

实验七

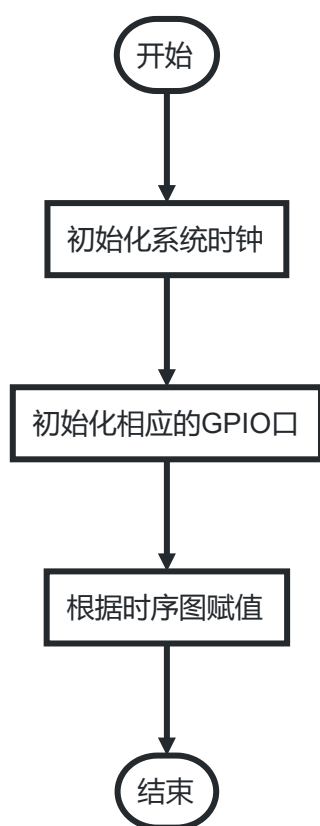
实验目的

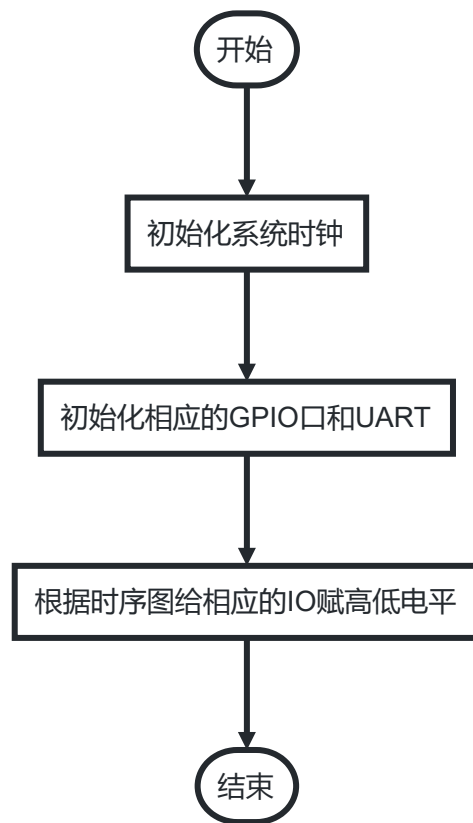
1. 熟悉并口 AD/DA 芯片的结构及工作方式
2. 熟悉并行口的扩展编程

实验内容

学习并使用并行ADC和DAC,能够根据时序图操作GPIO口

实验流程图





实验源代码

```
1 //DAC初始化
2 void DAC_Init()
3 {
4     //CS ~ PE1
5     //WR ~ PE2
6     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
7     SysCtlGPIOAHBEnable(SYSCTL_PERIPH_GPIOE);
8     GPIOPinTypeGPIOOutput(GPIO_PORTE_BASE, GPIO_PIN_1|GPIO_PIN_2);
9
10    /*
11        DATA0 ~ PK3
12        DATA1 ~ PK2
13        DATA2 ~ PK1
14        DATA3 ~ PK0
15        DATA4 ~ PC7
16        DATA5 ~ PC6
17        DATA6 ~ PC5
18        DATA7 ~ PC4
19        DATA8 ~ PA6
20        DATA9 ~ PA7
21        DATA10 ~ PG1
22        DATA11 ~ PG0
23    */
24
25    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOK);
26    SysCtlGPIOAHBEnable(SYSCTL_PERIPH_GPIOK);
27    GPIOPinTypeGPIOOutput(GPIO_PORTK_BASE, GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1|GPIO_PIN_0);
28
29    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
30    SysCtlGPIOAHBEnable(SYSCTL_PERIPH_GPIOC);
31    GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_7|GPIO_PIN_6|GPIO_PIN_5|GPIO_PIN_4);
32
33    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
34    SysCtlGPIOAHBEnable(SYSCTL_PERIPH_GPIOA);
35    GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_6|GPIO_PIN_7);
```

```

36
37     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);
38     SysCtlGPIOAHBEnable(SYSCTL_PERIPH_GPIOG);
39     GPIOPinTypeGPIOOutput(GPIO_PORTG_BASE, GPIO_PIN_1|GPIO_PIN_0);
40
41 }
42
43 //DAC输出电压
44 void DAC_Set_Data(uint16_t out)
45 {
46     bool out_buffer[12];
47     uint16_t temp;
48     for(int i=0;i<12;i++)
49     {
50         out_buffer[i] = (out<<(4+i))>>15;
51     }
52
53     GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_0, out_buffer[0]?GPIO_PIN_0:0); //DATA11
54     GPIOPinWrite(GPIO_PORTG_BASE, GPIO_PIN_1, out_buffer[1]?GPIO_PIN_1:0); //DATA10
