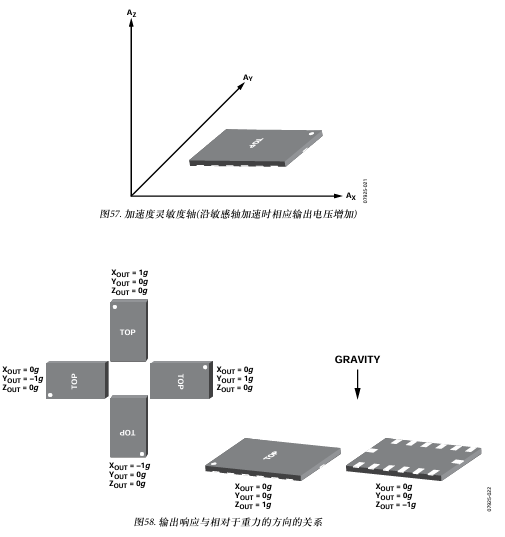
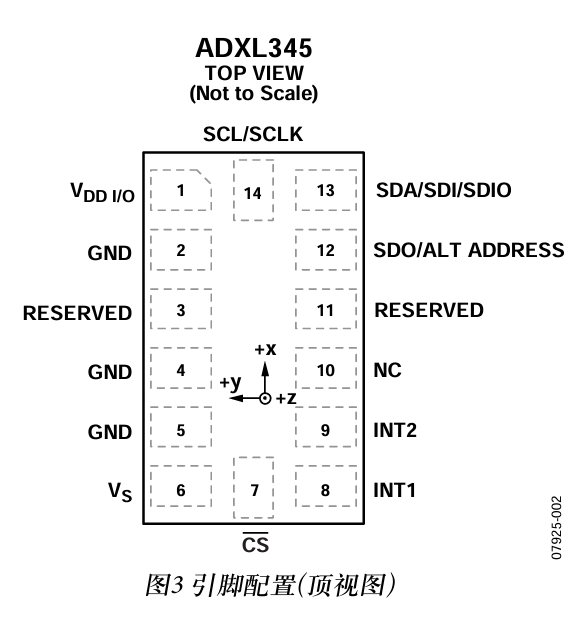
# ADXL345加速度传感器

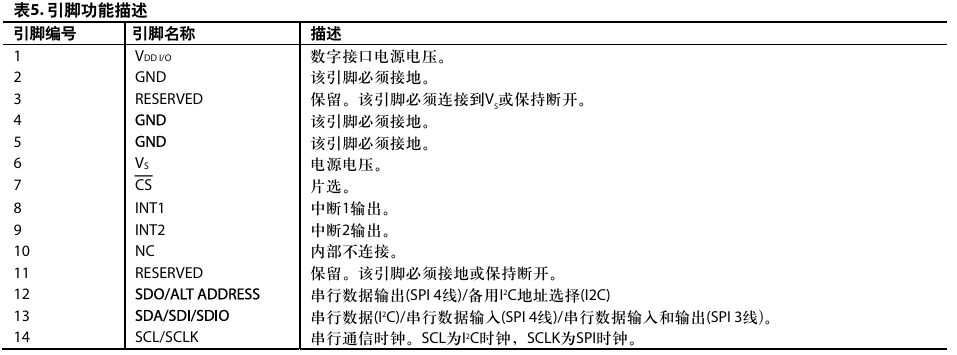
## 1.ADXL345简介

ADXL345是ADI公司生产的3轴加速度计，，分辨率 高(13位)，测量范围达±16 g。该器件提供多种特殊检测功能。活动和非活动检测功能通 过比较任意轴上的加速度与用户设置的阈值来检测有无运 动发生。敲击检测功能可以检测任意方向的单振和双振动 作。自由落体检测功能可以检测器件是否正在掉落。加速度与方向关系如下图所示：



