xMotion接口说明

版本	作者	日期	说明
1.0	ycwang@miramems.com	09/23/2016	初版
1.1	ycwang@miramems.com	11/17/2016	增加计步和卡路里清零接口

目录

1 目的

本文主要介绍 xMotion 的系统架构、功能模块、数据类型、应用接口，以及使用实例，从整体结构、功能特点和应用接口等方面详细介绍了本方案的具体内容。本文适合于研发、调试以及希望了解 xMotion 方案的人员阅读，帮助其更好的了解 xMotion。

2 介绍

2.1 系统架构

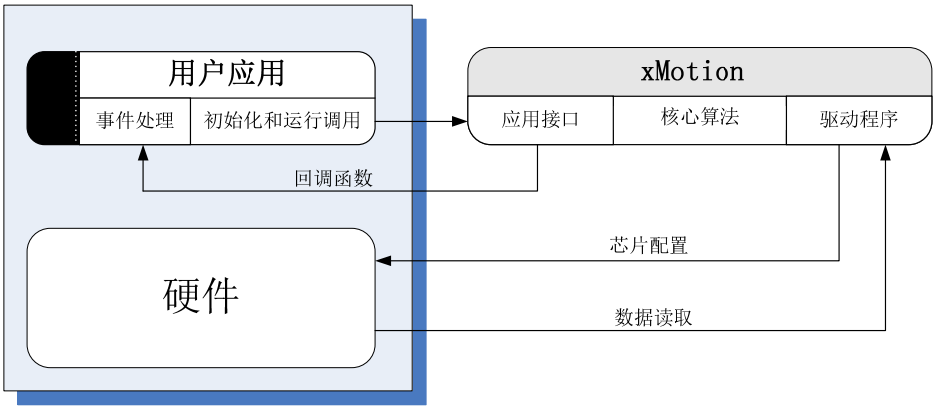


图 1 系统架构

xMotion 作为一种多平台的运动解决方案，具体灵活便捷的移植特点。参见图 1，xMotion 方案独立于平台本身，除必要的通信接口（i2c\spi 读写）仍然需要根据各平台实际项目情况进行实现外，其他核心部分，包括应用接口、核心算法和驱动程序，已经包含在本方案中，免去了用户繁琐的调试工作。用户端只需要实现简单的两个通信读写和一个事件处理的回调函数，然后简单按照流程配置初始化，定时（20Hz）调用运行函数，即可完成本方案的移植。

目前已经在 MTK6260、BCM20736、nRF51822、CC2540、STM32、QC1110、SC6531 和 CD2733 等平台上实现并投入量产。

2.2 功能模块

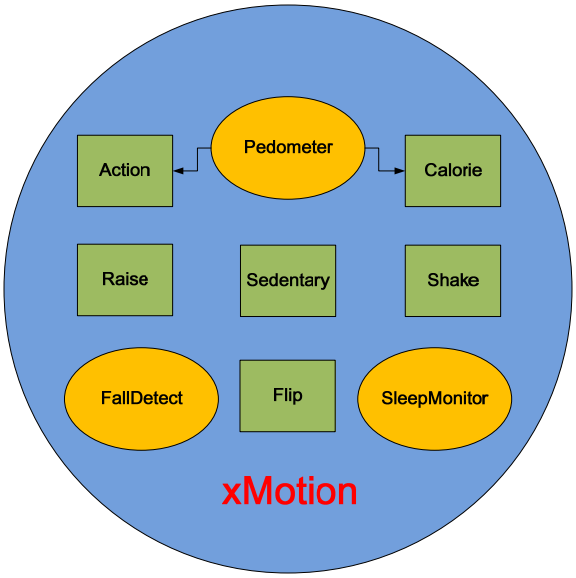


图 2 功能模块

xMotion 是一种主要应用于穿戴式产品的运动解决方案，包括计步、运动状态监测、卡路里计算、跌倒检测、睡眠监测、久坐提醒、抬手亮屏、翻转检测和摇摆检测等功能，其中运动状态监测和卡路里计算依赖于计步。