55     GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_7, out_buffer[2]?GPIO_PIN_7:0); //DATA9
56     GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_6, out_buffer[3]?GPIO_PIN_6:0); //DATA8
57     GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_4, out_buffer[4]?GPIO_PIN_4:0); //DATA7
58     GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_5, out_buffer[5]?GPIO_PIN_5:0); //DATA6
59     GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_6, out_buffer[6]?GPIO_PIN_6:0); //DATA5
60     GPIOPinWrite(GPIO_PORTC_BASE, GPIO_PIN_7, out_buffer[7]?GPIO_PIN_7:0); //DATA4
61     GPIOPinWrite(GPIO_PORTK_BASE, GPIO_PIN_0, out_buffer[8]?GPIO_PIN_0:0); //DATA3
62     GPIOPinWrite(GPIO_PORTK_BASE, GPIO_PIN_1, out_buffer[9]?GPIO_PIN_1:0); //DATA2
63     GPIOPinWrite(GPIO_PORTK_BASE, GPIO_PIN_2, out_buffer[10]?GPIO_PIN_2:0); //DATA1
64     GPIOPinWrite(GPIO_PORTK_BASE, GPIO_PIN_3, out_buffer[11]?GPIO_PIN_3:0); //DATA0
65 }
66
67 //out = 3.3*out/4096 (0<out<4096)
68 void DAC_Output(uint16_t out)
69 {
70     //PULL UP CS
71     GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_1, GPIO_PIN_1);
72     SysCtlDelay(1);
73
74     //PULL DOWN WR
75     GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_2, 0);
76     SysCtlDelay(1);
77
78     //PULL DOWN CS
79     GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_1, 0);
80     SysCtlDelay(1);
81
82     //SET DATA0~11
83     DAC_Set_Data(out);
84     SysCtlDelay(1);
85
86     //PULL UP CS
87     GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_1, GPIO_PIN_1);
88     SysCtlDelay(1);
89
90     //PULL up WR
91     GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_2, GPIO_PIN_2);
92     SysCtlDelay(1);
93 }
94
95 //DAC测试函数
96 void DAC_Test()
97 {
98     uint16_t i =0;
99     for(;;)
100     {
101         DAC_Output(i*1000);
102         i++;
103         if(i==5)i=0;
104         SysCtlDelay(SysCtlClockGet()/3000);
105     }
106 }
107
108
109 //ADC初始化

