智能防盗锁

组员：曾永健、张振明、丁帅

指导老师：余晓敏

目录

绪论

随着城市建设的加快，道路的建设开始跟不上城市发展的需要，使得市区越发拥挤。即便政府号召绿色出行的交通方式，也很难改变道路拥挤的问题。除了公共交通外，人们通过步行或者骑行作为代步方式。在共享单车出现之前，若选择骑行作为出行方式，就涉及到一个如何防盗的问题。自行车电单车这类非机动车，很难有像私家车一样有一个妥善的停车场负责安放。加上这类非机动车本身价格也比私家车地连续多，因此人们也很难接受使用价格高昂的防盗设备进行防盗窃行为。虽然现在人们开始习惯使用共享单车作为出行方式， 根据广州市统计局针对广州市十一区 5000 名市民调查的一项数据表示，“在前往 1 公里左右的目的地时，采用的交通方式是什么？其中选择“步行”的有 48.9%，“骑自行车或电动车”的有 30.0%，其他则选择网约车或者打出租车等。问及选择慢行交通出行的原因，76.8%的市民表示“方便”是首要因素”。就我国的目前路况来看，骑单车或电动车出行，是最适宜的出行方式，人们也开始习惯于骑车作为出行方式。也有部分人群喜欢骑行属于自己的单车或电动车，尤其针对现在的大学生白领阶层。这类人群往往是把骑车作为一种生活方式，并不只是把他作为出行方式。加上人们生活水平的提高，自行车也是一种奢侈品，这类自行车通常价格较为昂贵的山地车和进口自行车。而车主多数是使用简单的铁锁以及 U 型合金车锁进行防护。但是这样的防盗措施是远远不够的，在我国交通工具盗窃案发率导致许多公民造

成了个人私有财产的损失。而这类盗窃案的侦破率，能够让公民及时拿回自己的财产却少之又少。这对我国的治安环境稳定造成了严重的影响。就例如在本校，我们学生的自行车被盗屡见不鲜，报案后能够找回的，却寥寥无几。因此，如何防范单车或电动车被盗，并且如何第一时间的告知车主单车或电动车出现异常，让车主及时应对这突发状况。因此本小组通过学习相关的知识后设计了一种能够通过发送信息或者拨打电话报警，并且允许用户通过蓝牙模块进行开锁控制等操作的智能锁，希望能够用于自行车和电动车上。

1.智能防盗锁的基本结构图

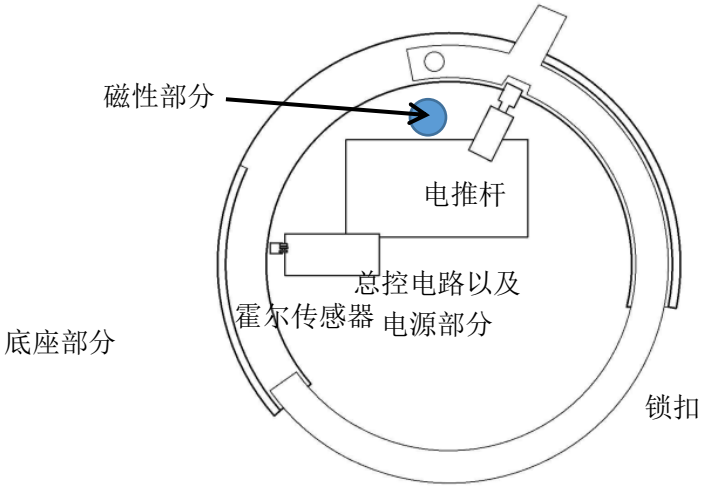

图 1-1 防盗锁内部结构图

本项目设计的智能防盗锁的内部结构图如图 1-1 所示，整个智能防盗锁由底座，锁扣，霍尔传感器，总控电路以及电推杆组成。底座，锁扣以及电推杆组成锁的基本构造，其开关由霍尔传感器以及总控电路完成。当锁处于打开状态的时候，总控电路放出高电平使电推杆收缩，使其锁扣可以自由活动，实现锁扣的释放，从而进行锁的开启；

当锁处于关闭状态或即将处于关闭状态时，此时的电推杆处于长时间未收缩状态，只要当锁扣到达底部，推杆可以刚好嵌入锁扣一侧的凹槽，使锁扣无法进行自由的释放，从而达到锁的关闭。而霍尔传感器作用是当锁即将处于关闭状态的时候，即锁扣移动使其探测测的磁场强度发生变化时，可知此时锁处于关闭状态。本次设计锁的 3D 结构图如图 1-2 所示。

图 a：锁扣与底座 3D 模型图

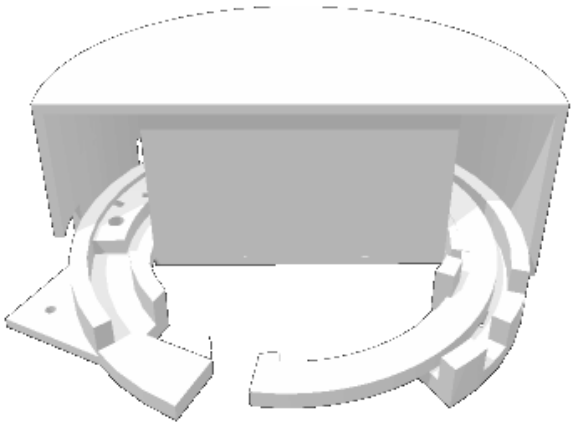

图 b：智能防盗锁 3D 模型简图

图 1-2 智能防盗锁 3D 图像

2.智能防盗锁的硬件组成

本次设计的智能锁的硬件由以下几个部分组成：控制核心STM32F103C8T6、陀螺仪 MPU9250、蓝牙 4.0 模块 BT-05、迷你 GSM 模块 SIM800L、霍尔开关 AH3144E 以及微型电推杆组成。其中 stm32 通过蓝牙模块与用户手机进行通信并且接受指令。陀螺仪用于姿态检测。当自行车的姿态发生异常时，通过 GSM 模块发送信息或拨打电话发出警报，或者通过蜂鸣器发出声音警报。总体的程序框图如下图所示。

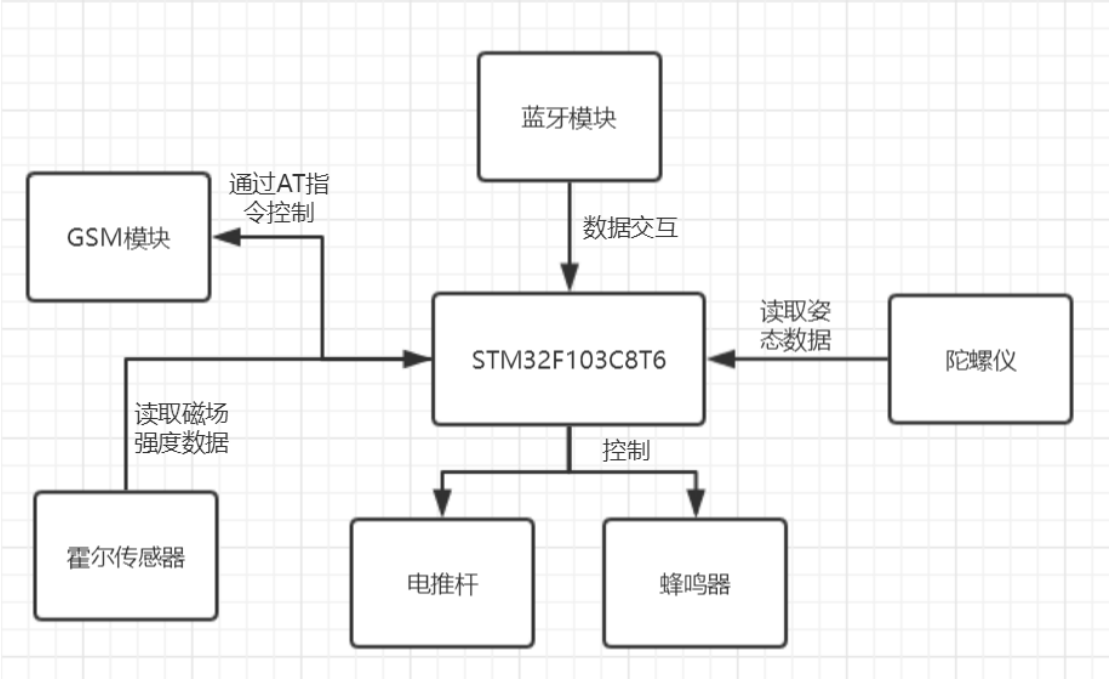

图 2-1 硬件系统框图

2.1 MCU 主控模块

本次设计采用 stm32f103C8t6 作为主控模块。STM32F103C8T6 是 32 位基于 ARM 核心的带 512K 字节闪存的微控制器 ，工作频率为 72MHz，内置高速存储器（高达 512K 字节的闪存和 64K 字节的 SRAM），丰富的增强 I/O 端口和联接到两条 APB 总线的外设。该芯

