**“中国软件杯”双创大赛**

**系**

**统**

**设**

**计**

**说**

**明**

**书**

|  |  |
| --- | --- |
| **作品名称：** | **物联网婴幼儿睡眠监测脚环** |
| **作品类别：** | **物联网类** |
| **作　　者：** | **石凯悦、霍嘉琳、王辣** |
| **指导老师：** | **陈刚** |
| **单　　位：** | **湖南农业大学** |

**二零一七年 十二月 一日**

****

物联网婴幼儿睡眠监测脚环

目录

[1.前言 4](#_Toc488173848)

[1.1作品开发背景 4](#_Toc488173849)

[1.2国内外现状 5](#_Toc488173850)

[1.3作品开发目的 6](#_Toc488173851)

[1.4作品简介 7](#_Toc488173852)

[2.硬件设计 8](#_Toc488173853)

[2.1脚环硬件结构图 8](#_Toc488173854)

[2.2BLE硬件选型 9](#_Toc488173855)

[2.3运动传感器硬件选型 11](#_Toc488173856)

[2.4心率检测程序实现 12](#_Toc488173857)

[2.5姿态采集 13](#_Toc488173858)

[3.软件平台设计 16](#_Toc488173859)

[3.1BLE蓝牙4.0 16](#_Toc488173860)

[3.2BLE蓝牙4.0嵌入式软件实现 25](#_Toc488173861)

[3.3MPU6050姿态采集程序实现 30](#_Toc488173862)

[3.4MAX30100心率采集程序 33](#_Toc488173863)

[3.5MEMS数字麦克风实现 37](#_Toc488173864)

[3.6安卓软件实现 39](#_Toc488173865)

[4.Android应用及Web服务平台 54](#_Toc488173866)

[4.1功能说明 54](#_Toc488173867)

[4.2软件设计实现 58](#_Toc488173868)

[5.作品运行说明 71](#_Toc488173869)

[5.1嵌入式系统软件的烧录 71](#_Toc488173870)

[5.2安卓平台软件的运行 72](#_Toc488173871)

[6.总结 76](#_Toc488173872)

物联网婴幼儿睡眠监测脚环

# 1.前言

## 1.1作品开发背景

2015年末，我国全面放开了二胎政策，规定一对夫妻可生育两个小孩。受政策利好，我国迎来第二轮婴儿潮。在第二轮婴儿潮席卷下，目前我国每年新增婴幼儿人数达1687万，同时全国还有106.9万对单独夫妇申请再生育，这必然使我国新生婴幼儿人数再达新的人数高峰。随着新生婴儿人数的增加，专业婴幼儿产业产品的需求势必也随之提高，这也就促进了婴幼儿产品市场的前景。

伴随着“单独二孩”政策的开放，家长们对新生婴儿的生活质量越来越重视，据调查表明，我国婴幼儿在家庭和社会的地位明显提升，婴幼儿的消费开支占家庭总开支的40%以上。而婴儿健康状况成为家长最关心的问题，因此关于监控婴幼儿健康的产品需求量也越来越大。就目前来看，我国关于婴儿健康监控产品设计仍处于尚未成熟的初级阶段，国内市场上仍未普遍出现关于婴儿健康监控的产品。婴儿产品开发的滞待性也使得如今社会上时常出现婴幼儿健康问题。

综上所述，婴幼儿期的孩子，由于不具备语言沟通能力，没有形成人类意识，经常自发地做出一些危及生命安全的行为，潜在地危害到婴幼儿的健康。然而如今的家长经常是身兼数职，并不能做到实时地监控婴幼儿的健康，这给如今许多新时代的年轻家长带来了巨大的困扰。基于以上背景，我们针对婴儿特殊的生理特点，对婴儿健康监测产品进行智能化、信息化的创新设计，研发了物联网婴幼儿睡眠监测脚环。这款产品使家长们能轻松获得一些关于婴幼儿的生理资讯，如体温，睡姿，心率，搭配手机app使用，就能将即时地将婴幼儿资讯转换成纪录图表，让新手家长们也能随时掌握婴幼儿的各种身体数值的变化，帮助新手家长们更准确地了解婴幼儿目前的生理状态，能够更好的照顾小宝宝。

## 1.2国内外现状

本作品开发团队的调研与检索，得知目前国内外有关针对婴儿睡眠、心率方面的护婴眠助健产品主要有以下几类：

（1）可穿戴智能连身衣：可穿戴连身衣通过芯片和传感器的连接，利用手机、物联网和云计算技术开发出这款智能婴儿连体衣—Mimo，通过对传感器传送的数据进行分析，能跟踪婴儿的睡眠时间，最终预测出宝宝何时会睡觉以及宝宝何时会醒过来，也可追踪到宝宝是否睡眠正常。但这项产品售价高达199美元，而且产品单一，不方便进行清洗。所以这类产品不适合国内收入不高但却具有极大潜在市场的消费群体，消费者对其的体验感也不会太高。



图1 mimo连身衣

（2）可穿戴智能袜：可穿戴智能袜核心硬件部分也是由MCU和传感器组成，该款可穿戴智能袜是由美国婴儿健康护理公司Owlet设计开发的婴儿智能袜，它包括四个传感器的脉冲血氧计，能通过红光和红外光线来测量宝宝心率血氧和温度，且设有加速计进行监测。用户可通过智能手机或电脑了解宝宝的皮肤温度、心率、血液含氧量以及睡眠质量。但该产品的官网售价高达249.99美元，相比智能连身衣价格更高，对于国内的消费对象来说存在很大的市场障碍，且在炎热的夏天不适合穿袜，宝宝会觉得不舒服，同时在美观上也有所欠缺。



图2 Owlet智能袜

中国产业调研网发布的2017年版中国可穿戴设备市场专题研究分析与发展趋势预测报告认为，2015年可穿戴设备市场预计将出现大幅度增长，达到24.5亿元，相比于2013年的6.1亿元，将环比增长301.64%。中国企业也在争抢先机，如，百度推出手环抢占市场，市面上已有针对老人、盲人等特殊人群的可穿戴产品。各厂商虽然初步切入不同的细分应用领域，但实用的、革命性的产品还比较缺乏，但这类产品价格不适合大部分群体。所以在国内开发出一种针对婴儿的性价比高、体验感强、价格适中的产品将会迎合潜在的国内大市场，是一个极大的机遇。

## 1.3作品开发目的

通过对作品开发背景的分析及国内外现状的分析，本项目提出了一种针对婴儿（一周岁以内）的睡姿调整和体温、心率监控的方案：“可穿戴脚环-无线传输系统-智能手机平台”的新型护宝睡眠监测服务系统。通过中国可穿戴设备市场分析以及对婴儿健康睡眠的综合分析、体温和心率的相关分析，结合在技术上将各类传感、识别、连接和云服务等技术综合嵌入到可穿戴设备形成的“可穿戴脚环-无线蓝牙传输系统-智能手机APP”系统，从功能，造型，功能界面三个方面，创新性地开创出这款新型的智能婴儿监控脚环，对婴儿睡姿，心率，体温三个基本特征进行分析，实现家长对婴儿的全方面监控，同时降低家长们在照顾婴儿方面的工作量，让他们从繁重的育儿工作中解脱出来，在宝宝睡觉的时候，不用再让家长们担心。

## 1.4作品简介

本系统主要分为数据的采集、Android手机软件界面设计和云端服务器。其中数据的无线采集是通过传感器模块采集：运动检查传感器MPU6050，心率和血氧测量传感器MAX30100传感器和姿态采集传感器MPU6050。可穿戴智能脚环通过低功耗可穿戴蓝牙4.0方案与APP进行通信。在使用智能手环时，无线传输将婴儿的睡姿状态、心率、血氧以及体温相关的数据实时传送至手机App上，云端数据库实时对数据进行智能分析，分析结果反馈给用户，发现异常及时预警，能实时对婴幼儿的健康状况的监测。极大地方便了用户对宝宝睡眠的监控，提高用户对宝宝健康睡眠重要性的意识，进而提升宝宝的睡眠质量。

物联网婴幼儿睡眠监测脚环



**创新点**如下：

1、物联网婴幼儿睡眠监测脚环主控芯片采用Nordic公司2015年最新推出低功耗蓝牙4.0芯片nRF52832，集成72M带浮点计算功能的ARM CortexM4F内核，实时处理传感技术。

2、MPU6050姿态传感器，能实时获取分析宝宝睡眠姿态，有助于婴儿睡觉时的正确睡眠姿态；Max30100心率传感器，能实时分析宝宝的睡眠状态，当宝宝有紧急情况，或者宝宝心率异常（例如宝宝噩梦），宝宝不会从噩梦中惊醒，妈妈能提前给予宝宝呵护。

3、集成双温度传感器，父母再也不用担心冬天宝宝踢走了被子，物联网婴幼儿睡眠监测脚环能分析宝宝体温，实现温度异常预警。

4、ADMP MEMS数字麦克风，能实时采集宝宝的声音，当宝宝醒来时能第一时间通知大人“宝宝醒了”。

# 2.硬件设计

基于物联网的可穿戴式婴幼儿睡眠监测系统硬件设计部分为基于可穿戴技术开发的智能脚环，智能脚环包含MAX30100和MPU6050两个传感器模块，安卓平台数据分析也是建立在传感器监测的基础上，硬件平台的开发是作品成功与否的一个关键。

## 2.1脚环硬件结构图

由于设计所需，脚环在保证功能实现的同时，必须有很好的便携性，则应该尽量减少设备的体积和重量，元器件应该具有体型微小，易于佩戴，耗电量低等特点，同时考虑到嵌入式创新实验室的设备情况，元件应该具有一定的可手工焊接性。本作品采用nRF52832蓝牙模块与MPU6050芯片做成数据采集平台，元件选用0604，0402的贴片封装，为了在提高工作时间进一步减小作品体积，作品选用了纽扣电池供电，采用LDO方案进行降压，作品手工焊接，电路图如下所示：



图4脚环主控芯片nRF52832电路图

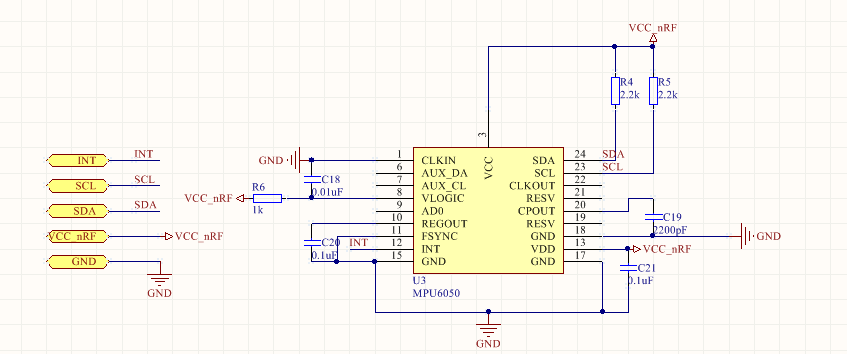


图5MPU6050电路图

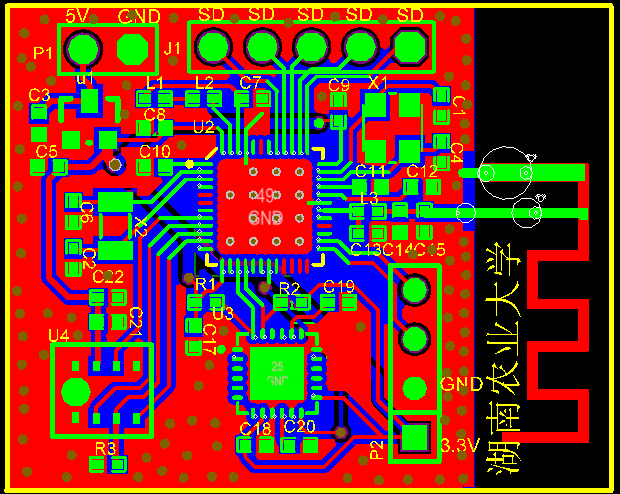


图6PCB布板图

作品可穿戴节点的实物图如下：

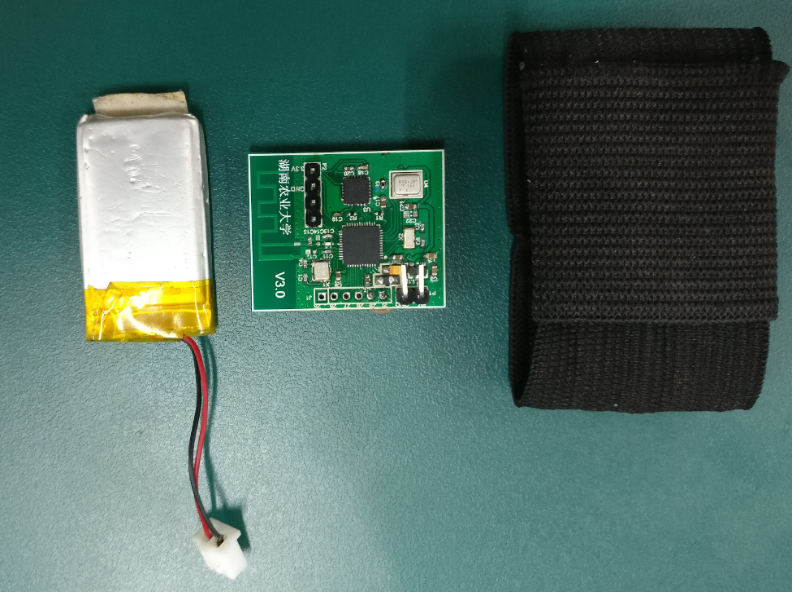


图7自主开发的可穿戴脚环控制中心实物图

## 2.2BLE硬件选型

蓝牙4.0协议颁布后，国内外学者对蓝牙4.0的在各行业应用进行了深入的研究，取得了一定的研究成果，但是主要基于TI公司的CC2540方案。CC2540最大256-KB的RAM，8-KB的SRAM以及高性能和低功耗8051微控制器。该芯片还集成了许多外围，例如，12位模/数转换器、通用的计时器（1个16位，2个8位）、AES安全处理器、每个CC2540包含一个全球唯一的地址、强大的6通道DMA、21个通用的I/O口、高速的USB接口等。在功耗方面非常低，工作时损耗最小为19.6mA，休闲模式电流损耗为0.9mA，唤醒系统可以釆用外部中断和RTC两种方法，而且从休眠模式到工作模式需要的唤醒时间特别短，这个特点对要求电池寿命非常长以及时延要求非常高的场合非常适应，此外，该芯片还具有强大的抗干扰能力和优良的无线接收灵敏度，工作电压为2.0V-3.6V，以及功能齐全且价格便宜的开发工具。

nRF52832是Nordic推出一款为超低功耗无线应用（ULPwirelesssapplications）打造的多协议单芯片解决方案。nRF52832与其它低功耗蓝牙芯片对比如表1所示：

表1BLE芯片对比表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | nRF52832 | CC2540 | CC2541 | CSR1000 |
| MCU | ARM-M432位 | 80518位 | 80518位 | 自有16位 |
| ROM | 256K | 128/256K | 128/256K | 64K |
| RAM | 16KB | 8KB | 8KB | 64KB |
| I/O数量 | 31（自由映射） | 21 | 23 | 15 |
| 电压 | 1.8~3.6V | 2.0~3.6V | 2.0~3.6V | 2.0-4.2V |
| TX(0dbm) | 10.5mA8.1mA | 27mA | 18.2mA | 16mA |
| RX | 13mA9.5mA | 19.6mA | 17.9mA | 16mA |
| 外接晶振 | 不需要 | 需要 | 需要 | 需要 |
| PCB | 双层板 | 四层板 | 四层板 | 四层板 |

相比其功耗，蓝牙4.0方案具有以下优点：

1）Nordic一流的无线传送器，一个ARMCortexM4核以及256KB的flash+16KB的RAM。nRF52832支持Bluetooth（R）lowenergy和专用的2.4GHz协议栈，nRF52832工作期间的射频峰值电流低于5mA，因此具有更快的计算速度，更低的功耗。

