定量工程设计 II

2024 至 2025 学年第 二 学期

学号： 20232373

姓名： 莫湘渝

学院： 国家卓越工程师学院

项目： 心电测量方案设计

1 项目背景介绍

心电信号（Electrocardiogram, ECG）是反映心脏电生理活动的重要生命体征，具有心率、节律、波形等关键特征，能够直观反映心脏功能状态。在临床医学中，心电信号被广泛应用于诊断心律失常、心肌缺血、传导阻滞等多种心脏疾病，是心血管诊疗的重要依据。

随着可穿戴技术与便携设备的发展，实时心电监测已逐渐融入日常健康管理，特别适用于心血管高风险人群的预警与随访，有助于降低发病率和致死率。此外，心电信号在生物医学研究中也具有重要价值，为心血管药物研发、人工心脏等先进医疗技术提供数据支撑。

在当前人口老龄化和心血管疾病高发的背景下，构建高精度、高可靠性的心电信号采集与分析系统尤为迫切。本项目将依托课程中所学的数字信号处理等专业技术，开发便捷智能的心电监测方案，助力实现个性化健康管理与精准医疗的深度融合。

2 实验需求分析

2.1 心电信号特征与设计需求

心电信号（ECG）是由心脏内部一系列协调的电刺激脉冲所产生。这些脉冲驱动心肌细胞有节律地舒张与收缩，并在体表不同部位形成微弱的电位差。通过专用仪器可从体表检测到这些电位差信号，称之为心电信号。本质上，心电信号反映的是心肌细胞膜在兴奋过程中产生的电势变化。

心电信号的临床价值极高，医生通常通过分析心率、幅值等参数对患者的心脏健康状况进行初步判断。因此，实现高精度的心率与幅值测量，是本项目所设计的心电信号采集与处理系统的核心目标。

正常心电信号的频率范围为 0.05 Hz 至 100 Hz，其中 99% 的能量集中在 0 – 35 Hz 的低频段。然而，在实际采集过程中，心电信号极易受到各类干扰影响，主要包括以下几类：

①工频及其谐波干扰：我国工频为 50 Hz，属于中频干扰；

②高频噪声：来自肌颤、电源纹波、参考电压不稳等，频率多在 100 Hz 上；

③低频干扰：如呼吸引起的基线漂移、电极滑动以及温度引起的参考电压变化，这类干扰频率一般在 0–0.7 Hz 范围内。

这些干扰会使采集到的心电信号中夹杂大量噪声，影响信号的质量和诊断准确性。因此，必须对原始信号进行有效滤波处理，以提取真实的生理信息，提高心率与幅值测量的可靠性。

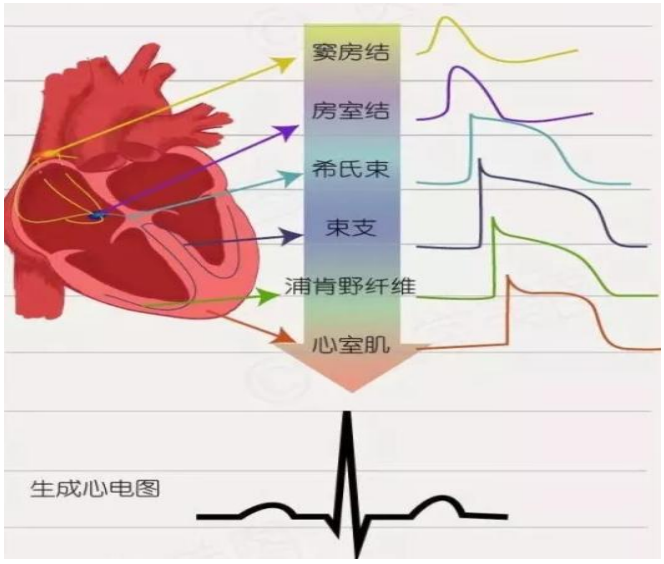

图 2.1.1 心电信号简介

考虑到低频干扰频率与心电信号频谱相近，滤波器设计需具备较窄的过渡带和良好的阻带性能，尤其是在 0 Hz 附近。为此，本实验将在嵌入式系统内运用直流陷波器级联 FIR 低通滤波器进行噪声抑制，同时也在上位机 PC 端的 matlab 中对传回的原始数据实现 IIR 级联 FIR，用以抑制缓变直流与基线漂移，确保心电信号保真度。

基于以上分析，本系统提出以下关键技术指标要求：

①在 0 Hz 处缓变直流信号的衰减不低于 30 dB，以有效抑制基线漂移；

②降噪滤波器的 3 dB 通带截止频率设为 35 Hz，过渡带宽度不超过 10 Hz，阻带衰减不低于 40 dB；

③心率估算误差不超过 ±10%，以满足临床使用与健康监测的精度需求。

通过上述设计，本系统将实现对心电信号的高保真采集与精确处理，并使用可视化波形打印，反映心电特征。

3 实现方案论证

3.1 系统框架设计

本项目的核心目标是实现心电信号的采集与滤波以及心率测量，同时需要在 TFT 屏幕上绘制测量数据以及时域波形与频谱图，附加实现上位机实时显示心电波形，心率等，最后可利用回传原始数据进行 PC 端的数据静态处理。具体而言，需要实现的功能如下：

①通过 ADS1292r 获取心电信号原始数据

②通过设计的直流陷波器和 FIR 滤波器进行数据滤波、降噪、排干扰。实现 STM32 单片机中对原始数据进行降噪和提取心率

③串口回传原始数据和滤波后的结果数据，并将结果数据实时绘图。

④通过 MATLAB 对原始数据进行时域和频域分析；

⑤通过 MATLAB 对原始数据进行直流陷波器，FIR 滤波器设计，并对数据降噪和提取心率；

⑥TFT 屏幕中绘制心电信号曲线和显示心率数值。

为实现基于 ADS1292R 芯片的心电采集与处理系统，整体实验流程如下：

①硬件通信验证与芯片初始化

通过调试 ADS1292R_PowerOnInit 初始化函数（已提供），对 ADS1292R 芯片进行配置（注意 scl 时钟极性），并读取其 device_id 寄存器内容。成功读取设备 ID 表明芯片上电正常、SPI 通讯接口连接正确，同时选取通道二数据（未接心电模拟仪）绘制出方波，验证了硬件系统的基本功能，芯片正常。

②原始心电信号采集

配置 ADS1292R 的 DRDY（数据就绪）引脚中断，建立中断服务程序。在中断触发时，实时读取芯片采集到的原始心电数据，并将其缓存至原始数据数组中，实时处理数据，清楚标志位以保障数据连续性和完整性。

③心电数据传输至上位机

通过串口（UART）接口将采集到的原始数据从单片机传输到上位机（PC）。数据格式使用 16bits 二级制补码存储，分为 2 个 8bits，其中低位在前，高位在后，便于后续分析与处理。同时将结果数据实时绘制波形。

④PC 端信号分析与滤波器设计

在 PC 端利用 MATLAB 对原始数据进行时域和频域分析，观察信号的噪声成分和特征波形（如 P、QRS、T 波）。基于分析结果设计合适的数字 FIR 低通滤波器、陷波器（工频干扰抑制）或带通滤波器（保留心电主频带），并验证其滤波效果。

⑤滤波算法嵌入式移植

将 PC 上设计并验证有效的 FIR 或 IIR 滤波器算法移植到单片机环境中。使用 C 语言对滤波器进行定点化处理，以兼容嵌入式资源受限的特点，并优化内存占用和运算效率。

⑥TFT 屏幕实时显示结果

将滤波后的心电波形、提取的心率（BPM）等参数实时绘制至 TFT 显示屏。用户可直观观察当前 ECG 波形、心跳频率等生理参数，提升系统可用性与交互性。

根据如上设计流程，结合目前提供的材料，设计了如下图所示的心电信号采集与分析系统：

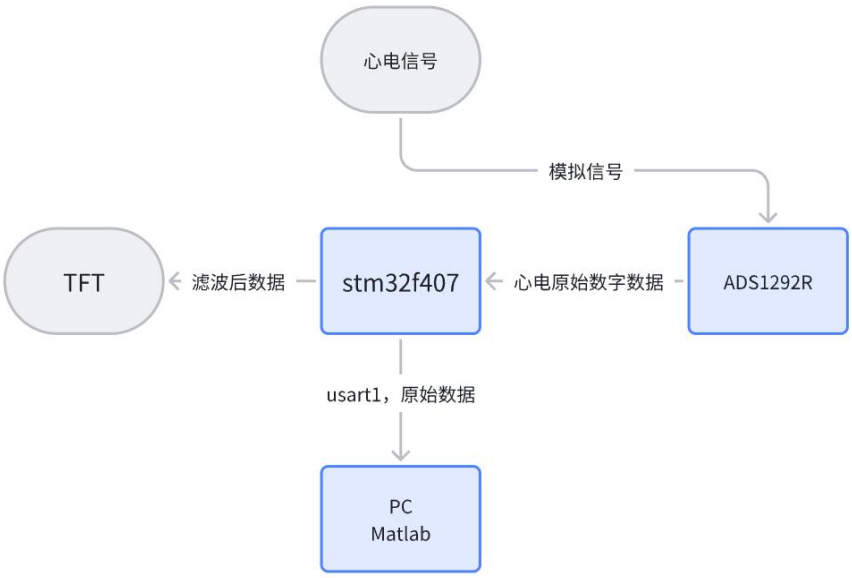

图 2.1：系统框架

在本项目中，我们主要使用正常的心电波形与标准心率信号作为输入。通过操作心电模拟仪，可灵活调节心电信号的心率和幅度，以满足系统测试需求。在信号接入方面，采用标准的三导联连接法：RA（红色）连接右手，LA（黄色）连接左手，LL（绿色）连接左脚。模拟仪输出的心电信号通过 3.5mm 耳机接口连接的动态导联线传输至 ADS1292R 模拟前端芯片，用于后续采集。ADS1292R 芯片负责接收并放大心电电极采集到的原始信号，并通过 SPI 接口将数据传输给 STM32F407 微控制器。为了提升系统响应速度和数据处理效率，STM32 采用基于中断驱动的方式进行数据采集：每当 ADS1292R 完成一次数据采样后，即触发 DRDY 中断，通知 STM32F407 实时读取并缓存当前心电数据。这种中断式采集机制有效避免了轮询带来的 CPU 占用，降低了功耗，同时大幅提升了系统的实时性与处理效率，为后续的数据分析与可视化提供了稳定保障。

