МІНІСТЕРСТВО ОСВІТИ І НАУКИ, УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

КАФЕДРА КОНСТРУЮВАННЯ ЕОА

ЗВІТ

про виконання проекту

по курсу «Алгоритмізація та програмування – 3»

Виконали:

студент гр. ДК-51

Махньов О. І.

Перевірив:

ас. Варфоломєєв А. Ю.

Київ – 2016

Постановка задачи

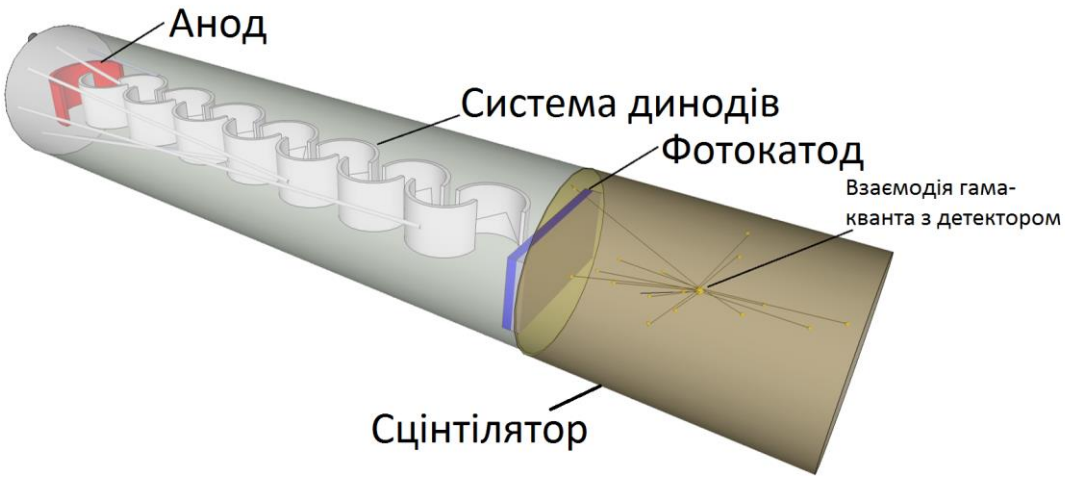
В практической радиоэкологии частой является задача измерения уровня радиоактивного излучения и определения типа радионуклида-загрязнителя. Для её решения используют различные приборы и приспособления, в частности, спектроскопы и спектрометры – устройства, способные определять энергию частицы, попавшей в детектор. Один из видов реализации такого устройства представляет из себя сборку из специального кристалла-сцинтиллятора и светового датчика. Сцинтиллятор разбивает гамма-квант на множество фотонов, которые регистрируются датчиком света, по сигналу от которого можно судить о энергии гамма-кванта. Обработка этого сигнала и его отображение – задача для программно-аппаратного комплекса (спектроскопа), который следует разработать в ходе данного проекта.

Аппаратная часть

Аппаратная часть устройства состоит из детектора, амплитудного анализатора, преобразователя тау-кода, микроконтроллера и персонального компьютера.

Детектор

Детектор гамма-излучения состоит из сцинтилляционного кристалла, и фотоэлектронного умножителя. Устройство детектора показано на иллюстрации:



Высоковольтный блок питания обеспечивает высокое отрицательное напряжение (~ -800В) для питания фотоэлектронного умножителя. Сигнал снимается с умножителя с помощью системы делителей.

Амплитудный анализатор

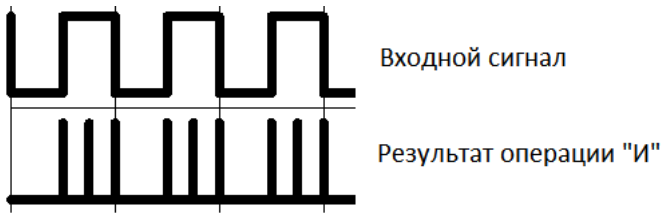
Амплитудный анализатор представляет из себя преобразователь аналогового сигнала в тау-код (последовательность TTL-импульсов, в которых передаваемые данные кодированы длиной) с помощью аппаратной реализации преобразования Вилкинсона. Выходной сигнал амплитудного анализатора несет в себе информацию о площади под кривой входного сигнала в виде логических импульсов различной длины, что с достаточной линейностью пропорционально энергии гамма-кванта, который этот сигнал вызвал.

Преобразователь тау-кода и микроконтроллер

Для получения информации с амплитудного анализатора, в форме, доступной для чтения простым микроконтроллером необходим специальный преобразователь. Задача несколько осложняется ненулевым временем реакции микроконтроллера и случайным характером входного сигнала.

Преобразователь тау-кода состоит из входного триггера Шмидта, схемы «входных ворот», генератора импульсов заполнения и высокочастотного счетчика.