该芯片可通过SPI或IIC协议通讯。引脚图如下所示：

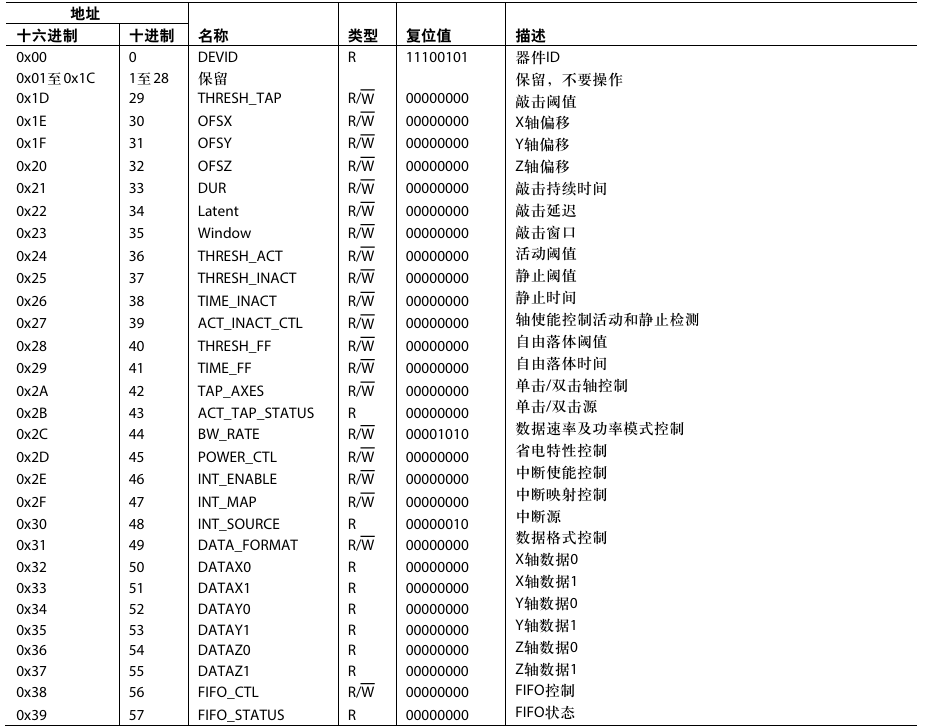




根据手册描述，CS引脚处于高电平时，选择IIC通信，S引脚处于低电平时，选择SPI通信。ALT ADDRESS引脚处于高电平时，芯片的7位IIC地址是0x1D，ALT ADDRESS引脚接地时，可以选择备用IIC地址0x53。

## 2.寄存器介绍

ADXL345的寄存器功能如下图所示：



篇幅有限，具体的寄存器使用方式详细请看手册。以采集三个方向的加速度为例，最小初始化方式仅需配置DATA\_FORMAT、POWER\_CTL、INT\_ENABLE寄存器。需要一提的是ADXL345的校准。将ADXL345水平放到桌面上，无偏移情况，X、Y、Z轴输出标准量应该是X=0LSB,Y=0LSB,Z=255LSB,但实际情况下会有一些偏差量，这个偏差量与标准量的差就是需要校准的。但ADXL345默认13位分辨率，16g量程时，1LSB = 16g/2^13=1/256=3.9mg/LSB，而偏移寄存器的分辨率时15.6mg/LSB（手册中说明），相比较分辨率更粗糙，也就是说，偏移寄存器写入的值需要是目标偏移量的1/4。

## 3.驱动例程

本例程使用的MCU为沁恒的CH32V307VCT6，频率72Mhz。

本例程使用硬件IIC通信，ALT ADDRESS引脚接地，ADXL345的7位地址为0x53。

硬件IIC的配置这里就不多概述，详细请看之后的例程。硬件IIC使用IIC2，频率100KHx,主机地址0x01。接着是配置ADXL345的寄存器，DATA\_FORMAT寄存器(0x31)配置为0x0B，表示禁用自测力，全分辨率，右对齐模式，g范围为±16 g。POWER\_CTL寄存器(0x2D)配置为0x28,表示链接使能，测量模式使能。INT\_ENABLE寄存器(0x2E)配置为0x00,关闭所有中断。OFSX、OFSY、OFSZ寄存器（0x1E、0x1F、0x20）配置为0x00,表示偏移校准为0，如果有需要，可以在这校准偏移。详细寄存器功能参照芯片手册。初始化程序如下：

void ADLX345\_Init()

