Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)**

Кафедра компьютерных систем в управлении и проектировании (КСУП)

**К ЗАЩИТЕ ДОПУСТИТЬ**

Заведующий кафедрой КСУП

д-р техн. наук, проф.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю.А. Шурыгин

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**МОДУЛЬ РАСЧЕТА ХАРАКТЕРИСТИК ПРИНЦИПИАЛЬНЫХ СХЕМ**

Бакалаврская работа (дипломный проект)

по направлению 09.03.01 – Информатика и вычислительная техника

Пояснительная записка

КСУП. 62 0111 – 01 81 01

Студент гр. 583-1

\_\_\_\_\_\_\_\_\_\_\_\_\_ Т.С. Скрябина

Руководитель

канд. техн. наук, доцент каф. КСУП

\_\_\_\_\_\_\_\_\_\_\_\_\_ А.А. Калентьев

Томск 2017

**Реферат**

Бакалаврская работа (дипломный проект) 63 с., 13 рис., 4 табл., 21 источников, 3 прилож.

ПРИНЦИПИАЛЬНАЯ СХЕМА, САПР, СВЧ, РАСЧЕТ, МЕТОД, МОДУЛЬ, БИБЛИОТЕКА, ЛИНЕЙНЫЙ АНАЛИЗ.

Объектом исследования является разработка программного продукта, реализующего анализ частотных характеристик принципиальных схем.

Цель работы – разработать модуль расчета характеристик принципиальных схем.

В первой главе работы представлен процесс исследования предметной области и сделан обзор существующих СПО для симуляции аналоговых и цифровых схем.

Во второй главе приведено описание функциональности выбранной библиотеки Ngspice.

В третьей части данной работы разбирается процесс интеграции API Ngspice в .NET C# проект.

В четвертой части описаны технические и функциональные возможности программного продукта в виде диаграмм UML.

Пятая глава посвящена тестированию программного продукта.

Результатом работы является программный продукт, осуществляющий анализ частотных характеристик принципиальных схем.

**Abstract**

CIRCUIT DIAGRAM, CAD, MICROWAVE, CALCULATION, METHOD, MODULE, LIBRARY, LINEAR ANALYSIS

An object of study of this work is the development of a software product which realizes the analysis of the frequency characteristics of circuit diagrams.

The purpose of the work is to develop the characteristics of schematic diagrams calculating module.

The first chapter of the paper presents the process of research on the subject domain and provides an overview of existing open source software for simulation of analog and digital circuits.

The second chapter describes the functionality of the selected Ngspice library.

The third part of this paper deals with the integration of the Ngspice API to the .NET C # project.

The fourth part describes the technical and functional capabilities of the software product in the form of UML diagrams.

The fifth chapter is devoted to testing a software product.

The result of the work is a software product that analyzes the frequency characteristics of the circuit diagrams.

Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное

учреждение высшего образования

**ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)**

Кафедра компьютерных систем в управлении и проектировании (КСУП)

УТВЕРЖДАЮ

Заведующий кафедрой КСУП

д-р техн. наук, проф.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю.А.Шурыгин

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_г.

**ЗАДАНИЕ**

на бакалаврскую работу (дипломный проект) студенту Скрябиной

Татьяне Семеновне группа 583-1 факультет ВС

1. Тема работы: Модуль расчета характеристик принципиальных схем (утверждена приказом по вузу от \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ № \_\_\_\_\_\_\_\_\_\_\_\_ )

2. Срок сдачи студентом законченного проекта: 13 июня 2017 г.

3. Назначение и область применения системы:

Разрабатываемый модуль предназначен для расчёта характеристик принципиальных схем, работающих в линейном режиме. Практическая ценность разрабатываемого модуля заключается в возможности его дальнейшего использования в составе коммерческого ПО для автоматизированного проектирования и синтеза принципиальных схем, а также автоматизированного построения моделей элементов

4. Требования к работе:

Разрабатываемый модуль должен быть разработан на языке C#, .NET 4.5, в качестве библиотеки для модульного тестирования необходимо использовать nUnit. В качестве базовой библиотеки для расчёта характеристик модуль должен использовать одну из найденных в процессе НИРС библиотек, основанных на SPICE симуляторе.

5. Перечень вопросов, подлежащих разработке:

Сделать обзор библиотек для анализа схем, выбрать библиотеку на основе которого будет разработан модуль, скомпилировать выбранную библиотеку, сделать маршалинг экспортируемых функций библиотеки, подключить библиотеку к .NET C# проекту, реализация модуля, тестирование.

6. Перечень графического материала (с точным указанием обязательных чертежей):

Слайды презентации, выполненые в Microsoft PowerPoint 2016.

ЗАДАНИЕ СОГЛАСОВАНО:

Консультант по нормам и требованиям ЕСКД

Хабибулина Надежда Юрьевна, канд. техн. наук, доцент каф. КСУП.

Руководитель проектирования

Калентьев Алексей Анатольевич, канд. техн. наук, доцент каф. КСУП.

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г. Подпись\_\_\_\_\_\_\_\_

Задание принято к исполнению

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г. Студент\_\_\_\_\_\_\_\_

подпись

Оглавление

Изм.

Лист

№ докум.

Подпись

Дата

Лист

6

КСУП 62 0111 – 01 81 01

Разраб.

Скрябина

Провер.

Калентьев

Н. Контр.

Хабибулина

Утверд.

Шурыгин

*Модуль расчета характеристик принципиальных схем*

Лит.

Листов

Общее число

ТУСУР, ФВС, каф.КСУП, гр.583-1

[1 Введение 8](#_Toc484723172)

[2 Выбор библиотеки 9](#_Toc484723173)

[2.1 Методы расчета сложных электрических цепей 10](#_Toc484723174)

[2.1.1 Метод токов ветвей 10](#_Toc484723175)

[2.1.2 Метод узловых напряжений 11](#_Toc484723176)

[2.1.3 Метод контурных токов 12](#_Toc484723177)

[2.2 Описание алгоритмов в коммерческих САПР 13](#_Toc484723178)

[2.3 Сравнение существующих библиотек 17](#_Toc484723179)

[3 Описание функциональности выбранной библиотеки 26](#_Toc484723180)

[4 Подключение выбранной библиотеки к проекту .NET С# 29](#_Toc484723181)

[4.1 Методы интеграции машинного кода в .NET C# 29](#_Toc484723182)

[4.1.1 Язык С++/CLI 30](#_Toc484723183)

[4.1.2 Механизм Platform Invoke 33](#_Toc484723184)

[4.2 Интеграция ngspice.dll в .NET C# 43](#_Toc484723185)

[5 Описание технических и функциональных возможностей модуля 48](#_Toc484723186)

[6 Тестирование модуля 50](#_Toc484723187)

[6.1 Функциональные тесты 50](#_Toc484723188)

[6.2 Модульные тесты 53](#_Toc484723189)

[7 Заключение 54](#_Toc484723190)

[Список использованных источников 55](#_Toc484723191)

[Приложение А 57](#_Toc484723192)

[Приложение Б 59](#_Toc484723193)

[Приложение В 64](#_Toc484723194)

|  |  |  |
| --- | --- | --- |
| Диск CD-R  Пояснительная записка к ВКР  Исходный файл «»  Презентация дипломного проекта «»  Отчет о проверке на плагиат  Графический материал: |  | В конверте на обороте обложки |
| Презентация ВКР «»  Твердая копия презентации |  | 13 слайд.  6 экз. |

Изм.

Лист

№ докум.

Подпись

Дата

Лист

7

КСУП 62 0111 – 01 81 01

# Введение

Изм.

Лист

№ докум.

Подпись

Дата

Лист

8

КСУП 62 0111 – 01 81 01

Человечество вступило в XXI век, когда необходимо решать ряд сложных проблем. Определяющая роль в решении проблем отводится информационным технологиям. Среди информационных технологий автоматизация проектирования занимает особое место [1]. Развитие систем автоматизированного проектирования (САПР) – решение научно-технического прогресса общества. САПР опирается на прочную научно техническую базу: современные средства вычислительной техники, новые способы представления и обработки информации, создание новых численных методов решение инженерных задач и оптимизации [2].

Если говорить об устройствах сверхвысоких частот (СВЧ), то это направление, в котором системы автоматизированного проектирования развиваются очень интенсивно. Современным проектировщикам представлен большой выбор программных средств для моделирования [3]. Но на данный момент коммерческие САПР для СВЧ устройств решают широкий спектр задач, которые могут быть ненужными для пользователя. Высокая функциональность влечет высокие цены. Разработка модулей для расчета характеристик устройств с высокой точностью - сложная задача, требующая много времени и затрат. В связи с этим российские производители создают своё программное обеспечение на основе существующих библиотек для анализа принципиальных схем.

Цель данной работы: разработка модуля расчета характеристик принципиальных схем.

Для достижения цели были поставлены следующие задачи:

- сделать обзор библиотек для анализа схем;

- выбрать библиотеку на основе которого будет разработан модуль;

- скомпилировать выбранную библиотеку как динамически подключаемую библиотеку;

- выполнить маршалинг экспортируемых функций;

- подключить скомпирлированную библиотеку с помощью механизма Platform Invoke;

- сделать реализацию модуля;

- провести функциональные и модульные тесты.

Результатом выполнения выпускной квалификационной работы является программный продукт, осуществляющий анализ частотных характеристик принципиальных схем. Схема может состоять из проcтых RLC элементов. Программа рассчитывает комплексное сопротивление заданной цепи, а также выводит график зависимости напряжения на узлах от частоты входного сигнала.

# Выбор библиотеки

Большинство проблем, связанных с анализом схем, решаются в два этапа [4]. Первый этап заключается в составлении уравнений электрического равновесия в форме, позволяющей использовать законы Кирхгофа и характеристики элементов, входящих в схему. Второй этап заключается в решении этих уравнений путем применения подходящих аналитических или численных методов. До появления ЭВМ эти уравнения, как правило, решались аналитическим путем; такой подход накладывал жесткие ограничения на размер и тип схем, которые могли быть подвергнуты анализу. Большие линейные схемы (содержащие, скажем, более 50 элементов) или даже небольшие нелинейные схемы редко поддавались точному анализу. Поэтому инженеры-проектировщики полагались в основном на интуицию и усидчивость, выполняя приближенный анализ таких схем. Во всех случаях процесс анализа завершался макетированием схемы и измерениями представляющих интерес переменных параметров.

Однако даже такой подход, предполагающий макетирование, становится непригодным для анализа интегральных схем, потому что оказывается невозможным дублировать интегральную схему дискретными компонентами. Очевидно, что путем макетирования с помощью дискретных элементов невозможно точно воспроизвести паразитные эффекты и характеристики согласования элементов, входящих в интегральную схему. Невозможно также выполнить с помощью макетирования анализ допусков или анализ наихудшего случая, так как практически невозможно осуществить при макетировании изменение параметров приборов. Вместо имитации схемы посредством макетирования были разработаны САПР для разработки электронных устройств, создания микросхем и печатных плат.

В данной главе будут рассмотрены основные методы расчета, существующие коммерческие САПР и программные библиотеки, позволяющие анализировать электрические цепи.

## Методы расчета сложных электрических цепей

В случае, когда схема электрической цепи достаточно сложна и не приводится к схеме одноконтурной цепи или цепи с двумя узлами, пользуются общими методами расчета [5].

Описываемые методы применимы для цепей постоянного и переменного тока.

### Метод токов ветвей

В данном методе сначала нужно определить направления токов в цепи, а затем написать уравнения, описывающие их отношения друг с другом через законы Кирхгофа и Ома [6]. Число уравнений равно количеству неизвестных токов ветвей и определяется как

NМТВ = NB – NJ, (2.1)

где NB – число ветвей электрической цепи;

NJ – число ветвей с источниками тока.

Количество уравнений, составляемых по Закону токов Кирхгофа (ЗТК) равно

N1 = NУЗЛОВ – 1, (2.2)

где NУЗЛОВ – количество узлов электрической цепи.

