上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

项目报告

PROJECT REPORT

题目： 基于MSP430的心电采集显示心率技术系统

成员：张露雨 郭焘玮 王润洲

目录

基于 MSP430 的心电采集显示心率计数系统

1. 实验目的

1.采用 AD8232 心电图监测生理身体指标测量板脉搏跳动心脏传感器创客模块
2.通过 MSP430 内部 ADC 模块采集 AD8232 输出的心电放大信号；
3.在 MSP430 的点阵 LCD 上显示心电波形，并计算心率；
4.通过 UART-RS232-USB 接口将心电信号传输到 PC 端，显示原始数据或者波形；

2. 实验内容

2.1 系统功能架构设计

由 AD8232 读取心电数据，处理后形成模拟信号输入实验板，MSP430 使用自带 ADC 转换为数字信号，经过软件处理后显示在 TFT 屏幕上，同时通过串口输出到电脑端。

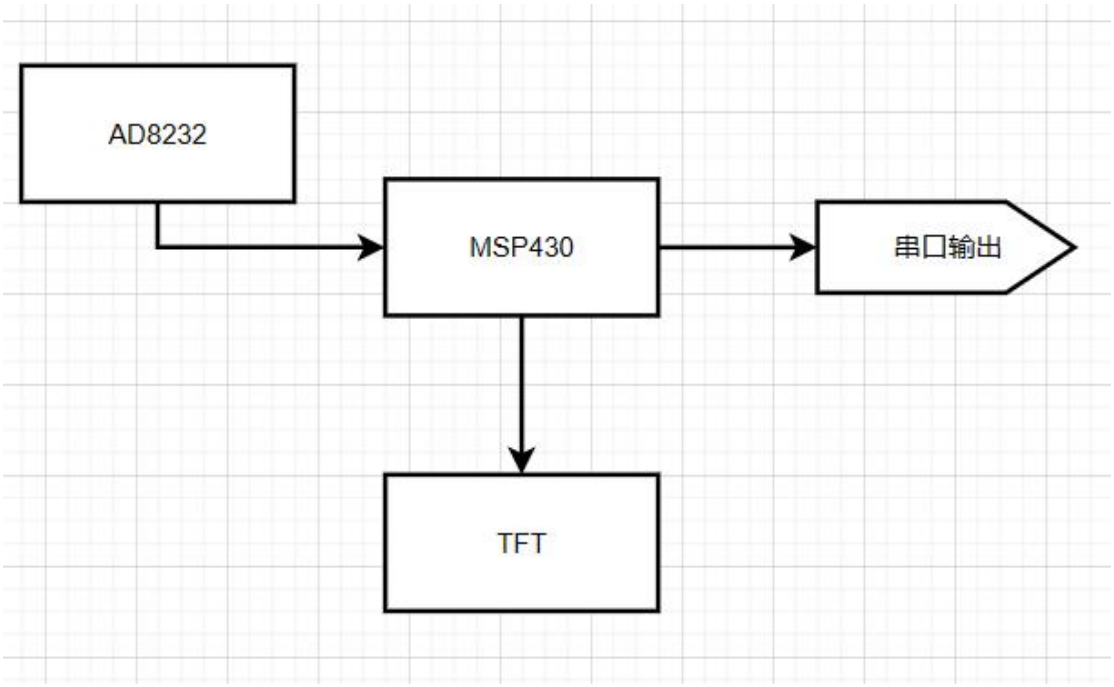


图 2.1 系统框图

2.2 硬件接口

接线引脚：选择 adc 通道 12= p7.4 引脚， 连接心电芯片 output 引脚（黄线），心电芯片 3.3v 和 gnd 分别接 vcc 和 gnd。

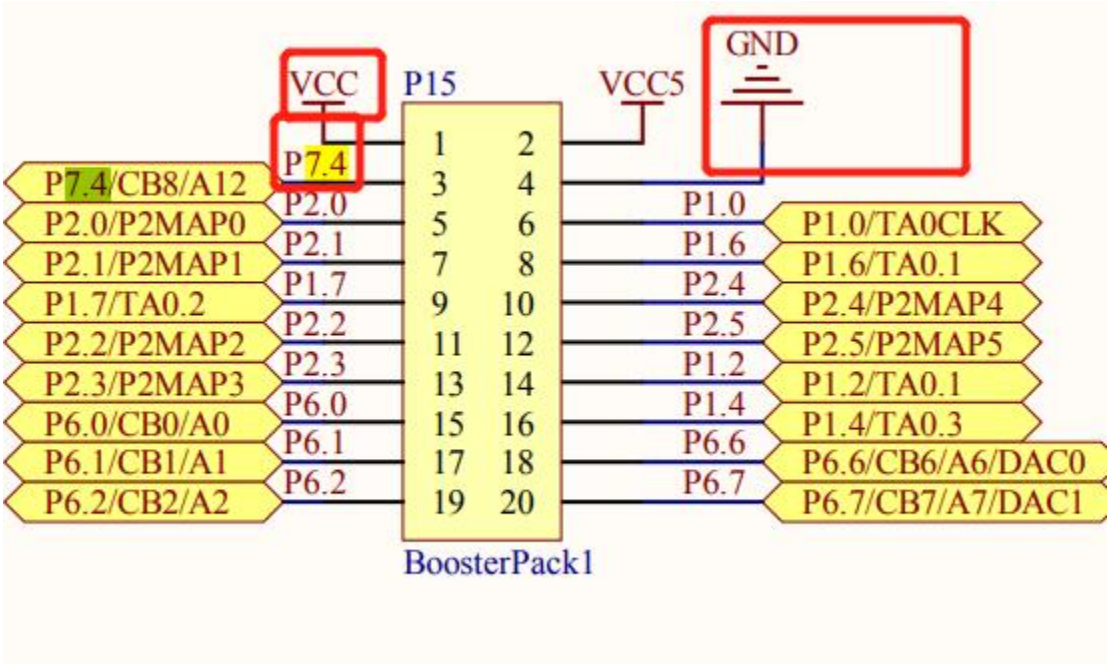


图 2.2 ADC 引脚原理图

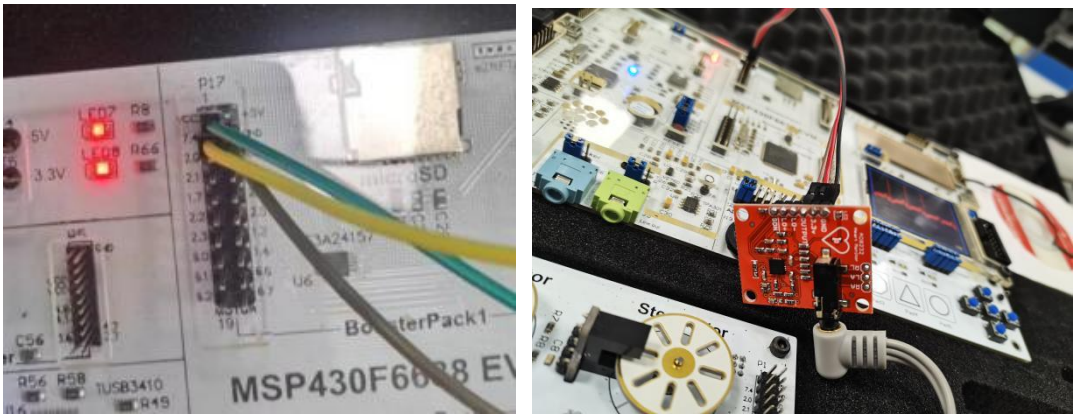


图 2.3 ADC 实物连接图

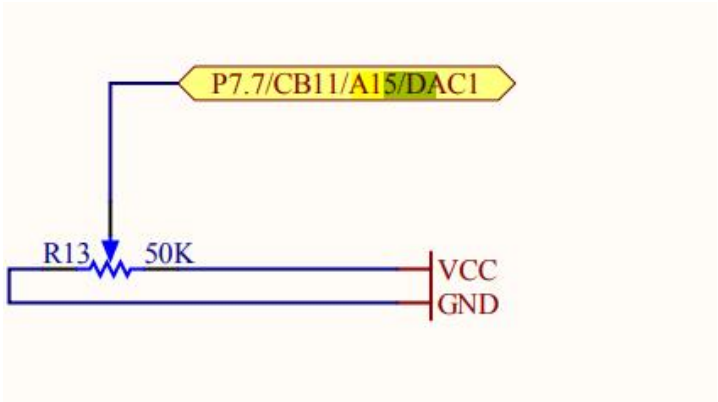


图 2.4 ADC 15 引脚

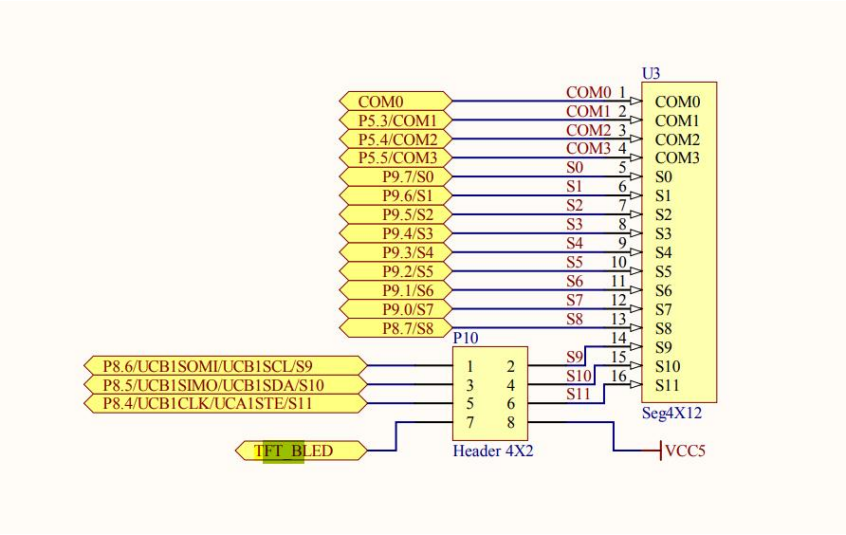
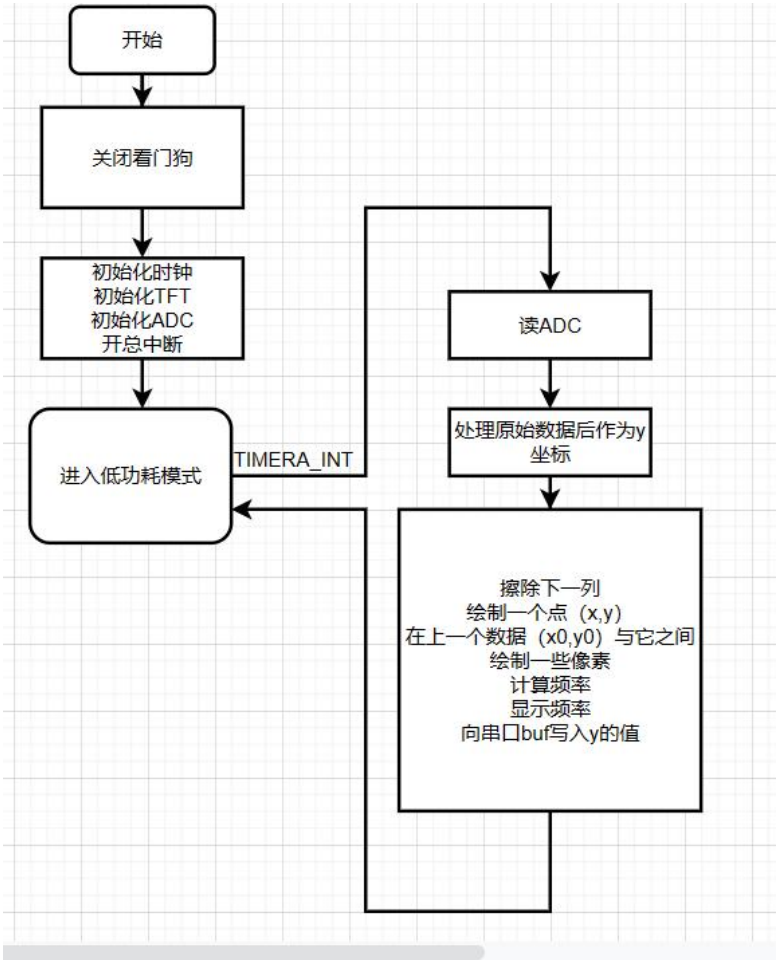