2）nRF52832所有的数字IO引脚都是可以自由定义和映射的，如：UART\SPI等功能引脚可以自由映射到各IO；大大方便了硬件设计，PCB使用普通双层板就可以，甚至可以使用单面板。

3）nRF52832蓝牙协议和应用代码的存储空间相互分开，互不干扰，客户只需关注应用代码开发，不需花大量的时间去了解蓝牙协议是如何运行的，只需配置API参数就可运行起来。nRF52832的蓝牙协议和应用代码是相互独立的，且蓝牙协议已通过了认证，此蓝牙协议认证费用可以省去，客户只需做应用代码认证就行，产品如有更新也只需做应用代码认证。

4）nRF52832内部有32.768KHz晶振，不需要外挂。nRF52832空中升级的时候只需下载应用代码然后更新，即便在升级中出现中断，也不会影响产品的正常使用，更安全、可靠。

因此，作品选用nRF52832开发可穿戴的传感器网络，相比目前流行的设计方案，具有先进性。

节点的整体结构如图8所示。

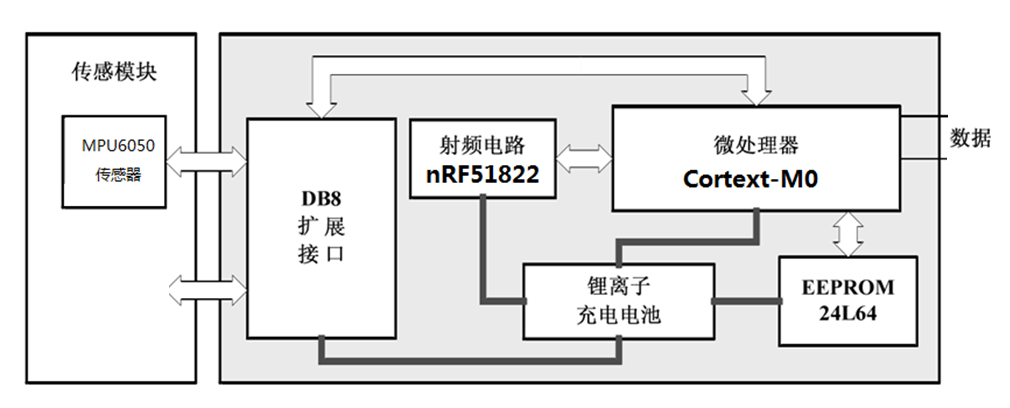


图8节点结构图

硬件电路如图所有，由nRF52832蓝牙模块和运动检测芯片MPU6050构成，采用200ma锂电池供电，整个系统（含超声波模块）工作电流不超过15ma。

## 2.3运动传感器硬件选型

常用的运动检查传感器有ADIS16355、L3G4200D+ADXL345、MPU6050。从器件分辨率、输出噪声、工作能耗、尺寸及价位这些方面考虑，MPU6050为最合适的选择。MPU6050是InvenSense公司生产的全球首列整合性6轴运动处理组件，相比较于多组件方案，免除了组合陀螺仪与加速度器轴间时差的问题，从而可保证两类传感器数据更新的同步，其另一个很突出的特性是，该器件基于芯片级的集成了陀螺仪与加速度计，使这两类器件使用同一时钟电路，从而保证数据更新的同步性，该器件的这种设计可有效提高器件性能，使更方便应用。虽然MPU6050的陀螺仪零位偏移相对于其他的陀螺仪较大，但零位偏移所导致的误差不是简单地比较出厂参数就可以避免的，有效遏制零位偏移误差积累的方法是通过加速度计输出数据进行补偿，故此系统对陀螺仪零位偏移参数无过多严格要求，只要有相应的补偿措施都是可行方案。MPU6050整合了3轴陀螺仪、3轴加速度计的同时，由第二个I2C端口连接其他惯性传感器，内置的数字运动处理（DMP：DigitalMotionProcessor）硬件加速引擎，可对数据进行实时处理。

MPU6050的内部结构图如图9所示：

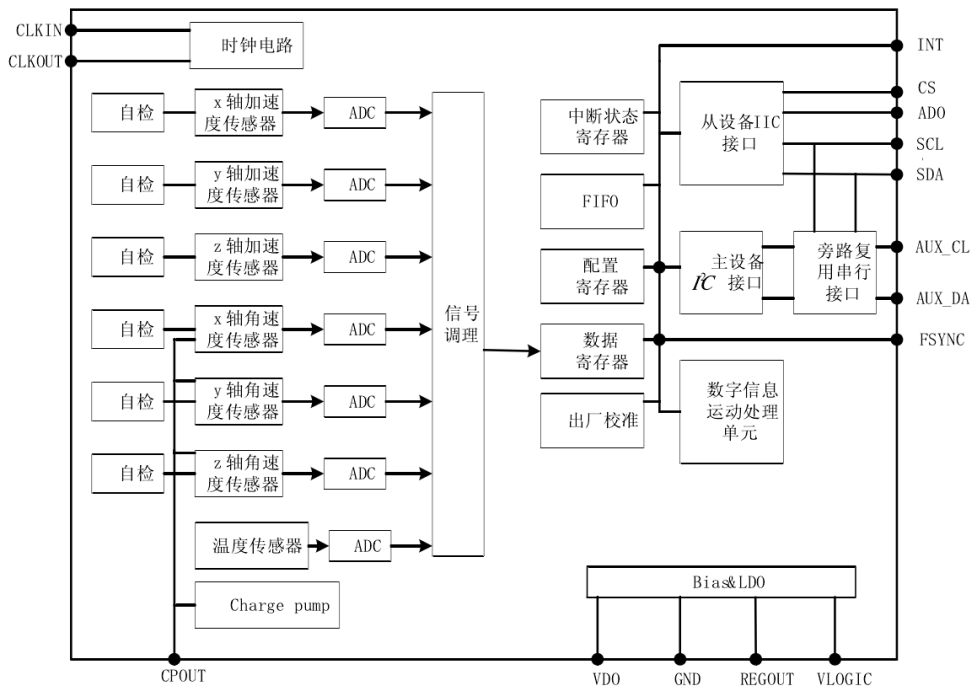


图9MPU6050结构图

由图9可知，该类器件内部两类传感器采用统一的时钟，传感器测得的原始数据相应的处理后以数字的形式由I2C接口输出。陀螺仪可通过人为配置设置量程范围为±250o/s。3轴加速度计的量程范围也可通过程控选择为±2g，。内置可程控的中断系统可支持快速下降中断、零动作感应中断、摇动感应事件及触击感应事件等。该器件内建的频率产生器在所有温度范围（-40°~+85°）仅有±1%的频率变化。MPU6050的包装尺寸为4mm\*4mm\*0.9mm，其空间占用率较小，适合可穿戴节点设计。

## 2.4心率检测程序实现

心率测量选择了一款集监测血氧饱和度和脉搏波速率于一体的传感器MAX30100。它包含两个发光二极管，一个光检测器。拥有光学优化和低噪声的模拟信号处理能力。MAX30100采用IIC通信协议，通过配置其中断寄存器、模式寄存器来控制其工作状态。设置MAX30100工作在SPO2模式，AD转换精度为14位，对应的最大采样速度为400sample/s，设置红外和红光的LED典型电流为14.2mA。MAX30100工作在SPO2状态时，一个样本由4个字节组成，包含红光检测值与红外光检测值，从FIFO寄存器中读取。人体血流反射红外光。人体食指遍布毛细血管，当血液流过时，会使红外光反射值增大。记录反射的红外光值随时间推移的变化，即可检测脉搏波的变化情况。通过检测脉搏波中极其陡峭的上升沿之间的时间间隔，即可推算用户心率大小。



图10Max30100心率传感器模块图

## 2.5姿态采集

传感器选用MPU6050,MPU6050集成了轴陀螺仪和轴加速度计以及一个可扩展的数字运动处理器，可用接口连接一个第三方的数字传感器（比如磁力计），扩展之后可以通过其接口输出九轴的信号。三轴加速度计和三轴陀螺仪分别集成了三个位的速度，并且将测量的模拟量转化为可输出的数字量。陀螺仪可测范围为±250°/s、±500°/s、±1000/s°和±2000°/s，加速度计可测范围为±2g°、±4g°、±8g°和±16g，由三个独立的陀螺仪组成，用于检测旋转的x轴、y轴、z轴。当传感器绕任何一个轴旋转，通过电容截止产生震动引起科里奥利效应，将得到的信号进行放大、调解和过滤，使产生的电压正比于角速度。然后每个轴的角速度通过芯片上的独立的位，将电压转换成数字信号。采样率是可编程的，也可以使用低通滤波器来截止一定范围的频率。MPU6050的三轴加速度计每个轴采用独立分开检测。每个轴上加速度产生相应的位移检测和电容式传感器检测到的位移差异。这样的架构降低了加速度计的加工敏感性以及漂移情况。当该装置被放置在平坦的表面上时，测量得出的是的x轴、y轴和的z轴是平行不旋转的。

（1）5.4 DMP算法原理

通过陀螺仪数据输出寄存器和加速度传感器数据输出寄存器获取的是MPU6050加速度传感器和陀螺仪的原始数据，而实验需要的是姿态数据，即欧拉角：航向角（Yaw）、横滚角（Roll）和俯仰角（Pitch）。

要得到欧拉角数据，就得利用我们的原始数据，进行姿态融合解算，这个比较复杂，知识点比较多，而MPU6050自带了数字运动处理器，即DMP，并且，InvenSense提供了一个MPU6050的嵌入式运动驱动库，结合MPU6050的DMP，可以将我们的原始数据，直接转换成四元数输出，而得到四元数之后，就可以很方便的计算出欧拉角，从而得到Yaw、Roll和Pitch。

使用MPU6050的DMP输出的四元数是q30格式的，也就是浮点数放大了2的30次方倍。在换算成欧拉角之前，必须先将其转换为浮点数，也就是除以2的30次方，然后再进行计算，计算公式如下：

其中quat[0]~quat[3]是MPU6050的DMP解算后的四元数，为q30格式，所以要除以一个2的30次方，其中q30是一个常量为1073741824，即2的30次方，然后带入公式，计算出欧拉角。上述计算公式的57.3是弧度转换为角度，即180/π，这样得到的结果就是以度(°)为单位的。

|  |
| --- |
| q0=quat[0]/q30;//q30格式转换为浮点数  q1=quat[1]/q30;  q2=quat[2]/q30;  q3=quat[3]/q30;  //计算得到俯仰角/横滚角/航向角  Pitch=asin(-2\*q1\*q3+2\*q0\*q2)\*57.3;//俯仰角  Roll=atan2(2\*q2\*q3+2\*q0\*q1,-2\*q1\*q1-2\*q2\*q2+1)\*57.3; //横滚角  Yaw=atan2(2\*(q1\*q2+q0\*q3),q0\*q0+q1\*q1-q2\*q2-q3\*q3)\*57.3; //航向角 |

（2）AHRS

AHRS称为航姿参考系统，包括多个轴向传感器，能够为飞行器提供航向，横滚和侧翻信息，这类系统用来为飞行器提供准确可靠的姿态与航行信息。

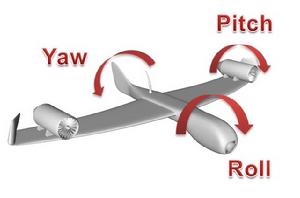


图11航姿参考系统

航姿参考系统包括基于MEMS的陀螺仪，加速度计和磁强计。航姿参考系统与惯性测量单元IMU的区别在于，航姿参考系统（AHRS）包含了嵌入式的姿态数据解算单元与航向信息，惯性测量单元（IMU）仅仅提供传感器数据，并不具有提供准确可靠的姿态数据。目前常用的航姿参考系统（AHRS）内部采用的多传感器数据融合进行的航姿解算单元为卡尔曼滤波器。

航姿参考系统（AHRS）具有两个主要特点：（1）高精度360度全方位位置姿态输出，但采用欧拉角的会具有万向锁，不能全向转动。（2）高效的数据融合算法快速动态响应与长时间稳定性（无漂移，无积累误差）相结合。航姿参考系统（AHRS）输出模式为三维全姿态数据（四元数/欧拉角/旋转矩阵）。

通过姿态结算，nRF52832通过DMP实现MPU6050姿态解算，通过脚环，获取宝宝实时姿态。

2.6蓝牙4.0

蓝牙4.0是[蓝牙3.0](http://baike.baidu.com/edit/%E8%93%9D%E7%89%994.0/8440996?dl=1)+HS规范的补充，专门面向对成本和功耗都有较高要求的无线方案，可广泛用于卫生保健、[体育健身](http://baike.baidu.com/edit/%E8%93%9D%E7%89%994.0/8440996?dl=1)、家庭娱乐、安全保障等诸多领域。它支持两种部署方式：双模式和单模式。双模式中，低功耗[蓝牙功能](http://baike.baidu.com/edit/%E8%93%9D%E7%89%994.0/8440996?dl=1)集成在现有的经典蓝牙控制器中，或在现有经典蓝牙技术(2.1+EDR/3.0+HS)芯片上增加低功耗堆栈，整体架构基本不变，因此成本增加有限。单模式面向[高度](http://baike.baidu.com/edit/%E8%93%9D%E7%89%994.0/8440996?dl=1)集成、[紧凑](http://baike.baidu.com/edit/%E8%93%9D%E7%89%994.0/8440996?dl=1)的设备，使用一个轻量级连接层(LinkLayer)提供超低功耗的待机模式操作、简单设备恢复和可靠的点对多点数据传输，还能让联网传感器在蓝牙传输中安排好低功耗蓝牙流量的次序，同时还有高级节能和安全加密连接。

蓝牙4.0是一个全新的开放式技术标准，这一技术未来应用前景非常广阔，目前国外已经对低功耗蓝牙应用开展研究，国内针对低功耗蓝牙应用研究刚刚起步，研究文献较少，因此有重要的应用前景。

nRF52832是功能强大、高灵活性的多协议SoC，非常适用于Bluetooth®低功耗和2.4GHz超低功耗无线应用。nRF52832基于配备256kBflash+16kBRAM的32位ARM®CortexM0CPU而构建。嵌入式2.4GHz收发器支持蓝牙低功耗及2.4GHz操作，其中2.4GHz模式与NordicSemiconductor的nRF24L系列产品无线兼容。nRF52832还具备丰富的模拟和数字周边产品，可以在无需CPU参与的情况下通过可编程周边产品互联(PPI)系统进行互动。灵活的31引脚GPIO映射方案可使I/O(例如串行接口、PWM和正弦解调器)根据PCB需求指示映射到任何设备引脚。这可实现完全的设计灵活性及引脚位置和功能。nRF52832支持S110蓝牙低功耗协议堆栈及2.4GHz协议堆栈(包括Gazell)，这两种协议堆栈在nRF518软件开发套件中均免费提供。nRF52832需要单独供电，如果供电范围在1.8-3.6V之间，用户可选择使用芯片上的线性整流器，如果供电范围在2.1-3.6V之间，可以选择直流1.8V模式和芯片上的DCDC变压器。DC-DC变压器的使用可在工作期间动态控制，并使nRF52832工作期间的射频峰值电流低于10mA。

# 3.软件平台设计

## 3.1BLE蓝牙4.0

3.1.1BLE简介：蓝牙技术标准，是以公元十世纪统一了丹麦和挪威的丹麦国王HaraldBlaatand（Bluetooth）II而命名的。它是由易利信（Ericsson）所研发的一种短距离无线传输介面，并在1998年联合Nokia、IBM、Toshiba、Intel五家厂商成立BluetoothSIG蓝牙技术联盟（BluetoothSpecialInterestGroup）。它寓意着实现通讯与计算机工业的无缝连接。事实上，它很快从最初的电缆替代延伸为面向个人无线网（WPAN）的应用标准。