Количество уравнений, составляемых по Закону напряжений Кирхгофа (ЗНК) равно

N2 = NМТВ – N1. (2.3)

При составлении уравнений по II Закону Кирхгофа следует выбирать независимые контуры, не содержащие источников тока. Как только будут получены уравнения для каждого из неизвестных токов, можно будет решить систему уравнений, рассчитав тем самым все токи, а затем и все напряжения в цепи.

Последовательность расчета:

1. проводится топологический анализ схемы:
   1. обозначаются токи во всех ветвях, произвольно выбираются их положительное направление и обозначаются на схеме стрелками;
   2. подсчитывается общее число узлов и определяется число независимых узлов, и обозначаются на схеме;
   3. подсчитывается число независимых контуров и обозначаются на схеме дугой;
2. по ЗТК для независимых узлов и по ЗНК для независимых контуров относительно токов ветвей записывают уравнения. После привидения подобных членов они сводятся к системе линейных алгебраический уравнений (ЛАУ);
3. Решив систему уравнений относительно токов по методу Крамера находят токи во всех ветвях схемы. Если значения токов отрицательные, то действительные направления их будут противоположны первоначально выбранным направлениям.

### Метод узловых напряжений

Метод заключается в том, что на основании ЗТК определяются потенциалы в узлах электрической цепи относительно некоторого базисного узла. Эти разницы потенциалов называются узловыми напряжениями, причем положительное направление их указывается стрелкой от рассматриваемого узла к базисному.

Напряжение на какой-либо ветви равно разности узловых напряжений концов данной ветви; произведение же этого напряжения на комплексную проводимость данной ветви равно току в этой ветви. Таким образом, зная узловые напряжения в электрической цепи, можно найти токи в ветвях.

Если принять потенциал базовой ветви равным нулю, то напряжения между остальными узлами и базисным узлом будут равны также потенциалам этих узлов. Поэтому данный метод называется также методом узловых потенциалов (МУП).

Если электрическая схема содержит q узлов, то на основании первого закона Кирхгофа получается система из q-1 уравнений.

Порядок расчета:

1. обозначить все токи ветвей и их положительное направление;
2. произвольно выбрать опорный узел и пронумеровать все остальные узлы;
3. определить собственные и общие проводимости узлов, а также узловые токи, т.е. рассчитать коэффициенты в системе уравнений;
4. записывается система уравнений по ЗТК;
5. полученную систему уравнений решить относительно неизвестных потенциалов при помощи метода Крамера;
6. с помощью обобщенного закона Ома рассчитать неизвестные токи;
7. проверить баланс мощности.

### Метод контурных токов

Метод контурных токов (МКТ) при расчете токов цепи позволяет уменьшить количество уравнений, составляемых по ЗНК:

NМКТ = N2 = NМТВ – N1 (1.4)

Ток в любой ветви цепи модно представить в виде алгебраической суммы контурных токов, протекающих по этой ветви. Выбирают и обозначают известные и неизвестные контурные токи. Известные контурные токи можно считать совпадающими с соответствующими токами источников тока. Неизвестные контурные токи определяются по ЗНК и для них составляется система уравнений метода контурных токов. Решив данную систему находятся неизвестные контурные токи. И находят токи ветвей как сумма контурных токов, протекающих в этой ветви.

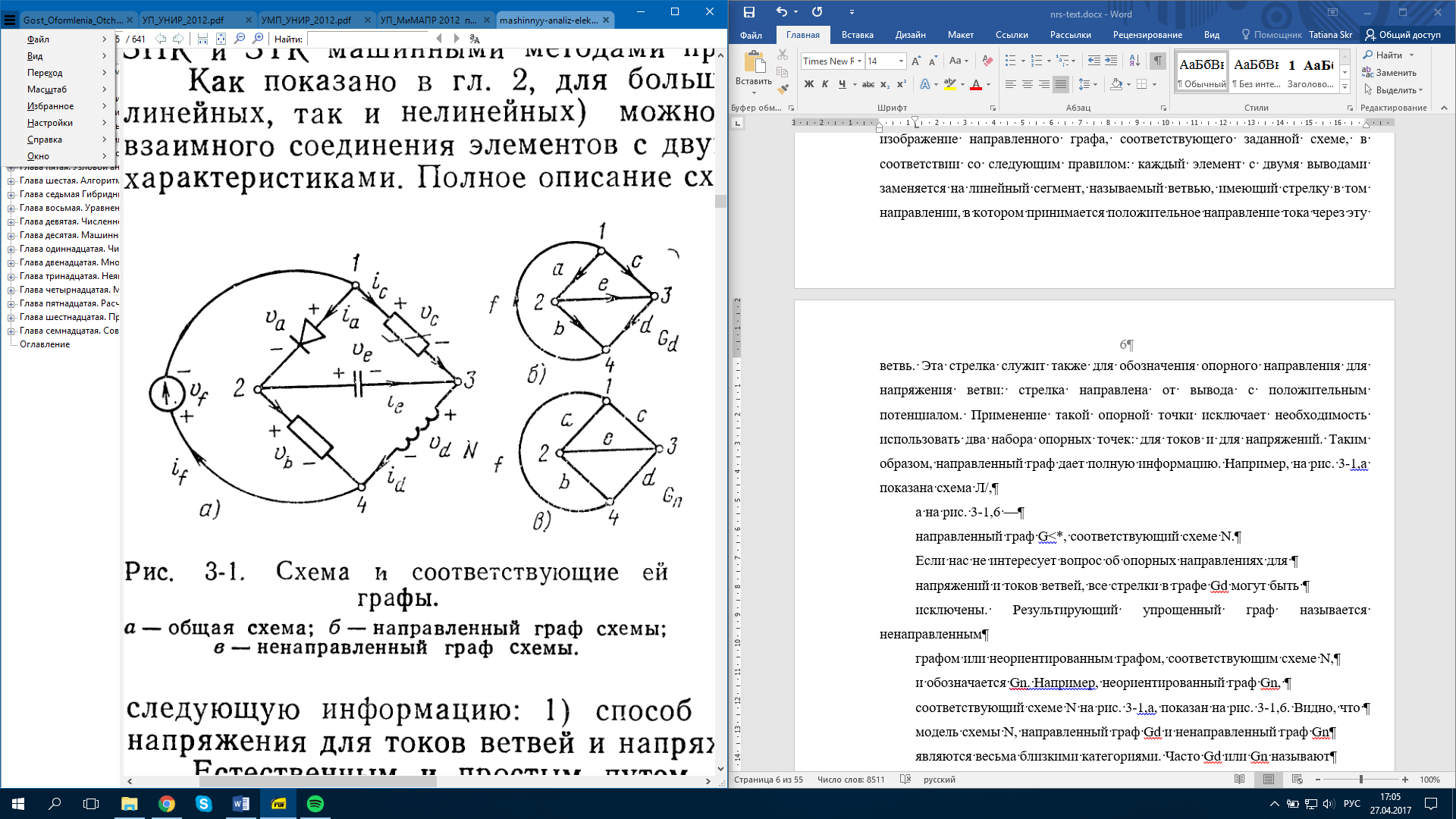
Последовательность расчета:

1. топологический анализ схемы:
   1. определяют число ветвей;
   2. определяют число узлов;
   3. посчитывают число независимых контуров;
2. все независимые контуры показывают дугой со стрелкой на них, которая показывает положительное направление обхода контура. Все контуры нумеруют и контуру присваивают свой контурный ток. За положительное направление контура принимают положительное направление обхода контура;
3. по ЗНК относительно контурных токов записывают уравнения, которые после приведения подобных членов образуют системы линейных уравнений;
4. по правилу Крамера находят контурные токи;
5. токи в ветвях находят как алгебраическую сумму контурных токов, протекающих через данную ветвь. В алгебраической сумме контурные токи берутся с положительным знаком, если ток ветви совпадает с контурным и отрицательным, если не совпадает. Если токи ветви оказались положительными, то выбранное направление тока совпадает с истинным и наоборот.

## Описание алгоритмов в коммерческих САПР

Любая схема подчиняется трем основным законам: ЗНК, ЗТК и закону элементов (характеристики ветви) [5]. Топология схем рассматривает такие свойства сложных схем, которые связаны только с соединениями ветвей. Она является одним из направлений математики, называемой теорией графов.

Естественным и простым путем для описания информации является изображение направленного графа, соответствующего заданной схеме, в соответствии со следующим правилом: каждый элемент с двумя выводами заменяется на линейный сегмент, называемый ветвью, имеющий стрелку в том направлении, в котором принимается положительное направление тока через эту ветвь. Эта стрелка служит также для обозначения опорного направления для напряжения ветви: стрелка направлена от вывода с положительным потенциалом. Применение такой опорной точки исключает необходимость использовать два набора опорных точек: для токов и для напряжений. Таким образом, направленный граф дает полную информацию. Например, на рисунке 2.1 показана схема N, а на рисунке 2.2 – направленный граф G, соответствующий схеме N.

Рисунок 2.1 – Схема N

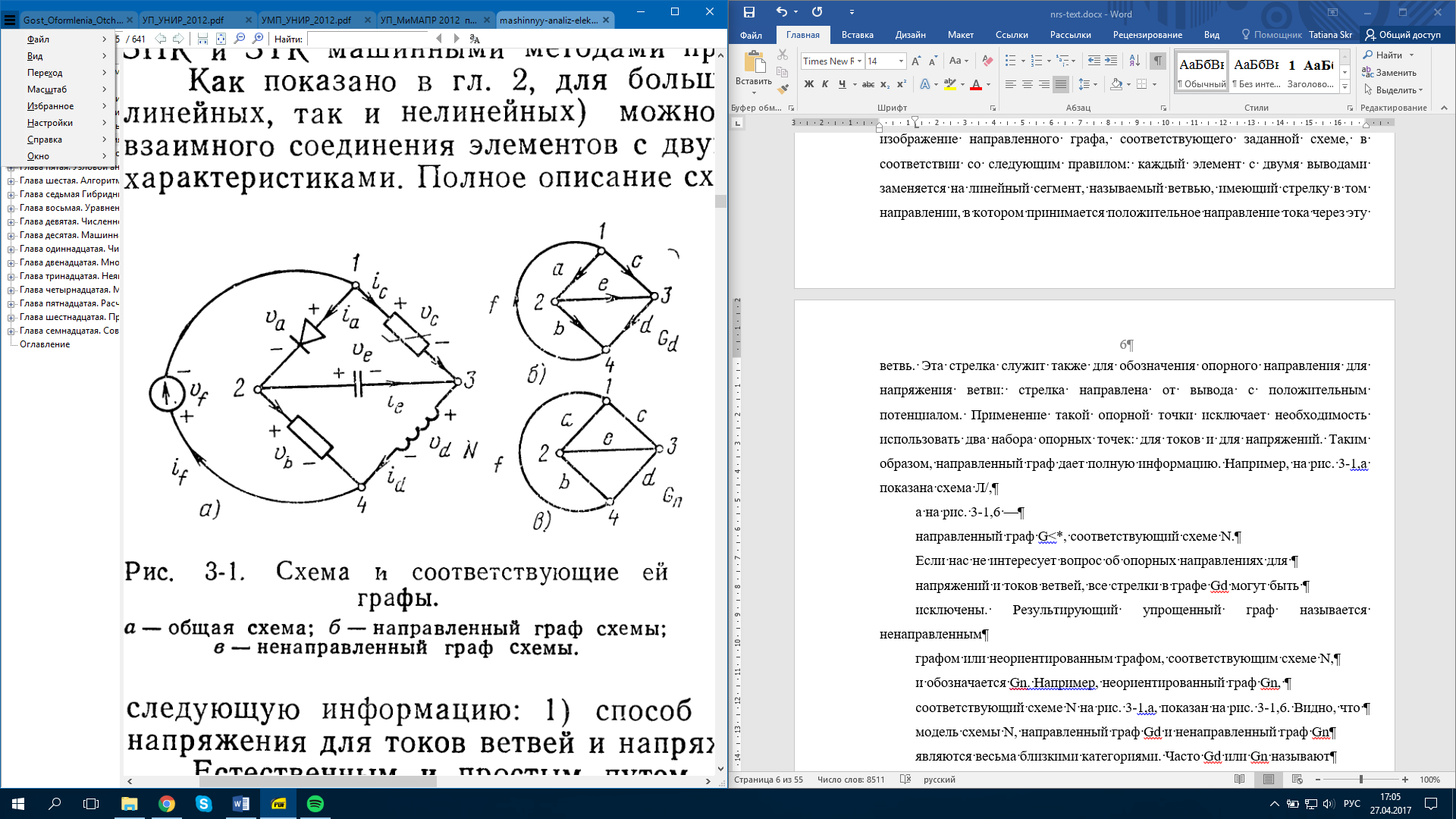


Рисунок 2.2 – Граф G

Хотя направленный граф G полностью описывается соединениями и опорными направлениями ветвей сложной схемы, эта форма выражения неудобна для записи в памяти ЭВМ. Для записи в цифровом виде наиболее удобно представление в виде матриц. Содержащаяся в направленном графе информация может быть полностью представлена матрицей, называемой матрицей инциденций.

