南昌航空大学

基于 Qt 设计的

生理信号监护系统

项目名称：基于 Qt 设计的生理信号监护系统

年　　级：大　三

姓　　名：李 奕 澄

2022 年五月

目录

摘要

本项目是在 Windows10 上，基于 Qt 5.12.0 设计程序，实现生理信号监护系统，其可以针对发送的数据进行数据解包、分类并实时显示在程序设计的界面上，并且可以进行一定程度上的用户交互。本报告包括设计要求、设计过程（界面设计、数据包结构、解包算法、显示算法）、设计结果、心得体会。

前言

本分析报告对该项目的前期内容进行分析，对该程序的实现方式进行分析，此说明书针对该项目目前情况进行概括，以便让读者快速了解该项目。

程序分析是对已完成的程序进行分析，通过开发人员的分析概括，整理成完整的分析报告，并形成文档的过程。分析报告应当表述出该程序的功能和原理，确保最终程序可以运行以及修改。

程序分析是一件严谨且重要的工作，对程序分析的是否完整透彻将直接影响程序的开发和质量。在一般情况下，开发者和读者之间编程知识以及逻辑并不一致，其将会为读者带来极大的困难而导致无法理解程序运行逻辑。因此需要通过程序分析。而读者在理解程序时通常是不完整、不准确的，因此不仅需要将程序的进行分析总结，并要以通俗易懂的的语言进行描述。

程序分析不仅是处于程序开发的早期工作，而是随着程序的开发进度以及后期编程遇到的问题而不断的更新与改进的，旨在最后程序开发完成时可以在绝大多数环境下运行。

最后，为了后续的评审、修改等工作，程序分析的描述应当做到具体详细，并具备可以修改的特性。

设计要求

(1) 利用 QT 设计一个生理信号监护系统;

(2) 数据来源，未解包数据.csv;

(3) 要求程序可以打开该文件并读取数据;

(4) 使用特定方式对数据进行解包;

(5) 设要求的界面显示波形和数据。

设计过程

界面设计

Mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QLabel>
#include "formsetuart.h"
#include "QSerialPort"
#include "QSerialPortInfo"
#include "packunpack.h"
#include <QMutex>
#include <QTimer>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT
```

```
public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

    void uiInit();

    PackUnpack *mPackUnpack; //打包解包类

protected:
    void timerEvent(QTimerEvent *event); //重载定时器方法
    bool eventFilter(QObject *obj, QEvent *ev);

private slots:
    void menuSetUART();
    void menuOpen();
    void initSerial(QString portNum, int baudRate, int dataBits, int stopBits, int parity,
bool isOpened);
    void readSerial();                  //串口数据读取
    void writeSerial(QByteArray data); //串口数据写入

    //体温设置
    void recPrbType(QString type, QByteArray data);

    //血压设置
    void recNIBPSetData(QString mode);

    //呼吸设置
    void recRespGain(QString gain, QByteArray data);

    //血氧设置
    void recSPO2Data(QString sens, QByteArray data);

    //心电设置
    void recECGData(int lead1, int gain1, int lead2, int gain2, QByteArray data);

private:
    Ui::MainWindow *ui;
    QStringList mCSVList;

    QLabel *mFirstStatusLabel;
    QSerialPort *mSerialPort;

    QByteArray mRxData; //接收数据暂存
```

```
    QByteArray mTxData; //发送数据暂存
    QList<uchar> mPackAfterUnpackList;  //解包后的数据
    uchar **mPackAfterUnpackArr;          //定义一个二维数组作为接收解包后的数据缓冲
    int mPackHead;    //当前需要处理的缓冲包的序号
    int mPackTail;    //最后处理的缓冲包的序号，mPackHead 不能追上 mPackTail,追上的话表示收到的数据超出 2000 的缓冲

    //接收定时器及线程锁
    QMutex mMutex;

    //定时器
    int mTimer;

    //演示相关
    bool mDisplayModeFlag;              //演示标志位

    //体温探头
    QString mPrbType;

    //血压
    QString mNIBPMode;        //血压测量模式
    bool mNIBPEnd;            //血压测量结束标志位
    int mCufPressure = 0;    //袖带压
    int mSysPressure = 0;    //收缩压
    int mDiaPressure = 0;    //舒张压
    int mMapPressure = 0;    //平均压
    int mPulseRate    = 0;    //脉率

    //呼吸
    QString mRespGain;            //呼吸增益
    QList<ushort> mRespWave;     //线性链表，内容为 Resp 的波形数据
    QPixmap mPixmapResp;          //呼吸画布
    int mRespXStep;              //Resp 横坐标
    int mRespWaveNum;             //Resp 波形数
    bool mDrawResp;               //画图标志位
    ushort mRespWaveData;         //演示模式画图纵坐标

    //血氧
    QString mSPO2Sens;            //血氧灵敏度
    QList<ushort> mSPO2Wave;     //线性链表，内容为 SPO2 的波形数据
    QPixmap mPixmapSPO2;          //血氧画布
    int mSPO2XStep;              //SPO2 横坐标
    int mSPO2WaveNum;             //SPO2 波形数
```

```
        bool mDrawSPO2;                 //画图标志位
        ushort mSPO2WaveData;           //演示模式画图纵坐标

        //心电
        int mECG1Gain;                  //ECG1 增益
        int mECG1Lead;                  //ECG1 导联
        int mECG2Gain;                  //ECG2 增益
        int mECG2Lead;                  //ECG2 导联
        QPixmap mPixmapECG1;            //ECG1 画布
        QPixmap mPixmapECG2;            //ECG2 画布
        QList<ushort> mECG1Wave;        //线性链表，内容为 ECG1 的波形数据
        QList<ushort> mECG2Wave;        //线性链表，内容为 ECG2 的波形数据
        int mECG1XStep;                 //ECG1 横坐标
        int mECG2XStep;                 //ECG2 横坐标
        int mECG1WaveNum;               //ECG1 波形数
        int mECG2WaveNum;               //ECG2 波形数
        bool mDrawECG1;
        bool mDrawECG2;
        ushort mECG1WaveData;           //画图模式纵坐标
        ushort mECG2WaveData;           //画图模式纵坐标

        //数据处理方法
        void procUARTData();
        void dealRcvPackData();
        bool unpackRcvData(uchar recData);

        void analyzeTempData(uchar packAfterUnpack[]); //分析体温
        void analyzeNIBPData(uchar packAfterUnpack[]); //分析血压
        void analyzeRespData(uchar packAfterUnpack[]); //分析呼吸
        void drawRespWave();                           //画呼吸波形
        void analyzeSPO2Data(uchar packAfterUnpack[]); //分析血氧
        void drawSPO2Wave();                           //画血氧波形
        void analyzeECGData(uchar packAfterUnpack[]);  //分析心电
        void drawECG1Wave();                           //画心电 1 波形
        void drawECG2Wave();                           //画心电 2 波形
    };

    #endif // MAINWINDOW_H
```

Mainwindow.cpp

```
    #include "mainwindow.h"
    #include "ui_mainwindow.h"
    #include "global.h"
    #include <QMessageBox>
```

```
#include <QDebug>
#include <QPainter>
#include <QFileDialog>
#include "formtemp.h"
#include "formnibp.h"
#include "formresp.h"
#include "formspo2.h"
#include "formecg.h"

#define byte unsigned char

//模块 ID
const uchar DAT_SYS        = 0x01;              //系统信息
const uchar DAT_ECG        = 0x10;              //心电信息
const uchar DAT_RESP       = 0x11;              //呼吸信息
const uchar DAT_TEMP       = 0x12;              //体温信息
const uchar DAT_SPO2       = 0x13;              //血氧信息
const uchar DAT_NIBP       = 0x14;              //无创血压信息

//二级 ID
const uchar DAT_TEMP_DATA     = 0x02;   //体温实时数据

const uchar DAT_NIBP_CUFPRE = 0x02;   //无创血压实时数据
const uchar DAT_NIBP_END       = 0x03;   //无创血压测量结束
const uchar DAT_NIBP_RSLT1    = 0x04;   //无创血压测量结果 1
const uchar DAT_NIBP_RSLT2    = 0x05;   //无创血压测量结果 2

const uchar DAT_RESP_WAVE     = 0x02;   //呼吸波形数据
const uchar DAT_RESP_RR        = 0x03;   //呼吸率

const uchar DAT_SPO2_WAVE     = 0x02;   //血氧波形数据
const uchar DAT_SPO2_DATA     = 0x03;   //血氧数据

const uchar DAT_ECG_WAVE       = 0x02;   //心电波形数据
const uchar DAT_ECG_LEAD       = 0x03;   //心电导联信息
const uchar DAT_ECG_HR         = 0x04;   //心率

const int PACK_QUEUE_CNT = 4000;        //包数量

const int WAVE_X_SIZE = 1081; //图形区域长度
const int WAVE_Y_SIZE = 131;   //图形区域高度

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
```

```
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    setWindowTitle(tr("生理监护系统"));   //设置标题
    setStyleSheet("background-color:black;");

    //边框
    ui->ecg1WaveLabel->setStyleSheet("border:1px solid white;");
    ui->ecg2WaveLabel->setStyleSheet("border:1px solid white;");
    ui->spo2WaveLabel->setStyleSheet("border:1px solid white;");
    ui->respWaveLabel->setStyleSheet("border:1px solid white;");