Если схема готова к работе, то очередной входной импульс пропускается через входные ворота. Теперь, пока не будет подан сигнал «сброс», входные импульсы будут игнорироваться. С импульсом, который прошел через ворота и последовательностью импульсов с генератора выполняется логическая операция «И», в результате которой получается череда высокочастотных импульсов, количество которых пропорционально длине входного импульса. Эта череда импульсов подается на счетчик, подключенный к микроконтроллеру. Окончание импульса приводит к вызову прерывания, в ходе которого контроллер считывает состояние входных регистров, к которым подключен счетчик, и сохраняет полученное значение в памяти. Контроллер хранит накопленные данные в виде одномерного массива чисел, где номер ячейки отвечает за длину импульса, а значение в ячейке – за количество таких импульсов. После этого контроллер подает сигнал «сброс», «входные ворота» открываются, счетчик сбрасывается, и схема снова готова к получению импульсов. Также предусмотрена защита от переполнения счетчика: последний разряд выхода подключен к схеме выделения заднего фронта, которая искусственно закрывает входные ворота, что приводит к записыванию импульса недопустимой длины в один из начальных ячеек массива, которые можно отбрасывать.



Используется контроллер ATmega 2560 в составе платы Arduino 2560. Выходы счетчика подключены к выводам PINA(0…8) и PINC(0…3). Канал сброса подключен к выводу 32 платы. Код микропрограммы на Wiring приведен ниже:

```
#include <TimerOne.h>

byte spectre[4000];
bool transmit = false;

void setup() {
  pinMode(22, INPUT);
  pinMode(23, INPUT);
  pinMode(24, INPUT);
  pinMode(25, INPUT);
  pinMode(26, INPUT);
  pinMode(27, INPUT);
  pinMode(28, INPUT);
  pinMode(29, INPUT);
  pinMode(37, INPUT);
  pinMode(36, INPUT);
  pinMode(35, INPUT);
  pinMode(34, INPUT);
  pinMode(32, OUTPUT);
  Serial.begin(250000);
  attachInterrupt(0, imp, FALLING);
  Timer1.initialize();
  Timer1.attachInterrupt(sendData);
  digitalWrite(32, HIGH);
  digitalWrite(32, LOW);
}

void loop() {
  if (transmit) {
    for (int j = 0; j < 4000; j++) {
      if (spectre[j] != 0) {
        Serial.println(String(j) + '!' + String(spectre[j]));
        spectre[j] = 0;
      }
    }
    transmit = false;
  }
}

void sendData() {
  transmit = true;
}

void imp() {
    unsigned int sum = 0;
    for (byte i = 0; i < 8; i++) {
      sum += (bitRead(PINA, i) << i);
    }
    for (byte z = 0; z < 4; z++) {
      sum += (bitRead(PINC, z) << (z + 8));
    }
    digitalWrite(32, HIGH);
    digitalWrite(32, LOW);
    if (sum < 4000) {
      if (spectre[sum] < 256) spectre[sum]++;
    }
    wasEdge = false;
}
```

Программа 4 раза в секунду передает в порт RS-232 №0 ненулевые ячейки массива импульсов и обнуляет его, обрабатывает прерывания от преобразователя тау-кода и подает сигналы сброса. Передача данных ведется в формате:

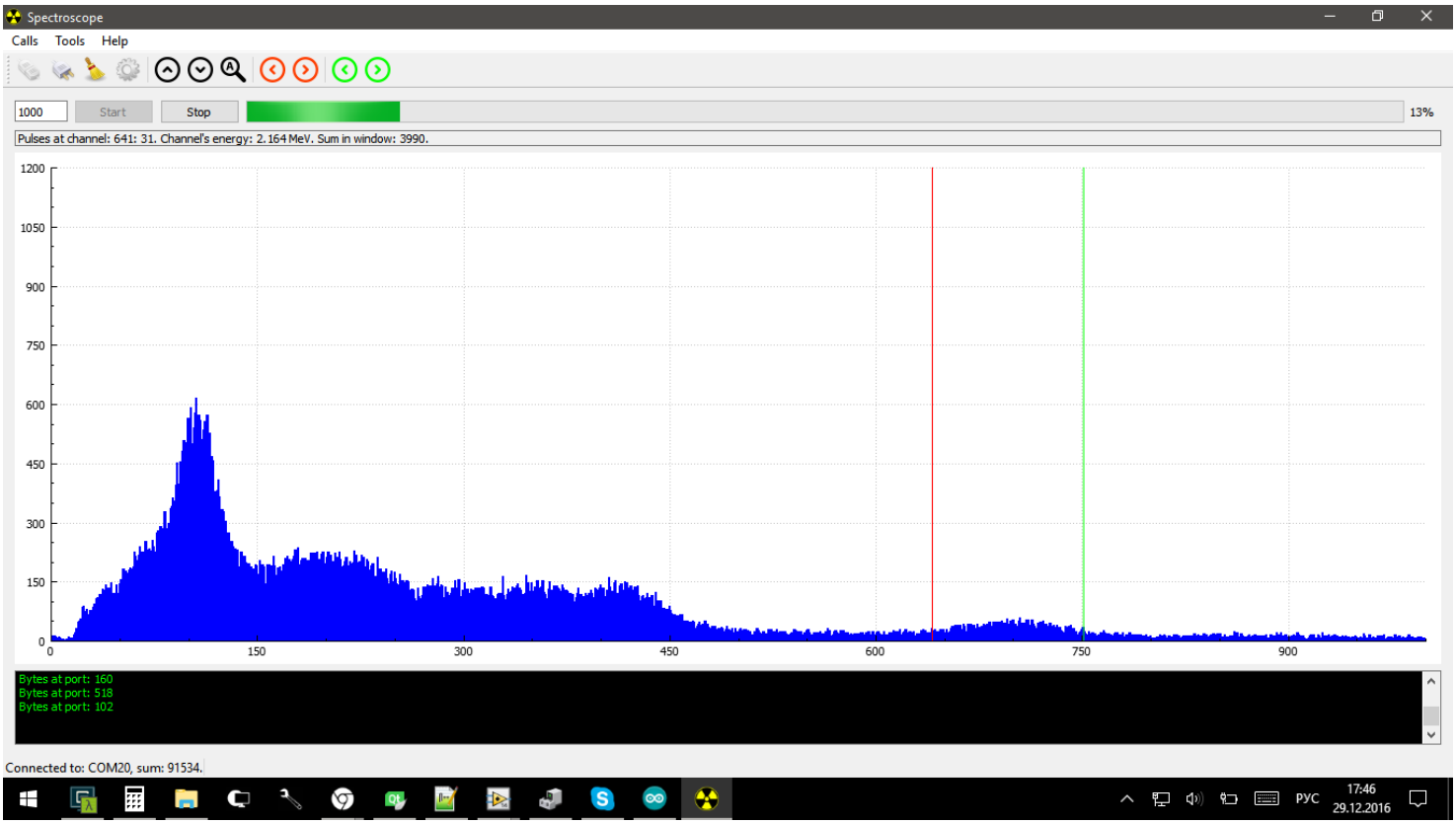
XYZ!IJK<CR><LF>

Где XYZ – номер канала, IJK – количество импульсов в этом канале с последнего обнуления. Таким образом, программа передает распределение импульсов по длине, или же распределение частиц по энергии.

В программе предусмотрена защита от переполнения массива. В ходе разработки аппаратной части и микропрограммы были проведены проверки на корректность регистрации случайного сигнала и линейности результатов.

Программная часть

Интерфейс



Интерфейс программы состоит из двух окон – рабочего окна и окна настроек. В рабочем окне находится график спектра излучения, панель управления подключением, накоплением, масштабом графика и движением курсоров, консоль с отладочными данными, строка информации и строка состояния.

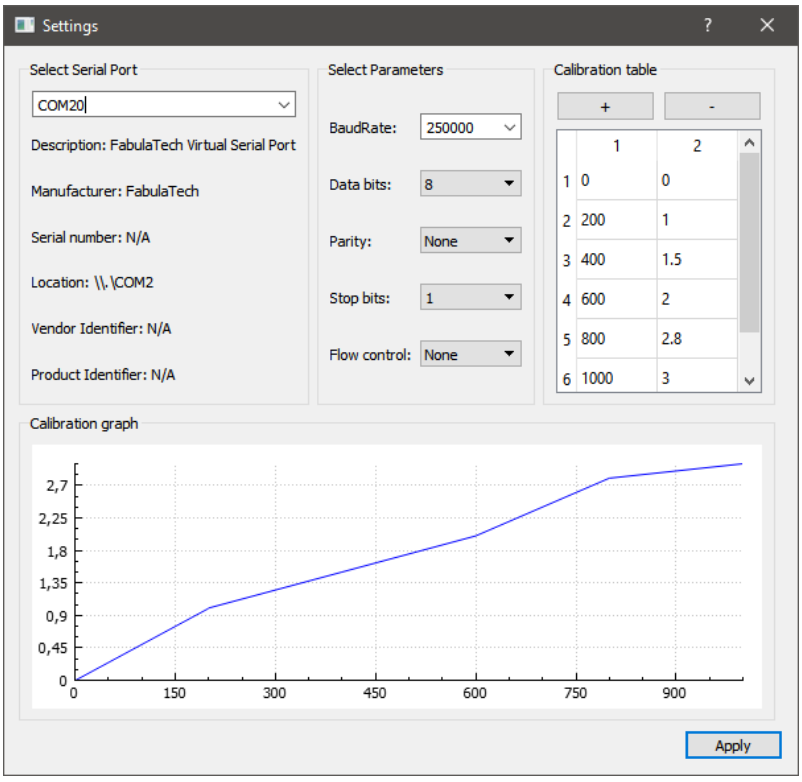
Панель управления подключением позволяет установить соединение с контроллером через виртуальные COM-порты, с параметрами, определенными текущими настройками.

На этой же панели размещены элементы управления вертикальным масштабом, предусмотрена функция автомасштабирования по максимальному значению графика.

Входящие данные отображаются на графике только во время работы в режиме накопления. Ниже расположены кнопки начала и окончания накопления данных, поле ввода времени накопления для его автоматического завершения и индикатор завершенности накопления (прогресс-бар).

Помимо гистограммы спектра, на графике имеются два курсора, перемещением которых можно управлять с помощью нажатий ЛКМ для одного курсора (красного) и ПКМ для другого (зеленого). Так же предусмотрены кнопки для перемещения обоих курсоров по одному каналу влево и вправо и возможность двигать красный курсор кнопками-стрелками на клавиатуре.

Ниже панелей управления расположена информационная строка, в которой выводится информация о канале, на котором в данный момент находится красный курсор: номер канала, количество импульсов в нем и его энергия. Так же в строке информации выводится информация о количестве импульсов в каналах на промежутке между красным и зеленым курсором.



