

Cortex-M4原理与实践实验报告

题目： 三轴加速度传感器实验

**作者姓名 章金皓**

**指导教师**  **付明磊**

**专业班级 自动化1901**

**学 院 信息工程学院**

**提交日期** 2021年11月日

**一、实验目的**

1.了解 I2C 总线的特点和功能；

2.了解加速度传感器的原理；

3.学会使用 I2C 总线对 ADXL345 芯片进行操作

**二、实验设备**

1.计算机一台，操作系统为WindosXP或Windos7，装有CCSv6.0软件。

2.EK-TM4C1294XL实验开发板一块。

3.USB连接线一条。

**三、实验原理和流程**

**1.加速度传感器简介**

加速度传感器是一种能够测量加速力的设备。加速力就是当物体在加速过程中作用在物 体上的力，就好比地球引力。加速度计有两种：一种就是线加速度计，另一种是角加速度计。 加速度是表征物体在空间运动本质的一个基本物理量。因此，可以通过测量加速度来测 量物体的运动状态。例如，惯性导航系统就是通过飞行器的加速度来测量它的加速度、速度（地速）、位置、已飞过的距离已经相对于预定到达点的方向等。通常还通过测量加速度来 判断运动机械系统所承受的加速度负荷的大小，以便正确设计其机械强度和按照设计指标正 确控制其运动加速度，以免机件损坏。对于加速度，常用绝对法测量，即把惯性型测量装置 安装在运动体上进行测量。

加速度传感器的基本结构通常是质量-弹簧-阻尼二阶惯性系统。由质量块 m、弹簧 k和阻尼器C所组成的惯性型二阶系统。质量块通过弹簧和阻尼器与传感器基座相连接。传感 器基座与被测运动体相固连，因而随运动体一起相对于运动体之外惯性空间的某一参考点作 相对运动。

由于质量块不与传感器基座相固连，因而在惯性作用下将于基座之间产生相对位移。质量块感受加速度并产生与加速度成比例的惯性力，从而使弹簧产生与质量块相对位移相等的 伸缩变性，弹簧变形又产生与变形量成比例的反作用力。当惯性力与弹簧反作用力相平衡时， 质量块相对于基座的位移与加速度成正比例，故可通过该位移或惯性力来测量加速度。

加速度器有很多类型：位移式加速度传感器，应变式加速度传感器，由陀螺仪（角速度 传感器）的改进的角加速度计等。按照原理可以分为变磁阻式、变电容式、霍尔式等等。

**2.加速度传感器资源及特性**

我们的实验板中采用三轴数字加速度计 ADXL345，ADXL345 是 ADI 公司于 2008 年推 出的采用 MEMS 技术具有 SPI 和 I2C 数字输出功能的三轴加速度计，具有小巧轻薄、超低 功耗、可变量程、高分辨率等特点：

它只有 2mm×5mm×1mm 的外形尺寸，面积大小相当于小拇指指甲盖的 1/3；在典型电 压。电压 VS=2.5V 时功耗电流约为 25~130μA，比先期采用模拟输出的产品 ADXL330 功耗 典型值低了约 70~175μA；最大量程可达±16g，另可选择±2、±4、±8g 量程，可采用固定的 4mg/LSB 分辨率模式，该分辨率可测得 0.25°的倾角变化。

ADXL345 提供一些特殊的运动侦测功能，可侦测出物体是否处于运动状态，并能敏感 出某一轴加速度是否超过了用户自定义门限，可侦测物体是否正在跌落。此外，还集成了一 个 32 级 FIFO 缓存器，用来缓存数据以减轻处理器的负担。ADXL345 可在倾斜敏感应用中 测量静态重力加速度，也可在运动甚至振动环境中测量动态加速度，非常适合于移动设备应 用，可望在手机、游戏和定位设备、微小型导航设备、硬盘保护、运动健身器材、数码照相 机等产品中得到广泛应用。

ADXL345丰富的功能是通过使用寄存器来实现的。这些丰富的寄存器用以选择数据格 式、FIFO工作模式、数字通信模式、节点模式、中断使能以及修正各轴偏差等等。

表格

描述已自动生成 **3.** **ADXL345 引脚配置和功能描述**

图 1 ADXL345 引脚配置（顶视图）

表 1 ADXL345 引脚功能描述

|  |  |  |
| --- | --- | --- |
| **引脚编号** | **引脚名称** | **描述** |
| 1 | VDD I/0 | 数字接口电源电压。 |
| 2 | GND | 该引脚必须接地。 |
| 3 | RESERVED | 保留。该引脚必须连接到 VC 或保持断开。 |
| 4 | GND | 该引脚必须接地。 |
| 5 | GND | 该引脚必须接地。 |
| 6 | VC | 电源电压。 |
| 7 | CS | 片选。 |
| 8 | INT1 | 中断 1 输出。 |
| 9 | INT2 | 中断 2 输出。 |
| 10 | NC | 内部不连接。 |
| 11 | RESERVED | 保留。该引脚必须接地或保持断开。 |
| 12 | SDO/ALT ADDRESS | 串行数据输出（SPI4 线）备用 I2C 地址选择（I2C） |
| 13 | SDA/SDI/SDIO | 串行数据（I2C）串行数据输入（SPI4 线）/串行数据输入和输出（SPI3 线）。 |
| 14 | SCL/SCLK | 串行通信时钟。SCL 为 I2C 时钟 SCLK 为 SPI 时钟。 |