计步：根据人体运动的加速度特征，通过计步算法来检测移动步数，目前准确度可以做到大于 95%。

跌倒检测：通过追踪产品传感器的数据变化，当检测到身体姿势发生变化时，利用算法对数据进行分析，从而判断佩戴者是否跌倒，目前准确度大于 93%。

睡眠监测：通过监测穿戴者睡眠阶段的数据特征，解析用户的浅睡眠和深度睡眠的累计时间，目前支持未佩戴状态，总体准确度大于 95%。

运动状态监测：通过用户运动过程中的步数变化，实现实时运动状态的监测过程，目前准确度达到 95%。

久坐提醒：通过分析用户坐姿的数据，判断预设时间内是否发生久坐事件，目前准确度 100%。

卡路里：根据公式计算每分钟热量的值，结果取决于用户的身高、体重和步频等参数。目前准确度为 90%。

翻转检测：通过分析翻转过程中的传感器数据变化，利用算法获取翻转事件，目前适合包括三轴和两轴加速器的应用，准确度大于 95%。

抬手检测：通过判断抬手过程中的角度变化，获取抬手事件，目前算法中加入放手事件的检测，总体准确度约 95%。

2.3 算法资源需求

以 STM32 平台的开发环境为例，各功能算法对 Flash 和 RAM 的资源消耗如下：

算法	Flash	RAM	单位
计步、运动状态、卡路里	2532	472	字节
跌倒检测	1302	144	字节
睡眠监测	582	44	字节
久坐提醒	462	272	字节
翻转检测	470	48	字节
抬手亮屏	606	69	字节
摇摆检测	714	66	字节
驱动部分	1612	104	字节

3 数据类型

3.1 寄存器定义

```
#define NSA_REG_SPI_I2C                 0x00
#define NSA_REG_WHO_AM_I                0x01
#define NSA_REG_ACC_X_LSB               0x02
#define NSA_REG_ACC_X_MSB               0x03
#define NSA_REG_ACC_Y_LSB               0x04
#define NSA_REG_ACC_Y_MSB               0x05
#define NSA_REG_ACC_Z_LSB               0x06
#define NSA_REG_ACC_Z_MSB               0x07
#define NSA_REG_MOTION_FLAG             0x09
#define NSA_REG_G_RANGE                 0x0f
#define NSA_REG_ODR_AXIS_DISABLE        0x10
#define NSA_REG_POWERMODE_BW            0x11
#define NSA_REG_SWAP_POLARITY           0x12
#define NSA_REG_INTERRUPT_SETTINGS1     0x16
#define NSA_REG_INTERRUPT_SETTINGS2     0x17
#define NSA_REG_INTERRUPT_MAPPING1      0x19
#define NSA_REG_INTERRUPT_MAPPING2      0x1a
#define NSA_REG_INTERRUPT_MAPPING3      0x1b
#define NSA_REG_INT_PIN_CONFIG          0x20
```

```
#define NSA_REG_INT_LATCH                0x21
#define NSA_REG_ACTIVE_DURATION          0x27
#define NSA_REG_ACTIVE_THRESHOLD         0x28
#define NSA_REG_TAP_DURATION             0x2A
#define NSA_REG_TAP_THRESHOLD            0x2B
#define NSA_REG_ENGINEERING_MODE         0x7f
#define NSA_REG_SENS_COMP                0x8c
#define NSA_REG_MEMS_OPTION              0x8f
#define NSA_REG_CHIP_INFO                0xc0
```

寄存器定义描述了明皜传感器芯片的寄存器地址，详情请参考数据手册。

3.2 数据类型

```
#if ARM_BIT_8
typedef unsigned char    u8_m;
typedef signed   char    s8_m;
typedef unsigned int     u16_m;
typedef signed   int     s16_m;
typedef unsigned long    u32_m;
typedef signed   long    s32_m;
typedef float            fp32_m;
typedef double           fp64_m;
#else
typedef unsigned char    u8_m;
typedef signed   char    s8_m;
typedef unsigned short   u16_m;
typedef signed   short   s16_m;
typedef unsigned int     u32_m;
typedef signed   int     s32_m;
typedef float            fp32_m;
typedef double           fp64_m;
#endif
```

数据类型是根据主芯片处理能力不同而定义不同，如果方案中使用的是 8 位的处理器，则把宏 ARM_BIT_8 设置为 1。

3.3 数据结构

```
typedef struct AccData_tag{
    s16_m ax;
    s16_m ay;
    s16_m az;
}AccData;
```

