ДОДАТОК А

(Обов’язковий)

КОД ПРОГРАМИ

Лістинг програми до завдання

Main.cpp

```
#include <QApplication> #include
"MainWindow.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();  return
    a.exec();
}
```

MainWindow.h

```
#pragma once

#include <QMainWindow> #include
"ConnectionChecker.h" #include
"Setting.h"
#include "TemperatureTab.h"
#include "PhotoresistorTab.h"
#include "LEDTab.h"
#include "UltrasonicTab.h"
#include  <QLabel> #include
<QSerialPort> #include
<QSerialPortInfo> #include
<QTabWidget> #include
<QString> #include <QTimer>

#include "xlsxdocument.h"

class MainWindow : public QMainWindow
{
    Q_OBJECT
```

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

private:
    QLabel* status; ConnectionChecker*
    connectChecker; QTimer* timer;
    Setting* portSettings; QSerialPort*
    comPort; QSerialPortInfo
    comPortInfo; QTabWidget* tabs;
    QString buffer;

    bool isSavingData;
    void saveLEDTab(LEDTab* widget, const QString& name, QXlsx::Document& doc);
    void savePhotoresistorTab(PhotoresistorTab* widget, const QString& name,
QXlsx::Document& doc);
    void saveTemperatureTab(TemperatureTab* widget, const QString& name,
QXlsx::Document& doc);
    void saveUltrasonicTab(UltrasonicTab* widget, const QString& name,
QXlsx::Document& doc);

private slots:
    void openSettings(); void
    saveData();

    void updateStatus(bool isConnected); void
    updatePortName(QString portName); void
    updateBaudRate(QString baudRate); void
    updateDataBits(QString dataBits); void
    updateStopBits(QString stopBits); void
    updateParity(QString parity);
    void updateFlowControl(QString flowControl);
    void updateTimeout(QString timeout);

    void readSerial();
};
```

MainWindow.cpp

```
#include "MainWindow.h"

#include <QMenuBar>

#include <QStatusBar>

#include <QMessageBox>

#include <QList> #include

<QString> #include

<QVariant> #include

<QFileDialog> #include

<QFontMetrics> #include

<QSize>


#include "xlsxcell.h" #include

"xlsxchart.h" #include

"xlsxcellrange.h"


MainWindow::MainWindow(QWidget *parent)

    : QMainWindow(parent)

    , connectChecker(new ConnectionChecker)

    , timer(new QTimer(this))

    , portSettings(new Setting)

    , buffer("")

    , isSavingData(false)

{

    timer->start(1000);


    comPort = new QSerialPort(this);

    comPort->setPortName("");

    comPort->setBaudRate(QSerialPort::Baud9600);

    comPort->setDataBits(QSerialPort::Data8);

    comPort->setStopBits(QSerialPort::OneStop);

    comPort->setParity(QSerialPort::NoParity);

    comPort->setFlowControl(QSerialPort::NoFlowControl);


    status = new QLabel("Disconnected", this);

    statusBar()->addWidget(status);


    menuBar()->addAction("Settings", this, SLOT(openSettings()));

    menuBar()->addAction("Save Data", this, SLOT(saveData()));
```

```
    tabs = new QTabWidget(this);
    setCentralWidget(tabs);
    connect(connectChecker,            SIGNAL(connectionChanged(bool)),            portSettings,
SLOT(changePortList(bool)));
    connect(connectChecker,               SIGNAL(connectionChanged(bool)),               this,
SLOT(updateStatus(bool)));

    connect(portSettings,               SIGNAL(portChangedSignal(QString)),               this,
SLOT(updatePortName(QString)));
    connect(portSettings,             SIGNAL(baudRateChangedSignal(QString)),               this,
SLOT(updateBaudRate(QString)));
    connect(portSettings,             SIGNAL(dataBitsChangedSignal(QString)),               this,
SLOT(updateDataBits(QString)));
    connect(portSettings,             SIGNAL(stopBitsChangedSignal(QString)),               this,
SLOT(updateStopBits(QString)));
     connect(portSettings,               SIGNAL(parityChangedSignal(QString)),               this,
  SLOT(updateParity(QString)));
     connect(portSettings,             SIGNAL(flowControlChangedSignal(QString)),             this,
SLOT(updateFlowControl(QString)));
    connect(portSettings,        SIGNAL(saveDataTimeoutChangedSignal(QString)),            this,
SLOT(updateTimeout(QString)));
    //connect(portSettings,       SIGNAL(saveDataTimeoutChangedSignal(QString)),          this,
SLOT(updateTimeout(QString)));

    connect(comPort, SIGNAL(readyRead()), this, SLOT(readSerial()));

    resize(350, 235);
    setWindowTitle("COM Port Data Monitor");
}

void MainWindow::saveLEDTab(LEDTab *widget, const QString& name, QXlsx::Document& doc)
{
    LEDTab* led = qobject_cast<LEDTab*>(widget);
    QList<int> values = led->getValues();
    QList<QString> dates = led->getDateTime();

    doc.addSheet(name);
```

```
    doc.write(1, 1, QVariant("Date and Time"));
    doc.write(1, 2, QVariant("Value"));

    QXlsx::Cell* dateCell = doc.cellAt(1, 1);
    QXlsx::Cell* valueCell = doc.cellAt(1, 2);
    double dateWidth = QFontMetrics(dateCell->format().font()).horizontalAdvance(dateCell->value().toString());
    double valueWidth = QFontMetrics(valueCell->format().font()).horizontalAdvance(valueCell->value().toString());

    for (int row = 0; row < values.size(); ++row)
    {
        doc.write(row + 2, 1, QVariant(dates[row].replace('T', ' ')));
        doc.write(row + 2, 2, QVariant(values[row]));

        QXlsx::Cell *firstCell = doc.cellAt(row + 2, 1); QXlsx::Cell
        *secondCell = doc.cellAt(row + 2, 2); QFontMetrics
        metrics(firstCell->format().font());

        dateWidth = qMax(dateWidth, static_cast<double>(metrics.horizontalAdvance(firstCell->value().toString())));
        valueWidth = qMax(valueWidth, static_cast<double>(metrics.horizontalAdvance(secondCell->value().toString())));
    }

    doc.setColumnWidth(1, dateWidth / 5);
    doc.setColumnWidth(2, valueWidth / 5);

    QXlsx::Chart* chart = doc.insertChart(1, 3, QSize(400, 300));
    chart->setChartType(QXlsx::Chart::CT_LineChart);
    //chart->addSeries(QXlsx::CellRange(2, 1, dates.size() + 1, 1));
    chart->addSeries(QXlsx::CellRange(2, 2, dates.size() + 1, 2));
}

void MainWindow::savePhotoresistorTab(PhotoresistorTab *widget, const QString& name,
QXlsx::Document& doc)
{
```