片上的外设有：

1.2 组 12 位的 AD 转换器，1 微秒转换时间（多达 10 个通道）。

2.7 通道 DMA，支持定时器、ADC、SPI、IIC 和 USART。

3.4 个定时器，可用于定时和输出 PWM 波。

4.多达 9 个通信通道，包括三个串口，两个 IIC，2 个 SPI，CAN 总线以及 USB2.0 全速接口。

本次使用的是 STM32F103C8T6 的最简核心板，部分原理图如图所示。

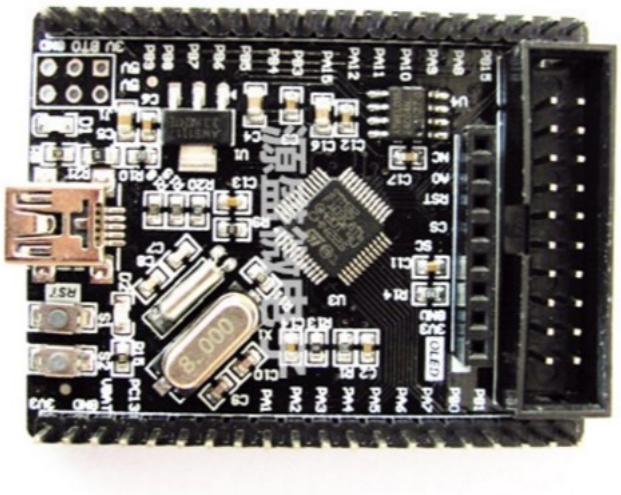

图 2-2　STM32F103C8T6 最简核心板外观图

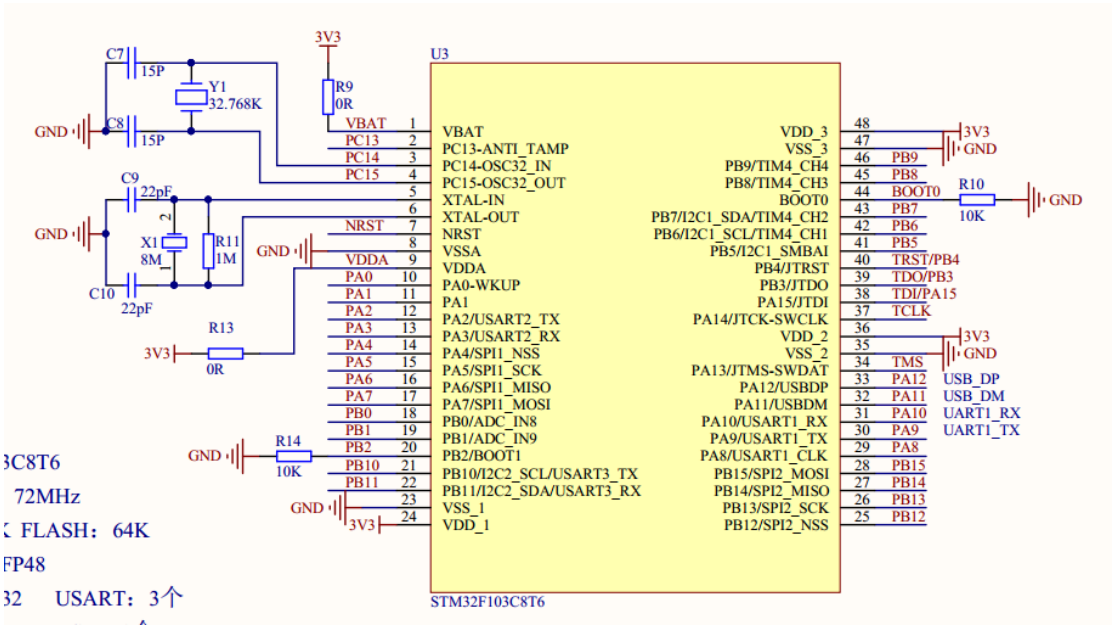

图 2-3 STM32F103C8T6 部分原理图

2.2 GSM 模块

本次设计中使用 GA6 mini GSM 模块报警。GA6 mini GSM 模块是一款基于 SIM800L 的 GSM 通信模块。该模块支持串口通信，用户可以通过串口发送 AT 指令控制其打电话、发短信等。拥有尺寸小，功耗低和工作温度范围宽等优点。该模块参数如表所示。

表 2-1 GSM 模块相关参数

尺寸	22.8× 16.8× 2.5mm
工作温度	-30°Cto+80°C
工作电压	3.3V-4.2V
开机电压	>3.4V
待机平均电流	<3ma

GA6 mini GSM 模块支持 GSM/GPRS 四个频段，包括 850,900,1800,1900MHZ，同时支持语音通话、SMS 短信、数字音频和模拟音频，同时支持 HR，FR，EFR，AMR 语音编码。另外还支持标准 AT 命令及 Ai Thinker 扩展命令。该模块使用的命令支持标准 AT 和 TCP/IP 命令接口。模块外观如下图所示。模块的管脚定义如表

所示。

表 2-2 GSM 模块管脚定义

VCC	电源输入引脚 5V -28V
GND	电源地
UTX	模块发送 （TTL 电平）
URX	模块接收 （TTL 电平）
HTX	串口升级接口
HRX	串口升级接口
INT	用于控制模块是否进入低功耗模式，高电平退出，低电平进入
EN	使能脚，拉高使能电源芯片
M-\M+	麦克风输入
PWR	开机键
E_L/E_R	耳机接口
NET	网络指示灯

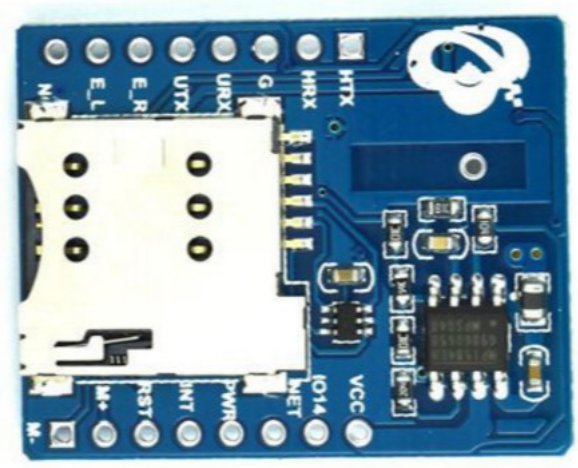

图 2-4 GSM 模块外观图

本次设计主要使用 GSM 模块的语音发送以及报警功能，因此模块将模块串口和 stm32 的串口 2 连接，用于发送 AT 指令。

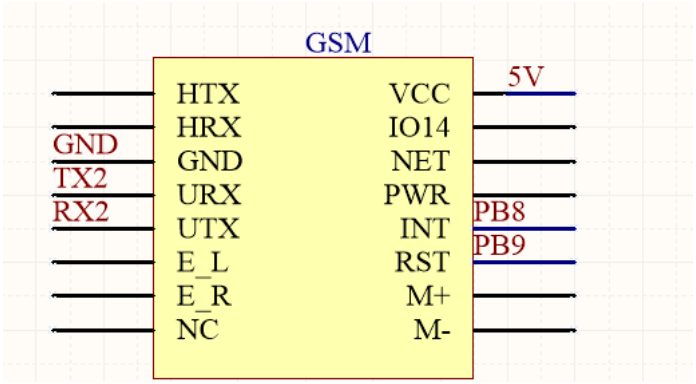

图 2-5 GSM 模块连接原理图

2.3 蓝牙模块 BT-05

本次设计采用的蓝牙模块是 DX-BT05 4.0，如图所示。该蓝牙模块采用美国 TI 公司 CC2541 芯片，配置 56Kb 空间，遵循 V4.0BLE 蓝牙规范。模块支持 AT 指令，用户可根据需要更改串口波特率、设备名称、配对密码等参数，使用灵活。另外该模块支持 UART 接口，并支持 SPP 蓝牙串口协议，具有成本低、体积小、功耗低、收发灵敏性高等优点，只需配备少许的外围元件就能实现蓝牙无线收发数据。BT-05 模块的部分参数如下表所示。