    //菜单栏设置
    //ui->menuBar->addAction(tr("串口设置"), this, SLOT(menuSetUART()));
    ui->menuBar->addAction(tr("数据存储"), this, SLOT(menuSaveData()));
    ui->menuBar->addAction(tr("打开数据"), this, SLOT(menuOpen()));
    ui->menuBar->setStyleSheet("color:white");
    //ui->menuBar->addAction(tr("退出"), this, SLOT(menuQuit()));

    //状态栏设置
    mFirstStatusLabel = new QLabel();
    mFirstStatusLabel->setMinimumSize(200, 18); //设置标签最小尺寸
    mFirstStatusLabel->setStyleSheet("color:white");
    QFont font("Microsoft YaHei", 10, 50);
    mFirstStatusLabel->setFont(font);
    mFirstStatusLabel->setText(tr("串口未配置"));
    ui->statusBar->addWidget(mFirstStatusLabel);

    mPackUnpack = new PackUnpack();//形成解包打包函数的对象
    mPackAfterUnpackArr = new uchar*[PACK_QUEUE_CNT];
    for(int i = 0; i < PACK_QUEUE_CNT; i++)
    {
        mPackAfterUnpackArr[i] = new uchar[10]{0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    }

    //呼吸画布初始化
    QPixmap pixResp(WAVE_X_SIZE, WAVE_Y_SIZE);
    mPixmapResp = pixResp;
    mPixmapResp.fill(Qt::black);

    //血氧画布初始化
    QPixmap pixSPO2(WAVE_X_SIZE, WAVE_Y_SIZE);
    mPixmapSPO2 = pixSPO2;
```

```
        mPixmapSPO2.fill(Qt::black);

        //心电画布初始化
        QPixmap pixECG(WAVE_X_SIZE, WAVE_Y_SIZE);
        mPixmapECG1 = pixECG;
        mPixmapECG2 = pixECG;
        mPixmapECG1.fill(Qt::black);
        mPixmapECG2.fill(Qt::black);

        //打开定时器
        mTimer = startTimer(20);

        //初始化参数
        uiInit();
        //事件滤波器，双击 group 组件可以响应，对鼠标的双击或单击产生反应
        ui->ecgInfoGroupBox->installEventFilter(this);
        ui->nibpInfoGroupBox->installEventFilter(this);
        ui->spo2InfoGroupBox->installEventFilter(this);
        ui->respInfoGroupBox->installEventFilter(this);
        ui->tempInfoGroupBox->installEventFilter(this);
    }

    MainWindow::~MainWindow()
    {
        delete ui;
    }
```

数据包结构

Formecg.h

```
    #ifndef FORMECG_H
    #define FORMECG_H

    #include <QWidget>
    #include "packunpack.h"

    namespace Ui {
    class FormECG;
    }

    class FormECG : public QWidget
    {
```

```
    Q_OBJECT

public:
    explicit FormECG(int lead1,int gain1, int lead2, int gain2, QWidget *parent = nullptr);
    ~FormECG();

    PackUnpack mPackUnpack;

signals:
    void sendECGData(int lead1, int gain1, int lead2, int gain2, QByteArray data);

private slots:
    void on_ecg1LeadSetComboBox_currentIndexChanged(const QString &arg1);
    void on_ecg1GainSetComboBox_currentIndexChanged(const QString &arg1);
    void on_ecg2LeadSetComboBox_currentIndexChanged(const QString &arg1);
    void on_ecg2GainSetComboBox_currentIndexChanged(const QString &arg1);

private:
    Ui::FormECG *ui;

    int mECG1Lead;
    int mECG1Gain;
    int mECG2Lead;
    int mECG2Gain;

};

#endif // FORMECG_H
```

Formecg.cpp

```
#include "formecg.h"
#include "ui_formecg.h"
#include <QDebug>

FormECG::FormECG(int lead1,int gain1, int lead2, int gain2, QWidget *parent) :
    QWidget(parent),
    ui(new Ui::FormECG)
{
    ui->setupUi(this);

    //设置窗体显示在最前，Qt::WindowStaysOnTopHint
    setWindowFlags(Qt::WindowCloseButtonHint | Qt::WindowStaysOnTopHint);
    setWindowTitle(tr("心电参数设置"));

    mECG1Lead = lead1;
```

```
    mECG1Gain = gain1;
    mECG2Lead = lead2;
    mECG2Gain = gain2;

    ui->ecg1LeadSetComboBox->setCurrentIndex(mECG1Lead);
    ui->ecg1GainSetComboBox->setCurrentIndex(mECG1Gain);
    ui->ecg2LeadSetComboBox->setCurrentIndex(mECG2Lead);
    ui->ecg2GainSetComboBox->setCurrentIndex(mECG2Gain);
}

FormECG::~FormECG()
{
    delete ui;
}

void FormECG::on_ecg1LeadSetComboBox_currentIndexChanged(const QString &arg1)
{
    mECG1Lead = ui->ecg1LeadSetComboBox->currentIndex();

    QList<uchar> dataList;
    dataList.append(0x14);
    dataList.append(0x81);

    uchar ch = (0 << 4) | (mECG1Lead + 1);
    qDebug() << ch;
    dataList.append(ch);

    for(int i = 0; i < 7; i++)
    {
        dataList.append(0x00);
    }

    QByteArray data;

    mPackUnpack.packData(dataList);

    for(int i = 0; i < dataList.count(); i++)
    {
        data.append(dataList.at(i));
    }

    emit(sendECGData(mECG1Lead, mECG1Gain, mECG2Lead, mECG2Gain, data));
}
```

```
void FormECG::on_ecg1GainSetComboBox_currentIndexChanged(const QString &arg1)
{
    mECG1Gain = ui->ecg1GainSetComboBox->currentIndex();

    QList<uchar> dataList;
    dataList.append(0x10);
    dataList.append(0x83);
    dataList.append(mECG1Gain);

    for(int i = 0; i < 7; i++)
    {
        dataList.append(0x00);
    }

    QByteArray data;

    mPackUnpack.packData(dataList);

    for(int i = 0; i < dataList.count(); i++)
    {
        data.append(dataList.at(i));
    }

    emit(sendECGData(mECG1Lead, mECG1Gain, mECG2Lead, mECG2Gain, data));
}

void FormECG::on_ecg2LeadSetComboBox_currentIndexChanged(const QString &arg1)
{
    mECG2Lead = ui->ecg2LeadSetComboBox->currentIndex();

    QList<uchar> dataList;
    dataList.append(0x10);
    dataList.append(0x81);

    uchar ch = (1 << 4) | (mECG2Lead + 1);
    qDebug() << ch;
    dataList.append(ch);

    for(int i = 0; i < 7; i++)
    {
        dataList.append(0x00);
    }

    QByteArray data;
```

```
        mPackUnpack.packData(dataList);

        for(int i = 0; i < dataList.count(); i++)
        {
            data.append(dataList.at(i));
        }

        emit(sendECGData(mECG1Lead, mECG1Gain, mECG2Lead, mECG2Gain, data));
    }

    void FormECG::on_ecg2GainSetComboBox_currentIndexChanged(const QString &arg1)
    {
        mECG2Gain = ui->ecg2GainSetComboBox->currentIndex();

        QList<uchar> dataList;
        dataList.append(0x10);
        dataList.append(0x83);
        uchar ch = (1 << 4) | (mECG2Gain);
        qDebug() << ch;

        dataList.append(ch);

        for(int i = 0; i < 7; i++)
        {
            dataList.append(0x00);
        }

        QByteArray data;

        mPackUnpack.packData(dataList);

        for(int i = 0; i < dataList.count(); i++)
        {
            data.append(dataList.at(i));
        }

        emit(sendECGData(mECG1Lead, mECG1Gain, mECG2Lead, mECG2Gain, data));
    }
```

Formnibp.h

```
    #ifndef FORMNIBP_H
    #define FORMNIBP_H
```

```
#include <QWidget>
#include "packunpack.h"

namespace Ui {
class FormNIBP;
}

class FormNIBP : public QWidget
{
    Q_OBJECT

public:
    explicit FormNIBP(QString mode, QWidget *parent = nullptr);
    ~FormNIBP();

    PackUnpack mPackUnpack;

signals:
    void sendNIBPSetData(QString mode);
    void sendNIBPData(QByteArray data);

private slots:
    void on_startMeasButton_clicked();
    void on_stopMeasButton_clicked();
    void on_nibpModeComboBox_currentIndexChanged(const QString &arg1);

private:
    Ui::FormNIBP *ui;

    QString mNIBPMode;
};

#endif // FORMNIBP_H
```

Formnibp.cpp

```
#include "formnibp.h"
#include "ui_formnibp.h"

FormNIBP::FormNIBP(QString mode, QWidget *parent) :
    QWidget(parent),
    ui(new Ui::FormNIBP)
{
    ui->setupUi(this);
```

```
    //设置窗体显示在最前，Qt::WindowStaysOnTopHint
    setWindowFlags(Qt::WindowCloseButtonHint | Qt::WindowStaysOnTopHint);
    setWindowTitle(tr("血压参数设置"));

    mNIBPMode = mode;
    ui->nibpModeComboBox->setCurrentText(mNIBPMode); //显示上次选择的模式
}

FormNIBP::~FormNIBP()
{
    delete ui;
}

void FormNIBP::on_startMeasButton_clicked()
{
    QList<uchar> dataList;
    dataList.append(0x14);
    dataList.append(0x80);

    for(int i = 0; i < 8; i++)
    {
        dataList.append(0x00);
    }

    QByteArray data;

    mPackUnpack.packData(dataList);

    for(int i = 0; i < dataList.count(); i++)
    {
        data.append(dataList.at(i));
    }

    emit(sendNIBPData(data));

    this->close();
}

void FormNIBP::on_stopMeasButton_clicked()
{
    QList<uchar> dataList;
    dataList.append(0x14);
    dataList.append(0x81);
```