В строке состояния выводится номер порта, к которому подключена программа и сумма накопленных импульсов.

В окне настроек, которое открывается по нажатию кнопки на панели рабочего окна, расположены меню для выбора COM порта и параметров подключения. Также в окне настроек расположена калибровочная таблица, с помощью которой пользователь может задать произвольное количество каналов с известной энергией. Энергии остальных каналов будут вычислены кусочно-линейной интерполяцией. Так же в окне настроек расположен график калибровочной кривой для оценки введенных данных.

Архитектура программы

Программная часть проекта разработана на языке C++ по принципам объектно-ориентированного программирования с использованием фреймворка Qt. В ходе разработки использована IDE Qt Creator 4.1. Программа основана на примере “Serial Terminal” из штатного набора примеров в документации на Qt.

Взаимодействия объектов происходят с помощью сигналов и слотов, которые позволяют реализовать принципы event-driven programming. Большая часть функционала программы реализована с помощью методов классов, широко используются контейнеры Qt и стандартные объекты интерфейса. Диаграмма наследования основных классов представлена ниже:

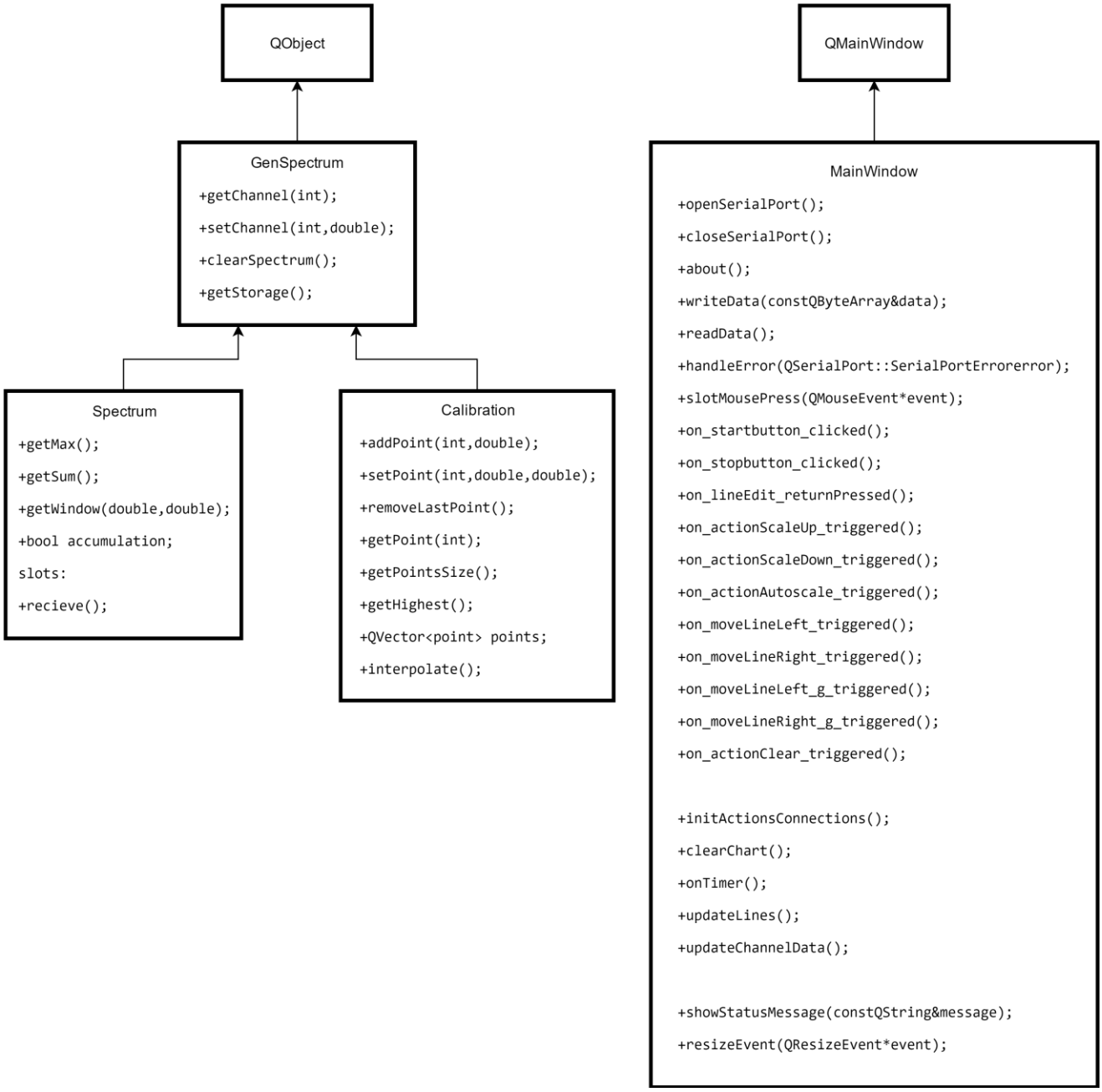
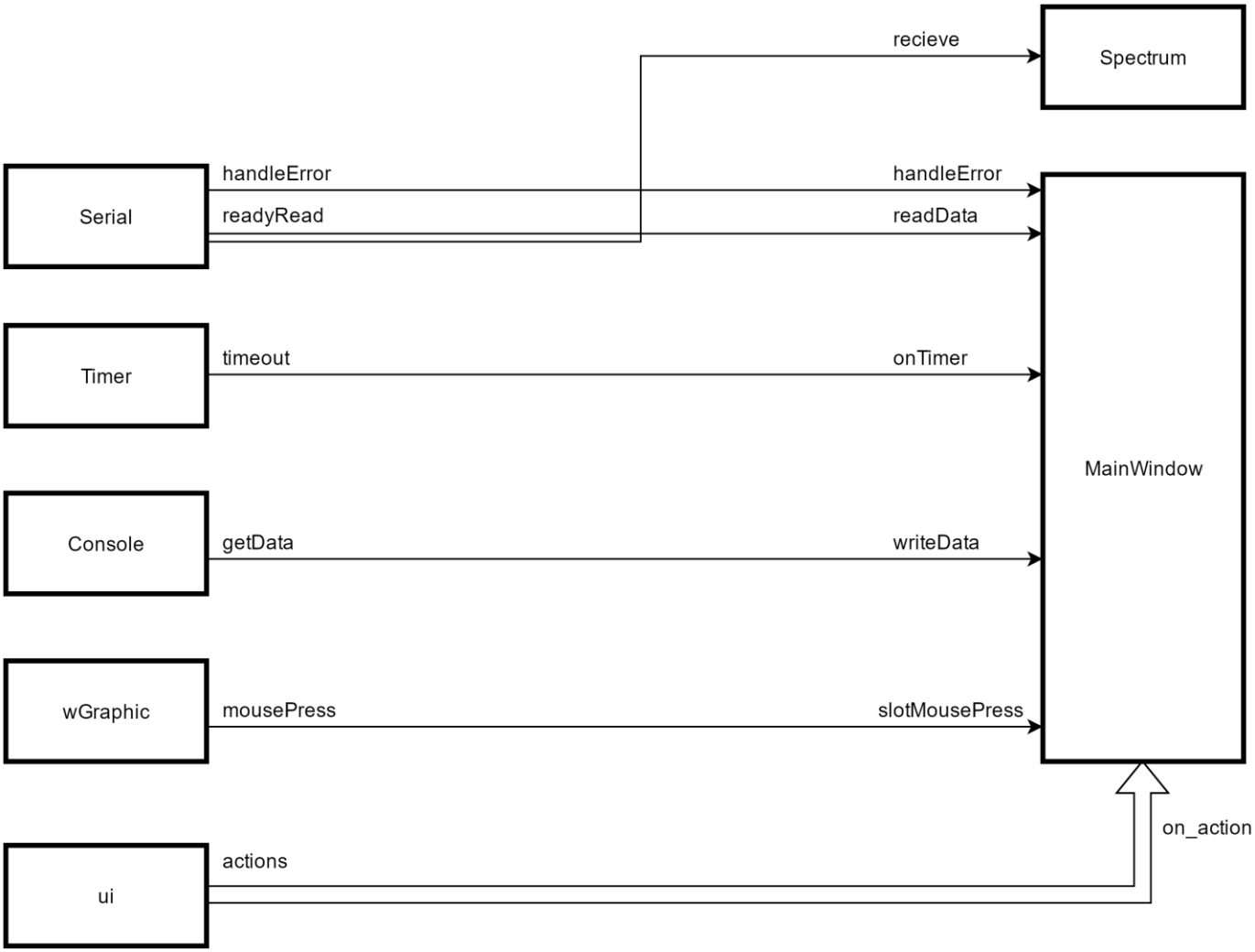


