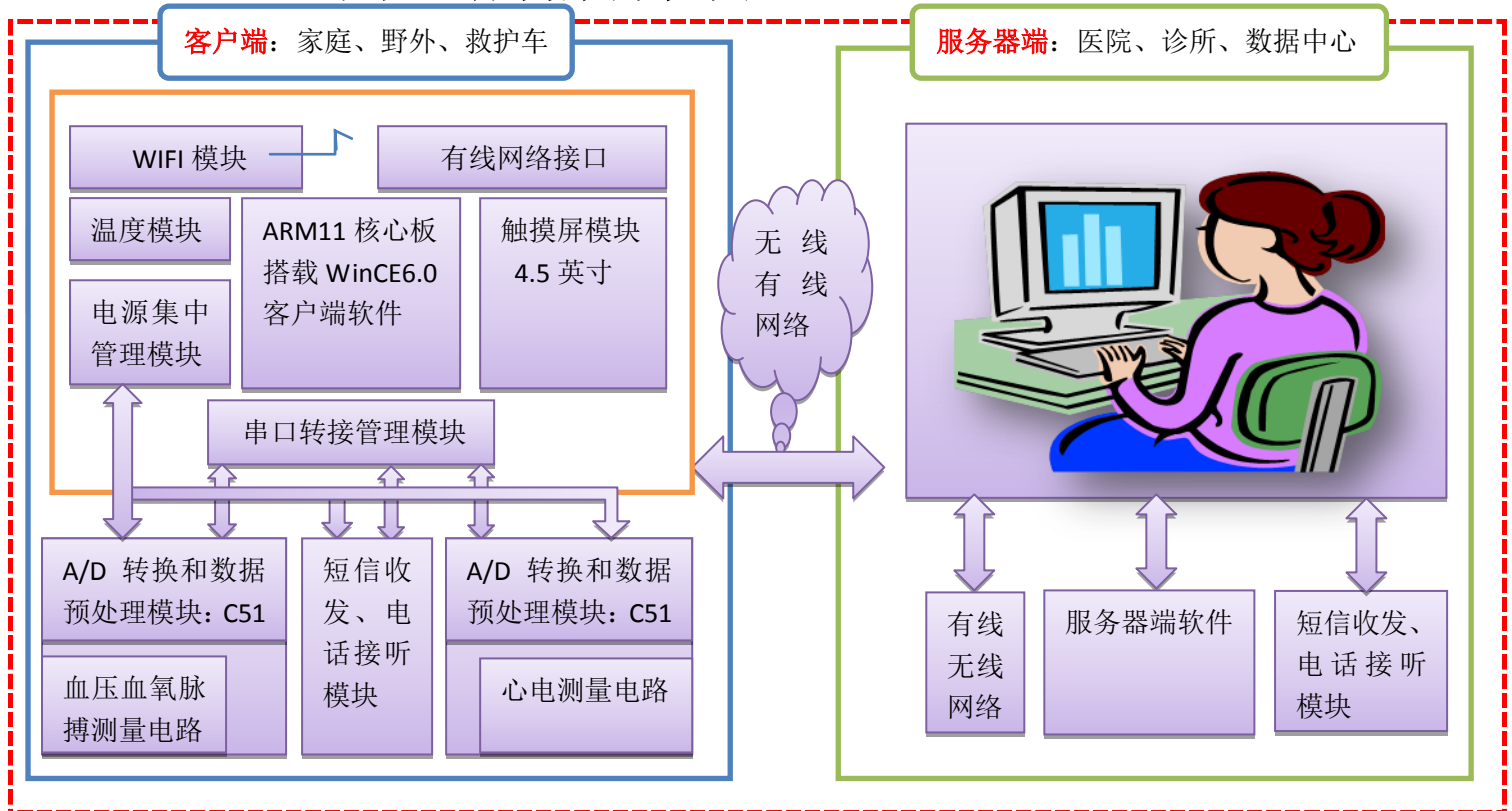


《便携式远程医疗监护系统设计方案》

说明：写这个开发文档只为了理解和阅读方便，我会根据我们亲历的整个创作过程书写，所以难免会不同于标准的开发文档格式，望见谅。

一．系统整体构架方框图



二．硬件部分原理与电路分模块介绍

1.心电数据测量模块

● 心电模块理论原理：

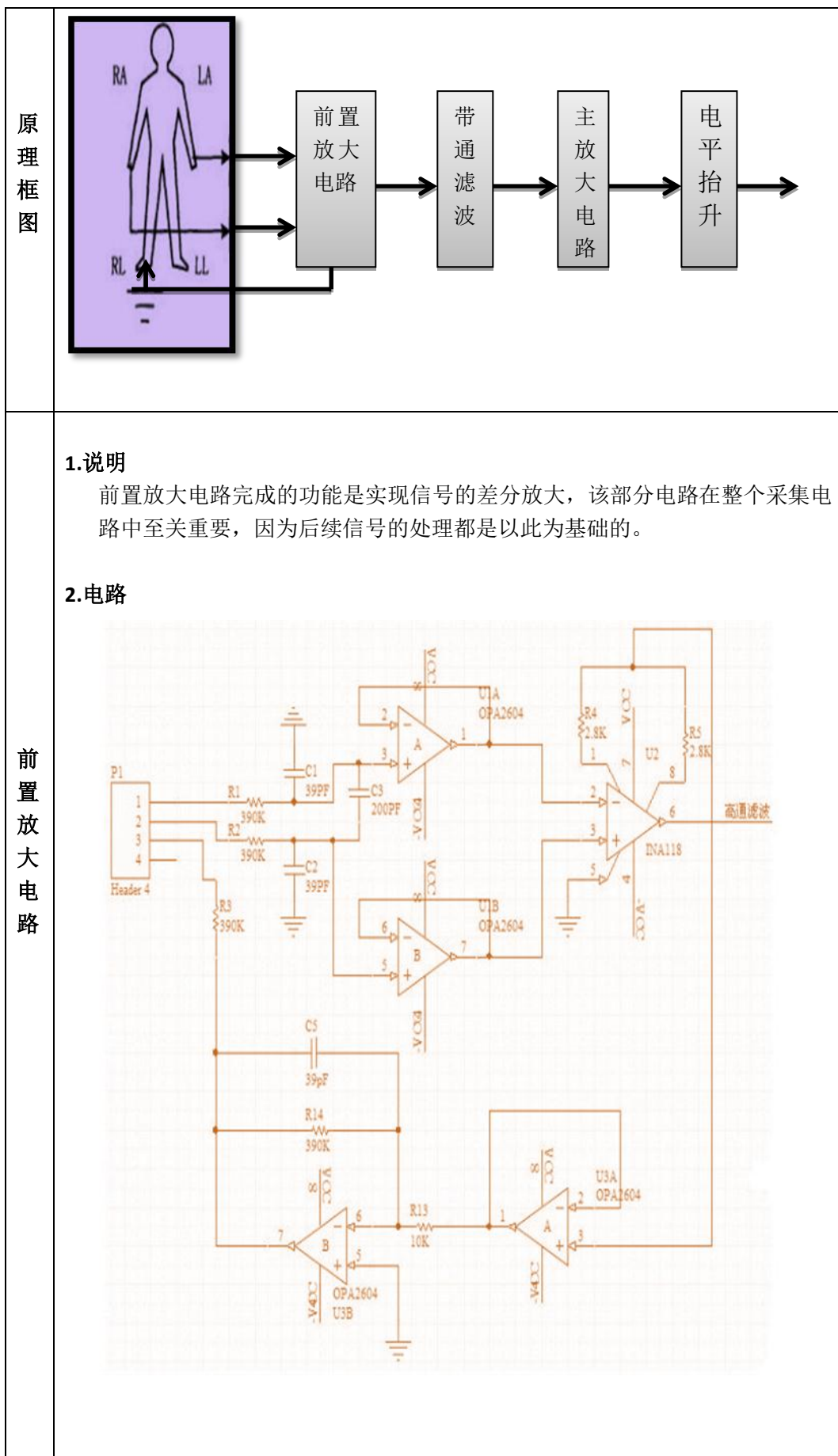
心脏能够有规律的发生兴奋和收缩，从而推动血液循环，在心在肌肉每次收缩之前，都会有一股微小的生物电流产生。由于人体的体液被曝光引起导电，这些微小的电流通过体液的传递就会反映到人体的表面皮肤上，由于身体各部分组织不同，距离心脏的距离不同，从而造成体表的不同部位的点位有所不同。

通过此原理，我们应用电极片将微小的体表电流传导到导联线中。其中导联线为 3 跟：左右手跟右腿，左右手传导电流形成差模信号，右腿传导电流主要传输给右腿驱动电路用来抑制共模信号。

由于差模输入电压在 1-5mv 之间，则需对信号进行放大，利用 INA118 将差分输入信号放大 10 倍，然后利用无源带通滤波器对信号进行滤波处理，再利用 OPA2604 放大器对金浩进行 100 倍放大，由于信号要进行 A/D 转换，且 TLC1543 输入电压要求为 5V，最后利用 OPA2604 对信号电压进行抬升。

然后通过 A/D 转换传给 51 单片机，经 51 单片机进行简单的数据封包和预处理操作后再通过串口传给上位机进行后续滤波、显示和远距离传送。

● 心电模块的原理框图以及电路图（实际应用中某些参量和线路略有改动）



