OV_watch

Tasks

`HardwareInitTask`:43
`LvHandlerTask`:8
`WDOGFeedTask`:42
`IdleEnterTask`:40
`StopEnterTask`:41
`KeyTask`:24
`ScrRenewTask`:9
`SensorDataTask`:9
`HRDataTask`:9
`ChargPageEnterTask`:9
`MessageSendTask`:9
`MPUCheckTask`:10
`DataSaveTask`:10

`MPUCheckTask`

1.检测mpu是否启用
2.检测mpu是不是水平，如果是的话，将 `wrist_state` 设置为 `WRIST_UP`（抬起状态）
3.如果不是，判断之前是不是水平状态。如果是的话现在就是从水平到不是水平，检测是不是具体页面(`ui_HomePage` 、 `ui_MenuPage` 或 `ui_SetPage`),发送消息给 `Stop_MessageQueue` ，触发休眠

`HRDataUpdateTask`

如果当前是 `ui_HRPage` 页面，通知 `IdleBreak_MessageQueue` 不进入休眠。唤醒心率传感器。测试心率传感器是不是链接正常，就是使用iic进行读取是不是正常返回。心率50到120之间正常更新

SensorDataUpdateTask

检测按键，读取`if(osMessageQueueGet(Key_MessageQueue,&keystr,NULL,0)==osOK)`如果是按键1，页面返回，检查是不是`ui_MenuPage`，如果是就关闭心率传感器。让其他传感器进入休眠状态
如果是按键2，回到底部页面，传感器休眠

IdleEnterTask

空闲状态就亮度5，空闲状态打断就重置空闲计数器，恢复亮度

StopEnterTask

检测到`Stop_MessageQueue`有消息传来，关闭屏幕，关闭传感器，暂停所有任务，禁用看门狗。
`使能设备进入STOP模式以降低功耗。此时设备将进入低功耗状态`

如果有中断打破低功耗，配置时钟，喂狗。
判断手腕检测功能是否启用。判断是不是水平。如果是水平且此时手腕处于放下状态，那么就把手腕状态更新为抬起且唤醒设备

如果设备不处于水平状态（`hor`为假），并且当前手腕状态为抬起（`WRIST_UP`），则将手腕状态更新为放下（`WRIST_DOWN`），重置空闲计时器（`IdleTimerCount = 0`），并跳转到`sleep`标签，使设备重新进入休眠模式。

判断其他唤醒条件，不满足继续休眠

唤醒之后触发消息更新

IdleTimerCallback

`用于检测用户是否长时间未操作设备`

通过`IdleTimerCount`增加，当`IdleTimerCount`到了某个值发送消息给`Idle_MessageQueue`关闭屏幕，再经过一段时间就通知`Stop_MessageQueue`

进入低功耗模式

MessageSendTask

串口任务，触发中断的时候这个任务会对消息进行判断

KeyTask

按键按下进行相关动作处理

keystr=1：发送给Key_MessageQueue keystr=1，IdleBreak_MessageQueue IdleBreakstr=0。
按键2:

- 如果当前页面是主页（ui_HomePage），则通过osMessageQueuePut向Stop_MessageQueue发送消息，通知其他任务进入休眠状态。
- 如果当前页面不是主页，则设置keystr为2，并通过osMessageQueuePut向Key_MessageQueue和IdleBreak_MessageQueue发送消息，通知其他任务按键 2 被按下，并打断空闲状态

HardwareInitTask

初始化硬件和lvgl

DataSaveTask

收到DataSave_MessageQueue，进行数据保存到EEPROM

Messagequeues

osMessageQueueId_t Key_MessageQueue;

`osMessageQueueId_t Idle_MessageQueue ;/*当系统检测到用户长时间未操作时，可以通过Idle_MessageQueue触发屏幕关闭、背光熄灭等操作*/`

