目录

1. 实验背景简介

加速度计在日常生活中使用广泛，智能手环、手机等电子产品的计步便是通过加速度计辅助实现的。除了用于计步，笔记本厂商的硬盘保护措施也是基于加速度计开发的。为防止因为猛烈碰撞、意外跌落损坏数据，笔记本厂商在设计笔记本的时候会利用加速度传感器判断硬盘是否剧烈震动、跌落，及时调整硬盘读写磁头到安全区，达到保护目的。本次实验便是基于这个背景，通过对 MMA7260 三轴加速度传感器进行 ADC 采样获取实验板加速度信号，进而判断实验板当前受力状态。

本次实验基于 K66 开发板为核心板进行二次扩展开发的 Cyber-Dorm 实验板，进行编程实践，实现对实验板的受力状态检测，Cyber-Dorm 实验板照片如图 1-1 所示。

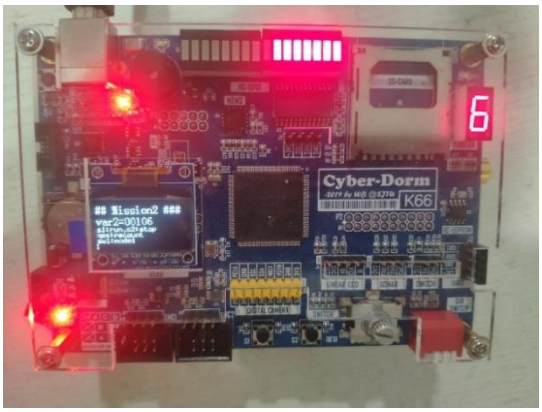


图 1-1 实验 Cyber-Dorm 开发板

2. 实验内容

本次实验需要完成的任务是使用ADC对MMA7260三轴加速度传感器的信号进行采集并判断实验板当前的受力状态。

实验具体内容如下：使用ADC采集MMA7260三轴加速度传感器信号，分析在不同运动状态下的信号的特征并通过设计算法对不同运动特征进行识别，在“跌落”状态时让蜂鸣器提示处于“跌落”状态。

实验任务
配置接口，采集 MMA7260 三轴加速度传感器信号并显示
分析三轴加速度信号在不同运动状态下的特征
设计数据处理算法，识别“跌落”状态并通过蜂鸣器提示

表格 2-1 实验任务表

3. 实验原理

3.1 MMA7260 三轴加速度传感器的使用

MMA7260 三轴加速度传感器能够感知 XYZ 三个垂直方向上的加速度，本次实验采用此传感器来检测实验板的受力和运动状态。

MMA7260 三轴加速度传感器的主要特点包括：具有三轴感应功能；可以选择检测的加速度范围和灵敏度；低功耗；有睡眠模式；低压运行；快速启动；低噪声；小体积。

MMA7260 的功能框图如图 3-1 所示。XYZ 三个相互垂直方向上的加速度由 G-Cell 传感器单元感知，经过容压变换器、增益放大、滤波器和温度补偿后以电压信号输出。

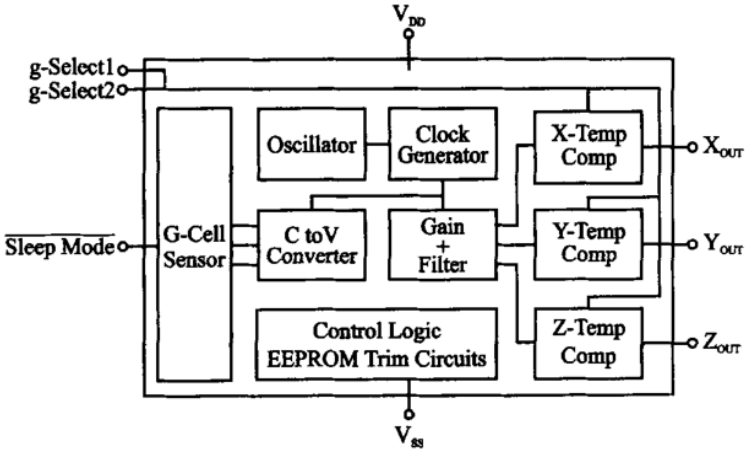


图 3-1 MMA7260 的功能框图

3.2 ADC 采样

通过对加速度传感器进行 ADC 采样，才能得到传感器返回的数值作为信号，用于分析实验板的受力和运动状态。这需要把加速度传感器和 MK66FN2M0VLA18 单片机上的 ADC 模块连接。MK66FN2M0VLA18 单片机上有 2 个 16 位精度的 SAR ADC 模块。