带通滤波	<div data-bbox="355 197 1265 313"><div>1. 说明</div><div>对经前置放大后的信号进行模拟滤波处理，滤除带外的杂散频率成分。</div><div>2. 电路</div></div> <div data-bbox="375 347 1244 638"></div>
主放大电路	<div data-bbox="355 689 1332 884"><div>1. 说明</div><div>A/D 转换的输入电平要求为 0—3.3V，因此必须实现心电信号的高增益放大 800—1000 倍左右。前置放大电路放大了 10 倍，理论上主运放放大 100 倍左右即可。在本设计中采用两级放大，第一级放大 10 倍，第二级通过 RJ 调节放大倍数，可调节最佳的增益输出。</div><div>2. 电路</div></div> <div data-bbox="406 940 1252 1265"></div>
电平抬升电路	<div data-bbox="355 1294 1332 1489"><div>1. 说明</div><div>放大后的心电信号电压大概为-0.5V—1.5V，而 A/D 的输入范围 为 0—3.3V，因此需要把信号抬升，保证能采集到全部的心电信号。下图为差分输入放大电路，输入信号反向后与正输入端电压相加，正输入端的电压可以通过 P3 滑动变阻器进行调节，从而达到电平抬升的目的。</div><div>2. 电路</div></div> <div data-bbox="454 1545 1220 1937"></div>