```
        for (int i = 0; i < 8; i++)
        {
            dataList.append(0x00);
        }

        QByteArray data;

        mPackUnpack.packData(dataList);

        for (int i = 0; i < dataList.count(); i++)
        {
            data.append(dataList.at(i));
        }

        emit(sendNIBPData(data));

        this->close();
    }

    void FormNIBP::on_nibpModeComboBox_currentIndexChanged(const QString &arg1)
    {
        mNIBPMode = ui->nibpModeComboBox->currentText();
        emit(sendNIBPSetData(mNIBPMode));
    }
```

Formresp.h

```
    #ifndef FORMRESP_H
    #define FORMRESP_H

    #include <QWidget>
    #include "packunpack.h"

    namespace Ui {
    class FormResp;
    }

    class FormResp : public QWidget
    {
        Q_OBJECT

    public:
        explicit FormResp(QString gain, QWidget *parent = nullptr);
        ~FormResp();
```

```
        PackUnpack mPackUnpack;

    protected:
        void closeEvent(QCloseEvent *event);

    signals:
        void sendRespData(QString gain, QByteArray data);

    private slots:
        void on_okButton_clicked();
        void on_cancelButton_clicked();
        void on_gainComboBox_currentIndexChanged(const QString &arg1);

    private:
        Ui::FormResp *ui;

        QString mRespGain;
    };

    #endif // FORMRESP_H
```

Formresp.cpp

```
    #include "formresp.h"
    #include "ui_formresp.h"
    #include <QDebug>

    FormResp::FormResp(QString gain, QWidget *parent) :
        QWidget(parent),
        ui(new Ui::FormResp)
    {
        ui->setupUi(this);

        //设置窗体显示在最前，Qt::WindowStaysOnTopHint
        setWindowFlags(Qt::WindowCloseButtonHint | Qt::WindowStaysOnTopHint);
        setWindowTitle(tr("呼吸参数设置"));

        mRespGain = gain;
        ui->gainComboBox->setCurrentText(mRespGain);
    }

    FormResp::~FormResp()
    {
        delete ui;
    }
```

```
void FormResp::closeEvent(QCloseEvent *event)

{
    //窗口关闭之前需要的操作
    qDebug() << "close call";
}

void FormResp::on_okButton_clicked()
{
    this->close();
}

void FormResp::on_cancelButton_clicked()
{
    this->close();
}

void FormResp::on_gainComboBox_currentIndexChanged(const QString &arg1)
{
    mRespGain = ui->gainComboBox->currentText();

    int respGain = ui->gainComboBox->currentIndex();

    if(respGain < 0)
    {
        return;
    }

    QList<uchar> dataList;
    uchar sensData = respGain ;     //增益设置：0-*0.25；1-*0.5；2-*1；3-*2；4-*4
    dataList.append(0x80);                  //二级 ID-0x80
    dataList.append(sensData);

    QByteArray data;
    dataList.insert(0, 0x11);               //一级 ID-0x11
    dataList.append(0);
    dataList.append(0);
    dataList.append(0);
    dataList.append(0);
    dataList.append(0);
    dataList.append(0);
    dataList.append(0);
    mPackUnpack.packData(dataList);
```

```
        int len = dataList.count();

        for(int i = 0; i < len; i++)
        {
            data.append(dataList.at(i));
        }

        emit(sendRespData(mRespGain, data));
    }
```

Formsetuart.h

```
    #ifndef FORMSETUART_H
    #define FORMSETUART_H

    #include <QWidget>
    #include "global.h"
    #include "QSerialPort"
    #include "QSerialPortInfo"

    namespace Ui {
    class FormSetUART;
    }

    class FormSetUART : public QWidget
    {
        Q_OBJECT

    public:
        explicit FormSetUART(QWidget *parent = nullptr);
        ~FormSetUART();

    signals:
        void uartSetData(QString portNum, int baudRate, int dataBits,
                         int stopBits, int parity, bool isOpened);

    private slots:
        void on_openUARTButton_clicked();

    private:
        Ui::FormSetUART *ui;
    };

    #endif // FORMSETUART_H
```

Formsetuart.cpp

```
#include "formsetuart.h"
#include "ui_formsetuart.h"

FormSetUART::FormSetUART(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::FormSetUART)
{
    ui->setupUi(this);

    setWindowFlags(Qt::WindowCloseButtonHint);
    setWindowTitle(tr("串口设置"));

    if(gUARTOpenFlag)
    {
        ui->openUARTButton->setText(tr("关闭串口"));
        ui->uartStsLabel->setPixmap(QPixmap(":/new/prefix1/image/open.png"));
    }
    else
    {
        ui->openUARTButton->setText(tr("打开串口"));
        ui->uartStsLabel->setPixmap(QPixmap(":/new/prefix1/image/close.png"));
    }

    ui->uartNumComboBox->clear();
    //通过 QSerialPortInfo 查找可用串口
    foreach(const QSerialPortInfo &info, QSerialPortInfo::availablePorts())
    {
        ui->uartNumComboBox->addItem(info.portName());
    }
}

FormSetUART::~FormSetUART()
{
    delete ui;
}

void FormSetUART::on_openUARTButton_clicked()
{
    emit(uartSetData(ui->uartNumComboBox->currentText(),
ui->baudRateComboBox->currentText().toInt(),
                     ui->dataBitsComboBox->currentIndex(),
ui->stopBitsComboBox->currentIndex(),
```

```
                                    ui->parityComboBox->currentIndex(), gUARTOpenFlag));
        close();
    }

formspo2.h
    #ifndef FORMSPO2_H
    #define FORMSPO2_H

    #include <QWidget>
    #include "packunpack.h"

    namespace Ui {
    class FormSPO2;
    }

    class FormSPO2 : public QWidget
    {
        Q_OBJECT

    public:
        explicit FormSPO2(QString sens, QWidget *parent = nullptr);
        ~FormSPO2();

        PackUnpack mPackUnpack;

    signals:
        void sendSPO2Data(QString sens, QByteArray data);

    private slots:
        void on_okButton_clicked();
        void on_cancelButton_clicked();
        void on_sensComboBox_currentIndexChanged(const QString &arg1);

    private:
        Ui::FormSPO2 *ui;

        QString mSPO2Sens;
    };

    #endif // FORMSPO2_H

formspo2.cpp
    #include "formspo2.h"
    #include "ui_formspo2.h"
```

```
FormSPO2::FormSPO2(QString sens, QWidget *parent) :
    QWidget(parent),
    ui(new Ui::FormSPO2)
{
    ui->setupUi(this);

    //设置窗体显示在最前，Qt::WindowStaysOnTopHint
    setWindowFlags(Qt::WindowCloseButtonHint | Qt::WindowStaysOnTopHint);
    setWindowTitle(tr("血氧参数设置"));

    mSPO2Sens = sens;
    ui->sensComboBox->setCurrentText(mSPO2Sens);
}

FormSPO2::~FormSPO2()
{
    delete ui;
}

void FormSPO2::on_okButton_clicked()
{
    this->close();
}

void FormSPO2::on_cancelButton_clicked()
{
    this->close();
}

void FormSPO2::on_sensComboBox_currentIndexChanged(const QString &arg1)
{
    mSPO2Sens = ui->sensComboBox->currentText();

    int spo2Sens = ui->sensComboBox->currentIndex();

    if(spo2Sens < 0)
    {
        return;
    }

    uchar sensData = spo2Sens + 1;    //计算灵敏度：1-高；2-中；3-低
    QList<uchar> dataList;
    dataList.append(0x80);                  //二级 ID-0x80
```

```
        dataList.append(sensData);
        dataList.insert(0, 0x13);              //模块 ID-0x13
        mPackUnpack.packData(dataList);

        int len = dataList.count();
        QByteArray data;
        for(int i = 0; i < len; i++)
        {
            data.append(dataList.at(i));
        }

        emit(sendSPO2Data(mSPO2Sens, data));
    }
```

Formtemp.h

```
    #ifndef FORMTEMP_H
    #define FORMTEMP_H

    #include <QWidget>
    #include "packunpack.h"

    namespace Ui {
    class FormTemp;
    }

    class FormTemp : public QWidget
    {
        Q_OBJECT

    public:
        explicit FormTemp(QString prbType, QWidget *parent = nullptr);
        ~FormTemp();

        PackUnpack mPackUnpack;

    signals:
        void sendTempData(QString prbType, QByteArray data);

    private slots:
        void on_okButton_clicked();
        void on_cancelButton_clicked();
        void on_prbTypeComboBox_currentIndexChanged(const QString &arg1);

    private:
```

```
        Ui::FormTemp *ui;

        QString mPrbType;

        //数据处理方法
        void procUARTData();
        void dealRcvPackData();
    };

    #endif // FORMTEMP_H
```

Formtemp.cpp

```
    #include "formtemp.h"
    #include "ui_formtemp.h"

    FormTemp::FormTemp(QString prbType, QWidget *parent) :
        QWidget(parent),
        ui(new Ui::FormTemp)
    {
        ui->setupUi(this);