ADC 模块的输入方式如图 3-2 所示。

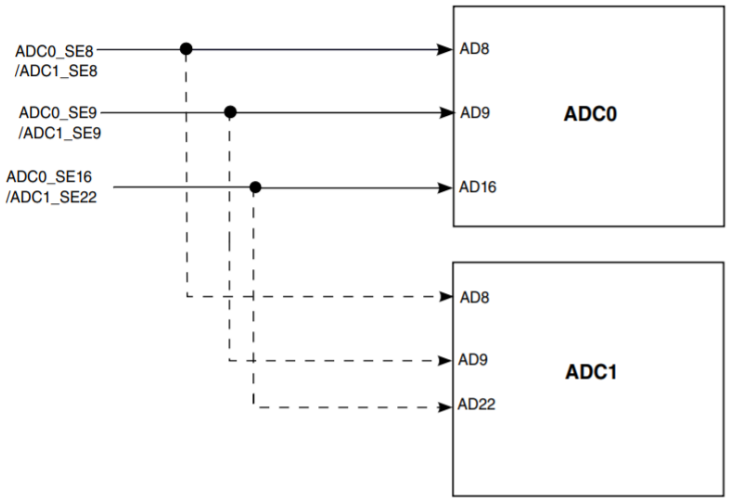


图 3-2 ADC 模块的输入通道图

ADC 进行程序化的连续采样。采样控制和数据捕获由采样序列发生器进行处理。

使用 ADC 模块时要先初始化。首先使能 ADC 模块，然后设置采样频率。采样的数据存放在 FIFO 中。当 XYZ 三轴的数据采集完成后发中断，将所采集的数据传输到存储器中，等待进一步的处理。

4. 实验方案

4.1 整体框架

本次实验的代码整体框架如下图 4-1 所示：

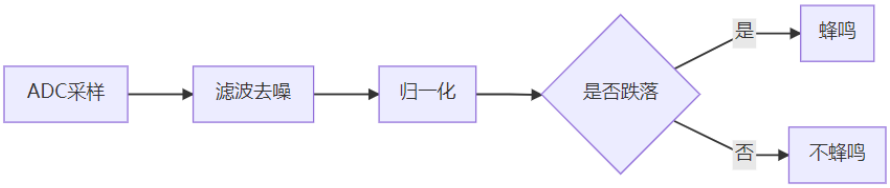


图 4-1 实验基本框图

我们小组利用基于 Ref2 示例代码文件进行开发拓展，完整实现了对 MMA7260 三轴加速度传感器的信号采样功能并对不同运动状态进行判别。如下文所述，我们将实验分为了 PIT 中断采样、滤波去噪、归一化、运动状态判定四部分。

4.2 PIT 中断采样

我们使用 PIT 定时器来完成按键扫描与程序计时器。

PIT 全称为 Programmable Interval Timer，是一个可编程的周期中断定时器。PIT 的时钟源只有一个，即总线时钟。在 PIT 定时器是一个减法定时器，内部存在许多个寄存器，有两个寄存器分别存储计数值与当前定时值，并在工作时会对存放的值以总线频率进行自减，而当计数值减为 0 时，PIT 就会被触发一次，并开启下一周期。对于 K66 核心板，可以开启的 PIT 中断共有四个通道（kPIT_Chnl_0-kPIT_Chnl_3），在实验中，我们采用 kPIT_Chnl_2 通道做为计时器中断对 ADC 进行定时采样，每 10ms 触发一次并进行采样。