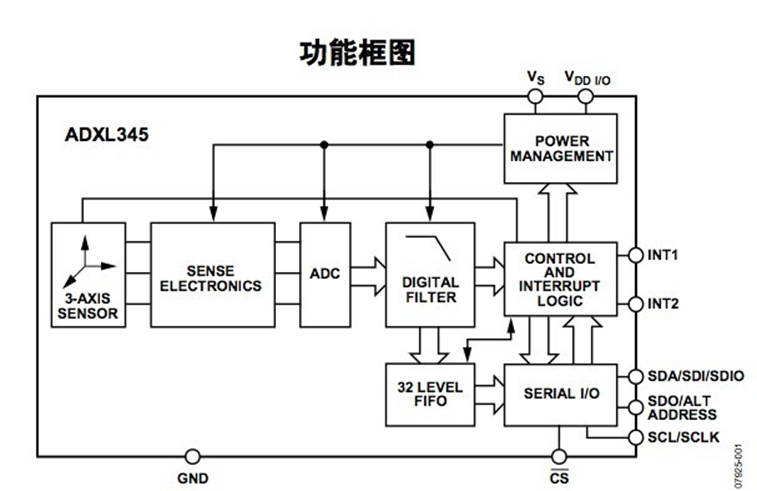
** 4、ADXL345 的内部功能图**

图 2 ADXL345 的内部功能框图

**5、ADXL345 的寄存器**

表 2寄存机映射表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **寄存器地址** | **名称** | **类型** | **复位值** | **描述** |
| 0x00 | DEVID | R | 11100101 | 器件 ID |
| 0x01-0x1C | 保留 |  |  | 保留，不操作 |
| 0x1D | THERSH\_TAP | R/W¯ | 00000000 | 敲击阈值 |
| 0x1E | OFSX | R/W¯ | 00000000 | X 轴偏移 |
| 0x1F | OFSY | R/W¯ | 00000000 | Y 轴偏移 |
| 0x20 | OFSZ | R/W¯ | 00000000 | Z 轴偏移 |
| 0x21 | DUR | R/W¯ | 00000000 | 敲击持续时间 |
| 0x22 | Latent | R/W¯ | 00000000 | 敲击延迟 |
| 0x23 | Window | R/W¯ | 00000000 | 敲击窗口 |
| 0x24 | THRESH\_ACT | R/W¯ | 00000000 | 活动阈值 |
| 0x25 | THRESH\_INACT | R/W¯ | 00000000 | 静止阈值 |
| 0x26 | TIME\_INACT | R/W¯ | 00000000 | 静止时间 |
| 0x27 | ACT\_TAP\_STATUS | R/W¯ | 00000000 | 轴使能控制活动和  静止检测 |
| 0x28 | THRESH\_FF | R/W¯ | 00000000 | 自由落体阈值 |
| 0x29 | TIME\_FF | R/W¯ | 00000000 | 自由落体时间 |
| 0x2A | TAP\_AXES | R/W¯ | 00000000 | 单机/双击轴控制 |
| 0x2B | ACT\_TAP\_STATUS | R | 00000000 | 单机/双击源 |
| 0x2C | BW\_RATE | R/W¯ | 00001010 | 数据速率及功率模  式控制 |
| 0x2D | POWER\_CTL | R/W¯ | 00000000 | 省电特性控制 |
| 0x2E | INT\_ENABLE | R/W¯ | 00000000 | 中断使能控制 |
| 0x2F | INT\_MAP | R/W¯ | 00000000 | 中断映射控制 |
| 0x30 | INT\_SOURCE | R | 00000010 | 中断源 |
| 0x31 | DATA\_FORMAT | R/W¯ | 00000000 | 数据格式控制 |
| 0x32 | DATAX0 | R | 00000000 | X 轴数据 0 |
| 0x33 | DATAX1 | R | 00000000 | X 轴数据 1 |
| 0x34 | DATAY0 | R | 00000000 | Y 轴数据 0 |
| 0x35 | DATAY1 | R | 00000000 | Y 轴数据 1 |
| 0x36 | DATAZ0 | R | 00000000 | Z 轴数据 0 |
| 0x37 | DATAZ1 | R | 00000000 | Z 轴数据 1 |
| 0x38 | FIFO\_CTL | R/W¯ | 00000000 | FIFO 控制 |
| 0x39 | FIFO\_STATUS | R/W¯ | 00000000 | FIFO 状态 |