```
    PhotoresistorTab* photoresistor = qobject_cast<PhotoresistorTab*>(widget); QList<int>
    values = photoresistor->getValues();
    QList<QString> dates = photoresistor->getDateTime();

    doc.addSheet(name);
    doc.write(1, 1, QVariant("Date and Time"));
    doc.write(1, 2, QVariant("Value"));

    QXlsx::Cell* dateCell = doc.cellAt(1, 1);
    QXlsx::Cell* valueCell = doc.cellAt(1, 2);
    double dateWidth = QFontMetrics(dateCell-
>format().font()).horizontalAdvance(dateCell->value().toString());
    double valueWidth = QFontMetrics(valueCell-
>format().font()).horizontalAdvance(valueCell->value().toString());

    for (int row = 0; row < values.size(); ++row)
    {
        doc.write(row + 2, 1, QVariant(dates[row].replace('T', ' ')));
        doc.write(row + 2, 2, QVariant(values[row]));

        QXlsx::Cell *firstCell = doc.cellAt(row + 2, 1); QXlsx::Cell
        *secondCell = doc.cellAt(row + 2, 2); QFontMetrics
        metrics(firstCell->format().font());

        dateWidth = qMax(dateWidth,
static_cast<double>(metrics.horizontalAdvance(firstCell->value().toString())));
        valueWidth = qMax(valueWidth,
static_cast<double>(metrics.horizontalAdvance(secondCell->value().toString())));
    }

    doc.setColumnWidth(1, dateWidth / 5);
    doc.setColumnWidth(2, valueWidth / 5);

    QXlsx::Chart* chart = doc.insertChart(1, 3, QSize(400, 300));
    chart->setChartType(QXlsx::Chart::CT_LineChart);
    chart->addSeries(QXlsx::CellRange(2, 1, dates.size() + 1, 1));
    chart->addSeries(QXlsx::CellRange(2, 2, dates.size() + 1, 2));
}
```

```
void MainWindow::saveTemperatureTab(TemperatureTab *widget, const QString& name,
QXlsx::Document& doc)
{
    TemperatureTab* dht = qobject_cast<TemperatureTab*>(widget);
    QList<double> temperature = dht->getTemperature(); QList<double>
    humidity = dht->getHumidity();

    QList<QString> dates = dht->getDateTime();

    doc.addSheet(name);

    doc.write(1, 1, QVariant("Date and Time"));
    doc.write(1, 2, QVariant("Temperature"));
    doc.write(1, 3, QVariant("Humidity"));

    QXlsx::Cell* dateCell = doc.cellAt(1, 1);
    QXlsx::Cell* tempCell = doc.cellAt(1, 2);
    QXlsx::Cell* humCell = doc.cellAt(1, 3);
    double dateWidth = QFontMetrics(dateCell-
>format().font()).horizontalAdvance(dateCell->value().toString());
    double tempWidth = QFontMetrics(tempCell-
>format().font()).horizontalAdvance(tempCell->value().toString());
    double humWidth = QFontMetrics(humCell-
>format().font()).horizontalAdvance(humCell->value().toString());

    for (int row = 0; row < dates.size(); ++row)
    {
        doc.write(row + 2, 1, QVariant(dates[row].replace('T', ' ')));
        doc.write(row + 2, 2, QVariant(temperature[row]));
        doc.write(row + 2, 3, QVariant(humidity[row]));

        QXlsx::Cell *firstCell = doc.cellAt(row + 2, 1); QXlsx::Cell
        *secondCell = doc.cellAt(row + 2, 2); QXlsx::Cell *thirdCell =
        doc.cellAt(row + 2, 3); QFontMetrics metrics(firstCell-
        >format().font());

        dateWidth = qMax(dateWidth,
```

```
static_cast<double>(metrics.horizontalAdvance(firstCell->value().toString()));
        tempWidth = qMax(tempWidth,
static_cast<double>(metrics.horizontalAdvance(secondCell->value().toString()));
        humWidth = qMax(humWidth,
static_cast<double>(metrics.horizontalAdvance(thirdCell->value().toString()));
    }

    doc.setColumnWidth(1,  dateWidth  /  5);
    doc.setColumnWidth(2,  tempWidth  /  5);
    doc.setColumnWidth(3, humWidth / 5);

    QXlsx::Chart* tempChart = doc.insertChart(1, 4, QSize(400, 300));
    tempChart->setChartType(QXlsx::Chart::CT_LineChart);
    //tempChart->addSeries(QXlsx::CellRange(2, 1, dates.size() + 1, 1));
    tempChart->addSeries(QXlsx::CellRange(2, 2, dates.size() + 1, 2));

    QXlsx::Chart* humChart = doc.insertChart((300 / 16), 4, QSize(400, 300));
    humChart->setChartType(QXlsx::Chart::CT_LineChart);
    //humChart->addSeries(QXlsx::CellRange(2, 1, dates.size() + 1, 1));
    humChart->addSeries(QXlsx::CellRange(2, 3, dates.size() + 1, 3));
}

void  MainWindow::saveUltrasonicTab(UltrasonicTab  *widget,  const  QString  &name,
QXlsx::Document &doc)
{
    UltrasonicTab* ultrasonic = qobject_cast<UltrasonicTab*>(widget); QList<int>
    values = ultrasonic->getValues();
    QList<QString> dates = ultrasonic->getDateTime();

    doc.addSheet(name);

    doc.write(1, 1, QVariant("Date and Time"));
    doc.write(1, 2, QVariant("Distance, cm"));

    QXlsx::Cell* dateCell = doc.cellAt(1, 1);
    QXlsx::Cell* valueCell = doc.cellAt(1, 2);
    double  dateWidth  =  QFontMetrics(dateCell-
>format().font()).horizontalAdvance(dateCell->value().toString());
```

```
    double valueWidth = QFontMetrics(valueCell->format().font()).horizontalAdvance(valueCell->value().toString());

    for (int row = 0; row < values.size(); ++row)
    {
        doc.write(row + 2, 1, QVariant(dates[row].replace('T', ' ')));
        doc.write(row + 2, 2, QVariant(values[row]));

        QXlsx::Cell *firstCell = doc.cellAt(row + 2, 1); QXlsx::Cell
        *secondCell = doc.cellAt(row + 2, 2); QFontMetrics
        metrics(firstCell->format().font());

        dateWidth = qMax(dateWidth, static_cast<double>(metrics.horizontalAdvance(firstCell->value().toString())));
        valueWidth = qMax(valueWidth, static_cast<double>(metrics.horizontalAdvance(secondCell->value().toString())));
    }

    doc.setColumnWidth(1, dateWidth / 5);
    doc.setColumnWidth(2, valueWidth / 5);