        //设置窗体显示在最前，Qt::WindowStaysOnTopHint
        setWindowFlags(Qt::WindowCloseButtonHint | Qt::WindowStaysOnTopHint);
        setWindowTitle(tr("体温参数设置"));

        mPrbType = prbType;
        ui->prbTypeComboBox->setCurrentText(mPrbType);
    }

    FormTemp::~FormTemp()
    {
        delete ui;
    }

    void FormTemp::on_okButton_clicked()
    {
        this->close();
    }

    void FormTemp::on_cancelButton_clicked()
    {
        this->close();
    }
```

```
void FormTemp::on_prbTypeComboBox_currentIndexChanged(const QString &arg1)
{
    mPrbType = ui->prbTypeComboBox->currentText();

    int tempPrbType = ui->prbTypeComboBox->currentIndex();

    if(tempPrbType < 0)
    {
        return;
    }

    QList<uchar> dataList;
    QByteArray data;

    dataList.append(0x12);
    dataList.append(0x80);

    switch(tempPrbType)
    {
    case 0:
        for(int i = 0; i < 8; i++)
        {
            dataList.append(0x00);
        }
        break;
    case 1:
        dataList.append(0x01);
        for(int i = 0; i < 7; i++)
        {
            dataList.append(0x00);
        }
        break;
    default:
        break;
    }

    mPackUnpack.packData(dataList);

    for(int i = 0; i < dataList.count(); i++)
    {
        data.append(dataList.at(i));
    }

    emit(sendTempData(mPrbType, data));
```

```
}
```

解包算法

Packunpack.h

```
#ifndef _PACKUNPACK_H_
#define _PACKUNPACK_H_


#include <QList>

class PackUnpack
{

public:
    const int MAX_PACK_LEN = 10;

    //定义 m_listBuf
    //m_listBuf[0]   : 对应 packId
    //m_listBuf[1-8]: 对应 arrData
    //m_listBuf[9]   : 对应 checkSum
    QList<uchar> mListBuf;   //缓冲，逐个读取字符
    QList<int> mListPackLen;    //文件包的长度，因为当第八位是 0 的时候，就开始计
数

    PackUnpack();
    int packData(QList<uchar> &listPack);
    bool unpackData(uchar data);
    QList<uchar> getUnpackRslt();

private:
    int   mPackLen;          //数据包长度
    bool mGotPackId;       //获取到 ID 的标志，获得正确的 ID 即为 TRUE，否则为
FALSE
    int   mRestByteNum;    //剩余字节数

    void packWithCheckSum(QList<uchar> &pack);
    bool unpackWithCheckSum(QList<uchar> &listPack);   //解包算上校验和

};

#endif //_PACKUNPACK_H_
```

Packunpack.cpp

```
#include "packunpack.h"
//打包,先得到数据头,把每一位的最高位拿出来,放的时候一次一位放进去(dataHead),
每拿一个最高位后把它低七位相加 (checkSum
enum EnumPackID
{
    DAT_SYS      = 0x01,          //系统信息
    DAT_ECG      = 0x10,          //心电信息
    DAT_RESP     = 0x11,          //呼吸信息
    DAT_TEMP     = 0x12,          //体温信息
    DAT_SPO2     = 0x13,          //血氧信息
    DAT_NIBP     = 0x14,          //无创血压信息

    MAX_PACK_ID = 0x80
};

PackUnpack::PackUnpack()
{
    for(int i = 0; i < MAX_PACK_LEN; i++)
    {
      mListBuf.insert(i, 0);   //ID、数据头、数据及校验和均清零
    }

    mPackLen       = 0;        //数据包的长度默认为 0
    mGotPackId     = false;    //获取到数据包 ID 标志默认为 false，即尚未获取到 ID
    mRestByteNum   = 0;        //剩余的字节数默认为 0
}

void PackUnpack::packWithCheckSum(QList<uchar> &pack)
{
    int i;
    uchar dataHead;            //数据头，在数据包的第 2 个位置，即 ID 之后
    uchar checkSum;            //数据校验和，在数据包的最后一个位置

    //对于包长小于 2 的数据，不需要打包，因此直接跳出此方法，注意，最短的包只
有包头和校验和，即系统复位
    if (10 != pack.length())
    {
        return;
    }

    checkSum = pack[0];        //取出 ID，加到校验和
```

```
    dataHead = 0;               //数据头清零

    for(i = 8; i > 1; i--)
    {
        //将最高位取出，数据头左移，后面数据的最高位位于 dataHead 的靠左
        dataHead <<= 1;

        //对数据进行最高位置 1 操作，并将数据右移一位（数据头插入原因）
        pack.replace(i, (uchar)((pack[i - 1]) | 0x80));

        //数据加到校验和
        checkSum += pack.at(i);

        //取出原始数据的最高位，与 dataHead 相或
        dataHead |= (uchar)(((pack[i - 1]) & 0x80) >> 7);
    }

    //数据头在数据包的第二个位置，仅次于包头，数据头的最高位也要置为 1
    pack.replace(1, (uchar)(dataHead | (0x80)));

    //将数据头加到校验和
    checkSum += pack[1];

    //校验和的最高位也要置为 1
    pack.replace(9, (uchar)(checkSum | 0x80));
}

bool PackUnpack::unpackWithCheckSum(QList<uchar> &listPack)
{
    int i;
    uchar dataHead;
    uchar checkSum;


    if(10 != listPack.count())              //len 不等于 10，返回 0
    {
        return false;
    }

    checkSum = listPack[0];     //取出 ID，加到校验和
    dataHead = listPack[1];     //取出数据包的数据头，赋给 dataHead
    checkSum += dataHead;       //将数据头加到校验和

    for(i = 1; i < 8; i++)
```

```
    {
        checkSum += listPack[i + 1];      //将数据依次加到校验和

        //还原有效的 8 位数据
        listPack[i] = (uchar)((listPack[i + 1] & 0x7f) | ((dataHead & 0x1) << 7)); //将高八位取出，补回到原来的数据里去
        dataHead >>= 1;                   //数据头右移一位
    }

    //判断 ID、数据头和数据求和的结果（低七位）是否与校验和的低七位相等，如果不等返回 0
    if((checkSum & 0x7f) != ((listPack[9]) & 0x7f))
    {
        return false;
    }

        return true;
}

int PackUnpack::packData(QList<uchar> &listPack)
{
    int len = 0;                          //数据包长度

    if (listPack[0] < 0x80)               //包 ID 必须在 0x00-0x7F 之间, listPack[0]为包 ID，当读取到最高位为 0 时，就表示数据开始
    {
        packWithCheckSum(listPack);
    }

    return len;
}

bool PackUnpack::unpackData(uchar data)
{
    bool findPack = false;

    if(mGotPackId)                        //已经接收到包 ID
    {
        if(0x80 <= data)                  //包数据（非包 ID，已找到的数据）必须大于或等于 0X80
        {
            //数据包中的数据从第二个字节开始存储，因为第一个字节是包 ID
            mListBuf[mPackLen] = data;
            mPackLen++;                   //包长自增
```

```
                mRestByteNum--;                         //剩余字节数自减

                //已经接收到完整的数据包
                if((0 >= mRestByteNum) && (10 == mPackLen)) //如果里面的数已经等于
10 了，就开始解包
                {
                    findPack = unpackWithCheckSum(mListBuf);    //接收到完整数据包
后尝试解包
                    mGotPackId = false;                 //清除获取到包 ID 标志，即重
新判断下一个数据包
                }
            }
            else
            {
                mGotPackId = false;                     //表示出错
            }
        }
        else if(data < 0x80)                        //当前的数据为包 ID，即数据头
        {
                mRestByteNum = 9;    //剩余的包长，即打包好的包长减去 1
                mPackLen = 1;                           //尚未接收到包 ID 即表示包长
为 1
                mListBuf[0] = data;                     //数据包的 ID
                mGotPackId = true;                      //表示已经接收到包 ID
        }

        return findPack;      //如果获取到完整的数据包，并解包成功，findPack 为 TRUE，
否则为 FALSE
    }


    QList<uchar> PackUnpack::getUnpackRslt()
    {
        return(mListBuf);
    }
```

显示算法

（接上一个 mainwindow.h）

```
    bool MainWindow::eventFilter(QObject *obj, QEvent *event)
    {
        if(obj == ui->ecgInfoGroupBox)
```

```
    {
        if(event->type() == QEvent::MouseButtonPress)
        {
            qDebug() << "ecgInfoGroupBox click";

            if((mSerialPort != NULL) && (mSerialPort->isOpen()))
            {
                FormECG *formECG = new FormECG(mECG1Lead, mECG1Gain,
mECG2Lead, mECG2Gain);
                connect(formECG, SIGNAL(sendECGData(int, int, int, int, QByteArray)),
this, SLOT(recECGData(int, int, int, int, QByteArray)));
                formECG->show();
            }
            else
            {

            }
            return true;
        }
        else
        {
            return false;
        }
    }
    else if(obj == ui->nibpInfoGroupBox)
    {
        if(event->type() == QEvent::MouseButtonPress)
        {
            qDebug() << "nibpInfoGroupBox click";

            if((mSerialPort != NULL) && (mSerialPort->isOpen()))
            {
                FormNIBP *formNIBP = new FormNIBP(mNIBPMode);
                //更新显示测量模式
                connect(formNIBP,      SIGNAL(sendNIBPSetData(QString)),      this,
SLOT(recNIBPSetData(QString)));
                formNIBP->show();
                //下发命令
                connect(formNIBP,      SIGNAL(sendNIBPData(QByteArray)),      this,
SLOT(writeSerial(QByteArray)));
            }
            else
            {
```