图 2.5 TFT 输出

3. 流程图



4. 关键代码实现

4.1 主函数思路

- **关闭看门狗计时器：**看门狗计时器是一种硬件计时器，用于检测和解决系统错误。如果系统运行正常，看门狗计时器会被定期重置；如果系统出现错误，看门狗计时器会溢出，从而重置系统或触发其他错误处理程序。
- **配置定时器 A0：**定时器 A0 被配置为使用 SMCLK 时钟，以比较模式运行，并在开始时清零计数器。定时器的比较/捕获寄存器 0（TA0CCTL0）被配置为启用中断，而比较/捕获寄存器 0（TA0CCR0）的值被设置为 50000，这意味着每当计数器达到 50000 时，就会触发一个中断。这相当于创建了一个 50ms 的时间间隔。
- **配置 IO 端口：**P5.7 和 P8.0 被配置为输出端口。
- **配置 ADC12：**ADC12 模块被配置为自动循环采样转换模式，这意味着 ADC 会连续采样和转换指定的一组通道。ADC12 模块被打开，采样保持模式被启用，通道 12（连接到 P7.4）被选择为 ADC 输入，ADC 转换被启用。
- **禁用中断：**为了防止在初始化时发生不期望的中断，代码禁用了全局中断。
- **初始化时钟和 TFT：**调用 initClock()和 initTFT()函数来初始化系统时钟和 TFT 显示。
- **设置 TFT 显示区域：**调用 etft_AreaSet()函数来设置 TFT 显示的区域。
- 启用中断并进入低功耗模式：代码重新启用了全局中断，并将系统设置为低功耗模式 0（LPM0），这意味着 CPU 将被关闭，但系统时钟和外设（如定时器和 ADC）仍将运行。

```
void main(void)
{
        WDTCTL = WDTPW + WDTHOLD;//关闭看门狗

        TA0CTL |= MC_1 + TASSEL_2 + TACLR;
        //时钟为 SMCLK,比较模式，开始时清零计数器
        TA0CCTL0 = CCIE;//比较器中断使能
        TA0CCR0  = 50000;//比较值设为 50000，相当于 50ms 的时间间隔

        P5DIR |= BIT7;
        P8DIR |= BIT0;
        ADC12CTL0 |= ADC12MSC;//自动循环采样转换
        ADC12CTL0 |= ADC12ON;//启动 ADC12 模块
        ADC12CTL1 |= ADC12CONSEQ1 ;//选择单通道循环采样转换
        ADC12CTL1 |= ADC12SHP;//采样保持模式
        ADC12MCTL0 |= ADC12INCH_12; //选择通道 15，连接拨码电位器；ch12:p7.4
        ADC12CTL0 |= ADC12ENC;

        _DINT();
```

```
        initClock();
        initTFT();

        etft_AreaSet(0,0,319,239,31);
        _EINT();
        __bis_SR_register(LPM0_bits + GIE);//进入低功耗并开启总中断

}
```

4.2 中断服务函数

这段代码定义了一个 Timer_A 的中断服务函数，当 Timer_A 的中断触发时，这个函数就会被调用。函数的主要作用是使用 ADC（模数转换器）对某个信号进行采样，并将采样结果在 TFT LCD 屏幕上以某种形式进行显示。具体的流程如下：

首先，函数启动 ADC 转换，并将结果保存在变量 value 中。

- 然后，函数对 x（可能代表 LCD 屏幕上的 x 坐标）进行自增。在 flag 为 0 的情况下，会对 y0 进行赋值操作，并再次启动 ADC 转换。如果 flag 不为 0，那么就直接进行后续的操作。
- 然后，函数对 y 进行赋值操作。这里 y 可能代表 LCD 屏幕上的 y 坐标。
- 接着，函数检查 x 是否大于或等于 309。如果是，就将 x 重新设置为 4，并在 LCD 屏幕上的指定区域设置像素颜色，然后再对 x 进行自增。如果不是，那么就直接进行后续的操作。
- 然后，函数计算 y0 和 y 的平均值，并保存在 tmp0 中。并在 LCD 屏幕上的另一个指定区域设置像素颜色。
- 接着，函数根据 y0 和 y 的大小关系，再在 LCD 屏幕上的两个指定区域设置像素颜色。
- 然后，函数调用 pulse_counter 函数，传入 y 和 y0 作为参数，计算出脉冲计数，并保存在 freq 中。这里的脉冲计数可能代表了某种物理量的频率。
- 如果 freq 大于 0，那么就将 freq 的百位、十位和个位分别转换为字符，保存在 disp 数组中。
- 最后，函数在 LCD 屏幕上的指定位置显示 disp 数组中的字符串，然后将 y 的值赋给 y0，为下一次循环做准备。

```
#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer_A (void) //定义 Timer_A 中断服务函数
{
    ADC12CTL0 |= ADC12SC; //启动 ADC 转换
    value = ADC12MEM0; //将 ADC 转换结果赋值给变量 value

    x++; //x 坐标加 1
```

```
    if (flag==0) //如果 flag 为 0
    {
        flag=1; //将 flag 设为 1
        y0=232-value/18; //将 y0 设为经过特定计算后的值

        ADC12CTL0 |= ADC12SC; //开始另一次 ADC 转换
        value = ADC12MEM0; //将 ADC 转换结果赋值给变量 value
    }

    y=232-value/18; //将 y 设为经过特定计算后的值

    if (x>=309) //如果 x 大于或等于 309
    {
        x=4; //将 x 设为 4
        etft_AreaSet(x,4,x+12,235,0); //在指定区域设置像素颜色
        x++; //x 坐标加 1
    }

    tmp0=(y0+y)/2; //将 tmp0 设为 y0 和 y 的平均值

    etft_AreaSet(x+1,4,x+6,235,0); //在指定区域设置像素颜色

    if (y0<y) //如果 y0 小于 y
    {
        etft_AreaSet(x,y0,x,tmp0,0xf800); //在指定区域设置像素颜色
        etft_AreaSet(x+1,tmp0,x+1,y,0xf800); //在指定区域设置像素颜色
    }
    else //否则
    {
        etft_AreaSet(x,tmp0,x,y0,0xf800); //在指定区域设置像素颜色
        etft_AreaSet(x+1,y,x+1,tmp0,0xf800); //在指定区域设置像素颜色
    }

    int freq=pulse_counter(y,y0); //计算 y 和 y0 的脉冲计数

    if(freq>0) //如果 freq 大于 0
    {
        disp[0]=freq/100+'0'; //将频率的百位数转换为字符并存入数组
        disp[1]=(freq%100)/10+'0'; //将频率的十位数转换为字符并存入数组
        disp[2]=freq%10+'0'; //将频率的个位数转换为字符并存入数组
    }
```

```
    etft_DisplayString(disp, 280,215,0xffff, 0x0000); //在指定位置显示字符串

    y0=y; //将 y 赋值给 y0
}
```

4.3 计算脉冲函数：

这段代码定义了一个名为 pulse_counter 的函数，它接收两个整型参数 y 和 y0。这个函数的主要目标是计算并返回脉冲频率。

- **计算脉冲的差值并更新脉冲总和：**函数首先从 pulse_sum（一个表示脉冲总和的全局变量）中减去 pulse_cache[pulse_n]的值（这是一个缓存数组，用于存储过去的脉冲值）。然后，它计算当前脉冲 y 和上一个脉冲 y0 之间的差值，并将这个差值存入 pulse_cache 数组的当前位置。最后，它将新的差值添加到 pulse_sum 中，这样 pulse_sum 就被更新为最近 15 个脉冲差值的总和。
- **更新脉冲缓存数组的索引：**pulse_n 是 pulse_cache 数组的索引，每次调用 pulse_counter 函数时，pulse_n 都会增加 1。当 pulse_n 达到 15 时，它被重置为 0，这意味着 pulse_cache 数组是一个大小为 15 的循环缓存。
- **检测并处理脉冲的开始和结束：**如果 counter_flag 为 0（表示没有检测到脉冲的开始）并且 pulse_sum 大于或等于 40（这是一个设定的阈值，用于确定脉冲的开始），函数将 counter_flag 设为 1（表示脉冲已开始），将 pulse_count（表示已经过去的脉冲数量）的值存入 tmp 中，并将 pulse_count 重置为 0。然后，函数返回 6000/tmp，这是脉冲的频率。如果 pulse_sum 小于 40，函数将 counter_flag 设为 0（表示脉冲已结束）。
- **更新脉冲数量并返回-1：**如果函数没有返回脉冲频率，它将增加 pulse_count 的值，并返回-1，表示在当前调用中没有检测到脉冲的结束。