    QXlsx::Chart* chart = doc.insertChart(1, 3, QSize(400, 300));
    chart->setChartType(QXlsx::Chart::CT_LineChart);
    //chart->addSeries(QXlsx::CellRange(2, 1, dates.size() + 1, 1));
    chart->addSeries(QXlsx::CellRange(2, 2, dates.size() + 1, 2));
}

void MainWindow::openSettings()
{
    portSettings->show();
}

void MainWindow::saveData()
{
    if (tabs->count() == 0)
    {
        QMessageBox::information(nullptr, "Warning", "No data to save"); return;
    }
```

```
    QString filePath = QFileDialog::getSaveFileName(nullptr, "Save Excel File",
QDir::currentPath(), "Excel Files (*.xlsx)");
    if (filePath.isEmpty())
    {
        return;
    }

    isSavingData = true;
    QXlsx::Document doc;

    for (int i = 0; i < tabs->count(); ++i)
    {
        QWidget* wgt = tabs->widget(i); if
        (qobject_cast<LEDTab*>(wgt))
        {
                saveLEDTab(qobject_cast<LEDTab*>(wgt), tabs->tabText(i), doc);
        }
        else if (qobject_cast<PhotoresistorTab*>(wgt))
        {
            savePhotoresistorTab(qobject_cast<PhotoresistorTab*>(wgt),                    tabs-
>tabText(i), doc);
        }
        else if (qobject_cast<TemperatureTab*>(wgt))
        {
            saveTemperatureTab(qobject_cast<TemperatureTab*>(wgt),   tabs->tabText(i),
doc);
        }
        else if (qobject_cast<UltrasonicTab*>(wgt))
        {
            saveUltrasonicTab(qobject_cast<UltrasonicTab*>(wgt),           tabs->tabText(i),
 doc);
        }
    }

    doc.saveAs(filePath);
    QMessageBox::information(nullptr, "Information", "The data is saved"); isSavingData =
    false;
}
```

```
void MainWindow::updateStatus(bool isConnected)
{
    if (isConnected)
    {
        if (comPort->open(QIODevice::ReadWrite))
        {
            qDebug() << "OPENED";
        }
        else
        {
            QMessageBox::critical(nullptr, "Error", comPort->errorString());
        }
        comPortInfo = QSerialPortInfo(*comPort);
        status->setText("Connected:  "  +  comPortInfo.description()  +  "  "  +
comPortInfo.portName());
    }
    else
    {
        if (comPort->isOpen())
        {
            comPort->close(); qDebug()
            << "CLOSED";
        }
        status->setText("Disconnected");
    }

    if (!buffer.isEmpty())
    {
        buffer.clear();
    }
}

void MainWindow::updatePortName(QString portName)
{
    comPort->setPortName(portName);
    qDebug() << "Port name: " << comPort->portName();
}
```

```
void MainWindow::updateBaudRate(QString baudRate)
{
    comPort->setBaudRate(baudRate.toInt());
    qDebug() << "Baud rate: " << comPort->baudRate();
}

void MainWindow::updateDataBits(QString dataBits)
{
    QSerialPort::DataBits data = QSerialPort::Data8; if
    (dataBits == "5")
    {
        data = QSerialPort::Data5;
    }
    else if (dataBits == "6")
    {
        data = QSerialPort::Data6;
    }
    else if (dataBits == "7")
    {
        data = QSerialPort::Data7;
    }
    else if (dataBits == "8")
    {
        data = QSerialPort::Data8;
    }

    comPort->setDataBits(data);
    qDebug() << "Data bits: " << comPort->dataBits();
}

void MainWindow::updateStopBits(QString stopBits)
{
    QSerialPort::StopBits stop = QSerialPort::OneStop; if
    (stopBits == "1 Bit")
    {
        stop = QSerialPort::OneStop;
    }
    else if (stopBits == "1.5 Bits")
```

```
    {
        stop = QSerialPort::OneAndHalfStop;
    }
    else if (stopBits == "2 Bits")
    {
        stop = QSerialPort::TwoStop;
    }

    comPort->setStopBits(stop);
    qDebug() << "Stop bits: " << comPort->stopBits();
}

void MainWindow::updateParity(QString parity)
{
    QSerialPort::Parity par = QSerialPort::NoParity; if
    (parity == "Even Parity")
    {
        par = QSerialPort::EvenParity;
    }
    else if (parity == "Odd Parity")
    {
        par = QSerialPort::OddParity;
    }
    else if (parity == "Space Parity")
    {
        par = QSerialPort::SpaceParity;
    }
    else if (parity == "Mark Parity")
    {
        par = QSerialPort::MarkParity;
    }
    else if (parity == "No Parity")
    {
        par = QSerialPort::NoParity;
    }

    comPort->setParity(par);
    qDebug() << "Parity: " << comPort->parity();
}
```

```
void MainWindow::updateFlowControl(QString flowControl)
{
    QSerialPort::FlowControl flow = QSerialPort::NoFlowControl; if
    (flowControl == "No Flow Control")
    {
        flow = QSerialPort::NoFlowControl;
    }
    else if (flowControl == "Hardware Control")
    {
        flow = QSerialPort::HardwareControl;
    }
    else if (flowControl == "Software Control")
    {
        flow = QSerialPort::SoftwareControl;
    }

    comPort->setFlowControl(flow);
    qDebug() << "Flow control: " << comPort->flowControl();
}

void MainWindow::updateTimeout(QString timeout)
{
    if (timeout == "1 sec")
    {
        timer->start(1000);
    }
    else if (timeout == "3 sec")
    {
        timer->start(3000);
    }
    else if (timeout == "5 sec")
    {
        timer->start(5000);
    }
    else if (timeout == "10 sec")
    {
        timer->start(10000);
```

```
    }
    else if (timeout == "30 sec")
    {
        timer->start(30000);
    }
    else if (timeout == "1 min")
    {
        timer->start(60000);
    }
    else if (timeout == "3 min")
    {
        timer->start(180000);
    }
    else if (timeout == "5 min")
    {
        timer->start(300000);
    }
    else if (timeout == "10 min")
    {
        timer->start(600000);
    }
    else if (timeout == "30 min")
    {
        timer->start(1800000);
    }
    else if (timeout == "1 hour")
    {
        timer->start(3600000);
    }
}

void MainWindow::readSerial()
{
    buffer.append(comPort->readAll());

    if (!isSavingData)
    {
        while (buffer.contains('{') && buffer.contains('}'))
        {
```

```
qDebug() << buffer;
int startIndex = buffer.indexOf('{'); int
endIndex = buffer.indexOf('}');
QString parameters = buffer.mid(startIndex + 1, endIndex - startIndex -
1);
qDebug() << parameters;
if (parameters.contains('}'))
{
```