**6、常用的寄存器**

1）BW\_RATE

BW\_RATE 用来设定功耗模式和数据率。LOW\_POWER Bit ：0，正常模式；1，低功率 模式。速率位：选择器件的带宽和数据速率。本实验设置为 0x0B，数据速率 25Hz。

表 3 寄存器 0x2C 位定义

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **D7** | **D6** | **D5** | **D4** | **D3** | **D2** | **D1** | **D0** |
| **0** | 0 | 0 | LOW\_POWER | 速率 | | | |

表 4 数据速率

|  |  |  |
| --- | --- | --- |
| **输出数据速率（Hz）** | **带宽（Hz）** | **速率代码** |
| **400** | 200 | 1100 |
| **200** | 100 | 1011 |
| **100** | 50 | 1010 |
| **50** | 25 | 1001 |
| **25** | 12.5 | 1000 |
| **12.5** | 6.25 | 0111 |

2）POWER\_CTL

POWER\_CTL 用来设定供电模式，与 BW\_RATE 配合，可设定数据率，默认值为 100Hz。 ADXL345 正常供电情况下，能根据输出数据率大小自动调节其功耗。如果要进一步降低功 耗，将 BW\_RATE 寄存器中的 LOW\_POWER 位置位，进入低功耗模式。Link Bit ：1，连 接；0，非连接。AUTO\_SLEEP Bit：0，非自动 sleep 模式；1 自动 sleep 模式。Measure Bit： 0 独立模式；1，测量模式。Sleep Bit：0，正常模式；1；sleep 模式。本实验设置为 0x08。

表 5 寄存器 0x2D 位定义

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **D7** | **D6** | **D5** | **D4** | **D3** | **D2** | **D1** | **D0** |
| **0** | 0 | Link | AUTO\_SLEEP | Measure | Sleep | Wakeup | |

表 6 Wakeup Bits 在 Sleep 模式下的读取频率设置

|  |  |  |
| --- | --- | --- |
| **设置** | | **频率（HZ）** |
| **D1** | D0 |
| **0** | 0 | 8 |
| **0** | 1 | 4 |
| **1** | 0 | 2 |
| **1** | 1 | 1 |

3）DATA\_FORMAT DATA\_FORMAT，该寄存器的设置影响着数据寄存器中的数据格式。DATA\_FORMAT该 8 位寄存器可控制 6 项设置，通过设置 SPI 位可设定 SPI 是采用 3 线还是 4 线接口模式， FULL\_RES 位与 RANGE 位，用于设定加速度量程和对应的分辨率模式，SELF\_TEST 位用于自检。INT\_INVERT 为中断模式设置：0 为相对高电平中断；1 为相对低电平中断。本实验设置位 0x0B。

表 7 寄存器 0x31 的位含义

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **D7** | **D6** | **D5** | **D4** | **D3** | **D2** | **D1** | **D0** |
| **SELF\_TEST** | SPI | INT\_INVERT | 0 | FULL\_RES | Justify | Range | |

表 8 Range 位设置

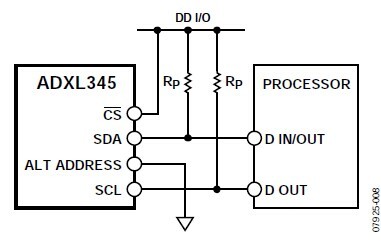
|  |  |  |
| --- | --- | --- |
| **设置** | | **频率（Hz）** |
| **D1** | D0 |
| **0** | 0 | ±2g |
| **0** | 1 | ±4g |
| **1** | 0 | ±8g |
| **1** | 1 | ±16g |

**4) OFSX、OFSY、OFSZ**

用来存放标定的 X、Y、Z 轴的偏移量，初始化传感器时使用。（详细请参考资料：ADXL345datasheet）

**7. ADXL345 芯片连接图**

ADXL345 有 I2C 和 SPI 两种操作操作模式，本实验采用 I2C 操作模式，此模式下 ADXL345 芯片的连接图如图 8-3 所示。ALT ADDRESS 引脚处于高电平，器件的 7 位 I2C 地址是 0x1D，随后为 R/W 位。这转化为 0x3A 写入，0x3B 读取。CS 引脚和 ALT ADDRESS 引脚连接至 VDDI/O（实验时将 I/O 口输出置高，也可用杜邦线将其连接至 3.3V 电压）。



图示, 示意图

描述已自动生成图 3 ADXL345 连接图（I2C 模式）

图 4 ADXL345 连接图（I2C 模式）

表 9 加速度计接口线资源

|  |  |  |  |
| --- | --- | --- | --- |
| **加速度计** | **CLK** | **PB5** |  |
| ADX\_INT | PE0 | 实验中没用到 |
| ADX\_CS | PB4 | 输出高 |
| ADX\_SDO | PE5 | 输出高 |
| ADX\_SDI | PE4 |  |

**流程图：**



**四、实验代码、注释**

**实验代码：**

#include <stdint.h>

#include <stdbool.h>

#include "inc/hw\_memmap.h"

#include "inc/hw\_types.h"