```

```

110 void EXADC_Init()
111 {
112     //RD~PE0    CS~PE1
113     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
114     SysCtlGPIOAHBEnable(SYSCTL_PERIPH_GPIOE);
115     GPIOPinTypeGPIOOutput(GPIO_PORTE_BASE, GPIO_PIN_1|GPIO_PIN_0);
116
117     //AD_BUSY~PD6
118     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
119     SysCtlGPIOAHBEnable(SYSCTL_PERIPH_GPIOD);
120     GPIOPinTypeGPIOInput(GPIO_PORTD_BASE, GPIO_PIN_6);
121
122     //AD_BYTE~PD5    AD_CONVST~PD4
123     GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, GPIO_PIN_5|GPIO_PIN_4);
124
125     /*
126         DATA0 ~ PK3
127         DATA1 ~ PK2
128         DATA2 ~ PK1
129         DATA3 ~ PK0
130         DATA4 ~ PC7
131         DATA5 ~ PC6
132         DATA6 ~ PC5
133         DATA7 ~ PC4
134         DATA8 ~ PA6
135         DATA9 ~ PA7
136         DATA10 ~ PG1
137         DATA11 ~ PG0
138     */
139
140     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOK);
141     SysCtlGPIOAHBEnable(SYSCTL_PERIPH_GPIOK);
142     GPIOPinTypeGPIOInput(GPIO_PORTK_BASE, GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1|GPIO_PIN_0);
143
144     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
145     SysCtlGPIOAHBEnable(SYSCTL_PERIPH_GPIOC);
146     GPIOPinTypeGPIOInput(GPIO_PORTC_BASE, GPIO_PIN_7|GPIO_PIN_6|GPIO_PIN_5|GPIO_PIN_4);
147
148     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
149     SysCtlGPIOAHBEnable(SYSCTL_PERIPH_GPIOA);
150     GPIOPinTypeGPIOInput(GPIO_PORTA_BASE, GPIO_PIN_6|GPIO_PIN_7);
151
152     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);
153     SysCtlGPIOAHBEnable(SYSCTL_PERIPH_GPIOG);
154     GPIOPinTypeGPIOInput(GPIO_PORTG_BASE, GPIO_PIN_1|GPIO_PIN_0);
155 }
156
157 //ADC读取
158 uint16_t EXADC_Read()
159 {
160
161     uint16_t exad_value=0;
162     uint16_t input_buffer[12]={};
163
164     //PULL CS & RD HIGH
165     GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_0, GPIO_PIN_0);
166     GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_1, GPIO_PIN_1);
167
168     //PULL AD_CONVST LOW  AD_BYTE LOW
169     GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4, 0);
170     GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_5, 0);
171
172     //PULL CS LOW
