程序介绍

1. 使用接线

例程的具体引脚分配与接线方式请查看 **24 位 ADC 模块与 STM32 开发板引脚连接说明.xlsx**，结合实际源码内容查看。

2. 实验操作

野火 F1、F4、F7 和 F7 开发板及电机板：

1.开发板掉电情况下，将模块引出脚按照 xxx 引脚连接说明.xlsx，接到板子对应脚上

2.确保引脚对应接上的情况下，开发板上电，将例程编译成功后，下载到开发板里

3.开发板的 USB 转串口通过数据线连接电脑，同时确保电脑安装了串口驱动并能识别到开发板的串口

4.电脑端使用串口调试助手，选择电脑与开发板相连的 COM 口，设置为 115200-N-8-1 并打开

5.复位开发板，即可接收开发板串口发送给电脑的数据

6.按照配置的模式，接对应通道进行采集

注意事项

1.在 ad7192_test.h 切换 MODE 宏定义来选择模式

2.在 bsp_ad7192.h 修改 FSPEED 的值控制输出数据速率

3.4 路伪差分通道为 AIN1 和 AGND、AIN2 和 AGND、AIN3 和 AGND 以及 AIN4 和 AGND

4.2 路差分通道为 AIN1 和 AIN2 以及 AIN3 和 AIN4

5.电子秤实验使用 AIN1 和 AIN2 差分通道

6.芯片手册说明只能接受正电压，因此不能采集低于 0V 的负电压

7.请勿将模块输入通道的 AGND 错误地与 VCC 相连，此操作会损坏模块和被测设备

8.对于没有电位器的开发板，可直接将 AIN1、AIN2、AIN3、AIN4 连接到 3V3 或 GND 测量引脚电压。模块在使用时要与被测设备做共地处理

3. 例程说明

野火 STM32F103 霸道开发板例程举例：

1.例程初始化系统时钟，HAL 库初始化，LED 端口初始化，串口 1 初始化， 按键初始化

2.初始化 AD7192 后将模块软件复位一下

3.使用 ReadFromAD7192ViaSPI 函数对模块连接状态检测

4.根据 MODE 宏定义来选择模式进行运行

首先我们先讲解一下 ReadFromAD7192ViaSPI 函数，这个函数的功能是读取 AD7192 内部寄存器的内容，我们可以看到它的第一个输入参数 RegisterStartAddress 指的就是寄存器地址。 其他输入参数的意思在代码里已经注明里，在这里主要讲解一下这个函数的整体思路。

该函数首先定义两个读写缓存数组 WriteBuf 和 ReadBuf，它们都是 4 个字节 32 位的，根据 AD7192 数据手册等资料我们知道，AD7192 片内有一个 8 位的通信寄存器和其他 8 个长度不一的寄存器， 而读取这 8 个长度不一的寄存器当中的那个最长的寄存器需要 32 位也就是 4 个字节的空间来保存读取到的数据，所以这两个数组的大小都是 4 个字节

然后，写入通信寄存器，这是必要的步骤，因为与 AD7192 的所有通信都必须以对通信寄存器的写操作开始，写入通信寄存器的数据决定了下一个操作是对哪一个寄存器进行读操作还是写操作。 接着因为该函数是读寄存器数据，不需要写入数据，所以将 WriteBuf 数组的全部 4 个字节都赋值为 NOP，也就是 0。在 switch 语句里面判断要读取的是哪一个寄存器， 因为这 8 个寄存器长度不一，所以实际上要根据寄存器长度分成三种情况：8 位寄存器（REG_ID、REG_COM_STA、REG_GPOCON）、24 位寄存器（REG_MODE、REG_CONF、REG_OFFSET、REG_FS）、 还有数据寄存器（REG_DATA），数据寄存器本是 24 位的寄存器，但在使用多通道的时候读取时会附加上状态寄存器的内容来指定是哪个通道的数据，所以读取时可能读取到的不止是 24 位的数据。 具体请参考数据手册里的相关内容