表 2-3 蓝牙模块参数表

蓝牙协议	Bluetooth Specification V4.0 BLE
工作频率	2.4GHz ISM band
灵敏度	<-84dBm at 0.1% BER
传输速率	异步通信: 6 kbps 同步通信: 6 kbps
功耗	自动休眠模式待机电流 400uA-1.5mA, 传输时 8.5mA
供电电源	+3.3VDC 50mA
工作温度	-40℃-150℃

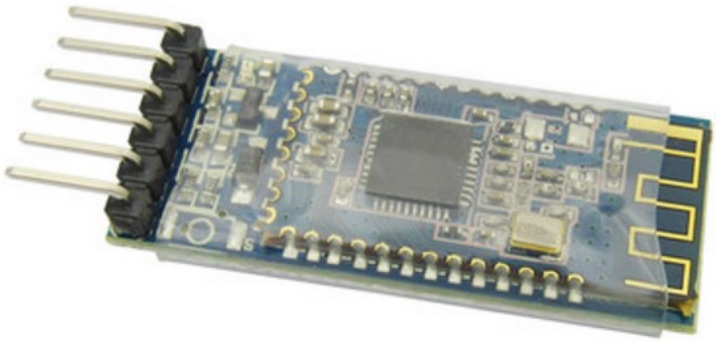

图 2-6 BT-05 蓝牙模块外观图

本次使用的是带底板的 BT-05，模块仅引出了电源口 VCC 和 GND、串口通信口 TX 和 RX、使用口 EN 以及状态口 STATE。使用

蓝牙模块前通过串口转 USB 模块连接至电脑并且配置波特率等，并且设置了连接密码。完成后连接到 stm32 的串口 1，用串口 1 控制蓝牙模块。具体连接如下。

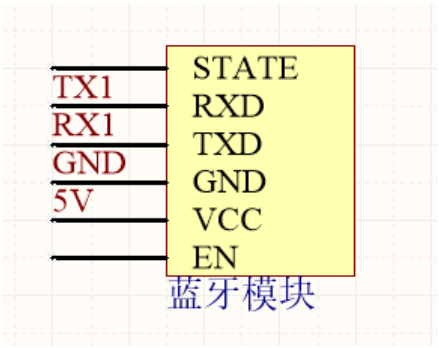

图 2-7 蓝牙模块连接原理图

2.4 陀螺仪 MPU9250 模块

为了检测自行车的姿态，智能锁配置了 MPU9250 型陀螺仪，如图所示。MPU9250 是九轴陀螺仪(三轴陀螺仪+三轴加速度+三周磁场)，该模块可以输出运动时产生的加速度、角速度、磁场强度，同时可通过配置寄存器改变数据输出的频率。MPU9250 的具体参数如下表所示。

表 2-4 MPU9250 部分参数

供电电源	3-5v（内部低压差稳压）
通信方式	标准 IIC/SPI 通信协议
内置 AD 转换器	16 位
陀螺仪范围	±250 500 1000 2000 °/s
加速度范围	±2±4±8±16g
磁场范围	±4800uT

图 2-8 MPU9250 外观图

MPU9250 模块内部配置了 DMP（Digital motion Processer），DMP 会将陀螺仪的加速度、角速度和磁场等数据融合计算之后输出姿态角数据，从而反向推算物体实际运动的情况。同时该模块内部配置了 512 字节的 FIFO，当陀螺仪获得数据后会先存在 FIFO 中等待用户编程读出，可通过相关寄存器控制将那些数据写入 FIFO 中。

陀螺仪 MPU9250 支持 SPI 协议和 IIC 协议进行读取数据。本次使用的 IIC 协议。接线如下图所示。

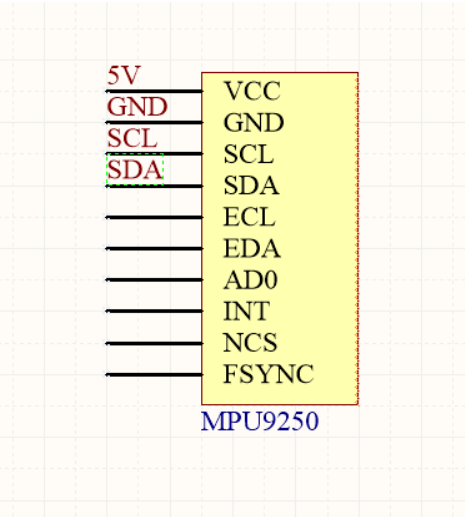

图 2-9 MPU9250 接线原理图

2.5 霍尔开关 AH3144E

本次设计的智能锁利用霍尔开关 AH3144E 检测是否上锁。AH3144E 是由电压调整器，霍尔电压发生器，差分放大器，史密特

触发器和集电极开路的输出级组成的磁敏传感电路，其输入为磁感应强度，输出是一个数字电压讯号。它是一种单磁极工作的磁敏电路，适合于无触点开关、位置检测等情况。该款霍尔开关工作温度范围为-40~150℃，并且只有三个引脚，分别是 VCC、GND 和信号输出口，接线简单。AH3144E 的功能框图如图所示。

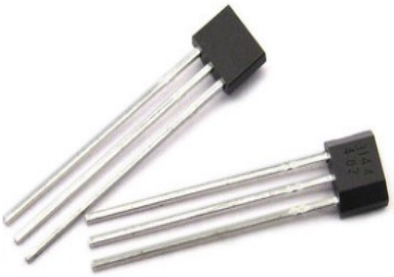

图 2-10 霍尔开关 AH3144E 外观图

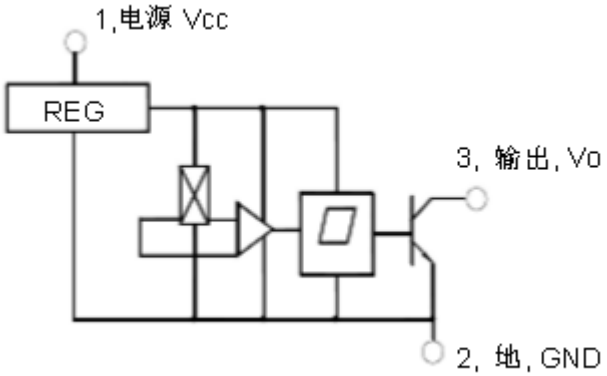

图 2-11 霍尔开关 AH3144E 功能框图

AH3144E 通电后默认输出高电平，当遇到磁场时输出变为低电平，因此配合磁铁能够检测出智能锁是否锁上。但是由于输出高电平时电压过低，stm32 检测不到，因此要通过上拉电阻将其输出电压增大。具体的连接方式如图所示。