173     GPIOPinWrite(GPIO_PORTE_BASE, GPIO_PIN_1, 0);
174
175     //PULL AD_CONVST HIGH
176     GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4, GPIO_PIN_4);
177     SysCtlDelay(2);
178
179     //PULL AD_CONVST LOW
180     GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_4, 0);
181     SysCtlDelay(3);
182
183     //PULL RD LOW

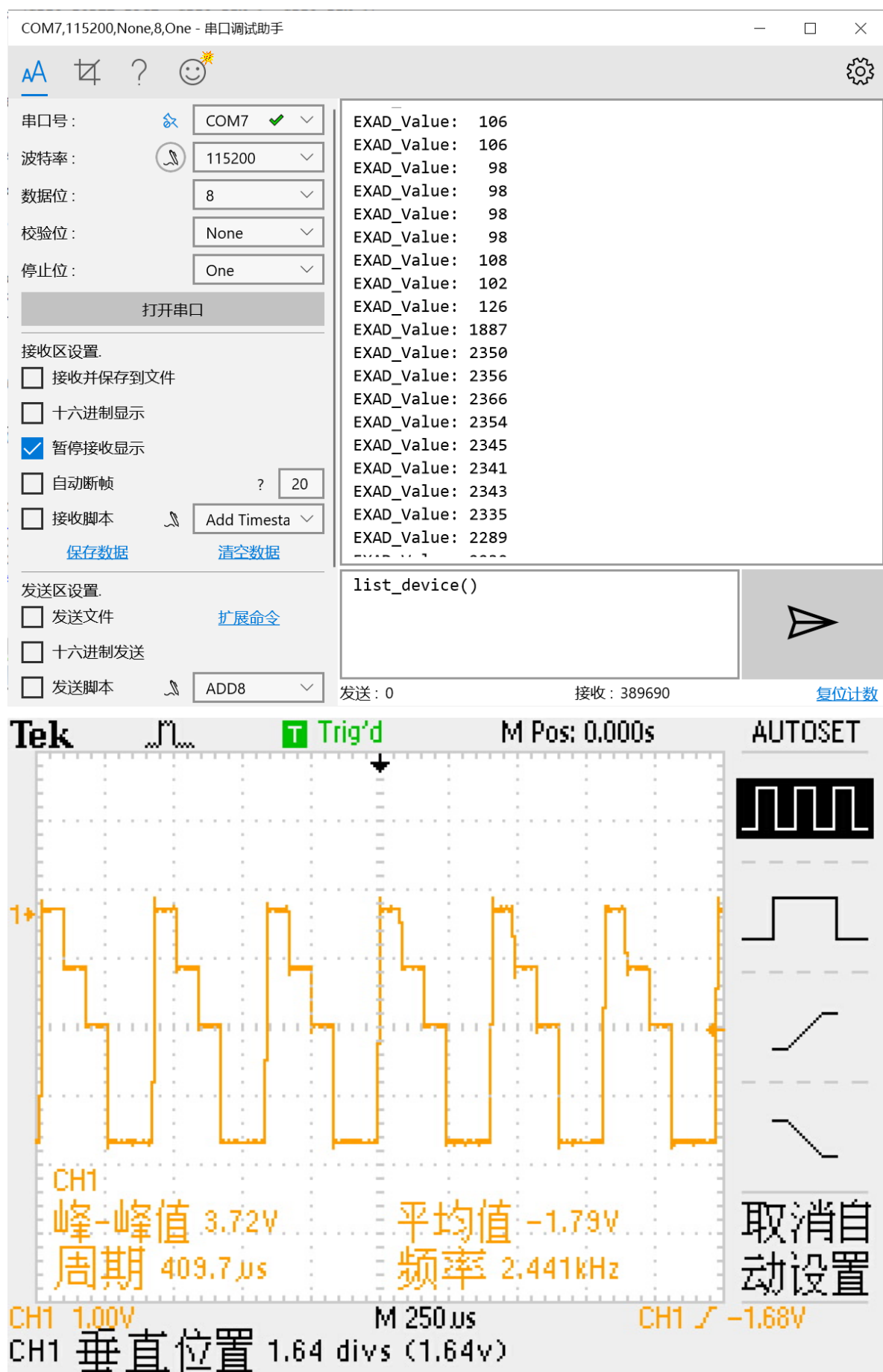
```

```

184     GPIOWrite(GPIO_PORTC_BASE, GPIO_PIN_0, 0);
185     SysCtlDelay(1);
186
187     //DATA READ
188     input_buffer[0] = GPIOWrite(GPIO_PORTG_BASE, GPIO_PIN_0); //DATA11
189     input_buffer[1] = GPIOWrite(GPIO_PORTG_BASE, GPIO_PIN_1)>>1; //DATA10
190     input_buffer[2] = GPIOWrite(GPIO_PORTA_BASE, GPIO_PIN_7)>>7; //DATA9
191     input_buffer[3] = GPIOWrite(GPIO_PORTA_BASE, GPIO_PIN_6)>>6; //DATA8
192     input_buffer[4] = GPIOWrite(GPIO_PORTC_BASE, GPIO_PIN_4)>>4; //DATA7
193     input_buffer[5] = GPIOWrite(GPIO_PORTC_BASE, GPIO_PIN_5)>>5; //DATA6
194     input_buffer[6] = GPIOWrite(GPIO_PORTC_BASE, GPIO_PIN_6)>>6; //DATA5
195     input_buffer[7] = GPIOWrite(GPIO_PORTC_BASE, GPIO_PIN_7)>>7; //DATA4
196     input_buffer[8] = GPIOWrite(GPIO_PORTK_BASE, GPIO_PIN_0); //DATA3
197     input_buffer[9] = GPIOWrite(GPIO_PORTK_BASE, GPIO_PIN_1)>>1; //DATA2
198     input_buffer[10] = GPIOWrite(GPIO_PORTK_BASE, GPIO_PIN_2)>>2; //DATA1
199     input_buffer[11] = GPIOWrite(GPIO_PORTK_BASE, GPIO_PIN_3)>>3; //DATA0
200
201     //PULL CS & RD HIGH
202     GPIOWrite(GPIO_PORTC_BASE, GPIO_PIN_0, GPIO_PIN_0);
203     GPIOWrite(GPIO_PORTC_BASE, GPIO_PIN_1, GPIO_PIN_1);
204
205     exad_value =
206         input_buffer[0]*2048+ input_buffer[1]*1024+ input_buffer[2]*512 +\
207         input_buffer[3]*256 + input_buffer[4]*128 + input_buffer[5]*64 +\
208         input_buffer[6]*32 + input_buffer[7]*16 + input_buffer[8]*8 +\
209         input_buffer[9]*4 + input_buffer[10]*2 + input_buffer[11];
210
211     return exad_value;
212 }
213
214 //ADC测试函数
215 void EXADC_Test()
216 {
217     uint16_t exad_value;
218     float v;
219     char s[20];
220     while(1)
221     {
222         exad_value = EXADC_Read();
223         UARTprintf("%4d\n", exad_value);
224         SysCtlDelay(SysCtlClockGet()/300);
225     }
226 }