蓝牙无线技术是使用范围最广泛的全球短距离无线标准之一，自从1998年蓝牙技术标准被制定以来，蓝牙技术在全球范围内得到了迅猛发展，蓝牙技术联盟（SIG）—直致力于蓝牙技术的研究与发展，蓝牙兴趣小组的公司成员也迅猛胀大，已经有15000多个公司成员加入，期间先后颁布蓝牙1.0，1.1，1.2，V2.0+EDR，V2.1+EDR，3.0+HS等多个版本，并于2010年7月发布蓝牙4.0版本核心，规范蓝牙正式迈入4.0时代。

蓝牙低能耗（BLE）技术是低成本、短距离、可互操作的鲁棒性无线技术，工作在免许可的2.4GHzISM射频频段。它从一开始就设计为超低功耗（ULP）无线技术。它利用许多智能手段最大限度地降低功耗。蓝牙4.0规范是个将传统蓝牙、高速蓝牙和蓝牙低功耗（Bluetoothlowenergy）技术整合为一体的规范。该规范中最吸引人的就是蓝牙低功耗技术（BLE）。BLE拥有极低的运行和待机功耗，BLE设备使用一粒纽扣电池甚至可连续工作数年之久；同时还拥有低成本、向下兼容、跨厂商、使用方便特点。

低功耗蓝牙技术的主要优势是低功耗，即1枚纽扣般大小的电池就可以运行10年之久。因此该技术主要是实现两个设备之间的简单数据传输，同时还满足各种领域不同种类的要求，如消费型电子设备、工业控制、农业自行化、汽车自动化和医用设备控制等，在不就的将来，市场将会非常庞大。

3.1.2蓝牙低功耗技术及其与经典蓝牙的比较

蓝牙低耗能技术是蓝牙4.0版本最新提出的规范标准，完整的蓝牙低功耗技术包含三个部分：控制器部分，主机部分与应用Profiles规范部分。蓝牙低耗能技术主要有以下三个特点：待机时间长，连接速度快，发射和接收的高峰功率低。这些决定了它的超低功耗性能，使用标准纽扣电池足以操作数年。另外蓝牙低功耗技术还具有低成本、多种设备之间的互联互操作性能。下面对其中一些具体的特点进行详细的比较说明：

1）待机功耗方面

传统的蓝牙设备待机耗电量大，这与传统蓝牙技术动辄采用16~32个频道进行广播有着密切的关系，而低功耗蓝牙仅采用了3个广播通道，且每次广播时射频的开启时间也由传统的22.5ms减少到0.6~1.2ms。因此，蓝牙低耗能技术消耗的功率是传统蓝牙技术的十分之一至二十分之一。这两个协议规范上的改变显然大大降低了因为广播数据导致的待机功耗；此外低功耗蓝牙设计采用深度睡眠状态来替换传统蓝牙的空闲状态，在深度睡眠状态下，主机长时间处于超低的负载循环（DutyCycle）状态，只在需要运作时由控制器来启动，因为主机较控制器消耗更多的能源，因此这样的设计也节省了更多的能源；在深度睡眠状态下，协议也针对此通讯模式进行了优化，数据发送间隔时间也有了些不同增加，范围0.5~4s，通常传感器类应用程序发送的数据量较小，而且所有连接均采用先进的嗅探性次额定（Sniff-Subrating）功能模式，因此此时的射频能耗几乎可以忽略不计，综合以上因素，蓝牙低功耗的待机功耗较传统蓝牙有大幅度减少。

2）连接过程方面

一般蓝牙设备和主机设备的连接步骤通常分为以下五步：第一，通过扫描，试图发现新设备；第二，确认发现的设备没有处于使用状态，也没有处于锁定状况；第三，发送IP地址；第四，收到并解读待配对设备发送过来的数据；第五，建立并保存连接。蓝牙低功耗协议还对拓扑结构进行了优化，通过在每个从设备及每个数据包上使用32位的存取地址，能够让数十亿个设备能被同时连接。此技术不但将传统蓝牙的一对一连接进行了优化，同时也利用星状拓扑来完成一对多点的连接。在连接和断线切换迅速的应用场景下，数据能够在网状拓扑之间移动，但不至于为了维持此网络而显得过于复杂，这也有效减轻了连接复杂性，减少了连接建立时间。

3）峰值功率方面

低功耗蓝牙技术对数据包长度进行了更加严格的定义，支持超短（8~27Byte）数据包；比起传统蓝牙技术，它用更“宽松”的射频参数，这两种技术都使用高斯频移键控（GFSK）调制，然而'蓝牙低耗能技术的调制系数为0.5，而传统蓝牙技术的调制系数为0.35，即增加了GSFK调制索引，调制系数0.5接近高斯最小频移键控（GMSK调制）方案，降低了发射时需要的电源。较低的调制系数有两个有益的副作用，这就是增加作用距离，增强可靠性。这两个措施最大限度地减少了数据收发的复杂性。传统的蓝牙技术的数据包较长。在传送这些较长的数据包时，射频电路必须保持在较高功率的状态，持续时间较长，因而芯片会发热。这会改变材料的物理特性，并且会改变传输频率（打断链接），除非射频电路不断重新校准。重新校准会消耗很多功率（需要用闭环的方法，因而射频电路更加复杂，提高了设备的价钱）。与此相反，蓝牙低耗能技术使用的数据包很短，因而芯片的温度不会很高。于是，蓝牙低耗能收发器不需要耗费能量的校准，也不需要闭环架构。此外低功耗蓝牙还通过增加调变指数，并釆用24位的CRC（循环冗余检查）确保封包在受干扰时具有更大的稳定度，低功耗蓝牙的射程增加至100m以上，以上措施结合蓝牙传统的跳频原理，有效降低了峰值功率。

4）协议栈方面

经典的BT为支持许许多多的profile，制定了很多种协议，如RFCOMM，BNEP，AVCTP，AVDTP，SDP等等，所以所传送的packet也有很多种，其中的规范也是很多的，很难全面了解清楚；而BLE的设计则简单了许多，它只使用一种协议即属性协议ATT，在包的类型与结构上也进行简化，只有了一种packetstructure以及二个packetformats，分别为Advertisingpackets与Datapackets，实现原始数据的传输，上层再对数据进行按特定应用或指定意义进行封装，可以减化步骤，减少多层协议的数据头部，有效的减少数据大小，降低功耗更加的省电。更加具体的BT与BLE的特征变化见表2。

表2经典蓝牙与低功耗蓝牙特征比较

|  |  |  |
| --- | --- | --- |
| 特征 | 经典蓝牙（BR/EDR） | 低功耗蓝牙（LE） |
| 包的类型 | 5种必须包类型  13种BR包类型  10种EDR包类型 | 1种包结构  2种包格式（广播包与数据包） |
| LM/LL控制信息 | 75个链路管理协议信息 | 8种链路层控制信息 |
| LM/LL控制信息 | 9（RFCOMM，BNEP，AVCTP，AVDTP，  HCRP，TCSB1N，MCAP，0BEX，HID，SDP） | 1（ATT属性协议） |

5）成本方面

蓝牙比较擅长的地方就是能够保证互联互通性，蓝牙4.0实际上是在现有经典蓝牙技术（2.1+EDR/3.0+HS）的基础上，进行开发的，因此能够借助经典蓝牙在产业里成熟的渠道进行拓展。另外，蓝牙芯片也能够进行平滑升级，因为新的低功耗单模和双模芯片与传统的芯片接口封装几乎是一致的。也就是说很多手机生产厂商只要把芯片替换一下，软件做一些升级，就可以使用这种技术。在双模应用中，蓝牙低功耗功能集成在现有的经典蓝牙控制器中，或在现有经典蓝牙技术（2.1+EDR/3.0+HS）芯片上增加低功耗堆找，整体架构基本不变，因此成本增加有限。

对于经典蓝牙与低功耗技术在协议不同如表3所示。

表3经典蓝牙与低功耗蓝牙技术协议对比

|  |  |  |
| --- | --- | --- |
| 技术规范 | 经典蓝牙技术 | 低功耗蓝牙技术无线频率 |
| 无线频率 | 2.4GHz | 2.4GHz |
| 距离/范围 | 10-100米 | 50米 |
| （空中数据）传输速率 | l-3Mbps | 1Mbps |
| 应用数据的吞吐速率 | 0.7-2.1Mbps | 0.2Mbps |
| 结点个数 | 7 | 无限多个 |
| 安全性 | 56to128bit | 128-bitAES |
| 健壮性 | FHSS（frequency-hopping | FHSS（spreadspectrum） |
| 延迟（从未连接到发送数据） | 100ms | <3ms |
| 使用范围 | 全球 | 全球 |
| 认证机构 | BluetoothSIG | BluetoothSIG |
| 语音支持 | 能 | 否 |
| 网络拓扑结构 | 点对点，分散网络 | 点对点，星形 |
| 功耗 | 作为参考值1 | 0.01到0.5（根据不同的用例） |
| 服务发现 | M | M |
| Profile定义 | 有 | 有 |
| 主要的应用领域 | 手机，耳机，音像，汽车，手机，游戏，PC机，运动器材，PC机等 | 健康医疗设备，汽车，工业领域，家庭电子设备等 |

3.1.3低功耗蓝牙应用现状

智能蓝牙芯片的功耗极低，是真正的低功耗通信连接技术，仅一枚普通硬币电池即可让智能蓝牙配件工作长达数月甚至数年之久。智能蓝牙装置可将数据信息传输到另一个智能蓝牙装置或集成双模智能蓝牙的智能手机、平板或笔记本电脑。在商业应用领域，苹果，小米，摩托罗拉，微软等公司都开发了基于商业应用的蓝牙4.0可穿戴设备，如我国小米公司的小米手环。

3.1.4BLE体系结构

蓝牙是一种无线通信协议的标准，规定了蓝牙应用产品的软硬件资源应遵循的一些标准和需要达到的要求。到目前为止，蓝牙SIG已经颁布了1.0，1.1，1.2，V2.0+EDR，V2.1+EDR，3.0+HS和4.0第多个版本。本作品的工作主要是根据目前最新版本的V4.0协议进行的。蓝牙协议规范遵循放系统互连参考模型，从低到高定义了蓝牙协议栈的各个层次。蓝牙低功耗协议找的体系结构如图12所示：



图12蓝牙4.0体系结构

在蓝牙4.0核心规范中主要对协议栈按照控制器部分，主机部分及Profile三个部分来进行分别定义。其中各层的主要功能简介如下：

控制器部分（Controller）：

PHY（PhysicalLayer）：用来传送位流的比特原始数据；

LKLinkLayer）：进行数据包的收发和控制。

主机部分（Host）：

L2CAP：协议/通道多路复用，进行数据包的分段与重组；

AttributeProtocol：数据接入协议，实现数据的访问；

AttributeProfile：定义数据的组织结构，主要是由属性和属性值来构成。

Profile部分：

ApplicationProfiles：针对具体应用设备配置实现进行定义相关功能的定义与说明，如BloodPressure，HeartRate，Thermostat等等应用Profiles，目前官方已经定义了二十多种特定应用的规范Profiles。

另外针对不同的应用模式可能还可能需要使用不同的主机控制器接口（HCI）方式，将控制器协议栈部分与主机协议栈部分功能进行相连起来。市场上的蓝牙产品有很多种，如蓝牙终端产品、实现协议控制器部分子系统的蓝牙模块、实现协议主机部分子系统的蓝牙模块，实现协议Profile子系统的蓝牙模块，蓝牙开发工具等。而通常我们使用的蓝牙模块产品可能只是实现控制器协议栈部分，然后提供相应的HCI层接口，供用户在其它的设备如MCU或PC机端来实现协议栈的主机部分及Profile功能，然后通过控制器提供的HCI接口来完成整个蓝牙协议栈的功能，实现特定的应用功能。

3.1.5GAP、GATT的原理及应用

蓝牙4.0协议栈由于在开发过程中底层协议栈的相关代码相对比较固定，与蓝牙应用开发相关的主要是协议栈中的GAP和GATT这两层，所以非常有必要对这两层进行进一步的分析和学习。

（1）GAP的全称为GenericAccessProfile，我们将其翻译为“通用访问规范在蓝牙4.0协议栈中Profile是一个很重要的概念，可以理解为共同约定的配置或规范。只要遵守同一个Profile，不同的蓝牙产品就可以相互发现并和对方建立连接。在实际应用中，GAP层总是工作在以下角色中的一种：

广播员（Broadcaster）：广播数据包，但此时并不连接；

观察者（Observer）：扫描周围的广播数据包，但是不能发起连接；

外设从机（Peripheral）：广播数据包并表明自己处于可连接状态，收到其他设备的连接请求就与其发生连接，大多数蓝牙设备都工作于该状态；

中心主机（Central）：扫描周围的广播数据包并向从机发起连接请求，连接后链路层进入主机角色。可工作于单层或者多层连接。

在大多数蓝牙低功耗系统中，GAP层连接过程如下：外设从机广播特定格式数据包以此让中央主机明白它是可连接的，广播数据一般包括诸如设备地址、设备名称等附加信息。中央主机接收到广播数据后马上会向外设从机发送扫描请求，外设从机收到请求后立刻返回对该请求的回复。至此，设备发现的过程完成。这个时候中央主机已经确认外设从机的存在并且可以与其连接，然后中央主机就可向外设从机发送连接请求以建立连接。

一个连接请求主要包含以下三个连接参数：

通信间隙：低功耗蓝牙连接和通信釆用跳频技术实现。蓝牙设备在某次连接中在某个特定频道上收发数据，在下一次连接时又一起到“约定”好的另外的特定频道上实现数据的收发。两次连接或通信之间的间隔时间就称为通信间隙。使用跳频实现间隔通信的好处一是可以避免频道拥堵，二是可以在一定程度上提高蓝牙通信的安全性。蓝牙设备收发数据的过程我们称之为“连接事件”，在两个“连接事件”之间，设备便处于睡眠状态从而降低耗电量。蓝牙间隙最小单位为0.625ms，故通信间隙只能为1.25ms的整数倍，并且其范围为7.5ms~4s。

不同的应用所要求的通信间隙各不相同，增加或减少通信间隙时间都有各自的利弊，如表4所示：

表4通信间隙长短的利弊

|  |  |
| --- | --- |
| 增加通信间隙时间 | 减小通信间隙时间 |
| 功耗小 | 功耗大 |
| 低数据吞吐量 | 高数据吞吐量 |
| 发送等待时间长 | 发送等待时间短 |
| 数据处理能力减弱 | 数据处理能力增强 |

从机延时：建立连接后，主机会定时发送信息到外设从机，若从机没有信息发送或反馈则可以对这些信息不予理睬，继续休眠。我们将该现象称为从机延时。可以发现，在大多数低功耗蓝牙应用中，只有从机对低功耗有比较奇刻的要求，所以从机延时从一定程度上解决了功耗问题并且增加了连接事件的灵活性。从机延时的最大忽略事件数不能超过499个，时间也不能超过32s。从机延时示意图如图13所示。与通信间隙类似，高从机延时值或低延时值都各有利弊，如表5所示。在设置时我们需要根据实际情况选取合适的延时值以保证系统的正常运行。

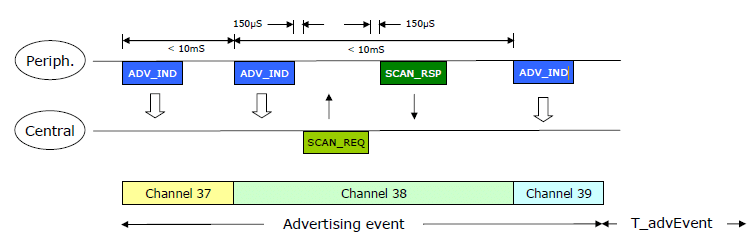


图13从机延时示意图

表5从机延时长短的利弊

|  |  |
| --- | --- |
| 高从机延时 | 低从机延时 |
| 外设功耗低 | 外设功耗高 |
| 外设无法及时收到主机的数据 | 外设能及时收到主机的数据 |