总的来说，这个函数通过维护一个脉冲差值的滑动窗口，并监测窗口内的脉冲总和，来检测和计算脉冲的频率。

```
int pulse_counter(int y,int t0)
{
    // 从总的脉冲总和中减去旧的脉冲值
    pulse_sum-=pulse_cache[pulse_n];
    // 计算当前脉冲和上一个脉冲的差值，并更新脉冲缓存数组
    pulse_cache[pulse_n]=y-y0;
    // 将新的脉冲差值加入总的脉冲总和
    pulse_sum+=pulse_cache[pulse_n];
    // 将脉冲缓存数组的索引增加 1
    pulse_n++;

    // 如果脉冲缓存数组的索引达到 15（数组大小），则将其重置为 0
    if(pulse_n>=15) pulse_n=0;
```

```
    // 如果未检测到脉冲的开始且脉冲总和大于等于 40
    if((counter_flag==0)&&(pulse_sum>=40))
        {
            // 设置标志位，表示检测到脉冲的开始
            counter_flag=1;

            // 保存当前的脉冲数量，并将其重置为 0
            int tmp=pulse_count;
            pulse_count=0;

            // 返回脉冲的频率
            return 6000/tmp;
        }
    // 如果脉冲总和小于 40
    else if(pulse_sum<40) counter_flag=0; // 将标志位重置，表示脉冲已结束

    // 增加脉冲数量
    pulse_count++;
    // 如果在当前函数调用中未检测到脉冲的结束，返回-1
    return -1;
}
```