这是加速计三轴数据值的结构体。

3.4 算法类型

```
typedef enum {
    NONE_X,
    PEDOMETER_X,
    FALL_X,
    SLEEP_X,
    ACTION_X,
    SEDENTARY_X,
    CALORIE_X,
    FLIP_X,
    RAISE_X,
    SHAKE_X
}Algorithm;
```

这是 xMotion 算法类型的枚举类型。

表 1 算法类型

算法类型	数值	描述
NONE_X	0	无
PEDOMETER_X	1	计步算法
FALL_X	2	跌倒检测算法
SLEEP_X	3	睡眠监测算法
ACTION_X	4	运动状态监测算法
SEDENTARY_X	5	久坐提醒算法
CALORIE_X	6	卡路里算法
FLIP_X	7	翻转检测算法
RAISE_X	8	抬手亮屏算法
SHAKE_X	9	摇摆检测算法

3.5 开关类型

```
typedef enum {
 DISABLE_X ,
 ENABLE_X
}SwitchCmd;
```

开关类型是为了开关算法使用。

表 2 命令类型

命令类型	数值	描述
DISABLE_X	0	关闭对应算法
ENABLE_X	1	启动对应算法

3.6 事件类型

```
typedef enum {
    EVENT_NONE,
    EVENT_STEP_NOTIFY,
    EVENT_FALL_NOTIFY,
    EVENT_SLEEP_NOTIFY,
    EVENT_ACTION_NOTIFY,
    EVENT_SEDENTARY_NOTIFY,
    EVENT_CALORIE_NOTIFY,
    EVENT_FLIP_NOTIFY,
    EVENT_RAISE_NOTIFY,
    EVENT_SHAKE_NOTIFY
 }XMOTION_EVENT;
```

事件类型是对应于 xMotion 中各类算法功能。

表 3 事件类型

事件类型	数值	描述
EVENT_NONE	0	无定义
EVENT_STEP_NOTIFY	1	计步事件
EVENT_FALL_NOTIFY	2	跌倒事件
EVENT_SLEEP_NOTIFY	3	睡眠事件
EVENT_ACTION_NOTIFY	4	运动状态事件

EVENT_SEDENTARY_NOTIFY	5	久坐事件
EVENT_CALORIE_NOTIFY	6	卡路里事件
EVENT_FLIP_NOTIFY	7	翻转事件
EVENT_RAISE_NOTIFY	8	抬手事件
EVENT_SHAKE_NOTIFY	9	摇摆事件

3.7 睡眠类型

```
typedef enum {
  SLEEP_AWAKE,
  SLEEP_LIGHT_1,
  SLEEP_DEEP,
  SLEEP_LIGHT_2,
  SLEEP_UNWEAR
}Tsleep;
```

睡眠类型是用来区分各种睡眠状态，默认是 SLEEP_AWAKE 状态。

表 4 睡眠类型

睡眠类型	数值	描述
SLEEP_AWAKE	0	清醒状态
SLEEP_LIGHT_1	1	浅睡眠 1 期状态，前一个状态为清醒状态
SLEEP_DEEP	2	深睡眠状态，前一个状态为浅睡眠 1 期状态
SLEEP_LIGHT_2	3	浅睡眠 2 期状态，前一个状态为深睡眠状态
SLEEP_UNWEAR	4	未佩戴状态，下一个状态只能为清醒状态

3.8 运动状态类型

```
typedef enum {
  MOTIONLESS,
  WALK,
  RUN
}Taction;
```

运动状态类型是用来区分各种运动状态，默认是 MOTIONLESS 状态。

表 5 运动状态类型

运动状态类型	数值	描述
MOTIONLESS	0	静止状态
WALK	1	走路状态
RUN	2	跑步状态

3.9 抬手类型

```
typedef enum {
  RAISE_NONE = 0,
  RAISE_ON   = 1,
  RAISE_OFF  = -1
}Traise;
```

抬手类型是用来区分各种抬手状态，默认是 RAISE_NONE 状态。

表 6 抬手类型

抬手类型	数值	描述
RAISE_NONE	0	无抬手
RAISE_ON	1	抬手亮屏
RAISE_OFF	-1	垂手灭屏

3.10 摇摆类型

```
typedef enum {
  SHAKE_NONE    = 0x00,
  SHAKE_RIGHT   = 0x01,
  SHAKE_LEFT    = 0x02,
  SHAKE_FRONT   = 0x04,
  SHAKE_BACK    = 0x08,
  SHAKE_UP      = 0x10,
  SHAKE_DOWN    = 0x20
}Tshake;
```