监管超时：该参数为两个连接事件之间最大的时间间隔。如果在该时间内连接没有成功，设备就会默认连接巳经断开并重新回到未连接的状态。监管超时的单位时间为10ms，最大时间为32s。需要注意的是，监管超时值一定要比有效通信间隙时间大。

由上述三个参数我们可以得到有效通信间隙：

等效通信间隙=通信间隙\*（1+从机延时）

根据上述公式开发者可计算出蓝牙设备之间传输数据的平均速率，并以此为依据实现通信参数的最佳配置。

前面介绍的连接参数都是由中央主机单方面决定的，外设从机只能服从这些设定。但在蓝牙4.0协议中，还需考虑外设从机不合适中央主机决定的连接参数，在连接过程中想要改变参数的可能性。于是GAP层协议中引入协商机制：外设从机可向中央主机发送“连接参数更新请求”，中央主机收到该请求后在L2CAP层中进行处理。从机请求一般包括以下四个参数：最大通信间隙、最小通信间隙、监管超时以及从机延时值。参数描述了外设从机对连接的要求，中央主机接收到请求后可以选择拒绝或者执行该从机请求，若选择执行，主机就会更新参数值以实现和外设从机的连接。

虽然连接时只能按照中央主设备发出连接请求，外围设备接受的规范进行，但只要有一方选择断开连接，另外一方必须做出响应并结束当前的连接。

在BLE连接过程中，GAP层同时还会处理数据加密过程，数据只有在有认证的连接中才会处于可读和可写的状态。一旦GAP层链路建立后，主机和从机就进入到配对过程。一般情况下，外设从机会要求中央主机提供一个密码，该密码可以是固定的也可以是随机生成的。配对成功后，主机和从机交换安全密胡并用来加密和认证连接。蓝牙4.0协议中采用长期密胡机制从而省去相同设备连接时不必要的配对过程。这个机制我们称其为“绑定”。可以看到：与传统蓝牙一样，蓝牙4.0协议中的GAP层同样使用“一次配对，永久绑定”的机制。这样既方便了连接，也在一定程度上省去了连接时的功耗。连接建立过程和配对过程如图14所示。



图14 连接建立和配对

（2）GATT的原理和应用

通用属性配置文件层（GATT）定义了使用属性协议（ATT）的服务框架该框架定义了和服务步骤、格式以及属性。包括发现，读，写，通知、显示以及在设备上配置数据属性。该层作为应用程序数据交换的基础层，定义了两种角色：

GATT客户端：为一个启动命令并向服务器请求数据，只能读取或修改服务器发送出给自己的数据。

GATT服务器：通信过程中向外提供数据并响应客户端请求的一端。

GATT指定了profile数据交换所在的结构，除了数据封装方式不同，客户端以及服务器端和Attribute协议结构是相同的：数据封装在Services中，并用Characteristic表示。一个Service可能包含多个特征值（Characteristic），每一个特征值又可以包含一个可选的描述（descriptor字串），用以表不（示）这个特征值的含义。上述描述和第三章中将要介绍的蓝牙连接过程的代码正好对应起来，GATT客户端和服务器的相互响应如图15所示。



图15 GATT服务器和客户端相互响应

## 3.2BLE蓝牙4.0嵌入式软件实现

nRF52832的S110SoftDevice软件包是预编译二进制蓝牙低功耗协议堆栈，集成了低功耗控制器与主机的功能，可独立编程和更新，提供了全功能的API供系统调用。特点包括：异步和事件驱动SVC的API；运行时保护；GATT、GAP和L2CAP级别API；广播器功能；GATT客户端和服务器；具有少于128kB的代码和6kB的RAM使用量，为应用程序留有超过128kB的闪存和10kB的RAM；S110SoftDevice和nRF52832加上nRF51SDK相互配合，包含全面的蓝牙低功耗配置文件、服务以及示例应用集合。下载nRF52832的S110软件包及nRF51SDK并安装，即可在软件中调用相关API实现系统功能。S110SoftDevice位于nRF52832芯片的Memory底部，调用过程如下：

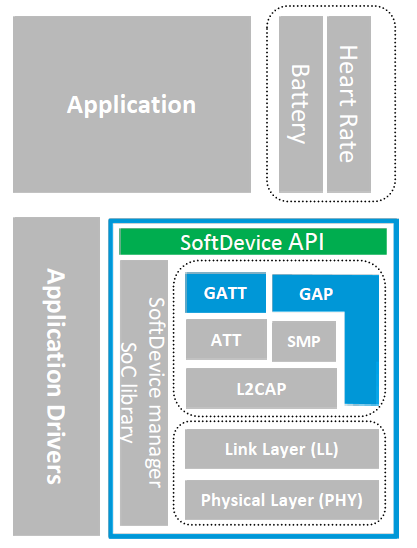
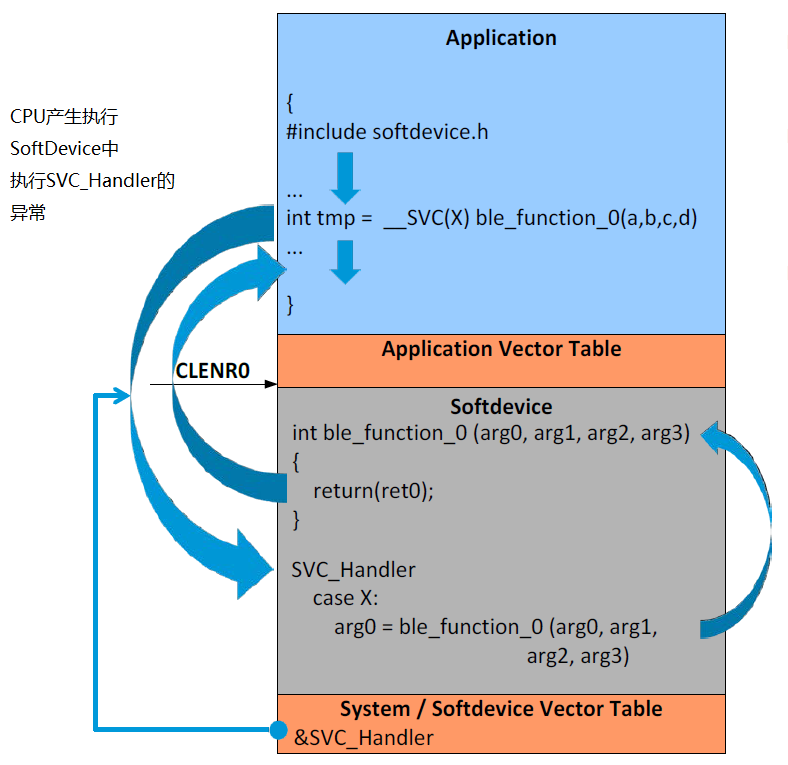


图16 Softdevice调用过程

4.0蓝牙的具体通讯实现步骤如下：

1）主程序通过ble\_evt\_get()查找所有的BLE低功耗蓝牙设备，如图17所示；



图17ble\_evt\_get()调用图

2）通过GAP层的SVC调用函数进行蓝牙匹配，主要函数如下：

GAP服务：

ble\_gap\_device\_name\_set（security，name）：设置获取的蓝牙设备名

ble\_gap\_appearance\_set（appearance）：设置当前设备准备提交另一蓝牙设备的指令

ble\_gap\_ppcp\_set（ppcp）：定义链接参数

GAP广播：

ble\_gap\_adv\_data\_set（adv\_data，ad\_len，sr\_data，sr\_len）：设置中心设备收到的广播参数

ble\_gap\_adv\_start（adv\_params）：开始发送广播包

GAP事件：

BLE\_GAP\_EVT\_CONNECTED{conn\_handle，peer\_addr，conn\_params}：中心蓝牙设备已经建立一个物理连接

BLE\_GAP\_EVT\_DISCONNECTED{conn\_handle，reason}：连接终止

BLE\_GAP\_EVT\_CONN\_PARAM\_UPDATE{conn\_params}：连接参数更新过程已经完成

BLE\_GAP\_EVT\_TIMEOUT{source}：连接超时

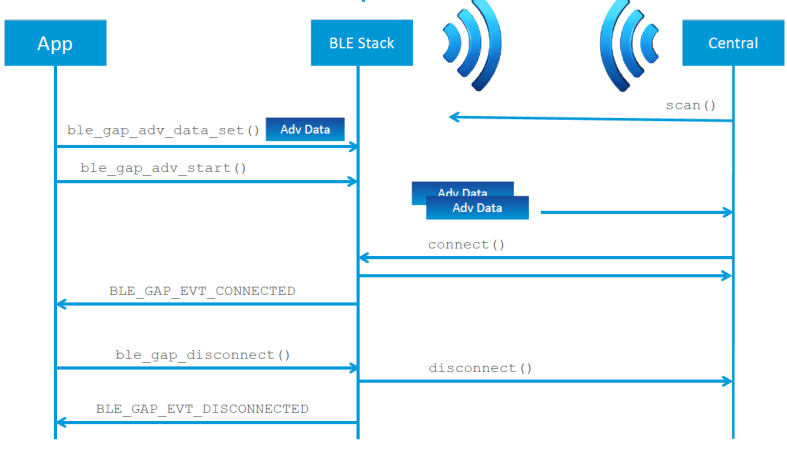


图18GAP建立连接流程

3）GATTS为设备的互联层，服务器（Server）与客户端（Client）之间的数据交换是通过属性表来维护的。

ble\_gatts\_service\_add（type，UUID，out\_handle）：给ATT加一个空表

ble\_gatts\_characteristic\_add（svc\_handle，md，value，out\_handles）：给当前服务增加Characteristic

ATT本地属性表设置：

ble\_gatts\_value\_set（handle，offset，len，value）：设置属性表值

ble\_gatts\_value\_get（handle，offset，len，value）：获得某一属性表值

Getsthevalueofanyparticularattribute：服务初始化

ble\_gatts\_MOVE（conn\_handle，params）：发送节点传感器参数

GATTS事件：

BLE\_GATTS\_EVT\_WRITE{conn\_handle，handle，data}：ATT写操作完成

BLE\_GATTS\_EVT\_HVC{conn\_handle，handle}：从另一蓝牙设备成功接收一个句柄配置值

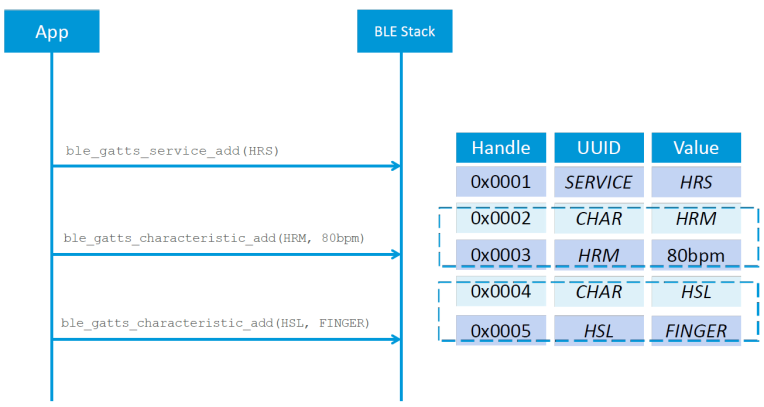


图19GAPP建立连接流程

4）GATTS为设备的互联层



图20BLE数据交换

5）蓝牙嵌入式系统主程序

intmain(void)

{

uint8\_tt;

uint8\_tid;

uint32\_terr\_code;

//存放采集的运动数据，6组数据共占12字节，剩余8字节预留

externuint8\_tdata\_6[20];

int16\_ttem1[3],tem2[3];

twi\_master\_init(); //初始化MPU6050与nRF52832接口

if(mpu6050\_init(0x68)==false)

{//调试使用

}

mpu6050\_register\_read(0x75U,&id,1);

timers\_init();

gpiote\_init();

buttons\_init();

if(is\_first\_start())

{

GPIO\_WAKEUP\_BUTTON\_CONFIG(HR\_INC\_BUTTON\_PIN\_NO);

GPIO\_WAKEUP\_BUTTON\_CONFIG(HR\_DEC\_BUTTON\_PIN\_NO);

NRF\_POWER->SYSTEMOFF=0;//启动时不休眠”

ble\_stack\_init();//初始化蓝牙协议栈

radio\_notification\_init();

gap\_params\_init();//gap蓝牙参数

advertising\_init();//蓝牙广播初始化

services\_init();//蓝牙服务协议初始化

conn\_params\_init();//连接参数初始化

sec\_params\_init();

advertising\_start();//开始广播

//设置MPU6050采集器数据

if(mpu6050\_init(0x68)==false)

{//调试用

}

mpu6050\_register\_read(0x75U,&id,1);

for(;;)

{

MPU6050\_ReadAcc0(&tem1[0],&tem1[1],&tem1[2]);//读加速度数据

\*(data\_6+1)=tem1[0]&0xff;

\*(data\_6+0)=tem1[0]>>8;

\*(data\_6+3)=tem1[1]&0xff;

\*(data\_6+2)=tem1[1]>>8;

\*(data\_6+5)=tem1[2]&0xff;

\*(data\_6+4)=tem1[2]>>8;

MPU6050\_ReadGyro0(&tem2[0],&tem2[1],&tem2[2]);//读陀螺仪数据

\*(data\_6+7)=tem2[0]&0xff;

\*(data\_6+6)=tem2[0]>>8;

\*(data\_6+9)=tem2[1]&0xff;

\*(data\_6+8)=tem2[1]>>8;

\*(data\_6+11)=tem2[2]&0xff;

\*(data\_6+10)=tem2[2]>>8;

err\_code=sd\_app\_event\_wait();

APP\_ERROR\_CHECK(err\_code);

nrf\_delay\_us(10000);

nrf\_delay\_us(10000);

}

}

## 3.3MPU6050姿态采集程序实现

MPU6050与nRF52832通过I2C总线接口进行通信，其电路结构如图所示。总线拥有二线接口，包括串行数据线(SDA)和串行时钟线(SCL)。通过串行数据线和串行时钟线在连接到总线的器件间传递信息，数据从高位到低位依次传输。接口串行数据传输速率在标准模式下可达100kbit/s，快速模式下可达400kbit/s，高速模式下可达3.4.Mbit/s，工作模式是主从式，连接到接口的设备可做主设备或从设备，如图所示。

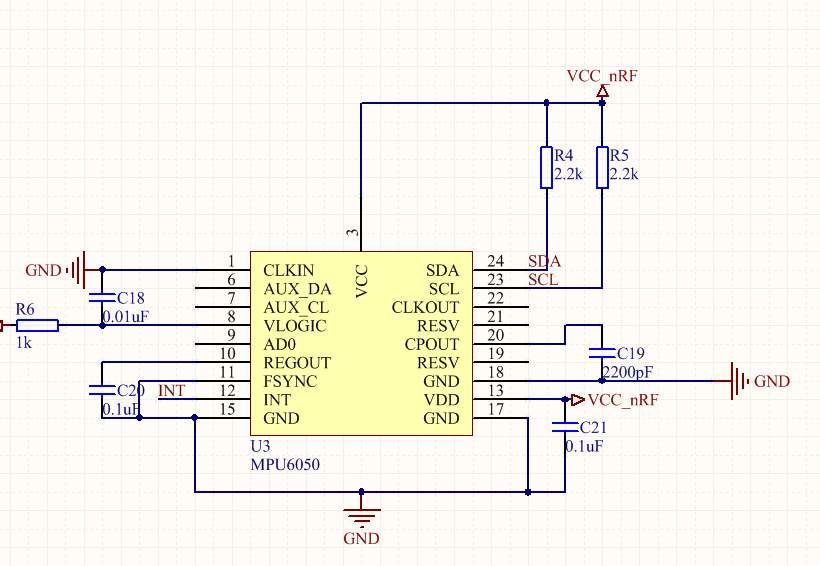


图21MPU6050电路图

作品开发小组在设计电路时，充分考虑电路的升级特性，因此电路与MPU9150兼容。如图所示，MPU6050通过23，24引脚与MPU实现I2C通信。为了更大限度的增加i-Baby的工作时间，MPU605012引脚的INT引脚与nRF52832的14引脚相连，当一段时间nRF52832没有收到有效的MPU6050运动数据，系统认为宝宝正在休息，nRF52832进入到休眠状态。当监测到有姿态变化时，MPU6050的通过中断唤醒nRF52832。