PIT 定时器的中断服务部分程序代码及 ADC 采样的部分程序代码如下所示，完整代码可见文件 pit_timer.c 与 pit_timer.h 及/CDK66/MEM.h 和/CDK66/MEM.c 。

```
1.  void ADC_PIT_HANDLER(void)
2.  {
3.      /* Clear interrupt flag.*/
4.      PIT_ClearStatusFlags(PIT, kPIT_Chnl_2, kPIT_TimerFlag);
5.      MEMS_get(&MEMS);
```

```
#if defined __CORTEX_M && (__CORTEX_M == 4U)
    __DSB();
#endif
}
```

```
//-------------------------------------------------------------------------
//  @brief          Get message from adc
//  @return         adc_value of MEMS
//  @data           2020/10/16
//  Sample usage:
//-------------------------------------------------------------------------
uint16_t MEM_convert(ADC_Type *adcn, uint16_t ch)
{
    adc16_channel_config_t adc16ChannelConfigStruct;
    adc16ChannelConfigStruct.channelNumber = ch;
    adc16ChannelConfigStruct.enableInterruptOnConversionCompleted = false;

    ADC16_SetChannelConfig(adcn, 0, &adc16ChannelConfigStruct);
    while (0U == (kADC16_ChannelConversionDoneFlag &
                  ADC16_GetChannelStatusFlags(adcn, 0U))) {}
    uint16_t adc_value=(ADC16_GetChannelConversionValue(adcn, 0) & 0xFFF);
    return  adc_value;
}
```

4.3 滤波去噪

采用 ADC 采样得到的结果噪声较大，数据不能够直接使用，需要使用滤波进行去噪处理。我们小组在滤波算法上，先后尝试了中位值平均滤波、中值滤波、低通滤波等滤波方法，最终采取中值滤波与中位值平均滤波的滤波算法进行降噪处理。

滤波前后的效果图如下所示，可以看到，滤波极大地消去了噪声的干扰，为之后的数据分析处理打下了基础。

图 4-2 原始采样数据

图 4-3 滤波后数据

4.3.1 中值滤波

中值滤波采用在一次中断时间连续采样N 次（N 取奇数），把N 次采样值按大小排列，取中间值为本次有效值的基本方法。

中值滤波具有能有效克服因偶然因素引起的波动干扰、对温度、液位的变化缓慢的被测参数有良好的滤波效果等优点。

在实验中我们选取N=3作为滤波参数，并采用快排的方法来降低运算量。中值滤波的部分代码如下，完整代码可见库文件/CDK66/MEM.h与/CDK66/MEM.c。

```
1.  //-------------------------------------------------------------------------
2.  //  @brief          mid filter
3.  //  @return         Normalize the sampled value
4.  //  @data           2020/10/16
5.  //  Sample usage:
6.  //-------------------------------------------------------------------------
7.  uint16_t MEM_mid(ADC_Type *adcn, uint16_t ch)
8.  {
9.      uint16_t tmp[3] = {0, 0, 0};
10.     for (int i = 0; i < 3; i++)
11.     {
12.         tmp[i] = MEM_convert(adcn,ch);
13.     }
14.     quicksort(MEM1_TEMP, 3, 0, 2);
15.
16.     return tmp[1];
17. }
18.
19. //-------------------------------------------------------------------------
20. //  @brief          Quick sort
21. //  @data           2020/10/16
22. //  Sample usage:
23. //-------------------------------------------------------------------------
24. void quicksort(volatile uint16_t array[], int maxlen, int begin, int end)
25. {
26.     int i, j;
27.
28.     if (begin < end)
29.     {
30.         i = begin + 1; // 将 array[begin]作为基准数，因此从 array[begin+1]开始与基准数比较!
31.         j = end;       // array[end]是数组的最后一位
```

```
32.
33.         while (i < j)
34.         {
35.             if (array[i] > array[begin]) // 如果比较的数组元素大于基准数，则交换位置。
36.             {
37.                 swap(&array[i], &array[j]); // 交换两个数
38.                 --j;
39.             }
40.             else
41.             {
42.                 ++i; // 将数组向后移一位，继续与基准数比较。
43.             }
44.         }
45.         if (array[i] >= array[begin]) // 这里必须要取等“>=”，否则数组元素由相同的值时，会出现错误！
46.         {
47.             --i;
48.         }
49.
50.         swap(&array[begin], &array[i]); // 交换 array[i]与 array[begin]
51.
52.         quicksort(array, maxlen, begin, i);
53.         quicksort(array, maxlen, j, end);
54.     }
55. }
56. //--------------------------------------------------------------------------
57. //  @brief         swap
58. //  @data          2020/10/16
59. //  Sample usage:
60. //--------------------------------------------------------------------------
61. void swap(volatile uint16_t *a, volatile uint16_t *b)
62. {
63.     int temp;
64.
65.     temp = *a;
66.     *a = *b;
67.     *b = temp;
68.
69.     return;
70. }
```