摇摆类型是用来区分各种摇摆状态，默认是 SHAKE _NONE 状态。

表 7 摇摆类型

摇摆类型	数值	描述
SHAKE_NONE	0x00	无摇摆
SHAKE_RIGHT	0x01	向右摇动
SHAKE_LEFT	0x02	向左摆动
SHAKE_FRONT	0x04	向前摆动
SHAKE_BACK	0x08	向后摆动
SHAKE_UP	0x10	向上摆动
SHAKE_DOWN	0x20	向下摆动

3.11 函数结构

```
struct xMotion_op_s {
  s8_m (*event_send)(XMOTION_EVENT event, s32_m data_m);
  s8_m (*mir3da_register_read)(u8_m addr, u8_m *data_m);
  s8_m (*mir3da_register_write)(u8_m addr, u8_m data_m);
  s8_m (*mir3da_save_cali_file)(s16_m x, s16_m y,s16_m z);
  s8_m (*mir3da_get_cali_file)(s16_m *x, s16_m *y,s16_m *z);
};
```

函数结构是用来定义注册接口函数的类型。

4 应用程序接口

4.1 流程图

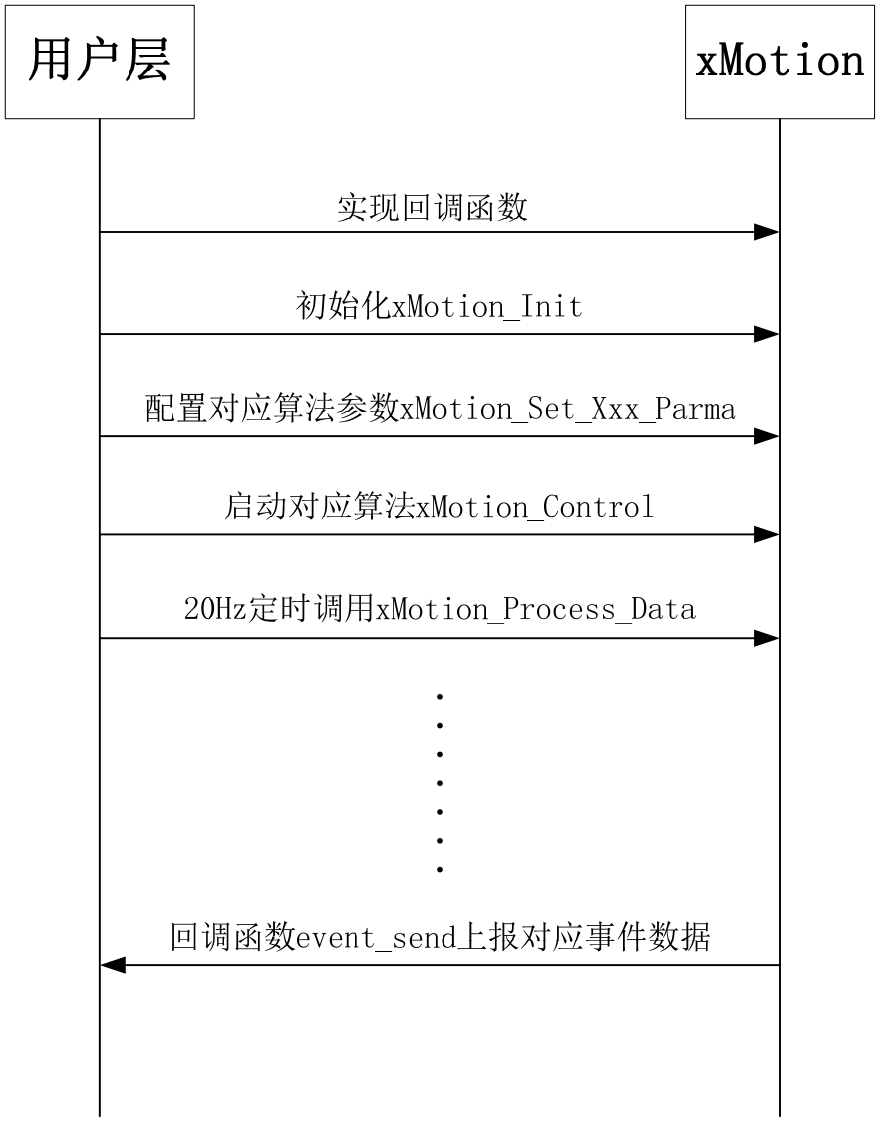


图 3 流程图

在 xMotion 初始化之前，必须提供对应的回调函数来处理方案内部的通信和对应的事件处理，这部分需要用户根据实际情况自行实现；然后，通过初始化接口注册回调函数和配置芯片寄存器，并配置对应算法参数，接着启动对应算法功能；最后，通过定时器调用处理函数来实现算法运行。当算法运行产生状态或数据改变时，则通过已注册的回调函数 event_send 上报对应事件数据。