Диаграмма взаимодействий через сигналы и слоты:



Код программы:

mainwindow.cpp:

```
/****************************************************************************
** Spectroscope application
** This is a modified Qt's Serial Monitor example created by:
** Copyright (C) 2012 Denis Shienkov <denis.shienkov@gmail.com>
** Copyright (C) 2012 Laszlo Papp <lpapp@kde.org>
** Contact: https://www.qt.io/licensing/
**
** Modified by Aleksander Mahnyov (C) 2016
** cropemail@gmail.com
** NTUU "KPI"
**
** Created as a final C++ project at KEOA, FEL
***/

#include "mainwindow.h"
#include "ui_mainwindow.h"
```

```
#include "console.h"
#include "settingsdialog.h"
#include "qcustomplot.h"
#include "spectrum.h"
#include <math.h>

#include <QMessageBox>
#include <QLabel>
#include <QtSerialPort/QSerialPort>

QVector<double> coords(1000);

//! [0]
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    //timer, controls and progress bar setup
    timer = new QTimer;
    timer->setInterval(1000);
    ui->lineEdit->setText("1000");
    ui->progressBar->setMinimum(0);
    ui->progressBar->setMaximum(1000);
    ui->startbutton->setEnabled(false);
    ui->stopbutton->setEnabled(false);

    autoscale = false;

    //create console instance
    console = new Console;
    console->setEnabled(false);
    console->setFocusPolicy(Qt::NoFocus);

    //create spectrum storage instance
    spectrum = new Spectrum;
    spectrum->accumulation = false;

    //create bar chart instance
    wGraphic = new QCustomPlot();
    bars = new QCPBars(wGraphic->xAxis, wGraphic->yAxis);

    //fill x vector
    for(int i = 0; i < 1000; i++) coords[i] = i;

    //add widgets to UI
    ui->verticalLayout->addWidget(wGraphic);
    ui->verticalLayout->addWidget(console);
    console->setFixedHeight(70);

    //set chart range
    wGraphic->xAxis->setRange(0,1000);
    wGraphic->yAxis->setRange(0,300);

    //set chart style
    bars->setWidthType(QCPBars::wtPlotCoords);

    bars->setAntialiased(false);//makes them occupy one px
```

```
    bars->setAntialiasedFill(false); //gets rid of outlines
    QBrush barsBrush(Qt::blue, Qt::SolidPattern);
    bars->setBrush(barsBrush);

    xRange = 300;

    //cursors
    QVector<double> x(2) , y(2);
            x[0] = 0;
            y[0] = 0;
            x[1] = 0;
            y[1] = xRange;
    leftLine = new QCPCurve(wGraphic->xAxis, wGraphic->yAxis);
    rightLine = new QCPCurve(wGraphic->xAxis, wGraphic->yAxis);
    wGraphic->addPlottable(leftLine);
    wGraphic->addPlottable(rightLine);
    leftLine->setData(x, y);
    rightLine->setData(x, y);
    leftLine->setPen(QPen(Qt::red));
    leftLine->setAntialiased(false);
    rightLine->setPen(QPen(Qt::green));
    leftLine->setAntialiased(false);

    wGraphic->replot();

    //create serial instance
    serial = new QSerialPort(this);

    settings = new SettingsDialog;

    //activate actions
    ui->actionConnect->setEnabled(true);
    ui->actionDisconnect->setEnabled(false);
    ui->actionQuit->setEnabled(true);
    ui->actionConfigure->setEnabled(true);
    ui->actionAutoscale->setEnabled(true);
    ui->actionScaleDown->setEnabled(true);
    ui->actionScaleUp->setEnabled(true);
    ui->moveLineLeft->setEnabled(true);
    ui->moveLineRight->setEnabled(true);

    status = new QLabel;
    ui->statusBar->addWidget(status);

    initActionsConnections();
}


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


void MainWindow::openSerialPort(){
    SettingsDialog::Settings p = settings->settings();
    qDebug() << p.name << p.baudRate << p.dataBits << p.parity << p.stopBits <<
p.flowControl;
    serial->setPortName(p.name);
```

```
    serial->setBaudRate(p.baudRate);
    serial->setDataBits(p.dataBits);
    serial->setParity(p.parity);
    serial->setStopBits(p.stopBits);
    serial->setFlowControl(p.flowControl);
    if (serial->open(QIODevice::ReadWrite)) {
        console->setEnabled(true);
        //console->setLocalEchoEnabled(p.localEchoEnabled);
        ui->actionConnect->setEnabled(false);
        ui->actionDisconnect->setEnabled(true);
        ui->actionConfigure->setEnabled(false);
        showStatusMessage(tr("Connected to: %1, sum: %2.").arg(serial->portName())
                          .arg(spectrum->getSum()));
        ui->startbutton->setEnabled(true);
    } else {
        //QMessageBox::critical(this, tr("Error"), serial->errorString());
        console->putData("Serial Error! " + serial->errorString() + '\n');
        showStatusMessage(tr("Open error"));
    }
}



void MainWindow::closeSerialPort(){
    if (serial->isOpen())
        serial->close();
    console->setEnabled(false);
    ui->actionConnect->setEnabled(true);
    ui->actionDisconnect->setEnabled(false);
    ui->actionConfigure->setEnabled(true);
    showStatusMessage(tr("Disconnected"));
    on_stopbutton_clicked();
    ui->startbutton->setEnabled(false);
}

void MainWindow::about(){
    QMessageBox::about(this, tr("Spectroscope about"),
                       tr("The <b>Spectroscope app</b> is a deeply modified Qt's "
                          "Terminal example "
                          "done by Mahnyov Aleksander, NTUU KPI. "
                          "contact: cropemail@gmail.com"));
}


void MainWindow::writeData(const QByteArray &data){
    serial->write(data);
}


void MainWindow::readData(){
    console->putData("Bytes at port: ");
    QString data = QString::number(serial->bytesAvailable());
    console->putData(data);
    console->putData("\n");
    showStatusMessage(tr("Connected to: %1, sum: %2.").arg(serial->portName())
                      .arg(spectrum->getSum()));