{

    uint8\_t data1 = 0x0B;

    uint8\_t data3 = 0x28;

    uint8\_t data4 = 0x00;

    uint8\_t data5 = 0x00;

    uint8\_t data6 = 0x00;

    uint8\_t data7 = 0x00;

    IIC\_Init();//100kHz,使用IIC2

    IIC\_WriteLen(ADXL345\_ADDR,DATA\_FORMAT,1,&data1);

    IIC\_WriteLen(ADXL345\_ADDR,POWER\_CTL,1,&data3);//链接使能，测量模式

    IIC\_WriteLen(ADXL345\_ADDR,INT\_ENABLE,1,&data4);

    IIC\_WriteLen(ADXL345\_ADDR,OFSX,1,&data5);

    IIC\_WriteLen(ADXL345\_ADDR,OFSY,1,&data6);

    IIC\_WriteLen(ADXL345\_ADDR,OFSZ,1,&data7);

}

初始化完成后，ADXL345会自动获取加速度值存放到寄存器，所以我们需要通过读取寄存器的值来获取三轴加速度的状态。存放X轴方向加速度值得寄存器为DATAX0(0x32)和DATAX1(0x33), 存放Y轴方向加速度值得寄存器为DATAY0(0x34)和DATAY1(0x35), 存放Z轴方向加速度值得寄存器为DATAZ0(0x36)和DATAZ1(0x37)。获取加速度的例程如下：

void ADLX345\_GetData(uint8\_t \*data,uint8\_t num)

{

  IIC\_ReadLen(ADXL345\_ADDR, DATA\_X0, num, data);

}

void ADXL345\_TEST(void)

{

    int16\_t aaX =0,aaY = 0,aaZ = 0;

    uint8\_t xyzData[6] = {0};         //x、y、z轴加速度，均占两个字节

    ADLX345\_GetData(xyzData,6);

    aaX = (int16\_t)((xyzData[1]<<8)+xyzData[0]);    //x方向加速度

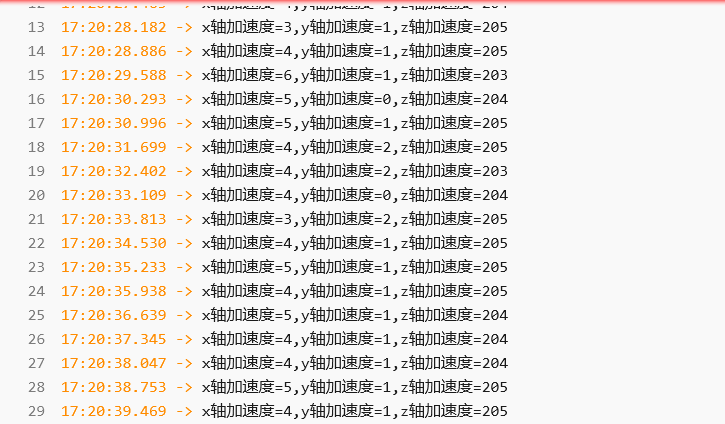
    aaY = (int16\_t)((xyzData[3]<<8)+xyzData[2]);    //y方向加速度

    aaZ = (int16\_t)((xyzData[5]<<8)+xyzData[4]);    //z方向加速度

printf("x轴加速度=%d,y轴加速度=%d,z轴加速度=%d\r\n",aaX,aaY,aaZ);

}

串口输出三轴加速度如下图所示**：**

****

**完整程序如下所示：**

**IIC.h**

#ifndef \_\_IIC\_H

#define \_\_IIC\_H

#include "debug.h"