#include "driverlib/pin\_map.h"

#include "driverlib/sysctl.h"

#include "driverlib/pwm.h"

#include "driverlib/gpio.h"

#include "driverlib/uart.h"

#include "utils/uartstdio.h"

uint32\_t ui32SysClock;

#define \_NOP() \_nop()

#define SLAVE\_ADDRESS\_W 0x3A //写ADXL345L时的从机地址

#define SLAVE\_ADDRESS\_R 0x3B //读ADXL345L时的从机地址

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// UART初始化

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void

InitConsole(void)

{

// 由于UART0使用PA0,PA1两个引脚，因此需要使能GPIOA模块

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

// 因为有引脚复用，所以要对PA0和PA1两个引脚的功能进行选择

// 这里将它们选择为执行UART0模块的功能

GPIOPinConfigure(GPIO\_PA0\_U0RX);

GPIOPinConfigure(GPIO\_PA1\_U0TX);

// 对于PA0和PA1两个引脚，在将它们作为UART功能使用之前，需要对它们做一些有关UART的配置

GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

UARTStdioConfig(0, 115200, ui32SysClock); //波特率设为115200

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//延时5个指令周期

//不同的工作环境,需要调整此函数，注意时钟过快时需要修改

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Delay5us()

{

\_NOP();\_NOP();\_NOP();\_NOP();\_NOP();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#define NUM 0

//IIC 接受数据临时缓冲区

char I2C\_RECV\_DATA[] =

{

0x00,

0x00,

0x00,

0x00,

0x00,

0x00

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

拉高 SDA 信号

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void I2C\_Set\_sda\_high( void )

{

GPIOPinWrite(GPIO\_PORTE\_BASE,GPIO\_PIN\_4,GPIO\_PIN\_4); //拉高PE4

\_NOP();

\_NOP();

return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

拉低SDA 信号

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void I2C\_Set\_sda\_low ( void )

{

GPIOPinWrite(GPIO\_PORTE\_BASE,GPIO\_PIN\_4,0X00000000); //拉低PE4

\_NOP();

\_NOP();

return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

拉高SCL 信号

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void I2C\_Set\_scl\_high( void )

{

GPIOPinWrite(GPIO\_PORTB\_BASE,GPIO\_PIN\_5,GPIO\_PIN\_5); //拉高PB5

\_NOP();

\_NOP();

return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

拉低SCL 信号

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void I2C\_Set\_scl\_low ( void )

{

GPIOPinWrite(GPIO\_PORTB\_BASE,GPIO\_PIN\_5,0X00000000); //拉低PB5

\_NOP();

\_NOP();

return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IIC 信号结束信号函数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void I2C\_STOP(void)

{

int i;

GPIOPinTypeGPIOOutput(GPIO\_PORTE\_BASE, GPIO\_PIN\_4); //配置PE4为输出口

I2C\_Set\_sda\_low();

for(i = NUM;i > 0;i--);

I2C\_Set\_scl\_low();

for(i = NUM;i > 0;i--);

I2C\_Set\_scl\_high();

for(i = NUM;i > 0;i--);

I2C\_Set\_sda\_high();

for(i = NUM+1;i > 0;i--);

return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IIC 信号初始化

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void I2C\_Initial( void )

{

I2C\_Set\_scl\_low();

I2C\_STOP();

return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IIC 信号起始信号函数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void I2C\_START(void)

{

int i;

GPIOPinTypeGPIOOutput(GPIO\_PORTE\_BASE, GPIO\_PIN\_4); //配置PE4为输出口

I2C\_Set\_sda\_high();

for(i = NUM;i > 0;i--);

I2C\_Set\_scl\_high();

for(i = NUM;i > 0;i--);

I2C\_Set\_sda\_low();

for(i = NUM;i > 0;i--);

I2C\_Set\_scl\_low();

return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IIC 获取应答函数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int I2C\_GetACK(void)

{

int j;

GPIOPinTypeGPIOOutput(GPIO\_PORTE\_BASE, GPIO\_PIN\_4); //配置PE4为输出口

\_NOP();

\_NOP();

I2C\_Set\_scl\_low();

for(j = NUM;j> 0;j--);

I2C\_Set\_scl\_high();

for(j = NUM;j> 0;j--);

I2C\_Set\_sda\_low();

for(j = NUM;j > 0;j--);

I2C\_Set\_scl\_low();

return 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IIC 设置应答函数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void I2C\_SetNAk(void)

{

GPIOPinTypeGPIOOutput(GPIO\_PORTE\_BASE, GPIO\_PIN\_4); //配置PE4为输出口

I2C\_Set\_scl\_low();

I2C\_Set\_sda\_high();

I2C\_Set\_scl\_high();

I2C\_Set\_scl\_low();

return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IIC 发送字节函数

参数 1：要发送字节值

return ：无返回

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void I2C\_TxByte(unsigned char nValue)

{

GPIOPinTypeGPIOOutput(GPIO\_PORTE\_BASE, GPIO\_PIN\_4);//配置PE4为输出口

int i;

int j;

for(i = 0;i < 8;i++)