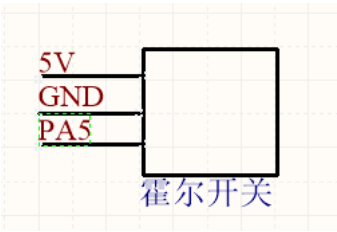

图 2-12 霍尔开关接线原理图

2.6 电推杆和蜂鸣器

本设计的电推杆为 3-6V 的小电压电推杆，如图所示。当通电时电推杆会收缩，用于控制锁的开和关。为了驱动电推杆，使用三极管

驱动电路。声音报警方面使用有源蜂鸣器，当姿态不正常时发出警报。具体的连接如下图所示。

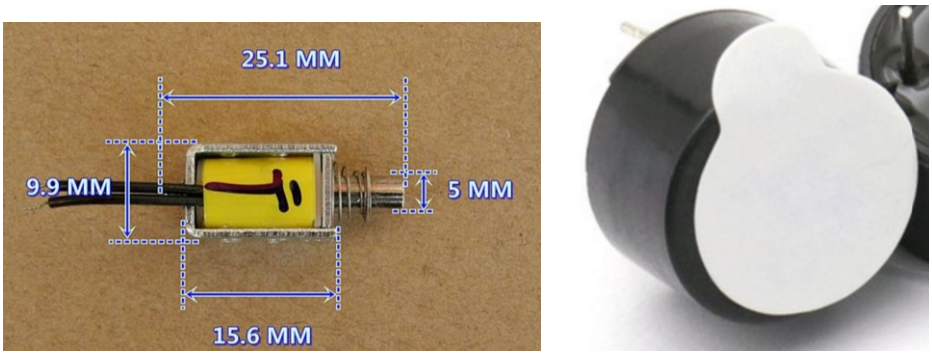

图 2-13 电推杆和蜂鸣器

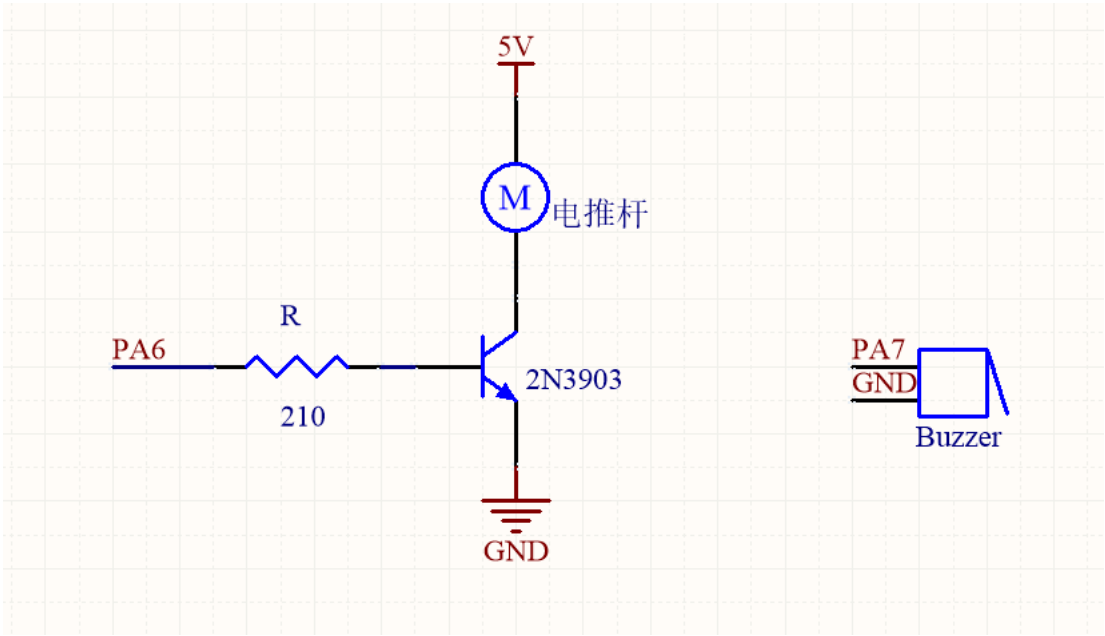

图 2-14 电推杆和蜂鸣器接线原理图

3 程序设计

3.1 整体程序分析

本次设计的智能锁的功能是要实现姿态检测，并且在姿态不正常时通过发短信或者拨打电话向用户发出警报。另外用户可以通过蓝牙进行无线开锁，并且进行相关配置。因此可将整个程序分为多个部分：

陀螺仪姿态读取部分、蓝牙交互部分、蜂鸣器报警部分、GSM 模块报警部分、姿态以及霍尔传感器判断部分以及电推杆伸缩部分。

由于在使用 MPU9250 时发现该款传感器对数据读取速度有一定的要求。当程序有超过 400ms 或以上的延时的时候陀螺仪输出有问题。这是因为陀螺仪会将获得的数据存入 FIFO，而用户读取速度过慢的话会导致 FIFO 溢出。因此程序中不能有过多的延时。结合以上分析的情况，本次智能锁程序使用 uC/OS-III 实时操作系统，通过在系统中创建多个任务完成各部分工作，并且互相之间不会因为延时而阻碍，正好符合本次程序设计的需求。程序系统框图如下。

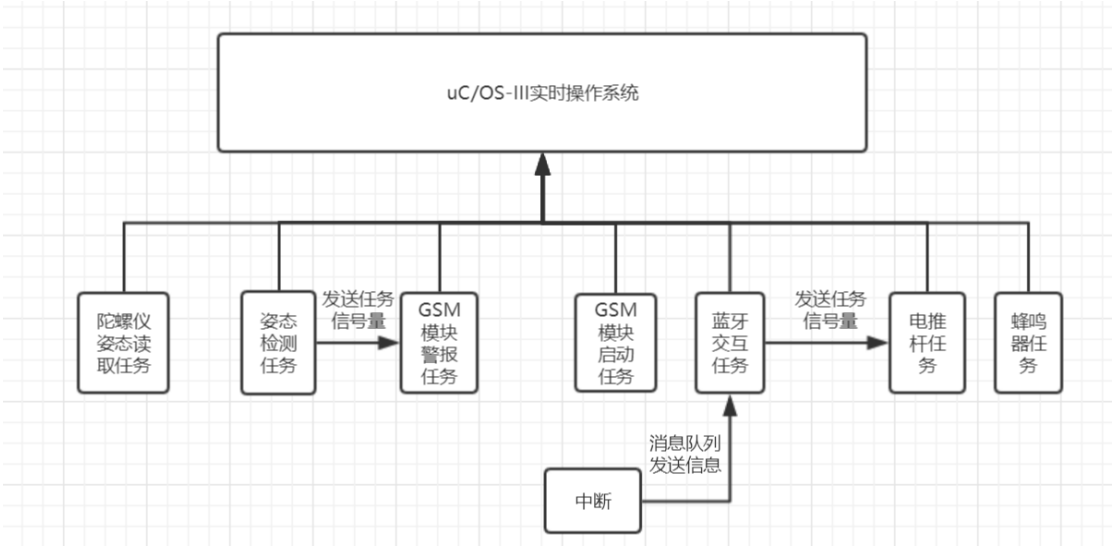

图 3-1 程序系统框图

3.2 uC/OS-III

uC/OS-III 是一个开源、可裁剪、绝大部分由 C 语言编写、可读性强的实时操作系统，由 Micrium 团队编写。该系统是一个抢占式多任务处理内核，即用户可以在该系统中用于可以创建无数多个优先级不同的任务。其中低优先级的任务运行时会被高优先级的任务会抢

占，当高优先级任务执行完后才会返回执行低优先级的任务。因此在运行时能够保证高优先级的任务不会因为别的任务的延时而一直无法执行。在一些不想进行任务切换的位置，如串口传输过程中，为了防止因为任务切换而打断，可通过进入临界代码段防止任务切换。

uC/OS-III 支持许多的内核对象，如任务、堆栈、信号量、事件标志组、消息队列、消息、互斥信号量、内存分区、软件定时器等。通过这些内核对象可以实现任务之间的协调和通信，将所有的任务连接起来。