2. 脉搏血压数据测量模块

● 脉搏血压模块理论原理：

由上位机控制 51 单片机，单片机控制电路中各个模块的工作情况。首先由 51 控制气泵向袖带内充气至一定压力值（如 180mmHg），确保超过收缩压，使血流阻断，然后控制气阀以 3-5mmHg 的速率阶梯式放弃。在放气过程中，压力传感器 MPX5050 将袖带内压力信号转换为电信号，电信号经过低通滤波器，得到袖带的静压力信号和十分微弱的脉搏信号。其中直流信号用以转换袖带静压力信号成血压值；另一路经过高通滤波器滤波，得到脉搏震荡信号后传给单片机用来计算脉率，当检测到收缩压、平均压和舒张压后，打开气阀，使袖带全部放气。

● 血压模块的原理说明以及电路图（实际应用中某些参量和线路略有改动）

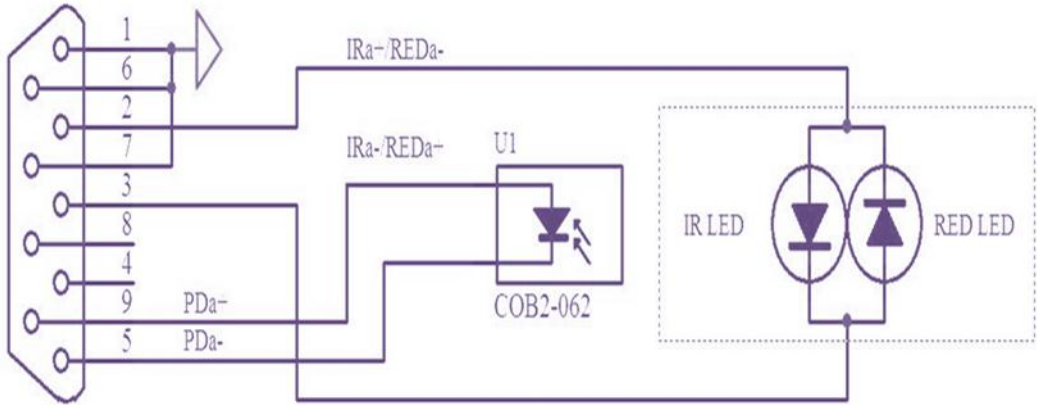
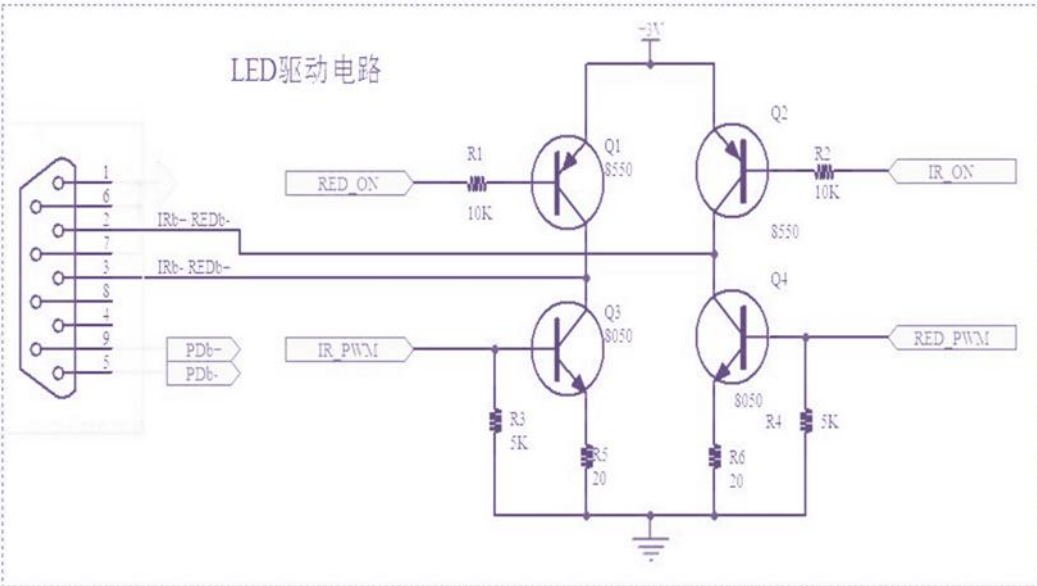
原理说明	<p>血压的测量范围一般是 0—200mmHg，我们选用硅压力传感器 MPX5050GP 实现对压力的测量。</p> <p>TLC2274-1 构成 40Hz 二阶低通滤波器，运放接成跟随器，放大倍数为 1。</p> <p>TLC2274-2 构成 0.4Hz 二阶高通滤波器，运放接成跟随器，放大倍数为 1。</p> <p>TLC2274-3 构成反相放大器，闭环放大倍数可以调节到 3.75 倍。</p> <p>TLC2274-4 构成加法器，用来对脉搏信号进行相位和基线的调整，电路采取反相放大接法，增益可达 40 倍</p>
电路原理图	