Матрица инциденций (инцидентности) – это одна из форм представления графа, в которой указываются связи между инцидентными элементами графа (ребро(дуга) и вершина). Столбцы матрицы соответствуют ребрам, строки — вершинам.

Для того чтобы компактно выразить уравнения ЗНК в виде единственного матричного уравнения, необходимо ввести матрицу, называемую матрицей контуров (матрицей схемы), соответствующую направленному графу.

Чтобы выразить уравнения обобщенного ЗТК в виде единственного матричного уравнения, введем матрицу, называемую матрицей сечений (применительно к направленному графу). Законы Кирхгофа для тока и напряжения накладывают определенные условия на токи и напряжения ветвей в схеме. Вследствие этого лишь небольшая часть токов и напряжений ветвей независимы, а остальные переменные ветвей могут быть выражены через независимые величины.

Независимая система ЗНК и ЗТК уравнений для любого схемного графа может быть получена автоматически с помощью ЭВМ в виде топологических матриц. Результирующие уравнения относятся только к соединениям ветвей в схеме и содержат удвоенное количество переменных относительно количества независимых уравнений. Остальные уравнения можно получить из характеристик элементов. В этой работе рассматриваются только линейные схемы, содержащие RLC-элементы, независимые источники напряжения или тока и управляемые напряжением источники тока.

Для анализа линейных электрических цепей в САПР используют модифицированный МУП. Были рассмотрены следующие САПР: Microwave Office [7] (National Instruments Applied Wave Research (NI AWR), США) и Advanced Design System [8] (Keysight Technologies, США).

Идея модифицированного МУП заключается в разбиении элементов цепи (ветвей) на группы [9]:

1. ветви, которые можно описать через проводимости (ток через них не будет определен);
2. ветви, которые нельзя описать через проводимости, либо те элементы, ток в которых необходимо определить;
3. ветви независимых источников тока.

В результате решения будут найдены напряжения узлов Un и токи ветвей второй группы I2. Напряжение ветвей можно определить позже по уравнении связи напряжений ветвей и узлов Ub = At∙Un, а токи ветвей первой группы – на основании компонентных уравнений I1 = Y1∙U1. При расчете во временной области начальные токи в катушках и напряжения на конденсаторах учитываются с помощью эквивалентных источников, следующих из преобразования Лапласа.

Матричная форма уравнения модифицированного метода узловых потенциалов имеет вид:

(2.5)

YR – сокращенная подматрица узловых проводимостей, не учитывающая управляемые током элементы, G – подвектор независимых источников тока, подматрица B содержит частные производные от полученных по закону Кирхгофа для токов уравнений по дополнительным переменным. Вольтамперные связи, дифференцированные по дополнительным переменным, представлены подматрицами C и D. Подвектор F содержит вклад реактивных элементов в полный вектор независимых источников тока. Реактивные элементы рассматриваются только во временной области с учетом их конечно-разностного представления.

Таким образом, модифицированная узловая система представляет собой обычную узловую матрицу, построенную из ветвей первой группы, дополненную по принципу модифицированных табличных уравнений уравнениями ветвей второй группы. Вектор свободных членов соответственно представляет собой вектор эквивалентных узловых источников тока, дополненный источниками напряжениями ветвей второй группы. Вектор неизвестных содержит узловые напряжения и токи ветвей второй группы.

Реализация модифицированного метода узловых потенциалов достаточно проста и состоит в анализе признака ветви и внесении определенных коэффициентов либо в узловую матрицу, либо в ее дополнение. Источники тока вносятся в первую часть вектора свободных членов, а источники напряжения – во вторую.

В Microwave Office интегрированы библиотеки HSPICE (Synopsys), APLAC (NI AWR) и Spectre (Cadence). В Advanced Design System за линейный анализ отвечает элемент линейного моделирования – W2306EP Linear Simulator.

## Сравнение существующих библиотек

В ходе данной работы были рассмотрены следующие симуляторы электронных схем: SPICE, SIMetrix/SIMPLIS, Ngspice, LTspice, TINA TI, Qucs, HSPICE. Ниже приведены их описания.

**SPICE** (Simulation Program with Integrated Circuit Emphasis) — симулятор электронных схем общего назначения с открытым исходным кодом. Является мощной программой, используемой в разработке как интегральных схем, так и печатных плат для проверки целостности схемы и для анализа её поведения [10].

SPICE был разработан в Electronics Research Laboratory в Калифорнийском университете в Беркли Лоуренсом Нагелем. SPICE стала индустриальным стандартом симуляции электрических схем. SPICE послужил основой для разработки множества других программ симуляции схем, как в академической, так и в промышленной среде.

Первая коммерческая версия — ISPICE, (National CSS). Наиболее выдающиеся коммерческие версии SPICE: HSPICE (изначально Meta Software, ныне Synopsys) и PSPICE (ныне Cadence Design Systems). Академические версии программы: XSPICE (Georgia Tech, с поддержкой смешанных аналого-цифровых моделей) и Cider (ранее CODECS, UC Berkeley и Oregon State Univ.; с поддержкой полупроводниковых устройств). Индустрия проектирования интегральных схем достаточно рано начала пользоваться SPICE, и, до развития коммерческих реализаций, многие компании-разработчики микросхем имели собственные версии SPICE. В настоящее время крупные производители микросхем развивают собственные программы симулирования на базе SPICE. Например, ADICE у компании Analog Devices, LTspice у Linear Technology, Mica у Freescale Semiconductor, TISPICE у Texas Instruments.

Впервые SPICE1 был представлен на конференции в 1973. Программа была написана на языке FORTRAN и использовала анализ цепи методом узловых потенциалов для построения уравнений схемы. Метод узловых потенциалов имел ограничения в работе с индуктивностями, источниками переменного напряжения и с различными вариантами управляемых генераторов тока и напряжения. В SPICE1 было доступно небольшое количество элементов, программа использовала анализ переходных процессов (transient analysis) с фиксированным шагом по времени.

Популярность пришла ко второй версии программы, SPICE2, в 1975 году. Она тоже была написана на FORTRAN, но имела больше элементов, позволяла изменять шаг по времени при анализе переходных процессов, уравнения цепей формулировались при помощи модифицированного метода узловых потенциалов (modified nodal analysis), устраняя тем самым ограничения метода узловых потенциалов. Последняя версия SPICE, написанная на языке FORTRAN, - 2G.6 (1983 год). Следующую версию, SPICE3 разработал Томас Кворлс (Thomas Quarles) в 1989. Она написана на языке Си, использует тот же формат данных об электрических цепях (netlist) и поддерживает визуализацию в среде X Window System.

В 2011 году появление SPICE было отнесено к IEEE Milestone, было отмечено, что SPICE и его производные стали неотъемлемой частью разработки практически любой интегральной схемы.

SPICE стал популярен, поскольку поддерживал анализ и содержал модели, необходимые для разработки интегральных схем того времени, и при этом был достаточно быстрым для практического использования. Предшественники SPICE часто имели лишь одно предназначение, например, BIAS позволял рассчитывать режимы биполярных транзисторов; SLIC производил анализ малых сигналов. SPICE комбинировал в себе несколько режимов анализа и достаточно обширную библиотеку моделей устройств.

SPICE2 включает в себя:

- AC анализ (анализ по переменному току);

- DC анализ (анализ по постоянному току) для слабых сигналов;

- анализ DC transfer curve;

- анализ шумов;

- анализ передаточной функции (входное и выходное усиление малых сигналов и вычисление импеданса);

- анализ переходных процессов.

Все SPICE симуляторы работают с текстовым файлом списка соединений (netlist). Список соединений сдержит перечень схемных элементов, узлов, с которыми эти элементы связаны, определения моделей и различные SPICE директивы.

Далее представлено очень краткое описание SPICE языка:

\* КОММЕНТАРИЙ

[ТИП\_ИМЯ\_КОМПОНЕНТА] [УЗЕЛ1] .. [УЗЕЛ\_N] [ПАРАМЕТР\_1] ... [ПАРАМЕТР\_N]

.END ставится в конце файла

Типы компонентов:

Rxx resistor резистор

Lxx inductor катушка индуктивности

Cxx capacitor конденсатор

Vxx voltage source источник напряжения

Ixx current source источник тока

Exx voltage controlled voltage source

Hxx current controlled voltage source

Fxx current controlled current source

Gxx voltage controlled current source

Dxx diode диод

Jxx junction FET JFET транзистор

Mxx MOSFET МОС транзистор

Bxx GaAsFET транзистор на арсениде галлия GaAs

Qxx BJT биполярный транзистор

**SIMetrix/SIMPLIS** – уникальный симулятор, созданный для высокоскоростного анализа линейных и смешанных цепей [11].

Приложение SIMPLIS (аббревиатура от «SIMulation for Piecewise LInear System») было создано с целью быстрого моделирования источников питания и импульсных стабилизаторов, а также исследования ключевых схем преобразователей напряжения. Оно также, как и программа SPICE, функционирует на уровне компонентов радиосхемы, однако анализы переходных процессов выполняет в 10-50 раз быстрее. Подобной скорости моделирования модуль SIMPLIS достигает благодаря применению кусочно-линейной аппроксимации взамен решения систем нелинейных уравнений, используемых в программе SPICE. Приложение выполняет следующие виды анализов: переходных процессов, ФЧХ/АЧХ с замкнутой петлей авторегулирования и в рабочей точке. Последний анализ представляет собой ускоренный расчет установившегося режима работы импульсного блока питания при постоянных нагрузке и питании. Для особо сложных в плане симуляции случаев предусмотрен еще один режим анализа – Multi-Tone AC, являющийся подобием двухчастотного метода определения интермодуляционных искажений. Кроме того, в программе имеется функция автоматического преобразования SPICE-моделей в модели формата SIMPLIS.

Вторая программа SIMetrix является полноценным симулятором аналогово-цифровых схем. В ее основе лежат два распространенных продукта – XSPICE и SPIСE. Приложение имеет схемный редактор, симулятор и графический постпроцессор. В качестве основного метода итерации применяется решение Ньютона-Рэфсона, обеспечивающего быстроту расчетов режимов переходных процессов и по постоянному току. Для особо «каверзных» схем имеется алгоритм псевдопереходного анализа с изоляцией/фиксацией не вычисляемых узловых потенциалов и адаптируемым шагом итерации. Программа SIMetrix выполняет поддержку моделей элементов спецификации MOS9, BSIM3/4, Mextram, VBIC и совместима с файлами HSPICE. Кроме того, приложение имеет пробник Бодэ и калькулятор КПД, а также поддерживает создание произвольного логического блока, имитирующего работу цифрового устройства.

Программы, входящие в данный комплект, были разработаны благодаря усилиям двух компаний-партнеров – SIMetrix Technologies (Великобритания, графство Беркшир, город Тэтчем) и SIMPLIS Technologies (США, штат Орегон, город Портленд).

**Ngspice** – симулятор электронных схем общего назначения с открытым исходным кодом, обеспечивающий моделирование в режиме смешанных сигналов и на смешанном уровне [12]. Является расширением Spice3f5, включившим в себя код проекта Cider, обеспечивающий моделирование на смешанном уровне, и проекта XSPICE, обеспечивающий моделирование смешанных сигналов. Является мощной программой, используемой в разработке как интегральных схем, так и печатных плат для проверки целостности схемы и для анализа её поведения.