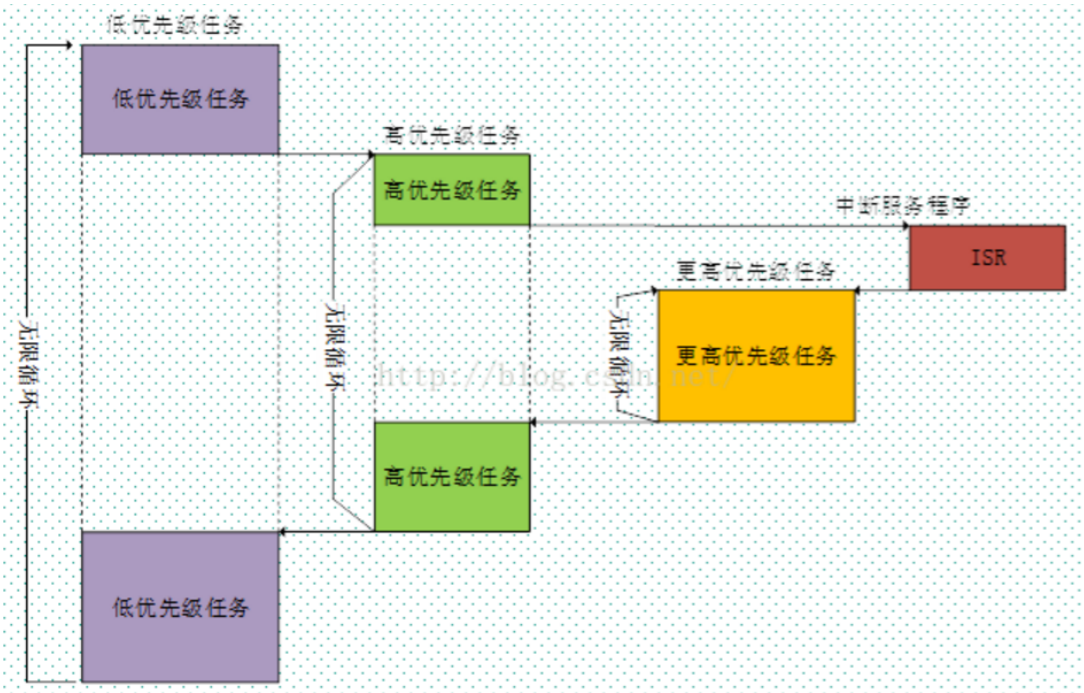

图 3-2 实时操作系统中任务的运行情况

3.3 任务编写

3.3.1 陀螺仪姿态读取任务

陀螺仪读取任务是整个锁的关键，并且由于延时太久会导致数据出错甚至程序奔溃。因此在读取将该任务的优先级设置为最高，且每次执行后仅延时 100ms，保证数据的准确性。陀螺仪数据读取加速度

计输出的三轴加速度，同时使能内部 DMP，利用其计算姿态角后读取数据。从而获得加速度和姿态角。

3.3.2 蓝牙交互任务

蓝牙交互任务作用是与用户手机进行交互。该任务一开始会一直等待消息队列的信号。当蓝牙模块接收到手机发送的信息后会触发中断，将得到的信息存入字符，然后进行判断，并且通过消息队列向蓝牙交互任务发送信息。人接收到信息后进行判断，若为相关指令则执行对应的操作。本次定义了指令格式，用于提供给用于进行操作，如下表所示。该任务流程图如图所示。

表 3-1 蓝牙模块指令表

指令	操作
AT1	解锁
AT2	开/关 GSM 警报
AT3	开/关声音警报
AT4	切换 GSM 警报模式

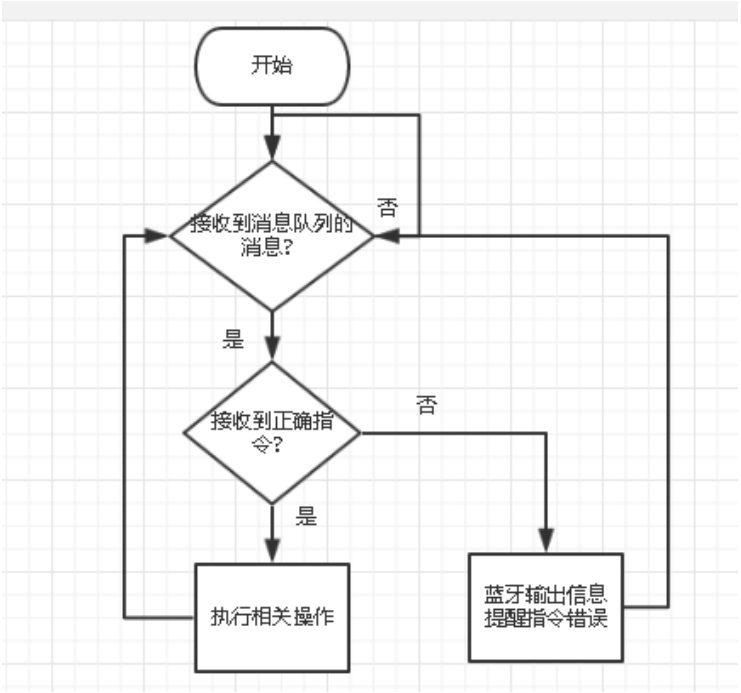

图 3-3 蓝牙交互任务流程图

3.3.3 姿态检测任务

姿态检测任务是在获取到姿态角、加速度数据和霍尔传感器的数据之后进行判断并且进行相应的操作。当锁处于锁上状态时，检测获得的姿态数据从而判断锁是否处于正常状态，若处于非正常状态，分两种情况：1、声音提醒功能开启，则将事件标志组的 beep 位置 1；2、GSM 提醒功能开启，则向 GSM 模块启动任务发送任务信号量，并且利用实现创建的软件定时器延时，以保证 GSM 提醒一次后的一段时间内不会再次提醒。

若锁处于非上锁状态，则一直检测霍尔传感器的输入。当输入为低电平时进行时。持续 3 秒以上则认为锁处于上锁状态。任务的流程图如下。

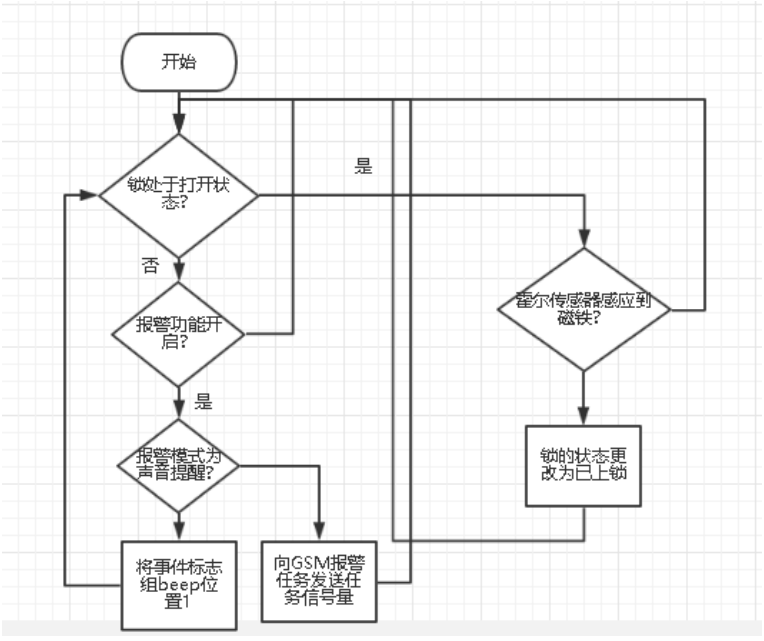

图 3-4 姿态检测任务流程图

3.3.4 GSM 模块启动任务

由于 GSM 模块本身涉及到的功能较为复杂，因此在使用时会有较

多的不确定因素，因此程序中特地创建了 GSM 模块启动任务。GSM 模块启动任务用于在开机后检测 GSM 模块是否能够成功通信。开机后该任务会检测 GSM 模块，若检测到，将事件标志组的 gsm 标志位置 1，并且将自身任务挂起。如果检测不到 GSM 则会重复控制 GSM 模块进行重启，并且检测，直到检测到 GSM 模块为止。该任务流程图如下。