4.3.2 中位值平均滤波法

中位值平均滤波法相当于中位值滤波法与算术平均滤波法的融合，基本思想为：连续采样N 个数据，去掉一个最大值和一个最小值，然后计算N-2 个数据的算术平均值。

中位值平均滤波法融合了两种滤波法的优点，对于偶然出现的脉冲性干扰，可消除由于脉冲干扰所引起的采样值偏差。

在实验中我们选取N=4作为滤波参数，并采用快排的方法来降低运算量。中位值平均滤波法的部分代码如下，完整代码可见库文件/CDK66/MEM.h与/CDK66/MEM.c。

```
1.  //-------------------------------------------------------------------------
2.  //  @brief         Get adc value after fiter and normalizing
3.  //  @return        Fianl adc_value of MEMS
4.  //  @data          2020/10/16
5.  //  Sample usage:
6.  //-------------------------------------------------------------------------
7.  void MEMS_get(MMA7260_t *AnalogInput)
8.  {
9.       //Mean filtering
10.      for (int i = 0; i <3; i++)
11.      {
12.           MEM1_TEMP[i] = MEM_mid(MEMSx_BASE,MEMSx_CHANNEL);
13.           MEM2_TEMP[i] = MEM_mid(MEMSy_BASE,MEMSy_CHANNEL);
14.           MEM3_TEMP[i] = MEM_mid(MEMSz_BASE,MEMSz_CHANNEL);
15.      }
16.      quicksort(MEM1_TEMP, 4, 0, 3);
17.      quicksort(MEM2_TEMP, 4, 0, 3);
18.      quicksort(MEM3_TEMP, 4, 0, 3);
19.      //the middle two
20.      AnalogInput->x =(MEM1_TEMP[1] + MEM1_TEMP[2])/2;
21.      AnalogInput->y =(MEM2_TEMP[1] + MEM2_TEMP[2])/2;
22.      AnalogInput->z =(MEM3_TEMP[1] + MEM3_TEMP[2])/2;
23.
24.      MEMS_Normalized(AnalogInput);
25.
26. }
```

4.4 归一化算法

在经过滤波去噪后，我们还对数据进行了归一化处理。在实验板处于不同静止状态时MMA7260 三轴加速度传感器的输出具有不同的特性，为了方便后续的数据处理以及更好的从输出数据判断实验板所处运动状态，我们在对 MMA7260 三轴加速度传感器的输出进行