5.结果展示

5.1 电极片采集位置

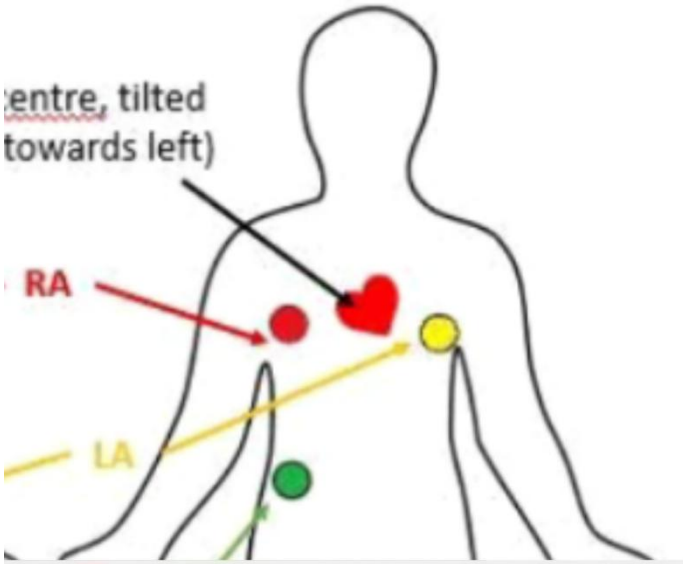


图 5.1 电极片位置

5.2 波形展示

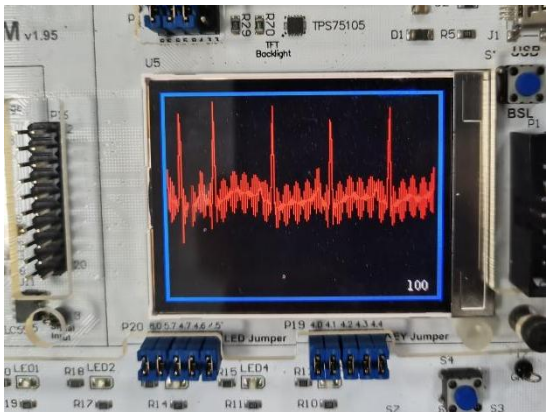


图 5.2 心电波形展示

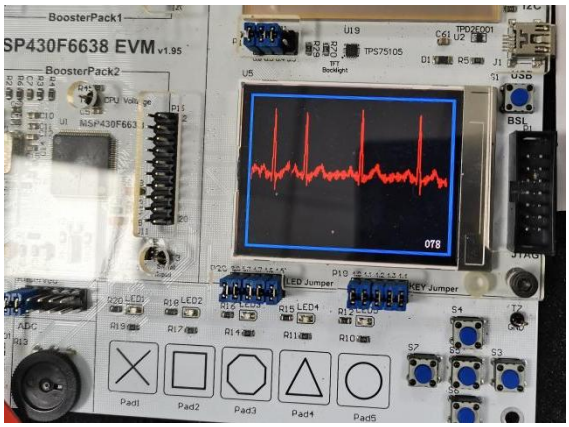


图 5.3 心电波形展示

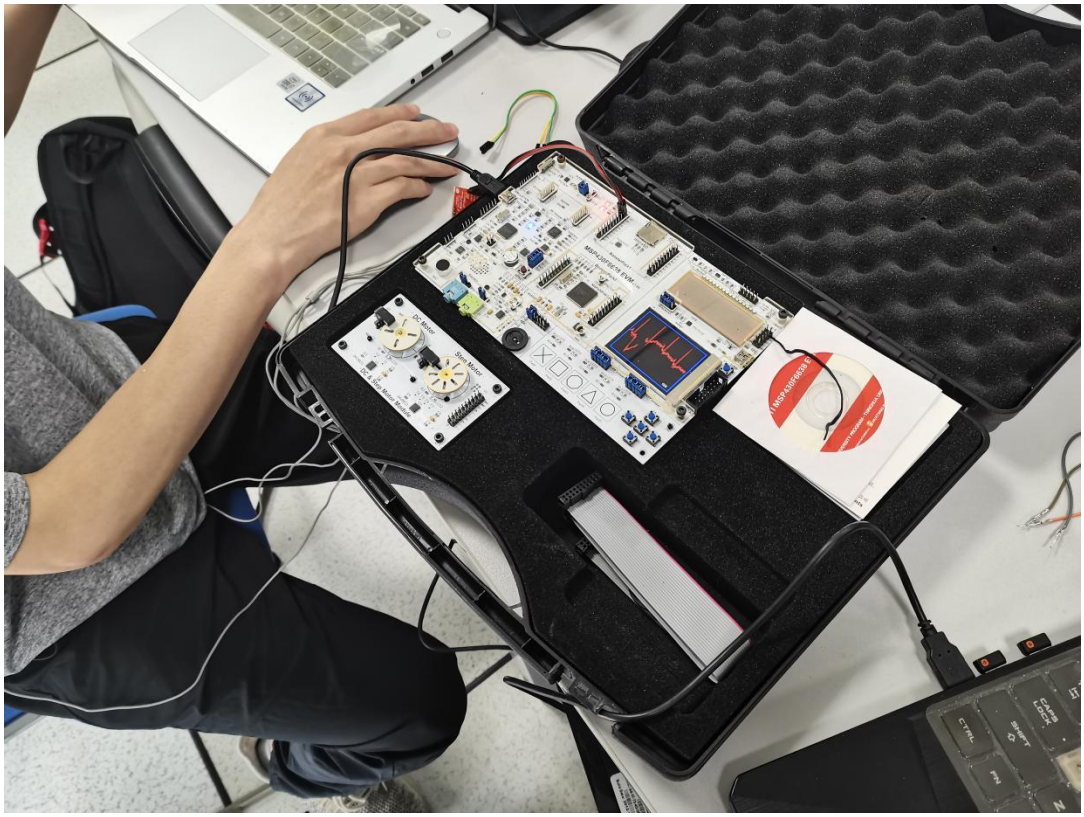


图 5.4 实物展示

5.3 串口通信结果展示

我们将串口通信结果存入 txt 文件中，并将其转成十进制文件进行后续处理

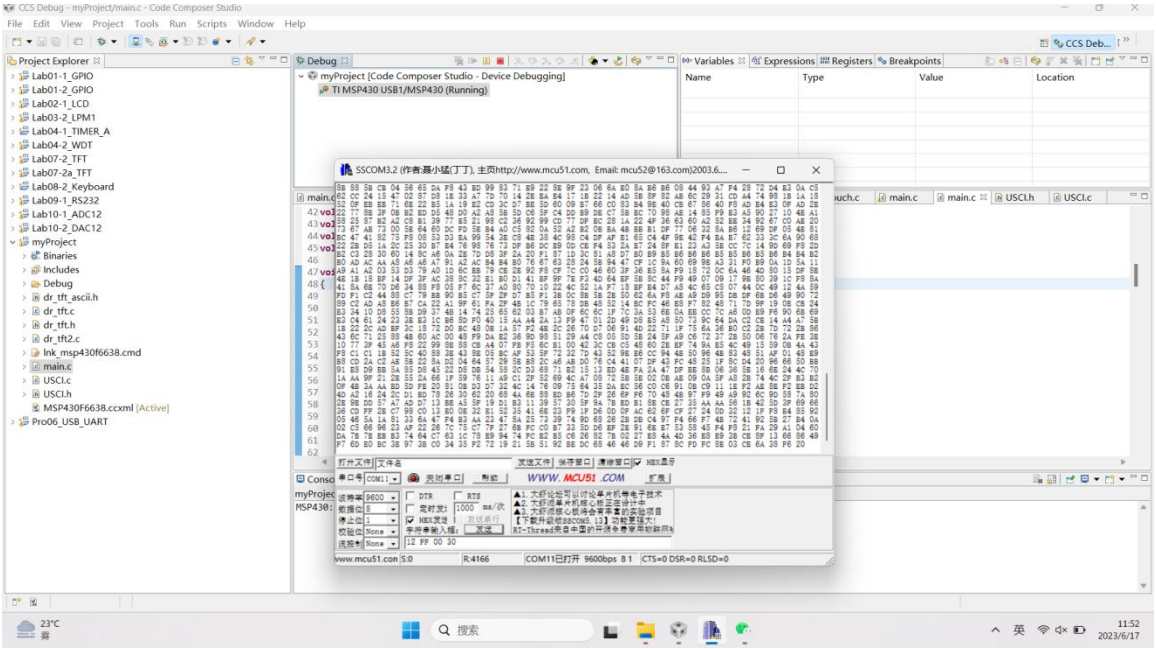


图 5.5 串口通信

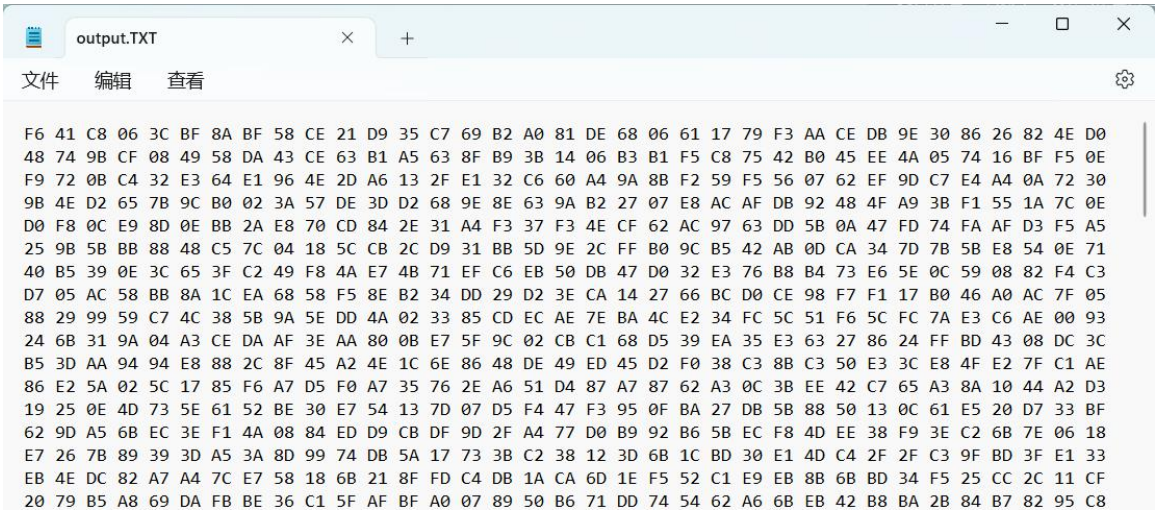


图 5.6 16 串口接收的 16 进制文件

5.4 桌面端处理展示

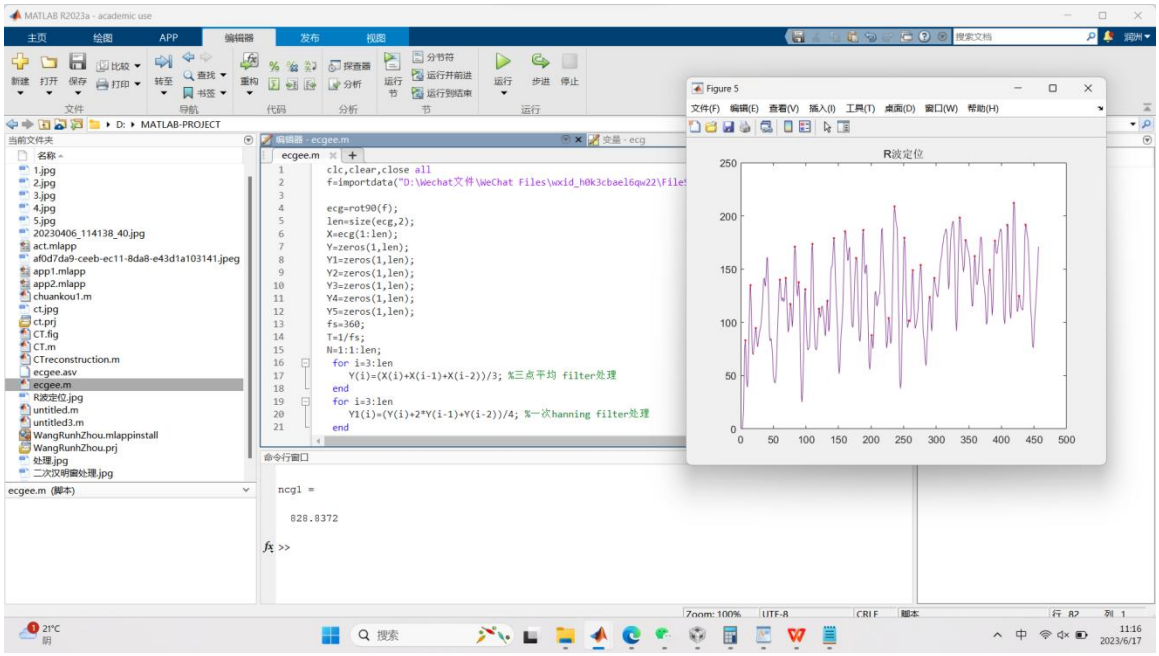