```
                qDebug() << "DELETED";
                buffer.erase(buffer.begin(), buffer.begin() + endIndex + 1);
                continue;
            }
            if (parameters.contains('{'))
            {
                qDebug() << "DELETED";
                buffer.erase(buffer.begin(),  buffer.begin()  +  buffer.indexOf('{',
buffer.indexOf('{') + 1));
                continue;
            }
            if (endIndex < startIndex)
            {
                qDebug() << "DELETED";
                buffer.erase(buffer.begin(), buffer.begin() + endIndex + 1);
                continue;
            }
            QStringList list = parameters.split(','); bool
            isFound = false;
            if (list[0] == "TEMP_AND_HUM")
            {
                for (int i = 0; i < tabs->count(); ++i)
                {
                    if (tabs->tabText(i) == list[1])
                    {
                        qobject_cast<TemperatureTab*>(tabs->widget(i))-
>setTemperature(list[2]);
                        qobject_cast<TemperatureTab*>(tabs->widget(i))-
                        d = true; break;
>setHumidity(list[3]);
                    }
qDebug() << "UPDATED";
i
s
F
o
u
n
```

```
        }

        if (!isFound)
        {
            TemperatureTab* temperature = new TemperatureTab;
            temperature->setTemperature(list[2]);  temperature-
            >setHumidity(list[3]);
            tabs->addTab(temperature, list[1]);
            qDebug() << "CREATED";
            connect(timer, SIGNAL(timeout()), temperature, SLOT(saveData()));
        }
    }
    else if (list[0] == "LED")
    {
        for (int i = 0; i < tabs->count(); ++i)
        {
            if (tabs->tabText(i) == list[1])
            {
                qobject_cast<LEDTab*>(tabs->widget(i))->setLEDValue(list[2]); qDebug()
                << "UPDATED";

                isFound = true;
                break;
            }
        }

        if (!isFound)
        {
            LEDTab* led = new LEDTab;
            led->setLEDValue(list[2]);
            tabs->addTab(led, list[1]);
            qDebug() << "CREATED";
            connect(timer, SIGNAL(timeout()), led, SLOT(saveData()));
        }
    }
    else if (list[0] == "PHOTORESISTOR")
    {
        for (int i = 0; i < tabs->count(); ++i)
```

```
                {
                    if (tabs->tabText(i) == list[1])
                    {
                        qobject_cast<PhotoresistorTab*>(tabs->widget(i))-
>setPhotoresistorValue(list[2]);
                        qDebug() << "UPDATED";
                        isFound = true;
                        break;
                    }
                }

                if (!isFound)
                {
                    PhotoresistorTab* photoresistor = new PhotoresistorTab;
                    photoresistor->setPhotoresistorValue(list[2]);
                    tabs->addTab(photoresistor, list[1]); qDebug()
                    << "CREATED";
                    connect(timer,          SIGNAL(timeout()),          photoresistor,
SLOT(saveData()));
                }
```

```
            }
            else if (list[0] == "HC_SR04")
            {
                for (int i = 0; i < tabs->count(); ++i)
                {
                    if (tabs->tabText(i) == list[1])
                    {

>setValue(list[2]);

                    }
                }
qobject_cast<Ultrasonic
Tab*>(tabs->widget(i))-


qDebug() << "UPDATED";
i
s
F
o
u
n
d

=

t
r
u
e
;

b
r
e
a
k
;
```

```
                if (!isFound)
                {
                    UltrasonicTab* ultrasonic = new UltrasonicTab;
                    ultrasonic->setValue(list[2]);
                    tabs->addTab(ultrasonic, list[1]);
                    qDebug() << "CREATED";
                    connect(timer, SIGNAL(timeout()), ultrasonic, SLOT(saveData()));
                }
            }
               buffer.erase(buffer.begin(), buffer.begin() + endIndex + 1);
        }
    }
}
```

ConnectionChecker.h

```
#pragma once

#include <QWidget>
#include <QTimer>
#include <QThread>

class ConnectionChecker : public QObject
{
    Q_OBJECT
public:
    explicit ConnectionChecker(QObject *parent = nullptr);
    ~ConnectionChecker();

private:
    bool isConnected;
    QTimer* timer;
    QThread* thread;

private slots:
    void checkConnection();

signals:
```

```
    void connectionChanged(bool isConnected);
};
```

ConnectionChecker.cpp

```
#include "ConnectionChecker.h"
#include <QSerialPort> #include
<QSerialPortInfo>

ConnectionChecker::ConnectionChecker(QObject *parent)
    : QObject{parent}
    , isConnected{false}
{
    timer = new QTimer(this);
    connect(timer, SIGNAL(timeout()), this, SLOT(checkConnection()));
    timer->start(300);

    QThread* thread = new QThread(this);
    this->moveToThread(thread);
    thread->start();
}

ConnectionChecker::~ConnectionChecker()
{
    thread->quit();
}

void ConnectionChecker::checkConnection()
{
    QList<QSerialPortInfo> portsInfo = QSerialPortInfo::availablePorts(); bool
    currentConnected = isConnected;

    if (!portsInfo.isEmpty())
    {
        isConnected = true;
        //qDebug() << isConnected << " " << portsInfo.at(0).portName();
    }
    else
    {
```

```
            isConnected = false;
            //qDebug() << isConnected;
        }

        if (currentConnected != isConnected)
        {
            qDebug() << "Emitted";
            emit connectionChanged(isConnected);
        }
}
```

Settings.h

```
#pragma once

#include <QWidget>
#include <QComboBox>
#include <QLabel>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QGroupBox>

class Setting : public QWidget
{
    Q_OBJECT
public:
    Setting(QWidget *parent = nullptr);

private:
    QGroupBox* portSettings;
    QGroupBox* otherSettings;

    QLabel* portLabel;
    QLabel* baudRateLabel;
    QLabel* dataBitsLabel;
    QLabel* stopBitsLabel;
    QLabel* parityLabel;
    QLabel* flowControlLabel;
```

```
    QLabel* saveDataTimeoutLabel;

    QComboBox* portComboBox; QComboBox*
    baudRateComboBox; QComboBox*
    dataBitsComboBox; QComboBox*
    stopBitsComboBox; QComboBox*
    parityComboBox; QComboBox*
    flowControlComboBox;
    QComboBox* saveDataTimeoutComboBox;

    void fillComboBoxes();

public slots:
    void   changePortList(bool   isConnected);
    void portChanged(QString port);
    void   baudRateChanged(QString   baudRate);
    void   dataBitsChanged(QString   dataBits);
    void   stopBitsChanged(QString   stopBits);
    void parityChanged(QString parity);
    void   flowControlChanged(QString   flowControl);
    void saveDataTimeoutChanged(QString timeout);

signals:
    void portChangedSignal(QString port);
    void   baudRateChangedSignal(QString   baudRate);
    void   dataBitsChangedSignal(QString   dataBits);
    void   stopBitsChangedSignal(QString   stopBits);
    void parityChangedSignal(QString parity);
    void flowControlChangedSignal(QString flowControl); void
    saveDataTimeoutChangedSignal(QString timeout);
};
```

Settings.cpp

