Микроволновая Электроника

Радиоприемные устройства измерительные MWR-40U MWR-85U MWR-100U MWR-135U

Примеры программ управления

Редакция 002

CSMWR002

01.2017

Содержание

1 Системные требования

	МИНИМАЛЬНЫЕ СИСТЕМНЫЕ ТРЕБОВАНИЯ
Операционная система	Windows® 7 и новее, GNU/Linux Ubuntu 16.04 или новее
Процессор	Архитектура x86 или ARM Little-endian
Оперативная память	1 GB RAM
Свободное место	128 MB свободного места на жестком диске
Интернет подключение	Ethernet-подключение, 1Gbit/s

Замечание: в таблице указаны минимальные параметры для запуска программ. Следует отметить, что максимальная длительность временной выборки и максимальная полоса спектра определяется, в том числе, объемом имеющейся свободной оперативной памяти и свободного объема памяти НЖМД.

2 Требования к программному обеспечению

Для работы примеров получения данных:

Среда разработки	QtCreator 4.10.2 и новее или Microsoft Visual Studio 2015 и новее, либо текстовый редактор и командная строка
Библиотеки	Фреймворк Qt 5.10 и новее
Компиляторы	MSVC 2015 и новее или GCC 5 и новее (необходима поддержка стандарта C++11)

Для сборки примеров кода следует использовать сборочную систему, основанную на qmake, из состава фреймворка Qt, согласно документации на фреймворк (https://doc.qt.io/qt-5/qmake-manual.html).
Для работы примеров визуализации полученных данных возможно использование Mathworks MATLAB либо следующего набора компонентов Python:

Библиотеки	Matplotlib, Numpy, Tkinter
Интерпретатор	Python 3.7

3 Пример получения спектральной выборки

Пример получения спектральной выборки реализован на языке C++ с использованием библиотеки Qt. Пример ожидает IP адрес и TCP порт РПУ в качестве входных параметров. Параметры сканирования заданы в разделе //*Параметры сканирования*.

```
#include <QtCore>
#include <QtNetwork>
#include <QtDebug>

#include <iostream>

bool parseUdpDatagram(const char * buffer, int sz, int r_id_0,
                      int & offset,int & dataOffset, int & dataSize, bool &mf) {
    const char * end=buffer+sz;
    const char * pe=std::find(buffer+6,end,';');
    if(pe!=end) {
        const char * pb=buffer+6;
        if(pb<end) {
            int r_id=strtol(pb,0,10); // R_ID, идентификатор запроса
            if(r_id!=r_id_0) {
                return false;
            }
            pb=pe+1;
            if(pb<end) {
                pe=std::find(pb,end,';');
                if(pe<end) {
                    offset=strtol(pb,0,10); // OFFSET, смещение начала данных
                                            // пакета в потоке данных
                    pb=pe+1;
                    if(pb<end) {
                        pe=std::find(pb,end,';');
                        if(pe<end) {
                            dataSize=strtol(pb,0,10); // DATA SIZE, количество байт
                                                      // с данным в этом пакете
                            pb=pe+1;
                            if(pb<end) {
                                pe=std::find(pb,end,';');
                                if(pe<end) {
                                    int nmf=strtol(pb,0,10); // MF, флаг наличия
                                                             // следующих пакетов с данными
                                    pb=pe+1;
                                    dataOffset=pb-buffer;
                                    if((sz-dataOffset)==dataSize) {
                                        mf=nmf;
                                        return true;
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
```

```
        }
    }
    return false;
}

// Отправка команды по TCP протоколу и запрос кодов ошибок
// Если не было обнаружено ошибок возвращает true иначе false

// write command to tcp socket and checks if there is no error
// return true in no errors, otherwise return false
bool sendCommand(QTcpSocket &tcp, const char *command) {
    tcp.write(command);
    if(strlen(command)>0) {
        qDebug()<<(QByteArray("<<< ")+command).constData();
    }
    tcp.flush();
    tcp.write("SYST:ERR:CODE:ALL?\n");
    qDebug()<<"<<< SYST:ERR:CODE:ALL?\n";
    tcp.flush();
    tcp.waitForReadyRead(1000);
    auto err = tcp.readAll();
    qDebug()<<(">>> "+err).constData();
    if (!err.startsWith("0")) {
        return false;
    }

    return true;
}


int main(int argc, char*argv[]) {
    QCoreApplication app(argc,argv);

    QTextStream cout(stdout, QIODevice::WriteOnly);
    QTextStream cin(stdin);

    cout << "Enter device ip" <<endl;
    auto deviceIP = cin.readLine();
    if (deviceIP.isEmpty()) { deviceIP = "192.168.7.10"; }

    cout << "Enter device Tcp port"<<endl;
    quint16 deviceTcpPort = cin.readLine().toUInt();
    if (deviceTcpPort==0) { deviceTcpPort = qint16(10100); }

    // Параметры запроса спектра
    // Request params
    const double centerFrequencyHz=1800e6;  // центральная частота
    const double rbw=10000;                      // спектральное разрешение,
                                                 // указываемое в запросе
    const double rbw_real=12207.03125;           // спектральное разрешение,
                                                 // возвращаемое РПУ
    const int if_bw=260;                         // полоса ПЧ, МГц
    const int r_id=568;                          // идентификатор запроса
```

```
    const int nFFT=32768;                                   // число спектральных отсчетов,
                                                            // которые вернет приемник
                                                            // при заданном RBW

    // Определение IP адреса локального компьютера.
    // Пример исходит из наличия единственного сетевого адаптера
    // у данного компьютера

    // PC params
    // Get available ip address to bind
    QHostAddress pcIP = QHostAddress(QHostAddress::LocalHost);
    for (const QHostAddress &address: QNetworkInterface::allAddresses()) {
        if (address.protocol() == QAbstractSocket::IPv4Protocol && address != pcIP)
{
            pcIP = address;
            break;
        }
    }

    // порт UDP для прослушивания. Взят произвольным образом >1024.
    const quint16 pcUdpPort=10091;

    // оценка таймаута получения данных
    const qint64 timeoutMS=std::ceil(310.0+1000.0*1.4/rbw) + 1000;

    // Буфер для получаемых данных
    // Buffer
    QByteArray rcvBuf(nFFT*sizeof(qint16),char(0));

    // число полученных от РПУ байт
    int bytesWritten=0;

    // Таймер, отсчитывающий время с момента отправки команды инициации регистрации
    // данных до прихода самих данных
    QTimer t;
    t.setSingleShot(true);

    // Эта функция вызывается, когда все данные будут получены, или если возникла
    // ошибка
    // Is called when all data received or when some error ocuured
    auto finalize=[&]() {
        t.stop();

        if(bytesWritten<int(nFFT*sizeof(qint16))) {
            // часть данных была потеряна из-за использования протокола UDP без
            // гарантированной доставки
            cout <<"UDP packets lost, spectrum is incorrect. Received "
                 <<bytesWritten<<", expected "<<rcvBuf.size()<<endl;

            return;
        }

        // Преобразование во float и вывод в файл "out.bin"
```

```
        // Write collected data to file
        QVector<float> dBm(nFFT);
        const qint16 * pData=reinterpret_cast<const qint16*>(rcvBuf.constData());
        std::transform(pData,pData+nFFT,dBm.begin(),[](const qint16 d)->float{
            return d*0.0117589842-102;
        });
        QFile f("out.bin");
        if(!f.open(QIODevice::WriteOnly)) {
            cout <<"Error opening file"<<endl;
            app.exit(3);
            return;
        }
        f.write(reinterpret_cast<const char*>(dBm.constData()),nFFT*sizeof(float));
        f.close();
        // ---------------------------