**LTspice** (он же SwitcherCAD) представляет собой универсальную среду для проектирования и создания электрических схем с интегрированным симулятором смешанного моделирования [11]. Программа позволяет быстро менять компоненты и параметры электронных схем, испытывать работоспособность новых вариантов, находить оптимальные решения. Возможна загрузка списка соединений, сгенерированного другими инструментами для рисования схем или созданного вручную. От аналогичных программ (Microcap, OrCAD) рассматриваемое ПО отличается малым объемом необходимого дискового пространства и более высокой скоростью моделирования процессов.

LTspice содержит полную библиотеку компонентов компании Linear Technology Corporation (пассивные элементы и интегральные схемы, включая редкие модели импульсных контроллеров и регуляторов). Поскольку программа использует стандартные SPICE-модели электронных деталей, к имеющейся базе можно добавлять библиотеки сторонних производителей, а также создавать свои собственные модели.

**TINA-TI** представляет собой обычный SPICE-симулятор с простым, интуитивно понятным графическим интерфейсом, позволяющим освоить программу в кратчайшие сроки [11]. Все компоненты, представленные в TINA-TI, распределены по шести группам: основные пассивные радиодетали, ключи, полупроводники, измерительные приборы, макромодели сложных устройств и источники. Также данная программа включает в себя несколько десятков разнообразных примеров.

TINA-TI предоставляет широкие возможности по рисованию и редактированию электронных схем. Имеются следующие виды анализов: по постоянному и переменному току (сюда входит: вычисление узловых напряжений, создание таблицы результатов, построение переходных характеристик и температурный анализ), переходных процессов, шумов, преобразование Фурье и некоторые другие. В зависимости от вида выполняемого анализа программа генерирует результаты в виде графиков или таблиц. В программе TINA-TI также доступны возможности тестирования и измерения сигналов. Для этого существуют следующие виртуальные приборы: осциллограф, анализатор сигналов, цифровой тестер (с измерителем частоты), генератор функций и записывающее устройство. Виртуальные приборы программного комплекса максимально приближены по использованию к реальным устройствам. «Подключить» их можно к любой точке рассматриваемой схемы. Вся информация, снятая виртуальными приборами, может сохраняться в памяти компьютера.

**Qucs** – свободная программа, предназначенная для моделирования электронных цепей. Распространяется по лицензии GPL. Позволяет моделировать электронную аппаратуру в режиме малого и большого сигнала, а также шумовые характеристики. Пользовательский интерфейс — графический. Цифровая аппаратура моделируется с использованием VHDL и/или Verilog [13].

Отличительной особенность данного симулятора является расширенные функции для моделирования схем СВЧ-устройств: моделирование S-параметров, специальные модели СВЧ-компонентов и постпроцессор с расширенными функциями анализа комплексных сопротивлений в частотной области. По возможностям анализа СВЧ-устройств Qucs приближается к таким проеприетарным аналогам как AWR MicrowaveOffice. Но симулятор Qucsator несовместим со SPICE и поддерживает SPICE только через слой совместимости, что значительно осложняет использование существующих библиотек моделей электронных компонентов. Несмотря на отличные характеристики движка Qucsator для моделирования СВЧ-схем в частотной, во временной области имеются многочисленные ошибки.

**HSPICE** – система моделирования, а также сами модели, основанные на SPICE [14]. Изначально был создан Ashawna Hailey, позднее был куплен фирмой Synopsys. В настоящее время HSPICE является стандартом де-факто ("золотой" стандарт индустрии) для производителей микросхем, т.к. обеспечивает максимально правдоподобное моделирование узлов на транзисторном уровне. В настоящее время пакет сильно расширен за счет внедрения возможностей анализа в частотной области, анализа кабелей и полосковых линий, а также возможности импорта результата моделирования в других САПР (например, импорт S-параметров, с последующим обратным преобразованием Фурье). Стоит отметить, что эти возможности уже не могут обеспечить точности, которую дают специализированные САПР (например, HFSS (AnSys), или ADS (Agilent Techologies) для анализа кабелей, или печатных проводников), но позволяют качественно оценить работу системы.

В таблице 2.1 представлено сравнение существующих библиотек для анализа принципиальных схем СВЧ диапазона.

Таблица 2.1 – Сравнительная таблица

|  |  |  |
| --- | --- | --- |
| Библиотека | Лицензия | Анализ |
| SPICE | BSD | По переменному току, по постоянному току, передаточной функции, шумов, переходных процессов |
| SIMetrix/ SIMPLIS | платная | Переходного процесса, периодический анализ рабочих точек, по переменному току |
| Ngspice | BSD | Нелинейный по постоянному току, нелинейный переходных процессов, линейный по переменному току |

|  |  |  |
| --- | --- | --- |
|  | | |
|  |  |  |
| LTSpice | freeware | Переходных процессов, по переменному току, шумов и по постоянному току, преобразование Фурье |
| TINA | freeware | По переменному току, переходных процессов, по постоянному току, преобразование Фурье |
|  | | |
|  |  |  |
| qucsator | GPL | По постоянному току, по переменному току, гармонический баланс, по цифровому сигналу, переходных процессов, S-параметров, развертка по параметру, оптимизация |
| HSPICE | платная  от $1000 в год (учебная версия) | По переменному току, по постоянному току, по цифровому сигналу, переходных процессов, спектральный, линейный, статистический, Монте-Карло, оптимизация |

Информация по типам лицензий представлена в приложении А.

**Выводы**

- возможности бесплатных библиотеки не сильно уступают платным;

- qucsator имеет расширенные функции для анализа СВЧ-устройств. Однако он несовместим со SPICE и поддерживает SPICE только через конвертер, что значительно осложняет использование существующих библиотек моделей электронных компонентов;

- библиотеки SPICE и Ngspice являются хорошим вариантом для разработки своего программного продукта.

# Описание функциональности выбранной библиотеки

Исходный код API Ngspice содержится в файле sharedspice.c и в соответствующем файле заголовка sharedspice.h. Все объявления типов и функций содержатся в файле заголовка, полный листинг которого приведен в приложении Б. В таблице 3.1 приведены названия структур и функций и их краткие описания.

Таблица 3.1 – Описание структур и функций библиотеки Ngspice

|  |  |
| --- | --- |
| Имя | Описание |
| vector\_info | Структура, которая хранит имя вектора, тип, действительные или комплексные данные, которые передаются во время или после симуляции. |
| vecinfo  vecinfoall | Структуры, используемые функцией обратного вызова SendInitData. |
| vecvalues  vecvaluesall | Структуры, используемые функцией обратного вызова SendData. |
| int ngSpice\_Init(..) | Функция для инициализации моделирования. |
| int ngSpice\_Init\_Sync(..) | Функция для инициализации параллельных потоков моделирования. |
| int ngSpice\_Command(..) | Отправляет команды вызывающей сущности в ngspile.dll |
| bool ngSpice\_running() | Проверяет запущен ли ngspice в потоке. |
| pvector\_info ngGet\_Vec\_Info(..) | Использует имя вектора в качестве параметра и возвращает указатель на структуру vector\_info. Затем вызывающая сущность может непосредственно изменять векторные данные (но лучше не изменять их). |
| int ngSpice\_Circ(..) | Функция для загрузки схемы с помощью массива строки. |
| char\* ngSpice\_CurPlot() | Возвращает вызывающей сущности указатель на имя текущего графика. |
| char\*\* ngSpice\_AllPlots() | Возвращает вызывающей сущности указатель на массив всех графиков. |
| char\*\* ngSpice\_AllVecs() | Возвращает вызывающей сущности указатель на массив имен векторов. |

|  |  |
| --- | --- |
| Продолжение таблицы 3.1 | |
| Имя | Описание |
| int (SendChar)(..) | Отправляет выходные данные со стандартного потока вызывающей сущности. |
| int (SendStat)(..) | Отправляет статус моделирования вызывающей сущности. |
| int (ControlledExit)(..) | Отправляет информацию о контролируемом выходе из моделирования. |
| int (SendData)(..) | Отправляет данные. |
| int (SendInitData)(..) | Отправляет данные о векторе инициализации. |
| int (BGThreadRunnig)(..) | Указывает работает ли фоновый поток. |

Входные данные для библиотеки можно загрузить из файла требуемого формата, либо загружать построчно с помощью команды “circbyline …” пользуясь при этом функцией ngSpice\_Command(“circbyline …”), либо записать в массив строки circarray и воспользоваться функцией ngSpice\_Circ(circarray). Входной файл должен иметь расширение \*.cir и должен следовать SPICE формату:

TITLE

ELEMENT DESCRIPTIONS

.MODEL STATEMENTS

ANALYSIS COMMANDS

OUTPUT COMMANDS

.END

Формат описания элементов схемы:

<letter><name> <n1><n2> …[mname] [parvals],

где <letter> – строка, указывающая тип элемента(копмонента);

<name> – уникальный порядковый номер компоненты;

<ni> – номер узла;

[mname] – имя модели;

[parvals] – некоторые опциональные значения параметров.

После того, как моделирование было закончено, воспользовавшись командами write или wrdata данные можно записать в файл. А можно вызвать команду print для получения данных через функцию обратного вызова SendChar, одним из параметров которого является указатель на массив символов.

**Выводы**

API Ngspice позволяет использовать весь функционал, который имеет данный симулятор электронных схем. Работа с входными и выходными данными осуществляется с помощью символьных типов данных.

# Подключение выбранной библиотеки к проекту .NET С#

На сегодняшний день стала очень популярна платформа .Net Framework и язык C#. На ней пишутся приложения, игры, инструменты [15]. Язык программирования C# является наиболее гибким языком, реализующий быструю и удобную работу с формами. Однако существует много кода, написанного на Cи или C++, который часто нужно использовать с C# проектах.

Это могут быть математические библиотеки, код оставшийся от прошлых проектов или другой проект, для которого нужно сделать графический интерфейс.

В данной главе рассмотрены методы, как интегрировать машинный код в C# проект и успешно его развивать. В качестве интегрированной среды разработки используется Microsoft Visual Studio.

## Методы интеграции машинного кода в .NET C#

Для начала необходимо определить понятия управляемого и неуправляемого кода. Управляемый код (англ. managed code) – термин, введённый фирмой Microsoft, для обозначения кода программы, исполняемой под «управлением» виртуальной машины .NET – Common Language Runtime (CLR) или Mono. При этом машинный код называется неуправляемым кодом (англ. unmanaged code) [16]. В общем случае при написании С#-программы создается управляемый код.

Основная часть выбранного симулятора электронных схем – Ngspice реализована на языке Си, в том числе и динамически подключаемая библиотека, которая является интерфейсом прикладного программирования (API) симулятора. Соответственно, API у Ngspice является неуправляемым кодом.

Существуют два способа интеграции неуправляемого кода в C# проект.

1. Сделать обертку на языке C++/CLI.
2. Использовать механизм Platform Invoke.

### Язык С++/CLI

C++/CLI – промежуточный язык для склеивания кода на C/C++ и .NET.

Это идеальный язык для генерации связующего кода между управляемым и неуправляемыми средами исполнения, поскольку он позволяет генерировать код для обоих сред и генерирует код перехода, избавляя разработчиков от необходимости склеивать что-то вручную [17]. Суффикс CLI – обозначает то, что язык реализует спецификацию Common Language Infrastructure, т.е. является полноправным членом семейства языков платформы .NET.

Есть множество ситуаций, требующих написания обёртки над библиотекой. Можно создавать обёртку для библиотеки, чей код можно изменять [18]. Можно оборачивать часть Win32 API, обёртка для которой отсутствует в Framework Class Library (FCL). Возможно, создаётся обёртка для сторонней библиотеки. Библиотека может быть как статической, так и динамической (DLL). Более того, это может быть как C, так и C++ библиотека. Этот раздел содержит практические рекомендации, общие советы и решения для нескольких конкретных проблем.