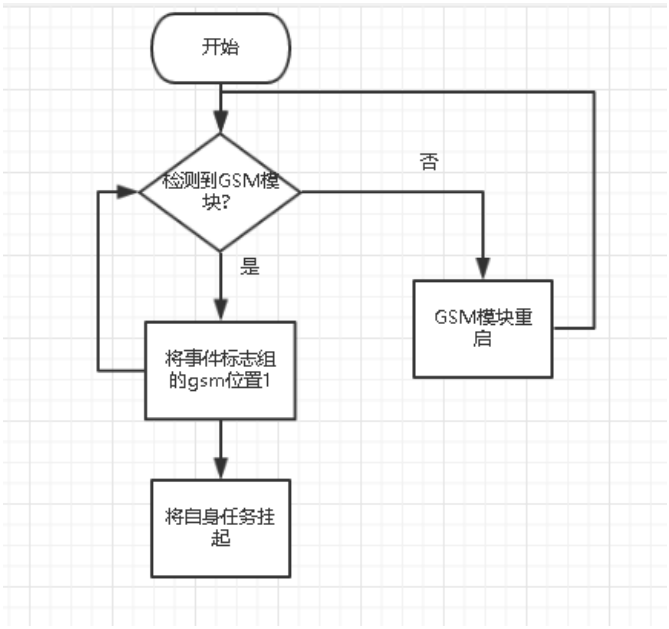

图 3-5 GSM 模块启动任务流程图

3.3.5 GSM 模块警报任务

GSM 模块警报任务用于在姿态不正常时通过发送短信或拨打电话通知用户。任务一开始会等待事件标志组的 GSM 位置 1。若标志位置 1 后任务进入循环。进入循环后等待任务信号量，若接收到后进行报警，并且等待下一个任务信号量。该任务的流程图如下。

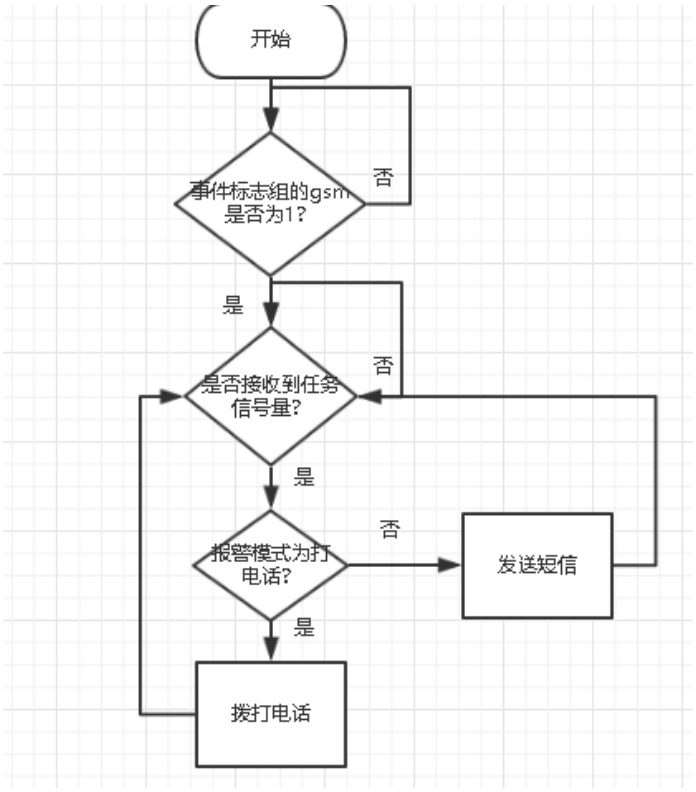

图 3-6 GSM 模块报警任务流程图

3.3.6 电推杆任务

电推杆任务是用于控制电推杆的伸和缩从而实现锁的开和关。该任务开始后会等待任务信号量，若接收到任务信号量则控制电推杆收缩一秒，使锁打开，然后电推杆回复自然状态。程序流程图如下。

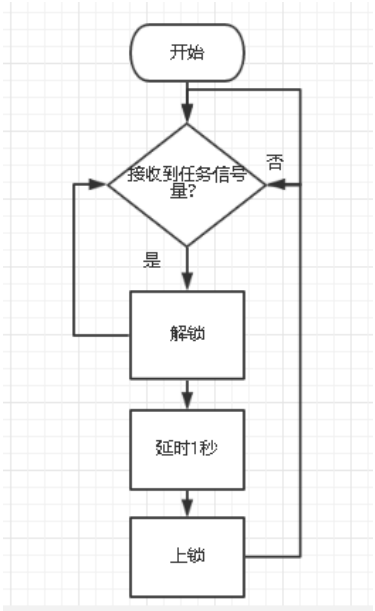

图 3-7 电推杆任务流程图

3.3.7 蜂鸣器任务

蜂鸣器任务用于在姿态不正常时发出声音警报。该任务开始后会一直等待事件标志组的 beep 位置 1，若该标志位置 1 则以一定频率发出声音警报。程序流程图如下。

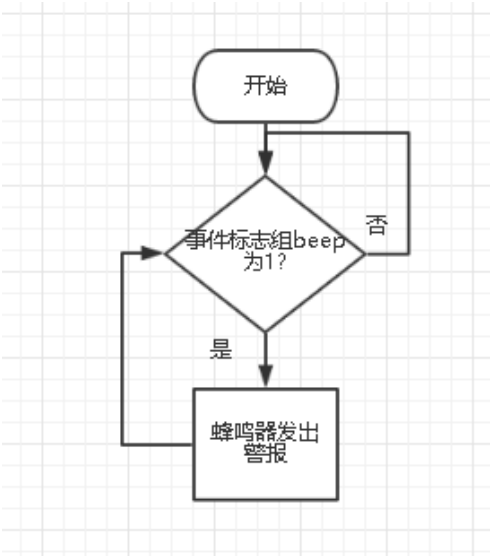

图 3-8 蜂鸣器任务流程图

结论与展望

本小组以研究和制作一款能够用于自行车或者电动车上的智能锁为目标，学习了相关的知识，通过结合蓝牙模块、GSM 模块、陀螺仪 MPU9250 等传感器制作了一款能够报警，并且利用蓝牙控制的智能锁。利用该款智能锁，用户可以实现利用手机解锁，进行一些配置修改。并且在车姿态不稳定时，能够打电话或者发信息向用户报警，也能够发出声音警报。创新点在于：

1、锁的结构相对简单，因此在想对锁做修改时较为方便；

2、可以通过手机蓝牙进行解锁，不许再带钥匙。另外由于设置了蓝牙连接密码配对，因此就算没有带手机，接其他人的手机依旧可以开锁，且不怕之后被他人用自己的手机随意控制。

3、实现远距离发短信提醒或拨打电话提醒，这样的提醒方式更为直接；

4、uC/OS-III 实时操作系统的移植使锁的稳定性更佳。

本次设计的智能锁能够在功能上能够满足一定需求，但是仍然存在许多的问题，因此对于本次设计的锁未来的展望如下：

1.本次设计的锁的结构较为简单，没有考虑材料坚硬度还有防水的问题。因此之后会在材料和防水结构上深入研究，使锁能够满足实际的使用需要。

2.智能锁的运行需要电源，因此在使用时要保证锁的持续续航，否则会影响使用。若设计成换电池或者充电则会影响坚固性和防水性，因此之后会考虑利用太阳能电池或者是其他的磁生电等能够持续

供电的方式为锁供电。

3.由于能力和时间有限，没有完成锁的配套 APP 的编写，因此在实际使用中利用蓝牙助手发送指令执行相关操作，并且蓝牙模块在于手机配对时设置了密码。这样的方式的优点是保证用户只要有手机就能通过蓝牙助手控制智能锁开锁。然而发送指令的操作过于繁琐，因此后期会尝试完成配套 APP，简化用户的操作。