作品开发小组通过测试发现，陀螺仪工作运作电流约为5mA，陀螺仪待命电流只有5uA，角速度加速器工作电流为500uA，加速器省电模式电流仅为40uA@10Hz，因为当nRF52832休眠时候，编程陀螺仪进入到休眠状态，直到MPU6050加速度传感器有运动数据时唤醒。MPU6050寄存器设定通过mpu6050\_register\_write()、mpu6050\_register\_read()两个函数实现。

boolmpu6050\_register\_write(uint8\_tregister\_address,uint8\_tvalue)

{

uint8\_tw2\_data[2];

w2\_data[0]=register\_address;

w2\_data[1]=value;

returntwi\_master\_transfer(m\_device\_address,w2\_data,2,TWI\_ISSUE\_STOP);

}

boolmpu6050\_register\_read(uint8\_tregister\_address,uint8\_t\*destination,uint8\_tnumber\_of\_bytes)

{

booltransfer\_succeeded;

transfer\_succeeded=twi\_master\_transfer(m\_device\_address,&register\_address,1,TWI\_DONT\_ISSUE\_STOP);

transfer\_succeeded&=twi\_master\_transfer(m\_device\_address|TWI\_READ\_BIT,destination,number\_of\_bytes,TWI\_ISSUE\_STOP);

returntransfer\_succeeded;

}

MPU6050读加速度和陀螺仪数据分别通过MPU6050\_ReadGyro()和MPU6050\_ReadAcc()两个函数实现。

voidMPU6050\_ReadGyro(int16\_t\*pGYRO\_X,int16\_t\*pGYRO\_Y,int16\_t\*pGYRO\_Z)

{

uint8\_tbuf[6];

uint8\_taddr=MPU6050\_ADDRESS<<1;

mpu6050\_register\_read(MPU6050\_GYRO\_OUT,buf,6);

\*pGYRO\_X=(buf[0]<<8)|buf[1];

if(\*pGYRO\_X&0x8000)\*pGYRO\_X-=65536;

\*pGYRO\_Y=(buf[2]<<8)|buf[3];

if(\*pGYRO\_Y&0x8000)\*pGYRO\_Y-=65536;

\*pGYRO\_Z=(buf[4]<<8)|buf[5];

if(\*pGYRO\_Z&0x8000)\*pGYRO\_Z-=65536;

}

voidMPU6050\_ReadAcc(int16\_t\*pACC\_X,int16\_t\*pACC\_Y,int16\_t\*pACC\_Z)

{

uint8\_tbuf[6];

uint8\_taddr=MPU6050\_ADDRESS<<1;

mpu6050\_register\_read(MPU6050\_ACC\_OUT,buf,6);

\*pACC\_X=(buf[0]<<8)|buf[1];

if(\*pACC\_X&0x8000)\*pACC\_X-=65536;

\*pACC\_Y=(buf[2]<<8)|buf[3];

if(\*pACC\_Y&0x8000)\*pACC\_Y-=65536;

\*pACC\_Z=(buf[4]<<8)|buf[5];

if(\*pACC\_Z&0x8000)\*pACC\_Z-=65536;

}

MPU6050通过4元素方法，获取宝宝实时姿态，其核心函数如下：

uint8\_tmpu\_dmp\_get\_data(float\*pitch,float\*roll,float\*yaw)

{

floatq0=1.0f,q1=0.0f,q2=0.0f,q3=0.0f;

unsignedlongsensor\_timestamp;

shortgyro[3],accel[3],sensors;

unsignedcharmore;

longquat[4];

if(dmp\_read\_fifo(gyro,accel,quat,&sensor\_timestamp,&sensors,&more))return1; if(sensors&INV\_WXYZ\_QUAT)

{

q0=quat[0]/q30; //q30格式转换为浮点数

q1=quat[1]/q30;

q2=quat[2]/q30;

q3=quat[3]/q30;

//计算得到俯仰角/横滚角/航向角

\*pitch=asin(-2\*q1\*q3+2\*q0\*q2)\*57.3; //pitch

\*roll=atan2(2\*q2\*q3+2\*q0\*q1,-2\*q1\*q1-2\*q2\*q2+1)\*57.3; //roll

\*yaw=atan2(2\*(q1\*q2+q0\*q3),q0\*q0+q1\*q1-q2\*q2-q3\*q3)\*57.3; //yaw

}elsereturn2;

return0;

}

## 3.4MAX30100心率采集程序

i-Baby采用MAX30100传感器芯片模块，MAX30100是一款集成有脉搏血氧仪和心率监测传感器的模块。该器件集成有两个LED、一个光电探测器,经过优化的光学器件和低噪声模拟信号处理器，可检测脉搏血氧及心率信号。MAX30100采用1.8V和3.3V的电源电压。可通过软件来关断电源，待机模式下的电流消耗量可忽略不计，因而可以始终保持电源连接。典型应用有健身辅助设备、医疗监控设备和可穿戴设备。

在写MAX30100芯片硬件驱动时主要问题在于对FIFO的操作上，在此需要注意的细节主要体现在，在使用IIC协议对FIFO的寄存器进行操作时，首先根据项目需求，如采集样本数及精度等，配置config寄存器，接着需要对FIFO的各个寄存器进行清零（写0x00），其中包括写寄存器、读寄存器、计数寄存器和数据寄存器，最后才进行写操作或读操作。

其心率核心代码如下：

staticuint8\_treport\_byte=0;

shorttemp\_short;

floattemp\_float;

POupdate();

if(millis()-tsLastReport>=50){

HeartRate=getHeartRate();

if(!finger\_leave){

HR=(uint16\_t)(100.0f\*(60.0f+HeartRate/4.7f));

}else{

HR=0;

}

switch(last\_byte[report\_byte])

{

caseLAST\_BYTE\_ACCEL:{

ble\_data[0]=((accel[0]&0xff00)>>8)&0xff;

ble\_data[1]=((accel[0]&0x00ff)>>0)&0xff;

ble\_data[2]=((accel[1]&0xff00)>>8)&0xff;

ble\_data[3]=((accel[1]&0x00ff)>>0)&0xff;

ble\_data[4]=((accel[2]&0xff00)>>8)&0xff;

ble\_data[5]=((accel[2]&0x00ff)>>0)&0xff;

temp\_short=(ble\_data[0]<<8)|ble\_data[1];

printf("accel:%10d%10d%10d\r\n",accel[0],accel[1],accel[2]);

ble\_data[6]=LAST\_BYTE\_ACCEL;

}break;

caseLAST\_BYTE\_GYRO:{

ble\_data[0]=((gyro[0]&0xff00)>>8)&0xff;

ble\_data[1]=((gyro[0]&0x00ff)>>0)&0xff;

ble\_data[2]=((gyro[1]&0xff00)>>8)&0xff;

ble\_data[3]=((gyro[1]&0x00ff)>>0)&0xff;

ble\_data[4]=((gyro[2]&0xff00)>>8)&0xff;

ble\_data[5]=((gyro[2]&0x00ff)>>0)&0xff;

temp\_short=(ble\_data[0]<<8)|ble\_data[1];

printf("gyro0:%10d%10d%10d\r\n",gyro[0],gyro[1],gyro[2]);

ble\_data[6]=LAST\_BYTE\_GYRO;

}break;

caseLAST\_BYTE\_HEART\_RATE:{

ble\_data[0]=(((HR&0xff00)>>8))&0xff;

ble\_data[1]=(((HR&0x00ff)>>0))&0xff;

ble\_data[2]=0;

ble\_data[3]=0;

ble\_data[4]=0;

ble\_data[5]=0;

ble\_data[6]=LAST\_BYTE\_HEART\_RATE;

temp\_float=(float)((ble\_data[0]<<24)|(ble\_data[1]<<16)|(ble\_data[2]<<8)|(ble\_data[3]))/1000;

// printf("HR:%10dtemp:%lf\r\n",HR,temp\_float);

}break;

caseLAST\_BYTE\_TEMPERATURE:{

ble\_data[0]=(((temperature&0xff00)>>8))&0xff;

ble\_data[1]=(((temperature&0x00ff)>>0))&0xff;

ble\_data[2]=0;

ble\_data[3]=0;

ble\_data[4]=0;

ble\_data[5]=0;

ble\_data[6]=LAST\_BYTE\_TEMPERATURE;

temp\_short=(ble\_data[0]<<8)|ble\_data[1];

printf("temperature:%10dtemp:%d\r\n",temperature,temp\_short);

}break;

caseLAST\_BODY\_STATUS:{

temp\_short=roll\*100;

ble\_data[0]=(((temp\_short&0xff00)>>8))&0xff;

ble\_data[1]=(((temp\_short&0x00ff)>>0))&0xff;

temp\_short=pitch\*100;

ble\_data[2]=(((temp\_short&0x00ff)>>8))&0xff;

ble\_data[3]=(((temp\_short&0x00ff)>>0))&0xff;

temp\_short=yaw\*100;

ble\_data[4]=(((temp\_short&0x00ff)>>8))&0xff;

ble\_data[5]=(((temp\_short&0x00ff)>>0))&0xff;

ble\_data[6]=LAST\_BODY\_STATUS;

temp\_float=(float)((int16\_t)((ble\_data[0]<<8)|ble\_data[1]))/100;

printf("roll:%10lfpit:%10lfyaw:%10lf\r\n",roll,pitch,yaw);

}break;

default:break;

}

ble\_nus\_string\_send(&m\_nus,ble\_data,7);

report\_byte++;

if(report\_byte>DATA\_CYCLE)report\_byte=1;

tsLastReport=millis();

}

## 3.5MEMS数字麦克风实现

MEMS（微型机电系统）麦克风是基于MEMS技术制造的麦克风，简单的说就是一个电容器集成在微晶片上，可以采用表贴工艺制造，能够承受很高的回流焊温度。根据本作品的应用特点，采用ADI公司的ADMP441数字麦克风作为作品声音采集，具有高性能，低功耗，数字输出等特点。ADMP441采用I2S与nRF52832接口，其核心代码如下：

uint32\_t\*p\_data\_received=NULL;

uint32\_t\*p\_data\_to\_send=NULL;

if(nrf\_i2s\_event\_check(NRF\_I2S,NRF\_I2S\_EVENT\_TXPTRUPD))

{

nrf\_i2s\_event\_clear(NRF\_I2S,NRF\_I2S\_EVENT\_TXPTRUPD);

if(m\_cb.p\_tx\_buffer!=NULL)

{

uint32\_t\*p\_tx\_buffer\_next;

if(nrf\_i2s\_tx\_buffer\_get(NRF\_I2S)==m\_cb.p\_tx\_buffer)

{

p\_tx\_buffer\_next=m\_cb.p\_tx\_buffer+m\_cb.buffer\_half\_size;

}

else

{

p\_tx\_buffer\_next=m\_cb.p\_tx\_buffer;

}

nrf\_i2s\_tx\_buffer\_set(NRF\_I2S,p\_tx\_buffer\_next);

m\_cb.tx\_ready=true;

p\_data\_to\_send=p\_tx\_buffer\_next;

}

}

if(nrf\_i2s\_event\_check(NRF\_I2S,NRF\_I2S\_EVENT\_RXPTRUPD))

{

nrf\_i2s\_event\_clear(NRF\_I2S,NRF\_I2S\_EVENT\_RXPTRUPD);

if(m\_cb.p\_rx\_buffer!=NULL)

{

uint32\_t\*p\_rx\_buffer\_next;

if(nrf\_i2s\_rx\_buffer\_get(NRF\_I2S)==m\_cb.p\_rx\_buffer)

{

p\_rx\_buffer\_next=m\_cb.p\_rx\_buffer+m\_cb.buffer\_half\_size;

}

else

{

p\_rx\_buffer\_next=m\_cb.p\_rx\_buffer;

}

nrf\_i2s\_rx\_buffer\_set(NRF\_I2S,p\_rx\_buffer\_next);

m\_cb.rx\_ready=true;

if(m\_cb.just\_started&&!m\_cb.synchronized\_mode)

{

m\_cb.just\_started=false;

}

else

{

p\_data\_received=p\_rx\_buffer\_next;

}

}

}

if(!m\_cb.synchronized\_mode)

{

if((p\_data\_received!=NULL)||(p\_data\_to\_send!=NULL))

{

m\_cb.handler(p\_data\_received,p\_data\_to\_send,

m\_cb.buffer\_half\_size);

}

}

{

if(m\_cb.rx\_ready&&m\_cb.tx\_ready)

{

m\_cb.rx\_ready=false;

m\_cb.tx\_ready=false;

if(m\_cb.just\_started)

{

m\_cb.just\_started=false;

}

else

{

m\_cb.handler(nrf\_i2s\_rx\_buffer\_get(NRF\_I2S),

nrf\_i2s\_tx\_buffer\_get(NRF\_I2S),

m\_cb.buffer\_half\_size);

}

}

}

## 3.6安卓软件实现

GenericAttributeProfile（GATT）—GATT配置文件是一个通用规范，用于在BLE链路上发送和接收被称为“属性”的数据块。目前所有的BLE应用都基于GATT。蓝牙SIG规定了许多低功耗设备的配置文件。配置文件是设备如何在特定的应用程序中工作的规格说明。注意一个设备可以实现多个配置文件。例如，一个设备可能包括心率监测仪和电量检测。

（1）BLE权限

为了在app中使用蓝牙功能，必须声明蓝牙权限BLUETOOTH。利用这个权限去执行蓝牙通信，例如请求连接、接受连接、和传输数据[17]。

如果想让你的app启动设备发现或操纵蓝牙设置，必须声明BLUETOOTH\_ADMIN权限。注意：如果你使用BLUETOOTH\_ADMIN权限，你也必须声明BLUETOOTH权限。

在你的appmanifest文件中声明蓝牙权限：

<uses-permissionandroid：name="android.permission.BLUETOOTH"/>  
<uses-permissionandroid：name="android.permission.BLUETOOTH\_ADMIN"/>

<uses-featureandroid：name="android.hardware.bluetooth\_le"android：required="true"/>

但是如果想让你的app提供给那些不支持BLE的设备，需要在manifest中包括上面代

码并设置required="false"，然后在运行时可以通过PackageManager.hasSystemFeature()确定BLE的可用性。

//使用此检查确定BLE是否支持在设备上，然后你可以有选择性禁用BLE相关的功能。

if(!getPackageManager().hasSystemFeature(PackageManager.FEATURE\_BLUETOOTH\_L)

{Toast.makeText(this,R.string.ble\_not\_supported,Toast.LENGTH\_SHORT).show();  
finish();  
}

（2）设置BLE

app能与BLE通信之前，你需要确认设备是否支持BLE，如果支持，确认已经启用。注意如果<uses-feature.../>设置为false，这个检查才是必需的。

如果不支持BLE，那么你应该适当地禁用部分BLE功能。如果支持BLE但被禁用，你可以无需离开应用程序而要求用户启动蓝牙。使用BluetoothAdapter两步完成该设置。

获取BluetoothAdapter。

所有的蓝牙活动都需要蓝牙适配器。BluetoothAdapter代表设备本身的蓝牙适配器(蓝牙无线）。整个系统只有一个蓝牙适配器，而且你的app使用它与系统交互。下面的代码片段显示了如何得到适配器。注意该方法使用getSystemService（）]返回BluetoothManager，然后将其用于获取适配器的一个实例。Android4.3（API18）引入BluetoothManager。