Одной из основных целей при создании .NET было обеспечение взаимодействия разных языков. Если вы создаёте обёртку для нативной библиотеки, то возможность взаимодействия с разными языками приобретает особенную важность, поскольку разработчики, использующие библиотеку, скорее всего, будут использовать C# или другие языки .NET. Common Language Infrastructure (CLI) – это основа спецификации. Важной частью этой спецификации является Common Type System (CTS). Хотя все языки .NET основываются на общей системе типов, не все языки поддерживают все возможности этой системы.

Чтобы чётко определить, смогут ли языки взаимодействовать между собой, CLI содержит Common Language Specification (CLS). CLS — это контракт между разработчиками, использующими языки .NET, и разработчиками библиотек, которые можно использовать из разных языков. CLS задаёт минимальный набор возможностей, который обязан поддерживать каждый язык .NET. Чтобы библиотеку можно было использовать из любого языка .NET, соответствующего CLS, возможности языка, используемые в публичном интерфейсе библиотеки, должны быть ограничены возможностями CLS. Под публичным интерфейсом библиотеки понимаются все типы с видимостью public, определённые в сборке, и все члены таких типов с видимостью public, public protected или protected.

Можно использовать атрибут CLSCompliantAttribute, чтобы обозначить тип или его член как соответствующий CLS. По умолчанию, не помеченные этим атрибутом типы считаются не соответствующими CLS. Если вы примените этот атрибут на уровне сборки, то по умолчанию все типы будут считаться соответствующими CLS. Следующий пример показывает, как применять этот атрибут к сборкам и типам:

[assembly: CLSCompliant(true)]; // типы с видимостью public соответствуют CLS, если не указано иное

namespace ManagedWrapper

{

public ref class SampleCipher sealed

// ...

};

[CLSCompliant(false)] // этот класс явно помечен как не соответствующий CLS

public ref class AnotherCipherAlgorithm sealed

{

// ...

};

}

Согласно правилу 11 CLS все типы, присутствующие в сигнатурах членов класса (методов, свойств, полей и событий), видимых снаружи сборки, должны соответствовать CLS. Чтобы правильно применять атрибут [CLSCompliant], нужно знать, соответствуют ли типы параметров метода CLS. Чтобы определить соответствие CLS, надо проверить атрибуты сборки, в которой объявлен тип, а также атрибуты самого типа.

В FCL также используется атрибут CLSCompliant. mscorlib и большинство других библиотек FCL применяют атрибут [CLSCompliant(true)] на уровне сборки и помечают типы, не соответствующие CLS, атрибутом [CLSCompliant(false)].

Также нужно учесть, что следующие примитивные типы в mscorlib помечены как несоответствующие CLS: System::SByte, System::UInt16, System::UInt32 и System::UInt64. Эти типы (или эквивалентные им имена типов char, unsigned short, unsigned int, unsigned long и unsigned long long в C++) нельзя использовать в сигнатурах членов типов, которые считаются соответствующими CLS.

Если тип считается соответствующим CLS, то все его члены также считаются таковыми, если явно не указано обратного. Пример:

using namespace System;

[assembly: CLSCompliant(true)]; // типы с видимостью public соответствуют CLS, если не указано иное

namespace ManagedWrapper

{

public ref class SampleCipher sealed // SampleCipher соответствует CLS из-за атрибута сборки

{

public:

void M1(int);

// M2 помечен как несоответствующий CLS, потому что тип одного из его аргументов не соответствует CLS

[CLSCompliant(false)]

void M2(unsigned int);

};

}

К сожалению, компилятор C++/CLI не показывает предупреждений, когда тип, помеченный как соответствующий CLS, нарушает правила CLS. Чтобы понять, помечать тип как соответствующий CLS или нет, надо знать следующие важные правила CLS:

- имена типов и их членов должны быть различимы в языках с регистро-независимыми идентификаторами (правило 4);

- глобальные статические (static) поля и методы не совместимы с CLS (правило 36);

- пользовательские атрибуты должны содержать поля только следующих типов: System::Type, System::String, System::Char, System::Boolean, System::Int[16|32|64], System::Single и System::Double (правило 34);

- объекты исключений должны иметь тип System::Exception или унаследованный от него тип (правило 40);

- все аксессоры свойств должны быть либо виртуальными, либо не виртуальными одновременно, то есть не допускается смешение виртуальных и не виртуальных аксессоров (правило 26);

- упакованные значения не соответствуют CLS (правило 3);

- неуправляемые указатели не соответствуют CLS (правило 17). Под это правило также подпадают и ссылки в C++. Доступ к нативным классам, структурам и объединениям также осуществляется по указателю. Из этого следует, что данные нативные типы также не соответствуют CLS.

Прежде чем создавать обёртку для некоторой библиотеки, сначала нужно кбедиться, что FCL не содержит готового класса с требуемыми методами. FCL может предложить больше, чем кажется на первый взгляд. Например, FCL уже содержит довольно много алгоритмов шифрования. Они находятся в пространстве имён System::Security::Cryptography. Если нужным вам алгоритм шифрования уже есть в FCL, вам не нужно заново создавать для него обёртку. Если FCL не содержит реализации алгоритма, для которого вы хотите создать обёртку, но приложение не завязано на алгоритм, реализованный в нативном API, то обычно предпочтительней использовать один из стандартных алгоритмов из FCL.

Создаваемые типы должны не только соблюдать правила CLS, но и, по возможности, вписываться в философию .NET.

### Механизм Platform Invoke

С помощью механизма Platform Invoke, более известного как P/Invoke, можно вызывать из управляемого кода функции в стиле языка C, которые экспортируются библиотеками DLL [19]. Для применения данного механизма, в управляемый код нужно объявить статический внешний (static extern) метод с эквивалентной функции на языке C сигнатурой. Метод должен иметь атрибут DllImport, который, как минимум, определяет библиотеку DLL, экспортирующую требуемую функцию. Для создания прототипа, который допускает вызов неуправляемого кода для правильного маршалинга данных, необходимо выполнить следующие действия:

- применить атрибут DllImportArtibute к статической функции или методу в управляемом коде;

- заменить неуправляемые типы данных на управляемые.

Вызов неуправляемого кода – это служба, позволяющая управляемому коду вызывать неуправляемые функции, реализованные в библиотеках динамической компоновки (DLL), например, функции библиотек Win32 API. Он обнаруживает и вызывает экспортированную функцию и при необходимости маршалирует ее аргументы (целые числа, строки, массивы, структуры и так далее) через границы взаимодействия.

Использование экспортированных функций DLL включает в себя следующие пункты [20].

1. Идентификация функций в библиотеках DLL.

Как минимум, необходимо указать имя функции и имя библиотеки DLL, содержащей ее.

1. Создание класса, содержащего функции DLL.

Можно использовать существующий класс, создать отдельный класс для каждой неуправляемой функции или создать один класс, содержащий набор связанных неуправляемых функций.

1. Создание прототипов в управляемом коде.

В случае разработки на языке Visual Basic используестя оператор Declare с ключевыми словами Function и Lib. В редких случаях можно использовать атрибут DllImportAttribute с ключевыми словами Shared Function. Такие случаи рассматриваются далее.

В случае разработки на языке C# используется атрибут DllImportAttribute для идентификации библиотеки DLL и функции. Также метод помечается модификаторами static и extern.

В случае разработки на языке C++ используется атрибут DllImportAttribute для идентификации библиотеки DLL и функции. Метод помечается или функцию оболочки модификатором extern "C".

Метод управляемого вызывается класса так же, как и любой другой управляемый метод. Передача структур и реализация функций обратного вызова представляют собой особые случаи.

Вызов неуправляемого кода использует метаданные для нахождения экспортированных функций и маршалинга их аргументов во время выполнения. Этот процесс показан на рисунке 3.1.

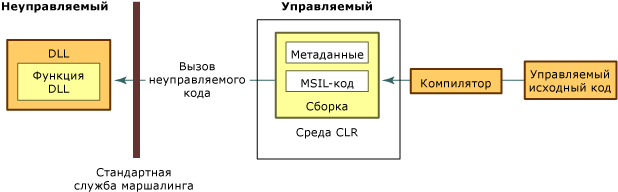


Рисунок 4.1 – Вызов неуправляемой функции DLL с помощью вызова неуправляемого кода

Когда неуправляемый код вызывает неуправляемую функцию, то он выполняет следующую последовательность действий:

1. определяет библиотеку DLL, содержащую функцию;
2. загружает библиотеку DLL в память;
3. находит адрес функции в памяти и помещает ее аргументы в стек, выполнив по необходимости маршалинг данных. Поиск и загрузка библиотеки DLL, а также определение адреса функции в памяти выполняются только при первом вызове функции;
4. передает управление неуправляемой функции.

Вызов неуправляемого кода вызывает исключения, создаваемые неуправляемой функцией для управляемого вызывающего объекта.

Ниже в таблице 3.1 перечислены типы данных, используемые в функциях Win32 API (перечислены в файле Wtypes.h) и в функциях в стиле C. Многие неуправляемые библиотеки содержат функции, передающие эти типы данных в качестве параметров и возвращаемых значений. В третьем столбце представлены соответствующие встроенные типы значений .NET Framework или классы, используемые в управляемом коде. В некоторых случаях типы, перечисленные в таблице, можно заменить на типы того же размера.

Таблица 4.1 – Соответствие типов данных

|  |  |  |  |
| --- | --- | --- | --- |
| Неуправляемый тип в Wtypes.h | Неуправляемый тип языка C | Имя управляемого класса | Описание |
| HANDLE | void\* | System.IntPtr | 32 бита в 32-разрядных операционных системах Windows, 64 бита в 64-разрядных операционных системах Windows. |
| BYTE | unsigned char | System.Byte | 8 бит |
| Short | Short | System.Int16 | 16 бит |
| WORD | unsigned short | System.UInt16 | 16 бит |
| int | int | System.Int32 | 32 бита |
| UINT | unsigned int | System.UInt32 | 32 бита |
| LONG | long | System.Int32 | 32 бита |
| BOOL | long | System.Byte | 32 бита |
| DWORD | unsigned long | System.UInt32 | 32 бита |
| ULONG | unsigned long | System.UInt32 | 32 бита |
| CHAR | char | System.Char | В кодировке ANSI. |
| WCHAR | wchar\_t | System.Char | В кодировке Юникод. |
| LPSTR | char\* | System.String или System.Text.StringBuilder | В кодировке ANSI. |
| LPCSTR | Const char\* | System.String или System.Text.StringBuilder | В кодировке ANSI. |

|  |  |  |  |
| --- | --- | --- | --- |
| Продолжение таблицы 4.1 | | | |
| Неуправляемый тип в Wtypes.h | Неуправляемый тип языка C | Имя управляемого класса | Описание |
| LPWSTR | wchar\_t\* | System.String или System.Text.StringBuilder | В кодировке Юникод. |
| LPCWSTR | Const wchar\_t\* | System.String или System.Text.StringBuilder | В кодировке Юникод. |
| float | float | System.Single | 32 бита |
| DOUBLE | Double | System.Double | 64 бита |

Если в объявлении функции присутствуют структуры, то нужно определить структуры или классы в управляемом коде. При этом порядок и тип полей должен быть одинаковым. Иногда будет необходимо воспользоваться атрибутом MarshalAs, чтобы настроить поведение механизма маршалинга.

При первом обращении к функции с помощью механизма P/Invoke, сразу после загрузки библиотеки DLL, заглушки маршалера P/Invoke генерируются по требованию, и повторно используются в последующих вызовах. Маршалер выполняет следующие действия при вызове:

1. проверка наличия у вызывающего процесса права на выполнение неуправляемого кода;
2. преобразование управляемых аргументов в их неуправляемые аналоги, возможно с выделением памяти;
3. установка вытесняющего режима работы для потока выполнения сборщика мусора;
4. вызов библиотечной функции;
5. восстановление кооперативный режим работы потока выполнения сборщика мусора;
6. при необходимости сохраняет код ошибки Win32 в локальной переменной потока с последующим доступом с помощью метода Marshal.GetLastWin32Error();
7. преобразование значения типа HRESULT в исключение и возбуждает его при необходимости;
8. преобразование низкоуровневого исключения, если оно было возбуждено, в управляемое исключение;
9. преобразование возвращаемого значения и выходных параметров обратно в их управляемые аналоги4
10. освобождение временных блоков динамической памяти, выделенных при вызове.