图 2.2：中断驱动 SPI 通信采集数据的系统框架简图

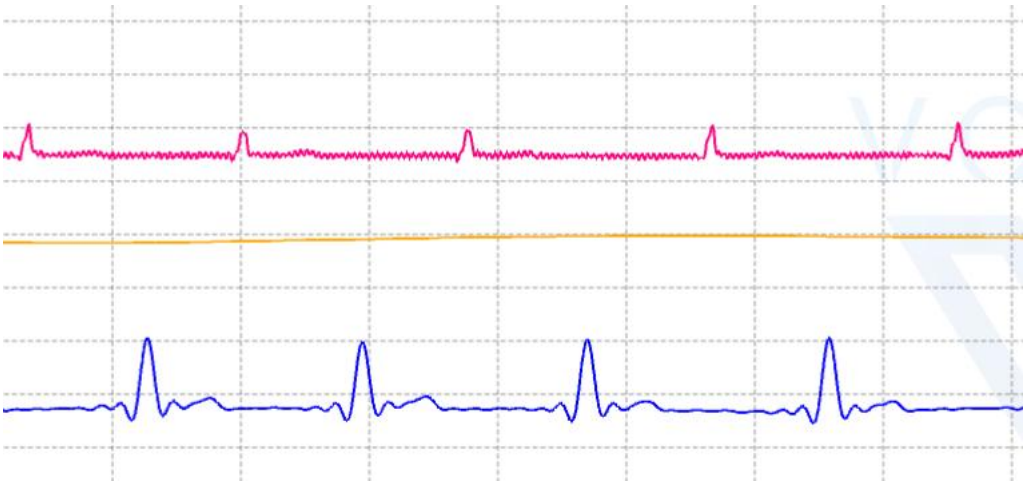

图 2.3：PC 端可视化软件中显示采集到的原始心电信号和滤波后信号

4 理论推导与 MATLAB 计算

为验证滤波器在心电信号处理中的实际效果，本项目在 MATLAB 环境下对采集到的原始心电数据进行了深入分析与仿真处理。主要流程包括：原始数据的读取、时域波形绘制、频谱分析、数字滤波器设计与效果验证。

具体而言，首先对原始心电信号进行时域可视化，并利用傅里叶变换分析其频谱特性，以识别其中的主要频率成分和噪声干扰。随后，设计并应用一系列数字滤波器，包括用于消除直流偏移的陷波器（Notch Filter）以及 FIR 类型的低通滤波器，以抑制高频噪声干扰。

在滤波处理后，进一步对比分析滤波前后的频谱变化，并估算信号中包含的心率（BPM）信息，从而验证滤波器在噪声抑制与特征保留方面的有效性。通过这一过程，系统能够明确滤波器参数设置的合理性，并为后续将滤波算法移植至嵌入式系统提供可靠的理论依据与算法参考。

4.1 原始心电信号数据的读取与频谱分析

使用已提供的 serial_data_raw2000.dat 文件中含有的 4096 个原始心电信号数据完成本次 matlab 实验，后续只需要将串口数据转成.dat 文件即可实现 PC 端分析。首先根据以下规则对数据进行解码：

ADS1292R 输出的心电原始数据采用 24 位二进制补码格式进行编码。其中，0x7FFFFF 表示满量程正电压（+2.4V），而 0x800000 表示满量程负电压（-2.4V），信号整体呈对称分布。在实际数据处理过程中，考虑到嵌入式系统存储与计算资源的限制，系统选取原始 24 位数据的高 16 位作为主要处理对象。由于原始数据为补码格式，因此截取后的 16 位数据仍需按照补码方式解析。同时，截取高 16 位等效于将原始 24 位数据右移 8 位，因此其电压映射关系需重新调整：0x7FFF（即 16 位最大正数）对应模拟信号 +2.4V，0x8000（即 16 位最小负数）对应模拟信号-2.4V。

8.3.8 Data Format

The ADS1291, ADS1292, and ADS1292R outputs 24 bits of data per channel in binary twos complement format, MSB first. The LSB has a weight of V_{REF} / (2^{23} – 1). A positive full-scale input produces an output code of 7FFFFFh and the negative full-scale input produces an output code of 800000h. The output clips at these codes for signals exceeding full-scale. Table 10 summarizes ideal output codes for different input signals. All 24 bits toggle when the analog input is at positive or negative full-scale.

Table 10. Ideal Output Code versus Input Signal

INPUT SIGNAL, V_{IN} (AINP – AINN)	IDEAL OUTPUT CODE(1)
≥ V_{REF}	7FFFFFh
+V_{REF} / (2^{23} – 1)	000001h
0	000000h
–V_{REF} / (2^{23} – 1)	FFFFFFh
≤ –V_{REF} (2^{23} / 2^{23} – 1)	800000h

(1) Excludes effects of noise, linearity, offset, and gain error.

图 4.1：原始信号电压与读取信号数据间的关系

每个心电数据点由两个字节组成（低位在前，高位在后），合并后形成一个 16 位的补码整数。通过判断数值大小，将无符号整数正确转换为有符号数，确保负值数据被准确解析。最终得到的心电序列以 double 类型存储，供后续波形绘图、频谱分析和滤波。

```
%% 参数设定
fs = 500;                    % 采样频率 (Hz)
dataLen = 4096;              % 数据长度
notch_a = 0.995;             % 陷波器参数
lp_cutoff = 35;              % FIR低通滤波器截止频率
trans_width = 10;            % 过渡带宽度
dynamic_range = 120;         % 绘图缩放范围

%% Step 1: 读取原始ECG数据
fid = fopen('serial_data_raw2000.dat', 'r');
tmpOut = fread(fid, dataLen * 2, 'uint8');
fclose(fid);

stm32data = zeros(1, dataLen);
for i = 1:dataLen
    val = tmpOut((i-1)*2+1) + tmpOut((i-1)*2+2)*256;
    if val > 32768
        val = val - 65536;
    end
    stm32data(i) = val;
end
raw_ecg = double(stm32data);
```

图 4.2：原始心电信号数据解码读取

数据解析完成后，使用 plot 函数绘制出心电信号的时域波形，同时，应用快速傅里叶变换（FFT）对信号进行频域分析，并绘制频谱图，以观察其频率组成、主要能量分布及噪声干扰特征。

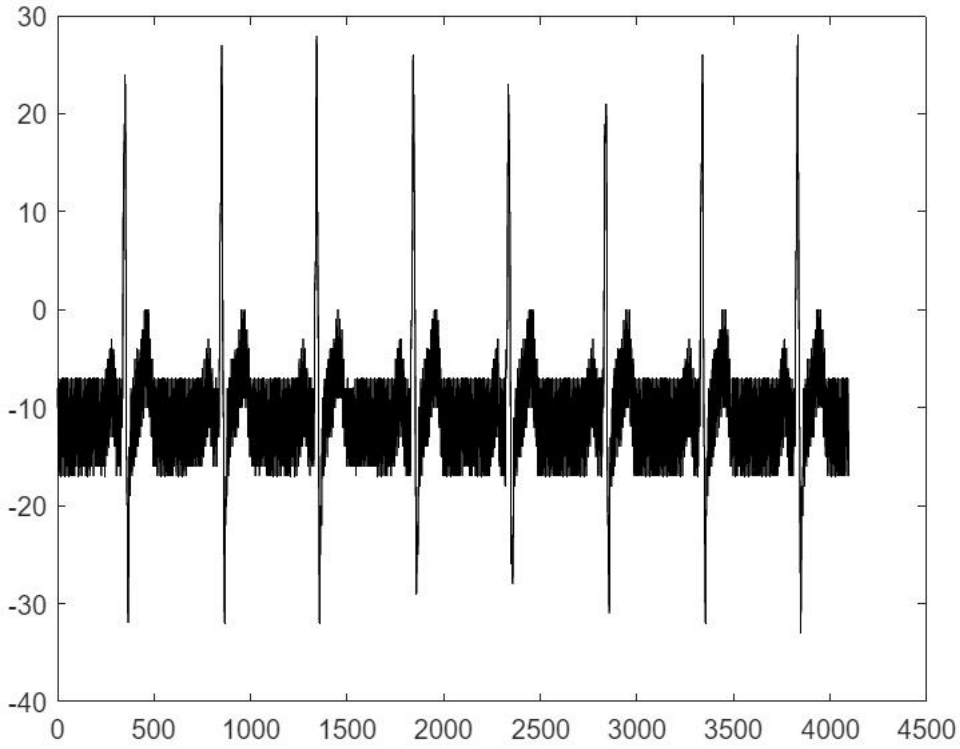

图 4.3：原始心电信号时域波形

```
%% Step 2: 原始信号频谱
f = (0:dataLen-1)*(fs/dataLen);
Y_raw = abs(fft(raw_ecg));

figure;
subplot(4,2,1);
plot(f(1:dataLen/2), Y_raw(1:dataLen/2));
title('1. 原始ECG频谱'); xlabel('频率 (Hz)'); ylabel('幅度');
```