4.本次设计的锁能够实现远程通过打电话或者发短信对用户进行提醒，但如果自行车失窃的情况下，没有相应的定位功能帮助用户找回车辆。因此之后会尝试添加 GPS 定位功能，帮助用户找回失窃车辆。

通过本次的项目的，加深了我组成员对 GSM 模块、陀螺仪、机械结构、单片机、uC/OS-III 实时操作系统以及 APP 编程的认识，增强了我们的动手能力，也对团队合作和协调分工越发熟悉。。

附录

智能锁的部分代码：

```
int  main (void)
{
    OS_ERR  err;
    OSInit(&err);                                                /* Init uC/OS-III.
*/
    OSTaskCreate((OS_TCB      *)&AppTaskStartTCB,                    /* Create the start task
*/
                 (CPU_CHAR   *)"App Task Start",
                 (OS_TASK_PTR ) AppTaskStart,
                 (void       *) 0,
                 (OS_PRIO     ) APP_TASK_START_PRIO,
                 (CPU_STK    *)&AppTaskStartStk[0],
                 (CPU_STK_SIZE) APP_TASK_START_STK_SIZE / 10,
                 (CPU_STK_SIZE) APP_TASK_START_STK_SIZE,
                 (OS_MSG_QTY  ) 20u,
                 (OS_TICK     ) 0u,
                 (void       *) 0,
                 (OS_OPT      )(OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                 (OS_ERR     *)&err);

    OSStart(&err);                                             /* Start multitasking (i.e.
give control to uC/OS-III). */


}


static  void  AppTaskStart (void *p_arg)
{

    CPU_INT32U  cpu_clk_freq;
    CPU_INT32U  cnts;
    OS_ERR      err;


   (void)p_arg;

    BSP_Init();                                                /* Initialize BSP functions
*/
    CPU_Init();
```

```
    cpu_clk_freq = BSP_CPU_ClkFreq();                                   /* Determine SysTick reference
freq.                       */
    cnts = cpu_clk_freq / (CPU_INT32U)OSCfg_TickRate_Hz;           /* Determine nbr SysTick
increments                        */
    OS_CPU_SysTickInit(cnts);                                        /* Init uC/OS periodic time
src (SysTick).              */

    Mem_Init();                                                                 /* Initialize Memory
Management Module                   */
#if OS_CFG_STAT_TASK_EN > 0u
    OSStatTaskCPUUsageInit(&err);                                    /* Compute CPU capacity with
no task running            */
#endif

    CPU_IntDisMeasMaxCurReset();

     OSFlagCreate ((OS_FLAG_GRP  *)&flag_grp,         //指向事件标志组的指针
                                (CPU_CHAR     *)"FLAG For Gesture detection",  //事件标志组
的名字
                                (OS_FLAGS      )0,                 //事件标志组的初始值
                                (OS_ERR       *)&err);                        //返回错误类型


     OSTmrCreate ((OS_TMR              *)&my_tmr,              //软件定时器对象
                        (CPU_CHAR            *)"MySoftTimer",        //命名软件定时器
                        (OS_TICK              )60,                     //定时器初始值,频率为
1HZ
                        (OS_TICK              )0,                    //定时器周期重载值
                        (OS_OPT               )OS_OPT_TMR_ONE_SHOT, //周期性定时
                        (OS_TMR_CALLBACK_PTR  )TmrCallback,          //回调函数
                        (void                *)0,        //传递实参给回调函数
                        (OS_ERR              *)&err);

    OSTaskCreate((OS_TCB     *)&AppTaskLEDTCB,
                 (CPU_CHAR   *)"App Task LED",
                 (OS_TASK_PTR ) AppTaskLED,
                 (void       *) 0,
                 (OS_PRIO     ) APP_TASK_LED_PRIO,
                 (CPU_STK    *)&AppTaskLEDStk[0],
                 (CPU_STK_SIZE) APP_TASK_LED_STK_SIZE / 10,
                 (CPU_STK_SIZE) APP_TASK_LED_STK_SIZE,
                 (OS_MSG_QTY  ) 0u,
                 (OS_TICK     ) 0u,
```

```
                (void       *) 0,
                (OS_OPT      )(OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                (OS_ERR     *)&err);

    OSTaskCreate((OS_TCB     *)&AppTaskMPUTCB,
                (CPU_CHAR   *)"App Task MPU",
                (OS_TASK_PTR ) AppTaskMPU,
                (void       *) 0,
                (OS_PRIO     ) APP_TASK_MPU_PRIO,
                (CPU_STK    *)&AppTaskMPUStk[0],
                (CPU_STK_SIZE) APP_TASK_MPU_STK_SIZE / 10,
                (CPU_STK_SIZE) APP_TASK_MPU_STK_SIZE,
                (OS_MSG_QTY  ) 0u,
                (OS_TICK     ) 0u,
                (void       *) 0,
                (OS_OPT      )(OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                (OS_ERR     *)&err);
       OSTaskCreate((OS_TCB     *)&AppTaskCHECKTCB,
                (CPU_CHAR   *)"App Task CHECK",
                (OS_TASK_PTR ) AppTaskCHECK,
                (void       *) 0,
                (OS_PRIO     ) APP_TASK_CHECK_PRIO,
                (CPU_STK    *)&AppTaskCHECKStk[0],
                (CPU_STK_SIZE) APP_TASK_CHECK_STK_SIZE / 10,
                (CPU_STK_SIZE) APP_TASK_CHECK_STK_SIZE,
                (OS_MSG_QTY  ) 0u,
                (OS_TICK     ) 0u,
                (void       *) 0,
                (OS_OPT      )(OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                (OS_ERR     *)&err);

     OSTaskCreate((OS_TCB     *)&AppTaskBLETCB,
                (CPU_CHAR   *)"App Task BLE",
                (OS_TASK_PTR ) AppTaskBLE,
                (void       *) 0,
                                    (OS_PRIO     ) APP_TASK_BLE_PRIO,
                (CPU_STK    *)&AppTaskBLEStk[0],
                (CPU_STK_SIZE) APP_TASK_BLE_STK_SIZE / 10,
                (CPU_STK_SIZE) APP_TASK_BLE_STK_SIZE,
                (OS_MSG_QTY  ) 20u,
                (OS_TICK     ) 0u,
                (void       *) 0,
                (OS_OPT      )(OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
                (OS_ERR     *)&err);
```

```
  OSTaskCreate((OS_TCB     *)&AppTaskLOCKTCB,
               (CPU_CHAR   *)"App Task LOCK",
               (OS_TASK_PTR ) AppTaskLOCK,
               (void       *) 0,
                                    (OS_PRIO     ) APP_TASK_LOCK_PRIO,
               (CPU_STK    *)&AppTaskLOCKStk[0],
               (CPU_STK_SIZE) APP_TASK_LOCK_STK_SIZE / 10,
               (CPU_STK_SIZE) APP_TASK_LOCK_STK_SIZE,
               (OS_MSG_QTY  ) 0u,
               (OS_TICK     ) 0u,
               (void       *) 0,
               (OS_OPT      )(OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
               (OS_ERR     *)&err);

 OSTaskCreate((OS_TCB     *)&AppTaskGSMTCB,
               (CPU_CHAR   *)"App Task GSM",
               (OS_TASK_PTR ) AppTaskGSM,
               (void       *) 0,
                                    (OS_PRIO     ) APP_TASK_GSM_PRIO,
               (CPU_STK    *)&AppTaskGSMStk[0],
               (CPU_STK_SIZE) APP_TASK_GSM_STK_SIZE / 10,
               (CPU_STK_SIZE) APP_TASK_GSM_STK_SIZE,
               (OS_MSG_QTY  ) 0u,
               (OS_TICK     ) 0u,
               (void       *) 0,
               (OS_OPT      )(OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
               (OS_ERR     *)&err);

     OSTaskCreate((OS_TCB     *)&AppTaskBEEPTCB,
               (CPU_CHAR   *)"App Task BEEP",
               (OS_TASK_PTR ) AppTaskBEEP,
               (void       *) 0,
                                    (OS_PRIO     ) APP_TASK_BEEP_PRIO,
               (CPU_STK    *)&AppTaskBEEPStk[0],
               (CPU_STK_SIZE) APP_TASK_BEEP_STK_SIZE / 10,
               (CPU_STK_SIZE) APP_TASK_BEEP_STK_SIZE,
               (OS_MSG_QTY  ) 0u,
               (OS_TICK     ) 0u,
               (void       *) 0,
               (OS_OPT      )(OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