    if(autoscale){
        xRange = spectrum->getMax()*1.1;
        if(xRange == 0) xRange = 1;
```

```
        wGraphic->yAxis->setRangeUpper(xRange);
    }
    bars->setData(coords, spectrum->getStorage());
    updateLines();
    wGraphic->replot();
}


void MainWindow::handleError(QSerialPort::SerialPortError error){
    if (error == QSerialPort::ResourceError) {
        console->putData("Serial Error! " + serial->errorString());
        on_stopbutton_clicked();
        closeSerialPort();
    }
}

void MainWindow::initActionsConnections(){
    connect(serial, static_cast<void
(QSerialPort::*)(QSerialPort::SerialPortError)>(&QSerialPort::error),
            this, &MainWindow::handleError);
    connect(timer, &QTimer::timeout, this, &MainWindow::onTimer);
    connect(serial, &QSerialPort::readyRead, this, &MainWindow::readData);
    connect(console, &Console::getData, this, &MainWindow::writeData);
    connect(serial, &QSerialPort::readyRead, spectrum, &Spectrum::recieve);
    connect(ui->actionConnect, &QAction::triggered, this, &MainWindow::openSerialPort);
    connect(ui->actionDisconnect, &QAction::triggered, this,
&MainWindow::closeSerialPort);
    connect(ui->actionQuit, &QAction::triggered, this, &MainWindow::close);
    connect(ui->actionConfigure, &QAction::triggered, settings, &MainWindow::show);
    connect(ui->actionConfigure, &QAction::triggered, settings,
&SettingsDialog::changedByUser);
    connect(ui->actionClear, &QAction::triggered, console, &Console::clear);
    connect(ui->actionClear, &QAction::triggered, this, &MainWindow::clearChart);
    connect(ui->actionAbout, &QAction::triggered, this, &MainWindow::about);
    connect(ui->actionAboutQt, &QAction::triggered, qApp, &QApplication::aboutQt);
    connect(wGraphic, &QCustomPlot::mousePress, this, &MainWindow::slotMousePress);
}

void MainWindow::showStatusMessage(const QString &message){
    status->setText(message);
}


void MainWindow::clearChart(){
    spectrum->clearSpectrum();
    bars->setData(coords, spectrum->getStorage());
    wGraphic->replot();
    ui->progressBar->setValue(0);
}

void MainWindow::onTimer(){ //timer value is stored via progress bar (sic)
    int timeout = ui->lineEdit->text().toInt();
    if(ui->progressBar->value() >= timeout){
        on_stopbutton_clicked();
        ui->progressBar->setValue(0);
    }
    else ui->progressBar->setValue(ui->progressBar->value()+1);
}
```

```
void MainWindow::on_startbutton_clicked(){
    ui->progressBar->setMaximum(ui->lineEdit->text().toInt());
    timer->start();
    spectrum->accumulation = true;
    ui->startbutton->setEnabled(false);
    ui->stopbutton->setEnabled(true);
}

void MainWindow::on_stopbutton_clicked(){
    timer->stop();
    spectrum->accumulation = false;
    ui->stopbutton->setEnabled(false);
    ui->startbutton->setEnabled(true);
}

void MainWindow::on_lineEdit_returnPressed(){
    wGraphic->setFocus();
    ui->progressBar->setMaximum(ui->lineEdit->text().toInt());
}

void MainWindow::on_actionScaleUp_triggered(){
    xRange*=2;
    wGraphic->yAxis->setRangeUpper(xRange);
    updateLines();
    wGraphic->replot();
}

void MainWindow::on_actionScaleDown_triggered(){
    xRange/=2;
    wGraphic->yAxis->setRangeUpper(xRange);
    updateLines();
    wGraphic->replot();
}

void MainWindow::on_actionAutoscale_triggered(){
    if(autoscale){
        autoscale = false;
        if(spectrum->getMax()!=0)
            xRange = spectrum->getMax()*1.1;
    }else{
        autoscale = true;
    }
}

void MainWindow::slotMousePress(QMouseEvent *event){
    double clickPos = wGraphic->xAxis->pixelToCoord(event->pos().x());
    if(clickPos > 0 && clickPos < 1000){
        if(event->button() == Qt::LeftButton) leftLinePos = floor(clickPos);
        else if(event->button() == Qt::RightButton) rightLinePos = floor(clickPos);
    }
    updateLines();
    wGraphic->replot();
}

void MainWindow::updateLines(){
    QVector<double> x(2), y(2);
    x[0] = leftLinePos;
    y[0] = 0;
    x[1] = leftLinePos;
```

```
    y[1] = xRange;
    leftLine->setData(x, y);
    x[0] = rightLinePos;
    x[1] = rightLinePos;
    rightLine->setData(x, y);
    updateChannelData();
}

void MainWindow::on_moveLineLeft_triggered(){
    leftLinePos--;
    if(leftLinePos < 0) leftLinePos = 0;
    updateLines();
    wGraphic->replot();
}

void MainWindow::on_moveLineRight_triggered(){
    leftLinePos++;
    if(leftLinePos > 999) leftLinePos = 999;
    updateLines();
    wGraphic->replot();
}

void MainWindow::updateChannelData(){
    SettingsDialog::Settings p = settings->settings();
    ui->channelData->setText(tr("Pulses at channel: %1: %2. Channel's energy: %3 MeV.
Sum in window: %4.")
                             .arg(leftLinePos, 0, 'L', 0)
                             .arg(spectrum->getChannel(leftLinePos), 0, 'L', 0)
                             .arg(p.curCalibration->getStorage().at(leftLinePos))
                             .arg(spectrum->getWindow(leftLinePos, rightLinePos), 0,
'L', 0));
}

void MainWindow::on_moveLineLeft_g_triggered(){
    rightLinePos--;
    if(rightLinePos < 0) rightLinePos = 0;
    updateLines();
    wGraphic->replot();
}

void MainWindow::on_moveLineRight_g_triggered(){
    rightLinePos++;
    if(rightLinePos > 999) rightLinePos = 999;
    updateLines();
    wGraphic->replot();
}

void MainWindow::on_actionClear_triggered(){
    xRange = 300;
    updateLines();
}

void MainWindow::resizeEvent(QResizeEvent* event){
    QMainWindow::resizeEvent(event);
    qDebug() << "Resize";
    bars->setWidth(wGraphic->width()/1000); //gets rid of white stripes
}
```