```
                }
                return true;
            }
            else
            {
                return false;
            }
        }
        else if(obj == ui->spo2InfoGroupBox)
        {
            if(event->type() == QEvent::MouseButtonPress)
            {
                qDebug() << "spo2InfoGroupBox click";

                if((mSerialPort != NULL) && (mSerialPort->isOpen()))
                {
                    FormSPO2 *formSPO2 = new FormSPO2(mSPO2Sens);
                    formSPO2->show();
                    connect(formSPO2,  SIGNAL(sendSPO2Data(QString,  QByteArray)),
this, SLOT(recSPO2Data(QString, QByteArray)));
                }
                else
                {

                }
                return true;
            }
            else
            {
                return false;
            }
        }
        else if(obj == ui->respInfoGroupBox)
        {
            if(event->type() == QEvent::MouseButtonPress)
            {
                qDebug() << "respInfoGroupBox click";

                if((mSerialPort != NULL) && (mSerialPort->isOpen()))
                {
                    FormResp *formResp = new FormResp(mRespGain);
                    formResp->show();
                    connect(formResp, SIGNAL(sendRespData(QString, QByteArray)), this,
SLOT(recRespGain(QString, QByteArray)));
```

```
                }
                else
                {

                }
                return true;
            }
            else
            {
                return false;
            }
        }
        else if(obj == ui->tempInfoGroupBox)
        {
            if(event->type() == QEvent::MouseButtonPress)
            {
                qDebug() << "tempInfoGroupBox click";

                if((mSerialPort != NULL) && (mSerialPort->isOpen()))
                {
                    FormTemp *formTemp = new FormTemp(mPrbType);
                    formTemp->show();
                    connect(formTemp,  SIGNAL(sendTempData(QString,  QByteArray)),
this, SLOT(recPrbType(QString, QByteArray)));
                }
                else
                {

                }
                return true;
            }
            else
            {
                return false;
            }
        }
        else
        {
            // pass the event on to the parent class
            return QMainWindow::eventFilter(obj, event);
        }
}

void MainWindow::uiInit()
```

```
{
    mSerialPort = NULL;

    mPackHead = -1;      //当前需要处理的缓冲包的序号
    mPackTail = -1;      //最后处理的缓冲包的序号，mPackHead 不能追上 mPackTail,
追上的话表示收到的数据超出 2000 的缓冲

    //初始化体温探头
    mPrbType   = "YSI";

    //初始化血压参数
    mNIBPMode = "手动";
    mNIBPEnd   = 0;
    mCufPressure = 0;    //袖带压
    mSysPressure = 0;    //收缩压
    mDiaPressure = 0;    //舒张压
    mMapPressure = 0;    //平均压
    mPulseRate     = 0;    //脉率

    //初始化呼吸参数
    mRespWave.clear();
    mRespXStep     = 0;        //Resp 横坐标
    mRespWaveNum = 0;        //Resp 波形数
    mDrawResp = false;
    mRespGain = "X0.25";   //呼吸增益设置
    mRespWaveData = 50;

    //初始化血氧参数
    mSPO2Wave.clear();
    mSPO2XStep     = 0;        //SPO2 横坐标
    mSPO2WaveNum = 0;        //SPO2 波形数
    mDrawSPO2 = false;
    mSPO2Sens = "中";        //血氧灵敏度设置
    mSPO2WaveData = 77;

    //初始化心电参数
    mECG1Gain = 2;
    mECG1Lead = 1;
    mECG2Gain = 2;
    mECG2Lead = 0;
    mECG1Wave.clear();
    mECG2Wave.clear();
    mECG1XStep = 0;      //ECG1 横坐标
    mECG2XStep = 0;      //ECG2 横坐标
```

```
    mECG1WaveNum = 0;   //ECG1 波形数
    mECG2WaveNum = 0;   //ECG2 波形数
    mDrawECG1 = false;
    mDrawECG2 = false;
    mECG1WaveData = 50;
    mECG2WaveData = 50;

    //初始化演示参数
    mDisplayModeFlag = true;
}

void MainWindow::menuSetUART()
{
    qDebug() << "click setUART";
    FormSetUART *formSetUART = new FormSetUART();
    connect(formSetUART, SIGNAL(uartSetData(QString, int, int, int, int, bool)), this,
SLOT(initSerial(QString, int, int, int, int, bool)));
    formSetUART->show();
}

void MainWindow::initSerial(QString portNum, int baudRate, int dataBits, int stopBits, int
parity, bool isOpened)
{
    qDebug() << "init serial ";
    qDebug() << portNum;

    if(!isOpened)
    {
        mSerialPort = new QSerialPort;
        mSerialPort->setPortName(portNum);

        if(mSerialPort->open(QIODevice::ReadWrite))
        {
            //设置波特率
            qDebug() << "baudRate:" << baudRate;
            mSerialPort->setBaudRate(baudRate);

            //设置数据位
            mSerialPort->setDataBits(QSerialPort::Data8);

            //设置停止位
            mSerialPort->setStopBits(QSerialPort::OneStop);

            //设置校验位
```

```
                mSerialPort->setParity(QSerialPort::NoParity);

                //设置流控制
                mSerialPort->setFlowControl(QSerialPort::NoFlowControl);
                mFirstStatusLabel->setStyleSheet("color:black;");
                mFirstStatusLabel->setText(portNum + "已打开");

                //连接信号槽
                connect(mSerialPort, &QSerialPort::readyRead, this,
&MainWindow::readSerial);
            }
            else
            {
                QMessageBox::about(NULL, "提示", "串口无法打开\r\n 不存在或已被占用
");
                mFirstStatusLabel->setStyleSheet("color:red;");
                mFirstStatusLabel->setText(portNum + "无法打开");
                return;
            }

            gUARTOpenFlag = true;
        }
        else
        {
            mSerialPort->clear();
            mSerialPort->close();
            mSerialPort->deleteLater();

            gUARTOpenFlag = false;
            mFirstStatusLabel->setStyleSheet("color:black;");
            mFirstStatusLabel->setText(portNum + "已关闭");
        }
    }

    void MainWindow::readSerial()
    {
        mRxData = mSerialPort->readAll();

        if(!mRxData.isEmpty())
        {
            procUARTData();      //PCT 解包
            dealRcvPackData(); //数据分类保存
        }
    }
```

```
void MainWindow::writeSerial(QByteArray data)
{
    qDebug() << "write serial";

    if(mSerialPort != NULL)
    {
        if(mSerialPort->isOpen())
        {
            mTxData = data;
            mSerialPort->write(mTxData);
        }
    }
}

void MainWindow::procUARTData()
{
    char *buf;

    if(mRxData.size() > 0)
    {
        buf = mRxData.data();
    }

    for(int i = 0; i < mRxData.size(); i++)
    {
        unpackRcvData(*(buf + i));
    }

    mRxData.clear();
}

bool MainWindow::unpackRcvData(uchar recData)
{
    bool findPack = false;
    findPack = mPackUnpack->unpackData(recData);

    if(findPack)
    {
        mPackAfterUnpackList = mPackUnpack->getUnpackRslt();   //获取解包结果

        if(mPackAfterUnpackList.size() > 10)
        {
            QMessageBox::information(NULL, tr("Info"), tr("长度异常"), "确定");
```

```
            }

            int head = (mPackHead + 1) % PACK_QUEUE_CNT;   //作显示用，一个 head 一个尾，叠加一次就要往后移位

            for(int i = 0; i < 8; i++)
            {
                mPackAfterUnpackArr[head][i] = mPackAfterUnpackList[i];
            }

            mMutex.lock();
            mPackHead = (mPackHead + 1) % PACK_QUEUE_CNT;    //当数据超过了限定长度，重新拉回来，所以求模
            //qDebug()<<mPackHead;

            if(mPackTail == -1)
            {
                mPackTail = 0;
            }

            mMutex.unlock();
        }

        return findPack;
}

void MainWindow::dealRcvPackData()
{
        int headIndex = -1;
        int tailIndex = -1;

        mMutex.lock();
        headIndex = mPackHead;   //串口进来的数据的序号
        tailIndex = mPackTail;   //处理串口数据的序号
        mMutex.unlock();

        if(headIndex < tailIndex)
        {
            headIndex = headIndex + PACK_QUEUE_CNT;
        }

        int index;
        int cnt = headIndex - tailIndex;
```

```
for(int i = tailIndex; i < headIndex; i++)
{
    index = i % PACK_QUEUE_CNT;
    //根据模块 ID 处理数据包
    switch(mPackAfterUnpackArr[index][0])
    {
    case DAT_TEMP:
        analyzeTempData(mPackAfterUnpackArr[index]);
        break;
    case DAT_NIBP:
        analyzeNIBPData(mPackAfterUnpackArr[index]);
        break;
    case DAT_RESP:
        analyzeRespData(mPackAfterUnpackArr[index]);
        break;
    case DAT_SPO2:
        analyzeSPO2Data(mPackAfterUnpackArr[index]);
        break;
    case DAT_ECG:
        analyzeECGData(mPackAfterUnpackArr[index]);
        break;
    default:
        break;
    }
}

mMutex.lock();
mPackTail = (mPackTail + cnt) % PACK_QUEUE_CNT;
mMutex.unlock();