//初始化蓝牙适配器  
finalBluetoothManagerbluetoothManager=  
(BluetoothManager)getSystemService(Context.BLUETOOTH\_SERVICE);  
mBluetoothAdapter=bluetoothManager.getAdapter();

开启蓝牙

接下来，你需要确认蓝牙是否开启。调用[isEnabled()](http://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#isEnabled())去检测蓝牙当前是否开启。如果该方法返回false,蓝牙被禁用。下面的代码检查蓝牙是否开启，如果没有开启，将显示错误提示用户去设置开启蓝牙。

//确保蓝牙在设备上可以开启

if(mBluetoothAdapter==null||!mBluetoothAdapter.isEnabled()){

IntentenableBtIntent=newIntent(BluetoothAdapter.ACTION\_REQUEST\_ENABLE);  
startActivityForResult(enableBtIntent,REQUEST\_ENABLE\_BT);

（3）发现BLE设备

为了发现BLE设备，使用[startLeScan()](http://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#startLeScan(android.bluetooth.BluetoothAdapter.LeScanCallback))方法。这个方法需要一个参数[BluetoothAdapter.LeScanCallback](http://developer.android.com/reference/android/bluetooth/BluetoothAdapter.LeScanCallback.html)。你必须实现它的回调函数，那就是返回的扫描结果。因为扫描非常消耗电量，你应当遵守以下准则：

只要找到所需的设备，停止扫描。

不要在循环里扫描，并且对扫描设置时间限制。以前可用的设备可能已经移出范围，继续扫描消耗电池电量。

/\*\*  
\*扫描和显示可以提供的蓝牙设备.  
\*/  
publicclassDeviceScanActivityextendsListActivity{  
privateBluetoothAdaptermBluetoothAdapter;  
privatebooleanmScanning;  
privateHandlermHandler;  
//10秒后停止寻找.  
privatestaticfinallongSCAN\_PERIOD=10000;  
privatevoidscanLeDevice(finalbooleanenable){

if(enable){  
//经过预定扫描期后停止扫描  
mHandler.postDelayed(newRunnable(){  
@Override  
publicvoidrun(){  
mScanning=false;  
mBluetoothAdapter.stopLeScan(mLeScanCallback);  
}  
},SCAN\_PERIOD);  
mScanning=true;  
mBluetoothAdapter.startLeScan(mLeScanCallback);  
}else{  
mScanning=false;  
mBluetoothAdapter.stopLeScan(mLeScanCallback);  
}  
}

返回DeviceControlActivity,这些事件由一个BroadcastReceiver来处理：

//通过服务控制不同的事件  
//ACTION\_GATT\_CONNECTED：连接到GATT服务端  
//ACTION\_GATT\_DISCONNECTED：未连接GATT服务端.  
//ACTION\_GATT\_SERVICES\_DISCOVERED：未发现GATT服务.  
//ACTION\_DATA\_AVAILABLE：接受来自设备的数据，可以通过读或通知操作获得。  
privatefinalBroadcastReceivermGattUpdateReceiver=new

BroadcastReceiver(){  
@Override  
publicvoidonReceive(Contextcontext,Intentintent){

finalStringaction=intent.getAction();  
if(BluetoothLeService.ACTION\_GATT\_CONNECTED.equals(action)){  
mConnected=true;  
updateConnectionState(R.string.connected);  
invalidateOptionsMenu();

}elseif

(BluetoothLeService.ACTION\_GATT\_DISCONNECTED.equals(action))

{  
mConnected=false;  
updateConnectionState(R.string.disconnected);  
invalidateOptionsMenu();  
clearUI();  
}elseif(BluetoothLeService.ACTION\_GATT\_SERVICES\_DISCOVERED.equals(action))

{//在用户接口上展示所有的servicesandcharacteristicsdisplayGattServices(mBluetoothLeService.getSupportedGattServices());  
}elseif(BluetoothLeService.ACTION\_DATA\_AVAILABLE.equals(action)){displayData(intent.getStringExtra(BluetoothLeService.EXTRA\_DATA));}  
}  
};  
}

（4）连接到GATT服务端

与一个BLE设备交互的第一步就是连接它——更具体的，连接到BLE设备上的GATT服务端。为了连接到BLE设备上的GATT服务端，需要使用connectGatt()方法。这个方法需要三个参数：一个Context对象，自动连接（boolean值,表示只要BLE设备可用是否自动连接到它），和BluetoothGattCallback调用。

mBluetoothGatt=device.connectGatt(this,false,mGattCallback);

连接到GATT服务端时，由BLE设备做主机，并返回一个BluetoothGatt实例，然后你可以使用这个实例来进行GATT客户端操作。请求方（Androidapp)是GATT客户端。BluetoothGattCallback用于传递结果给用户，例如连接状态，以及任何进一步GATT客户端操作。在这个例子中，这个BLEAPP提供了一个activity（DeviceControlActivity)来连接，显示数据，显示该设备支持的GATTservices和characteristics。根据用户的输入，这个activity与BluetoothLeService通信，通过AndroidBLEAPI实现与BLE设备交互。

//通过BLEAPI服务端与BLE设备交互publicclassBluetoothLeServiceextendsService{  
privatefinalstaticStringTAG=BluetoothLeService.class.getSimpleName();  
privateBluetoothManagermBluetoothManager;//蓝牙管理器  
privateBluetoothAdaptermBluetoothAdapter;//蓝牙适配器  
privateStringmBluetoothDeviceAddress;//蓝牙设备地址  
privateBluetoothGattmBluetoothGatt;privateintmConnectionState=STATE\_DISCONNECTED;  
privatestaticfinalintSTATE\_DISCONNECTED=0;//设备无法连接  
privatestaticfinalintSTATE\_CONNECTING=1;//设备正在连接状态  
privatestaticfinalintSTATE\_CONNECTED=2;//设备连接完毕  
publicfinalstaticStringACTION\_GATT\_CONNECTED=  
"com.example.bluetooth.le.ACTION\_GATT\_CONNECTED";  
publicfinalstaticStringACTION\_GATT\_DISCONNECTED=  
"com.example.bluetooth.le.ACTION\_GATT\_DISCONNECTED";  
publicfinalstaticStringACTION\_GATT\_SERVICES\_DISCOVERED=  
"com.example.bluetooth.le.ACTION\_GATT\_SERVICES\_DISCOVERED";  
publicfinalstaticStringACTION\_DATA\_AVAILABLE=  
"com.example.bluetooth.le.ACTION\_DATA\_AVAILABLE";  
publicfinalstaticStringEXTRA\_DATA=  
"com.example.bluetooth.le.EXTRA\_DATA";  
publicfinalstaticUUIDUUID\_HEART\_RATE\_MEASUREMENT=  
UUID.fromString(SampleGattAttributes.HEART\_RATE\_MEASUREMENT);  
//通过BLEAPI的不同类型的回调方法  
privatefinalBluetoothGattCallbackmGattCallback=  
newBluetoothGattCallback(){  
@Override  
publicvoidonConnectionStateChange(BluetoothGattgatt,intstatus,  
intnewState){

//当连接状态发生改变  
StringintentAction;

if(newState==BluetoothProfile.STATE\_CONNECTED){

//当蓝牙设备已经连接  
intentAction=ACTION\_GATT\_CONNECTED;  
mConnectionState=STATE\_CONNECTED;  
broadcastUpdate(intentAction);  
Log.i(TAG,"ConnectedtoGATTserver.");  
Log.i(TAG,"Attemptingtostartservicediscovery："+  
mBluetoothGatt.discoverServices());  
}elseif(newState==BluetoothProfile.STATE\_DISCONNECTED){//当设备无法连接  
intentAction=ACTION\_GATT\_DISCONNECTED;  
mConnectionState=STATE\_DISCONNECTED;  
Log.i(TAG,"DisconnectedfromGATTserver.");  
broadcastUpdate(intentAction);  
}  
}  
@Override  
//发现新服务端

publicvoidonServicesDiscovered(BluetoothGattgatt,intstatus){  
if(status==BluetoothGatt.GATT\_SUCCESS){  
broadcastUpdate(ACTION\_GATT\_SERVICES\_DISCOVERED);  
}else{  
Log.w(TAG,"onServicesDiscoveredreceived："+status);  
}  
}  
@Override

//读写特性

publicvoidonCharacteristicRead(BluetoothGattgatt,  
BluetoothGattCharacteristiccharacteristic,  
intstatus){  
if(status==BluetoothGatt.GATT\_SUCCESS){  
broadcastUpdate(ACTION\_DATA\_AVAILABLE,characteristic);  
}  
}  
...  
};  
...  
}

当一个特定的回调被触发的时候，它会调用相应的broadcastUpdate()辅助方法并且传递给它一个action。注意在该部分中的数据解析按照蓝牙心率测量配置文件规格进行。

privatevoidbroadcastUpdate(finalStringaction){  
finalIntentintent=newIntent(action);  
sendBroadcast(intent);  
privatevoidbroadcastUpdate(finalStringaction,  
finalBluetoothGattCharacteristiccharacteristic){  
finalIntentintent=newIntent(action);  
  
//这是测量配置文件。  
if(UUID\_HEART\_RATE\_MEASUREMENT.equals(characteristic.getUuid())){  
intflag=characteristic.getProperties();  
intformat=-1;  
if((flag&x01)!=0){  
format=BluetoothGattCharacteristic.FORMAT\_UINT16;  
Log.d(TAG,"HeartrateformatUINT16.");  
}else{  
format=BluetoothGattCharacteristic.FORMAT\_UINT8;  
Log.d(TAG,"HeartrateformatUINT8.");  
}  
finalintheartRate=characteristic.getIntValue(format,1);  
Log.d(TAG,String.format("Receivedheartrate：%d",heartRate));  
intent.putExtra(EXTRA\_DATA,String.valueOf(heartRate));  
}else{  
//对于所有其它的配置文件，用十六进制格式写数据  
finalbyte[]data=characteristic.getValue();  
if(data!=null&&data.length>0){  
finalStringBuilderstringBuilder=newStringBuilder(data.length);  
for(bytebyteChar：data)  
stringBuilder.append(String.format("%02X",byteChar));  
intent.putExtra(EXTRA\_DATA,newString(data)+"\n"+  
stringBuilder.toString());  
}  
}  
sendBroadcast(intent);  
}

（5）接收GATT通知

当设备上的特性改变时会通知BLE应用程序。这段代码显示了如何使用setCharacteristicNotification()给一个特性设置通知

privateBluetoothGattmBluetoothGatt;  
BluetoothGattCharacteristiccharacteristic;  
booleanenable  
mBluetoothGatt.setCharacteristicNotification(characteristic,enabled);  
BluetoothGattDescriptordescriptor=characteristic.getDescriptor(UUID.fromString(SampleGattAttributes.CLIENT\_CHARACTERISTIC\_CONFIG));  
descriptor.setValue(BluetoothGattDescriptor.ENABLE\_NOTIFICATION\_VALUE);  
mBluetoothGatt.writeDescriptor(descriptor);

如果对一个特性启用通知,当远程蓝牙设备特性发送变化，回调函数[onCharacteristicChanged（()](http://developer.android.com/reference/android/bluetooth/BluetoothGattCallback.html" \l "onCharacteristicChanged(android.bluetooth.BluetoothGatt, android.bluetooth.BluetoothGattCharacteristic" \t "http://ricardoli.com/2014/07/31/%E8%93%9D%E7%89%9940%E2%80%94%E2%80%94android-ble%E5%BC%80%E5%8F%91%E5%AE%98%E6%96%B9%E6%96%87%E6%A1%A3%E7%BF%BB%E8%AF%91/_blank))被触发。

@Override  
//广播更新  
publicvoidonCharacteristicChanged(BluetoothGattgatt,  
BluetoothGattCharacteristiccharacteristic){  
broadcastUpdate(ACTION\_DATA\_AVAILABLE,characteristic);  
}

（6）断开连接

publicvoidclose(){  
if(mBluetoothGatt==null){  
return;  
}  
mBluetoothGatt.close();  
mBluetoothGatt=null;

（7）体温监控

publicclassTemperatureHandlerimplementsMessageHandler{

privateMessageHelperhelper;

privateInfoTypepreState=InfoType.NOTIFY\_TEMP\_NORMAL;

publicTemperatureHandler(MessageHelperhelper){

this.helper=helper;

}

@Override

publicvoidhandle(Infoinfo){

if(info.getType()!=InfoType.VALUE\_TEMPERATURE){

return;

}

Temperaturetemperature=((Temperature)info.getContent());

if(temperature==null){

return;

}

doublevalue=temperature.getValue();

InfoTypestate;

if(value>38.5){

state=InfoType.NOTIFY\_TEMP\_HIGH;

}elseif(value>36.5){

state=InfoType.NOTIFY\_TEMP\_NORMAL;

}else{

state=InfoType.NOTIFY\_TEMP\_LOW;

}

if(state!=preState){

preState=state;

InfonewInfo=newInfo(state);

helper.broadcast(newInfo);

if(state==InfoType.NOTIFY\_TEMP\_HIGH){

helper.tempHighNotify(newInfo);

}elseif(state==InfoType.NOTIFY\_TEMP\_LOW){

helper.tempLowNotify(newInfo);

}

}

}

}

（8）睡姿监控

publicclassGyroscopeHandlerimplementsMessageHandler{

privateMessageHelperhelper;

privateGesturepreGesture;

publicGyroscopeHandler(MessageHelperhelper){

this.helper=helper;

}

@Override

publicvoidhandle(Infoinfo){

if(info.getType()!=InfoType.VALUE\_GYROSCOPE){

return;

}

Gyroscopegyroscope=(Gyroscope)info.getContent();

if(gyroscope==null){

return;

}

Accelerationacceleration=gyroscope.getAcceleration();

Gesturegesture=newGesture();

if(acceleration.getX()<-10000){

gesture.setDesc("站立");

}elseif(acceleration.getY()<-10000){

gesture.setDesc("左侧");

}elseif(acceleration.getY()>10000){

gesture.setDesc("右侧");

}elseif(acceleration.getZ()<-10000){

gesture.setDesc("趴着睡");

}elseif(acceleration.getZ()>10000){

gesture.setDesc("仰卧");

}else{

gesture.setDesc("正常");

}

if(preGesture!=null&&

gesture.getDesc().equals(preGesture.getDesc())){

return;

}

preGesture=gesture;

Infoinfo1=newInfo(InfoType

.VALUE\_GESTURE,gesture);

helper.broadcast(info1);

helper.saveGesture(info1);

if(gesture.getDesc().equals("趴着睡")){

Stringcontent="小宝贝的睡姿不正确哦！";

helper.gestureError(newInfo(InfoType.NOTIFY\_GESTURE\_WRONG,content));

}

}

}

（9）图表显示

publicclassHeartRateActivityextendsBaseActivityimplementsViewportChangeListener{

privateLineChartViewchart;

privateLineChartDatadata;

privateArrayList<PointValue>list;

privateToolbarmToolbar;

privatestaticfinalintPOINT\_NUM=50;

@Override

protectedvoidinit(BundlesavedInstanceState){

setContentView(R.layout.activity\_heart\_rate);

}

@Override

protectedvoidinitViews(){

mToolbar=(Toolbar)findViewById(R.id.id\_toolbar);

setSupportActionBar(mToolbar);

getSupportActionBar().setTitle("心率折线图");

getSupportActionBar().setHomeButtonEnabled(true);

getSupportActionBar().setDisplayHomeAsUpEnabled(true);

chart=(LineChartView)findViewById(R.id.chart);

list=newArrayList<>(POINT\_NUM);

for(inti=0;i<POINT\_NUM;i++){

list.add(newPointValue(i,0));

