Назначения программы

Программно-аппаратный комплекс предназначен для:

1) Мониторинга температуры в помещении с использованием двух датчиков PTC_NICKEL.
2) Вычисления среднего значения температуры.
3) Управление реле (имитация открытия/закрытия окна) при достижении критических значений.
4) Визуализации данных на ПК с цветовой индексацией состояния температуры.
5) Сохранения и загрузки пороговых значений температуры в XML-файл.

Условия выполнения программы

Аппаратные требования:

1) Микроконтроллер AT89C51.
2) Датчик PTC_NICKEL в количестве двух штук.
3) АЦП ADC_8.
4) Реле RLY-SPNO и LRLY1COILSPDT.
5) Светодиод LED-BIGY.
6) Резисторы на 300 Ом, 5 кОм, 3 кОм.
7) Питание 5 В.
8) Биполярный транзистор типа NPN.

Программные требования:

1) Приложения ПК Proteus 8 и QT.

Выполнение программы

Последовательность действий оператора:

1) Запустить файл программно-аппаратного комплекса в Proteus.
2) Запустить файл программы QT.
3) В программе QT необходимо:
    a. Выбрать необходимый язык (рус/анг/нем).
    b. Нажать на кнопку “Подключить порт”.
    c. При необходимости поменять пороговые значения температуры в формате числа с плавающей точкой. Затем нажать на кнопку “Установить” для сохранения пороговых значений в XML-файл.
4) При необходимости поменять данные температуры на датчиках PTC_NICKEL в Proteus.

Сообщения оператору

Цветовая индикация температуры:

1) Если средняя температура с датчиков меньше минимальной температуры, то цвет температуры будет отображаться синим цветом, а реле будет разомкнуто.
2) Если средняя температура датчиков находится в диапазоне между пороговыми значениями, то цвет температуры будет отображаться зеленым цветом, реле будет разомкнуто.
3) Если средняя температура датчиков больше максимально температуры, то температура будет отображаться красным цветом, а реле будет замкнуто и будет гореть светодиод.