#define USER\_IIC              I2C2

#define USER\_IIC\_CLK          RCC\_APB1Periph\_I2C2

#define USER\_IIC\_GPIO\_CLK     RCC\_APB2Periph\_GPIOB

#define USER\_IIC\_GPIO\_PORT    GPIOB

#define USER\_IIC\_SCL\_PIN      GPIO\_Pin\_10

#define USER\_IIC\_SDA\_PIN      GPIO\_Pin\_11

#define USER\_IIC\_BOUND        100000        //100KHz

#define USER\_IIC\_ADDR         0x01

void IIC\_Init();

u8 IIC\_WaitEvent(I2C\_TypeDef\* I2Cx, uint32\_t I2C\_EVENT);

u8 IIC\_ReadByte(u8 addr,u8 reg);

u8 IIC\_WriteLen(u8 addr, u8 reg, u8 len, u8 \*buf);

u8 IIC\_ReadLen(u8 addr, u8 reg, u8 len, u8 \*buf);

u8 IIC\_WriteByte(u8 addr,uint8\_t reg, uint8\_t data);

#endif

**IIC.c**

#include "IIC.h"

void IIC\_Init()

{

    GPIO\_InitTypeDef GPIO\_InitStructure;

    I2C\_InitTypeDef I2C\_InitTSturcture;

    RCC\_APB2PeriphClockCmd( USER\_IIC\_GPIO\_CLK , ENABLE );

    RCC\_APB1PeriphClockCmd( USER\_IIC\_CLK, ENABLE );

    GPIO\_InitStructure.GPIO\_Pin = USER\_IIC\_SCL\_PIN;

    GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AF\_OD;

    GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz;

    GPIO\_Init( USER\_IIC\_GPIO\_PORT, &GPIO\_InitStructure );

    GPIO\_InitStructure.GPIO\_Pin = USER\_IIC\_SDA\_PIN;

    GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AF\_OD;

    GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz;

    GPIO\_Init( USER\_IIC\_GPIO\_PORT, &GPIO\_InitStructure );

    I2C\_InitTSturcture.I2C\_ClockSpeed = USER\_IIC\_BOUND;

    I2C\_InitTSturcture.I2C\_Mode = I2C\_Mode\_I2C;

    I2C\_InitTSturcture.I2C\_DutyCycle = I2C\_DutyCycle\_16\_9;

    I2C\_InitTSturcture.I2C\_OwnAddress1 = USER\_IIC\_ADDR;

    I2C\_InitTSturcture.I2C\_Ack = I2C\_Ack\_Enable;

    I2C\_InitTSturcture.I2C\_AcknowledgedAddress = I2C\_AcknowledgedAddress\_7bit;

    I2C\_Init( USER\_IIC, &I2C\_InitTSturcture );

    I2C\_Cmd( USER\_IIC, ENABLE );

}

u8 IIC\_WaitEvent(I2C\_TypeDef\* I2Cx, uint32\_t I2C\_EVENT){

    u16 counter = 0xffff;

    while( !I2C\_CheckEvent( I2Cx, I2C\_EVENT ) ){

        counter--;

        if(counter == 0){

            return 1;

        }

    }

    return 0;

}

u8 IIC\_WriteLen(u8 addr, u8 reg, u8 len, u8 \*buf)

{

    u8 i = 0;

    I2C\_GenerateSTART(USER\_IIC, ENABLE);

    if(IIC\_WaitEvent(USER\_IIC, I2C\_EVENT\_MASTER\_MODE\_SELECT))

        return 1;

    I2C\_Send7bitAddress(USER\_IIC, (addr<<1), I2C\_Direction\_Transmitter); // 发送器件地址+写命令

    if(IIC\_WaitEvent(USER\_IIC, I2C\_EVENT\_MASTER\_TRANSMITTER\_MODE\_SELECTED))

        return 1; // 等待应答

    while (I2C\_GetFlagStatus(USER\_IIC, I2C\_FLAG\_TXE) == RESET)

        ;

    I2C\_SendData(USER\_IIC, reg); // 写寄存器地址

    while (i < len)

    {

        if (I2C\_GetFlagStatus(USER\_IIC, I2C\_FLAG\_TXE) != RESET)

        {

            I2C\_SendData(USER\_IIC, buf[i]); // 发送数据

            i++;

        }

    }

    while (I2C\_GetFlagStatus(USER\_IIC, I2C\_FLAG\_TXE) == RESET);

    I2C\_GenerateSTOP(USER\_IIC, ENABLE);

    return 0;

}

u8 IIC\_ReadLen(u8 addr, u8 reg, u8 len, u8 \*buf)

{

    u8 i = 0;