//刷新波形
if(mRespWave.count() > 2)
{
    mDrawResp = true;
}

//刷新波形
if(mSPO2Wave.count() > 2)
{
    mDrawSPO2 = true;
}

//刷新波形
if(mECG1Wave.count() > 10)
```

```
    {
        mDrawECG1 = true;
        mDrawECG2 = true;
    }
}

void MainWindow::analyzeTempData(uchar packAfterUnpack[])
{
    float temp1Data;            //通道 1 体温数据
    float temp2Data;            //通道 2 体温数据
    bool temp1Lead;              //通道 1 导联状态
    bool temp2Lead;              //通道 2 导联状态

    int data;
    switch(packAfterUnpack[1])
    {
    case DAT_TEMP_DATA:
        data = packAfterUnpack[2];
        temp1Lead = (data & 0x01) != 1;
        temp2Lead = ((data >> 1) & 0x01) != 1;
        data = packAfterUnpack[3];
        data = (data << 8) | packAfterUnpack[4];
        temp1Data = (float)(data / 10.0);
        data = packAfterUnpack[5];
        data = (data << 8) | packAfterUnpack[6];
        temp2Data = (float)(data / 10.0);

        if(temp1Lead)
        {
            ui->temp1LeadLabel->setText("T1 导联");
            ui->temp1LeadLabel->setStyleSheet("color:green;");
            ui->temp1ValLabel->setText(QString::number(temp1Data));
        }
        else
        {
            ui->temp1LeadLabel->setText("T1 脱落");
            ui->temp1LeadLabel->setStyleSheet("color:red;");
            ui->temp1ValLabel->setText("--");
        }

        if(temp2Lead)
        {
            ui->temp2LeadLabel->setText("T2 导联");
            ui->temp2LeadLabel->setStyleSheet("color:green;");
```

```
                ui->temp2ValLabel->setText(QString::number(temp2Data));
            }
            else
            {
                ui->temp2LeadLabel->setText("T2 脱落");
                ui->temp2LeadLabel->setStyleSheet("color:red;");
                ui->temp2ValLabel->setText("--");
            }

            break;

        default:
            break;
        }
    }

    void MainWindow::analyzeNIBPData(uchar packAfterUnpack[])
    {
        int data;

        switch (packAfterUnpack[1])
        {
        case DAT_NIBP_CUFPRE:
            data = packAfterUnpack[2];
            mCufPressure = (data << 8) | packAfterUnpack[3];
            break;

        case DAT_NIBP_END:
            data = packAfterUnpack[3];
            if(data != 0)
            {
                mNIBPEnd = true;
            }
            break;

        case DAT_NIBP_RSLT1:
            data = packAfterUnpack[2];
            mSysPressure = (data << 8) | packAfterUnpack[3];
            data = packAfterUnpack[4];
            mDiaPressure = (data << 8) | packAfterUnpack[5];
            data = packAfterUnpack[6];
            mMapPressure = (data << 8) | packAfterUnpack[7];
            break;
```

```
        case DAT_NIBP_RSLT2:
            data = packAfterUnpack[2];
            mPulseRate = (data << 8) | packAfterUnpack[3];
            mNIBPEnd = true;
            break;

        default:
            break;
        }

        ui->cufPressureLabel->setText(QString::number(mCufPressure));
        ui->sysPressureLabel->setText(QString::number(mSysPressure));
        ui->diaPressureLabel->setText(QString::number(mDiaPressure));
        ui->mapPressureLabel->setText(QString::number(mMapPressure));
        ui->nibpPRLabel->setText(QString::number(mPulseRate));
    }

    void MainWindow::analyzeRespData(uchar packAfterUnpack[])
    {
        uchar respWave = 0;              //呼吸波形数据
        ushort respRate = 0;           //呼吸率值
        byte respRateHByte = 0;        //呼吸率高字节
        byte respRateLByte = 0;          //呼吸率低字节

        switch(packAfterUnpack[1])
        {
        //波形数据
        case DAT_RESP_WAVE:
            for(int i = 2; i < 7; i++)
            {
                respWave = packAfterUnpack[i];
                mRespWave.append(respWave);
            }
            break;

        //呼吸数据
        case DAT_RESP_RR:
            respRateHByte = packAfterUnpack[2];
            respRateLByte = packAfterUnpack[3];
            respRate = (ushort)(respRate | respRateHByte);
            respRate = (ushort)(respRate << 8);
            respRate = (ushort)(respRate | respRateLByte);

            if(respRate >= 255)
```

```
            {
                ui->respRateLabel->setText("---"); //呼吸率值最大不超过 255，超过
255 则视为无效值，不显示呼吸率
            }
            else
            {
                ui->respRateLabel->setText(QString::number(respRate, 10));
            }

            break;

        default:
            break;
        }
}

void MainWindow::drawRespWave()
{
    if(mDrawResp)
    {
        //mDrawResp = false;

        int respCnt;
        ushort unRespData;
        ushort unRespWaveData;

        if(!mDisplayModeFlag)
        {
            respCnt = mRespWave.count() - 1;
        }
        else
        {
            respCnt = 2;
        }

        if(respCnt < 2)
        {
            return;
        }

        QPainter painterResp(&mPixmapResp);
        painterResp.setBrush(Qt::black);
        painterResp.setPen(QPen(Qt::black, 1, Qt::SolidLine));
```

```
            if(respCnt >= 1080 - mRespXStep)
            {
                QRect rct(mRespXStep, 0, 1080 - mRespXStep, 130);
                painterResp.drawRect(rct);

                QRect rect(0, 0, respCnt + mRespXStep - 1080, 130);
                painterResp.drawRect(rect);
            }
            else
            {
                QRect rct(mRespXStep, 0, respCnt, 130);
                painterResp.drawRect(rct);
            }

            painterResp.setPen(QPen(Qt::red, 2, Qt::SolidLine));

            //监护模式绘图
            if(!mDisplayModeFlag)
            {
                for(int i = 0; i < respCnt; i++)
                {
                    QPoint point1(mRespXStep, 110 - mRespWave.at(i) / 2.55);
                    QPoint point2(mRespXStep + 1, 110 - mRespWave.at(i + 1)/ 2.55);
                    painterResp.drawLine(point1, point2);
                    mRespXStep++;
                    if(mRespXStep >= 1080)
                    {
                        mRespXStep = 1;
                    }
                }

                double tail = mRespWave.last();
                mRespWave.clear();
                mRespWave.append(tail);
            }
            //演示模式绘图
            else if(mDisplayModeFlag)
            {
                for(int i = 0; i < respCnt; i++)
                {
                    unRespData = mRespWave.takeFirst();
                    unRespWaveData = 110 - unRespData / 2.55;
                    painterResp.drawLine(mRespXStep, mRespWaveData, mRespXStep +
1, unRespWaveData);
```

```
                    mRespWaveData = unRespWaveData;
                    mRespXStep++;

                    if(mRespXStep >= 1080)
                    {
                        mRespXStep = 1;
                    }
                    if(mRespWave.length()>1)
                    {
                        mRespWave.removeFirst();
                    }

                }
            }

            ui->respWaveLabel->setPixmap(mPixmapResp);
        }
    }

    void MainWindow::analyzeSPO2Data(uchar packAfterUnpack[])
    {
        ushort spo2Wave = 0;            //血氧波形数据
        ushort pulseRate = 0;           //脉率值
        byte pulseRateHByte = 0;        //脉率高字节
        byte pulseRateLByte = 0;        //脉率低字节
        byte spo2Value = 0;                 //血氧饱和度
        byte fingerLead = 0;            //手指连接信息
        byte probeLead = 0;                 //探头连接信息

        switch(packAfterUnpack[1])
        {
        //波形数据
        case DAT_SPO2_WAVE:
            for(int i = 2; i < 7; i++)
            {
                spo2Wave = (ushort)packAfterUnpack[i];
                mSPO2Wave.append(spo2Wave);
            }

            fingerLead = (byte)((packAfterUnpack[7] & 0x80) >> 7);      //手指脱落信息
            probeLead = (byte)((packAfterUnpack[7] & 0x10) >> 4);       //探头脱落信息

            if(fingerLead == 0x01)
            {
```

```
            ui->spo2FingerLeadLabel->setStyleSheet("color:red;");
            ui->spo2FingerLeadLabel->setText("手指脱落");
        }
        else
        {
            ui->spo2FingerLeadLabel->setStyleSheet("color:green;");
            ui->spo2FingerLeadLabel->setText("手指连接");
        }

        if(probeLead == 0x01)
        {
            ui->spo2PrbLeadLabel->setStyleSheet("color:red;");
            ui->spo2PrbLeadLabel->setText("探头脱落");
        }
        else
        {
            ui->spo2PrbLeadLabel->setStyleSheet("color:green;");
            ui->spo2PrbLeadLabel->setText("探头连接");
        }
        break;

    //血氧数据
    case DAT_SPO2_DATA:
        //脉率
        pulseRateHByte = packAfterUnpack[3];
        pulseRateLByte = packAfterUnpack[4];
        pulseRate = (ushort)(pulseRate | pulseRateHByte);
        pulseRate = (ushort)(pulseRate << 8);
        pulseRate = (ushort)(pulseRate | pulseRateLByte);

        if(pulseRate >= 300)
        {
            pulseRate = 0;          //脉率值最大不超过 300, 超过 300 则视为无效值,
给其赋 0 即可
        }

        ui->spo2PRLabel->setText(QString::number(pulseRate, 10));

        //血氧饱和度
        spo2Value = packAfterUnpack[5];

        if((0 < spo2Value) && (100 > spo2Value))
        {
            ui->spo2DataLabel->setText(QString::number(spo2Value, 10));
```