Приложение

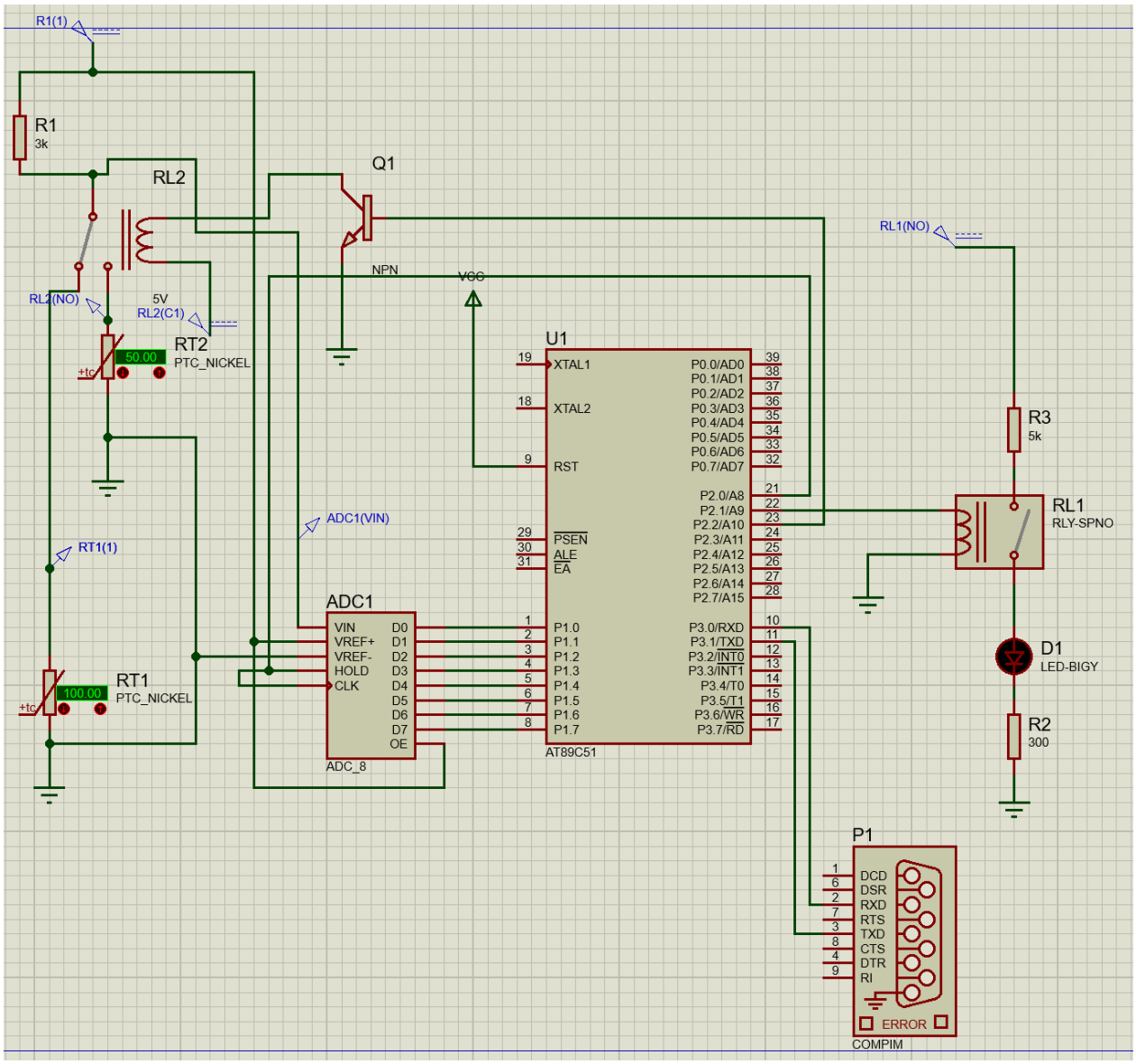

Рисунок 1. Схема программно-аппаратного комплекса

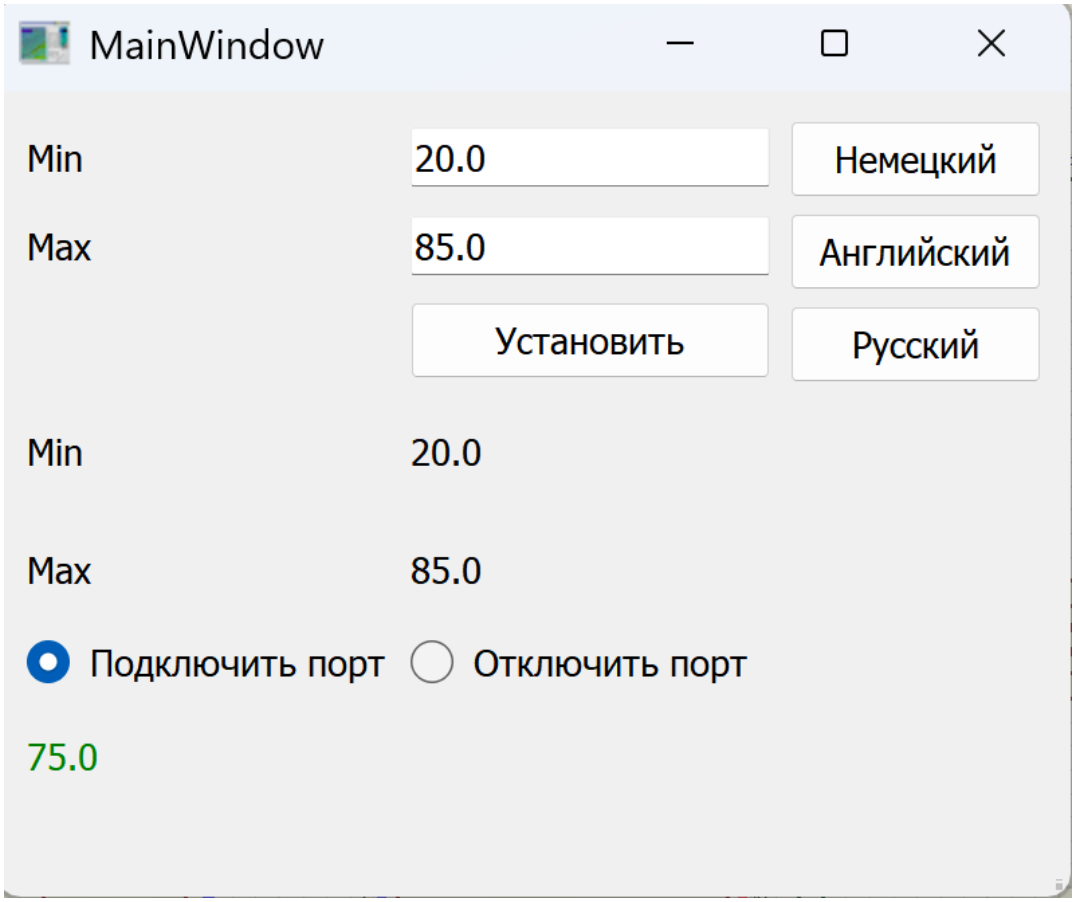