`osMessageQueueId_t Stop_MessageQueue ;// 发送系统休眠消息到Stop_MessageQueue`

`osMessageQueueId_t IdleBreak_MessageQueue ;`

`osMessageQueueId_t HomeUpdata_MessageQueue ;` 更新硬件参数

`osMessageQueueId_t DataSave_MessageQueue ;`

翻腕亮屏

1.在上面 `MPUCheckTask` 的任务里面，当手腕反转的时候会变成writup的状态
2.RTC唤醒时钟后会判断MPU是不是使能了。如果使能，判断MPU是不是水平状态且对比上一次状态，如果是从非水平到水平就继续亮屏
3.如果不是就继续睡眠

如何判断水平

ROLL和RICH角，根据MPU6050加速度计算的

任务函数

`User_Tasks_Init`： 初始化queue和task

`TaskTickHook`：系统时钟节拍中断钩子函数，

看门狗

1. WDOG采用外置的原因是，想要做睡眠低功耗，那么使用MCU内部的看门狗关闭不了，只能一直唤醒喂狗，否则就要重启，那么这样就失去了睡

> 眠的意义了；

页面管理

使用一个大小为6的线性栈进行管理，并通过一个top指针进行操作。top指针指向页面的下一个。

弹出逻辑:top指针--
弹出之后如果还为空则入栈home和menu界面

I2C

使用软件I2C，为了更方便移植

`voidSDA_Input_Mode(iic_bus_t*bus)`

`voidSDA_Output_Mode(iic_bus_t*bus)`

`voidIICStart(iic_bus_t*bus)`

typedef struct
{
	GPIO_TypeDef * IIC_SDA_PORT;
	GPIO_TypeDef * IIC_SCL_PORT;
	uint16_t IIC_SDA_PIN;
	uint16_t IIC_SCL_PIN;
	//void (*CLK_ENABLE)(void);
}iic_bus_t;

`voidIICStart(iic_bus_t*bus)`

> SDA 1
> SCL 1

> SDA 0
> SCL0

`voidIICStop(iic_bus_t*bus)`

> SCL 0
> SDA 0
> SCL 1
> SDA 1

`IICSendNotAck`

> SDA 1
> SCL 1
> SCL 0

`voidIICSendByte(iic_bus_t*bus,unsignedcharcSendByte)`

> SCL 0

每个模块会配置单独的IIC

```
iic_bus_t AHT_bus =
{
	.IIC_SDA_PORT = GPIOB,
	.IIC_SCL_PORT = GPIOB,
	.IIC_SDA_PIN  = GPIO_PIN_13,
	.IIC_SCL_PIN  = GPIO_PIN_14,
};
```

`0x70`
`0xBE`
`0x08`
`0x00`

BL24C02： EEPROM存储芯片

`检查EEPROM的第一个位置和第二个位置*/`

`uint8_tEEPROM_Check(void)`：检查eeprom是不是正常工作
保存日期和步数数据

> 1.从 BL24C02 EEPROM 的地址 `0x00` 开始读取 2 个字节的数据，并将结果存储在 `check_buff` 数组中。
> 2.检查是不是0x55和0xAA
> 3.如果不是的话就在这两个位置写入0x00和0x55

保存硬件状态，mpu6050的手腕检测设置是否启用

LSM303:六轴传感器（加速度计和磁力计）

`设置MPU6050的采样率`：`4~1000Hz`
`获取MPU6050温度值`
`获取MPU6050陀螺仪原始值`
`获取MPU6050加速度原始值`

```
void MPU_Get_Angles(float * roll,float * pitch)
{
	short ax,ay,az;
	MPU_Get_Accelerometer(&ax,&ay,&az);
	pitch = -atanf(ax/sqrtf(ayay+az*az));//计算俯仰角
	*roll = atanf((float)ay/(float)az);//计算滚转角
}
```