```
            }
            else
            {
                ui->spo2DataLabel->setText("---");
            }

            break;

        default:
            break;
        }
    }

    void MainWindow::drawSPO2Wave()
    {
        if(mDrawSPO2)
        {
            mDrawSPO2 = true;
            int spo2Cnt;
            ushort unSPO2Data;
            ushort unSPO2WaveData;

            if(!mDisplayModeFlag)
            {
                spo2Cnt = mSPO2Wave.count() - 1;
            }
            else
            {
                spo2Cnt = 10;
            }

            if(spo2Cnt < 2)
            {
                return;
            }

            QPainter painterSPO2(&mPixmapSPO2);
            painterSPO2.setBrush(Qt::black);
            painterSPO2.setPen(QPen(Qt::black, 1, Qt::SolidLine));

            if(spo2Cnt >= 1080 - mSPO2XStep)
            {
                QRect rct(mSPO2XStep, 0, 1080 - mSPO2XStep, 130);
                painterSPO2.drawRect(rct);
```

```
            QRect rect(0, 0, spo2Cnt + mSPO2XStep - 1080, 130);
            painterSPO2.drawRect(rect);
        }
        else
        {
            QRect rct(mSPO2XStep, 0, spo2Cnt, 130);
            painterSPO2.drawRect(rct);
        }

        painterSPO2.setPen(QPen(Qt::yellow, 2, Qt::SolidLine));

        //监护模式绘图
        if(!mDisplayModeFlag)
        {
            for(int i = 0; i < spo2Cnt; i++)
            {
                QPoint point1(mSPO2XStep, 110 - mSPO2Wave.at(i) / 2.55);
                QPoint point2(mSPO2XStep + 1, 110 - mSPO2Wave.at(i + 1)/ 2.55);

                painterSPO2.drawLine(point1, point2);

                mSPO2XStep++;

                if(mSPO2XStep >= 1080)
                {
                    mSPO2XStep = 1;
                }
            }

            double tail = mSPO2Wave.last();
            mSPO2Wave.clear();
            mSPO2Wave.append(tail);
        }
        //演示模式绘图
        else if(mDisplayModeFlag)
        {
            for(int i = 0; i < spo2Cnt; i++)
            {
                unSPO2Data = mSPO2Wave.at(0);
                unSPO2WaveData = 110 - unSPO2Data / 2.55;
                painterSPO2.drawLine(mSPO2XStep,                mSPO2WaveData,
mSPO2XStep + 1, unSPO2WaveData);
```

```
                    mSPO2WaveData = unSPO2WaveData;
                    mSPO2XStep++;

                    if(mSPO2XStep >= 1080)
                    {
                        mSPO2XStep = 1;
                    }

                    if(mSPO2Wave.length()>1)
                    {
                        mSPO2Wave.removeFirst();
                    }
                }
            }

            ui->spo2WaveLabel->setPixmap(mPixmapSPO2);
        }
    }

    void MainWindow::analyzeECGData(uchar packAfterUnpack[])
    {
        uchar ecg1HByte = 0;                //心电 1 波形高字节
        uchar ecg1LByte = 0;                //心电 1 波形低字节
        uchar ecg2HByte = 0;                //心电 2 波形高字节
        uchar ecg2LByte = 0;                //心电 2 波形低字节
        ushort ecgWave1 = 0;                //心电 1 波形数据
        ushort ecgWave2 = 0;                //心电 2 波形数据
        byte leadV = 0;                     //导联 V 导联信息
        byte leadRA = 0;                    //导联 RA 导联信息
        byte leadLA = 0;                    //导联 LA 导联信息
        byte leadLL = 0;                    //导联 LL 导联信息
        byte hrHByte = 0;                   //心率高字节
        byte hrLByte = 0;                   //心率低字节
        ushort hr = 0;                      //心率
        static byte paceFlag = 0;           //起搏

        switch (packAfterUnpack[1])
        {
        //心电波形
        case DAT_ECG_WAVE:
            ecg1HByte = packAfterUnpack[2];//两个 byte 构成十六位数据
            ecg1LByte = packAfterUnpack[3];
            ecgWave1 = (ushort)(ecgWave1 | ecg1HByte);
            ecgWave1 = (ushort)(ecgWave1 << 8);
```

```
        ecgWave1 = (ushort)(ecgWave1 | ecg1LByte);

        ecg2HByte = packAfterUnpack[4];
        ecg2LByte = packAfterUnpack[5];
        ecgWave2 = (ushort)(ecgWave2 | ecg2HByte);
        ecgWave2 = (ushort)(ecgWave2 << 8);
        ecgWave2 = (ushort)(ecgWave2 | ecg2LByte);

        mECG1WaveNum++;
        mECG2WaveNum++;

        if(mECG1WaveNum == 3) //每 3 个点取 1 个点（数据压缩）
        {
            mECG1WaveNum = 0;
            mECG1Wave.append(ecgWave1);
        }

        if(mECG2WaveNum == 3)   //每 3 个点取 1 个点
        {
            mECG2WaveNum = 0;
            mECG2Wave.append(ecgWave2);
        }

        break;

    //心电导联信息
    case DAT_ECG_LEAD:
        leadLL = (byte)(packAfterUnpack[2] & 0x01);
        leadLA = (byte)(packAfterUnpack[2] & 0x02);
        leadRA = (byte)(packAfterUnpack[2] & 0x04);
        leadV = (byte)(packAfterUnpack[2] & 0x08);     //定义的值

        if(leadLL == 0x01)
        {
            ui->leadLLLabel->setStyleSheet("color:red;");
            ui->leadLLLabel->setText("LL");
        }
        else
        {
            ui->leadLLLabel->setStyleSheet("color:green;");
            ui->leadLLLabel->setText("LL");
        }

        if(leadLA == 0x02)
```

```
        {
            ui->leadLALabel->setStyleSheet("color:red;");
            ui->leadLALabel->setText("LA");
        }
        else
        {
            ui->leadLALabel->setStyleSheet("color:green;");
            ui->leadLALabel->setText("LA");
        }

        if(leadRA == 0x04)
        {
            ui->leadRALabel->setStyleSheet("color:red;");
            ui->leadRALabel->setText("RA");
        }
        else
        {
            ui->leadRALabel->setStyleSheet("color:green;");
            ui->leadRALabel->setText("RA");
        }

        if(leadV == 0x08)
        {
            ui->leadVLabel->setStyleSheet("color:red;");
            ui->leadVLabel->setText("V");
        }
        else
        {
            ui->leadVLabel->setStyleSheet("color:green;");
            ui->leadVLabel->setText("V");
        }
        break;

    //心率
    case DAT_ECG_HR:
        hrHByte = (byte)(packAfterUnpack[2]);
        hrLByte = (byte)(packAfterUnpack[3]);
        hr = (ushort)(hr | hrHByte);
        hr = (ushort)(hr << 8);
        hr = (ushort)(hr | hrLByte);

        ui->heartRateLabel->setAlignment(Qt::AlignHCenter);

        if((0 < hr) && (300 > hr))
```

```
            {
                ui->heartRateLabel->setText(QString::number(hr, 10));
            }
            else
            {
                ui->heartRateLabel->setText("--");
            }

            break;

        default:
            break;
        }
    }

    void MainWindow::drawECG1Wave()
    {
        if(mDrawECG1)
        {
            // mDrawECG1 = false;

            int ecg1Cnt;
            ushort unECG1Data;
            ushort unECG1WaveData;

            if(!mDisplayModeFlag)
            {
                ecg1Cnt = mECG1Wave.count() - 1;
            }
            else
            {
                ecg1Cnt = 10;
            }

            if(ecg1Cnt < 2)
            {
                return;
            }

            QPainter painterECG1(&mPixmapECG1);
            painterECG1.setBrush(Qt::black);
            painterECG1.setPen(QPen(Qt::black, 1, Qt::SolidLine));

            if(ecg1Cnt >= 1080 - mECG1XStep)
```

```
            {
                QRect rct(mECG1XStep, 0, 1080 - mECG1XStep, 130);
                painterECG1.drawRect(rct);

                QRect rect(0, 0, 10 + ecg1Cnt + mECG1XStep - 1080, 130);
                painterECG1.drawRect(rect);
            }
            else
            {
                QRect rct(mECG1XStep, 0, ecg1Cnt + 10, 130);
                painterECG1.drawRect(rct);
            }

            painterECG1.setPen(QPen(Qt::green, 2, Qt::SolidLine));

            //监护模式绘图
            if(!mDisplayModeFlag)
            {
                for(int i = 0; i < ecg1Cnt; i++)
                {
                    QPoint point1(mECG1XStep, 50 - (mECG1Wave.at(i) - 2048) / 18);
                    QPoint point2(mECG1XStep + 1, 50 - (mECG1Wave.at(i + 1) - 2048)
/ 18);

                    painterECG1.drawLine(point1, point2);

                    mECG1XStep++;

                    if(mECG1XStep >= 1080)
                    {
                        mECG1XStep = 1;
                    }
                }

                double tail = mECG1Wave.last();
                mECG1Wave.clear();
                mECG1Wave.append(tail);
            }
            //演示模式绘图
            else if(mDisplayModeFlag)
            {
                for(int i = 0; i < ecg1Cnt; i++)
                {
                    unECG1Data = mECG1Wave.at(0);
```

```
                    unECG1WaveData = 50 - (unECG1Data - 2048) / 18;
                    painterECG1.drawLine(mECG1XStep, mECG1WaveData, mECG1XStep
+ 1, unECG1WaveData);

                    mECG1WaveData = unECG1WaveData;
                    mECG1XStep++;

                    if(mECG1XStep >= 1080)
                    {
                        mECG1XStep = 1;
                    }

                    if(mECG1Wave.length()>1)
                    {
                        mECG1Wave.removeFirst();
                    }