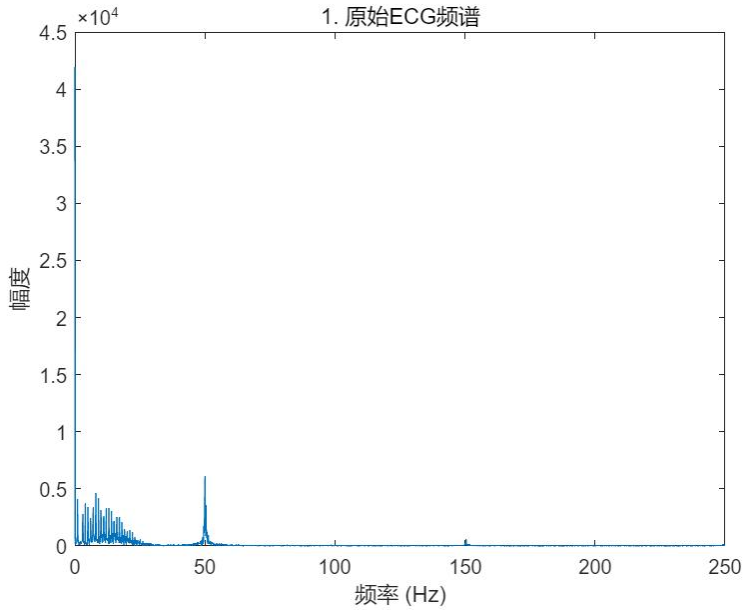

图 4.4：原始心电信号频谱

频谱图中可以明显观察到信号中的低频基线漂移和工频干扰（50HZ 左右），这些成分需要通过滤波器加以去除。

4.2 数字直流陷波器设计与滤波效果

在心电信号采集过程中，呼吸运动、电极松动等因素常常引起基线漂移。这类干扰通常属于低频成分，且其频率范围与心电信号的有效频段十分接近，难以通过常规滤波直接去除。为此，设计一个带宽极窄的直流陷波器成为有效解决方案，用于精准抑制直流及极低频漂移分量。同时，为验证滤波器的效果，采用快速傅里叶变换（FFT）对滤波前后的信号进行频谱分析并绘图，从频域角度直观比较滤波前后的差异。。

直流陷波器的传递函数H(z)为：$H(z) = \frac{z-1}{z-a}$ 。

```
% 原始滤波操作
dc_filtered = filter([1 -1], [1 -notch_a], raw_ecg);

% 频谱分析
Y_dc = abs(fft(dc_filtered));
f = fs * (0:dataLen-1) / dataLen;
figure;
plot(f(1:dataLen/2), Y_dc(1:dataLen/2));
title('2. 去直流后的频谱');
xlabel('频率 (Hz)');
ylabel('幅度');

% 陷波器频率响应分析
[H, W] = freqz([1 -1], [1 -notch_a], 1024, fs);
mag = 20*log10(abs(H));          % 幅度（dB）
phase = angle(H) * 180/pi;       % 相位（角度）

% 找出零频（DC）处的增益
dc_gain = 20 * log10(abs(H(1)));

% 幅角-增益图绘制
figure;
yyaxis left;
plot(W, mag, 'b-', 'LineWidth', 1.5);
ylabel('幅度 (dB)');
yyaxis right;
plot(W, phase, 'r--', 'LineWidth', 1);
ylabel('相位 (度)');
xlabel('频率 (Hz)');
title('陷波器频率响应（幅度 & 相位）');
grid on;

% 标出 DC 点的衰减值
hold on;
plot(W(1), mag(1), 'ko', 'MarkerFaceColor', 'k');
text(W(1)+1, mag(1), sprintf('DC衰减: %.2f dB', dc_gain), 'FontSize', 10);
```

图 4.5：直流陷波，增益与相位特性计算

该数字滤波器系统的零点位于 z=1，极点位于 z=a，其中参数 a 是一个接近 1 的正实数。该结构构成了一个高通型陷波器，专用于抑制直流（零频）成分。由于零点正好位于单位圆上的 ω=0（即直流频率处），因此滤波器在该频率处产生显著的增益衰减。其衰减程度和滤波器的过渡带宽度主要由参数 a 控制：当 a 趋近于 1 时，极点靠近单位圆，使得直流分量的衰减减弱、过渡带变窄，有利于保留低频心电信息；而当 a 取值较小，则导致衰减增强但过渡带变宽，可能影响有效心电波形的保真度。综合实验效果和信号质量考虑，最终选取 a = 0.995 作为权衡，既有效抑制了基线漂移，又较好保留了心电信号的主要特征。

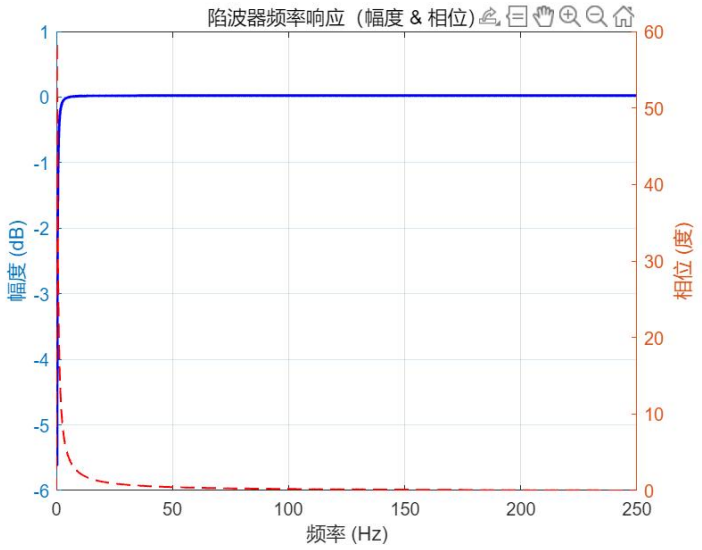

图 4.6：a=0.995 时直流陷波器 H(z)的增益和相位特性

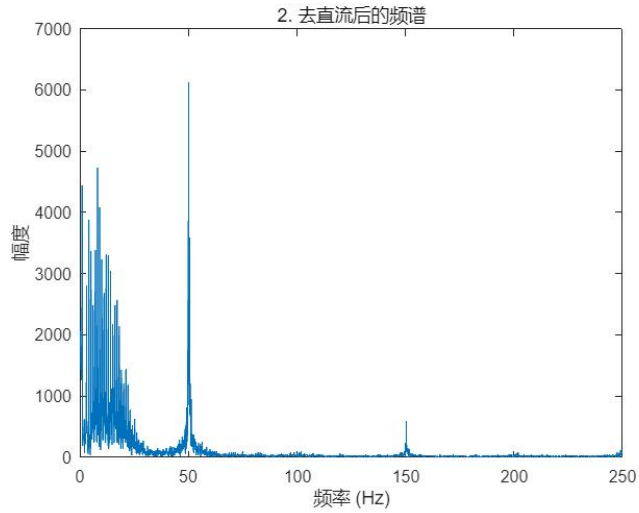

图 4.7：数字直流陷波器滤波后频谱分析

可以看到，在加入了直流陷波器进行滤波后，0 频处的基线漂移基本被消除。

4.3 FIR 数字低通滤波器设计与滤波效果

目标是对去除了直流分量的心电信号进行低通滤波，以去除高于 35Hz 的噪声成分（如肌电噪声），从而进一步净化信号，保留心电信号的关键频率成分（主要集中在 0.5–40Hz 范围）。为实现这一目标，我基于 Hamming 窗函数对滤波器系数进行了窗函数加权，以优化其频谱响应性能。

首先，通过设定过渡带宽度为 10Hz，并结合采样频率为 500Hz，计算得到归一化过渡带宽。利用经验公式 $N \approx \frac{4}{\Delta f}$ ，初步估算出所需滤波器的阶数 N，

并进一步调整为奇数阶以确保滤波器系数的对称性，从而具备线性相位特性。接着，使用 MATLAB 的 fir1 函数设计了一个 Hamming 窗加权的低通滤波器，其截止频率设定为 35Hz，对应归一化频率为 35/250=0.14。该滤波器能够有效抑制超过 35Hz 的高频噪声，同时对保留信号中关键的心电成分具有良好的保真度。随后将设计完成的 FIR 滤波器应用于之前去直流处理后的心电数据上，通过 filter 函数实现实际滤波过程。最后，采用快速傅里叶变换（FFT）对滤波后的信号进行频谱分析，并绘制出频谱图。

```
%% Step 4: FIR低通滤波器
delta_f = trans_width / fs;
N = ceil(4 / delta_f);              % Hamming窗经验公式
N = N + mod(N+1,2);                 % 保证是奇数阶
b = fir1(N-1, lp_cutoff/(fs/2), hamming(N));
filtered_ecg = filter(b, 1, dc_filtered);

Y_fir = abs(fft(filtered_ecg));
figure;
plot(f(1:dataLen/2), Y_fir(1:dataLen/2));
title('3. FIR滤波后频谱'); xlabel('频率 (Hz)'); ylabel('幅度');
```