Многие неуправляемые типы двоично совместимы с управляемым кодом. Эти типы, называемые двоично совместимыми (blittable), не требуют преобразования и передаются через границу между управляемым и неуправляемым кодом намного быстрее, чем двоично несовместимые (non-blittable) типы. В действительности заглушка маршалера может оптимизировать такую передачу еще больше, закрепляя управляемый объект и передавая неуправляемому коду указатель на него, исключая одну или две операции копирования (по одной для каждого направления передачи).

К двоично совместимым типам относятся:

- System.Byte (byte);

- System.SByte (sbyte);

- System.Int16 (short);

- System.UInt16 (ushort);

- System.Int32(int);

- System.UInt32 (uint);

- System.Int64 (long);

- System.UInt64 (ulong);

- System.IntPtr;

- System.UIntPtr;

- System.Single (float);

- System.Double (double).

Кроме того, одномерные массивы с элементами двоично совместимых типов (где все элементы принадлежат одному и тому же типу) также являются двоично совместимыми, так же как структуры или классы, состоящие только из полей двоично совместимых типов.

Тип System.Boolean (bool) не является двоично совместимым, потому что в неуправляемом коде он может иметь 1, 2 или 4-байтное представление. Тип System.Char (char) не является двоично совместимым, потому что может представлять символ ANSI или Юникода. Тип System.String (string) не является двоично совместимым, потому что его неуправляемое представление может состоять из символов ANSI или Юникода, и может быть строкой в стиле языка C или COM BSTR, а также потому, что управляемая строка должна быть неизменяемой.

Наивысшей производительности можно добиться, реализовав маршалинг входных строковых параметров вручную (см. следующий фрагмент кода). Но при этом вызываемая неуправляемая функция должна принимать строку в кодировке UTF-16, в стиле языка C, и никогда не писать в память, занимаемую строкой, из-за чего такая оптимизация редко бывает применима. Чтобы выполнить маршалинг вручную, необходимо закрепить входную строку, изменить сигнатуру P/Invoke так, чтобы неуправляемая функция выглядела, как принимающая IntPtr вместо String, и передавать ей указатель на закрепленную строку:

class Win32Interop

{

[DllImport("NativeDLL.DLL", CallingConvention = CallingConvention.Cdecl)]

// IntPtr вместо string

public static extern void NativeFunc(IntPtr pStr);

}

// ...

// Управляемый код вызывает функцию P/Invoke внутри области видимости

// fixed, что обеспечивает закрепление строки

unsafe

{

string str = "MyString";

fixed (char\* pStr = str)

{

// pStr можно использовать в нескольких вызовах

Win32Interop.NativeFunc((IntPtr)pStr);

}

}

Преобразование неуправляемой строки UTF-16 в стиле языка C в управляемую строку также можно оптимизировать, применив конструктор System.String, принимающий параметр типа char\*. Конструктор System.String создает копию буфера, поэтому неуправляемую память, занимаемую строкой, можно освободить сразу после создания управляемой строки. Обратите внимание, что здесь не выполняется никакой проверки, чтобы убедиться, что строка содержит только допустимые символы Юникода.

Как упоминалось выше, параметры функции могут передаваться заглушкой маршалера в одном или в двух направлениях. Направление передачи параметра определяется рядом факторов:

- является ли параметр ссылочным типом или типом значения;

- передается ли параметр по ссылке или по значению;

- является ли тип параметра двоично совместимым или нет;

- применяются ли к параметрам атрибуты, изменяющие направление маршалинга (System.RuntimeInteropService.InAttribute и System.RuntimeInteropService.OutAttribute).

Для дальнейшего обсуждения направление маршалинга из управляемого кода в неуправляемый будут обозначены как «in»; а направление из неуправляемого кода в управляемый - как «out». Ниже приводится список правил определения направления маршалинга по умолчанию:

- параметры, передаваемые по значению, независимо от того, являются ли они ссылочными типами или типами значений, передаются только в направлении «in» (не требуется применять атрибут In вручную, StringBuilder является исключением из правила и всегда передается в направлении «in/out»);

- параметры, передаваемые по ссылке (с применением ключевого слова ref в C# или ByRef в VB.NET), независимо от того, являются ли они ссылочными типами или типами значений, передаются в направлении «in/out»;

- добавление атрибута OutAttribute запрещает маршалинг в направлении «in», поэтому вызываемый неуправляемый код может не получить значение, переданное вызывающим кодом. Ключевое слово out в языке C# действует подобно ключевому слову ref, но добавляет атрибут OutAttribute;

- если типы параметров не являются двоично совместимыми в вызове P/Invoke и вам требуется организовать маршалинг только в направлении «out», ненужного маршалинга в направлении «in» можно избежать, использовав ключевое слово out вместо ref.

Из-за закрепления параметров двоично совместимых типов при маршалинге, как описывалось выше, для двоично совместимых ссылочных типов автоматически устанавливается направление «in/out», даже если правила выше утверждают иное. Однако не следует полагаться на эту особенность, когда требуется получить поведение «out» или «in/out» маршалинга, а вместо этого указывать направление явно, с помощью атрибутов, так как данная особенность перестанет действовать, если позднее вы добавите поле двоично несовместимого типа или если это вызов COM-объекта, пересекающий границы подразделений (apartments).

Разница между маршалингом типов значений и ссылочных типов заключается в особенностях их передачи через стек:

- типы значений, которые передаются по значению, копируются на стек, поэтому они всегда передаются в направлении «in», независимо от используемых атрибутов;

- типы значений, которые передаются по ссылке, и ссылочные типы, которые передаются по значению, передаются как указатель;

- ссылочные типы, которые передаются по ссылке, передаются как указатель на указатель.

Передача объемных типов значений (более десятка байт) по значению стоит дороже передачи их по ссылке. То же относится к объемным возвращаемым значениям, вместо которых может оказаться предпочтительнее использовать выходные параметры.

Механизм .NET Code Access Security позволяет выполнять код, не вызывающий доверия, в изолированном окружении, называемом «песочницей» (sandbox), с ограниченным доступом к возможностям среды выполнения (например, P/Invoke) и BCL (например, доступ к файлам и реестру). Когда вызывается неуправляемый код, механизм Code Access Security требует, чтобы все сборки, чьи методы будут вызываться, имели право UnmanagedCode. Заглушка маршалера будет проверять это право для каждого вызова, что влечет за собой обход стека вызовов, чтобы убедиться, что весь код в цепочке вызовов обладает данным правом.

Если вы выполняете только код, пользующийся доверием, или у вас имеются иные средства, гарантирующие безопасность, вы можете значительно увеличить производительность, добавив атрибут SuppressUnmanagedCodeSecurityAttribute в объявление метода, класса (в этом случае данный атрибут применяется ко всем методам), интерфейса или делегата.

## Интеграция ngspice.dll в .NET C#

Ниже приведенная таблица 4.2 содержит объявления структур и функций в файле заголовка и полученные их прототипы в управляемом коде.

Таблица 4.2 – Таблица маршалинга

|  |  |
| --- | --- |
| sharedspice.h | C# |
| struct ngcomplex {  double cx\_real;  double cx\_imag;  }; | [StructLayout(LayoutKind.Sequential)]  public struct ngcomplex  {  public double cx\_real;  public double cx\_imag;  } |
| typedef struct vector\_info {  char \*v\_name;  int v\_type;  short v\_flags;  double \*v\_realdata;  ngcomplex\_t \*v\_compdata;  int v\_length;  } vector\_info, \*pvector\_info; | [StructLayout(LayoutKind.Sequential)]  public struct vector\_info  {  /// char\*  [MarshalAs(UnmanagedType.LPStr)]  public string v\_name;  /// int  public int v\_type;  /// short  public short v\_flags;  /// double\*  public IntPtr v\_realdata;  /// ngcomplex\_t\*  public IntPtr v\_compdata;  /// int  public int v\_length;  } |
| typedef struct vecvalues {  char\* name;  double creal;  bool is\_scale;  bool is\_complex;  } vecvalues, \*pvecvalues; | [StructLayout(LayoutKind.Sequential)]  public struct vecvalues  {  /// char\*  [MarshalAs(UnmanagedType.LPStr)]  public string name;  /// double  public double creal;  /// double  public double cimag;  /// boolean  [MarshalAs(UnmanagedType.U1)]  public bool is\_scale;  /// boolean  [MarshalAs(UnmanagedType.U1)]  public bool is\_complex;  } |

|  |  |
| --- | --- |
| Продолжение таблицы 4.2 | |
| sharedspice.h | C# |
| typedef struct vecvaluesall {  int veccount;  int vecindex;  pvecvalues \*vecsa;  } vecvaluesall, \*pvecvaluesall; | [StructLayout(LayoutKind.Sequential)]  public struct vecvaluesall  {  /// int  public int veccount;  /// int  public int vecindex;  /// pvecvalues\*  public IntPtr vecsa;  } |
| typedef struct vecinfo  {  int number;  char \*vecname;  bool is\_real;  void \*pdvec;  void \*pdvecscale;  } vecinfo, \*pvecinfo; | [StructLayout(LayoutKind.Sequential)]  public struct vecinfo  {  /// int  public int number;  /// char\*  [MarshalAs(UnmanagedType.LPStr)]  public string vecname;  /// boolean  public bool is\_real;  /// void\*  public IntPtr pdvec;  /// void\*  public IntPtr pdvecscale;  } |
| typedef struct vecinfoall  {  char \*name;  char \*title;  char \*date;  char \*type;  int veccount;  pvecinfo \*vecs;  } vecinfoall, \*pvecinfoall; | [StructLayout(LayoutKind.Sequential)]  public struct vecinfoall  {  /// char\*  [MarshalAs(UnmanagedType.LPStr)]  public string name;  /// char\*  [MarshalAs(UnmanagedType.LPStr)]  public string title;  /// char\*  [MarshalAs(UnmanagedType.LPStr)]  public string date;  /// char\*  [MarshalAs(UnmanagedType.LPStr)]  public string type;  /// int  public int veccount;  /// pvecinfo\*  public IntPtr vecs;  } |
| typedef int (SendChar)(char\*, int, void\*); | [UnmanagedFunctionPointer(CallingConvention.Cdecl)]  public delegate int SendChar(IntPtr param0, int param1, IntPtr param2); |