bsp_ad7192.c

```
/**
  * @brief   通过 SPI 对 AD7192 执行一次读操作函数
  * @param   RegisterStartAddress ： 要读寄存器的起始地址(取值范围是从 0x00——0x07)
  * @param   NumberOfRegistersToRead ：  要读寄存器的个数
  * @param   DataBuffer ：  要读入的值
  * @param   OffsetInBuffer ：  缓存内偏移
  * @retval
  */
unsigned char ReadFromAD7192ViaSPI(const unsigned char RegisterStartAddress, const unsigned char NumberOfRegistersToRead, uint32_t *DataBuffer, const unsigned char OffsetInBuffer)
{
//形参包括要读寄存器的起始地址 RegisterStartAddress(取值范围是从 0x00——0x07)，要读取寄存器的个数，指向将读取 AD7192 寄存器数据存入的数组的指针(DataBuffer 才是要读入的值其他是中间变量)，
//一般指向 AD7192Registers[8]，
```

```
 //const unsigned char OffsetInBuffer，字面意思是缓存内偏移，是指AD7192Registers[8]数组内部偏移，注意是数组哦，之前我们说过AD7192Registers[8]之所以定义8 个元素，
 //一个寄存器对应AD7192Registers[8]数组的一个元素，互不干扰。

   unsigned char WriteBuf[4]={0,0,0,0}; //定义有4 个元素的写缓存数组，每个数组元素占1 个字节。
   unsigned char ReadBuf[4]={0,0,0,0};  //定义有4 个元素的读缓存数组，每个数组元素占1 个字节。
   unsigned char i;


   //Delay(0xFFFF);
   for(i=0; i < NumberOfRegistersToRead; i++)
   {
     WriteBuf[0] = WEN|RW_R|((RegisterStartAddress + i)<<3)|CREAD_DIS;        //写入通信寄存器;8位数据;下一个操作是对指定寄存器执行读操作。CREAD_DIS 表示不使能连续读。
     //确定下一步进行寄存器读操作，那么写操作自然无效喽。
     AD7192readdata(WriteBuf, ReadBuf, 1);//首先通过写入通信寄存器来选定下一步要读取的寄存器
                                          //然后再将WriteBuf 清空
     WriteBuf[0] = NOP;
     WriteBuf[1]     = NOP;
     WriteBuf[2]     = NOP;
     WriteBuf[3]     = NOP;

     switch(RegisterStartAddress + i)
     {
       case REG_ID              :     //ID 寄存器(0x04，8 位寄存器)
       case REG_COM_STA :     //状态寄存器(0x00，8 位寄存器)
       case REG_GPOCON  :     //通用数字输出控制寄存器(0x05，8 位寄存器)
         AD7192readdata(WriteBuf, ReadBuf, 1);    //此3 种情况是读取一个字节
         DataBuffer[OffsetInBuffer + i ] =  ReadBuf[0];
         break;

       case REG_MODE    :     //模式寄存器(0x01，24 位)
       case REG_CONF    :     //配置寄存器(0x02，24 位)
       case REG_OFFSET :     //失调寄存器(0x06，24 位)
       case REG_FS              :     //满量程寄存器(0x07，24 位)
         AD7192readdata(WriteBuf, ReadBuf, 3);             //此4 种情况是读取3 个字节
         DataBuffer[OffsetInBuffer + i ] = ReadBuf[0];
         DataBuffer[OffsetInBuffer + i ] = (DataBuffer[OffsetInBuffer + i ]<<8) + ReadBuf[1];
         DataBuffer[OffsetInBuffer + i ] = (DataBuffer[OffsetInBuffer + i ]<<8) + ReadBuf[2];
         break;

       case REG_DATA  :     //数据寄存器(0x03，24 位或32 位)
         if (AD7192Registers[REG_MODE] & DAT_STA_EN)       //多通道使能，将状态寄存器的内容附加到数据寄存器24 位的数据上，所以是32 位数据
           {
           AD7192readdata(WriteBuf, ReadBuf, 4);         //所以此情况是读4 个字节
           DataBuffer[OffsetInBuffer + i ] = ReadBuf[0];
           DataBuffer[OffsetInBuffer + i ] = (DataBuffer[OffsetInBuffer + i ]<<8) + ReadBuf[1];
           DataBuffer[OffsetInBuffer + i ] = (DataBuffer[OffsetInBuffer + i ]<<8) + ReadBuf[2];
           DataBuffer[OffsetInBuffer + i ] = (DataBuffer[OffsetInBuffer + i ]<<8) + ReadBuf[3];
           break;
           }
         else
           {
           AD7192readdata(WriteBuf, ReadBuf, 3);        //do not transfer the status contents after read data register
           DataBuffer[OffsetInBuffer + i ] = ReadBuf[0];
           DataBuffer[OffsetInBuffer + i ] = (DataBuffer[OffsetInBuffer + i ]<<8) + ReadBuf[1];
           DataBuffer[OffsetInBuffer + i ] = (DataBuffer[OffsetInBuffer + i ]<<8) + ReadBuf[2];
           break;
           }

       default                          :
           break;
```

```
    }
  }

  return 0;
}
```

在 2 路差分通道连续读实验下，看 difference_continuous_conversion_voltage 这个函数其实非常简单。 它的功能是对 2 路差分通道：“AIN1_AIN2” 和 “AIN3_AIN4” 进行连续地 AD 转换，转换完成之后读取这两通道 ADC 转换结果并打印出来。 进入该函数里面首先会读取并打印 8 个 AD7192 片内寄存器的值，方便查看和调试，然后调用 ad7192_mode_cfg_reg 函数配置模式寄存器和配置寄存器， 进行内部校准，最后启动连续转换，开启连续读模式，在 while 循环里不断读取和打印两路差分通道的 ADC 电压值