标定后通过标定数据对数据进行了归一化处理。标定原理图见图 4-1 ，标定结果见表格 4-1 所示，整定结果见表格 4-2。

由于 MEMS 模块安装倾角其它各类原因，我们分别对三个方向的参数进行归一整定归一化基本公式为：

$$F = \frac{x_t - x_{min}}{x_{\max} - x_{m_i n}} \times 256 - 128$$

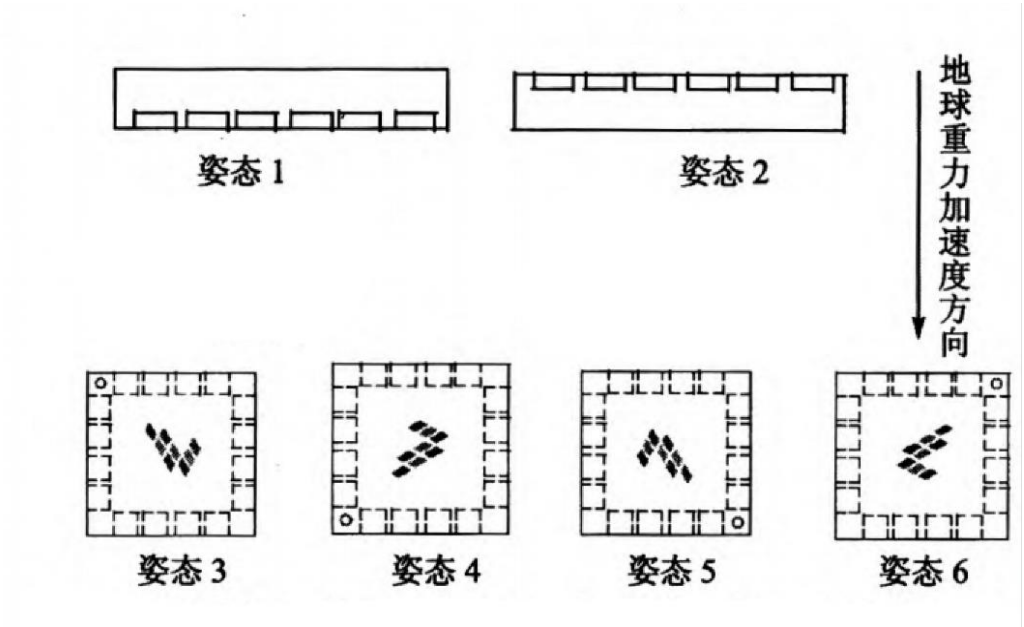


图 4-1 标定原理图

姿态	X 输出/mV	Y 输出/mV	Z 输出/mV
姿态 1	2170	2430	3230
姿态 2	2170	2450	1280
姿态 3	2180	1440	2310
姿态 4	3210	2450	2240
姿态 5	2170	3440	2220
姿态 6	1160	2440	2290

表格 4-1 实测参数

姿态	X 输出	Y 输出	Z 输出
姿态 1	0	0	128
姿态 2	0	0	-128
姿态 3	0	-128	0
姿态 4	128	0	0
姿态 5	0	128	0
姿态 6	-128	0	0

表格 4-2 归一结果

归一化代码如下所示，完整代码可见库文件/CDK66/MEM.h与/CDK66/MEM.c。

```
1.
2.  #define MEMSx_MAX 3220
3.  #define MEMSx_MIN 1140
4.  #define MEMSy_MAX 3460
5.  #define MEMSy_MIN 1430
6.  #define MEMSz_MAX 3240
7.  #define MEMSz_MIN 1280
8.
9.  //-------------------------------------------------------------------------
10. //  @brief         Normalized(-128 to 128)
11. //  @return        Normalize the sampled value
12. //  @data          2020/10/16
13. //  Sample usage:
14. //-------------------------------------------------------------------------
15.
16. void MEMS_Normalized(MMA7260_t *AnalogInput)
17. {
18.
19.     //Low pass filter,removed finally
20.     /*
21.     AnalogInput->x=MEMS_temp.x*0.4+AnalogInput->x*0.6;
22.     AnalogInput->y=MEMS_temp.y*0.4+AnalogInput->y*0.6;
23.     AnalogInput->z=MEMS_temp.z*0.4+AnalogInput->z*0.6;
24.
25.     MEMS_temp.x=AnalogInput->x;
26.     MEMS_temp.y=AnalogInput->y;
27.     MEMS_temp.z=AnalogInput->z;
28.     */
29.
30.     AnalogInput->x = (AnalogInput->x-MEMSx_MIN)*256/(MEMSx_MAX-MEMSx_MIN)-
    128;
31.     AnalogInput->y = (AnalogInput->y-MEMSy_MIN)*256/(MEMSy_MAX-MEMSy_MIN)-
    128;
32.     AnalogInput->z = (AnalogInput->z-MEMSz_MIN)*256/(MEMSz_MAX-MEMSz_MIN)-
    128;
33. }
34.
```

4.5 运动状态判定

4.5.1 运动状态判定方法

根据受力分析，在自由落体中，实验板仅受重力作用，即若此时对MEMS反馈数据进行合成，反馈数据因为接近0的数字。

$$a = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

我们经过对输出数据进行观察，发现实验板是否处于跌落状态与MMA7260三轴加速度传感器的三个输出信号的平方和有关，当处于自由落体状态既跌落状态时输出信号的平方和会接近零，所以我们以平方和是否接近于零作为运动状态的判定依据。