Рисунок 2. Вывод температуры (удовлетворяет диапазону)

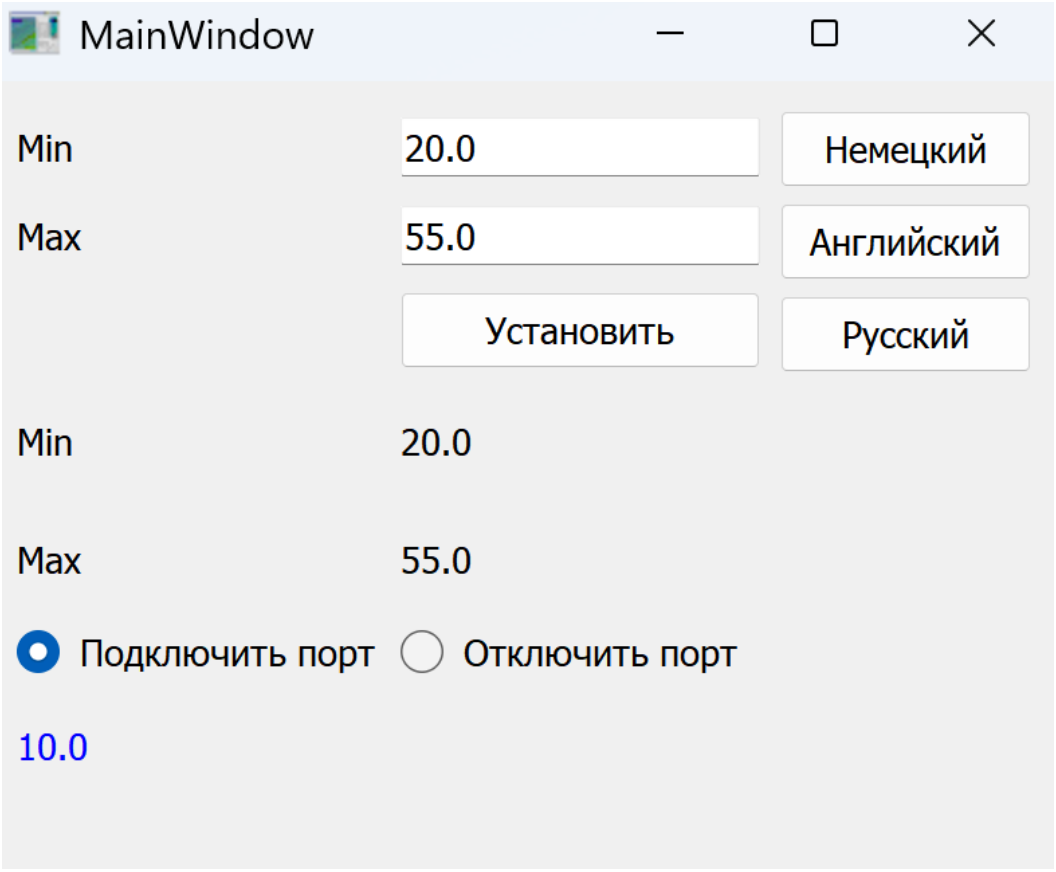

Рисунок 3. Вывод температуры (находится ниже минимальной)