    I2C\_GenerateSTART(USER\_IIC, ENABLE);

    if(IIC\_WaitEvent(USER\_IIC, I2C\_EVENT\_MASTER\_MODE\_SELECT))return 1       ;

    I2C\_Send7bitAddress(USER\_IIC, (addr << 1) | 0X00, I2C\_Direction\_Transmitter); //发送器件地址+写命令

    if(IIC\_WaitEvent(USER\_IIC, I2C\_EVENT\_MASTER\_TRANSMITTER\_MODE\_SELECTED))return 1       ; //等待应答

    I2C\_SendData(USER\_IIC, reg); //写寄存器地址

    I2C\_GenerateSTART(USER\_IIC, ENABLE);

    if(IIC\_WaitEvent(USER\_IIC, I2C\_EVENT\_MASTER\_MODE\_SELECT))return 1       ;

    I2C\_Send7bitAddress(USER\_IIC, ((addr << 1) | 0x01), I2C\_Direction\_Receiver); //发送器件地址+读命令

    if(IIC\_WaitEvent(USER\_IIC, I2C\_EVENT\_MASTER\_RECEIVER\_MODE\_SELECTED))return 1       ; //等待应答

    while (i < len)

    {

        if (I2C\_GetFlagStatus(USER\_IIC, I2C\_FLAG\_RXNE) != RESET)

        {

            if (i == (len - 1))

            {

                I2C\_AcknowledgeConfig(USER\_IIC, DISABLE);

                buf[i] = I2C\_ReceiveData(USER\_IIC); //读数据,发送nACK

            }

            else

            {

                buf[i] = I2C\_ReceiveData(USER\_IIC); //读数据,发送ACK

            }

            i++;

        }

    }

    I2C\_AcknowledgeConfig(USER\_IIC, ENABLE);//重新启动自动ACK,不然程序会停在接收

    I2C\_GenerateSTOP(USER\_IIC, ENABLE); //产生一个停止条件

    return 0;

}

u8 IIC\_ReadByte(u8 addr,u8 reg)

{

    u8 res;

    I2C\_GenerateSTART( USER\_IIC, ENABLE );

    while( !I2C\_CheckEvent( USER\_IIC, I2C\_EVENT\_MASTER\_MODE\_SELECT ) );

    I2C\_Send7bitAddress(USER\_IIC,(addr << 1) | 0X00,I2C\_Direction\_Transmitter); //发送器件地址+写命令

    if(IIC\_WaitEvent( USER\_IIC, I2C\_EVENT\_MASTER\_TRANSMITTER\_MODE\_SELECTED ) )return 1;  //等待应答

    I2C\_SendData(USER\_IIC,reg);         //写寄存器地址

    I2C\_GenerateSTART( USER\_IIC, ENABLE );

    while( !I2C\_CheckEvent( USER\_IIC, I2C\_EVENT\_MASTER\_MODE\_SELECT ) );

    I2C\_Send7bitAddress(USER\_IIC,((addr << 1) | 0x01),I2C\_Direction\_Receiver);//发送器件地址+读命令

    while( !I2C\_CheckEvent( USER\_IIC, I2C\_EVENT\_MASTER\_RECEIVER\_MODE\_SELECTED ) ); //等待应答

    I2C\_AcknowledgeConfig( USER\_IIC, DISABLE );

    while( I2C\_GetFlagStatus( USER\_IIC, I2C\_FLAG\_RXNE ) ==  RESET );

    res = I2C\_ReceiveData( USER\_IIC ); //读数据,发送nACK

    I2C\_GenerateSTOP( USER\_IIC, ENABLE );//产生一个停止条件

    return res;

}

u8 IIC\_WriteByte(u8 addr,uint8\_t reg, uint8\_t data)

{

    uint8\_t res = 0;

    I2C\_GenerateSTART(USER\_IIC, ENABLE);

    while (1) {

        if (IIC\_WaitEvent(USER\_IIC, I2C\_EVENT\_MASTER\_MODE\_SELECT)) {

            res = 1;

            break;