MPU6050陀螺仪：如何获取加速度和角速度

ADC：分辨率与精度的区别，选择的ADC需要考虑什么因素

dma：外设到存储器/存储器到外设的具体用法，怎么开启dma传输数据

i2c：读写多个字节的时序，设备空闲时上拉电阻阻值大小

stm32里，串口接收哪几种模式，你比较常用什么模式，为什么呢。它是接受断如何判断数据接受结束的。

讲讲项目里mpu6050如何工作的，有哪些参数，有什么作用。

mpu6050有哪些寄存器

计算步数

> https://blog.csdn.net/qq_45595840/article/details/12619495

人体的矢状面是指前后方向将人体分为左右两部分的纵切面

MPU6050

低功耗

1. 使用内部 8MHz 时钟源。
2. 启用循环模式（在睡眠和唤醒之间切换）。
3. 禁用温度传感器以降低功耗
4. 不启用持续睡眠模式。
5. - **禁用陀螺仪**：降低功耗，适用于不需要检测旋转运动的应用。
   - **陀螺仪功能**：测量角速度和姿态变化。
   - **唤醒频率 20Hz**：MPU6050 每秒钟唤醒 20 次，以平衡功耗和数据采集的实时性。
   - 关闭FIFO和I2C主模式
   - **动态调整采样率**

MPU6050陀螺仪：如何获取加速度和角速度，中断还是轮询，温度值要获取吗

3. `获取MPU6050加速度原始值`

```
u8 MPU_Get_Accelerometer(short *ax,short *ay,short *az)
{
    u8 buf[6],res;
        res=MPU_Read_Len(MPU_ADDR,MPU_ACCEL_XOUTH_REG,6,buf);
        if(res==0)
        {
            *ax=((u16)buf[0]<<8)|buf[1];
            *ay=((u16)buf[2]<<8)|buf[3];
            *az=((u16)buf[4]<<8)|buf[5];
        }
    return res;;
}
```

4. `voidMPU_Get_Angles(float*roll,float*pitch)`

{

short ax,ay,az;

MPU_Get_Accelerometer(&ax,&ay,&az);// `获取MPU6050加速度原始值`

pitch = -atanf(ax/sqrtf(ayay+az*az));//计算俯仰角

*roll = atanf((float)ay/(float)az);//计算滚转角

}

5.

```
uint8_t MPU_isHorizontal(void)
{
    float roll,pitch;
    MPU_Get_Angles(&roll,&pitch);//获取俯仰角
    if(roll<=0.50 && roll>=-0.50 && pitch<=0.50 && pitch>=-0.50) //如果滚转角和俯仰角的绝对值都小于或等于 0.50（弧度），则认为传感器处于水平状态，返回 1
    {return 1;}
    return 0;
}
```

本项目mpu6050用来检测是否水平：

1.获取滚转角和俯仰角

2.如果这两个在正负0.5就算水平
3.通过加速度来计算滚转角和俯仰角：

> 读取三轴加速度就可以计算

ADC

12位ADC，精度4096

> 计算公式：`BatVoltage=dat*2*3.3/4096;`

EEPROM

1.如果需要频繁擦写小数据量（如配置参数、校准数据），EEPROM更合适。不需要擦写
2.FLASH写入时间长，EEPROM写入时间短
3.FLASH擦写次数少（1万次），EEPROM次数多（100万次）

	NAND Flash	NOR Flash
芯片容量	<32GBit	<1GBit
访问方式	顺序读写	随机读写
接口方式	任意I/O口	特定完整存储器接口
读写性能	读取快（顺序读） 写入快 擦除快（可按块擦除）	读取快（RAM方式） 写入慢 写入慢
使用寿命	百万次	十万次
价格	低廉	高昂

从实用的角度来看,和Flash和NAND闪存之间的主要区别在于接口。NOR Flash全随机访问内存映射和专用接口(如EPROM)地址和数据行。另一方面,NAND闪存没有地址专线。它是由通过8/16发送命令,地址和数据总线位宽(I / O接口)内部寄存器，这样就为许多主控提供了更灵活的配置方式。
NAND Flash更适合在各类需要大数据的设备中使用，如U盘、各种存储卡、MP3播放器等，而NOR Flash更适合用在高性能的工业产品中。