图 4.8：FIR 数字低通滤波器设计

结果如下所示：直观展示滤波效果，能够明显观察到高频干扰成分已被有效衰减。50Hz 处的工频干扰完全被消除。

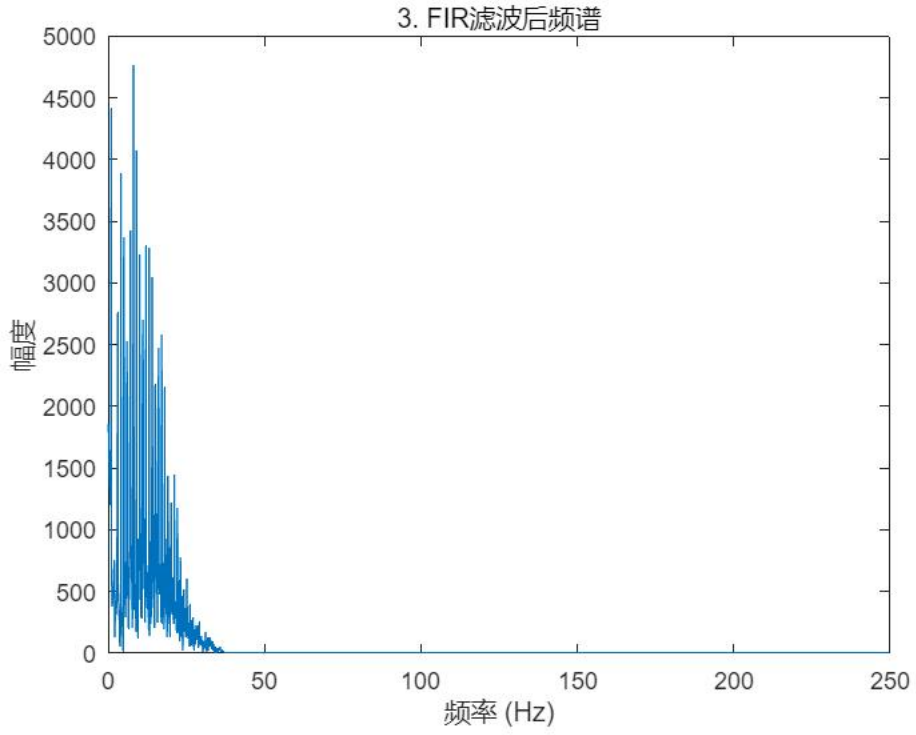

图 4.9：信号经 FIR 数字低通滤波后频谱分析绘制

为了直观看出阻带衰减不低于 40dB，过渡带不超过 10Hz；绘制出幅频响应与增益，结果显示满足设计要求。

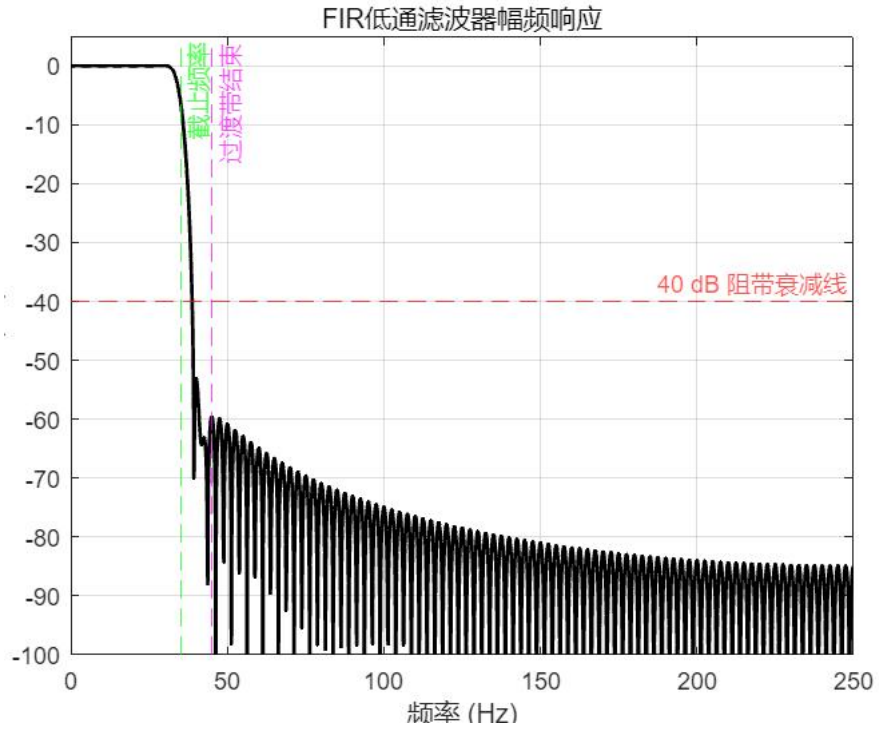

图 4.10：幅频响应

4.4 滤波结果综合分析

通过直流陷波和 FIR 滤波器的级联处理，最终获得了较为清晰的心电信号波形。

```
%% Step 5: 时域缩放至120动态范围
min_val = min(filtered_ecg);
max_val = max(filtered_ecg);
ecg_scaled = dynamic_range * (filtered_ecg - min_val) / (max_val - min_val);

figure;
plot(ecg_scaled); title('4. 缩放后的心电图波形');
xlabel('样点'); ylabel('幅值 (0~120)');
```

图 4.11 绘制波形

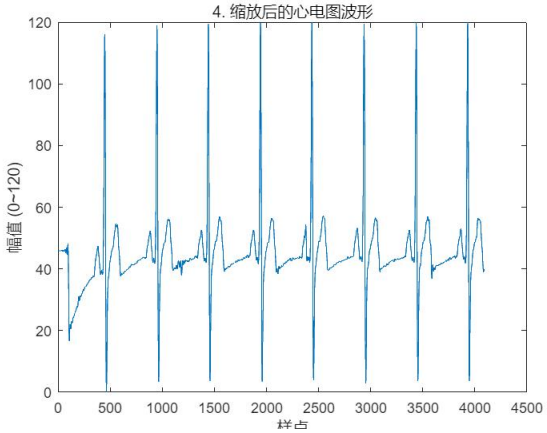

图 4.12 绘制波形

为了更直观地展示信号处理过程的效果，对心电信号在各个处理阶段的频谱进行了对比分析。图 3.10 中显示了原始心电信号以及经过直流陷波器和 FIR 低通滤波器联合处理后的信号频谱图。

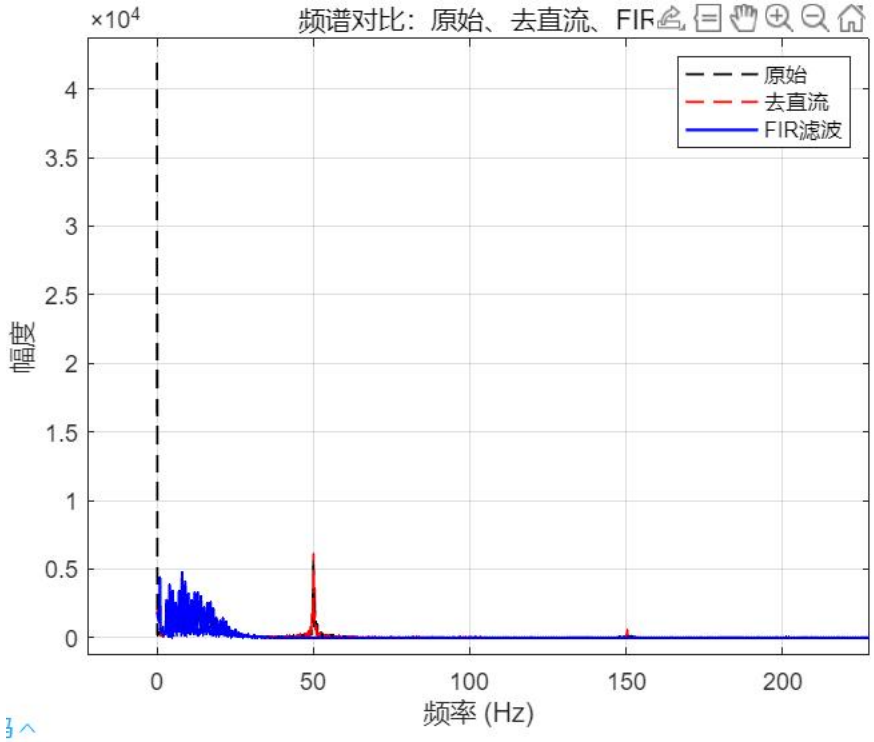

图 4.13 信号频谱对比图

经过直流陷波器处理后，心电信号中的低频基线漂移得到了明显抑制，信号的低频分量显著减少，但高频未变化。进一步应用 FIR 低通滤波器后，信号中的 50Hz 工频干扰被有效抑制，同时高频噪声也大幅削弱，整体信噪比明显提升。

最终处理后的频谱图显示出心电信号的主要频率成分集中在 30Hz 以下，波形清晰，为后续的 R 波提取与心率分析提供了干净且可靠的数据基础。

4.5 心率计算

通过检测心电信号中的 R 波位置来估算心率。首先，使用 `findpeaks` 函数在预处理后的心电信号 `ecg_scaled` 中查找峰值，设置峰值最小高度为 60，以及相邻峰值之间的最小间隔为 0.4 秒（对应心率上限约 150 bpm），以避免误检。检测到的 R 波位置存储在 `locs` 中。

接着，通过 `diff(locs)` 计算相邻 R 波之间的间隔（即 RR 间期），并除以采样频率 `fs` 将其转换为秒。然后利用公式 `heart_rate = 60 / RR间期` 计算每对 R 波之间的瞬时心率（单位为 bpm）。最后，使用 `mean` 函数对所有瞬时心率取平均，得到当前数据段的平均心率 `avg_hr`。

```
%% Step 6: 心率估算（R波检测）
[~, locs] = findpeaks(ecg_scaled, 'MinPeakHeight', 60, 'MinPeakDistance', round(0.4 * fs));
rr_intervals = diff(locs) / fs;       % 单位：秒
heart_rate = 60 ./ rr_intervals;      % bpm
avg_hr = mean(heart_rate);

figure;
plot(heart_rate, '-o'); grid on;
title(['5. 心率估算 (平均: ' num2str(avg_hr, '%.1f') ' bpm)']);
xlabel('节拍编号'); ylabel('心率 (bpm)');
```