        // успех, завершение работы, печать команд для визуализации результатов
        // в случае использования Mathworks MATLAB
        cout<<"Mission Accomplished, "<<bytesWritten<<" bytes written ("
            <<nFFT<<" samples)"<<endl;
        cout<<"Your file is located at "<<QCoreApplication::applicationDirPath()
            +"/" << f.fileName()<<endl;
        cout<<endl;
        cout<<"You can plot the spectrum in MATLAB using "
              "following MATLAB commands:"<<endl;
        cout<<"---MATLAB code begin---"<<endl;
        cout<<("cd '"+
            QCoreApplication::applicationDirPath()+"'").toStdString().c_str()
            <<endl;
        cout<<("f=fopen('"+f.fileName()+"','r');").toStdString().c_str()<<endl;
        cout<<"A=fread(f,inf,'single=>single');"<<endl;
        cout<<"NFFT="<<nFFT<<";"<<endl;
        cout<<"rbw="<<rbw_real<<";"<<endl;
        cout<<"f=[-NFFT/2+(0:(NFFT-1))]*rbw+"<<centerFrequencyHz<<";"<<endl;
        cout<<"plot(f(2:end),A(2:end));"<<endl;
        cout<<"---MATLAB code end---"<<endl;
        app.quit();
    };


    QTcpSocket tcp;
    QUdpSocket udp;

    // Задание параметров и начало записи, вызывается при подключении сокета TCP
    // Call to setup params and start recording
    auto startTask = [&] {
        tcp.setSocketOption(QAbstractSocket::LowDelayOption, 1);
        cout << "tcp socket connected" << endl;
        // получение ошибок из очереди ошибок
        // purge error queue
        if (!sendCommand(tcp, ""))                    { finalize(); return; }

        // Удаление старого тега UDP
        // Delete old data
```

```
        if (!sendCommand(tcp, "TRAC:UDP:DEL ALL\n"))    { finalize(); return; }

        // Добавление тега UDP-потока, который будет содержать
        // результаты измерений, на локальный адрес и порт
        // Create UDP flow with scan result on localHost address and port
        if (!sendCommand(tcp, "TRAC:UDP:TAG \"" +
            QByteArray::fromStdString(pcIP.toString().toStdString())
            + "\", " + "10091" + ", FSC\n"))
        { finalize(); return; }

        // Задание центральной частоты
        // Set center Frequency
        if (!sendCommand(tcp, "FREQ " +
            QByteArray::number(centerFrequencyHz, 'f', 3) + " Hz\n"))
        { finalize(); return; }

        // Задание спектрального разрешения
        // Set RBW
        if (!sendCommand(tcp, "BWID " + QByteArray::number(rbw, 'f', 1) + " Hz\n"))
        { finalize(); return; }

        // Задание фильтра ПЧ
        // Задание IFBW
        // Set Intermediate Frequency filter bandwidth
        if (!sendCommand(tcp, "BWID:IF " + QByteArray::number(if_bw) + " MHz\n"))
        { finalize(); return; }

        // Установка идентификатора запроса
        // Set UDP packet id
        if (!sendCommand(tcp, "TRAC:UDP:RID " + QByteArray::number(r_id) + "\n"))
        { finalize(); return; }

        // Инициация регистрации данных
        // Execute commands
        if (!sendCommand(tcp, "INIT:IMM\n") )
        { finalize(); return; }

        cout << "Start recording" << endl;
        t.start(timeoutMS);
    };

    // Обработка данных, получаемых по UDP.
    // Вызывается по сигналу QUDPSocket::readyRead
    // Call when udp socket is readyRead
    auto udpDataReceived = [&] {
        qint64 datagramSize=0;
        bool mf=true;
        QByteArray buf(65536,char(0));
        // если есть входящие пакеты
        while(udp.hasPendingDatagrams()) {
            datagramSize=udp.pendingDatagramSize();

            // пакет не может быть менее 40 байт размером -
```

```
            // значит, это пакет не от РПУ
            if(datagramSize<40) {
                udp.readDatagram(0,0);
                continue;
            }

            if(datagramSize>buf.size()) {
                buf.resize(datagramSize);
            }

            // получение пакета
            datagramSize=udp.readDatagram(buf.data(),datagramSize);
            int offset=0;
            int dataOffset=0;
            int dataSize=0;
            bool moreFrames=true;
            // разбор заголовка пакета
            if(parseUdpDatagram(buf.constData(), datagramSize, r_id,
                                offset,dataOffset, dataSize, moreFrames)) {
                mf = mf && moreFrames;
                cout <<  offset << " " << dataSize << " " << bool(moreFrames)
                     << endl;
                // копирование данных пакета на нужное место в выходной буфер
                std::copy_n(buf.constData()+dataOffset,dataSize,
                    rcvBuf.data()+offset);
                bytesWritten+=dataSize;
            }
        }
        if(!mf) {
            // получен последнй пакет, можно выходить
            finalize();
        }
    };

    // Обработка изменения состояния сокета TCP
    // Call when tcp state changed
    auto onTcpStateChanged = [&](QAbstractSocket::SocketState state) {
        switch(state) {
        case QAbstractSocket::UnconnectedState:
            qDebug()<<"TCP Socket not connected";
            app.exit(1);
            return;
        case QAbstractSocket::ConnectedState:
            // сокет подключен, начинается задание записи
            startTask();
            break;
        default:
            break;
        }
    };

    // Обработка изменения состояния сокета UDP
    // Call when udp state changed
```

```
    auto onUdpStateChanged = [&](QAbstractSocket::SocketState state) {
        switch(state) {
        case QAbstractSocket::BoundState:
            // Настройка буфера ОС для получаемых через данный сокет данных.
            // Без этой настройки данные могут чаще теряться
            udp.setSocketOption(QAbstractSocket::ReceiveBufferSizeSocketOption,
               1024*1024);
            break;
        default:
            break;
        }
    };

    // Установка нужных соединений
    // Establish connections
    QObject::connect(&t, &QTimer::timeout, finalize);

    QObject::connect(&tcp, &QTcpSocket::stateChanged, onTcpStateChanged);

    QObject::connect(&udp, &QUdpSocket::stateChanged, onUdpStateChanged);

    QObject::connect(&udp, &QUdpSocket::readyRead, udpDataReceived);

    // ---------------

    // Подключение сокета UDP
    if(!udp.bind(pcUdpPort)) {
        qDebug()<<"Unable to bind port "<< pcUdpPort;
        return 2;
    }

    // Подключение сокета TCP
    // Connect to receiver
    tcp.connectToHost(deviceIP, deviceTcpPort);

    return app.exec();
}
```