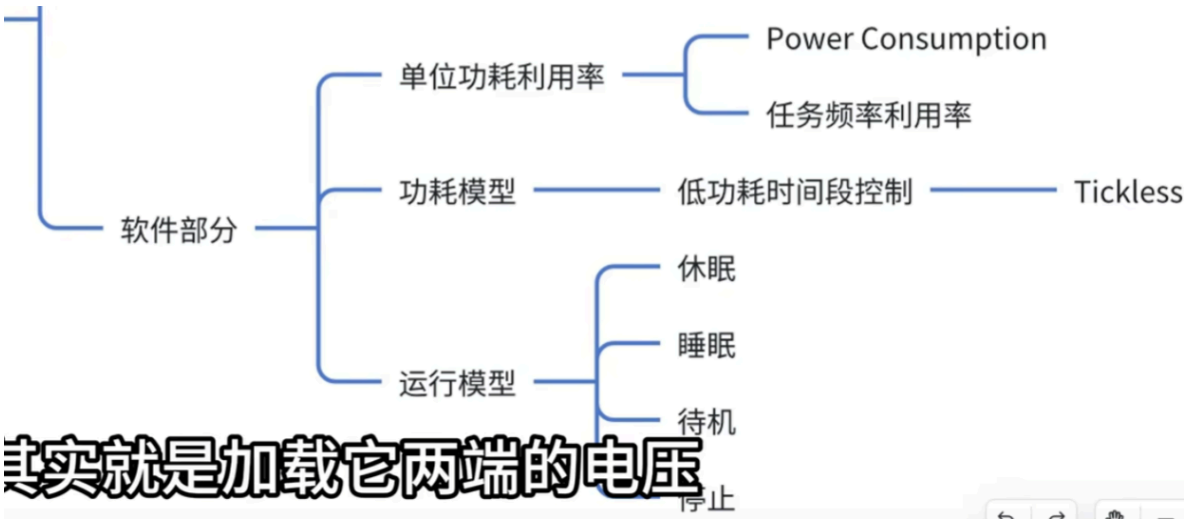
SRAM： 触发器存储，不需要定期刷新 速度快 功耗高 成本高 小容量
常用于计算机CPU高速缓存

DRAM： 电容存储，需要定期刷新　　速度较快 功耗稍低 成本低 容量大 适用于大容量存储需求

在SRAM里面，一个bit的数据，通常需要6个晶体管，所以SRAM的存储密度不高，同样的物理空间下，能存储的数据是有限的，不过也因为SRAM的电路简单，所以访问速度非常快。

- SRAM的存储单元是基于**触发器**实现的，通常由6个晶体管构成一个交叉耦合的反相器对。这种结构具有**双稳态特性**，即它可以稳定地保持两种状态（0或1），并且只要电源持续供电，数据就会一直保持在该状态。

  - **速度快的原因**：
    - **无需刷新**：SRAM不需要定期刷新，因此不会因为刷新操作而占用访问时间。
    - **简单直接的读写操作**：SRAM的读写操作非常简单直接。当访问一个存储单元时，控制信号通过访问晶体管直接读取或写入数据，不需要复杂的操作流程。
    - **低延迟**：由于存储单元的自保持特性，数据的读取和写入延迟非常低，能够快速响应CPU的指令。
  - **电路简单性**：
    - 尽管SRAM的存储单元由6个晶体管组成，但它的电路逻辑相对简单。每个存储单元的读写操作只需要通过控制信号激活访问晶体管即可，不需要复杂的刷新逻辑或额外的电路来维持数据状态。



在软件业务逻辑的利用率上，在事件驱动的情况下，尽可能通过使用较低的频率运行（有些操作系统也为我们提供了比较方便的低功耗措施，比如FreeRTOS中的tickeless为我们提供了对应的阻塞式进出低功耗的方法，具体Demo文档可以参考：

IAP升级

中断向量表的偏移量设置方法

> systemInit函数：初始化时钟，设置中断向量表
>
> ```
> #ifdef VECT_TAB_SRAM
> SCB->VTOR = SRAM_BASE | VECT_TAB_OFFSET;
> /* Vector Table Relocation in Internal SRAM. */
> #else
> SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET;
> /* Vector Table Relocation in Internal FLASH. */
> #endif
> ```
>
> SCB->VTOR=FLASH_BASE|0x10000;flashapp偏移0x100000