}

Lineline=newLine(list);

line.setColor(ChartUtils.COLOR\_VIOLET);

line.setHasPoints(false);

line.setCubic(true);

List<Line>lines=newArrayList<>();

lines.add(line);

data=newLineChartData(lines);

data.setAxisYLeft(newAxis().setHasLines(true));

chart.setLineChartData(data);

chart.setZoomEnabled(false);

chart.setScrollEnabled(false);

}

@Override

protectedvoidinitEvents(){

mToolbar.setNavigationOnClickListener(newView.OnClickListener(){

@Override

publicvoidonClick(Viewv){

finish();

}

});

}

@Override

publicvoidonViewportChanged(Viewportviewport){

chart.setCurrentViewport(viewport);

}

privatevoidupdateData(HeartRatecontent){

for(inti=0;i<list.size()-1;i++){

floatnextValue=list.get(i+1).getY();

list.get(i).set(i,nextValue);

}

intposition=list.size()-1;

floatvalue=content==null?0:(float)content.getValue();

list.get(position).set(position,value);

Lineline=newLine(list);

line.setColor(ChartUtils.COLOR\_VIOLET);

line.setHasPoints(false);

line.setCubic(true);

List<Line>lines=newArrayList<>();

lines.add(line);

data=newLineChartData(lines);

data.setAxisYLeft(newAxis().setHasLines(true));

chart.setLineChartData(data);

}

@Override

protectedvoidonMsgReceived(Infomsg){

if(msg.getType()==InfoType.VALUE\_HEART\_RATE){

updateData((HeartRate)msg.getContent());

}

}

}

# 4.Android应用及Web服务平台

## 4.1功能说明

（1）用户登录，通过账户控制用户的权限

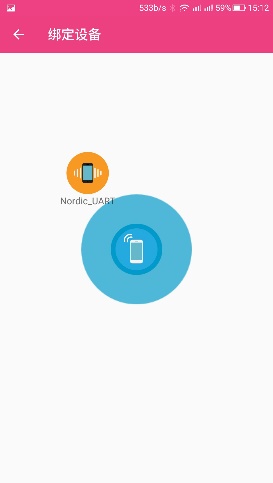


图23app登录界面和连接蓝牙界面

（2）利用Android手机蓝牙功能寻找附近设备并为该账户绑定脚环设备

（3）通过蓝牙连接到脚环，接收数据并进行显示，同时将数据上传到服务器

（4）对宝宝的心率，体温，睡姿进行监测，发现异常及时预警



图24 app功能演示界面

（5）提供宝宝近一周的体温、心率数据，家长可以随时进行查看。

（6）提供宝宝近一周的睡姿记录，家长可以查看宝宝在具体时段的睡姿，或者通过扇形图查看宝宝一天内各个姿态的分布，总时长，以及占比。

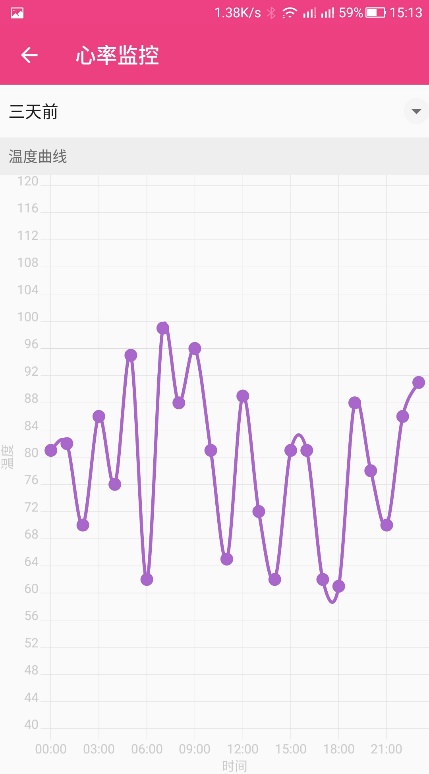
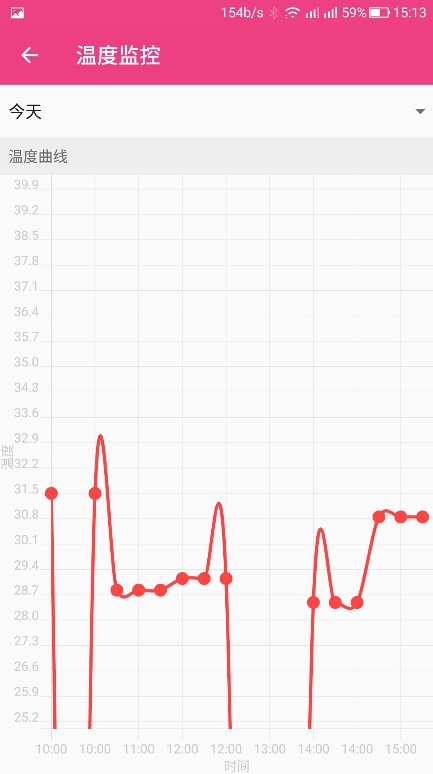


图25 app心率检测功能

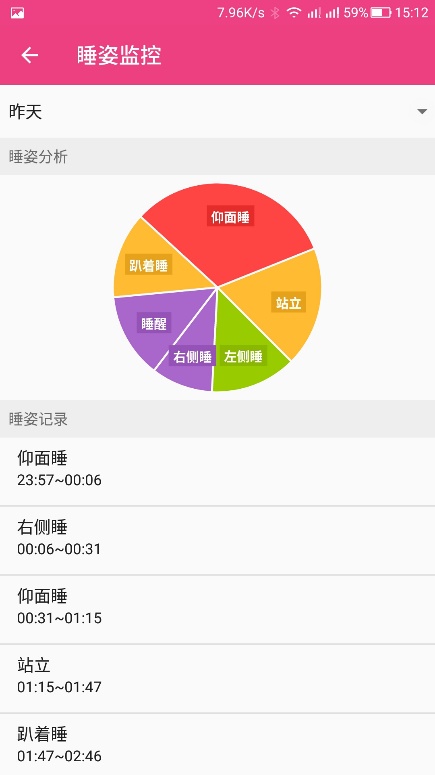


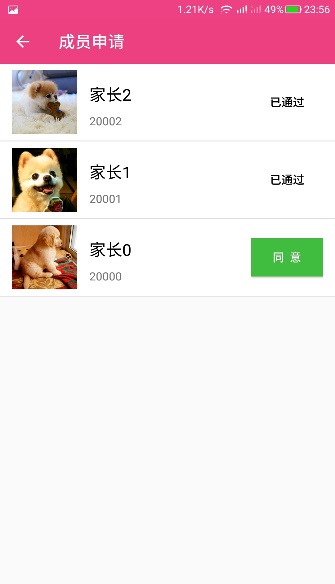
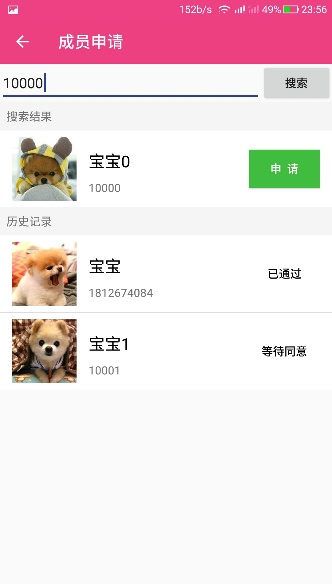
图26 app睡姿统计

（7）不定期的为家长推送各类育儿知识，指导家长正确的照顾好宝宝。



图27 app育儿文章推送

（8）支持家长账号绑定宝宝账号，多个家长共同为宝宝的健康成长保驾护航。只需用家长账号登录，点击添加宝宝账号，输入宝宝账号，进行搜索，点击搜索，只需宝宝账号同意即可查看宝宝数据。



（9）用户也可以对软件的一些功能进行系统设置，使之更符合用户需求。

（10）可以向我们提交意见反馈，我们会及时处理，如有应用更新，用户可在指定页面进行软件更新。



## 4.2软件设计实现

（1）使用GATT协议实现脚环与Android手机通信

GenericAttributeProfile（GATT）—GATT配置文件是一个通用规范，用于在BLE链路上发送和接收被称为“属性”的数据块。目前所有的BLE应用都基于GATT。蓝牙SIG规定了许多低功耗设备的配置文件。配置文件是设备如何在特定的应用程序中工作的规格说明。注意一个设备可以实现多个配置文件。例如，一个设备可能包括心率监测仪和电量检测。

（2）使用HTTP协议实现Android与服务器通信

超文本传输协议（HTTP，HyperTextTransferProtocol)是互联网上应用最为广泛的一种网络协议。所有的WWW文件都必须遵守这个标准。设计HTTP最初的目的是为了提供一种发布和接收HTML页面的方法。1960年美国人TedNelson构思了一种通过计算机处理文本信息的方法，并称之为超文本（hypertext）,这成为了HTTP超文本传输协议标准架构的发展根基。TedNelson组织协调万维网协会（WorldWideWebConsortium）和互联网工程工作小组（InternetEngineeringTaskForce）共同合作研究，最终发布了一系列的RFC，其中著名的RFC2616定义了HTTP1.1。

（3）使用MySQL实现服务器端数据存储

MySQL是一个关系型数据库管理系统，由瑞典MySQLAB公司开发，目前属于Oracle旗下产品。MySQL是最流行的关系型数据库管理系统之一，在WEB应用方面，MySQL是最好的RDBMS(RelationalDatabaseManagementSystem，关系数据库管理系统)应用软件。

MySQL是一种关系数据库管理系统，关系数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。

MySQL所使用的SQL语言是用于访问数据库的最常用标准化语言。MySQL软件采用了双授权政策，分为社区版和商业版，由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，一般中小型网站的开发都选择MySQL作为网站数据库。

（4）使用SSM框架进行服务器的架构

SSM（Spring+SpringMVC+MyBatis）框架集由Spring、SpringMVC、MyBatis三个开源框架整合而成，常作为数据源较简单的web项目的框架。

其中spring是一个轻量级的控制反转（IoC）和面向切面（AOP）的容器框架。

SpringMVC分离了控制器、模型对象、分派器以及处理程序对象的角色，这种分离让它们更容易进行定制。

MyBatis是一个支持普通SQL查询，存储过程和高级映射的优秀持久层框架。

（5）使用JPush向Android推送最新的用户数据。

JPush是经过考验的大规模APP推送平台，每天推送消息数超过5亿条。开发者集成SDK后，可以通过调用API推送消息。同时，JPush提供可视化的web端控制台发送通知，统计分析推送效果。JPush全面支持Android,iOS,Winphone三大手机平台。

4.3数据库设计



表2存储用户数据的users表

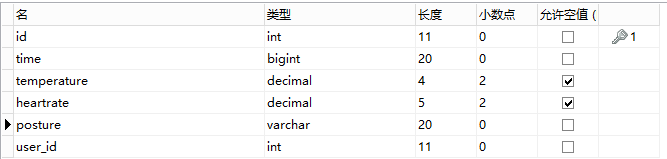


表3: 存储宝宝账号上传的原始传感器数据的temp\_data表

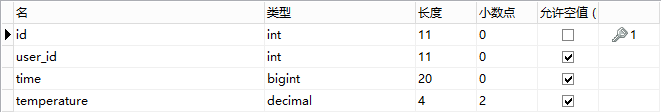


表4存储经过分析之后的宝宝温度记录temperature表

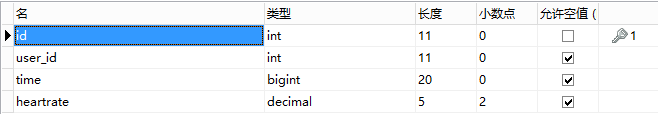


表5存储经过分析之后的宝宝心率记录heart\_rate表



表6存储经过分析之后的宝宝心率记录heart\_rate表



表7存储宝宝账号与家长账号的关系的relation表

4.4关键部分源码

（1）权限申请

<!--Required一些系统要求的权限，如访问网络等-->  
<uses-permissionandroid:name="android.permission.BLUETOOTH"/>  
<uses-permissionandroid:name="android.permission.BLUETOOTH\_ADMIN"/>  
<uses-permissionandroid:name="com.wdtx.littlebell.permission.JPUSH\_MESSAGE"/>  
<uses-permissionandroid:name="android.permission.RECEIVE\_USER\_PRESENT"/>  
<uses-permissionandroid:name="android.permission.INTERNET"/>  
<uses-permissionandroid:name="android.permission.WAKE\_LOCK"/>  
<uses-permissionandroid:name="android.permission.READ\_PHONE\_STATE"/>  
<uses-permissionandroid:name="android.permission.WRITE\_EXTERNAL\_STORAGE"/>  
<uses-permissionandroid:name="android.permission.READ\_EXTERNAL\_STORAGE"/>  
<uses-permissionandroid:name="android.permission.WRITE\_SETTINGS"/>  
<uses-permissionandroid:name="android.permission.VIBRATE"/>  
<uses-permissionandroid:name="android.permission.MOUNT\_UNMOUNT\_FILESYSTEMS"/>  
<uses-permissionandroid:name="android.permission.ACCESS\_NETWORK\_STATE"/>  
<uses-permissionandroid:name="android.permission.ACCESS\_WIFI\_STATE"/>  
  
<!--Optionalforlocation-->  
<uses-permissionandroid:name="android.permission.SYSTEM\_ALERT\_WINDOW"/><!--用于开启debug版本的应用在6.0系统上层叠窗口权限-->  
<uses-permissionandroid:name="android.permission.ACCESS\_COARSE\_LOCATION"/>  
<uses-permissionandroid:name="android.permission.CHANGE\_WIFI\_STATE"/>  
<uses-permissionandroid:name="android.permission.ACCESS\_FINE\_LOCATION"/>  
<uses-permissionandroid:name="android.permission.ACCESS\_LOCATION\_EXTRA\_COMMANDS"/>  
<uses-permissionandroid:name="android.permission.CHANGE\_NETWORK\_STATE"/>  
<uses-permissionandroid:name="android.permission.GET\_TASKS"/>

（2）设置BLE

finalBluetoothManagerbluetoothManager=  
(BluetoothManager)getSystemService(Context.BLUETOOTH\_SERVICE);  
BluetoothAdaptermBluetoothAdapter=bluetoothManager.getAdapter();  
//确保蓝牙在设备上可以开启  
if(mBluetoothAdapter==null||!mBluetoothAdapter.isEnabled()){  
IntentenableBtIntent=newIntent(BluetoothAdapter.ACTION\_REQUEST\_ENABLE);  
startActivityForResult(enableBtIntent,1);

（3）发现BLE设备

BluetoothAdapter.getDefaultAdapter().startLeScan(this);

BluetoothAdapter.getDefaultAdapter().stopLeScan(this);

@Override  
publicvoidonLeScan(BluetoothDevicedevice,intrssi,byte[]scanRecord){  
if(!foundDevices.contains(device)){  
foundDevices.add(device);  
updateUI();  
}  
}

（4）与服务器进行通信

publicclassVolleyUtil{  
privatestaticVolleyUtilmVolleyUtil;  
//volley请求队列  
privatestaticRequestQueuemRequestQueue;  
privateGsonmGson;  
privatestaticfinalStringBASE\_URL="http://192.168.191.1/api/v1";  
//连接超时时间  
privatestaticfinalintREQUEST\_TIMEOUT\_TIME=60\*1000;  
privatestaticfinalintSUCCESS\_CODE=100;  
publicVolleyUtil(){  
mGson=newGsonBuilder()  
.registerTypeAdapter(Date.class,newDateDeserializer())  
.setDateFormat(DateFormat.LONG)  
.create();  
mRequestQueue=Volley.newRequestQueue(MyApplication.instance);  
}  
  
publicstaticVolleyUtilgetInstance(){  
if(mVolleyUtil==null){  
synchronized(VolleyUtil.class){  
if(mVolleyUtil==null){  
mVolleyUtil=newVolleyUtil();  
}  
}  
}  
returnmVolleyUtil;  
}  
  
private<T>voidrequest(Stringurl,intmethod,finalMap<String,String>param,  
finalVolleyCallBack<T>volleyCallBack){  
Listener<T>listener=newListener<>(volleyCallBack);  
StringRequeststringRequest=newStringRequest(method,BASE\_URL+url,  
listener,listener){  
@Override  
protectedMap<String,String>getParams()throwsAuthFailureError{  
//请求参数  
returnparam;  
}  
};  
  