4 Пример получения временной выборки

Пример получения временной выборки реализован на языке C++ с использованием библиотеки Qt. Пример ожидает IP адрес и TCP порт РПУ в качестве входных параметров. Параметры сканирования заданы в разделе //*Параметры сканирования*.

```
#include <QtCore>
#include <QtNetwork>
#include <QtDebug>

#include <iostream>

bool parseUdpDatagram(const char * buffer, int sz, int r_id_0,
                      int & offset,int & dataOffset, int & dataSize, bool &mf) {
    const char * end=buffer+sz;
    const char * pe=std::find(buffer+6,end,';');
    if(pe!=end) {
        const char * pb=buffer+6;
        if(pb<end) {
            int r_id=strtol(pb,0,10); // R_ID, идентификатор запроса
            if(r_id!=r_id_0) {
                return false;
            }
            pb=pe+1;
            if(pb<end) {
                pe=std::find(pb,end,';');
                if(pe<end) {
                    offset=strtol(pb,0,10); // OFFSET, смещение начала данных
                                            // пакета в потоке данных
                    pb=pe+1;
                    if(pb<end) {
                        pe=std::find(pb,end,';');
                        if(pe<end) {
                            dataSize=strtol(pb,0,10); // DATA SIZE, количество байт
                                                      // с данными в этом пакете
                            pb=pe+1;
                            if(pb<end) {
                                pe=std::find(pb,end,';');
                                if(pe<end) {
                                    int nmf=strtol(pb,0,10); // MF, флаг наличия
                                                             // следующих пакетов с данными
                                    pb=pe+1;
                                    dataOffset=pb-buffer;
                                    if((sz-dataOffset)==dataSize) {
                                        mf=nmf;
                                        return true;
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
```

```
        }
    }
    return false;
}

// Отправка команды по TCP протоколу и запрос кодов ошибок
// Если не было обнаружено ошибок возвращает true иначе false

// write command to tcp socket and checks if there is no error
// return true in no errors, otherwise return false
bool sendCommand(QTcpSocket &tcp, const char *command) {
    tcp.write(command);
    if(strlen(command)>0) {
        qDebug()<<(QByteArray("<<< ")+command).constData();
    }
    tcp.flush();
    tcp.write("SYST:ERR:CODE:ALL?\n");
    qDebug()<<"<<< SYST:ERR:CODE:ALL?\n";
    tcp.flush();
    tcp.waitForReadyRead(1000);
    auto err = tcp.readAll();
    qDebug()<<(">>> "+err).constData();
    if (!err.startsWith("0")) {
        return false;
    }

    return true;
}


int main(int argc, char*argv[]) {
    QCoreApplication app(argc,argv);

    QTextStream cout(stdout, QIODevice::WriteOnly);
    QTextStream cin(stdin);

    cout << "Enter device ip" <<endl;
    auto deviceIP = cin.readLine();
    if (deviceIP.isEmpty()) { deviceIP = "192.168.7.10"; }

    cout << "Enter device Tcp port"<<endl;
    quint16 deviceTcpPort = cin.readLine().toUInt();
    if (deviceTcpPort==0) { deviceTcpPort = qint16(10100); }

    // Параметры запроса временной выборки
    // Request params
    const double centerFrequencyHz=101.7e6; // центральная частота 101.7 МГц
    const int if_bw=20;                     // полоса ПЧ, 20 МГц
    const int r_id=568;                     // идентификатор запроса
    const int sampleCount=4000;             // число получаемых отсчетов
    const double adcSampleRate=400e6;       // частота дискретизации АЦП РПУ
                                            // равна 400 MHz
    const int decimationFactor=1200;        // коэффициент децимации
```

```
const double sampleRate=adcSampleRate/decimationFactor;
                                        // частота дискретизации получаемой
                                        // временной выборки
const double atten=0;                   // входной аттенюатор 0 дБ


// Определение IP адреса локального компьютера.
// Пример исходит из наличия единственного сетевого адаптера
// у данного компьютера

// PC params
// Get available ip address to bind