{

if(nValue & 0x80) //逐位判断，1拉高SDA，0拉低SDA

I2C\_Set\_sda\_high();

else

I2C\_Set\_sda\_low();

for(j = NUM;j > 0;j--);

I2C\_Set\_scl\_high();

nValue <<= 1; //左移1位

for(j = NUM;j > 0;j--);

I2C\_Set\_scl\_low();

}

return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IIC 接收字节函数

参数 无

return ：无返回

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

unsigned char I2C\_RxByte(void)

{

GPIOPinTypeGPIOInput(GPIO\_PORTE\_BASE, GPIO\_PIN\_4);//配置PE4为输入口

char nTemp=0 ;

int i;

I2C\_Set\_sda\_high();

\_NOP();

\_NOP();

\_NOP();

\_NOP();

for(i = 0;i < 8;i++)

{

I2C\_Set\_scl\_high(); //模拟SCL信号

Delay5us();

if(GPIOPinRead(GPIO\_PORTE\_BASE,GPIO\_PIN\_4) == 0x10) //依次判断PE4（SDA）输入

{

nTemp |= (0x01 << (7-i)); //8位SDA数据哪一位为高就置1

}

I2C\_Set\_scl\_low();

Delay5us();

}

return nTemp;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IIC 发送数组函数

参数 1 num : 发送字节数

2 device\_addr : iic目标地址

3 \*data ：发送数组地址

return ：无返回

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void i2c\_write(int num, unsigned char device\_addr, char \*data)

{

int i = 0;

int count = num;

char \*send\_data = data;

unsigned char write\_addr = device\_addr;

I2C\_Set\_scl\_high();

for(i = NUM;i > 0;i--);

I2C\_Set\_sda\_high();

for(i = NUM;i > 0;i--);

for(i = 0;i < count;i++)

{

I2C\_START(); //模拟I2C写数据的时序

I2C\_TxByte(write\_addr);

I2C\_GetACK();

I2C\_TxByte(send\_data[i]);

I2C\_GetACK();

i++;

I2C\_TxByte(send\_data[i]);

I2C\_GetACK();

I2C\_STOP();

Delay5us();

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IIC 读取数组函数

参数 1 num : 发送字节数

2 device\_addr : iic目标地址

3 \*data ：接收数组地址

return ：无返回

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void i2c\_read(int num, unsigned char device\_addr, char \*data)

{

int i = 0;

int count = num;

char \*send\_data = data;

unsigned char read\_addr = device\_addr;

I2C\_Set\_scl\_high();

for(i = NUM;i > 0;i--);

I2C\_Set\_sda\_high();

for(i = NUM;i > 0;i--);

for(i = 0; i < count;i++)

{

I2C\_START(); //模拟I2C读数据

I2C\_TxByte((read\_addr - 1));

I2C\_GetACK();

I2C\_TxByte(send\_data[i]);

I2C\_GetACK();

I2C\_START();

I2C\_TxByte(read\_addr);

I2C\_GetACK();

I2C\_RECV\_DATA[i] = I2C\_RxByte();

I2C\_SetNAk();

I2C\_STOP();

Delay5us();

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*单读一个字节数据\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

unsigned char SINGLE\_read(unsigned char device\_addr,char data)

{

int i = 0;

char value;

char send\_data = data;

unsigned char read\_addr = device\_addr;

I2C\_Set\_scl\_high();

for(i = NUM;i > 0;i--);

I2C\_Set\_sda\_high();

for(i = NUM;i > 0;i--);

I2C\_START();

I2C\_TxByte((read\_addr - 1));

I2C\_GetACK();

I2C\_TxByte(send\_data);

I2C\_GetACK();

I2C\_START();

I2C\_TxByte(read\_addr);

I2C\_GetACK();

value = I2C\_RxByte();

I2C\_SetNAk();

I2C\_STOP();

Delay5us();

return value;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//GPIO口初始化

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void I2C0GPIOBEnable(void) //配置I2C0模块的IO引脚，

{

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOB); //使能 GPIO portB

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOE); //使能 GPIO portE

GPIOPinTypeGPIOOutput(GPIO\_PORTB\_BASE, GPIO\_PIN\_5|GPIO\_PIN\_4);//使能PB4、PB5作为输出口

GPIOPinTypeGPIOOutput(GPIO\_PORTE\_BASE, GPIO\_PIN\_5);//使能PE5作为输出口

GPIOPinWrite(GPIO\_PORTB\_BASE,GPIO\_PIN\_4,GPIO\_PIN\_4);//PB4口置高，即ADXL345L的CS引脚

GPIOPinWrite(GPIO\_PORTE\_BASE,GPIO\_PIN\_5,GPIO\_PIN\_5);//PE5口置高，即ADXL345L的ALT ADDRESS引脚

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//初始化ADXL345，根据需要请参考pdf进行修改

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void Init\_ADXL345(void)//pdf 45页