4.2 回调注册函数

4.2.1 event_send

s8_m (*event_send)(XMOTION_EVENT event, s32_m data_m);

表 8 event_send

描述

简介

这个函数是用来实现处理 xMotion 各类功能事件的，具体实现需要客户根据自己的设计来完成。

参数

event 是 xMotion 的事件类型

data_m 是各类事件对应的数据或者类型

返回值

0 成功

others 失败

实例：

```
s8_m sendEvent(XMOTION_EVENT event,s32_m data_m)
{
  switch(event)
  {
    case EVENT_STEP_NOTIFY:
      printf(" [ step = %ld ].\r\n",data_m);              \\ 当前总步数
      break;
    case EVENT_FALL_NOTIFY:
      printf(" [ fall = %ld ].\r\n",data_m);              \\ 当值为 1 时，发生跌倒
      break;
    case EVENT_SLEEP_NOTIFY:
      printf(" [ sleep = %ld ].\r\n",data_m);             \\当前睡眠状态
      break;
    case EVENT_ACTION_NOTIFY:
      printf(" [ action = %ld ].\r\n",data_m);            \\ 当前运动状态
      break;
    case EVENT_SEDENTARY_NOTIFY:
      printf(" [ sedentary = %ld ].\r\n",data_m);         \\当值为 1 时，发生久坐
      break;
    case EVENT_CALORIE_NOTIFY:
```

```
        printf(" [ calorie = %ld ].\r\n",data_m);          \\ 当前总卡路里
        break;
      case EVENT_FLIP_NOTIFY:
        printf(" [ flip = %ld ].\r\n",data_m);             \\当值为 1 时，发生翻转
        break;
      case EVENT_RAISE_NOTIFY:
        printf(" [ raise = %ld ].\r\n",data_m);            \\ 当前抬放手状态
        break;
      case EVENT_SHAKE_NOTIFY:
        printf(" [ shark = 0x%x ].\r\n",data_m);           \\ 当前摇摆状态
        break;
    default:
       break;
    }
     return 0;
  }
```

4.2.2 mir3da_register_read

s8_m (*mir3da_register_read)(u8_m addr, u8_m *data_m, u8_m len);

表 9 mir3da_register_read

描述

简介

这个函数用来读寄存器操作

参数

addr 是寄存器的地址
data_m 是指向读到的数据的指针
len 是读取的字节个数

返回值

0 成功
others 失败

实例：

```
s8_m readReg(u8_m addr, u8_m *data_m, u8_m len)
{
 s8_m ret;
```

```
    ret = I2C_Read(i2c_bus, ic_addr, addr,data_m, len);
    if(!ret)
      return -1;

    return 0;
  }
```

4.2.3 mir3da_register_write

s8_m (*mir3da_register_write)(u8_m addr, u8_m data_m);

表 10 mir3da_register_write

描述

简介

这个函数用来写寄存器操作

参数

addr 是寄存器的地址

data_m 是写入的值

返回值

0 成功

others 失败

实例：

```
  s8_m writeReg(u8_m addr, u8_m data_m)
  {
    s8_m ret;

    ret = I2C_Write(i2c_bus, ic_addr, addr,&data_m);
    if(!ret)
      return -1;

    return 0;

  }
```

4.2.4 mir3da_save_cali_file

s8_m mir3da_save_cali_file(s16_m x, s16_m y,s16_m z);

表 11 mir3da_save_cali_file

描述

简介

这个函数用来保存的传感器偏移校准值

参数

x 是 x 轴的校准值
y 是 y 轴的校准值
z 是 z 轴的校准值

返回值

0 成功
others 失败

实例：

```
s8_m save_cali_file(s16_m x, s16_m y,s16_m z)
{
 return 0;
}
```

4.2.5 mir3da_get_cali_file

s8_m mir3da_get_cali_file(s16_m *x, s16_m *y,s16_m *z);