一页1kb=1024 字节下标从0-1023：

123的二进制0011 1111 1111：十个二进制位就可以表示所有的地址

比如页号是19　　19<<10=4C00,所以地址就是0x8004C00

w25q64

每块64KB,一共分成8mb/64kb=128块
每块分成64KB/4KB=16个扇区，每个扇区4kb,256b一页，分成16页

我一共需要48KB的空间，一共需要12个扇区

同理：4kb是0到4095，将4095换成二进制 1111 1111 1111，需要12个1进行表示

1. 睡眠时DeInit串口的IO口，设置为输入，修复休眠功耗很高的情况，现在休眠状态电流800多uA.

1. BootLoader和APP都加入长按KEY1关机功能.

手表的模式分为3个。第一个是正常的运行模式，手表正常运行；第二个是睡眠模式，MCU进入STOP模式，MPU6050仍在记步数；第三个是关机模式，TPS63020直接关闭使能，此时无3V3供电，只有Vbat有供电

最后还是用的RTC定时中断，然后定时检测当前手势状态，如果有抬腕动作则唤醒。

MPU6050不能直接使用DMP库，初始化后功耗很高，需要进行一些改动，才能让功耗下来，具体看工程代码。

运行模式70-80mA，待机模式1mA左右

RTC实时时钟

掉电还继续运行的特性，指主电源VDD断开的情况，为了RTC外设掉电继续运行，必须接上锂电池给STM32的RTC、备份发卡通过VBAT引脚供电。当主电源VDD有效时，由VDD给RTC外设供电；而当VDD掉电后，由VBAT给RTC外设供电。但无论由什么电源供电，RTC中的数据都保存在属于RTC的备份域中，若主电源VDD和VBAT都掉电，那么备份域中保存的所有数据将丢失。备份域除了RTC模块的寄存器，还有42个16位的寄存器可以在VDD掉电的情况下保存用户程序的数据，系统复位或电源复位时，这些数据也不会被复位。

时钟来源：

使用LSE作为时钟来源

指南针

海拔如何测量

心率检测

em7028:发射绿光。人体血液的流动会导致血管的收缩，从而导致光吸收率的变化简介导致光反射的变化。

光敏二极管会产生不同的电流，在em7028的16位ADC 作用下进行电流转换成数字信号。

自己写了一个局部峰值算法

led1进行心率测量，结果存储在`HRS_DATA0`

- HRS1 数据由两个 8 位寄存器组成：`HRS1_DATA0_H`（高 8 位）和`HRS1_DATA0_L`（低 8 位）。
- 函数将高 8 位和低 8 位数据合并为一个 16 位数据，并返回该值。