{

char dataBuf1[2] = {0x31, 0x0B};//数据格式控制

i2c\_write(2,SLAVE\_ADDRESS\_W,dataBuf1);

//0000 1011 测量范围,正负16g，DATA\_FORMAT,4mg/LSB的比例因子,FULL\_RES

char dataBuf2[2] = {0x2C, 0x18};//数据速率及功率模式控制

i2c\_write(2,SLAVE\_ADDRESS\_W,dataBuf2); //速率设定为12.5 ，BW\_RATE

char dataBuf3[2] = {0x2D, 0x08};//省电特性

i2c\_write(2,SLAVE\_ADDRESS\_W,dataBuf3); //选择电源模式 ， POWER\_CTL

char dataBuf4[2] = {0x2E, 0x00};//中断使能控制

i2c\_write(2,SLAVE\_ADDRESS\_W,dataBuf4); //不使用 DATA\_READY 中断

char dataBuf5[2] = {0x1E, 0x00};//X 偏移量

i2c\_write(2,SLAVE\_ADDRESS\_W,dataBuf5); //根据测试传感器的状态写入

char dataBuf6[2] = {0x1F, 0x00};//Y 偏移量

i2c\_write(2,SLAVE\_ADDRESS\_W,dataBuf6); //根据测试传感器的状态写入

char dataBuf7[2] = {0x20, 0x06};//Z 偏移量

i2c\_write(2,SLAVE\_ADDRESS\_W,dataBuf7); //根据测试传感器的状态写入

}

void main(void)

{

int mg[3]={0x00,0x00,0x00};

char REG\_ADDR[6]={0x32,0x33,0x34,0x35,0x36,0x37};//x、y、z轴数据

ui32SysClock = SysCtlClockFreqSet((SYSCTL\_XTAL\_25MHZ |

SYSCTL\_OSC\_MAIN |

SYSCTL\_USE\_PLL |

SYSCTL\_CFG\_VCO\_480), 50000000); //设置系统时间为50MHZ

InitConsole(); //uart初始化

I2C0GPIOBEnable(); //配置I2C0模块的IO引脚

Init\_ADXL345(); //ADXL345初始化

UARTprintf("\nX: Y: Z:\n");

int mgb[3]={0x00,0x00,0x00};

while(1)

{

i2c\_read(6,SLAVE\_ADDRESS\_R,REG\_ADDR); //连续6次分别读取0x32-0x37中的数据。存于临时数据存放区I2C\_RECV\_DATA[]，补码形式存储

//\*\*\*\*\*\*\*\*\*\*\*原始数据转换为加速度数据显示\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*if(I2C\_RECV\_DATA[1]>16)//高8位与16进行比较

{//若大于16则进行取反操作，然后将高低8位进行合并

I2C\_RECV\_DATA[1]=0xFF-I2C\_RECV\_DATA[1];

I2C\_RECV\_DATA[0]=0xFF-I2C\_RECV\_DATA[0];

mg[0]=I2C\_RECV\_DATA[0]\*4+I2C\_RECV\_DATA[1]\*1024;

UARTprintf("-%d mg ",mg[0]);

}

else

{//否则就直接进行合并

mg[0]=I2C\_RECV\_DATA[0]\*4+I2C\_RECV\_DATA[1]\*1024;

UARTprintf("%d mg ",mg[0]);

}

if(I2C\_RECV\_DATA[3]>16)

{

I2C\_RECV\_DATA[3]=0xFF-I2C\_RECV\_DATA[3];

I2C\_RECV\_DATA[2]=0xFF-I2C\_RECV\_DATA[2];

mg[1]=I2C\_RECV\_DATA[2]\*4+I2C\_RECV\_DATA[3]\*1024;

UARTprintf("-%d mg ",mg[1]);

}

else

{

mg[1]=I2C\_RECV\_DATA[2]\*4+I2C\_RECV\_DATA[3]\*1024;

UARTprintf("%d mg ",mg[1]);

}

if(I2C\_RECV\_DATA[5]>16)

{

I2C\_RECV\_DATA[5]=0xFF-I2C\_RECV\_DATA[5];

I2C\_RECV\_DATA[4]=0xFF-I2C\_RECV\_DATA[4];

mg[2]=I2C\_RECV\_DATA[4]\*4+I2C\_RECV\_DATA[5]\*1024;

UARTprintf("-%d mg ",mg[2]);

UARTprintf("\n");

}

else

{

mg[2]=I2C\_RECV\_DATA[4]\*4+I2C\_RECV\_DATA[5]\*1024;

UARTprintf("%d mg ",mg[2]);

UARTprintf("\n");

}\*/

int mga[3]={0x00,0x00,0x00};

int i=1;

for(i=1;i<=10;i++){

if(I2C\_RECV\_DATA[1]>16)//高8位与16进行比较

{//若大于16则进行取反操作，然后将高低8位进行合并

I2C\_RECV\_DATA[1]=0xFF-I2C\_RECV\_DATA[1];