```

实验现象



思考题

已使用库函数实现

实验八

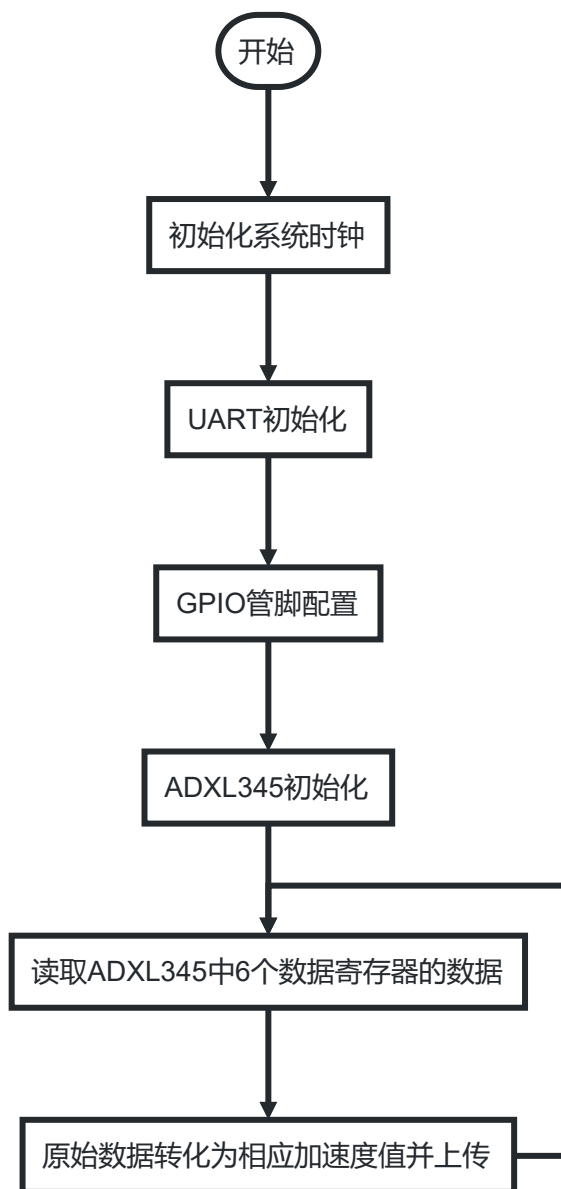
实验目的

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

实验内容

读取三轴加速计的三个数据并通过串口显示

实验流程图



实验源代码

```
1 //加速计初始化
2 void ACCELERATOR_Init(void)
```

```

3 {
4     //管脚配置
5     I2C0GPIOBEnable();
6
7     char buffer;
8
9     //数据格式
10    buffer = 0x08;
11    I2CSend(ACCELERATOR_W,ACCELERATOR_DATA_FORMAT,&buffer,1);
12    //测量范围
13    buffer = 0x18;
14    I2CSend(ACCELERATOR_W,ACCELERATOR_BW_RATE,&buffer,1);
15    //电源控制
16    buffer = 0x08;
17    I2CSend(ACCELERATOR_W,ACCELERATOR_POWER_CTL,&buffer,1);
18    //中断控制
19    buffer = 0x00;
20    I2CSend(ACCELERATOR_W,ACCELERATOR_INT_ENABLE,&buffer,1);
21
22    //X轴偏置
23    buffer = 0x00;
24    I2CSend(ACCELERATOR_W,ACCELERATOR_OFSX,&buffer,1);
25    //Y轴偏置
26    buffer = 0x00;
27    I2CSend(ACCELERATOR_W,ACCELERATOR_OFSY,&buffer,1);
28    //Z轴偏置
29    buffer = 0x00;
30    I2CSend(ACCELERATOR_W,ACCELERATOR_OFSZ,&buffer,1);
31
32
33    uint8_t devid = I2CRead(ACCELERATOR_R,ACCELERATOR_DEVID);
34    if(devid == 0xe5)
35    {
36        UARTprintf("accelerator init success!\n");
37    }
38 }
39
40 //加速计读取指定轴数据
41 int ACCELERATOR_Read(char channel)
42 {
43     int res;
44     uint8_t register_value[2]={};
45     switch(channel)
46     {
47     case('X'):
48     {
49         register_value[0] = I2CRead(ACCELERATOR_R,ACCELERATOR_X_L);
50         register_value[1] = I2CRead(ACCELERATOR_R,ACCELERATOR_X_H);
51         if(register_value[1]>16)
52         {
53             register_value[1]=0xFF-register_value[1];
54             register_value[0]=0xFF-register_value[0];
55             res = register_value[0]*4 + register_value[1]*1024;
56             res = res*-1;
57             break;
58         }
59         res = register_value[0]*4 + register_value[1]*1024;
60         break;
61     }
62
63     case('Y'):
64     {
65         register_value[0] = I2CRead(ACCELERATOR_R,ACCELERATOR_Y_L);
66         register_value[1] = I2CRead(ACCELERATOR_R,ACCELERATOR_Y_H);
67         if(register_value[1]>16)
68         {
69             register_value[1]=0xFF-register_value[1];
70             register_value[0]=0xFF-register_value[0];
71             res = register_value[0]*4 + register_value[1]*1024;
72             res = res*-1;
73             break;
74         }
75         res = register_value[0]*4 + register_value[1]*1024;
76     }