ad7192_test.c

```
/**
  * @brief  2 路差分连续转换电压实验
  * @param  无
  * @retval 无
  */
void difference_continuous_conversion_voltage(void)
{
  uint32_t ad_data = 0;
  float v = 0.0;
  uint32_t mode = 0, cfg = 0;

  printf("野火 AD9172 2 路 差分连续转换读电压实验\r\n");

  /* 读 AD7192 寄存器 */
  ReadFromAD7192ViaSPI(REG_COM_STA, 8, AD7192Registers, REG_COM_STA);
  for(int i=0; i < 8; i++)
  {
    printf("AD7192Register[%d] = 0x%06X \r\n", i+REG_COM_STA ,
AD7192Registers[i+REG_COM_STA]);
  }

      /* 单次转换|使能状态传输|外部时钟|sinc4 滤波器|禁用奇偶校验|时钟不分频|禁用单周期转换|禁用 60Hz
陷波|输出数据速率 */
  mode =
MODE_SING|DAT_STA_EN|EXT_XTAL|SINC_4|ENPAR_DIS|CLK_DIV_DIS|SINGLECYCLE_DIS|REJ60_DIS|FSPEED;
  cfg  = CHOP_DIS|REF_IN1|AIN1_AIN2|BURN_DIS|REFDET_DIS|BUF_DIS|UB_BI|GAIN_1;
      /*禁用斩波|外部基准电压 1|12 差分通道(单)|禁用激励电流|禁用基准电压检测|禁用模拟输入缓冲|双极性
模式|增益为 1 */

  ad7192_mode_cfg_reg(mode, cfg);    // 配置模式寄存器和配置寄存器

  /* 校准 */
  printf("内部校准中\r\n");
  AD7192InternalZeroScaleCalibration();
  AD7192InternalFullScaleCalibration();
  printf("内部校准完成\r\n");

  AD7192StartContinuousConvertion(AIN1_AIN2|AIN3_AIN4);    // 启动连续转换
  AD7192StartContinuousRead();
  while(1)
  {
    ad_data = AD7192ContinuousRead();

    switch(ad_data & 7)
    {
```

```
      case 0:
        v = ((ad_data >> 8) / 8388608.0 - 1)*3.3;
        printf("AIN1_AIN2 = %fV\n", v);
      break;

      case 1:
        v = ((ad_data >> 8) / 8388608.0 - 1)*3.3;
        printf("AIN3_AIN4 = %fV\n", v);
      break;

    }
  }
}
```

在 4 路伪差分通道连续读实验调用的 single_continuous_conversion_voltage 函数其实也与 2 路差分连续读实验中的 difference_continuous_conversion_voltage 函数时差不多的。 只不过本实验启动的是 4 个通道的伪差分信号输入转换

ad7192_test.c

```
/**
  * @brief  4 路伪差分连续转换电压实验
  * @param  无
  * @retval 无
  */
void single_continuous_conversion_voltage(void)
{
  uint32_t ad_data = 0;
  float v = 0.0;
  uint32_t mode = 0, cfg = 0;

  printf("野火 AD9172 4 路连续转换读电压实验\r\n");

  /* 读 AD7192 寄存器 */
  ReadFromAD7192ViaSPI(REG_COM_STA, 8, AD7192Registers, REG_COM_STA);
  for(int i=0; i < 8; i++)
  {
    printf("AD7192Register[%d] = 0x%06X \r\n", i+REG_COM_STA ,
AD7192Registers[i+REG_COM_STA]);
  }

      /* 单次转换|使能状态传输|外部时钟|sinc4 滤波器|禁用奇偶校验|时钟不分频|禁用单周期转换|禁用 60Hz
陷波|输出数据速率*/
  mode = MODE_SING|DAT_STA_EN|EXT_XTAL|SINC_4|ENPAR_DIS|CLK_DIV_DIS|SINGLECYCLE_DIS|REJ60_DIS|
FSPEED;
  cfg  = CHOP_EN|REF_IN1|AIN1_AIN2|BURN_DIS|REFDET_DIS|BUF_DIS|UB_BI|GAIN_1;
      /*禁用斩波|外部基准电压 1|12 差分通道(单)|禁用激励电流|禁用基准电压检测|禁用模拟输入缓冲|双极性
模式|增益为 1 */

  ad7192_mode_cfg_reg(mode, cfg);    // 配置模式寄存器和配置寄存器

  /* 校准 */
  printf("内部校准中\r\n");
  AD7192InternalZeroScaleCalibration();
  AD7192InternalFullScaleCalibration();
  printf("内部校准完成\r\n");

  AD7192StartContinuousConvertion(AIN1_COM|AIN2_COM|AIN3_COM|AIN4_COM);    // 启动连续转换

  while(1)
  {
    ad_data = AD7192ContinuousRead();
```

```
    switch(ad_data & 7)
    {
      case 4:
        v = ((ad_data >> 8) / 8388608.0 - 1)*3.3;
        printf("AIN1_COM = %fV\n", v);
      break;

      case 5:
        v = ((ad_data >> 8) / 8388608.0 - 1)*3.3;
        printf("AIN2_COM = %fV\n", v);
      break;

      case 6:
        v = ((ad_data >> 8) / 8388608.0 - 1)*3.3;
        printf("AIN3_COM = %fV\n", v);
      break;

      case 7:
        v = ((ad_data >> 8) / 8388608.0 - 1)*3.3;
        printf("AIN4_COM = %fV\n", v);
      break;
    }
  }
}
```