I2C\_RECV\_DATA[0]=0xFF-I2C\_RECV\_DATA[0];

mg[0]=I2C\_RECV\_DATA[0]\*4+I2C\_RECV\_DATA[1]\*1024-8;

UARTprintf("-%d mg ",mgb[0]);

}

else

{//否则就直接进行合并

mg[0]=I2C\_RECV\_DATA[0]\*4+I2C\_RECV\_DATA[1]\*1024-8;

UARTprintf("%d mg ",mgb[0]);

}

mga[0]+=mg[0];

if(I2C\_RECV\_DATA[3]>16)

{

I2C\_RECV\_DATA[3]=0xFF-I2C\_RECV\_DATA[3];

I2C\_RECV\_DATA[2]=0xFF-I2C\_RECV\_DATA[2];

mg[1]=I2C\_RECV\_DATA[2]\*4+I2C\_RECV\_DATA[3]\*1024-12;

UARTprintf("-%d mg ",mgb[1]);

}

else

{

mg[1]=I2C\_RECV\_DATA[2]\*4+I2C\_RECV\_DATA[3]\*1024-12;

UARTprintf("%d mg ",mgb[1]);

}

mga[1]+=mg[1];

if(I2C\_RECV\_DATA[5]>16)

{

I2C\_RECV\_DATA[5]=0xFF-I2C\_RECV\_DATA[5];

I2C\_RECV\_DATA[4]=0xFF-I2C\_RECV\_DATA[4];

mg[2]=I2C\_RECV\_DATA[4]\*4+I2C\_RECV\_DATA[5]\*1024;

UARTprintf("-%d mg ",mgb[2]);

UARTprintf("\n");

}

else

{

mg[2]=I2C\_RECV\_DATA[4]\*4+I2C\_RECV\_DATA[5]\*1024;

UARTprintf("%d mg ",mgb[2]);

UARTprintf("\n");

}

mga[2]+=mg[2];

SysCtlDelay(40\*(50000000/3000)); //延时n\*1ms

}

mgb[0]=mga[0]/10;

mgb[1]=mga[1]/10;

mgb[2]=mga[2]/10;

//SysCtlDelay(40\*(50000000/3000)); //延时n\*1ms

}

}

**实验现象：**

打开串口调试助手进行相应设置，运行程序，从左到右依次显示X、Y、Z轴的加速度。实验板平放，X、Y数据接近0mg，而Z轴数据接近1g（即重力方向加速度）。晃动开发板，三轴数据会发生变化。

**五、思考题**

**1.程序中配置了ADXL345芯片哪几个寄存器？有何作用？**

答：用到了6个寄存器：

①BW\_RATE用来设定功耗模式和数据率，本次我们使用的是低功率模式和12.5的输出数据速率

②POWER\_CTL用来设定供电模式，与BW\_RATE配合可设定数据率

③DATA\_FORMAT设置影响数据寄存器中的数据格式，例如测量范围,正负16g，13位模式

④OFSX存放X轴的偏移量

⑤OFSY存放Y轴的偏移量

⑥OFSZ存放Z轴的偏移量

**2.如果静止平放时X、Y轴数据与0mg偏差过大，请试着矫正。**

答：

（1）如果静止平放时数据的绝对值偏差过大，可对输出数据进行校正，代码如下，改变dataBuf的第二个数字，就可以修改偏置（用补码）

char dataBuf5[2] = {0x1E, 0x00};

i2c\_write(2,SLAVE\_ADDRESS\_W,dataBuf5); //X 偏移量

char dataBuf6[2] = {0x1F, 0x00};

i2c\_write(2,SLAVE\_ADDRESS\_W,dataBuf6); //Y 偏移量

char dataBuf7[2] = {0x20, 0x00};

i2c\_write(2,SLAVE\_ADDRESS\_W,dataBuf7); //Z 偏移量

（2）在进行初步校正后，发现数据的波动值较大，难以达到0mg的稳定值。为了提高矫正的效果，采用循环测量取平均值的方法进行零篇矫正。代码如下，循环次数为10次。

int mga[3]={0x00,0x00,0x00};

int i=1;

for(i=1;i<=10;i++){

if(I2C\_RECV\_DATA[1]>16)//高8位与16进行比较

{//若大于16则进行取反操作，然后将高低8位进行合并

I2C\_RECV\_DATA[1]=0xFF-I2C\_RECV\_DATA[1];

I2C\_RECV\_DATA[0]=0xFF-I2C\_RECV\_DATA[0];

mg[0]=I2C\_RECV\_DATA[0]\*4+I2C\_RECV\_DATA[1]\*1024;

UARTprintf("-%d mg ",mgb[0]);

}

else

{//否则就直接进行合并

mg[0]=I2C\_RECV\_DATA[0]\*4+I2C\_RECV\_DATA[1]\*1024;

UARTprintf("%d mg ",mgb[0]);

}

mga[0]+=mg[0];

if(I2C\_RECV\_DATA[3]>16)