3.血氧饱和度测量

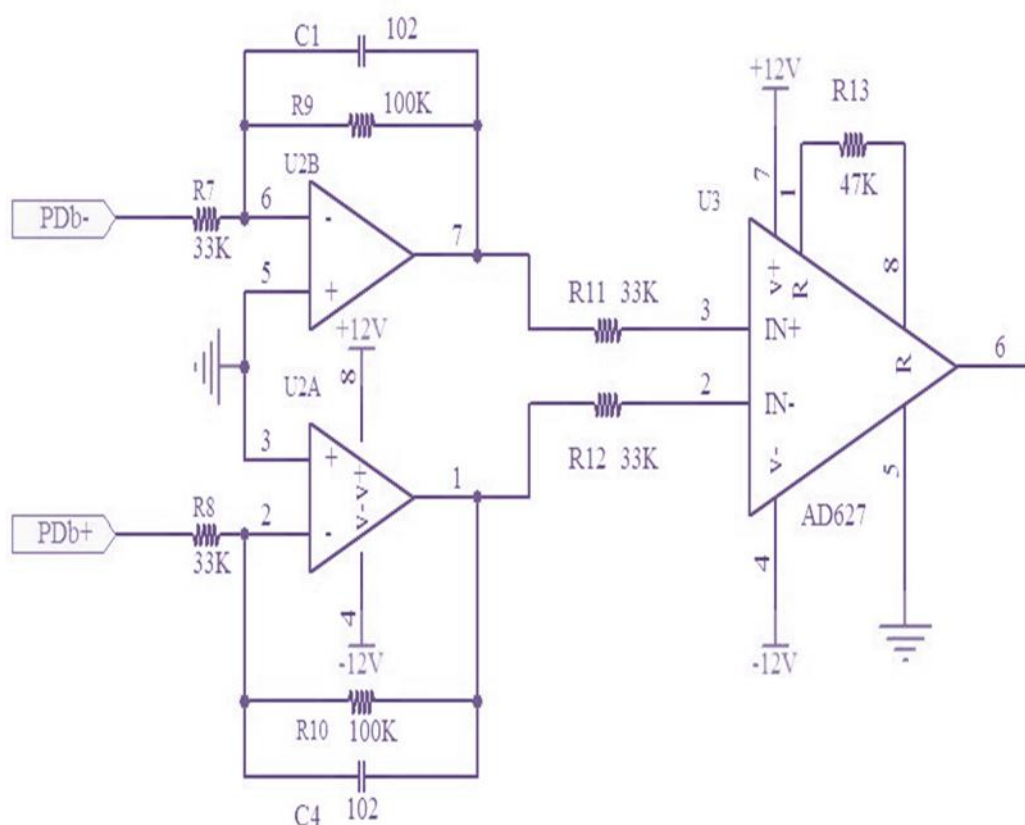
● 血氧饱和度测量原理

脉搏血氧模块包括探头电路、LED 驱动电路、放大采样点路以及数据接口。由主控制器生成脉冲方波来控制两个 LED 交替开关，再由光电二极管检测芯片检测光信号，实现光电转换。再将信号进行放大处理。

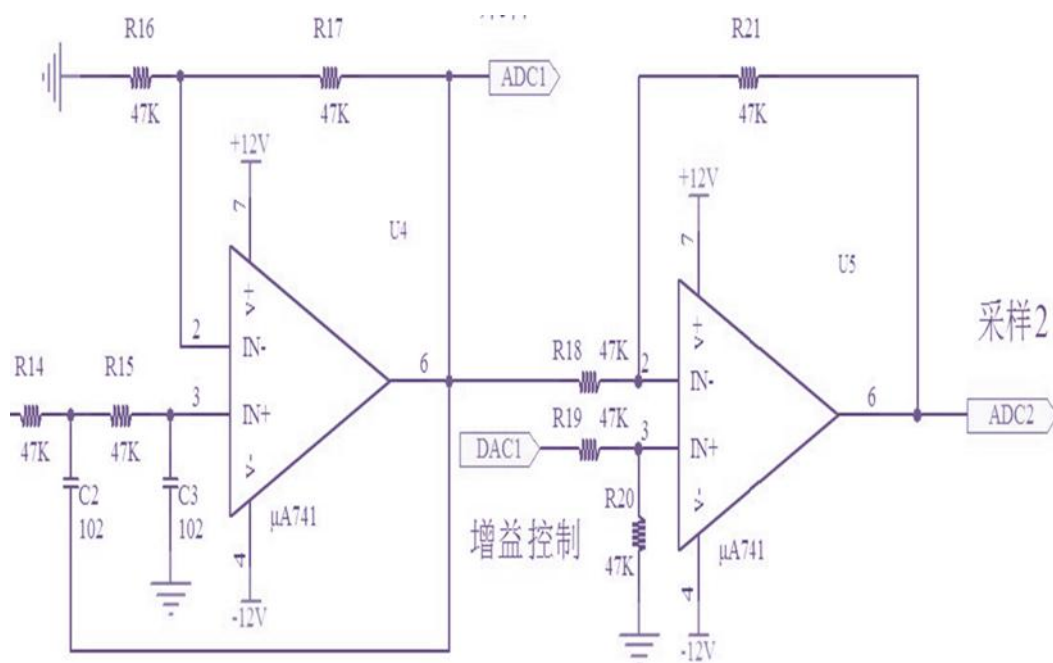
● 血氧饱和度测量原理说明及电路图（实际应用中某些参量和线路略有改动）

原理说明	<p>由主控制器生成脉冲方波来控制两个 LED 交替开关；</p> <p>再由光电二极管检测芯片检测光信号，实现光电转换；</p> <p>再将信号进行放大处理。</p>
血氧指夹探头电路	
LED 驱动电路	