        }

        I2C\_Send7bitAddress(USER\_IIC, ((addr << 1) | 0), I2C\_Direction\_Transmitter);

        if (IIC\_WaitEvent(USER\_IIC, I2C\_EVENT\_MASTER\_TRANSMITTER\_MODE\_SELECTED)) {

            res = 1;

            break;

        }

        I2C\_SendData(USER\_IIC, reg);

        if (IIC\_WaitEvent(USER\_IIC, I2C\_EVENT\_MASTER\_BYTE\_TRANSMITTED)) {

            res = 1;

            break;

        }

        while (I2C\_GetFlagStatus(USER\_IIC, I2C\_FLAG\_TXE) == RESET);

        I2C\_SendData(USER\_IIC, data);

        while (I2C\_GetFlagStatus(USER\_IIC, I2C\_FLAG\_TXE) == RESET);

        break;

    }

    I2C\_GenerateSTOP(USER\_IIC, ENABLE);

    return res;

}

**adxl345.h**

#ifndef \_\_ADLX345\_H     //防止重复定义（\_ADLX345\_H  开头)

#define \_\_ADLX345\_H

#include "ch32v30x.h"

#include "IIC.h"

//CS高电平启用IIC模式

//宏定义

#define   ADXL345\_ADDR    0x53    //该数据取决于地址线选择,SD0接地地址为0x53，接3.3V地址为0x1D(高7位）

#define DEVICE\_ID       0X00    //器件ID,0XE5

#define THRESH\_TAP      0X1D    //敲击阀值

#define OFSX            0X1E

#define OFSY            0X1F

#define OFSZ            0X20

#define DUR             0X21

#define Latent          0X22

#define Window          0X23

#define THRESH\_ACK      0X24

#define THRESH\_INACT    0X25

#define TIME\_INACT      0X26

#define ACT\_INACT\_CTL   0X27

#define THRESH\_FF       0X28

#define TIME\_FF         0X29

#define TAP\_AXES        0X2A

#define ACT\_TAP\_STATUS  0X2B

#define BW\_RATE         0X2C    //数据速率以及功率模式配置

#define POWER\_CTL       0x2D    //省电特性配置

#define INT\_ENABLE      0X2E

#define INT\_MAP         0X2F

#define INT\_SOURCE      0X30

#define DATA\_FORMAT     0x31    //数据格式控制

#define DATA\_X0         0X32

#define DATA\_X1         0X33

#define DATA\_Y0         0X34

#define DATA\_Y1         0X35

#define DATA\_Z0         0X36

#define DATA\_Z1         0X37

#define FIFO\_CTL        0X38

#define FIFO\_STATUS     0X39

//SD0接3.3V,0x3A写入，0x3B读取  SD0接地，0xA6写入，0xA7读取

#define ADXL\_READ    0XA7

#define ADXL\_WRITE   0XA6

void ADLX345\_Init();

void ADLX345\_GetData(uint8\_t \*data,uint8\_t num);

#endif    //防止重复定义（\_ADLX345\_H  结尾)

**adxl345.c**

#include "adxl345.h".

//加速度传感器

void ADLX345\_Init()

{

    uint8\_t data1 = 0x0B;

    uint8\_t data3 = 0x28;

    uint8\_t data4 = 0x00;

    uint8\_t data5 = 0x00;

    uint8\_t data6 = 0x00;

    uint8\_t data7 = 0x00;

    IIC\_Init();//100kHz,使用IIC2

    IIC\_WriteLen(ADXL345\_ADDR,DATA\_FORMAT,1,&data1);

    IIC\_WriteLen(ADXL345\_ADDR,POWER\_CTL,1,&data3);//链接使能，测量模式

    IIC\_WriteLen(ADXL345\_ADDR,INT\_ENABLE,1,&data4);

    IIC\_WriteLen(ADXL345\_ADDR,OFSX,1,&data5);

    IIC\_WriteLen(ADXL345\_ADDR,OFSY,1,&data6);

    IIC\_WriteLen(ADXL345\_ADDR,OFSZ,1,&data7);

}

void ADLX345\_GetData(uint8\_t \*data,uint8\_t num)

{

  IIC\_ReadLen(ADXL345\_ADDR, DATA\_X0, num, data);

}