                }
            }

            ui->ecg1WaveLabel->setPixmap(mPixmapECG1);
        }
}

void MainWindow::drawECG2Wave()
{
    if(mDrawECG2)
    {
        //mDrawECG2 = false;

        int ecg2Cnt;
        ushort unECG2Data;
        ushort unECG2WaveData;

        if(!mDisplayModeFlag)
        {
            ecg2Cnt = mECG2Wave.count() - 1;
        }
        else
        {
            ecg2Cnt = 10;
        }

        if(ecg2Cnt < 2)
```

```
        {
            return;
        }

        QPainter painterECG2(&mPixmapECG2);
        painterECG2.setBrush(Qt::black);
        painterECG2.setPen(QPen(Qt::black, 1, Qt::SolidLine));

        if(ecg2Cnt >= 1080 - mECG2XStep)
        {
            QRect rct(mECG2XStep, 0, 1080- mECG2XStep, 131);
            painterECG2.drawRect(rct);
            QRect rect(0, 0, 10 + ecg2Cnt + mECG2XStep - 1080, 131);
            painterECG2.drawRect(rect);
        }
        else
        {
            QRect rct(mECG2XStep, 0, ecg2Cnt + 10, 131);
            painterECG2.drawRect(rct);
        }

        painterECG2.setPen(QPen(Qt::blue, 2, Qt::SolidLine));

        //监护模式绘图
        if(!mDisplayModeFlag)
        {
            for(int i = 0; i < ecg2Cnt; i++)
            {
                QPoint point1(mECG2XStep, 50 - (mECG2Wave.at(i) - 2048) / 18);
                QPoint point2(mECG2XStep + 1,50 - (mECG2Wave.at(i + 1) - 2048) /
18);

                painterECG2.drawLine(point1, point2);

                mECG2XStep++;

                if(mECG2XStep >= 1080)
                {
                    mECG2XStep = 1;
                }
            }

            double tail = mECG2Wave.last();
            mECG2Wave.clear();
```

```
                mECG2Wave.append(tail);
            }
            //演示模式绘图
            else if(mDisplayModeFlag)
            {
                for(int i = 0; i < ecg2Cnt; i++)
                {
                    unECG2Data = mECG2Wave.at(0);
                    unECG2WaveData = 50 - (unECG2Data - 2048) / 18;
                    painterECG2.drawLine(mECG2XStep, mECG2WaveData, mECG2XStep
+ 1, unECG2WaveData);
                    mECG2WaveData = unECG2WaveData;
                    mECG2XStep++;

                    if(mECG2XStep >= 1080)
                    {
                        mECG2XStep = 1;
                    }
                    if(mECG2Wave.length()>1)
                    {
                        mECG2Wave.removeFirst();
                    }
                }
            }

            ui->ecg2WaveLabel->setPixmap(mPixmapECG2);
        }
    }

    void MainWindow::timerEvent(QTimerEvent *event)
    {
        if(event->timerId() == mTimer)
        {
            drawRespWave();
            drawSPO2Wave();
            drawECG1Wave();
            drawECG2Wave();
        }
    }

    void MainWindow::recPrbType(QString prbType, QByteArray data)
    {
        mPrbType = prbType;
        writeSerial(data);
```

```
}

void MainWindow::recNIBPSetData(QString mode)
{
    mNIBPMode = mode;
    ui->nibpModeLabel->setText(mode);
}

void MainWindow::recRespGain(QString gain, QByteArray data)
{
    mRespGain = gain;
    writeSerial(data);
}

void MainWindow::recSPO2Data(QString sens, QByteArray data)
{
    mSPO2Sens = sens;
    writeSerial(data);
}

void MainWindow::recECGData(int lead1, int gain1, int lead2, int gain2, QByteArray data)
{
    mECG1Lead = lead1;
    mECG1Gain = gain1;
    mECG2Lead = lead2;
    mECG2Gain = gain2;
    writeSerial(data);
}
```

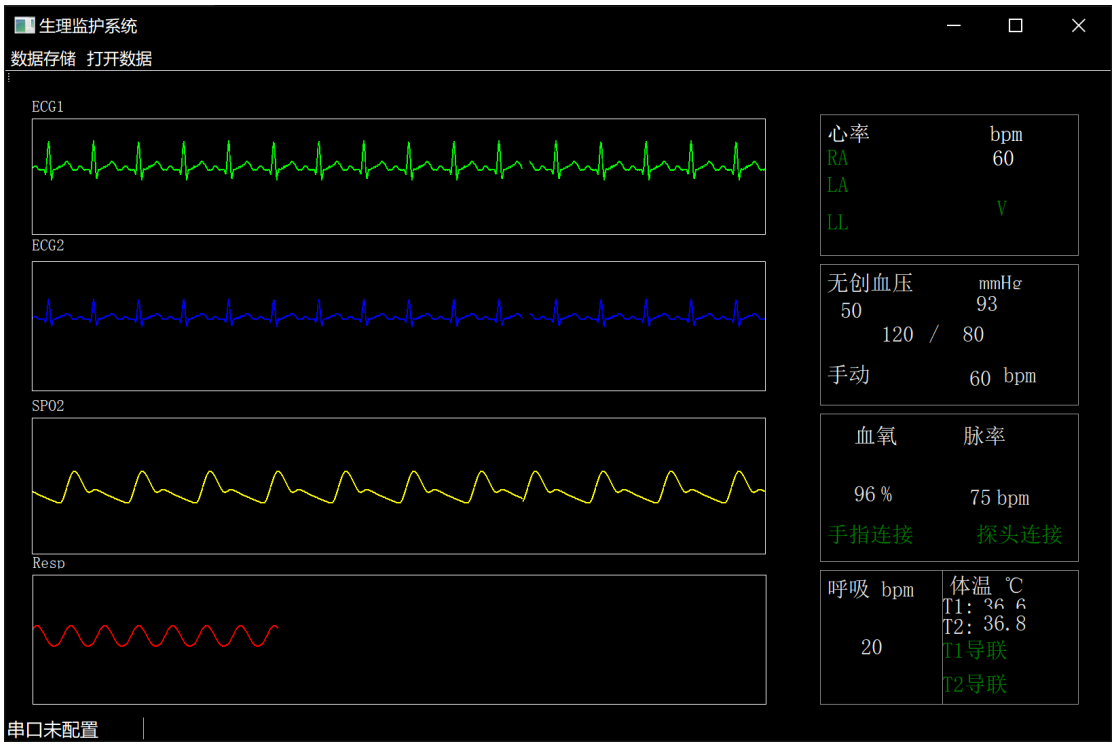
打开文件算法

（接显示算法后面）

```
void MainWindow::menuOpen()
{
    QString fileName = QFileDialog::getOpenFileName(this,"CSV Files","*.csv",0);   //打开，查找 csv 格式文件
    if(!fileName.isNull())
    {
        //filename 是文件名
        qDebug() << fileName;

        QFile csvFile(fileName);
```

```
            mCSVList.clear();

            if(csvFile.open(QIODevice::ReadWrite))     //对数据进行读写
            {
                QTextStream stream(&csvFile);
                int i = 0;
                //QByteArray mDataList;
                while(!stream.atEnd())
                {
                    mCSVList.push_back(stream.readLine());
                    QString dataLine=mCSVList.at(i);
                    QStringList dataList=dataLine.split(","); //以逗号间隔开，分割为独立
的一个个字符串
                    //qDebug()<<dataList;
                    mRxData.clear();
                    for (int j=0;j<dataList.length()-1;j++)     //将数据一个个读取进来
                    {
                        QString str=dataList.value(j);
                        //qDebug()<<str<<"o";
                        mRxData.append(str.toUShort());  //字符转换成数字，byte，因
为之前的 mRxData 也是 byte，要转换成一致
                        //qDebug()<<mRxData.at(j)<<"k";
                    }

                    //qDebug()<<mRxData;

                    //qDebug()<<mDataList.at(1);
                    //mSPO2Wave.clear();
                    procUARTData();//PCT 解包
                    dealRcvPackData();//数据分类保存
                    i++;
                }
                csvFile.close();
            }

        }
    }
```

设计结果



心得体会

在本次实训设计中，设计一个生理学信号监护系统。在设计过程中遇到了很多的问题和困难，例如针对 PCT 解包程序设计时就遇到了瓶颈，因为其 PCT 解包对我来说是初次遇见，其原理十分复杂，因其在对传输过来的数据要进行地址定位，并将数据进行分装，这在编写程序时所需要的代码量十分庞大，所以在编写的过程中十分容易出现错误。然后就是分别传输解包并且分装好的数据，此时其每种数据类型包含的数据量也是十分多的，所以其发送数据时要分清其数据类型以及需要显示数据的位置。然后在描绘波形时也是有很多的问题，例如其绘制时需要保留之前绘制图形，在到达终点后需要从头重新绘制，并且绘制时需要确定合适的 x 轴 y 轴间距。然后再利用定时器刷新数据并绘制图形，而其中需要绘制四种图形，每种图形需要接受不同数据，期间也遇到了很多的问题。最后在编写打开文件的数据时也与了不少的问题，而且这个程序十分重要，所以也花费了大量时间去解决。

在设计此次实训程序时，接受了很多人的帮助，并且在网上也查询了许多资料，当最后程序成功运行出来时，感觉之前的努力都是值得的。