差动放大器电路



二阶低通滤波器电路

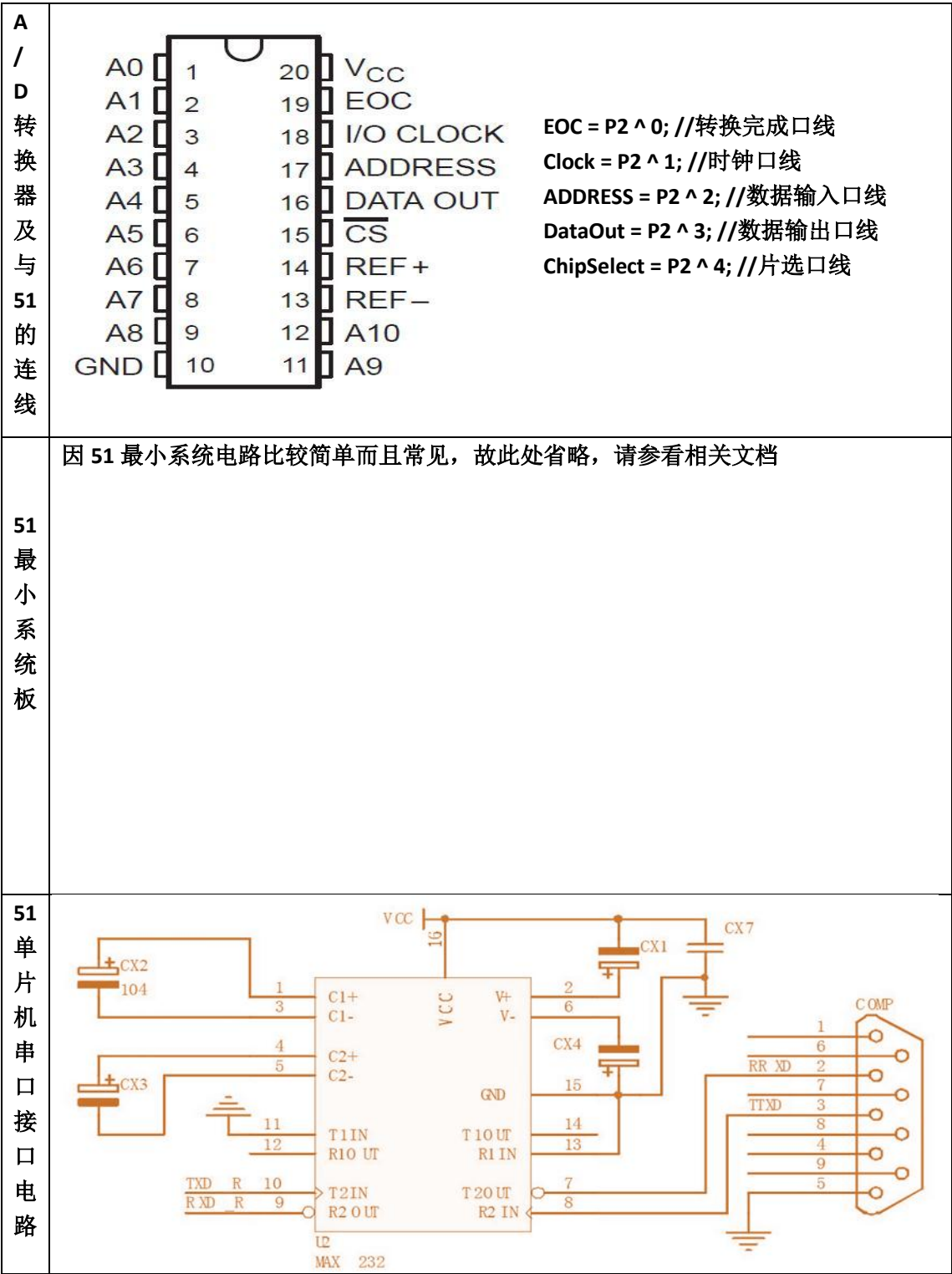


4.A/D 转换和数据预处理模块

● A/D 转换和数据预处理模块原理概述

利用 10bit A/D 转换器 TLC1543 对采集到的模拟信号进行 A/D 转换，由 51 单片机控制 A/D 转换器进行 A/D 转换，并将经由 A/D 转换后的得到的数字信号进行简单的滤波和分组分包处理，最后经由 51 单片机的串口传输给上位机，由上位机对数据进行进一步处理、显示和必要时通过网络向远端的服务器发送处理过的数据。