```
#include "Setting.h"

#include <QStringList> #include
<QSerialPortInfo>
```

```
Setting::Setting(QWidget *parent)
    : QWidget(parent)
{
    portSettings = new QGroupBox("Port Settings");
    otherSettings = new QGroupBox("Other Settings");

    portLabel = new QLabel("Port"); baudRateLabel
    = new QLabel("Baud Rate"); dataBitsLabel = new
    QLabel("Data Bits"); stopBitsLabel = new
    QLabel("Stop Bits"); parityLabel = new
    QLabel("Parity");
    flowControlLabel = new QLabel("Flow Control");
    saveDataTimeoutLabel = new QLabel("Saving Data Timeout");

    portComboBox = new QComboBox(); baudRateComboBox
    = new QComboBox(); dataBitsComboBox = new
    QComboBox(); stopBitsComboBox = new QComboBox();
    parityComboBox = new QComboBox();
    flowControlComboBox = new QComboBox();
    saveDataTimeoutComboBox = new QComboBox();
    fillComboBoxes();

    connect(portComboBox,              SIGNAL(currentTextChanged(QString)),              this,
SLOT(portChanged(QString)));
    connect(baudRateComboBox,            SIGNAL(currentTextChanged(QString)),            this,
SLOT(baudRateChanged(QString)));
    connect(dataBitsComboBox,            SIGNAL(currentTextChanged(QString)),            this,
SLOT(dataBitsChanged(QString)));
    connect(stopBitsComboBox,            SIGNAL(currentTextChanged(QString)),            this,
SLOT(stopBitsChanged(QString)));
    connect(parityComboBox,             SIGNAL(currentTextChanged(QString)),             this,
SLOT(parityChanged(QString)));
    connect(flowControlComboBox,          SIGNAL(currentTextChanged(QString)),          this,
SLOT(flowControlChanged(QString)));
    connect(saveDataTimeoutComboBox,        SIGNAL(currentTextChanged(QString)),        this,
SLOT(saveDataTimeoutChanged(QString)));

    QVBoxLayout* firstColumn = new QVBoxLayout;
    firstColumn->addWidget(portLabel); firstColumn-
```

```
>addWidget(baudRateLabel); firstColumn-
>addWidget(dataBitsLabel); firstColumn-
>addWidget(stopBitsLabel); firstColumn-
>addWidget(parityLabel); firstColumn-
>addWidget(flowControlLabel);

QVBoxLayout* secondColumn = new QVBoxLayout;
secondColumn->addWidget(portComboBox);
secondColumn->addWidget(baudRateComboBox);
secondColumn->addWidget(dataBitsComboBox);
secondColumn->addWidget(stopBitsComboBox);
secondColumn->addWidget(parityComboBox);
secondColumn->addWidget(flowControlComboBox);

QHBoxLayout* layoutForPortSettings = new QHBoxLayout;
layoutForPortSettings->addLayout(firstColumn);
layoutForPortSettings->addLayout(secondColumn);

QVBoxLayout* firstColumnForOtherSettings = new QVBoxLayout;
firstColumnForOtherSettings->addWidget(saveDataTimeoutLabel);

QVBoxLayout* secondColumnForOtherSettings = new QVBoxLayout;
secondColumnForOtherSettings->addWidget(saveDataTimeoutComboBox);

QHBoxLayout* layoutForOtherSettings = new QHBoxLayout;
layoutForOtherSettings->addLayout(firstColumnForOtherSettings);
layoutForOtherSettings->addLayout(secondColumnForOtherSettings);

portSettings->setLayout(layoutForPortSettings);
otherSettings->setLayout(layoutForOtherSettings);

QHBoxLayout* layout = new QHBoxLayout;
layout->addWidget(portSettings); layout-
>addWidget(otherSettings);

setLayout(layout);
setWindowTitle("Settings");
```

```
    resize(240, 180);
}

void Setting::fillComboBoxes()
{
    portComboBox->setDuplicatesEnabled(false);
    baudRateComboBox->setDuplicatesEnabled(false);
    QStringList baudRateList;
    foreach (qint32 rate, QSerialPortInfo::standardBaudRates())
    {
        baudRateList.append(QString::number(rate));
    }
    baudRateComboBox->addItems(baudRateList);
    baudRateComboBox->setCurrentIndex(6);
    dataBitsComboBox->setDuplicatesEnabled(false);
    dataBitsComboBox->addItems(QStringList() << "5" << "6" << "7" << "8");
    dataBitsComboBox->setCurrentIndex(3);
    stopBitsComboBox->setDuplicatesEnabled(false);
    stopBitsComboBox->addItems(QStringList() << "1 Bit" << "1.5 Bits" << "2 Bits");
    parityComboBox->setDuplicatesEnabled(false);
    parityComboBox->addItems(QStringList() << "No Parity" << "Even Parity" << "Odd Parity"
                                           << "Space Parity" << "Mark Parity");
    flowControlComboBox->setDuplicatesEnabled(false);
    flowControlComboBox->addItems(QStringList() << "No Flow Control" << "Hardware Control" <<
"Software Control");
    saveDataTimeoutComboBox->setDuplicatesEnabled(false);
    saveDataTimeoutComboBox->addItems(QStringList() << "1 sec" << "3 sec" << "5 sec"
<< "10 sec" << "30 sec" << "1 min"
                                                    << "3 min" << "5 min" << "10 min"
<< "30 min" << "1 hour");
}

void Setting::changePortList(bool isConnected)
{
    portComboBox->clear();
    QStringList list;
    if (isConnected)
    {
```

```
        foreach (QSerialPortInfo port, QSerialPortInfo::availablePorts())
        {
            list.append(port.portName());
        }
    }
    portComboBox->addItems(list);
}

void Setting::portChanged(QString port)
{
    qDebug() << "portChangedSignal" << port; emit
    portChangedSignal(port);
}

void Setting::baudRateChanged(QString baudRate)
{
    qDebug() << "baudRateChangedSignal" << baudRate; emit
    baudRateChangedSignal(baudRate);
}

void Setting::dataBitsChanged(QString dataBits)
{
    qDebug() << "dataBitsChangedSignal" << dataBits; emit
    dataBitsChangedSignal(dataBits);
}

void Setting::stopBitsChanged(QString stopBits)
{
    qDebug() << "stopBitsChangedSignal" << stopBits; emit
    stopBitsChangedSignal(stopBits);
}

void Setting::parityChanged(QString parity)
{
    qDebug() << "parityChangedSignal" << parity;
    emit parityChangedSignal(parity);
}
```