图4-5 静止状态

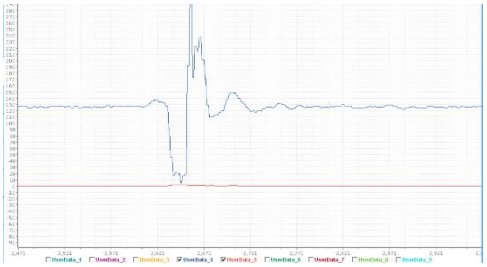


图4-6 跌落状态

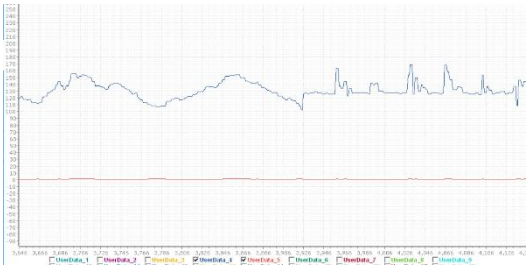


图4-7 移动和冲击状态

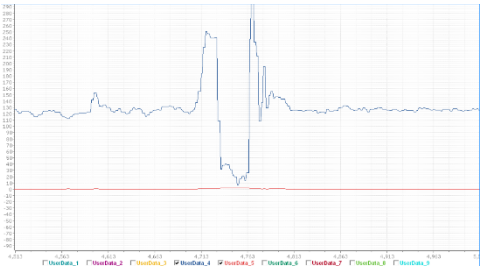


图4-8 斜抛状态

从上述四幅图我们可以看出，在不同运动状态下 MMA7260 三轴加速度传感器的输出信号的平方和具有不同的特征。我们据此判定是否处于跌落状态，由于斜抛在广义上也属于自由落体运动，所以我们便将斜抛和自由下落的判定归为一种，实际上也可以通过对不同特征的检测将此两种状态进行区分（自由下落是先趋近于零再上升，而斜抛是先上升再趋近于零再上升）。

我们将输出信号的平方和定义为力 F，当 F 在 128 左右的范围波动时，我们认为实验板静止；当 F 接近于 0 时我们认为实验板处于自由落体运动状态，此时蜂鸣器蜂鸣；当 F 不处于上述两个状态的时候，我们认为实验板处于移动状态或是受到了外力冲击。

而实际上，由于下落时间较短、数据零位等问题，F 很难完全等于 0，因此我们设定了跌落阈值作为可调参数，当 F 小于阈值时即认为处于跌落状态。

具体运动状态判定程序代码如下所示。

```
float Fsqrt(float x)
{
    float xhalf = 0.5f * x;
    int i = *(int *)&x; // evil floating point bit level hacking
    i = 0x5f375a86 - (i >> 1);
    x = *(float *)&i;
    x = x * (1.5f - (xhalf * x * x));
    return 1 / x;
}
//main 函数
while (1)
{
     serial_out();
     External_Force=Fsqrt((float)(MEMS.x*MEMS.x+MEMS.y*MEMS.y+MEMS.z*MEMS.z)
);
     //printf("force: %d\n",(uint16_t)(External_Force));
     if(abs(External_Force-128)<15)
     {
         MEMS_Status=0;
         BEEP_OFF();
         printf("Static!\n");
     }
     else if(abs(External_Force)<45)
     {
         MEMS_Status=2;
         BEEP_ON();
         printf("Free falling!\n");
     }
     else
     {
         MEMS_Status=1;
         BEEP_OFF();
         printf("Moving!\n");
     }
     //OLED show
     sprintf (MEMSValueX, "%04hd", MEMS.x);
     sprintf (MEMSValueY, "%04hd", MEMS.y);
     sprintf (MEMSValueZ, "%04hd", MEMS.z);

     OLED_P8x16Str(28,2,MEMSValueX);
     OLED_P8x16Str(28,4,MEMSValueY);
     OLED_P8x16Str(28,6,MEMSValueZ);
}
```