|  |  |
| --- | --- |
| Продолжение таблицы 4.2 | |
| sharedspice.h | C# |
| typedef int (SendStat)(char\*, int, void\*); | [UnmanagedFunctionPointer(CallingConvention.Cdecl)]  public delegate int SendStat(IntPtr param0, int param1, IntPtr param2); |
| typedef int (ControlledExit)(int, bool, bool, int, void\*); | [UnmanagedFunctionPointer(CallingConvention.Cdecl)]  public delegate int ControlledExit(int param0, [MarshalAs(UnmanagedType.I1)] bool param1, [MarshalAs(UnmanagedType.I1)] bool param2, int param3, IntPtr param4); |
| typedef int (SendData)(pvecvaluesall, int, int, void\*); | [UnmanagedFunctionPointer(CallingConvention.Cdecl)]  public delegate int SendInitData(IntPtr param0, int param1, IntPtr param2); |
| typedef int (SendInitData)(pvecinfoall, int, void\*); | [UnmanagedFunctionPointer(CallingConvention.Cdecl)]  public delegate int BGThreadRunning([MarshalAs(UnmanagedType.I1)] bool param0, int param1, IntPtr param2); |
| typedef int (BGThreadRunning)(bool, int, void\*); | [UnmanagedFunctionPointer(CallingConvention.Cdecl)]  public delegate int BGThreadRunning([MarshalAs(UnmanagedType.I1)] bool param0, int param1, IntPtr param2); |
| typedef int (GetVSRCData)(double\*, double, char\*, int, void\*); | [UnmanagedFunctionPointer(CallingConvention.Cdecl)]  public delegate int GetVSRCData(IntPtr param0, double param1, IntPtr param2, int param3, IntPtr param4); |
| typedef int (GetISRCData)(double\*, double, char\*, int, void\*); | [UnmanagedFunctionPointer(CallingConvention.Cdecl)]  public delegate int GetISRCData(IntPtr param0, double param1, IntPtr param2, int param3, IntPtr param4); |
| typedef int (GetSyncData)(double, double\*, double, int, int, int, void\*); | [UnmanagedFunctionPointer(CallingConvention.Cdecl)]  public delegate int GetSyncData(double param0, IntPtr param1, double param2, int param3, int param4, int param5, IntPtr param6); |
| int ngSpice\_Init(  SendChar\* printfcn, SendStat\* statfcn, ControlledExit\* ngexit, SendData\* sdata, SendInitData\* sinitdata, BGThreadRunning\* bgtrun, void\* userData); | [DllImport("ngspice.dll",  EntryPoint = "ngSpice\_Init",  SetLastError = true,  CallingConvention = CallingConvention.Cdecl)]  public static extern int ngSpice\_Init(IntPtr printfcn, IntPtr statfcn, IntPtr ngexit, IntPtr sdata, IntPtr sinitdata, IntPtr bgtrun, IntPtr userData); |
| Продолжение таблицы 4.2 | |
| sharedspice.h | C# |
| int ngSpice\_Init\_Sync(GetVSRCData \*vsrcdat, GetISRCData \*isrcdat, GetSyncData \*syncdat, int \*ident, void \*userData); | [DllImport("ngspice.dll",  EntryPoint = "ngSpice\_Init\_Sync",  CallingConvention = CallingConvention.Cdecl)]  public static extern int ngSpice\_Init\_Sync(IntPtr vsrcdat, IntPtr isrcdat, IntPtr syncdat, IntPtr ident, IntPtr userData); |
| int ngSpice\_Command(char\* command); | [DllImport("ngspice.dll",  EntryPoint = "ngSpice\_Command",  CallingConvention = CallingConvention.Cdecl)]  public static extern int ngSpice\_Command(  [MarshalAs(UnmanagedType.LPStr)] string command); |
| pvector\_info ngGet\_Vec\_Info(char\* vecname); | [DllImport("ngspice.dll",  EntryPoint = "ngGet\_Vec\_Info",  CallingConvention = CallingConvention.Cdecl)]  public static extern IntPtr ngGet\_Vec\_Info( [MarshalAs(UnmanagedType.LPStr)] string vecname); |
| int ngSpice\_Circ(char\*\* circarray); | [DllImport("ngspice.dll",  EntryPoint = "ngSpice\_Circ",  CallingConvention = CallingConvention.Cdecl)]  public static extern int ngSpice\_Circ(IntPtr circarray); |
| char\* ngSpice\_CurPlot(void); | [DllImport("ngspice.dll",  EntryPoint = "ngSpice\_CurPlot",  CallingConvention = CallingConvention.Cdecl)]  public static extern IntPtr ngSpice\_CurPlot(); |
| char\*\* ngSpice\_AllPlots(void); | [DllImport("ngspice.dll",  EntryPoint = "ngSpice\_AllPlots",  CallingConvention = CallingConvention.Cdecl)]  public static extern IntPtr ngSpice\_AllPlots(); |
| char\*\* ngSpice\_AllVecs(char\* plotname); | [DllImport("ngspice.dll",  EntryPoint = "ngSpice\_AllVecs",  CallingConvention = CallingConvention.Cdecl)]  public static extern IntPtr ngSpice\_AllVecs(string plotname); |
| bool ngSpice\_running(void); | [DllImport("ngspice.dll",  EntryPoint = "ngSpice\_running",  CallingConvention = CallingConvention.Cdecl)]  [return: MarshalAs(UnmanagedType.I1)]  public static extern bool ngSpice\_running(); |
| bool ngSpice\_SetBkpt(double time); | [DllImport("ngspice.dll",  EntryPoint = "ngSpice\_SetBkpt",  CallingConvention = CallingConvention.Cdecl)]  [return: MarshalAs(UnmanagedType.I1)]  public static extern bool ngSpice\_SetBkpt(double time); |

**Выводы**

Для создания хороших обёрток над нативными библиотеками нужно обладать определёнными качествами и возможностями. Прежде всего, требуются хорошие способности по разработке архитектуры. Отображение нативной библиотеки один-к-одному редко является хорошим решением. Отбросив возможности, которые не нужны потребителям библиотеки, можно значительно упростить задачу. Создание фасада для оставшихся возможностей библиотеки требует знания CLS, возможностей FCL и часто используемых в .NET подходов. Создание обёртки над нативной библиотекой также предполагает обёртывание нативных ресурсов (например, неуправляемой памяти, необходимой для создания оборачиваемых объектов).

Для интеграции API Ngspice был выбран второй вариант – механизм Platform Invoke как наименее затратный по времени. Возможностей P/Invoke хватает для обеспечения взаимодействия с неуправляемым кодом библиотеки.

На данном этапе работы был сделан маршалинг API структур и методов Ngspice. Данные объявления можно использовать в .NET C# проектах и вызывать методы из неуправляемого кода в проекте с управляемым кодом.

# Описание технических и функциональных возможностей модуля

Для графического описания абстрактной модели проекта, а также пользовательского взаимодействия (вариантов использования ПО) использован стандарт UML [21].

UML язык графического описания для объектного моделирования в области разработки программного обеспечения. UML является языком широкого профиля, это – открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML-моделью. UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. UML не является языком программирования, но на основании UML-моделей возможна генерация кода.

При использовании UML были построены: диаграмма вариантов использования, которая представлена на рисунке 5.1 и диаграмма классов представлена на рисунке 5.2.



Рисунок 5.1 – Диаграмма вариантов использования

Рисунок 5.2 – Диаграмма классов

# Тестирование модуля

Тестирование продукта является обязательной частью при разработке любого программного обеспечения. В ходе данной работы были проведены функциональные и модульные тесты модуля.

## Функциональные тесты

Функциональное тестирование — это тестирование в целях проверки реализуемости функциональных требований, то есть способности программного обеспечения (ПО) в определённых условиях решать задачи, нужные пользователям [16]. Функциональные требования определяют, что именно делает ПО, какие задачи оно решает.

При запуске плагина открывается окно, показанное на рисунке 6.1. Ячейки для ввода данных недоступны, а строка состояния уведомляет о необходимости выбора или создания схемы.

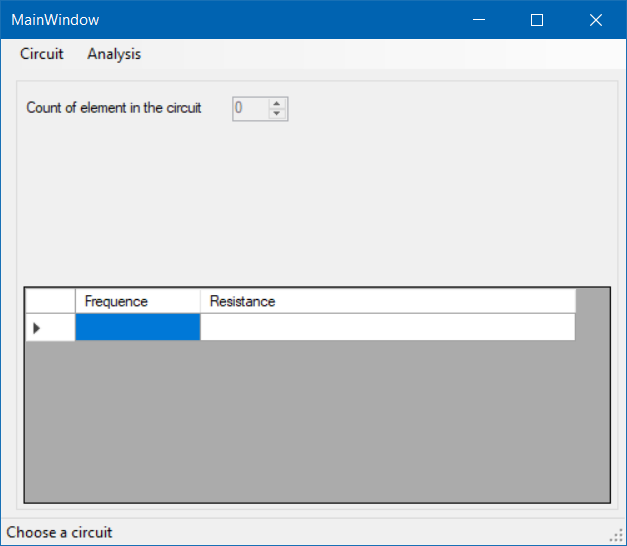


Рисунок 6.1 – Окно запуска

В разделе меню «Circuit» выбирается способ создания схемы. Как видно на рисунке 6.2, можно выбрать из выпадающего списка одну из пяти цепей, либо создать новый.

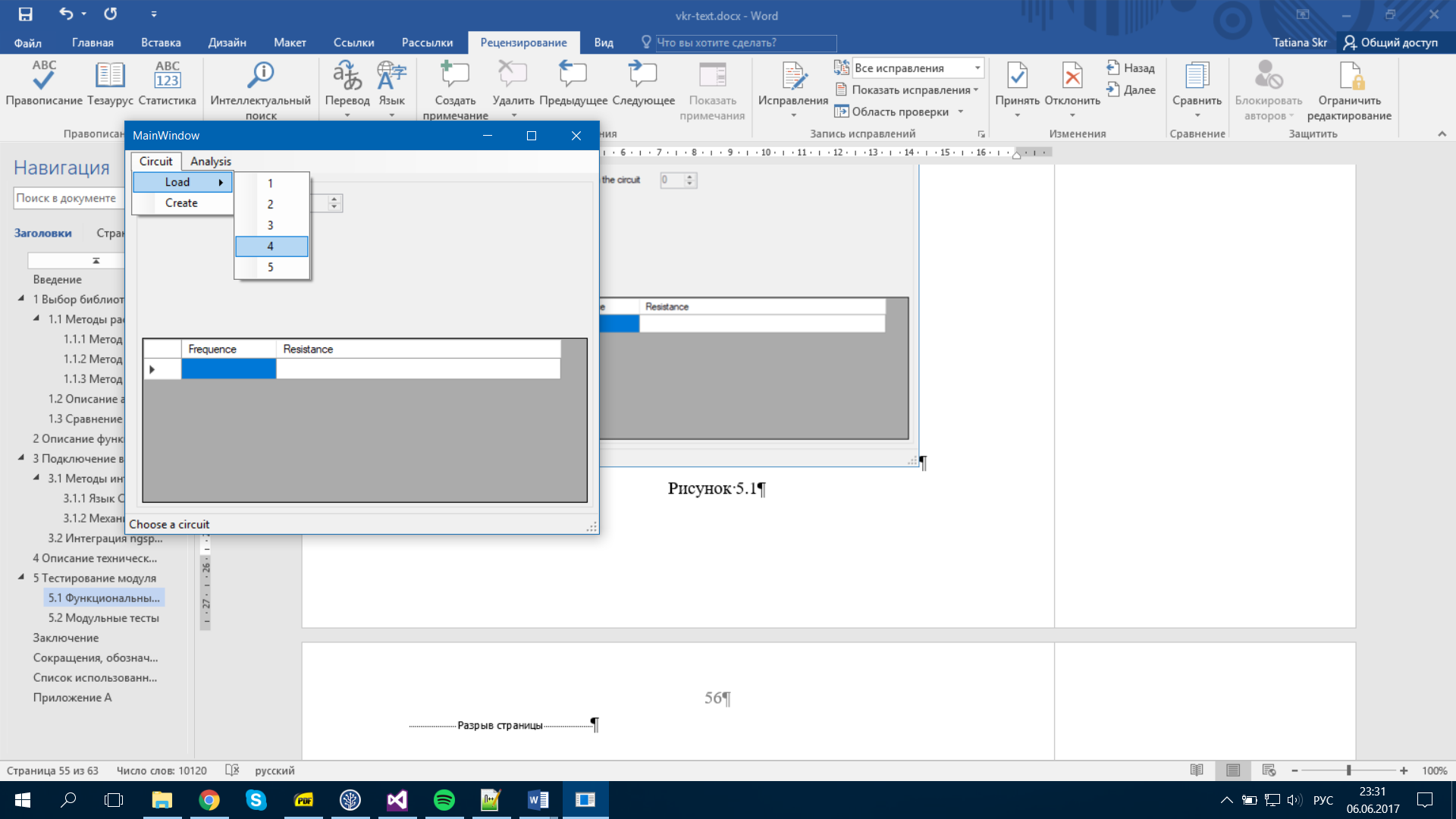


Рисунок 6.2 – Выбор схемы

При попытке ввода неправильных символов в ячейки ввода параметров, выдается ошибка ввода данных. Окно сообщения об ошибке представлено на рисунке 6.3.

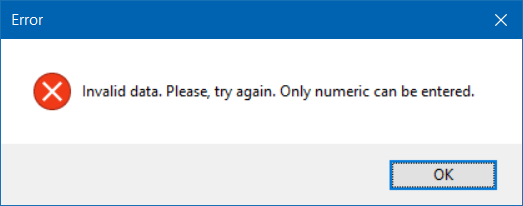


Рисунок 6.3 – Сообщение об ошибке ввода

Результат расчета комплексного сопротивления при вводе корректных данных показан на рисунке 6.4.