```








```



77         break;
78     }
79     case('Z'):
80     {
81         register_value[0] = I2CRead(ACCELERATOR_R,ACCELERATOR_Z_L);
82         register_value[1] = I2CRead(ACCELERATOR_R,ACCELERATOR_Z_H);
83         if(register_value[1]>16)
84         {
85             register_value[1]=0xFF-register_value[1];
86             register_value[0]=0xFF-register_value[0];
87             res = register_value[0]*4 + register_value[1]*1024;
88             res = res*-1;
89             break;
90         }
91         res = register_value[0]*4 + register_value[1]*1024;
92         break;
93     }
94 }
95 return res;
96 }
97
98 //加速剂测试函数
99 void ACCELERATOR_Test(void)
100 {
101     int Gx,Gy,Gz;
102     while(1)
103     {
104         Gx = ACCELERATOR_Read('X');
105         Gy = ACCELERATOR_Read('Y');
106         Gz = ACCELERATOR_Read('Z');
107         UARTprintf("X: %4dmg Y: %4dmg Z: %4dmg\n", Gx,Gy,Gz);
108         SysCtlDelay(1000*SysCtlClockGet()/3000);
109     }
110 }
111


```

实验现象

COM7,115200,None,8,One - 串口调试助手

串口号:  COM7 

波特率:  115200

数据位: 8

校验位: None

停止位: One

关闭串口


接收区设置.

☐ 接收并保存到文件

☐ 十六进制显示

☒ 暂停接收显示

☐ 自动断帧 ? 20

☐ 接收脚本  Add Timesta


保存数据

清空数据


发送区设置.

☐ 发送文件 扩展命令

☐ 十六进制发送

☐ 发送脚本  ADD8

list_device()



发送: 0 接收: 393451 [复位计数](#)

X: 52mg Y: -728mg Z: 596mg
 X: 0mg Y: -860mg Z: 412mg
 X: 48mg Y: -940mg Z: 244mg
 X: 84mg Y: -912mg Z: 112mg
 X: 104mg Y: -1000mg Z: -76mg
 X: 68mg Y: -972mg Z: 48mg
 X: -140mg Y: -1044mg Z: 68mg
 X: -244mg Y: -888mg Z: 408mg
 X: -640mg Y: -388mg Z: 412mg
 X: -824mg Y: -148mg Z: 388mg
 X: -844mg Y: -28mg Z: 368mg
 X: -784mg Y: 60mg Z: 412mg
 X: -588mg Y: 0mg Z: 688mg
 X: -248mg Y: -128mg Z: 868mg
 X: 184mg Y: -72mg Z: 788mg
 X: 552mg Y: -140mg Z: 712mg
 X: 776mg Y: -280mg Z: 504mg
 X: 908mg Y: -224mg Z: 272mg

思考题

1.

寄存器名称	寄存器数值	实现功能
BW_RATE	0x0B	正常模式,25HZ
POWER_CTL	0x08	非连接,非自动sleep,测量模式,正常模式,8HZ
DATA_FORMAT	0x0B	±16g
OFSX	0x00	偏置为0
OFSY	0x00	偏置为0
OFZ	0x00	偏置为0

2. 只需要在初始化时调整各个轴的OFFSET即可.

3. 已经实现单独读取每个轴数据