QHostAddress pcIP = QHostAddress(QHostAddress::LocalHost);
for (const QHostAddress &address: QNetworkInterface::allAddresses()) {
    if (address.protocol() == QAbstractSocket::IPv4Protocol && address != pcIP)
    {
         pcIP = address;
         break;
    }
}

// порт UDP для прослушивания. Взят произвольным образом >1024.
const quint16 pcUdpPort=10091;

// оценка таймаута получения данных
const qint64 timeoutMS=std::ceil(310.0+sampleCount*1000.0/sampleRate);

// Буфер для получаемых данных
// Buffer
QByteArray rcvBuf(sampleCount*sizeof(qint16) * 2,char(0));
int bytesWritten=0;

// Таймер, отсчитывающий время с момента отправки команды инициации регистрации
// данных до прихода самих данных
QTimer t;
t.setSingleShot(true);

// Эта функция вызывается, когда все данные будут получены,
// или если возникла ошибка
// Is called when all data received or when some error ocuured
auto finalize=[&]() {
    t.stop();

    if(bytesWritten<int(sampleCount * sizeof(qint16) * 2)) {
        // часть данных была потеряна из-за использования протокола UDP
        // без гарантированной доставки
        // можно попробовать уменьшить Flow Control или увеличить размер
        // QAbstractSocket::ReceiveBufferSizeSocketOption
        cout <<"UDP packets lost, spectrum is incorrect. Received"
               <<bytesWritten<<" Expected"<<rcvBuf.size()<<endl;

        return;
    }
```

```
        // Вывод в файл "out.bin"
        // Write collected data to file
        QFile f("out.bin");
        if(!f.open(QIODevice::WriteOnly)) {
            cout <<"Error opening file"<<endl;
            app.exit(3);
            return;
        }
        f.write(reinterpret_cast<const char *>(rcvBuf.constData()), rcvBuf.size());
        f.close();
        // ---------------------------
        // успех, завершение работы, печать команд для визуализации результатов
        // в случае использования Mathworks MATLAB
        cout<<"Mission Accomplished, "<<bytesWritten
            <<" bytes written ("<<sampleCount<<" samples)"<<endl;
        cout<<"Your file is located at "<<QCoreApplication::applicationDirPath()
             +"/" << f.fileName()<<endl;
        cout<<endl;
        cout<<"You can plot the time series in MATLAB using "
              "following MATLAB commands:"<<endl;
        cout<<"---MATLAB code begin---"<<endl;
        cout<<("cd '"+
               QCoreApplication::applicationDirPath()+"'").toStdString().c_str()
            <<endl;
        cout<<("f=fopen('"+f.fileName()+"','r');").toStdString().c_str()<<endl;
        cout<<"A=fread(f,inf,'int16=>int16');"<<endl;
        cout<<"fs="<<sampleRate<<";"<<endl;
        cout<<"complexSamples=[A(1:2:end) A(2:2:end)];"<<endl;
        cout<<"plot((1:(numel(A)/2))'/fs,complexSamples);"<<endl;
        cout<<"legend('I','Q');"<<endl;
        cout<<"---MATLAB code end---"<<endl;
        app.quit();
    };


    QTcpSocket tcp;
    QUdpSocket udp;

    // Задание параметров и начало записи, вызывается при подключении сокета TCP
    // Call to setup params and start recording
    auto startTask = [&] {
        tcp.setSocketOption(QAbstractSocket::LowDelayOption, 1);
        cout << "tcp socket connected" << endl;
        // получение ошибок из очереди ошибок
        // purge error queue
        if (!sendCommand(tcp, "")) { finalize(); return; }

        // Удаление старого тега UDP
        // Delete old data
        if (!sendCommand(tcp, "TRAC:UDP:DEL ALL\n"))
{ finalize(); return; }
```

```
        // Добавление тега UDP-потока, который будет содержать
        // результаты измерений, на локальный адрес и порт
        // Create UDP flow with scan result on localHost address and port
        if (!sendCommand(tcp, "TRAC:UDP:TAG \"" +
            QByteArray::fromStdString(pcIP.toString().toStdString())
                                + "\", " + "10091" + ", IQ\n"))
        { finalize(); return; }

        // Задание длительности регистрации (числа отсчетов)
        // Set number of points in timeseries
        if (!sendCommand(tcp, "TRAC:POIN " + QByteArray::number(sampleCount) +
           "\n"))
        { finalize(); return; }

        // Задание центральной частоты
        // Set center Frequency
        if (!sendCommand(tcp, "FREQ " +