```
void Setting::flowControlChanged(QString flowControl)
{
    qDebug() << "flowControlChangedSignal" << flowControl;

    emit flowControlChangedSignal(flowControl);
}

void Setting::saveDataTimeoutChanged(QString timeout)
{
    qDebug() << "saveDataTimeoutChangedSignal" << timeout; emit
    saveDataTimeoutChangedSignal(timeout);
}
```

LEDTab.h

```
#pragma once

#include <QWidget>
#include <QLabel>
#include <QLineEdit>
#include <QGridLayout>
#include <QList> #include
<QString>

class LEDTab : public QWidget
{
    Q_OBJECT
public:
    LEDTab(QWidget *parent = nullptr);

    void setLEDValue(const QString& value);

    const QList<int> getValues() const; const
    QList<QString> getDateTime() const;

private:
    QLabel* LEDLabel;
    QLineEdit* LEDLine;
```

```
    QGridLayout* layout;

    QList<int> values;
    QList<QString> dates;

public slots:

    void saveData();
};
```

LEDTab.cpp

```
#include "LEDTab.h"
#include <QDateTime>

LEDTab::LEDTab(QWidget *parent)
    : QWidget(parent)
    , values()
    , dates()
{
    LEDLabel = new QLabel("Photoresistor value"); LEDLine =
    new QLineEdit;

    LEDLine->setReadOnly(true);

    layout = new QGridLayout(this);
    layout->setHorizontalSpacing(10);
    layout->addWidget(LEDLabel, 0, 0);
    layout->addWidget(LEDLine, 0, 1);

    setLayout(layout);
}

void LEDTab::setLEDValue(const QString& value)
{
    LEDLine->setText(value);
}

const QList<int> LEDTab::getValues() const
{
```

```
    return values;
}

const QList<QString> LEDTab::getDateTime() const
{
    return dates;
}
void LEDTab::saveData()
{
    values.append(LEDLine->text().toInt());
    dates.append(QDateTime::currentDateTime().toString(Qt::ISODate));
}
```

PhotoresistorTab.h

```
#pragma once

#include <QWidget>
#include <QLabel>
#include <QLineEdit>
#include <QGridLayout>
#include <QList> #include
<QString>

class PhotoresistorTab : public QWidget
{
    Q_OBJECT
public:
    PhotoresistorTab(QWidget *parent = nullptr);

    void setPhotoresistorValue(const QString& value);

    const QList<int> getValues() const; const
    QList<QString> getDateTime() const;

private:
    QLabel* photoresistorLabel;
    QLineEdit* photoresistorLine;
```

```
    QGridLayout* layout;

    QList<int> values;
    QList<QString> dates;

public slots:
    void saveData();
};
```

PhotoresistorTab.cpp

```
#include "PhotoresistorTab.h"
#include <QDateTime>

PhotoresistorTab::PhotoresistorTab(QWidget *parent)
    : QWidget{parent}
{
    photoresistorLabel = new QLabel("Photoresistor value");
    photoresistorLine = new QLineEdit;

    photoresistorLine->setReadOnly(true);

    layout = new QGridLayout(this);
    layout->setHorizontalSpacing(10);
    layout->addWidget(photoresistorLabel, 0, 0);
    layout->addWidget(photoresistorLine, 0, 1);

    setLayout(layout);
}

void PhotoresistorTab::setPhotoresistorValue(const QString& value)
{
    photoresistorLine->setText(value);
}

const QList<int> PhotoresistorTab::getValues() const
{
    return values;
}
```

```
const QList<QString> PhotoresistorTab::getDateTime() const
{
    return dates;
}

void PhotoresistorTab::saveData()
{
    values.append(photoresistorLine->text().toInt());
    dates.append(QDateTime::currentDateTime().toString(Qt::ISODate));
}
```

TemperatureTab.h

```
#pragma once

#include <QWidget>
#include <QLabel>
#include <QLineEdit>
#include <QGridLayout>
#include <QList> #include
<QString>

class TemperatureTab : public QWidget
{
    Q_OBJECT
public:
    TemperatureTab(QWidget *parent = nullptr);

    void setTemperature(const QString& temperature); void
    setHumidity(const QString& humidity);

    const QList<double> getTemperature() const;
    const QList<double> getHumidity() const; const
    QList<QString> getDateTime() const;

private:
    QLabel* temperatureLabel; QLabel*
    humidityLabel; QLineEdit*
```

```
    temperatureLine; QLineEdit*
    humidityLine; QGridLayout* layout;

    QList<double> temperature;
    QList<double> humidity;
    QList<QString> dates;

public slots:
    void saveData();
};
```

TemperatureTab.cpp

```
#include "TemperatureTab.h"
#include <QDateTime>

TemperatureTab::TemperatureTab(QWidget *parent)
    : QWidget{parent}
{
    temperatureLabel = new QLabel("Temperature, ° C");
    humidityLabel = new QLabel("Humidity, %");
    temperatureLine = new QLineEdit;
    humidityLine = new QLineEdit;

    temperatureLine->setReadOnly(true);
    humidityLine->setReadOnly(true);

    layout = new QGridLayout(this);
    layout->setHorizontalSpacing(10);
    layout->addWidget(temperatureLabel, 0, 0);
    layout->addWidget(temperatureLine, 0, 1);
    layout->addWidget(humidityLabel, 1, 0);
    layout->addWidget(humidityLine, 1, 1);

    setLayout(layout);
}

void TemperatureTab::setTemperature(const QString& temperature)
{
```

```
    temperatureLine->setText(temperature);
}

void TemperatureTab::setHumidity(const QString& humidity)
{
    humidityLine->setText(humidity);
}

const QList<double> TemperatureTab::getTemperature() const
{

    return temperature;
}

const QList<double> TemperatureTab::getHumidity() const
{
    return humidity;
}

const QList<QString> TemperatureTab::getDateTime() const
{
    return dates;
}

void TemperatureTab::saveData()
{
    temperature.append(temperatureLine->text().toDouble());
    humidity.append(humidityLine->text().toDouble());
    dates.append(QDateTime::currentDateTime().toString(Qt::ISODate));
}
```

UltrasonicTab.h

```
#pragma once

#include <QWidget>
#include <QLabel>
#include <QLineEdit>
```

```
#include <QGridLayout>
#include <QList> #include
<QString>

class UltrasonicTab : public QWidget
{
    Q_OBJECT
public:
    UltrasonicTab(QWidget *parent = nullptr);

    void setValue(const QString& value);

    const QList<int> getValues() const;

    const QList<QString> getDateTime() const;

private:
    QLabel* ultrasonicLabel; QLineEdit*
    ultrasonicLine; QGridLayout*
    layout;