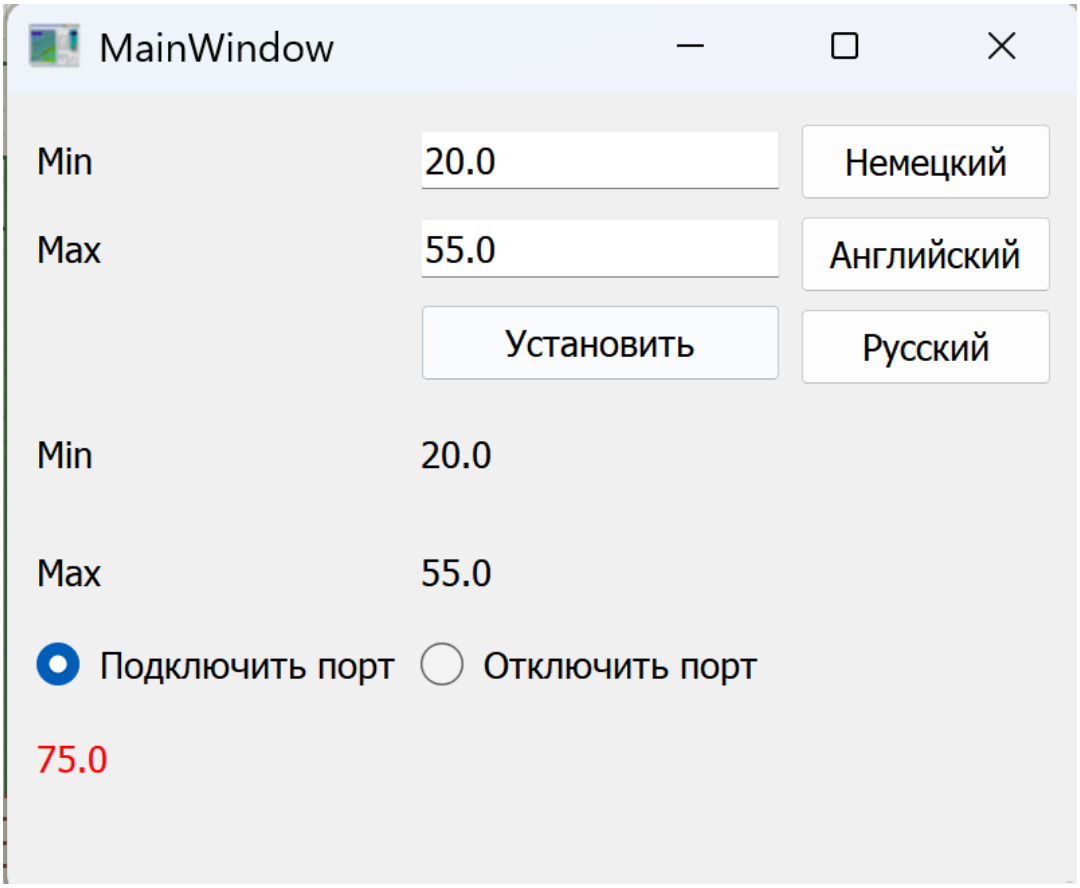

Рисунок 4. Вывод температуры (находится выше максимальной)