            QByteArray::number(centerFrequencyHz, 'f', 3) + " Hz\n"))
        { finalize(); return; }

        // Задание коэффициента децимации
        // Set decimation factor
        if (!sendCommand(tcp, "DECF "+QByteArray::number(decimationFactor)))
        { finalize(); return; }

        // Задание фильтра ПЧ
        // Set Intermediate frequency filter bandwidth
        if (!sendCommand(tcp, "BWID:IF " + QByteArray::number(if_bw) + " MHz\n"))
        { finalize(); return; }

        // Задание входного аттенюатора
        // Input Attenuator setup
        if (!sendCommand(tcp, "INP:ATT " + QByteArray::number(atten) + " dB"))
        { finalize(); return; }

        // Установка идентификатора запроса
        // Set UDP packet id
        if (!sendCommand(tcp, "TRAC:UDP:RID " + QByteArray::number(r_id) + "\n"))
        { finalize(); return; }

        // Инициация регистрации данных
        // Execute commands
        if (!sendCommand(tcp, "INIT:IMM\n") )
        { finalize(); return; }
        cout << "Start recording" << endl;
        t.start(timeoutMS);
    };

    // Обработка данных, получаемых по UDP.
    // Вызывается по сигналу QUDPSocket::readyRead
    // Call when udp socket is readyRead
    auto udpDataReceived = [&] {
        qint64 datagramSize=0;
```

```
        bool mf=true;
        QByteArray buf(65536,char(0));
        // если есть входящие пакеты
        while(udp.hasPendingDatagrams()) {
            datagramSize=udp.pendingDatagramSize();

            // пакет не может быть менее 40 байт размером
            // - значит, это пакет не от РПУ
            if(datagramSize<40) {
                udp.readDatagram(0,0);
                continue;
            }

            if(datagramSize>buf.size()) {
                buf.resize(datagramSize);
            }

            // получение пакета
            datagramSize=udp.readDatagram(buf.data(),datagramSize);
            int offset=0;
            int dataOffset=0;
            int dataSize=0;
            bool moreFrames=true;
            // разбор заголовка пакета
            if(parseUdpDatagram(buf.constData(), datagramSize, r_id,
                                offset,dataOffset, dataSize, moreFrames)) {
                mf = mf && moreFrames;
                cout <<  offset << " " << dataSize << " " << bool(moreFrames) <<
                         endl;
                // копирование данных пакета на нужное место в выходной буфер
                std::copy_n(buf.constData()+dataOffset,dataSize,
                    rcvBuf.data()+offset);
                bytesWritten+=dataSize;
            }
        }
        if(!mf) {
            // получен последнй пакет, можно выходить
            finalize();
        }
    };

    // Обработка изменения состояния сокета TCP
    // Call when tcp state changed
    auto onTcpStateChanged = [&](QAbstractSocket::SocketState state) {
        switch(state) {
        case QAbstractSocket::UnconnectedState:
            qDebug()<<"TCP Socket not connected";
            app.exit(1);
            return;
        case QAbstractSocket::ConnectedState:
            // сокет подключен, начинается задание записи
            startTask();
            break;
```

```
        default:
            break;
        }
    };

    // Обработка изменения состояния сокета UDP
    // Call when udp state changed
    auto onUdpStateChanged = [&](QAbstractSocket::SocketState state) {
        switch(state) {
        case QAbstractSocket::BoundState:
            udp.setSocketOption(QAbstractSocket::ReceiveBufferSizeSocketOption,
                1024*1024);
            break;
default:
            break;
        }
    };

    // Установка нужных соединений
    // Establish connections

    QObject::connect(&t, &QTimer::timeout, finalize);

    QObject::connect(&tcp, &QTcpSocket::stateChanged, onTcpStateChanged);

    QObject::connect(&udp, &QUdpSocket::stateChanged, onUdpStateChanged);

    QObject::connect(&udp, &QUdpSocket::readyRead, udpDataReceived);

    // ---------------

    // Подключение сокета UDP
    if(!udp.bind(pcUdpPort)) {
        qDebug()<<"Unable to bind port "<< pcUdpPort;
        return 2;
    }

    // Подключение сокета TCP
    // Connect to receiver
    tcp.connectToHost(deviceIP, deviceTcpPort);

    return app.exec();
}
```