表 12 mir3da_get_cali_file

描述

简介

这个函数用来获取已保存的传感器偏移校准值

参数

x 是指向 x 轴的校准值的指针
y 是指向 y 轴的校准值的指针
z 是指向 z 轴的校准值的指针

返回值

0 成功
others 失败

实例：

```
s8_m mir3da_get_cali_file(s16_m *x, s16_m *y,s16_m *z)
```

```
{
 return 0;
}
```

4.3 对外接口函数

4.3.1 xMotion_Init

s8_m xMotion_Init(struct xMotion_op_s *ops);

表 13 xMotion_Init

描述

简介

这个函数用来设置久坐算法的参数

参数

***ops** 是指向 xMotion_op_s 结构体的指针

返回值

0	成功
-1	数据类型错误，请确认 mira_std.h 中数据类型使用是否正确
-2	注册函数错误，请确认注册函数实现是否正确
-3	芯片初始化失败，请确认芯片通信是否正常或芯片是否为明皜产品

4.3.2 xMotion_Clear_Pedometer_Value

void xMotion_Clear_Pedometer_Value(void);

表 14 xMotion_Clear_Pedometer_Value

描述

简介

这个函数用来清零计步数据

参数

无

返回值

无

4.3.3 xMotion_Set_Sedentary_Parma

void xMotion_Set_Sedentary_Parma(u16_m time, u16_m interval);

表 15 xMotion_Set_Sedentary_Parma

描述

简介

这个函数用来设置久坐算法的参数

参数

time 是久坐的触发时间，单位分钟，范围 1 到 120 分钟，默认值 30
interval 是上报事件的间隔时间，单位分钟，范围 0 到 30 分钟，默认值 0

返回值

无

4.3.4 xMotion_Set_Calorie_Parma

void xMotion_Set_Calorie_Parma(u16_m height, u16_m weight);

表 16 xMotion_Set_Calorie_Parma

描述

简介

这个函数用来设置卡路里算法的参数

参数

height 是用户身高，单位厘米，默认值 180
weight 是用户的体重，单位千克，默认值 70

返回值

无

4.3.5 xMotion_Clear_Calorie_Value

void xMotion_Clear_Calorie_Value(void);

表 17 xMotion_Clear_Calorie_Value

描述

简介

这个函数用来清零卡路里数据

参数

无

返回值

无

4.3.6 xMotion_Set_Raise_Parma

void xMotion_Set_Raise_Parma(u8_m dir);