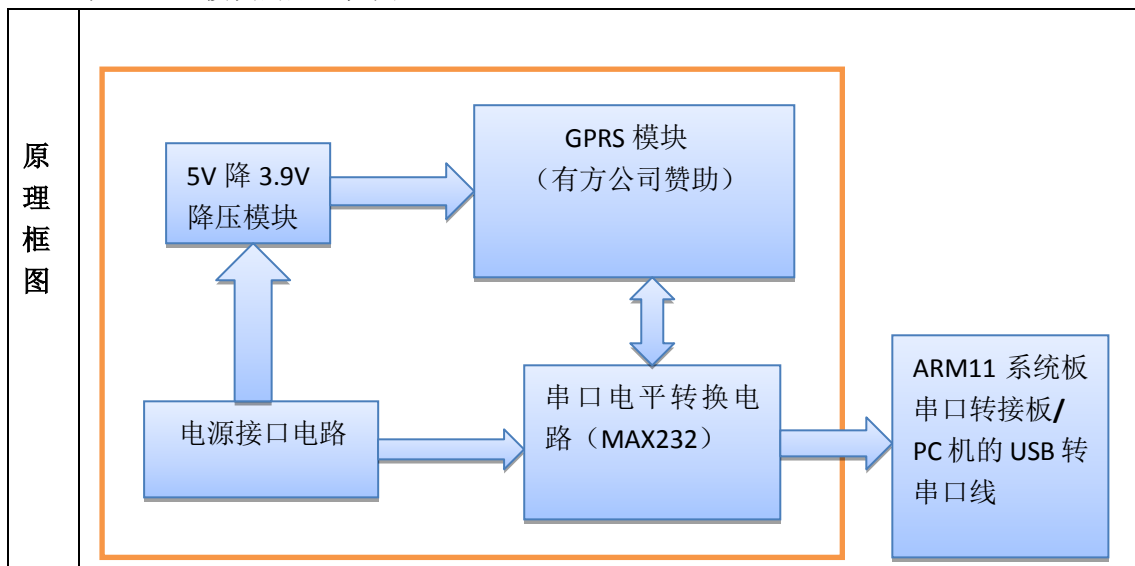
● A/D 转换和数据预处理模块电路图



5.串口短息发送接收 GSM 模块（语音通话功能可扩展）

ARM11 系统板上搭载 WinCE 操作系统，在系统中运行客户端软件，通过串口向 GSM 模块发送 AT 指令，并解析收到的回送指令，完成利用 GSM 模块收发短信以向远程服务器端请求其 IP 地址的功能，并可通过扩展，完成医患语音通话进行诊断的目的。

- 串口 GSM 模块的原理框图



二．软件部分代码组成与结构分析

说明：关于完整的源代码，您可在 www.pudn.com 网站上搜索，发布者 ansle. 或者，您可以在 CSDN 上搜索并下载，发布者 ansleliu.

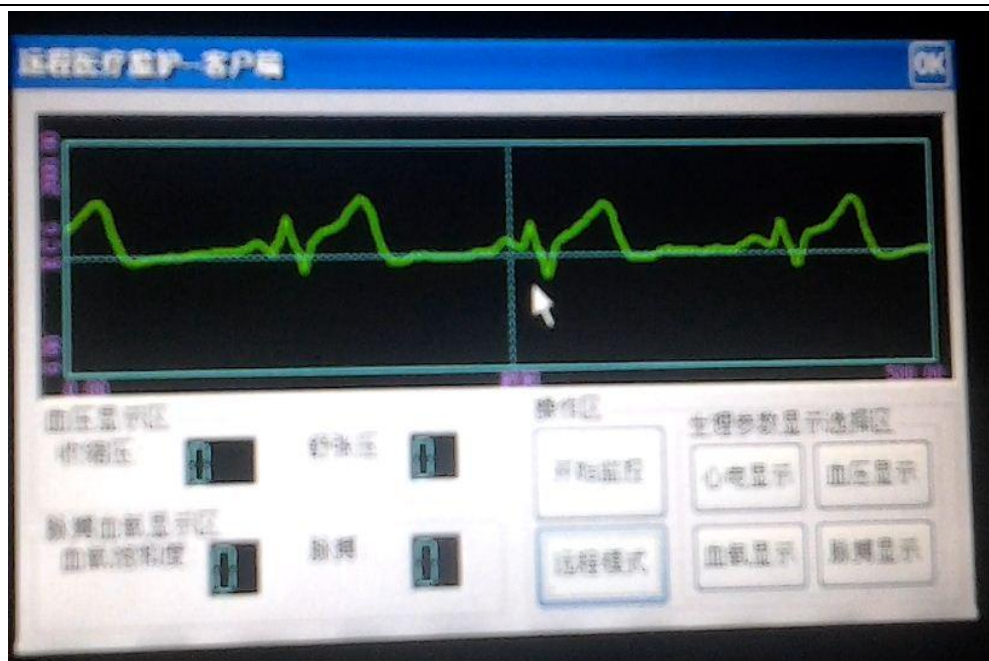
本工程源代码由 C/C++ 语言编写，使用操作系统为 Windows 操作系统（远程服务器端）和 Windows CE 6.0 操作系统（客户端），程序的开发工具为 VS2008