    QList<int> values;
    QList<QString> dates;

public slots:
    void saveData();
};
```

UltrasonicTab.cpp

```
#include "UltrasonicTab.h"
#include <QDateTime>

UltrasonicTab::UltrasonicTab(QWidget *parent)
    : QWidget(parent)
    , values()
    , dates()
{
    ultrasonicLabel = new QLabel("Distance, cm");
```

```
    ultrasonicLine = new QLineEdit;

    ultrasonicLine->setReadOnly(true);

    layout = new QGridLayout(this);
    layout->setHorizontalSpacing(10);
    layout->addWidget(ultrasonicLabel, 0, 0);
    layout->addWidget(ultrasonicLine, 0, 1);

    setLayout(layout);
}

void UltrasonicTab::setValue(const QString &value)
{
    ultrasonicLine->setText(value);
}

const QList<int> UltrasonicTab::getValues() const
{
    return values;
}

const QList<QString> UltrasonicTab::getDateTime() const
{
    return dates;
}

void UltrasonicTab::saveData()
{
    values.append(ultrasonicLine->text().toInt());
    dates.append(QDateTime::currentDateTime().toString(Qt::ISODate));
}
```

FirstProgram.ino

```
#include "Logger.h"

void Logger::write(SensorType type, String title, ...)
```

```
{
  va_list list;
  va_start(list, title);

  switch (type)
  {
    case SensorType::DHT:
    {
      Serial.print("{TEMP_AND_HUM,");
      Serial.print(title);
      Serial.print(",");
      double temp = va_arg(list, double);
      double hum = va_arg(list, double);
      Serial.print(temp); Serial.print(",");
      Serial.print(hum);
      Serial.println("}");

      break;
    }
    case SensorType::LED:
    {
      Serial.print("{LED,");
      Serial.print(title);
      Serial.print(",");
      Serial.print(va_arg(list, int));
      Serial.println("}");
      break;
    }
    case SensorType::PHOTORESISTOR:
    {
      Serial.print("{PHOTORESISTOR,");
      Serial.print(title);
      Serial.print(",");
      Serial.print(va_arg(list, int));
      Serial.println("}");
      break;
    }
    case SensorType::HC_SR04:
```

```
    {
      Serial.print("{HC_SR04,");
      Serial.print(title);
      Serial.print(",");
      Serial.print(va_arg(list, int));
      Serial.println("}");
      break;
    }
    default:
    {
      Serial.println("UNKNOWN SENSOR TYPE");
    }
  }

  va_end(list);
}
```

SecondProgram.ino

```
#include <Ultrasonic.h> #include
"Logger.h"

const int TRIG_PIN = 14; const
int ECHO_PIN = 27;
Ultrasonic sonic(TRIG_PIN, ECHO_PIN);

void setup() {
  Serial.begin(115200);
}

void loop() {
  int distance = sonic.read(); Logger::write(SensorType::HC_SR04,
  "HC-SR04", distance); delay(1000);
}
```

Logger.h

```
#pragma once

#include "Arduino.h"
```

```
enum class SensorType
{
  DHT,
  LED,
  PHOTORESISTOR,
  HC_SR04
};

class Logger
{
public:
  static void write(SensorType type, String title, ...);

private:
  Logger() = default;
};
```

Logger.cpp

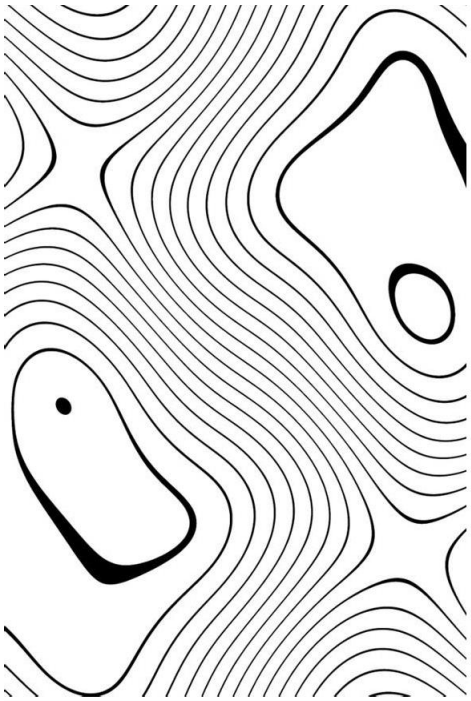
```
#include "Logger.h"

void Logger::write(SensorType type, String title, ...)
{
  va_list list;
  va_start(list, title);

  switch (type)
  {
    case SensorType::DHT:
    {
      Serial.print("{TEMP_AND_HUM,");
      Serial.print(title);
      Serial.print(",");
      double temp = va_arg(list, double);
      double hum = va_arg(list, double);
      Serial.print(temp); Serial.print(",");
      Serial.print(hum);
      Serial.println("}");
```

```
      break;
    }
    case SensorType::LED:
    {
      Serial.print("{LED,");
      Serial.print(title);
      Serial.print(",");
      Serial.print(va_arg(list, int));
      Serial.println("}");
      break;
    }
    case SensorType::PHOTORESISTOR:
    {
      Serial.print("{PHOTORESISTOR,");
      Serial.print(title);
      Serial.print(",");
      Serial.print(va_arg(list, int));

      Serial.println("}");
      break;
    }
    case SensorType::HC_SR04:
    {
      Serial.print("{HC_SR04,");
      Serial.print(title);
      Serial.print(",");
      Serial.print(va_arg(list, int));
      Serial.println("}");
      break;
    }
    default:
    {
      Serial.println("UNKNOWN SENSOR TYPE");
    }
  }