表 18 xMotion_Set_Raise_Parma

描述

简介

这个函数用来设置抬手算法的方向

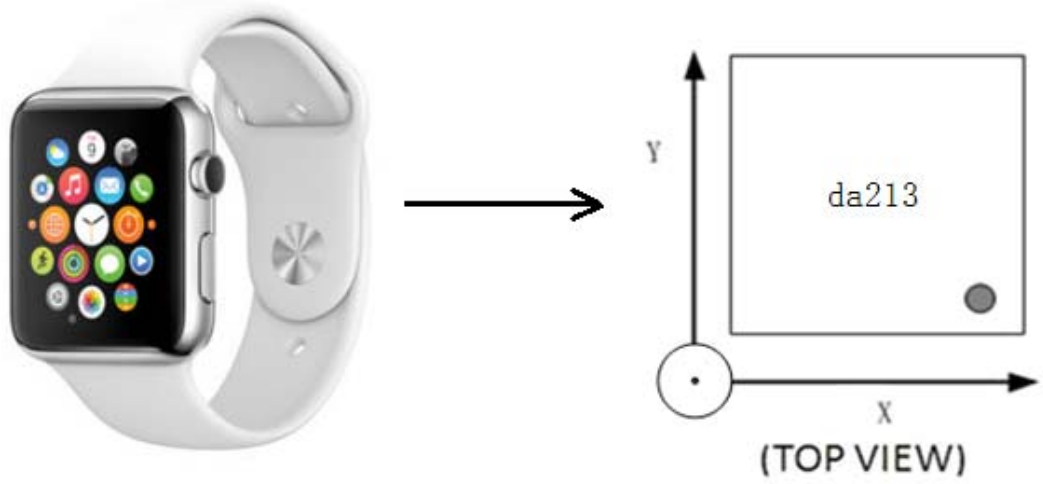
参数

dir 是坐标轴的方向，包括 0 到 7 八个方向值，默认值为 0

返回值

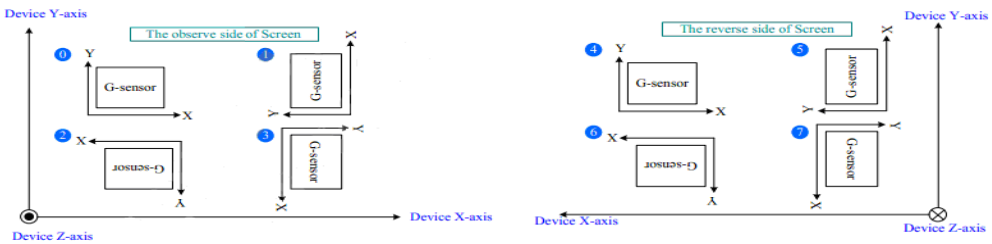
无

这个参数是用来调整手机或手表芯片的方向。默认的方向如下：



如果坐标轴方向不同于默认值，客户需要调整数据的方向。对于方向值的含义，请参阅下面的表格：

Value	Description
0	{x, y, z} => { x, y, z}
1	{x, y, z} => {-y, x, z}
2	{x, y, z} => {-x, -y, z}
3	{x, y, z} => { y, -x z}
4	{x, y, z} => {-x, w. -z}
5	{x, y, z} => { y, x, -z}
6	{x, y, z} => { x. -y, -z}
7	{x, y, z} => {-y, -x, -z}



4.3.7 xMotion_Set_Shake_Parma

void xMotion_Set_Shake_Parma(u8_m level);

表 19 xMotion_Set_Shake_Parma

描述

简介

这个函数用来设置摇摆算法的灵敏度

参数

level 是灵敏度等级，包括 1 到 6 六个等级，默认值为 2

返回值

无

4.3.8 xMotion_Set_Fall_Parma

void xMotion_Set_Fall_Parma(u8_m level);

表 20 xMotion_Set_Fall_Parma

描述

简介

这个函数用来设置跌倒算法的灵敏度

参数

level 是灵敏度等级，包括 1 到 3 三个等级，默认值为 3

返回值

无

4.3.9 xMotion_Control

void xMotion_Control (Algorithm name, SwitchCmd cmd);

表 21 xMotion_Control

描述

简介

这个函数用来开启或关闭对应的算法功能

参数

name 是算法名

cmd 是开关命令

返回值

无

4.3.10 xMotion_Process_Data

void xMotion_Process_Data(void);

表 22 xMotion_Process_Data

描述

简介

这个函数用来负责运行 xMotion，需定时调用，调用频率为 20Hz

参数

无

返回值

无

4.3.11 xMotion_Get_Version

void xMotion_Get_Version(u8_m ver);

表 23 xMotion_Get_Version

描述

简介

这个函数用来查询软件版本号

参数

ver 是指向版本号字符串的指针，例如“1.0.0”

返回值

无

4.3.12 xMotion_QueryControl

SwitchCmd xMotion_QueryControl(Algorithm name);

表 24 xMotion_QueryControl

描述

简介

这个函数用来查询对应算法功能的开关状态

参数

name 是算法名

返回值

使能或者关闭

4.3.13 xMotion_QueryEvent

s32_m xMotion_QueryEvent(XMOTION_EVENT event);

表 25 xMotion_QueryEvent

描述

简介

这个函数用来查询对应事件的数据

参数

event 是算法的事件类型

返回值

事件数据

4.3.14 xMotion_chip_check_id

s8_m xMotion_chip_check_id(void);

表 26 xMotion_chip_check_id

描述

简介

这个函数用来检验明皜传感器芯片的识别号是否正确

参数

无

返回值

0 成功

others 失败（可能原因:芯片没有焊好；芯片 i2c 地址错误；非明皜芯片）

4.3.15 xMotion_chip_read_xyz

s8_m xMotion_chip_read_xyz(s16_m *x,s16_m *y,s16_m *z);

表 27 xMotion_chip_read_xyz

描述

简介

这个函数用来读取 xyz 三轴加速度数值

参数

x 是指向 x 轴的数据的指针
y 是指向 y 轴的数据的指针
z 是指向 z 轴的数据的指针

返回值

0 成功
others 失败

4.3.16 xMotion_buffer_read_xyz

void xMotion_buffer_read_xyz(s16_m *x,s16_m *y,s16_m *z);