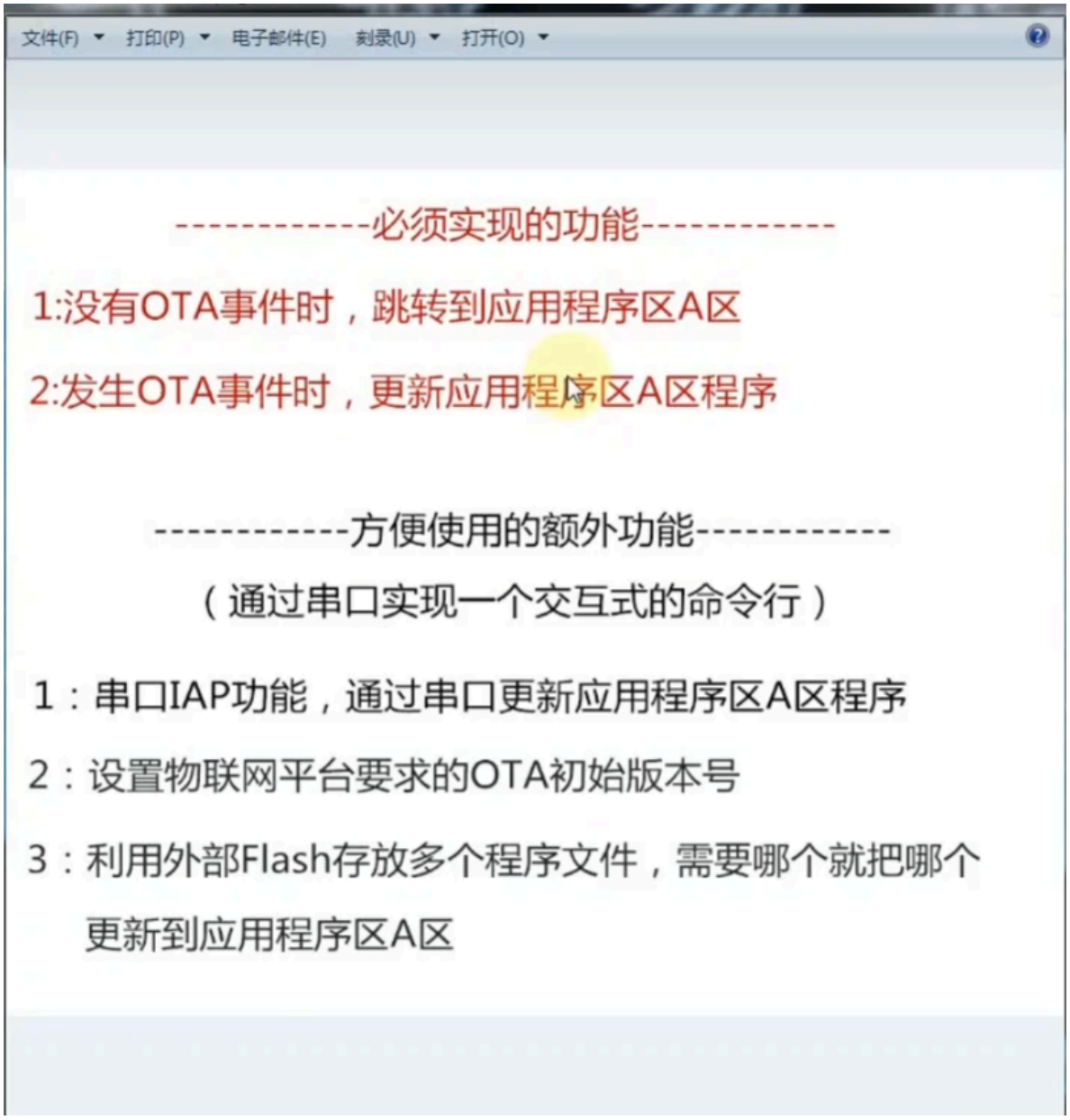
计算心率

1.设计三个队列：时间队列数据队列

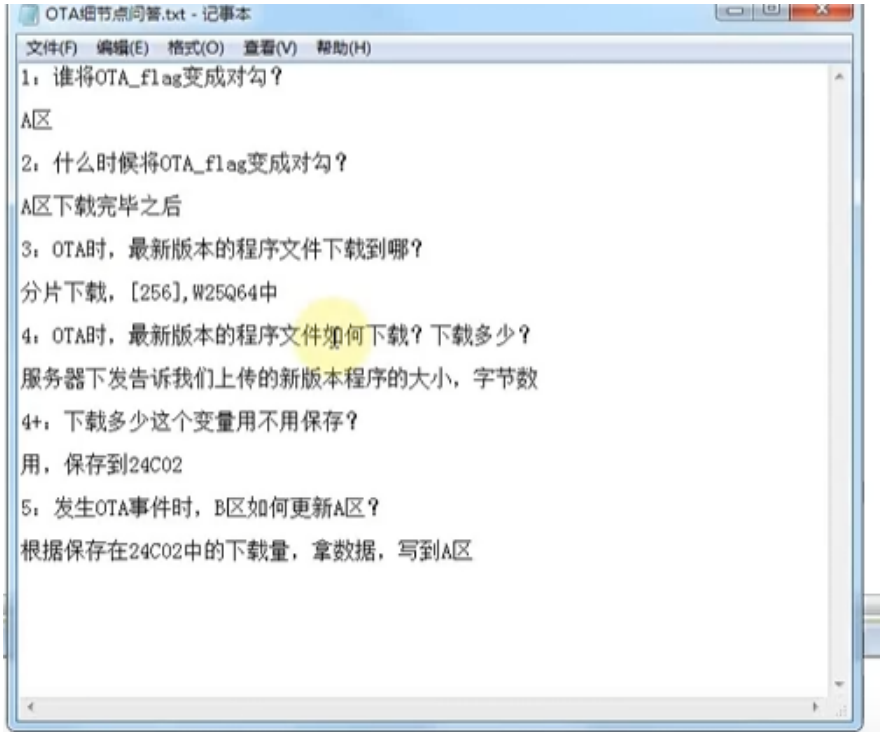
当前心率和时间入数据队列，队列中的第四个元素是不是大于前后三个，如果是则出现波峰。检测与上一个波峰时间差是不是425，如果是就存储上峰值时间，将

计算两个峰值之间的时间间隔，并将其转换为心率值（每分钟心跳次数），然后如果次数达到七次就进行滤波算法。

bootloader


文件(F) 打印(P) 电子邮件(E) 刻录(U) 打开(O)
----------必须实现的功能----------
1:没有OTA事件时，跳转到应用程序区A区
2:发生OTA事件时，更新应用程序区A区程序
----------方便使用的额外功能----------
（通过串口实现一个交互式的命令行）
1：串口IAP功能，通过串口更新应用程序区A区程序
2：设置物联网平台要求的OTA初始版本号
3：利用外部Flash存放多个程序文件，需要哪个就把哪个更新到应用程序区A区


#ifndef MAIN_H
#define MAIN_H
#include "stdint.h"
#define GD32_FLASH_SADDR 0x08000000 //FLASH起始地址
#define GD32_PAGE_SIZE 1024 //FLASH扇区大小
#define GD32_PAGE_NUM 64 //FLASH总扇区个数
#define GD32_B_PAGE_NUM 20 //B区扇区个数
#define GD32_A_PAGE_NUM GD32_PAGE_NUM - GD32_B_PAGE_NUM //A区扇区个数
#define GD32_A_START_PAGE GD32_B_PAGE_NUM //A区起始扇区编号