1.软件运行效果及其程序简化框图

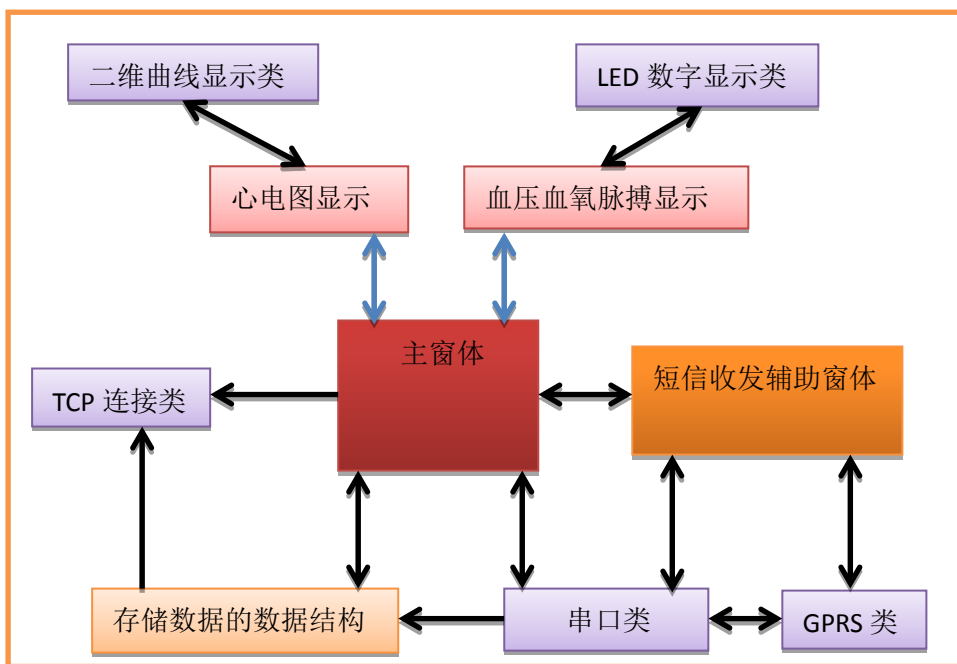
- 客户端软件



效果



程序简化流程图



二维曲线显示类完成心电图的显示；

LED 数字显示类完成血压脉搏血氧饱和度的显示；

GPRS 类完成短信的收发（获取远程服务器的 IP 地址）和电话的拨打接听；

串口类完成对下位机传来的数据的接收和使用 AT 指令对串口 GPRS 模块进行控制；

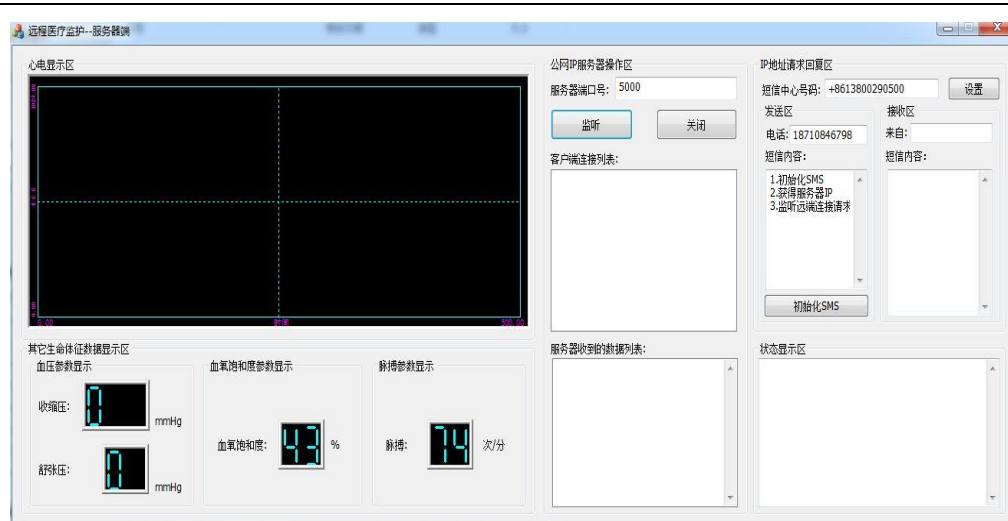
TCP 连接类完成与远端服务器进行 TCP 连接（传送数据）；

存储数据的数据结构以链表为基础，完成对串口接收到的数据进行存储，对滤波后的数据进行存储，

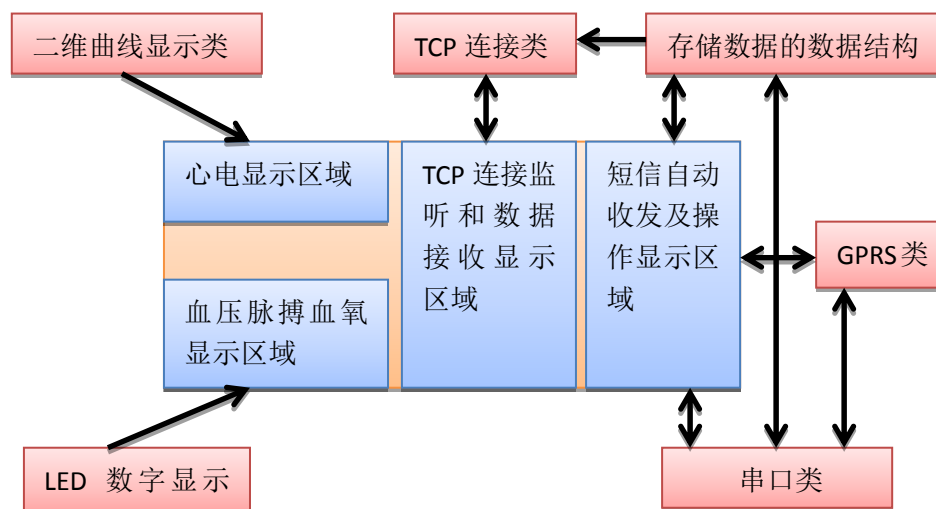
程序中采用多线程的手法以使程序可以并行运行各子功能，另外程序中还有简单的发送数据封包操作和简单的数字滤波操作。