Листиниг программно-аппаратного комплекса

```
#include <mcs51reg.h>
#include <stdio.h>
#include <ctype.h>
#include <stdint.h>

#define RELAY P2_1
#define SIGNAL  P2_0
#define P22 P2_2

#define seg7 P0

#define R1 3000.0
#define MAX_ADC 255.0

#define UART_BUF_SIZE 5

// доп задание
const unsigned char  seg_codes[] = {
    0xC0,
    0xF9,
    0xA4,
    0xB0,
    0x99,
    0x92,
    0x82,
    0xF8,
    0x80,
    0x90
};
// доп задание
volatile unsigned char received_data;
void seg(void);

float max_temp;  // max temp from uart
```

```c
float temp; // temp (t1+t2) / 2

uint8_t value_ptc_1; // value ptc 1 from adc
uint8_t value_ptc_2; // value ptc 2 from adc
float r_ptc_1; // res from ptc 1
float r_ptc_2; // res from ptc 2
float t1; // temp from ptc 1
float t2; // temp from ptc 2


char uart_buf[UART_BUF_SIZE]; // for output
unsigned char uart_index = 0;
__bit uart_line_ready = 0;

void uart_send_str(char *str); // send string
void send_float(float value); // send float -> string

void read_adc(void); // read adc

float adc_to_resistance(uint8_t adc_val); // convert adc value to res
float ptc_temp(float r_ptc); // convert res to temp
float adc_to_temp(uint8_t val_1, uint8_t val_2); // return main temp

void delay_ms(unsigned int ms);
void check_rly(void);

float parse_float(const char *str);
void parse_uart_line(char *buf);

// доп задание
void seg(void){
    unsigned char tmp  = (unsigned char)temp;
    seg7 = seg_codes[tmp / 10];
   delay_ms(100);
   seg7 = seg_codes[tmp % 10];

}

void init_serial() {
   TMOD = 0x20;
   TH1 = 0xFD;
   SCON = 0x50;
   TR1 = 1;
   ES = 1;
   EA = 1;
}

void uart_send_str(char *str) {
   while (*str) {
       SBUF = *str++;
       while (!TI);
       TI = 0;
    }
}

void send_float(float value) {
   int t = (int)(value * 10 + 0.5);
   char buf[8];
   sprintf(buf, "%d.%d", t / 10, t % 10);
   uart_send_str(buf);
}

void read_adc(void) {
   SIGNAL = 1;
   value_ptc_1= P1;
   SIGNAL = 0;

   P22 = 0;
```

```
    delay_ms(100);
    SIGNAL = 1;
    value_ptc_2 = P1;
    P22 = 1;
    SIGNAL = 0;

}

float ptc_temp(float resistance) {
    float temp = (-0.0000548791 * resistance * resistance) + (0.348705 *
resistance) - 270.10231;
    return temp;
}

float adc_to_resistance(uint8_t adc_val) {
    float v_ptc = ((float)adc_val / MAX_ADC) * 5.0;
    if (v_ptc >= 5.0) v_ptc = 4.99;
    return R1 * v_ptc / (5.0 - v_ptc);
}

float adc_to_temp(uint8_t val_1, uint8_t val_2){
    r_ptc_1 = adc_to_resistance(val_1);
    r_ptc_2 = adc_to_resistance(val_2);

    t1 = ptc_temp(r_ptc_1);
    t2 = ptc_temp(r_ptc_2);

    return (t1 + t2) / 2.0;
}

float parse_float(const char *str) {
    float result = 0;
    float fraction = 0.1;
    char decimal_flag = 0;
    int i = 0;

    for (; str[i] != '\0'; i++) {
        if (str[i] == '.') {
            decimal_flag = 1;
        }
        else if (str[i] >= '0' && str[i] <= '9') {
            if (decimal_flag) {
                result += (str[i] - '0') * fraction;
                fraction *= 0.1;
            }
            else {
                result = result * 10 + (str[i] - '0');
            }
        }
    }

    return result;
}

void parse_uart_line(char *buf) {
    max_temp = parse_float(buf);
    send_float(max_temp);
}

void uart_isr(void) __interrupt(4) {
   char c;
    if (RI) {
        RI = 0;
        c = SBUF;

        if (c == '\r' || c == '\n') {
            uart_buf[uart_index] = '\0';
            uart_line_ready = 1;
```

```
            uart_index = 0;
        }
        else if (c == '\b' && uart_index > 0) {
            uart_index--;
        }
        else if (uart_index < UART_BUF_SIZE - 1 &&
                (isdigit(c) || c == '.')) {
            uart_buf[uart_index++] = c;
        }
    }
}

void delay_ms(unsigned int ms) {
   unsigned int i, j;
   for (i = 0; i < ms; i++)
       for (j = 0; j < 1200; j++);
}

void check_rly(void){
   if (temp > max_temp) {
       RELAY = 1;
   }
   else {
       RELAY = 0;
   }
}

void main() {
   P1 = 0xFF;
   RELAY = 0;
   max_temp = 0.0;

   seg7= 0xFF;

   init_serial();

   while (1) {
       if (uart_line_ready) {
           uart_line_ready = 0;
           parse_uart_line(uart_buf);
       }

       read_adc();
       temp = adc_to_temp(value_ptc_1, value_ptc_2);

       check_rly();
       send_float(temp);
       uart_send_str("\r\n");
       seg(); // доп задание
       delay_ms(300);
   }
}
```