图 4.14 MATLAB 心率计算代码

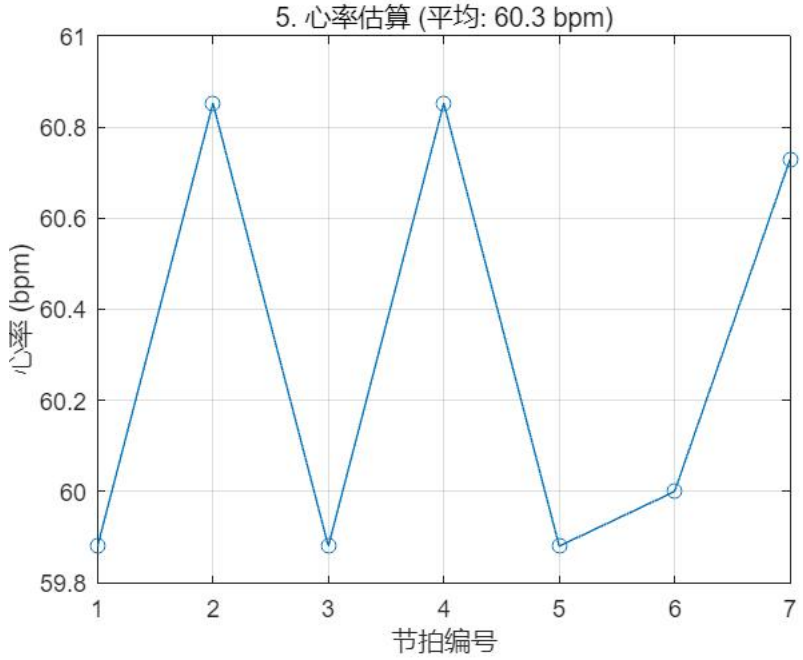

图 4.14 心率绘图

最终输 出结果为 60.3bpm，可以较为准确地估算心率。

5 程序设计与参数选择

5.1 单片机程序架构设计

通过 ADS1292R 模块实现心电（ECG）信号的采集，采用外部中断机制确保数据的实时性。采集到的原始数据首先经过 FIR 低通滤波以去除高频噪声，随后执行 R 波检测并计算心率，同时结合 DWT 计时器进行心率平滑和动态判断。当满足条件时，还会进行峰峰值计算和 FFT 频谱分析，用于实时监控信号质量。最终，所有处理结果通过 LCD 屏幕显示，并通过串口输出原始值与计算结果，形成一个完整的嵌入式心电监测系统闭环。

系统启动后，首先完成各模块的初始化：硬件驱动层初始化，ADS1292R 配置，配置滤波器参数，启动高精度定时器以便后续心跳间期测量，初始化并清空显示屏幕，设置显示方向，并准备好 FFT 和二阶滤波器所需的数据结构，同时

建立心率平滑所需的卡尔曼滤波器。所有准备工作就绪后，进入主循环，不断监测 ADC 数据就绪信号。一旦检测到新的 ECG 数据到达，系统立即读取两路原始采样，并将原始补码转换为有符号整数。随后对第一路信号进行低通滤波，去除高频噪声，然后进行 R 波检测：通过精确计时计算相邻心跳的时间间隔，将其转换为瞬时心率，并应用平滑与卡尔曼滤波来稳定输出。与此同时，滤波后数据被连续缓存，待累计到一定数量后，计算并显示峰峰值，并执行一次 FFT 分析以评估频谱特性。全程在屏幕上实时绘制双通道波形，并在显示区域更新当前心率，同时通过串口将关键数据（原始值、滤波结果、心率等）输出，为上位机记录和调试提供支持。整个流程将中断驱动的数据采集与主循环内的数据处理和可视化紧密结合，构成心电监测的完整闭环。

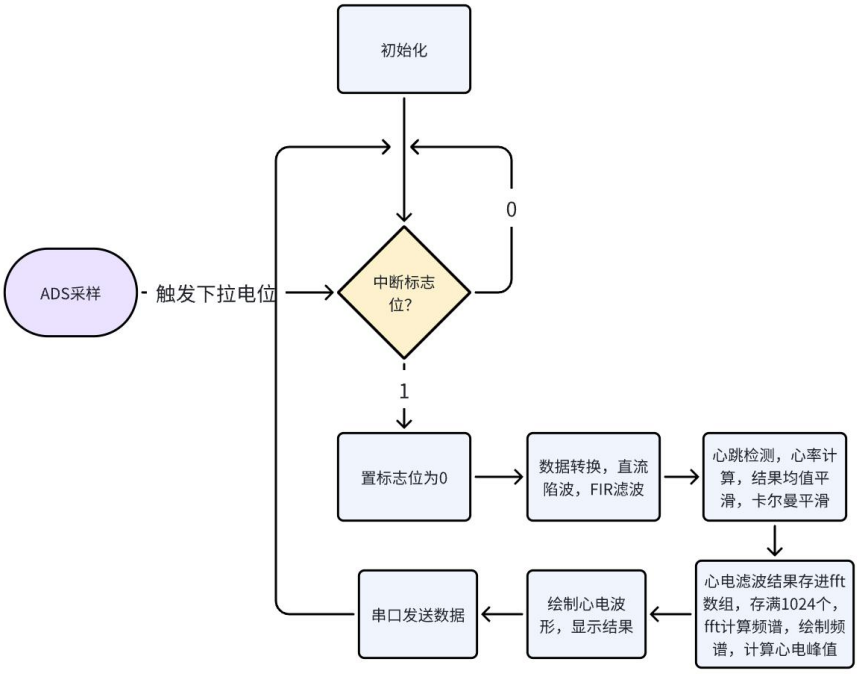

图 5.1 单片机程序工程基本工作框架

5.2 通信与数据预处理

每次采样中断到来后，首先执行数据读取函数，从 ADS1292R 中依次读取 9 字节数据。该过程通过片选信号拉低来启动，随后通过连续的 SPI 通信接收数据，接收完毕后拉高片选信号结束通信。

在数据读取完成后，进行数据拼接与裁剪的预处理操作。ADS1292R 采集的每个通道 ECG 数据为 24 位宽度，数据由三个连续的字节构成。程序将三字节数据通过移位与或操作拼接成一个 24 位整型，并进一步右移 8 位，仅保留高 16 位有效数据。此操作可视为舍弃最低 8 位以降低噪声敏感性，提升信噪比。随后对这两个通道的数据分别强制类型转换为带符号的 16 位整数，以确保其在后续处理中能正确表达正负电压变化。转换完成后，进一步将通道一的原始数字量（范围为 ±32767）乘以满量程电压 2.42V 与缩放因子进行归一化，转换为真实物理电压值（单位：伏特）。

```
void data_trans(void)
{
    for(uint8_t n=0;n<2;n++)//单次采集的CH1和CH2数据存入ADS1292R_ECG_BUF
    {
        ADS1292R_ECG_BUFFER[n]=ads1292r_data_buff[3+3*n];
        ADS1292R_ECG_BUFFER[n]=ADS1292R_ECG_BUFFER[n]<<8;
        ADS1292R_ECG_BUFFER[n]|=ads1292r_data_buff[3+3*n+1];
        ADS1292R_ECG_BUFFER[n]=ADS1292R_ECG_BUFFER[n]<<8;
        ADS1292R_ECG_BUFFER[n]|=ads1292r_data_buff[3+3*n+2];
    }

    ADS1292R_ECG_BUFFER[0]=ADS1292R_ECG_BUFFER[0]>>8;//舍去低8位
    ADS1292R_ECG_BUFFER[1]=ADS1292R_ECG_BUFFER[1]>>8;

    ADS1292R_ECG_BUFFER[0]&=0xffff;//消除补码算数移位对数据影响
    ADS1292R_ECG_BUFFER[1]&=0xffff;

    ECGRawData[0] = (int16_t)ADS1292R_ECG_BUFFER[0];
    ECGRawData[1] = (int16_t)ADS1292R_ECG_BUFFER[1];
    ecg_vol = ECGRawData[0]*2.42f/32767.0f;
    //update_ecg_peak(ecg_vol);

    //printf("%d\r\n",ECGRawData[0]);
}
```

图 5.2 ADS1292R 原始数据解码核心代码

5.3 滤波器设计

数据预处理好后，先通过 IIR 滤波器消除直流分量，然后利用 FIR 滤波器进一步去除高频噪声。经过 IIR 和 FIR 滤波后的信号显著改善，直流漂移、工频干扰以及高频噪声均被有效抑制。

5.3.1 IIR 数字直流陷波器

在 IIR 滤波阶段，STM32 程序借鉴了 PC 端设计的滤波器结构，实现了高效的实时信号处理。该滤波器采用递归形式的差分方程进行计算，核心思想是利用当前输入值与前一时刻的输入、输出值之间的关系来更新当前输出。具体实现中，系数 a = 0.995 控制了直流分量的衰减速度，使得信号中的低频漂移得以逐步滤除。由于该方法仅依赖少量的历史数据和固定的系数计算，因此在嵌入式环境下既节省存储空间，又具备较高的计算效率，能够实现对 ECG 信号中直流漂移的稳定、实时抑制。

```
//IIR滤波器相关参数
float a = 0.995;
int last_input = 0;
int last_output = 0;
uint16_t cur_input = 0;
int cur_output = 0;
int16_t IIR_Result;

void IIRFilter(uint16_t rawData){
    cur_input = rawData;
    cur_output = cur_input - last_input + a * last_output;
    IIR_Result = (int16_t)cur_output;
    //printf("%d\n",IIR_Result);
    last_output = cur_output;
    last_input = cur_input;
}
```

图 5.4：IIR 数字直流陷波器

5.3.2 FIR 数字低通滤波器

接下来使用 FIR 滤除工频干扰和其他高频噪声。FIR 滤波器采用固定长度的滑动窗口结构，参考上章 matlab 设计，通过 C 语言代码实现。为了适配嵌入式系统中定点运算的需求，滤波系数在初始化阶段被统一放大（如乘以 65536），并转换为 16 位整数格式，便于高效运算。

FIR 滤波的核心过程是：

在初始化函数中，首先将滤波队列清零，并填入固定数量的初始值（零值）以准备滤波过程。随后将滤波器系数按照定点运算规则进行缩放（乘以 65536）并保存在整型数组中，以提高运算效率并适配嵌入式环境的计算能力。