  va_end(list);
}
```

ДОДАТОК Б

(Рекомендований)

КОПІЇ ПРЕЗЕНТАЦІЇ



Розробка програми для зчитування та візуалізації даних з мікроконтролерів через COM-порт

Підготував ст. гр. ITIP-20-1
Гончар Б. В.

Науковий керівник:
ст. вик. каф. РТІКС
Ганшин Д. Г.

Цілі, актуальність і завдання дослідження

Актуальність

Дедалі більше використання мікроконтролерів у різних галузях робить нашу роботу дуже актуальною.

Цілі

Цілі наших досліджень включають розробку програмного забезпечення для читання даних через COM-порт, візуалізації та зручного зберігання даних.

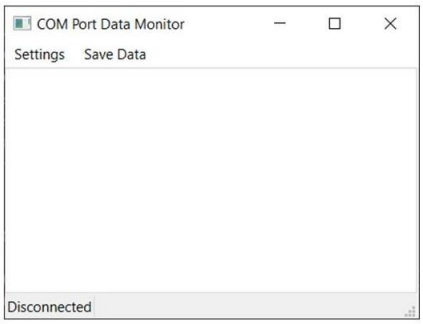
Завдання

Створити інтуїтивно зрозумілий інтерфейс для зчитування, візуалізації та зберігання даних, що дозволить ефективно керувати та аналізувати інформацію з мікроконтролерів.

Вибір фреймворку для розробки інтерфейсу

JavaFx .NET Framework Qt

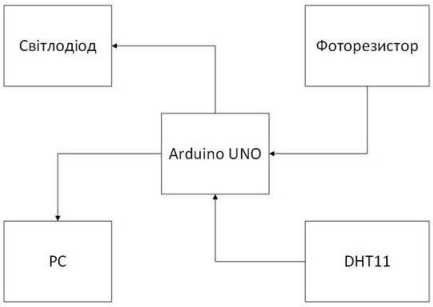
Опис розробленої програми

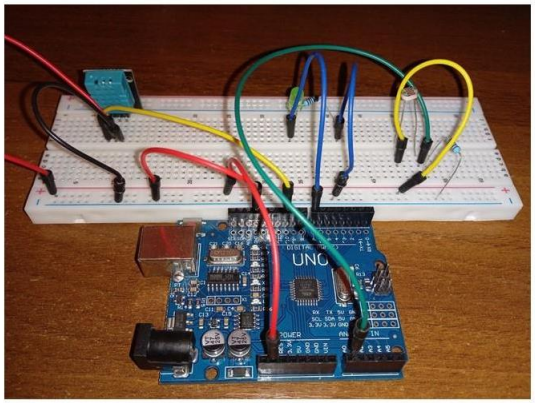


1 **Інтерфейс користувача**

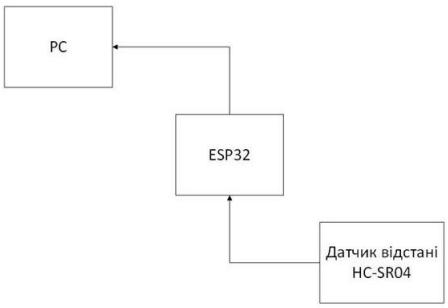
Ми створили зручний графічний інтерфейс для розміщення різних датчиків без зміни загальної структури програми.

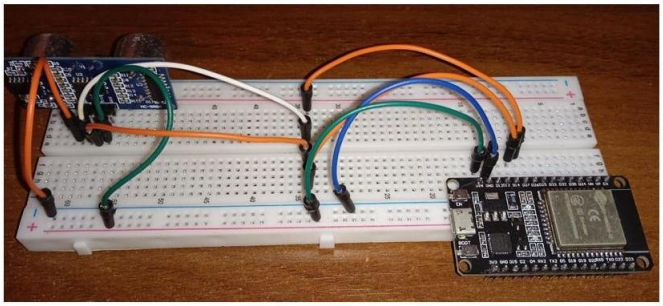
Макет плати та схема для Arduino UNO



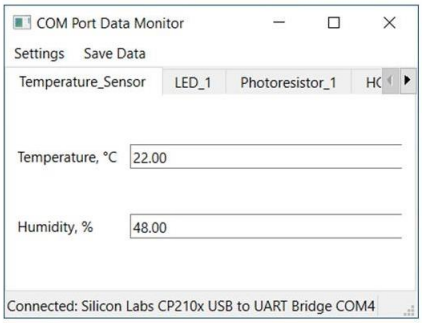


Макет плати та схема для ESP32





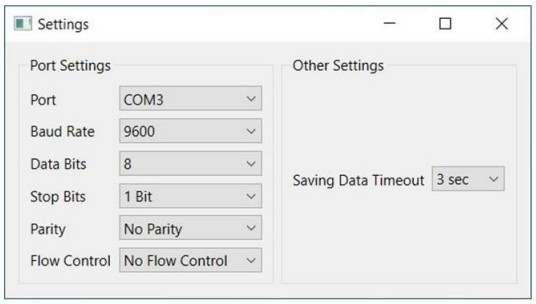
Опис розробленої програми



2 **Перегляд даних**

Інтерфейс з вкладками для зручного перегляду даних з різних датчиків.

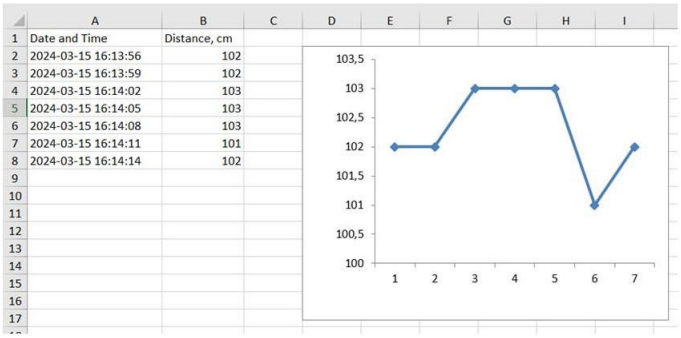
Опис розробленої програми



3 **Налаштування**

Сторінка налаштувань дозволяє користувачам налаштувати параметри читання даних з мікроконтролера.

Зберігання та аналіз даних



	A	B
1	Date and Time	Distance, cm
2	2024-03-15 16:13:56	102
3	2024-03-15 16:13:59	102
4	2024-03-15 16:14:02	103
5	2024-03-15 16:14:05	103
6	2024-03-15 16:14:08	103
7	2024-03-15 16:14:11	101
8	2024-03-15 16:14:14	102

Інтеграція з Excel

Наша програма повністю інтегрується з Excel, дозволяючи користувачам легко зберігати та аналізувати дані.

Аналіз даних

Зручний аналіз даних із кількома параметрами візуалізації для поглибленого аналізу.

Висновки

Ефективне управління даними

Наша програма забезпечує ефективне керування даними та візуалізацію, задовольняючи різні потреби користувачів.

Покращена взаємодія з користувачем

Покращена взаємодія з користувачем завдяки інтуїтивно зрозумілим функціям для легкого читання та аналізу даних.

Майбутній розвиток

Ми прагнемо ще більше розширити можливості програми та розширити її сумісність із різноманітними мікроконтролерами та датчиками.

ДОДАТОК В

(Обов’язковий)

ВІДОМІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ

№ п/п	Позначення	Найменування	Додаткові відомості
		Текстові документи	
		Пояснювальна записка	115 с.

Зм	Арк.	№ докум.	Підп.	Дата	ГЮІК.ХХХХХХ.000Д2		
Розроб.		Гончар			Розробка додатку для зчитування та візуалізації даних з МК через com порт. Відомість кваліфікаційної роботи.	Літера	Аркуш	Аркушів
Перевір.		Ганшин				У	1	1
Н. контр.		Бітченко				ХНУРЕ Кафедра РТІКС		
Затв.		Зарудний						