4.5.2 数据可视化显示

为了方便对程序进行调整以及判定方法的优化，我们通过串口将数据发送出来并使用上位机进行可视化，便于调试。

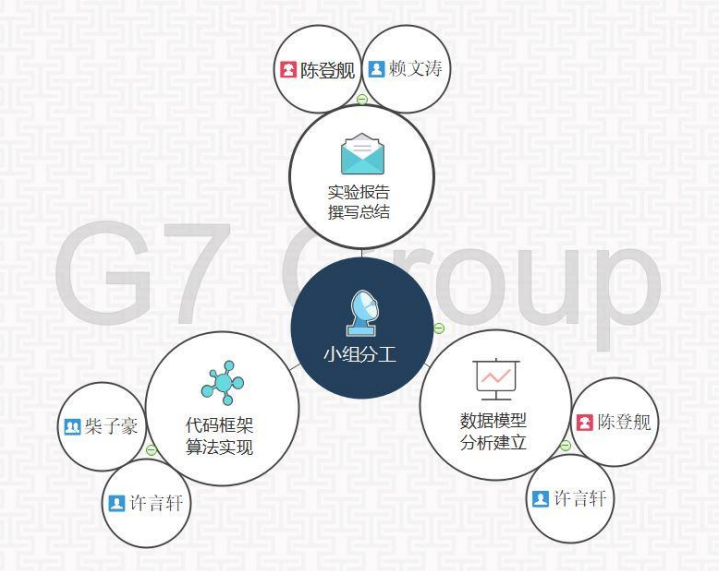
串口发送参数具体代码如下所示。

```
1.  //Send data to the host computer for visualization
2.  void serial_out()
3.  {
4.      static uint8_t data[25];
5.      data[0] = 0xAA;
6.      data[1] = 0xAA;
7.      data[2] = 0xF1;
8.      data[3] = 20;
9.      data[4] = BYTE3(MEMS.x);
10.     data[5] = BYTE2(MEMS.x);
11.     data[6] = BYTE1(MEMS.x);
12.     data[7] = BYTE0(MEMS.x);
13.     data[8] = BYTE3(MEMS.y);
14.     data[9] = BYTE2(MEMS.y);
15.     data[10] = BYTE1(MEMS.y);
16.     data[11] = BYTE0(MEMS.y);
17.     data[12] = BYTE3(MEMS.z);
18.     data[13] = BYTE2(MEMS.z);
19.     data[14] = BYTE1(MEMS.z);
20.     data[15] = BYTE0(MEMS.z);
21.     data[16] = BYTE3(force);
22.     data[17] = BYTE2(force);
23.     data[18] = BYTE1(force);
24.     data[19] = BYTE0(force);
25.     data[20] = BYTE3(MEMS_Status);
26.     data[21] = BYTE2(MEMS_Status);
27.     data[22] = BYTE1(MEMS_Status);
28.     data[23] = BYTE0(MEMS_Status);
29.     data[24] = 0;
30.
31.     for (uint8_t *ptr = data; ptr < data + sizeof(data) - 1; ptr++)
32.     {
33.         data[24] += *ptr;
34.     }
35.     UART_WriteBlocking(UART0, (uint8_t *)&data, sizeof(data));
36. }
```

5. 实验分析与结论

本次实验以MMA7260三轴加速度传感器信号检测为基础，通过对传感器输出信号的特征进行特征提取，进而根据不同输出特征判定实验板所处的不同运动状态。

我们小组基本分工如下图所示：



在本次实验的过程中，我们有不少想法，如在滤波的时候采用低通滤波，但后来发现加上低通滤波虽然对于数据去噪有较好的效果，但会降低跌落的反应速度，最终将低通滤波删去。

在对我们的算法进行鲁棒性检测时，当快速上下移动实验板模拟自由落体状态时，输出的数据也有概率会小于我们设定的跌落阈值，蜂鸣器也会蜂鸣。但是由于实际实验过程中无法进行长距离下落的实验，受力趋于F的波谷并非很明显，在我们实验中，我们逐步增加跌落距离，发现波谷会不断变长，若有足够的跌落空间，可通过对波谷置时间标志位来进行更好的改进。

在对移动和冲击的特征进行分析的时候，我们发现两者的数据波形大致相似，但是运动的波形变化较为舒缓而冲击的变化较迅速，后期我们可以根据此特征对移动和冲击进行区分。

经过本次实验，我们基本熟悉示例代码Ref2的基本思路与代码实现方法，并以此代码为基础进行调整与改进，进行基本库函数的完善与封装，为后续实验积累库函数与便于参照的工程代码，后续的实验内容我们也会在此次实验代码的基础上进行开发。同时，我们也成功的将串口的使用融入到了程序开发中，并对数据进行了可视化输出，这为我们以后的程序开发打下了良好的基础，在以后进行特征提取等类型的功能模块开发时我们可以快速上手，通过将数据可视化快速找到特征。

最后，感谢王冰老师为本门课程的精心准备与悉心讲解，深入浅出的为我们讲解嵌入式开发的诸多知识，将复杂的嵌入式知识网络按照对应模块分割开来，逐一细解，帮助我们拓宽视野，增长见识。