- 远程服务器端软件

运行效果



程序简化流程图



附录:

● A/D 转换和数据预处理模块程序

[illegible]

```

●    TMOD=0x21;//定时器 1 工作方式 2 定时器 0 工作方式 1
●    SCON=0x50;//串口方式 1
●    TH0=(65536-5000)/256;//定时器 0 出装值
●    TL0=(65536-5000)%256;//
●    TH1=0xfd;//定时器波特率为 9600
●    TL1=0xfd;
●    TR1=1;//启动定时器 1
●    ET0=1;//开定时器 0 中断
●    TR0=1;//启动定时器 0
●    SM0=0;
●    SM1=1;//通信方式为 1
●    REN=1;//允许串行口接受数据
●    EA=1; //开总中断
●    ES=1; //开串口中断
● }
● //串口初始化结束
● /*接口总线定义结束*/
● //////////////////////////////////////
● unsigned int ADCSelChannel(unsigned char Channel)
● {
●     unsigned int ConvertValue;
●     unsigned char i, Chan;
●     unsigned char ConvertValueL, ConvertValueH;
●     unsigned char delay;
●
●     ConvertValueL = ConvertValueH = 0; //初始化转换结果
●     delay = 0;
●     if(EOC)
●     {
●         Clock = 0;
●         ChipSelect = 1;
●         Wait2us;
●         ChipSelect = 0;
●         Wait2us;
●         Channel = Channel << 4;
●         for (i = 0; i < 4; i++) //输入需要转换的通道的编码
●         {
●             Chan = Channel;
●             Chan = Chan >> 7;
●             DataIn = (bit)Chan;
●             Wait2us;
●             Clock = 1;
●             Clock = 0;
●             Channel = Channel << 1;

```

```

●      }
●      for (i = 0; i < 6; i++) //输入转换时钟
●      {
●          Clock = 1;
●          Clock = 0;
●      }
●      ChipSelect = 1;
●      //开始检测转换结束标志，或者转换超时出错
●      while ((!EOC) && (delay < 10))
●      {
●          Wait2us;
●          delay++;
●      }
●      if (delay == 10)
●      {
●          return (0xFFFF); //转换超时，返回错误代码
●      }
●      else
●      {
●          Wait10us;
●          Clock = 0;
●          ChipSelect = 1;
●          Wait1us;
●          ChipSelect = 0;
●          Wait1us;
●          for (i = 0; i < 2; i++) //读取高二位 bit 值
●          {
●              Clock = 1;
●              DataOut = 1;
●              ConvertValueH <<= 1;
●              if (DataOut)
●                  ConvertValueH |= 0x1;
●              Clock = 0;
●              Wait1us;
●          }
●          for (i = 0; i < 8; i++) //读取低八位 bit 值
●          {
●              Clock = 1;
●              DataOut = 1;
●              ConvertValueL <<= 1;
●              if (DataOut)
●                  ConvertValueL |= 0x1;
●              Clock = 0;
●              Wait1us;

```



```

    }
    ChipSelect=1;
    ConvertValue = ConvertValueH;
    ConvertValue <= 8;
    ConvertValue |= ConvertValueL;
    return ConvertValue; //返回转换结果
}
}
}
////////////////////////////////////
/**//数字滤波
#define N 12

int filter()
{
    int count;
    int sum = 0;
    for ( count=0;count<N;count++)
    {
        sum += ADCSelChannel(4);
        delay();
    }
    return (sum/N);
}

//数字率比结束
*/
// 要选择的模拟信号输入函数结束
//主函数
void main()
{
    init();
    while(1)
    {
        if(flag_uart==1)//如果接收到上位机信息
        {
            flag_uart=0;
            ES=0;
            TI=1;
            switch(flag_on)//选择上位机发送信息选项
            {
                case 0:puts("Turn on ad!\n");
                    TR0=1;
                    break;

```

```

●          case 1:printf("Turn off ad!\n");
●              TR0=0;
●              break;
●          case 2:puts("Error!\n");
●              break;
●      }
●      while(!TI);
●      TI=0;
●      ES=1;
●  }
●  if(flag_time==1)
●  {
●      flag_time=0;
●      ad_val=ADCSelChannel(4);
●      ad_vo=ad_val;
●      ES=0;
●      TI=1;
●      printf("%d,",ad_vo);
●      while(!TI);
●      TI=0;
●      ES=1;
●  }
●  }
●  }
●  //////////////////////////////////////
●  //定时器 T0 中断
●  void timer0() interrupt 1
●  {
●      TH0=(65536-5000)/256;
●      TL0=(65536-5000)%256;
●      // t0_num++;
●      // if(t0_num==20)
●      // {
●      //     t0_num=0;
●      //     flag_time=1;
●      // }
●  }
●  //串口中断
●  void ser()interrupt 4
●  {
●      RI=0;
●      a=SBUF;
●      flag_uart=1;
●  }

```

```
● switch(a)
● {
●     case 'a':flag_on=0;
●         break;
●     case 'e':flag_on=1;
●         break;
●     default:flag_on=2;
●         break;
● }
● }
```