

Векторные Генераторы Сигналов

MWT-60U

MWT-100U

MWT-160U

Примеры программ управления



Содержание

1.	Системные требования	3
2.	Требования к программному обеспечению	3
3.	Пример задания сигнала с цифровой модуляцией	4

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.

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>).

3. Пример задания сигнала с цифровой модуляцией

Пример рассматривает задание сигнала со следующими параметрами:

Несущая частота	137 МГц
Тип модуляции	BPSK
Символьная частота	400 ксимв/сек
Baseband-фильтр	Корень из приподнятого косинуса
Коэффициент скругления фильтра	0.35
Источник данных для модуляции	Последовательность PRBS23

Пример реализован на языке C++ с использованием фреймворка Qt. Пример ожидает IP адрес и TCP порт генератора в качестве входных параметров.

```
#include <QtCore>

#include <QtNetwork>
#include <QtDebug>

#include <iostream>

// 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 = quint16(cin.readLine().toUInt());
if (deviceTcpPort==0) { deviceTcpPort = qint16(10100); }

// Эта функция вызывается,
// когда состояние генератора установлено
// или если возникла ошибка
// Call if everything is done or if some error occurred
auto finalize=[&]() {
    QCoreApplication::quit();
};

QTcpSocket tcp;

// Call to setup params
auto startTask = [&] {
    tcp.setSocketOption(QAbstractSocket::LowDelayOption, 1);
    cout << "tcp socket connected" << endl;
    // получение ошибок, оставшихся в генераторе
    // после предыдущего использования
    // purge error queue
if (!sendCommand(tcp, ""))
    { finalize(); return ; }

    // выключение выхода генератора для того,
    // чтобы не выдавать промежуточное состояние
    // в RF OUT
    // set RF off before parameters change
if (!sendCommand(tcp, ":OUTPUT:STATE OFF\n"))
    { finalize(); return ; }

    // установка внутреннего опорного генератора
    // internal reference oscillator
if (!sendCommand(tcp, ":SOURCE:ROSCillator:SOURce INT\n"))
    { finalize(); return ; }

    // переключение в режим фиксированной центральной частоты
    // fixed center frequency
if (!sendCommand(tcp, ":SOURCE:FREQuency:MODE FIXed\n"))
    { finalize(); return ; }

    // переключение в режим фиксированной выходной мощности
    // fixed output power
if (!sendCommand(tcp, ":SOURCE:POWer:MODE FIXed\n"))
    { finalize(); return ; }

    // переключение фильтра гармоник в автоматический режим
    // harmonics filter is set up automatically
if (!sendCommand(tcp, ":SOURCE:HARMFilter AUTO\n"))
    { finalize(); return ; }

    // установить центральную частоту 137 МГц
    // set center frequency to 137 MHz
if (!sendCommand(tcp, ":SOURCE:FREQuency:CW 137000000.000 Hz\n"))
    { finalize(); return ; }

```

```

// установить выходную мощность +10 дБм
// set output power to +10 dBm
if (!sendCommand(tcp, ":SOURce:POWer:POWer 10.0\n"))
{ finalize(); return ; }

// установить внутренний источник IQ
// set IQ source to internal
if (!sendCommand(tcp, ":SOURce:IQ:SOURce INT\n"))
{ finalize(); return ; }

// установить источник цифрового модулирующего сигнала
// встроенный цифровой baseband-генератор
// set BB waveform source to internal digital modulation,
if (!sendCommand(tcp, ":SOURce:BB:ARBitrary:WAVEform:SOURce
BASE\n"))
{ finalize(); return ; }

// установить тип модуляции BPSK
// set modulation type to BPSK, switch modulation on;
if (!sendCommand(tcp, ":SOURce:BB:DM:FORMat BPSK\n"))
{ finalize(); return ; }

// включить
// встроенный цифровой baseband-генератор
if (!sendCommand(tcp, ":SOURce:BB:DM:STATe ON\n"))
{ finalize(); return ; }

// включить модуляцию
if (!sendCommand(tcp, ":SOURce:MODulation:ALL:STATe ON\n"))
{ finalize(); return ; }

// установить битовую последовательность PRBS23
// в качестве информационного сигнала для модуляторов
// set data source for modulation to PRBS23
if (!sendCommand(tcp, ":SOURce:BB:DM:SOURce PRBS;"
":SOURce:BB:DM:PRBS:LENGth 23\n"))
{ finalize(); return ; }

// установить baseband-фильтр в форме корня из
// приподнятого косинуса
// set baseband filter to root-raised cosine
if (!sendCommand(tcp, ":SOURce:BB:DM:FILTer:TYPE RCOS\n"))
{ finalize(); return ; }

// установить коэффициент скругления baseband-фильтра
// set baseband filter roll-off factor to 0.35
if (!sendCommand(tcp, ":SOURce:DM:FILTer:PARAMeter 0.35\n"))
{ finalize(); return ; }

// установить символьную скорость 400 ксимв/сек
// set symbol rate to 400 kSyms
if (!sendCommand(tcp, ":SOURce:BB:DM:SRATe 400000s\n"))
{ finalize(); return ; }

// установить уровень выходной мощности +10 дБмВт
// set output power to +10 dBm

```

```
    if (!sendCommand(tcp, ":SOURCE:POWER:POWER 10.0\n"))
    { finalize(); return ; }

    // включить выход РЧ генератора
    // turn RF output on
    if (!sendCommand(tcp, ":OUTPUT:STATE ON\n"))
    { finalize(); return ; }

    cout << "Generator state is set up" << endl;
    finalize();
};

// вызывается при изменении 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:
            startTask();
            break ;
    default
        :
            break ;
    }
};

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

// Connect generator
tcp.connectToHost(deviceIP, deviceTcpPort);

return app.exec();
}
```



Микроволновая Электроника

**Радиоэлектронное оборудование
повышенной сложности.**

Разработка и производство

тел.: +7 (495) 137-53-35

e-mail: info@mwel.ru

<http://mwel.ru>