Листинг программы приложения

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTranslator>
#include <QtSerialPort/QSerialPort>
#include <QtSerialPort/QSerialPortInfo>
#include <QFile>
#include <QDomDocument>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE
```

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private slots:
    void readSerialData();

    void on_pushButton_set_temp_clicked();

    void on_pushButton_ger_clicked();

    void on_pushButton_eng_clicked();

    void on_pushButton_rus_clicked();

    void on_radioButton_off_clicked();

    void on_radioButton_on_clicked();

    QPair<double, double> readXML();

    void writeXML();

    void updateMinMaxLabels();


private:
    Ui::MainWindow *ui;
    QTranslator translator;
    QString currentLang = "ru";
    void switchLanguage(const QString &languageCode);
    QSerialPort *serial;
    QByteArray receivedData;
    QFile file;
    QDomElement maxElem;

};
#endif // MAINWINDOW_H


#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QtDebug>
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
    , serial(new QSerialPort(this))
    , file("temp.xml")
{
    ui->setupUi(this);
    connect(serial, &QSerialPort::readyRead, this,
&MainWindow::readSerialData);
    connect(ui->pushButton_rus, &QPushButton::clicked, this,
&MainWindow::on_pushButton_rus_clicked);
    connect(ui->pushButton_ger, &QPushButton::clicked, this,
&MainWindow::on_pushButton_ger_clicked);
    connect(ui->pushButton_eng, &QPushButton::clicked, this,
&MainWindow::on_pushButton_eng_clicked);
    updateMinMaxLabels();
}

MainWindow::~MainWindow()
{
```

```
    delete ui;
}

QPair<double, double> MainWindow::readXML() {
    double minTemp = 0, maxTemp = 0;

    if (!file.open(QIODevice::ReadOnly)) {
        return qMakePair(minTemp, maxTemp);
    }

    QDomDocument doc;
    if (!doc.setContent(&file)) {

        file.close();
        return qMakePair(minTemp, maxTemp);
    }
    file.close();

    QDomElement root = doc.documentElement();
    QDomElement minElem = root.firstChildElement("Min");
    QDomElement maxElem = root.firstChildElement("Max");

    if (!minElem.isNull())
        minTemp = minElem.text().toDouble();

    if (!maxElem.isNull())
        maxTemp = maxElem.text().toDouble();

    return qMakePair(minTemp, maxTemp);
}

void MainWindow::updateMinMaxLabels()
{
    QPair<double, double> minMax = readXML();
    double minTemp = minMax.first;
    double maxTemp = minMax.second;

    ui->label_5->setText(QString::number(minTemp, 'f', 1));
    ui->label_6->setText(QString::number(maxTemp, 'f', 1));
}


void MainWindow::writeXML() {
    QDomDocument doc;

    QDomElement root = doc.createElement("TemperatureSettings");
    doc.appendChild(root);

    QDomElement minElem = doc.createElement("Min");
    QDomText minText = doc.createTextNode(ui->set_min_temp->text());
    minElem.appendChild(minText);

    QDomElement maxElem = doc.createElement("Max");
    QDomText maxText = doc.createTextNode(ui->set_max_temp->text());
    maxElem.appendChild(maxText);

    root.appendChild(minElem);
    root.appendChild(maxElem);

    if (!file.open(QIODevice::WriteOnly | QIODevice::Truncate)) {
        qDebug() << "failed to open xml";
        return;
    }

    QTextStream stream(&file);
    stream << doc.toString();
    file.close();
    updateMinMaxLabels();
```

```
}


void MainWindow::readSerialData()
{
    static QString buffer;
    buffer += QString::fromUtf8(serial->readAll());

