain=(InputH-InputL)/(OutputH-OutputL)
        CalOffset=(InputB0*OutputA0-InputA0*OutputB0)/(OutputA0-OutputB0);
        //AD 校正算法用于计算的偏移，
        CalOffset=(InputH*OutputL-InputL*OutputH)/(OutputH-OutputL)

    }

    PieCtrlRegs.PIEACK.bit.ACK1=1;   //响应 PIE 同组中断
    AdcRegs.ADCST.bit.INT_SEQ1_CLR=1; //清除 AD 中断的标志位
    AdcRegs.ADCTRL2.bit.SOC_SEQ1=1;   //立即启动下一次转换
    EINT;   //使能全局中断
}
```

可以在程序中设置断点，如下图所示：



然后将变量 CalGain 和 CalOffset 添加到数据观察窗口中，当程序执行到断点处时，可以观察到计算出来的 CalGain 和 CalOffset 的值，如下图所示：

Name	Value	T	Radix
CalGain	0.9983379	1	float
CalOffset	-0.03803185	1	float