spectrum.cpp:

```
#include "spectrum.h"
#include "mainwindow.h"

Spectrum::Spectrum(){
    storage.resize(1000);
    storage.fill(0);
}

void Spectrum::recieve(){
    QSerialPort* serial = static_cast<QSerialPort*>(QObject::sender());
    QString data;
    while(serial->canReadLine()){ //yup, this is parser. TODO: create error handlers
        data = serial->readLine();
        if(accumulation){
            QStringList dataList = data.split('!');
            if(dataList.size() == 2){
                QString x = dataList.at(0);
                double X = x.toDouble();
                QString y = dataList.at(1);
                y.remove('\n');
                y.remove('\r');
                double Y = y.toDouble();
                if(X < 1000) GenSpectrum::storage[X] += Y;
            }
        }
    }
}

double Spectrum::getMax(){
    return *std::max_element(GenSpectrum::storage.begin(), GenSpectrum::storage.end());
}

double Spectrum::getSum(){
    double sum = 0;
    for(int i = 0; i < GenSpectrum::storage.size(); i++)
        sum += GenSpectrum::storage[i];
    return sum;
}

double Spectrum::getWindow(double start, double finish){
    double sum = 0;
    if(start > GenSpectrum::storage.size()) start = GenSpectrum::storage.size();
    if(finish > GenSpectrum::storage.size()) finish = GenSpectrum::storage.size();
    if(start == finish) return 0;
    if(start > finish){
        int tmp = start;
        start = finish;
        finish = tmp;
    }
    for(int i = start; i < finish; i++)
        sum += GenSpectrum::storage[i];
    return sum;
}

Calibration::Calibration(){
    GenSpectrum::storage.resize(1000);
```

```
    GenSpectrum::storage.fill(0);
    //points.resize(5);
}

void Calibration::addPoint(int x, double y){
    point tmpPoint;
    tmpPoint.x = x;
    tmpPoint.y = y;
    points.push_back(tmpPoint);
    interpolate();
}

void Calibration::removeLastPoint(){
    points.pop_back();
    interpolate();
}

point Calibration::getPoint(int index){
    if(index < points.size())
        return points[index];
    else{
        point tmp;
        tmp.x = 0;
        tmp.y = 0;
        return tmp;
    }
}

void Calibration::setPoint(int index, double X, double Y){
    if(index <= points.size()){
        points[index].x = X;
        points[index].y = Y;
        interpolate();
    }
}

int Calibration::getPointsSize(){
    return points.size();
}

point Calibration::getHighest(){
    point tmpPoint;
    tmpPoint.y = points[0].y;
    for(int i = 0; i < points.size(); i++){
        if(points[i].y > tmpPoint.y) tmpPoint = points[i];
    }
    return tmpPoint;
}

void Calibration::interpolate(){
    for(int i = 0; i < points.size()-1; i++){
        for(double j = points[i].x; j < points[i+1].x; j++){
            storage[j] = points[i].y + (j-points[i].x)*(points[i+1].y-
points[i].y)/(points[i+1].x-points[i].x);
        }
    }
}

GenSpectrum::GenSpectrum(){
```

```
    storage.resize(1000);
    storage.fill(0);
}

void GenSpectrum::setChannel(int channel, double value){
    storage[channel] = value;
}

QVector<double> GenSpectrum::getStorage(){
    return storage;
}

void GenSpectrum::clearSpectrum(){
    storage.fill(0);
}

double GenSpectrum::getChannel(int channel){
    return storage[channel];
}
```