图 5.7 桌面端处理

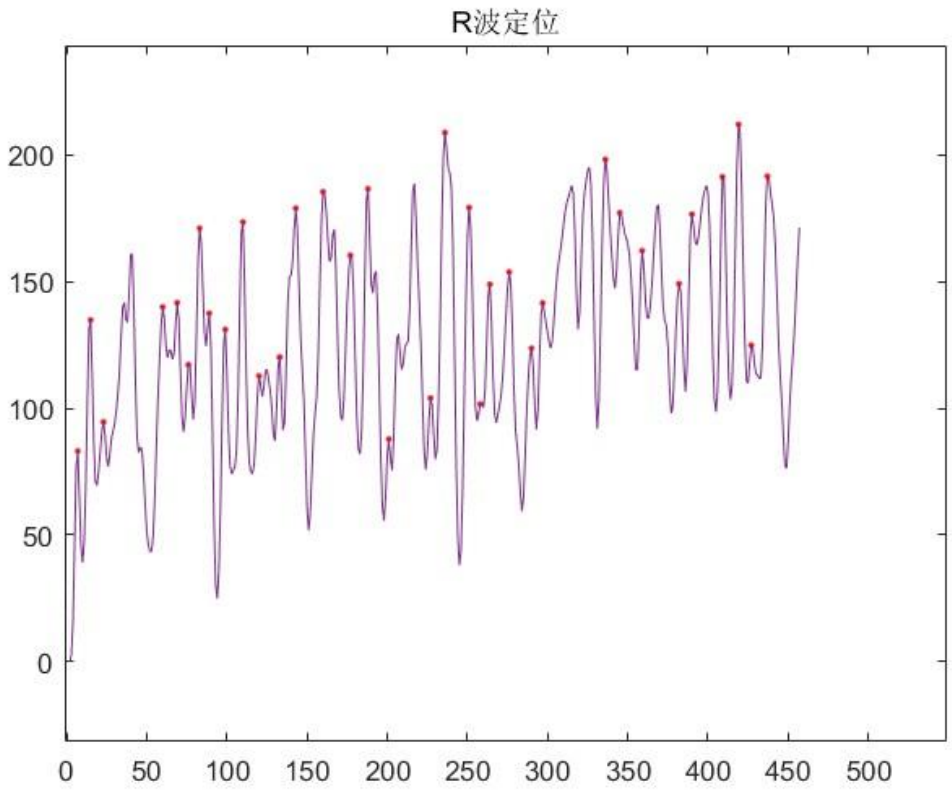


图 5.8 R 波电位

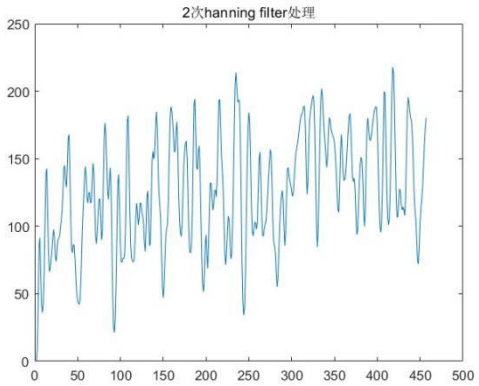


图 5.9. 2 阶 filter 处理

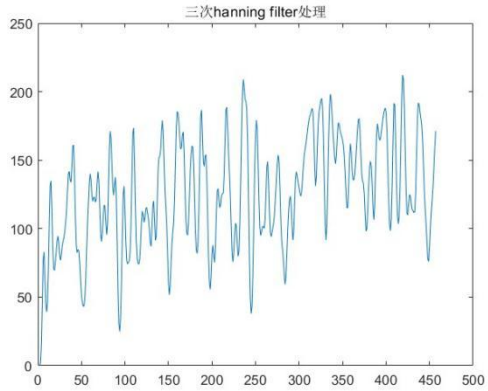


图 5.10. 2 阶 filter 处理

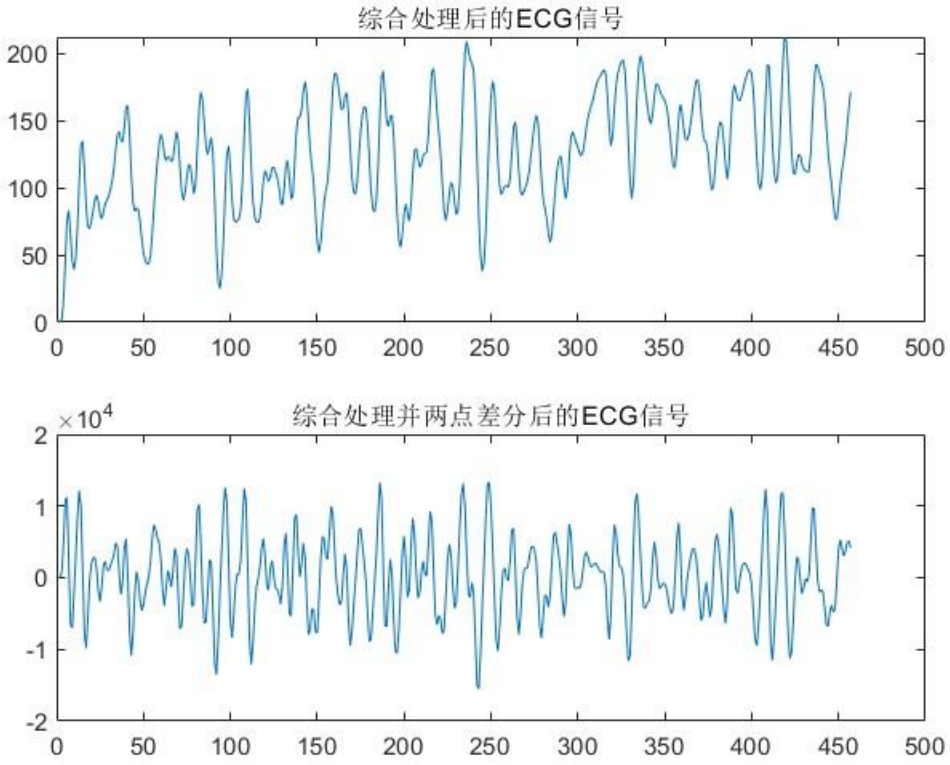


图 5.10 综合处理后的心电信号

6 困难和解决方案

6.1 问题一：心电信号噪声大

在我们初步实现心电信号采集时，我们发现采集到的数据中噪声较大，对心率的计算造成了困扰。

解决方案：我们采用了数字滤波技术来减少噪声的影响。通过实现一个适当的滤波器，我们能够有效地消除电源噪声和环境噪声，从而提高了心电信号的质量。

6.2 问题二：实时显示心率的延迟

在初步的实现中，我们发现系统在计算和显示心率时存在一定的延迟。

解决方案：我们优化了心率计算的算法，使用了滑动窗口来更快地检测和计算心跳。此外，我们还优化了显示模块的刷新率，以减少显示的延迟。

6.3 问题三：功耗较高

在测试系统的运行时间时，我们发现系统的功耗比预期的要高，这会限制其在无线电源的环境下的使用时间。

解决方案：我们针对硬件和软件进行了一系列的优化，以降低功耗。例如，我们使用了低功耗模式进行待机，优化了代码以减少不必要的计算，以及选择了低功耗的硬件组件。这些优化使得我们的系统在保持功能和性能的同时，显著降低了功耗。

7 总结与展望

7.1 总结

在本项目中，我们成功地设计并实现了基于 MSP430 微控制器的心电采集显示心率计数系统。这个系统能够采集和处理心电信号，准确地计算出心率，并将结果显示出来。我们的系统具有低功耗、高精度、快速响应等优点，可以广泛应用于医疗、健康管理、体育运动等领域。

在项目实施过程中，我们团队三人共同努力，各自发挥自己的专长，有效地完成了硬件设计、软件编程、系统测试等工作。我们通过实践锻炼了自己的专业技能，也提高了解决问题和团队协作的能力。