在实际滤波过程中，每当输入一个新的采样值，函数首先会判断该值是否与上一次输入相同，以避免对连续重复值进行冗余计算。若为新值，则通过队列弹出最旧的数据，插入当前采样值，维护一个恒定长度的滑动窗口。

随后，函数遍历整个队列，与滤波系数数组中对应位置的系数进行乘法运算并累加，实现滑动窗口的加权求和。由于该队列结构是循环队列，需分情况处理队首与队尾的位置关系，以确保所有数据按时间顺序与系数正确配对。

最后，计算出的累加值乘以比例因子 coefficient，输出为滤波后的结果 FIRResult。整个算法在结构上保证了线性相位、稳定性和实时性，适用于生物电信号等对滤波精度和延时要求较高的应用场景。滤波后，处理得到的心电信号 FIRResult 用于后续处理。

```
const double coefficient = 4.0;
struct queue FIRQueue;
int16_t FIRResult;
int FIRA[hLength];
const double B[161] = {
        -1.104587824635e-06,-3.30765288084e-07,6.469557905197e-07,2.354792370333e-06,
        4.613259754644e-06,6.877479993928e-06,8.237049198786e-06,7.558936626691e-06,
        3.788971148408e-06,-3.623848847902e-06,-1.427538238554e-05,-2.647155435505e-05,
        -3.717488350579e-05,-4.236786223392e-05,-3.788474498342e-05,-2.063216063947e-05,
        1.003324496041e-05,5.113408663599e-05,9.553680832838e-05,0.0001324352646621,
        0.0001490628825309,0.0001335519048098,7.850129689465e-05,-1.552301519343e-05,
        -0.000137547975341,-0.0002658042593173,-0.0003701010370572,-0.0004171261821494,
        -0.0003780894912395,-0.0002373438145375,-1.701986658015e-18,0.0003036972242474,
        0.000619188232845,0.0008747139088742,0.0009944873077503, 0.000916089676601,
        0.0006087713144456,8.835700253944e-05,-0.0005756611688794,-0.001264303369241,
        -0.00182600060318,-0.002104876596178,-0.001976059132852,-0.001381105279486,
        -0.0003552282206677,0.0009616648412149,  0.00233583791841, 0.003474169866146,
        0.004078695334442, 0.003912505763598, 0.002864124412349, 0.000995570008819,
        -0.001439527443595,-0.004016978287615,-0.006204281185006,-0.007457327047886,
        -0.00733619066631,-0.005618081778844,-0.002382762583926, 0.001952112154742,
        0.006661024610152,  0.01080982348891,  0.01341421446485,  0.01363410556943,
        0.01097007287533, 0.005423497886492,-0.002415610166809, -0.01137922416383,
        -0.01984044191607, -0.02593171901878, -0.02783322176153, -0.02408663464818,
        -0.01388122282339, 0.002740281371631,  0.02479787236787,  0.05043558413653,
        0.07712685401238,   0.1019977732739,   0.1222205837105,     0.13541600842,
        0.1399996906768,     0.13541600842,   0.1222205837105,   0.1019977732739,
        0.07712685401238,  0.05043558413653,  0.02479787236787, 0.002740281371631,
        -0.01388122282339, -0.02408663464818, -0.02783322176153, -0.02593171901878,
        -0.01984044191607, -0.01137922416383, -0.002415610166809, 0.005423497886492,
```

图 5.5：部分系数矩阵

5.4 心率与幅值计算

5.4.1 心率测量

系统的心跳检测功能基于心电信号波形的动态变化，采用阈值法识别 R 波峰值，实现高效的实时检测。核心检测函数通过持续跟踪输入信号的变化趋势，动态维护波峰与波谷的值。初始状态下，将输入值同时设为波峰与波谷；在后续数据处理过程中，若信号持续下降，则更新波谷值；当信号转为上升趋势时，会判断此次波峰与波谷之间的差值是否超过设定的阈值（代码中默认 50，对应 ECG 信号变化约为 0.55 mV）。若满足该条件，判定为一次有效心跳事件，函数返回 1，同时重置波峰和波谷值，准备进入下一次检测周期。若不满足条件，则认为是噪声或非心跳扰动，函数返回 0。

```
static int heartbeat_thresh = 50; //心跳下降沿阈值，下降沿高度大于该阈值视为一次心跳
static int is_init = 0;
static int up_value = 0;// 波峰数值
static int down_value = 0;// 波谷数值

if (!is_init) {
    is_init = 1;
    up_value = value; down_value = value; //首先将当前值视为波峰，波谷
    return 0;
}
if (value <= down_value) { //波形正在下降
    down_value = value;
} else { //波形正在上升，说明检测到波谷
    if ((up_value - down_value) >= heartbeat_thresh) { //检测到心跳
        up_value = value; down_value = value; //将当前值重新时为波峰，波谷，准备下一次检测心跳
        return 1;
    } else {
        up_value = value; down_value = value; //将当前值重新时为波峰，波谷，准备下一次检测心跳
    }
}
return 0;
```

图 5.6：心跳检测

系统利用高精度 DWT 定时器模块记录每次心跳发生的时间间隔（R-R 间期），结合采样率对心率进行计算。若当前未检测到心跳，系统将时间戳累加；一旦检测到心跳事件，即可通过时间间隔反推当前心率（bpm）。

```
if (heart_beat == 1)
  {
    HAL_GPIO_WritePin(GPIOF, GPIO_PIN_8, GPIO_PIN_SET);     // 蜂鸣器停
    float dt = DWT_GetDeltaT(&DWT_CNT1);
    if (dt > 0.2f && dt < 8.0f)
      {  // 限制间隔避免误触发
        new_hr = 60.0f / dt;
        heart_rate = smooth_heart_rate(new_hr);
        last_hr = heart_rate;
        if(new_hr>80)
        {
            heart_rate = CalcKalmanFilter(&KalmanFilter_heart_rate, (float) heart_rate);
        }
        heart_rate= roundf(heart_rate);  // 直接四舍五入
      }
  }
```

图 5.7：心率检测结果滤波

为实现平稳的心率估算，系统还引入了心率平滑处理机制。每次检测到心跳后，会将当前计算得到的心率值写入一个固定长度的缓冲区，并实时更新该区域中的平均值，以抑制单次误差或瞬时波动对最终结果的影响，从而输出一个更平稳可靠的心率值。

```
heart_rate_buffer[hr_index++] = new_hr;
if (hr_index >= HR_BUFFER_SIZE) hr_index = 0;

if (hr_count < HR_BUFFER_SIZE) hr_count++;  //

float sum = 0;
for (int i = 0; i < hr_count; i++) {
    sum += heart_rate_buffer[i];
}

float result = sum / hr_count;
return result;
```

图 5.8：滑动窗口平滑滤波

在心率异常偏高时（如新心率值超过 80 BPM），心率测量误差开始变大，数值跳动变大。系统还引入卡尔曼滤波器进一步优化输出结果，融合当前测量值与历史估计值，降低随机扰动的影响，最终实现准确、稳定且实时的心跳识别与心率计算。

```
void InitKalmanFilter(KALMAN_FILTER *pKalmanFilter)
{
    pKalmanFilter->xk_1 = 0;
    pKalmanFilter->Pk_1 = 1;
    pKalmanFilter->xkk_1 = 0;
    pKalmanFilter->Pkk_1 = 0;
    pKalmanFilter->Kk = 0;
    pKalmanFilter->xk = 0;
    pKalmanFilter->Pk = 0;
    pKalmanFilter->Q = 0.001f;
    pKalmanFilter->R = 0.1f;
}
```

图 5.9：一阶卡尔曼滤波参数

```
float CalcKalmanFilter(KALMAN_FILTER *pKalmanFilter, float zk)
{
    //prediction
    pKalmanFilter->xkk_1 = pKalmanFilter->xk_1;
    pKalmanFilter->Pkk_1 = pKalmanFilter->Pk_1 + pKalmanFilter->Q;

    //correction
    pKalmanFilter->Kk = pKalmanFilter->Pkk_1 / (pKalmanFilter->Pkk_1 + pKalmanFilter->R);
    pKalmanFilter->xk = pKalmanFilter->xkk_1 + pKalmanFilter->Kk * (zk - pKalmanFilter->xkk_1);
    pKalmanFilter->Pk = (1.0f - pKalmanFilter->Kk) * pKalmanFilter->Pkk_1;

    pKalmanFilter->xk_1 = pKalmanFilter->xk;
    pKalmanFilter->Pk_1 = pKalmanFilter->Pk;

    return pKalmanFilter->xk;
}
```

图 5.10：卡尔曼滤波计算

整体流程逻辑清晰、计算效率高，适用于对实时性和准确性要求较高的嵌入式生理监测系统中。计算结果误差为零，即使是

5.4.2 幅值测量

在采集过程中，每当一个新的滤波后信号值 FIRResult 被获得，系统会判断当前记录数量是否小于预设的最大采样点数 ECG_COUNT。若未满，当前值将存入心电信号数组 ECG_Signal，并实时参与最大值 ecg_pk_max 和最小值 ecg_pk_min 的更新过程。最大最小值初始化分别为 -32768 和 32767，以确保任何信号值都能正确更新。

当采集满 ECG_COUNT 个点后，系统自动触发幅值计算过程。首先将当前采样周期内的最大值与最小值之差计算为电压单位下的峰峰值。由于原始信号单位为 ADC 的数字量（LSB），代码使用一个预先计算的转换系数 mv_per_unit（约为 0.0000481 mV/LSB）将该差值转换为毫伏单位。为适配硬件特性和校准误差，最终输出值还乘以 1.25 并减去一个微小偏移（0.001），再乘以 10 放大以增强显示精度。