               (OS_ERR     *)&err);
```

```
        OSTaskDel ( & AppTaskStartTCB, & err );

}

void TmrCallback (OS_TMR *p_tmr, void *p_arg) //软件定时器 MyTmr 的回调函数
{
      OS_ERR      err;
      count++;
      if(count < delay_minute)
    {
            OSTmrStart ((OS_TMR   *)&my_tmr,
                  (OS_ERR   *)err);
        }
    else
      notification1 = 1;
}


static  void  AppTaskMPU ( void * p_arg )
{
    OS_ERR      err;
    CPU_SR_ALLOC();
    (void)p_arg;
    while (1) {                                          /* Task body, always written as an
infinite loop.       */
            OS_CRITICAL_ENTER();
            Get_Gesture();
    //  printf("lock:%d",lock);
         //   printf("Pitch:%f    ",Pitch);
         printf("%d",GPIO_ReadInputDataBit(Hall_Port,Hall_Pin));
            OS_CRITICAL_EXIT();

          OSTimeDly(100,OS_OPT_TIME_DLY,&err);
    }
}

static void  AppTaskCHECK( void * p_arg)
{
      OS_ERR    err;
      u8 i = 0;
    (void)p_arg;
```

```
    while(1)
{
if( lock = 1 && Pitch > 50)
    {
          if(alarm == 1)
             {OSFlagPost ((OS_FLAG_GRP  *)&flag_grp,
                 (OS_FLAGS      )beep,
                 (OS_OPT        )OS_OPT_POST_FLAG_SET,
                 (OS_ERR       *)&err);}


             if(notification == 1 && notification1 == 1)
             {
             OSTaskSemPost((OS_TCB  *)&AppTaskGSMTCB,                                    //目标
任务
                                                 (OS_OPT                        )OS_OPT_POST_NONE,
//没选项要求
                                                 (OS_ERR  *)&err);
                OSTmrStart ((OS_TMR   *)&my_tmr,
                    (OS_ERR   *)err);
             }
             }
    if(lock == 0 && GPIO_ReadInputDataBit(Hall_Port,Hall_Pin) == 0)
    {
          i++;
          if(i == 20)
             {
                 lock = 1;
                 i = 0;
             }
    }

          OSTimeDly(200,OS_OPT_TIME_DLY,&err);
        }
    }


static  void  AppTaskLED ( void * p_arg )
{
    OS_ERR err;
    uint8_t i = 0;
    char c = 'a';
  (void)p_arg;
```

```
    while(1){
        i = Check_Gsm(300,3);
        if(i == 1)
        {
            printf("Sucessfully dectect GSM!");
            GSM_OK();
            OSFlagPost ((OS_FLAG_GRP  *)&flag_grp,
                (OS_FLAGS      )gsm,
                (OS_OPT        )OS_OPT_POST_FLAG_SET,
                (OS_ERR       *)&err);
            OSTimeDly(5000,OS_OPT_TIME_DLY,&err);
            OSTaskSuspend ( 0, & err );
        }
        else
        {

            GSM_FAIL();
            //printf("Can't detect GSM");
            reset();
            OSTimeDly(1000,OS_OPT_TIME_DLY,&err);
        }
}
    }

static void  AppTaskBLE ( void * p_arg )
    {
     OS_ERR err;
     u8 *command;
     int i = 0;
     OS_MSG_SIZE  size;
     CPU_SR_ALLOC();
     (void)p_arg;

     while(1)
     {

        command = OSTaskQPend ((OS_TICK        )0,
                               (OS_OPT         )OS_OPT_PEND_BLOCKING,
                               (OS_MSG_SIZE   *)&size,
                               (CPU_TS        *)0,
                               (OS_ERR        *)&err);
    if(ble == 0)
        {
            if((command[0] == password[0]) && (command[1] == password[1]) &&  (command[2] ==
```

```
password[2])&&
                  (command[3] == password[3]) && (command[4] == password[4])&& (command[5] ==
password[5]))
               {
                    ble = 1;
                    printf("Password correct");
               }
               else if(command[0] == 'A' && command[1] == 'T')
               { printf("Please enter passwaord to connect first");}
               else
               {
               printf("Wrong password");}

         }
    else if(ble == 1)
         {
          if(command[0] == 'A' && command[1] == 'T')
          {
               switch(command[2])
               {
                    case '1':
                    {
          OSTaskSemPost((OS_TCB  *)&AppTaskLOCKTCB,
                                             (OS_OPT   )OS_OPT_POST_NONE,
                                             (OS_ERR  *)&err);
                         printf("sucessful operation!");
                         break;
                    }

                    case '2':
                    {
                         notification = 1 - notification;
                         printf("sucessful operation!");
                         break;
                    }
                    case '3':
                    {
                         alarm = 1 - alarm;
                         printf("sucessful operation!");
                         break;
                    }

                    case '4':
                    {
```

```
                    mode = 1 - mode;
                    printf("sucessful operation!");
                    printf("%d",mode);
                    break;
                }

                case '5':
                {
                    ble = 0;
                    printf("The bluetooth is locked");
                    break;
                }
                default:
                {
                     printf("Wrong command!");
                    break;
                }
            }
     }
       else
          {printf("Wrong command!");}
        OSTimeDly(100,OS_OPT_TIME_DLY,&err);
     }
}
    }

    static  void  AppTaskLOCK( void * p_arg )
    {
      OS_ERR err;
      (void)p_arg;

        while(1)
        {
                OSTaskSemPend ((OS_TICK   )0,
                                        (OS_OPT    )OS_OPT_PEND_BLOCKING,
                                        (CPU_TS   *)0,
                                        (OS_ERR   *)&err);
        Unlock();
                lock = 0;
                OSTimeDly(1000,OS_OPT_TIME_DLY,&err);
        Lock();
        }
    }
```

```
static  void  AppTaskGSM ( void * p_arg )
    {
        OS_ERR err;

        (void)p_arg;
        OSFlagPend ((OS_FLAG_GRP *)&flag_grp,
               (OS_FLAGS     )gsm,
               (OS_TICK      )0,
               (OS_OPT       )OS_OPT_PEND_FLAG_SET_ALL |OS_OPT_PEND_FLAG_CONSUME,
               (CPU_TS       *)0,
               (OS_ERR       *)&err);

        while(1)
        {
            OSTaskSemPend ((OS_TICK   )0,
                                        (OS_OPT    )OS_OPT_PEND_BLOCKING,
                                        (CPU_TS    *)0,
                                        (OS_ERR    *)&err);

              if(mode == 0)
                 {
                     Call(phone_number , 1000);
                     notification1 = 0;
             OSTimeDly(1000,OS_OPT_TIME_DLY,&err);
                 }

                 if(mode == 1)
                 {
                      Send_Message(phone_number,"The bike is abnormal");
                        OSTimeDly(1000,OS_OPT_TIME_DLY,&err);
                        notification1 = 0;
                 }
                    }
            }


    static void  AppTaskBEEP ( void * p_arg )
    {
     OS_ERR err;
     (void)p_arg;

        while(1)
        {
```

```
            OSFlagPend ((OS_FLAG_GRP *)&flag_grp,
              (OS_FLAGS     )beep ,
              (OS_TICK      )0,
              (OS_OPT       )OS_OPT_PEND_FLAG_SET_ALL |OS_OPT_PEND_FLAG_CONSUME,
              (CPU_TS      *)0,
              (OS_ERR      *)&err);

            BEEP(300);
            }
}
```