5 Визуализация результатов работы примеров

По окончании работы примеры выводят код, написанный на М-языке среды Mathworks MATLAB, который можно использовать для визуализации записанных с РПУ данных путем копирования и вставки из буфера обмена в окно команд данной среды. Также для визуализации можно использовать приведенные ниже скрипты на языке Python.

5.1 Пример визуализации спектральной выборки на языке Python

Пример визуализации спектральной выборки реализован на языке Python версия 3.7.

```
from matplotlib import pyplot as plt
import numpy as np
import tkinter as tk
from tkinter import filedialog

# Откройте файл с записанными данными.
root = tk.Tk()
root.withdraw()
fileName = filedialog.askopenfilename()
f = open(fileName, 'rb')

data = np.fromfile(f, dtype=np.float32)
f.close()
print (data)

plt.plot(data)
plt.grid()
plt.show()
```

5.2 Пример визуализации временной выборки на языке Python

Пример визуализации временной выборки реализован на языке Python версия 3.7.

```
from matplotlib import pyplot as plt
import numpy as np
import tkinter as tk
from tkinter import filedialog

# Откройте файл с записанными данными.
root = tk.Tk()
root.withdraw()
fileName = filedialog.askopenfilename()
f = open(fileName, 'rb')

data = np.fromfile(f, dtype=np.int16)
f.close()
print (data)

plt.plot(data[0:][::2], label="I")
plt.plot(data[1:][::2], label="Q")
plt.legend()
plt.grid()
plt.show()
```

Микроволновая Электроника

Радиоэлектронное оборудование повышенной сложности. Разработка и производство

тел.: +7 (495) 137-53-35
e-mail: info@mwel.ru
http://mwel.ru