{

I2C\_RECV\_DATA[3]=0xFF-I2C\_RECV\_DATA[3];

I2C\_RECV\_DATA[2]=0xFF-I2C\_RECV\_DATA[2];

mg[1]=I2C\_RECV\_DATA[2]\*4+I2C\_RECV\_DATA[3]\*1024;

UARTprintf("-%d mg ",mgb[1]);

}

else

{

mg[1]=I2C\_RECV\_DATA[2]\*4+I2C\_RECV\_DATA[3]\*1024;

UARTprintf("%d mg ",mgb[1]);

}

mga[1]+=mg[1];

if(I2C\_RECV\_DATA[5]>16)

{

I2C\_RECV\_DATA[5]=0xFF-I2C\_RECV\_DATA[5];

I2C\_RECV\_DATA[4]=0xFF-I2C\_RECV\_DATA[4];

mg[2]=I2C\_RECV\_DATA[4]\*4+I2C\_RECV\_DATA[5]\*1024;

UARTprintf("-%d mg ",mgb[2]);

UARTprintf("\n");

}

else

{

mg[2]=I2C\_RECV\_DATA[4]\*4+I2C\_RECV\_DATA[5]\*1024;

UARTprintf("%d mg ",mgb[2]);

UARTprintf("\n");

}

mga[2]+=mg[2];

SysCtlDelay(40\*(50000000/3000)); //延时n\*1ms

}

mgb[0]=mga[0]/10;

mgb[1]=mga[1]/10;

mgb[2]=mga[2]/10;

}

}

最后得到的的mg[0], mg[1], mg[2]就是对应的偏移量。

**3. 修改程序，自己写一个读一个字节数据的函数，调用此函数来分别读取XYZ数据寄存器中的数据，然后通过UART上传便于观察。**

（1）读一个字节数据的函数代码如下：

//\*\*\*\*\*\*\*\*\*\*\*\*单读一个字节数据\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

unsigned char SINGLE\_read(unsigned char device\_addr,char data)

{

int i = 0;

char value;

char send\_data = data;

unsigned char read\_addr = device\_addr;

I2C\_Set\_scl\_high();

for(i = NUM;i > 0;i--);

I2C\_Set\_sda\_high();

for(i = NUM;i > 0;i--);

I2C\_START();

I2C\_TxByte((read\_addr - 1));

I2C\_GetACK();

I2C\_TxByte(send\_data);

I2C\_GetACK();

I2C\_START();

I2C\_TxByte(read\_addr);

I2C\_GetACK();

value = I2C\_RxByte();

I2C\_SetNAk();

I2C\_STOP();

Delay5us();

return value;

}

（2）主函数中增加的内容：

I2C\_RECV\_DATA[0]=SINGLE\_read(SLAVE\_ADDRESS\_R,0x32);//单次读取0x32中的数据

I2C\_RECV\_DATA[1]=SINGLE\_read(SLAVE\_ADDRESS\_R,0x33);

I2C\_RECV\_DATA[2]=SINGLE\_read(SLAVE\_ADDRESS\_R,0x34);

I2C\_RECV\_DATA[3]=SINGLE\_read(SLAVE\_ADDRESS\_R,0x35);

I2C\_RECV\_DATA[4]=SINGLE\_read(SLAVE\_ADDRESS\_R,0x36);

I2C\_RECV\_DATA[5]=SINGLE\_read(SLAVE\_ADDRESS\_R,0x37);

//直接显示原始数据

int i;

for( i=0;i<6;i++)

{

UARTprintf("%x ",I2C\_RECV\_DATA[i]);

if(i==5)

UARTprintf("\n");

}

**六、实验体会与心得**

经过本次实验，我发现了I2C有很多优点：

(1)硬件上，I2C总线只需要一根数据线和一根时钟线两根线，而且总线接口已经集成在芯片内部，不需要特殊的接口电路。因此I2C总线简化了硬件电路PCB布线，降低了系统成本，提高了系统可靠性。

(2)软件上，I2C总线是一个多主机总线，如果两个或多个主机同时初始化数据传输，可以通过冲突检测（低电平优先）和仲裁防止数据破坏，每个连接到总线上的器件都有唯一的地址，任何器件既可以作为主机也可以作为从机，但同一时刻只允许有一个主机。数据传输和地址设定由软件设定，非常灵活，为了片选从机，主机每次发送数据串的开头要先发送个从机地址。除此之外，总线上的器件增加和删除不影响其他器件正常工作。

(3)陀螺仪在日常生活中十分常用，比如四旋翼的姿态传感，导弹的惯性导航等。本次实验用到的与我之前接触的不太一样，数据传回的是三轴的线加速度，之前接触的是传回四元数，还需要另外解算出三轴角加速度，或者就是陀螺仪模块上直接硬件解算好传回三轴数据。一般在陀螺仪的使用前需要零飘初始化，简单的算法就是将板子静止不动，取多次数据的平均值作为之后数据的偏置，这样可以抵消掉重力的影响，方便之后的使用。