}  
  
public<T>voidrequestGet(Stringurl,finalMap<String,String>param,  
VolleyCallBack<T>volleyCallBack){  
Listener<T>listener=newListener<>(volleyCallBack);  
StringBuildersb=newStringBuilder(BASE\_URL+url);  
if(param!=null&&param.size()>0){  
sb.append("?");  
for(Map.Entry<String,String>entry:param.entrySet()){  
sb.append(entry.getKey())  
.append("=")  
.append(entry.getValue())  
.append("&");  
}  
sb.deleteCharAt(sb.length()-1);  
Log.i("wending","requestGet:"+sb.toString());  
}  
  
StringRequeststringRequest=newStringRequest(  
Request.Method.GET,sb.toString(),listener,listener);  
addToQueue(stringRequest,url);  
}  
  
public<T>voidrequestPost(Stringurl,finalMap<String,String>param,  
VolleyCallBack<T>volleyCallBack){  
Listener<T>listener=newListener<>(volleyCallBack);  
StringRequeststringRequest=newStringRequest(  
Request.Method.POST,BASE\_URL+url,listener,listener){  
@Override  
protectedMap<String,String>getParams()throwsAuthFailureError{  
returnparam;  
}  
};  
addToQueue(stringRequest,url);  
}  
  
publicstaticRequestQueuegetRequestQueue(){  
returnmRequestQueue;  
}  
  
publicvoidcancel(Stringurl){  
mRequestQueue.cancelAll(url);  
}  
  
privateclassListener<T>implementsResponse.Listener<String>,Response.ErrorListener{  
  
VolleyCallBack<T>volleyCallBack;  
  
publicListener(VolleyCallBack<T>volleyCallBack){  
this.volleyCallBack=volleyCallBack;  
}  
  
@Override  
publicvoidonResponse(Strings){  
if(volleyCallBack==null){  
return;  
}  
Typetype=getTType(volleyCallBack.getClass());  
HttpResulthttpResult=mGson.fromJson(s,HttpResult.class);  
if(httpResult!=null){  
//失败  
if(httpResult.getCode()!=SUCCESS\_CODE){  
volleyCallBack.onFail(  
httpResult.getCode(),  
httpResult.getMessage());  
}else{//成功  
//获取data对应的json字符串  
Stringjson=mGson.toJson(httpResult.getContent());  
//泛型是String，返回结果json字符串  
if(type==String.class){  
volleyCallBack.onSuccess((T)json);  
//泛型是实体或者List<>  
}else{  
Tt=mGson.fromJson(json,type);  
volleyCallBack.onSuccess(t);  
}  
}  
}  
}  
@Override  
publicvoidonErrorResponse(VolleyErrorvolleyError){  
if(volleyCallBack==null){  
return;  
}  
volleyCallBack.onError(volleyError);  
}  
}  
privateTypegetTType(Class<?>clazz){  
TypemySuperClassType=null;  
Type[]genericInterfaces=clazz.getGenericInterfaces();  
for(TypegenericInterface:genericInterfaces){  
if(genericInterfaceinstanceofParameterizedType){  
ParameterizedTypeparamType=(ParameterizedType)genericInterface;  
if(paramType.getRawType().equals(VolleyCallBack.class)){  
Type[]types=paramType.getActualTypeArguments();  
if(types!=null&&types.length>0){  
returntypes[0];  
}  
}  
}  
}  
returnnull;  
}  
privatevoidaddToQueue(StringRequeststringRequest,Stringurl){  
//设置请求超时和重试  
stringRequest.setRetryPolicy(newDefaultRetryPolicy(REQUEST\_TIMEOUT\_TIME,1,1.0f));  
//加入到请求队列  
if(mRequestQueue!=null){  
stringRequest.setTag(url);  
mRequestQueue.add(stringRequest);  
}  
}  
}

（5）连接到脚环设备

publicbooleanconnect(){  
Stringaddress=getAddress();  
intconnState=getConnState();  
  
if(mBluetoothAdapter==null||address==null){  
returnfalse;  
}  
  
if(currentDevice!=null&&address.equals(currentDevice)&&  
mBluetoothGatt!=null&&connState==BluetoothProfile.STATE\_DISCONNECTED){  
returnmBluetoothGatt.connect();  
}  
  
currentDevice=address;  
BluetoothDevicedevice=mBluetoothAdapter.getRemoteDevice(address);  
mBluetoothGatt=device.connectGatt(BleService.this,false,mCallBack);  
returntrue;  
}

（6）监听Ble蓝牙的反馈

privateclassMyCallBackextendsBluetoothGattCallback{  
@Override  
publicvoidonCharacteristicChanged(BluetoothGattgatt,  
BluetoothGattCharacteristiccharacteristic){  
setData(newBlueData(characteristic.getValue()));  
}  
  
@Override  
publicvoidonConnectionStateChange(BluetoothGattgatt,intstatus,intnewState){  
if(status!=BluetoothGatt.GATT\_SUCCESS){  
return;  
}  
  
setConnState(newState);  
  
//连接成功后马上搜索服务  
if(newState==BluetoothProfile.STATE\_CONNECTED){  
mBluetoothGatt.discoverServices();  
}  
}  
  
@Override  
publicvoidonServicesDiscovered(BluetoothGattgatt,intstatus){  
//获得目标BluetoothGattService  
if(status!=BluetoothGatt.GATT\_SUCCESS){  
return;  
}  
BluetoothGattServiceservice=gatt.getService(UUID\_TARGET\_SERVICE);  
  
//获得目标BluetoothGattCharacteristic  
if(service==null){  
return;  
}  
BluetoothGattCharacteristiccharacteristic=service  
.getCharacteristic(UUID\_TARGET\_CHARACTERISTIC);  
  
//设置BluetoothGattCharacteristic通知  
if(characteristic==null){  
return;  
}  
mBluetoothGatt.setCharacteristicNotification(characteristic,true);  
BluetoothGattDescriptordescriptor=characteristic.getDescriptor(  
UUID\_CLIENT\_CHARACTERISTIC\_CONFIG);  
descriptor.setValue(BluetoothGattDescriptor.ENABLE\_NOTIFICATION\_VALUE);  
mBluetoothGatt.writeDescriptor(descriptor);  
}  
}

# 5.作品运行说明

## 5.1嵌入式系统软件的烧录

物联网婴幼儿睡眠监测脚环的嵌入式软件采用KEILC平台开发，实现蓝牙4.0协议和运动数据、障碍数据的采集和传送，烧录步骤为：

（1）可穿戴节点协议栈

由于nRF52832为Cortex-M4的32位ARM内核，因此嵌入式系统采用KEIL开发协议栈（SoftDevice）和应用软件完全分开，因此需要用nRFgoStudio烧入协议栈。步骤如下：

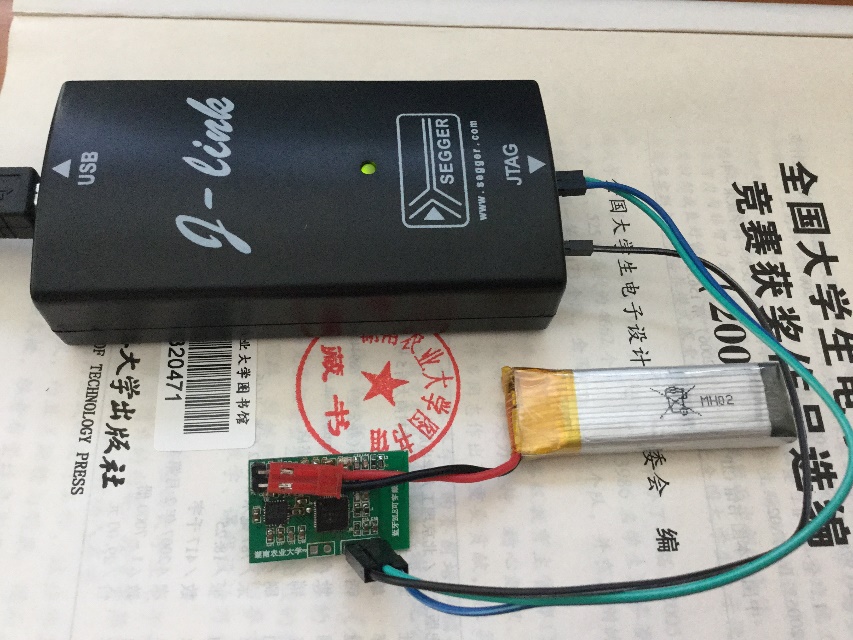


图28脚环主控连接示意图

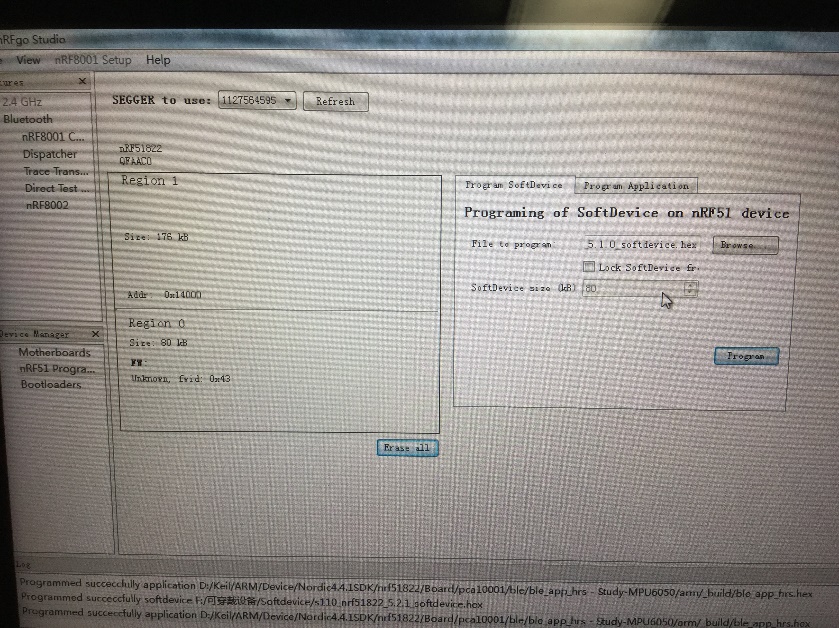


图29 刷开发协议栈（SoftDevice）

（2）脚环嵌入式软件下载

烧录Nrf52832的协议栈后，我们将嵌入式程序通过keil下载到可穿戴节点ROM的0x18000起始的区域，下载如图所示。

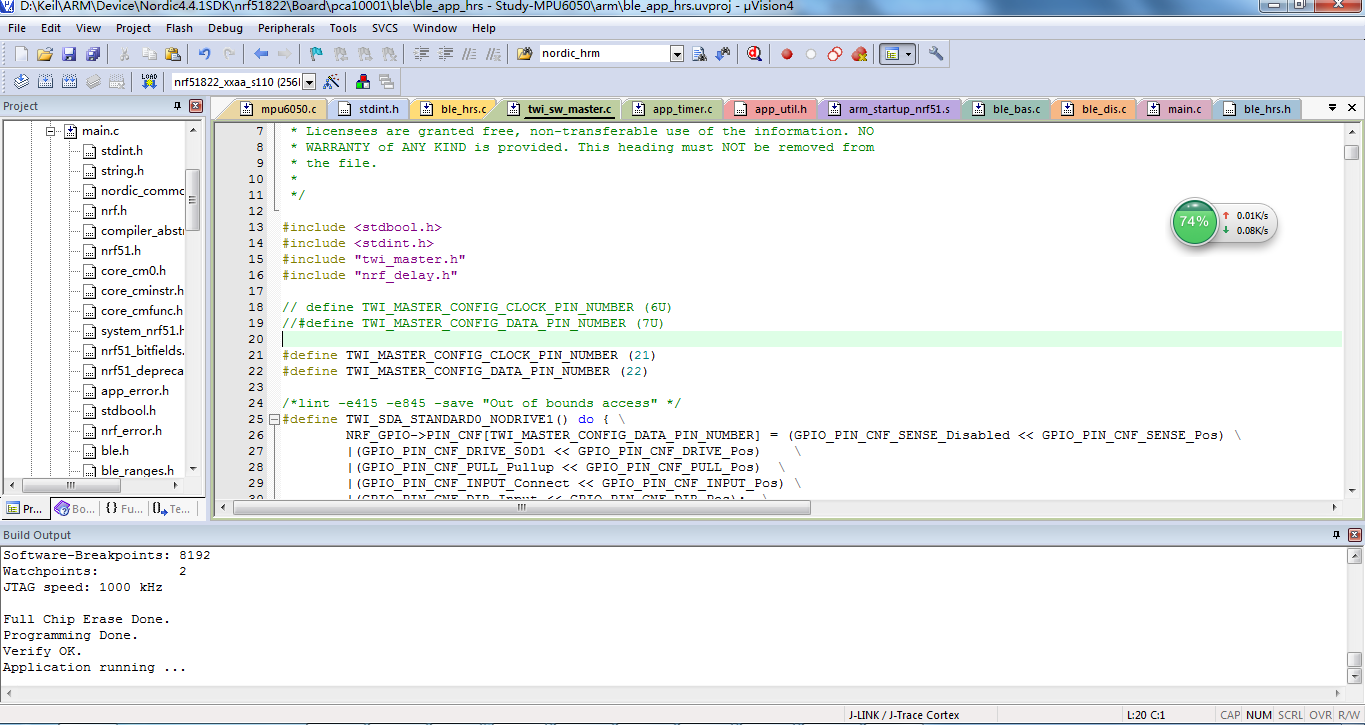


图30 嵌入式程序下载示意图

至此，可穿戴节点的硬件平台已经搭建后，插上锂电池，此时，可穿戴节点开始工作。

## 5.2安卓平台软件的运行

将APK软件安装至手机，手机硬件平台为蓝牙4.0的安卓平台手机，Android版本4.3以上。安装后，找到Android系统的“小铃铛”图标，如图所示。



图31 i-BabyAPP“小铃铛”应用图标

点击图标后，APP如图所示:



图31 app主界面图

点击下方图标，连接物联网婴幼儿睡眠监测脚环，可以进入到监测状态，监测睡姿如下：



图32宝宝姿态显示图

同时，系统可以实时分析心率图，当心率出现异常时预警。

# 6.总结

基于物联网的可穿戴式婴幼儿睡眠监测系统主要通过自主开发的可穿戴节点，通过MPU6050实现姿态采集，MAX30100实现心率温度采集，I2S数字音频总线实现了声音采集，通过集成72MARMCortex-M4F浮点计算功能的nRF52832多传感器数据融合计算，实现了宝宝睡眠智能检测，其记过通过蓝牙4.0无线发送到手机，实现宝宝睡眠智能检测，预警，使最新的可穿戴技术惠及所有的父母，具有创新性。

在作品开发过程中，基于蓝牙4.0的可穿戴设计的硬件和BLE通讯协议实现，开放的资料较少，开发起来有一定的困难，可穿戴节点和安卓软件的通讯实现较为复杂，姿态识别算法、心率检测滤波算法、数字麦克风数据传输处理算法，作品开发小组都进行深入的讨论分析，作品设计到硬件开发，焊接，调试，嵌入式软件实现，都使作品开发小组在本作品的开发过程中积累了较为丰富的经验。该作品站基于物联网的宝宝睡眠监测系统，作品方案新颖正确，有较大的创新价值和推广意义。