计算结果通过格式化为字符串（保留两位小数）后，在 LCD 显示屏上以 "P-P: X.XXV*e-2" 的形式输出，直观呈现一个周期内的幅度变化范围。随后，最大值与最小值变量被重置，为下一次采样周期做准备。

```
ECG_Signal[ecg_num] = FIRResult; //□□¾□□□□
IIR_Result =  Second_Order_TF_Calculate(&tf,FIRResult);
ecg_num++;
if (FIRResult > ecg_pk_max) ecg_pk_max = FIRResult;
if (FIRResult < ecg_pk_min) ecg_pk_min = FIRResult;
```

图 5.11：比较更新

```
ecg_num = 0;
float mv_per_unit = (2.42f / 6.0f) * 1000.0f / 8388608.0f;  // ≈ 0.0000481 mV/LSB
// 计算峰峰值
float peak_to_peak_mv = (ecg_pk_max - ecg_pk_min) * mv_per_unit;
```

图 5.12：计算峰值

经测量幅值误差基本为 0。

5.5 频谱分析

主循环中，为了保证 FFT 计算稳定，将在收满 1024 个滤波数据后进行 FFT 计算。

```
if(ecg_num < ECG_COUNT)
{
  ECG_Signal[ecg_num] = FIRResult; //□□¾□□□□
  IIR_Result =  Second_Order_TF_Calculate(&tf,FIRResult);
  ecg_num++;
  if (FIRResult > ecg_pk_max) ecg_pk_max = FIRResult;
  if (FIRResult < ecg_pk_min) ecg_pk_min = FIRResult;
}
else
{
  ecg_num = 0;
  float mv_per_unit = (2.42f / 6.0f) * 1000.0f / 8388608.0f;  // ≈ 0.0000481 mV/LSB
  // 计算峰峰值
  float peak_to_peak_mv = (ecg_pk_max - ecg_pk_min) * mv_per_unit;
  // 显示字符串（保留 3 位小数较合适）
  char peakStr[32];
  sprintf(peakStr, "P-P: %.2fmV*e-2", (peak_to_peak_mv*1.25-0.001)*10);
          lcd_show_string(10, 10, 160, 16, 16, peakStr, WHITE);
// 重置峰值
  ecg_pk_max = -32768;
  ecg_pk_min = 32767;
  fftcal();
  drawFFT();
```

图 5.12：主程序调用

程序使用CMSIS-DSP库提供的高效FFT算法实现心电信号的频域分析。首先，将滤波后的心电信号数据打包为复数输入，其中实部为滤波后的信号值，虚部统一置为零，构成复数数组。随后，调用 arm_cfft_radix4_f32 函数执行快速傅里叶变换（FFT），将时域信号转换为频域复数频谱。通过 arm_cmplx_mag_f32 函数计算频谱的幅值，即复数频谱的模值，结果存储于 FFT 输出缓冲区中，作为后续频谱绘制的输入。

```
void fftcal()
{
    for(int i=0; i < FFT_LENGTH; i++)
{
    FFT_InputBufmy[2*i]=(uint16_t)(ECG_Signal[i])/20;
    FFT_InputBufmy[2*i+1]=0;
}
arm_cfft_radix4_f32(&scfft,FFT_InputBufmy);
arm_cmplx_mag_f32(FFT_InputBufmy,FFT_OutputBufmy,FFT_LENGTH);

  for(int i = 0;i<FFT_LENGTH;i++)
  {
    FFT_OutputBufmy[i] =FFT_OutputBufmy[i]*0.36*0.1;
  }
  float max_value1 = 0, max_value2 = 0;
    int max_index1 = 0, max_index2 = 0;

    for (int i = 1; i < FFT_LENGTH / 2; i++) {
        if (FFT_OutputBufmy[i] > max_value1) {
            max_value2 = max_value1;
            max_index2 = max_index1;
            max_value1 = FFT_OutputBufmy[i];
            max_index1 = i;
        } else if (FFT_OutputBufmy[i] > max_value2) {
            max_value2 = FFT_OutputBufmy[i];
            max_index2 = i;
        }
    }
  fre=(float)max_index1 * 500/ FFT_LENGTH;
}
```

图 5.12：FFT 计算

5.6 TFT 屏幕绘制波形，测量数据，频谱

5.6.1 绘制波形

主要绘制滤波后心电信号以及 ADS1292R 测试波形，前者主要依靠数据 FIRResult，后者依靠 ADS1292R 通道二的原始数据，（注意这里通道二未接心电模拟仪，数据波形为原始方波，该方波用于测试 ADS1292R 通信，工作是否正常等），这里同时绘制两个波形。

波形绘制函数实现原理：接收两个整数值 FIRResult、ECGRawData[0]作为输入，分别代表两条曲线当前点的纵坐标。首先，代码根据屏幕高度和输入值，将这两个值映射到屏幕的 Y 轴坐标范围内，并确保坐标不会超出屏幕边界。然后，根据是否是第一次绘制点，程序决定是直接画出起始点，还是在上一次绘制点的基础上绘制一条连接前后两个点的线段。两条曲线分别用白色和绿色绘制，X 轴坐标随着每次绘制递增。当绘制到屏幕右边界时，屏幕会被清空，绘制位置重新回到起点，实现曲线的循环刷新。保证了双通道数据的实时连续显示，形成动态的波形曲线。

```
void drawDualCurve(int16_t value1, int16_t value2)
{
    uint16_t x, y1, y2;
    // 将数据映射到屏幕坐标上（可根据实际调整）
    y1 = (LCD_HEIGHT - 10) - value1 * (LCD_HEIGHT - 20) / 500 - 200;
    y2 = (LCD_HEIGHT - 10) - value2 * (LCD_HEIGHT - 20) / 500 - 200;
    // 限制坐标在屏幕范围内
    if (y1 > LCD_HEIGHT - 1) y1 = LCD_HEIGHT - 1;
    if (y2 > LCD_HEIGHT - 1) y2 = LCD_HEIGHT - 1;
    if (y1 < 0) y1 = 0;
    if (y2 < 0) y2 = 0;
    if (firstPointDual)
    {
        lcd_draw_point(10, y1, WHITE);  // 第一条曲线（白色）
        lcd_draw_point(10, y2, GREEN);  // 第二条曲线（绿色）
        lastX1 = lastX2 = 10;
        lastY1 = y1;
        lastY2 = y2;
        firstPointDual = 0;
    }
    else
    {
        x = lastX1 + 1;
        if (x < LCD_WIDTH - 10)
        {
            lcd_draw_line(lastX1, lastY1, x, y1, WHITE);  // 白色线
            lcd_draw_line(lastX2, lastY2, x, y2, GREEN);  // 绿色线

            lastX1 = lastX2 = x;
            lastY1 = y1;
            lastY2 = y2;
        }
        else
        {
            lcd_clear(BLACK);
            lcd_draw_point(10, y1, WHITE);
            lcd_draw_point(10, y2, GREEN);
            lastX1 = lastX2 = 10;
            lastY1 = y1;
            lastY2 = y2;
        }
    }
}
```

图 5.12：波形绘制

5.6.1 绘制频谱

函数 drawFFT()从全局缓冲区 FFT_OutputBufmy 中读取频域幅值数据，并将其中前 numBars（例如 50 个）频率分量以柱形图方式显示。每个柱代表一个频率段的幅值，其高度与该频率分量的幅度成比例。

首先设定了柱状图的总数、最大柱高以及绘图区域的水平范围。接着，通过计算每根柱之间的水平间距，将绘图区等分。为确保柱形图的相对高度具有良好的对比度，程序会在绘制前先扫描所有频率分量，找出其最大幅值（排除直流分量），作为归一化参考值。然后对每个频率分量进行归一化处理，使得最大幅值对应最大柱高，其余按比例缩放。

绘图时，从左向右依次绘制每个柱形，柱体从底部基线向上绘制，高度与对应频率分量幅值成正比，颜色统一为青色。为了方便阅读和定位频率分布，程序还在每隔若干个柱子下方显示对应的频率索引标签。

```c
void drawFFT(void)
{
    int numBars = 50;
    int maxBarHeight = 60;
    int startX = 10;
    int endX = LCD_WIDTH - 10;
    int spacing = (endX - startX) / numBars;

    int baseY = LCD_HEIGHT - 40;  // ?? 向上移动 30 像素，腾出底部空间

    // 查找最大值（排除 DC 分量）
    float maxValue = 1.0f;
    for (int i = 1; i <= numBars; i++) {
        if (FFT_OutputBufmy[i] > maxValue) {
            maxValue = FFT_OutputBufmy[i];
        }
    }

    for (int i = 1; i <= numBars; i++) {
        float normalized = FFT_OutputBufmy[i] / maxValue;
        if (normalized > 1.0f) normalized = 1.0f;

        int barHeight = (int)(normalized * maxBarHeight);
        int x = startX + (i - 1) * spacing;

        // 绘制频谱柱
        lcd_draw_line(x, baseY, x, baseY - barHeight, CYAN);

        // 每5个柱显示一次索引
        if (i % 5 == 0) {
            char label[4];
            sprintf(label, "%d", i);
            lcd_show_string(x - 6, baseY + 4, 20, 12, 12, label, WHITE);  // 标签向下显示
        }
    }
```

图 5.12：频谱绘制

6 程序测试方法

6.1 ADS1292R 驱动验证

ADS1292R 对通道 1 和通道 2 进行配置。其中，通道 1 的 CH1SET 寄存器设置为 0x00，表示通道 1 工作于正常采集模式，且未启用测试信号输入。通道 2 的 CH2SET 寄存器配置为 0x05，用于指定通道 2 采集来自内部测试信号的方波，方便在调试阶段验证系统的采集和传输功能。

通过上述配置，ADS1292R 可以稳定运行于双通道心电采集模式，通道 1 用于实时采集患者的心电信号，通道 2 可用于采集 ADS1292R 传感器的测试信号以验证该传感器模块是否被正常驱动。