Остальные файлы программы имеют низкую долю вмешательства автора, либо слишком велики. Целиком с кодом проекта можно ознакомиться на GitHub: https://github.com/AMahno/KPIcpp/tree/master/lab5qt/lab5 (коммит на момент сдачи отчета: `ca853ed`).

Дальнейшее развитие проекта

В ходе разработки программно-аппаратного комплекса была обнаружена надобность в таких усовершенствованиях:

- Переход на более сжатый формат передачи данных (пакеты чисел в битовом формате вместо ASCII-строк)
- Создание механизма определения обрывов связи и её автоматического восстановления
- Реализация метода определения типа нуклидов по полученным данным
- Реализация калибровки по эффективности регистрации для определения мощности эквивалентной дозы
- Реализация записи полученных данных в формате файлов записи спектров ведущих международных производителей спектрометров

Выводы

В ходе работы над данным проектом был создан рабочий и практически применимый гамма-спектроскоп. Аппаратная и программная часть устройства имеют потенциал на развитие и дальнейшее применение. Преобразователь тау-кода, который был создан в ходе работы, применим для использования с другими, более высококачественными детекторами.

Автором в ходе работы были изучены принципы работы и получены навыки работы с фреймворком Qt, создания приложений с графическим интерфейсом. Так же были получены навыки разработки программ для систем реального времени с асинхронным характером работы.