#define GD32_A_SADDR GD32_FLASH_SADDR + GD32_A_START_PAGE * GD32_PAGE_SIZE //FA区起始地址
#endif
AB分区规划总结.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
C8T6:64KB 一个扇区1KB 0~63
B区：20K 0~19
A区：44K 20~63



断电重连

1.三个参数

- _write_part_addr：记录当前写入 Flash 的偏移地址（相对地址）。
- _storage_data_size：记录当前暂存的固件分包大小。
- _is_start_write：标志位，表示是否已经开始写入固件。

2.下载进度记录

- **写入固件时更新进度**：在每次写入固件分包后，更新 `_write_part_addr`，记录当前写入的偏移地址。
- **断电恢复时读取进度**：重新上电后，bootloader 会读取 `_write_part_addr`，从记录的偏移地址继续下载。

3.

- _fpk_head.pkg_size：固件包的总大小（单位：字节）。
- _write_part_addr：当前已写入的偏移地址（单位：字节）。
- FPK_LEAST_HANDLE_BYTE：固件包处理的最小单位大小（默认 4096 字节）。

固件包头（FPK_HEAD）是一个结构体，包含了固件包的元信息，例如固件版本、CRC校验值、固件大小等。pkg_size是其中的一个字段，表示固件包的总大小。

无需deinit

更新完成之后不直接跳转app而是再次进入bootloader

```
LV_ATTRIBUTE_TICK_INCvoidlv_tick_inc(uint32_ttick_period);
```