表 28 xMotion_buffer_read_xyz

描述

简介

这个函数用来读取因调用 xMotion_Process_Data 而获取的 xyz 三轴加速度数值

参数

x 是指向 x 轴的数据的指针
y 是指向 y 轴的数据的指针
z 是指向 z 轴的数据的指针

返回值

无

4.3.17 xMotion_chip_calibrate_offset

s8_m xMotion_chip_calibrate_offset(void);

表 29 xMotion_chip_calibrate_offset

描述

简介

这个函数用来校准明皜传感器芯片焊接后产生的偏移

参数

无

返回值

0 成功

others 失败

4.3.18 xMotion_chip_power_on

s8_m xMotion_chip_power_on(void);

表 30 xMotion_chip_power_on

描述

简介

这个函数用来启动明皜传感器芯片

参数

无

返回值

0 成功

others 失败

4.3.19 xMotion_chip_power_off

s8_m xMotion_chip_power_off(void);

表 31 xMotion_chip_power_off

描述

简介

这个函数用来关闭明皜传感器芯片

参数

无

返回值

0 成功

others 失败

5 使用实例

```
// 实现 xMotion_op_s 结构体成员函数
s8_m sendEvent (XMOTION_EVENT event,s32_m data_m)
{
    //deal the events
   switch(event)
   {
      case EVENT_STEP_NOTIFY:
                ….
    }
     return 0;
}
s8_m readReg(u8_m addr, u8_m *data_m, u8_m len)
{
 s8_m ret;

 ret = I2C_Read(SENSOR_ACC_I2C_BUS, 0x27<<1, addr,data_m, len);
 //if(!ret)
 //  return -1;

 return 0;
}

s8_m writeReg(u8_m addr, u8_m data_m)
{
 s8_m ret;

 ret = I2C_Write(SENSOR_ACC_I2C_BUS, 0x27<<1, addr, &data_m, 1);
 //if(!ret)
 //  return -1;

 return 0;
}

struct xMotion_op_s     ops_handle = {  sendEvent, readReg, writeReg, NULL, NULL };

void main()
{
```

```
s8_m ret;
//初始化
ret = xMotion_Init(&ops_handle);
if(ret == -1)
{
  //数据类型错误，请检查 mira_std.h 中宏定义 ARM_BIT_8
  return;
}
else if(ret == -2)
{
  //xMotion_op_s 结构体实现不完整，其中 event_send、mir3da_register_read
  和 mir3da_register_write 三个必须实现
  return;
}
else if(ret == -3)
{
  //芯片初始化失败，原因可能是芯片焊接不良、I2C 通信问题等
  return;
}

//对应功能的参数配置
xMotion_Set_Sedentary_Parma(30,1)          //久坐 30 分钟提醒，上报间隔 1 分钟
xMotion_Set_Calorie_Parma(180,70);         //身高 180cm，体重 70kg
xMotion_Set_Raise_Parma(0);                //抬手亮屏坐标轴方位 0
xMotion_Set_Shake_Parma(2);                //摇摆灵敏度等级为 2
xMotion_Set_Fall_Parma(3);                 //跌倒灵敏度等级为 3

//打开对应功能
xMotion_Control(PEDOMETER_X, ENABLE_X);    //计步
xMotion_Control(FALL_X, ENABLE_X);         //跌倒
xMotion_Control(SLEEP_X, ENABLE_X);        //睡眠
xMotion_Control(FLIP_X,ENABLE_X);          //翻转
xMotion_Control(RAISE_X,ENABLE_X);         //抬手
xMotion_Control(SHAKE_X,ENABLE_X);         //摇摆
xMotion_Control(SEDENTARY_X,ENABLE_X);     //久坐

//以下两个功能使用前必须首先打开计步功能
xMotion_Control(ACTION_X, ENABLE_X);       //运动
```

```
    xMotion_Control(CALORIE_X,ENABLE_X);          //卡路里

    while(1)
     {
      //需要以 20Hz 的频率调用 xMotion_Process_Data
      if(msTime>50)     //50ms
      {
         msTime = 0;
         xMotion_Process_Data();
      }
      msTime++;
    }
}
```