7.2 展望

虽然我们的心电采集显示心率计数系统已经可以正常工作，但我们认识到，还有许多可以改进和优化的地方。在未来，我们计划进行以下方面的工作：

改善用户界面：我们计划优化系统的用户界面，使其更加友好和直观。我们也考虑增加一些新的功能，如心率异常警报、历史数据查看等。

提高系统精度：我们将研究更高级的信号处理算法，以提高系统对心率的测量精度。我们也会考虑使用更高性能的硬件组件，以提高系统的整体性能。

优化功耗：鉴于许多应用场景要求设备具有较长的续航时间，我们将研究如何进一步降低系统的功耗。

我们期待在未来的工作中，继续提高我们的系统，使其更好地满足用户的需求。

8 附录

8.1 主函数实现

```
/*
 * main.c
 */
#include <msp430.h>
#include <stdint.h>
#include <stdio.h>
#include "dr_tft.h"

//unsigned char flag0=0,flag1=0;
unsigned char send_data[]={'0','\0'};
unsigned char recv_data[]={'0','\0'};
void UART_RS232_Init(void);
void initClock()
{
      while(BAKCTL & LOCKIO)                // Unlock XT1 pins for operation
        BAKCTL &= ~(LOCKIO);
        UCSCTL6 &= ~XT1OFF;                 //启动 XT1
        P7SEL |= BIT2 + BIT3;               //XT2 引脚功能选择
        UCSCTL6 &= ~XT2OFF;                 //启动 XT2
        while (SFRIFG1 & OFIFG) {           //等待 XT1、XT2 与 DCO 稳定
            UCSCTL7 &= ~(DCOFFG+XT1LFOFFG+XT2OFFG);
            SFRIFG1 &= ~OFIFG;
          }

      UCSCTL4 = SELA__XT1CLK + SELS__XT2CLK + SELM__XT2CLK; //避免 DCO 调整中跑飞

      UCSCTL1 = DCORSEL_5;                          //6000kHz~23.7MHz
      UCSCTL2 = 20000000 / (4000000 / 16);          //XT2 频率较高，分频后作为基准
可获得更高的精度
      UCSCTL3 = SELREF__XT2CLK + FLLREFDIV__16;     //XT2 进行 16 分频后作为基准
      while (SFRIFG1 & OFIFG) {                     //等待 XT1、XT2 与 DCO 稳定
        UCSCTL7 &= ~(DCOFFG+XT1LFOFFG+XT2OFFG);
        SFRIFG1 &= ~OFIFG;
      }
      UCSCTL5 = DIVA__1 + DIVS__1 + DIVM__1; //设定几个 CLK 的分频
      UCSCTL4 = SELA__XT1CLK + SELS__DCOCLK + SELM__DCOCLK; //设定几个 CLK 的时钟
源
}
```

```
volatile unsigned int value = 0;//设置判断变量
volatile unsigned int flag=0;
volatile unsigned int y0 =0;
volatile unsigned int x =4;
volatile unsigned int y =0;
volatile unsigned int tmp0 =4;
volatile char disp[4]="000";

void main(void)
{
        WDTCTL = WDTPW + WDTHOLD;//关闭看门狗
        UART_RS232_Init();


        TA0CTL |= MC_1 + TASSEL_2 + TACLR;
        //时钟为 SMCLK,比较模式，开始时清零计数器
        TA0CCTL0 = CCIE;//比较器中断使能
        TA0CCR0  = 50000;//比较值设为 50000，相当于 50ms 的时间间隔

        P5DIR |= BIT7;
        P8DIR |= BIT0;
        ADC12CTL0 |= ADC12MSC;//自动循环采样转换
        ADC12CTL0 |= ADC12ON;//启动 ADC12 模块
        ADC12CTL1 |= ADC12CONSEQ1 ;//选择单通道循环采样转换
        ADC12CTL1 |= ADC12SHP;//采样保持模式
        ADC12MCTL0 |= ADC12INCH_12; //选择通道 15，连接拨码电位器；ch12:p7.4
        ADC12CTL0 |= ADC12ENC;

        _DINT();

        initClock();
        initTFT();


        etft_AreaSet(0,0,319,239,31);
        _EINT();
        __bis_SR_register(LPM0_bits + GIE);//进入低功耗并开启总中断

}
```

```
volatile int pulse_n=0;
volatile int pulse_cache[15]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
volatile int pulse_sum=0;
volatile int counter_flag=0;
volatile int pulse_count=0;

// 函数 pulse_counter 接收两个整数参数：y 和 y0
int pulse_counter(int y,int t0)
{
    // 从总的脉冲总和中减去旧的脉冲值
    pulse_sum-=pulse_cache[pulse_n];

    // 计算当前脉冲和上一个脉冲的差值，并更新脉冲缓存数组
    pulse_cache[pulse_n]=y-y0;

    // 将新的脉冲差值加入总的脉冲总和
    pulse_sum+=pulse_cache[pulse_n];

    // 将脉冲缓存数组的索引增加 1
    pulse_n++;

    // 如果脉冲缓存数组的索引达到 15（数组大小），则将其重置为 0
    if(pulse_n>=15) pulse_n=0;

    // 如果未检测到脉冲的开始且脉冲总和大于等于 40
    if((counter_flag==0)&&(pulse_sum>=40))
        {
            // 设置标志位，表示检测到脉冲的开始
            counter_flag=1;

            // 保存当前的脉冲数量，并将其重置为 0
            int tmp=pulse_count;
            pulse_count=0;

            // 返回脉冲的频率
            return 6000/tmp;
        }
    // 如果脉冲总和小于 40
    else if(pulse_sum<40) counter_flag=0; // 将标志位重置，表示脉冲已结束
```

```
    // 增加脉冲数量
    pulse_count++;

    // 如果在当前函数调用中未检测到脉冲的结束，返回-1
    return -1;
}

#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer_A (void) //定义 Timer_A 中断服务函数
{
    ADC12CTL0 |= ADC12SC; //启动 ADC 转换
    value = ADC12MEM0; //将 ADC 转换结果赋值给变量 value

    x++; //x 坐标加 1

    if (flag==0) //如果 flag 为 0
    {
        flag=1; //将 flag 设为 1
        y0=232-value/18; //将 y0 设为经过特定计算后的值

        ADC12CTL0 |= ADC12SC; //开始另一次 ADC 转换
        value = ADC12MEM0; //将 ADC 转换结果赋值给变量 value
        UCA1TXBUF=value;
    }


    y=232-value/18; //将 y 设为经过特定计算后的值
    UCA1TXBUF=value;

    if (x>=309) //如果 x 大于或等于 309
    {
        x=4; //将 x 设为 4
        etft_AreaSet(x,4,x+12,235,0); //在指定区域设置像素颜色
        x++; //x 坐标加 1
    }

    tmp0=(y0+y)/2; //将 tmp0 设为 y0 和 y 的平均值

    etft_AreaSet(x+1,4,x+6,235,0); //在指定区域设置像素颜色

    if (y0<y) //如果 y0 小于 y
```

```
    {
        etft_AreaSet(x,y0,x,tmp0,0xf800); //在指定区域设置像素颜色
        etft_AreaSet(x+1,tmp0,x+1,y,0xf800); //在指定区域设置像素颜色
    }
    else //否则
    {
        etft_AreaSet(x,tmp0,x,y0,0xf800); //在指定区域设置像素颜色
        etft_AreaSet(x+1,y,x+1,tmp0,0xf800); //在指定区域设置像素颜色
    }

    int freq=pulse_counter(y,y0); //计算 y 和 y0 的脉冲计数

    if(freq>0) //如果 freq 大于 0
    {
        disp[0]=freq/100+'0'; //将频率的百位数转换为字符并存入数组
        disp[1]=(freq%100)/10+'0'; //将频率的十位数转换为字符并存入数组
        disp[2]=freq%10+'0'; //将频率的个位数转换为字符并存入数组
    }

    etft_DisplayString(disp, 280,215,0xffff, 0x0000); //在指定位置显示字符串

    y0=y; //将 y 赋值给 y0
}

void UART_RS232_Init(void) //RS232 接口初始化函数
{
    /*通过对 P3.4。P3.5，P4.4，P4.5 的配置实现通道选择
         使 USCI 切换到 UART 模式*/
    P3DIR|=(1<<4)|(1<<5);
    P4DIR|=(1<<4)|(1<<5);
    P4OUT|=(1<<4);
    P4OUT&=~(1<<5);
    P3OUT|=(1<<5);
    P3OUT&=~(1<<4);
    P8SEL|=0x0c;    //模块功能接口设置，即 P8.2 与 P8.3 作为 USCI 的接收口与发射口
    UCA1CTL1|=UCSWRST; //复位 USCI
    UCA1CTL1|=UCSSEL_1;//设置辅助时钟，用于发生特定波特率
    UCA1BR0=0x03;       //设置波特率
    UCA1BR1=0x00;
    UCA1MCTL=UCBRS_3+UCBRF_0;
    UCA1CTL1&=~UCSWRST;//结束复位
```

```
    UCA1IE|=UCRXIE;     //使能接收中断
}
```

8.2 处理函数实现

```
clc,clear,close all
f=importdata("D:\Wechat 文 件 \WeChat Files\wxid_h0k3cbael6qw22\FileStorage\File\2023-06\dec_file2.txt");

ecg=rot90(f);
len=size(ecg,2);
X=ecg(1:len);
Y=zeros(1,len);
Y1=zeros(1,len);
Y2=zeros(1,len);
Y3=zeros(1,len);
Y4=zeros(1,len);
Y5=zeros(1,len);
fs=360;
T=1/fs;
N=1:1:len;
 for i=3:len
     Y(i)=(X(i)+X(i-1)+X(i-2))/3; %三点平均 filter 处理
 end
 for i=3:len
     Y1(i)=(Y(i)+2*Y(i-1)+Y(i-2))/4; %一次 hanning filter 处理
 end
 for i=3:len
     Y2(i)=(Y1(i)+2*Y1(i-1)+Y1(i-2))/4; %二次 hanning filter 处理
 end
 for i=3:len
     Y3(i)=(Y2(i)+2*Y2(i-1)+Y2(i-2))/4; %三次 hanning filter 处理
 end
  for i=2:len
     Y4(i)=(Y3(i)-Y3(i-1))/T; %两点差分求导
  end

 subplot(2,1,1)
 plot(Y3);title('综合处理后的 ECG 信号');
 subplot(2,1,2)
 plot(Y4);title('综合处理并两点差分后的 ECG 信号');
```

```
 figure
plot(Y3);title('三次 hanning filter 处理');
 figure
plot(Y2);title('2 次 hanning filter 处理');

% for i=0:len
Ymax=0;
for i=1:len
    if(Y4(i)<Ymax)
        Ymax=Ymax;
    else
        Ymax=Y4(i);
    end
end
%%%%%%%找到最大值%%%%%%%
Th=0.07*Ymax;%设置阈值
for i=1:len-1
      if((Y4(i)>Th)&&(Y3(i)>Y3(i-1))&&(Y3(i)>Y3(i+1)))
            Y5(i)=1;
      else
          Y5(i)=0;
      end
end
figure
plot(Y5);title('阈值检测');
%%%%%%%阈值判断%%%%%%%
 K=zeros(1,len);
j=1;
figure
for i=1:len
      if(Y5(i)==1)
           K(j)=i;
          plot(N,Y3,i,Y3(i),'r.');hold on;
           j=j+1;

      end
end
title('R 波定位');
%%%%%%%定位%%%%%%%
p=0;
for i=1:j-2
```

```
        p=p+(K(i+1)-K(i));
end
T1=p/(j-2)/180;
ncg1=60/T1
```