    while (buffer.contains('\n')) {
        int endIndex = buffer.indexOf('\n');
        QString line = buffer.left(endIndex).trimmed();
        buffer = buffer.mid(endIndex + 1);

        qDebug() << "line:" << line;

        QStringList values = line.split(',');

        if (values.size() <= 3) {
            bool ok;
            double temp_avg = values[0].toDouble(&ok);

            if (ok) {

                ui->get_info->setText(QString::number(temp_avg, 'f', 1));

                QPair<double, double> minMax = readXML();
                double minTemp = minMax.first;
                double maxTemp = minMax.second;

                if (temp_avg > maxTemp) {
                    ui->get_info->setStyleSheet("QLabel { color : red; }");
                }
                else if (temp_avg < minTemp) {
                    ui->get_info->setStyleSheet("QLabel { color : blue; }");
                }
                else {
                    ui->get_info->setStyleSheet("QLabel { color : green; }");
                }
            } else {
                qDebug() << "conversion failed" << values;
            }
        } else {
            qDebug() << "invalid data format";
        }
    }
    updateMinMaxLabels();
}

void MainWindow::on_pushButton_set_temp_clicked()
{
    auto minTemp = ui->set_min_temp->text().toDouble();
    auto maxTemp = ui->set_max_temp->text().toDouble();

    if (minTemp >= maxTemp) {
        qDebug() << "min can be less then max";
        return;
    }
    writeXML();

    if (!serial->isOpen()) {
        qDebug() << "port off";
        return;
    }

    QString tempValue = QString::number(maxTemp, 'f', 1);
    QByteArray dataToSend = tempValue.toUtf8();
    dataToSend.append('\n');
```

```
    serial->write(dataToSend);
}

void MainWindow::switchLanguage(const QString &languageCode)
{
    if (currentLang == languageCode)
        return;

    qApp->removeTranslator(&translator);

    if (languageCode == "en") {

translator.load("C:/Mac/Home/Documents/finalproject/finalproject_en.qm");
    } else if (languageCode == "en_GB") {

translator.load("C:/Mac/Home/Documents/finalproject/finalproject_en_GB.qm");
    } else {
        currentLang = "ru";
        qApp->removeTranslator(&translator);
        ui->retranslateUi(this);
        return;
    }

    qApp->installTranslator(&translator);
    ui->retranslateUi(this);
    currentLang = languageCode;
}


void MainWindow::on_pushButton_eng_clicked()
{
    switchLanguage("en");
}

void MainWindow::on_pushButton_ger_clicked()
{
    switchLanguage("en_GB");
}

void MainWindow::on_pushButton_rus_clicked()
{
    switchLanguage("ru");
}

void MainWindow::on_radioButton_off_clicked()
{
    if (serial->isOpen()) {
        serial->close();
        qDebug() << "off";
    }

}

void MainWindow::on_radioButton_on_clicked()
{
    if (serial->isOpen()) {
        qDebug() << "Port already open";
        return;
    }

    serial->setPortName("COM1");
    serial->setBaudRate(QSerialPort::Baud9600);
    serial->setDataBits(QSerialPort::Data8);
    serial->setStopBits(QSerialPort::OneStop);

    if (serial->open(QIODevice::ReadWrite)) {
        qDebug() << "cool";
    } else {
```

```
            qDebug() << "not cool";
        }

        QPair<double, double> minMax = readXML();
        double maxTemp = minMax.second;

        QString tempValue = QString::number(maxTemp, 'f', 1);
        QByteArray dataToSend = tempValue.toUtf8();
        dataToSend.append('\n');
        serial->write(dataToSend);
}
```