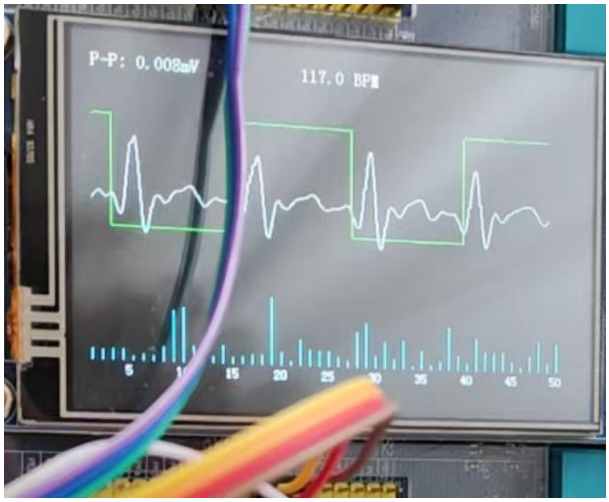

图 6.1：信号以及方波

6.2 ADS1292R 读取传感器设备 ID

对于 ADS1292R，若读取到的设备 ID 不为 73（ADS1292）或 115（ADS1292R）则会一直循环读取 ID 而不进行后续的初始化，以确保读取的心电信号不会出现无效数据且采集数据符合预期格式。由于本次项目使用的传感器芯片为 ADS1292R，故成功读取到并在串口助手中打印的 device_id 为 115，这也证明驱动库文件的移植是正确的且能正常驱动该传感器模块。

7 实验数据记录与分析

7.1 模拟器心电信号采集

将心电信号时域与频域的动态变化以实时曲线的形式绘制在屏幕上。同时，在屏幕上显示心率、波幅、频谱、测试方波等关键参数。其效果如下图所示：

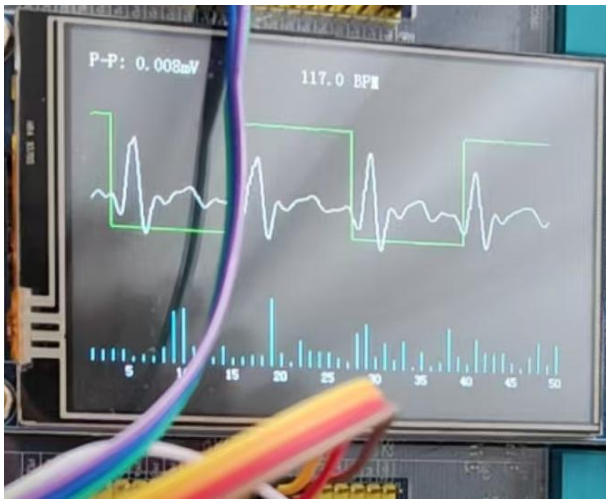

图 7.1：BPM 为 117 的心电信号（模拟器产生）在 LCD 屏幕上的显示效果

经过测试，测算出的心率值与心电模拟仪中给出的心率误差为 0（每分钟心跳数），满足系统设计需求；

7.2 模拟器心率信号采集

将模拟器调至 2 档（心率档），产生的心率信号默认为 75BPM，通过手动调节，使其为 65。切换模式 8，调节幅值为 100 即 1mv，此时时域与频域波形的显示效果如下：

图 7.2：（BPM 为 75、幅值为 100）在 LCD 屏幕上的显示效果

可以看到，此时测得的心率完全准确，误差为 0。且频谱图也有较明显的尖峰且噪声较少；幅值测量为 0.10*10e-2=0.001v 即是 1mv，完全正确，没有误差。

不调节心率，继续调节幅值为 160，即为 1.6mv 等待稳定后时时域与频域波形的显示效果如下：

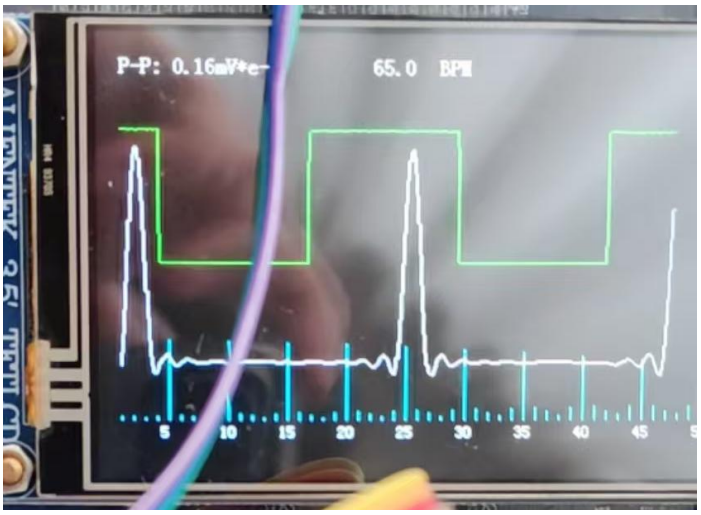

图 7.3：幅值为 1.6mv 的心率信号

实验结果显示，随着输入心率信号幅度的变化，系统测得的波形幅值也呈相应比例变化，频谱图表现出的纯净性，说明心电采集系统对不同强度信号具有良好的灵敏性和响应能力。

此外，在心率检测程序中，为了提升对微弱信号的识别能力，跳变检测所使用的阈值被设置得较低。但在信号幅值显著增大时，系统对高幅度震荡信号的响应也随之增强，易将非真实跳变误判为心跳，从而引入心率计算的误差。该问题表明，系统在处理大幅度信号时需进一步优化阈值判定逻辑，以提高心率测量的准确性和鲁棒性。

7.3 人体心电信号采集与测量

在实际测试时，将电极片分别贴于左右胸下和腿部，并分别与数据线连接，此时系统采集到的心电波形如下所示：

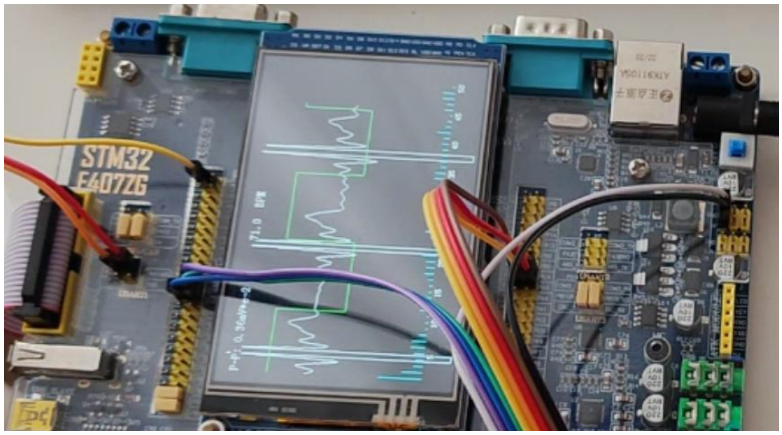

图 7.4：采集到的人体心电信号时域与频域波形显示结果

从图中可以看出，检测人体心电信号时，峰值的大小纯在差异，导致波形显示在 TFT 屏幕上超出范围。于是使用 PC 端进行查看，结果如下：

图 7.5：人体心电信号上位机显示

可以看出波形较好，且测量稳定，根据波形心率计算得 71，与上位机计算一样。但是值得注意的是，不同时间测量的效果不同，建议快速佩戴测量，尽量在 5 分钟内。

8 实验结果总结和心得体会

8.1 实验结果

在本项目中，基于 STM32F407ZG 主控芯片与 ADS1292R 心电信号采集模块，成功实现了对人体心电信号的实时采集、处理与可视化分析。针对心电信号中常见的工频干扰和低频漂移问题，系统设计并实现了 IIR 数字陷波器与 FIR 数字低通滤波器，显著提升了波形的清晰度与有效性。滤波器的设计最初在 MATLAB 环境下进行仿真，效果良好。在嵌入式系统中实现后，测量结果误差基本为 0。

项目开发过程中，借助心电信号模拟仪对整个数据采集链进行了调试验证，确保系统能够稳定采集并实时显示心电波形。系统还具备对 ADS1292R 内部测试信号的读取与分析能力，用于在调试阶段检验模块是否正常工作。最终，系统成功采集并显示了实际人体心电信号，完成了时域波形绘制与频域分析功能，心率与峰峰值的计算结果基本为 0，满足设计目标。

8.2 心得体会

通过本次项目实践，我深入掌握了数字信号的采集流程、滤波算法的设计实现以及嵌入式系统的整体开发方法。特别是在滤波器设计方面，不仅加深了我对 IIR 和 FIR 滤波原理的理解，也让我意识到理论仿真与嵌入式实现之间的差异，促使我更注重算法参数与系统资源之间的平衡。

同时，在 STM32F4 系列开发板的应用中，我对其硬件资源配置、外设初始化、ADC 采集与 DMA 传输等关键环节有了较为全面的认识。通过调试过程不断排查并解决问题，显著提升了我在单片机程序编写、系统集成与故障诊断方面的能力。

整个项目开发的过程充满了挑战，同时也伴随着大量的收获。我深刻体会到硬件开发与数字信号处理是一个不断探索和优化的过程，而这次项目实践无疑为我在这些领域的技能提升打下了坚实基础。

此外，本次项目还培养了我独立分析和解决问题的能力。从模块功能的实现到整体工程的构建，我逐步熟悉了完整的开发流程，对系统的设计、调试和优化有了更加系统化的认识。我相信，这些经验和能力将在今后的学习和工作中发挥重要作用，为更高层次的开发任务奠定基础。

参考资料

[1] 程佩青编著.数字信号处理教程[M].清华大学出版社,2015:524.
[2] 德州仪器. ADS1292 数据手册
[3] 朱冰莲,方敏编著.数字信号处理[M].电子工业出版社,2014:276.
[4] 任勇,曾浩编著.单片机原理及应用[M].清华大学出版社,2023.
[5] 任勇. CQU_S12XDEV 开发板原理图 微电子与通信工程学院
[6] 任勇. ADS1292-心电信号采集原理图及接口说明-RY