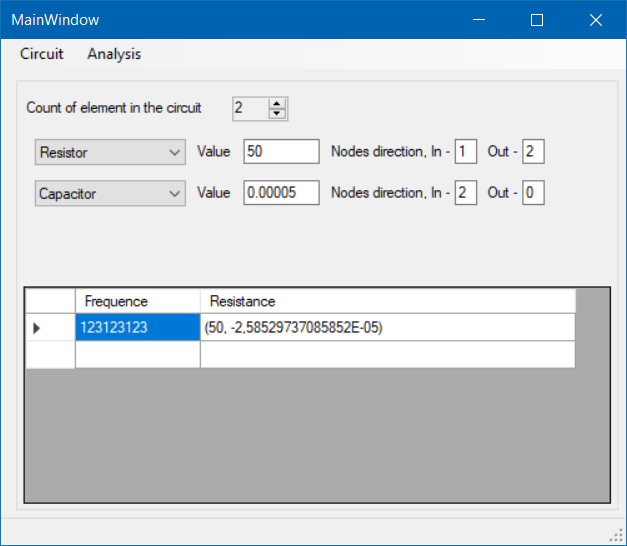


Рисунок 6.4 – Результат вычисления при корректно введенных данных

Операция перехода в режим анализа частотных характеристик показана на рисунке 6.5

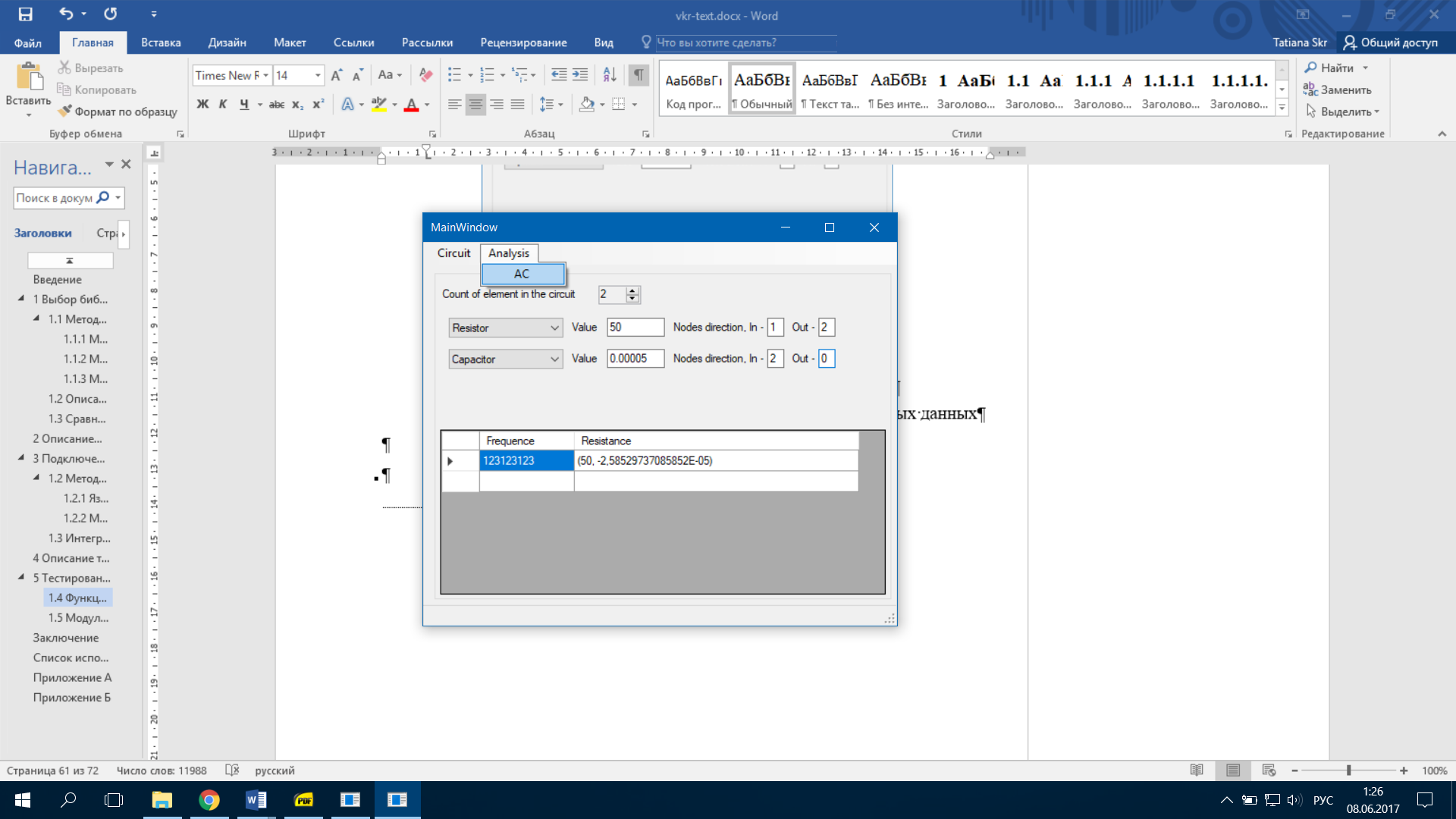


Рисунок 6.5 – Переход на анализ частотных характеристик

На рисунке 6.6 представлен результат анализа в логарифмическом масштабе.

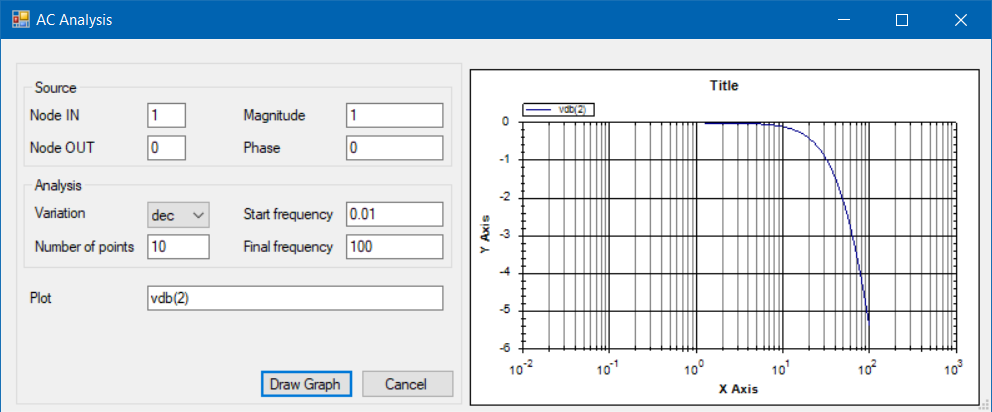


Рисунок 6.6 – Результаты анализа частотных характеристик в логарифмическом масштабе

На рисунке 6.7 представлен результат анализа в линейном масштабе.

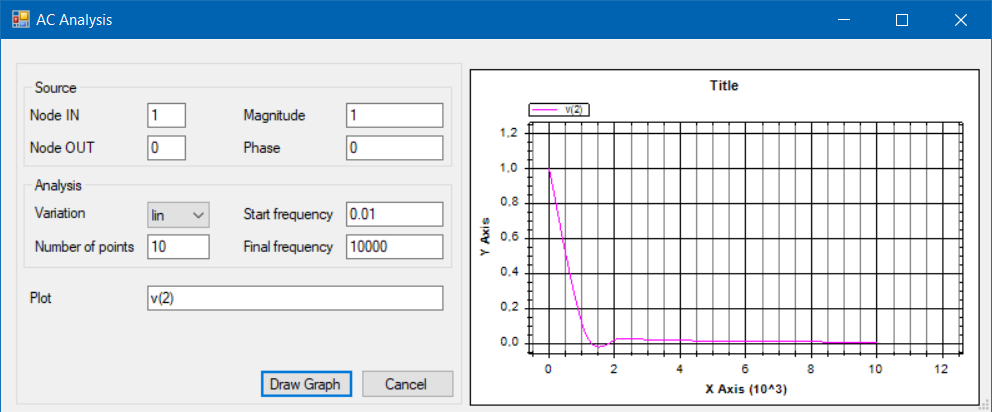


Рисунок 6.7 – Результаты анализа частотных характеристик в линейном масштабе

## Модульные тесты

В приложении В приведены значения входных параметров при модульном тестировании в виде листинга части программы. Результаты модульных тестов приведены на рисунке 6.8.

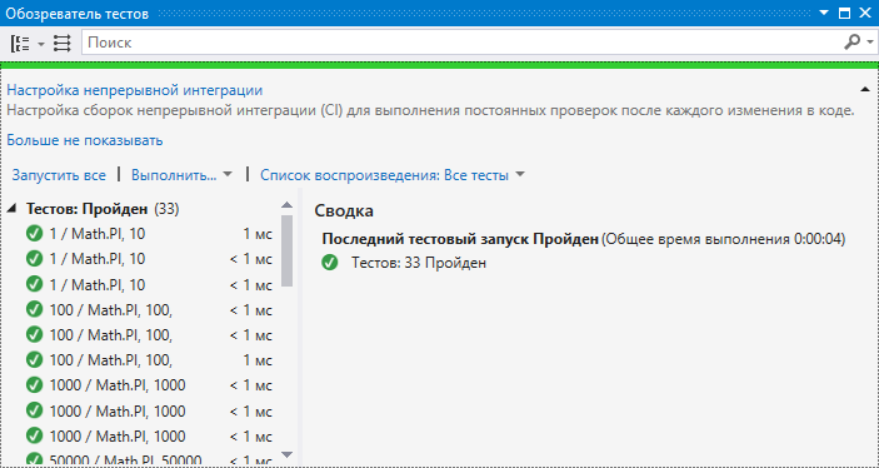


Рисунок 6.8 – Результат выполнения модульных тестов

# Заключение

В ходе выполнения выпускной квалификационной работы был проведен обзор СПО для симуляции аналоговых и цифровых схем. В результате исследования была выбрана библиотека, для дальнейшей разработки модуля. Далее были изучены функциональные возможности API Ngspice, а также рассмотрены варианты интеграции данной библиотеки к проекту C# .NET.

В результате выполнения выпускной квалификационной работы был разработан модуль для расчета параметров электронных схем. Схема может состоять из проcтых RLC элементов. В главной форме рассчитывается общее комплексное сопротивление заданной цепи. В форме для анализа частотных характеристик выводится график зависимости напряжения на узлах от частоты входного сигнала.

Перспективным расширением данного модуля является увеличение вариантов анализа и рассчитываемых параметров.

# Список использованных источников

1. Норенков И. П. Основы автоматизированного проектирования: учеб. для вузов / И. П. Норенков. – 4-е изд., перераб. и доп. – М. : Изд-во МГТУ им Н. Э. Баумана, 2009. – 430 с.
2. Трифоненко И. М., Горячев Н. В., Кочегаров И. И., Юрков Н. К. Обзор систем сквозного проектирования печатных плат радиоэлектронных средств [Электронный ресурс] // КиберЛенинка. – 2012. – URL: <http://cyberleninka.ru/article/n/obzor-sistem-skvoznogo-proektirovaniya-pechatnyh-plat-radioelektronnyh-sredstv> (дата обращения 03.05.17).
3. Сальникова Н. А., Астафурова О. А. Методы моделирования в системах автоматизированного проектирования СВЧ-устройств [Электронный ресурс] // КиберЛенинка. – 2014. – URL: <http://cyberleninka.ru/article/n/metody-modelirovaniya-v-sistemah-avtomatizirovannogo-proektirovaniya-svch-ustroystv> (дата обращения 03.05.17).
4. Leon O. Chua, Peng-Min Lin. Computer-aided analysis of electronic circuits / Перевод с английского Е.В Виленкина, В.Н, Елисеева и др. – М.: «Энергия», 1980. – 640 с., ил.
5. Атабеков Г. И. Теоретические основы электротехники. Линейные электрические цепи: учебное пособие / Г.И. Атабеков. – Спб.: Издательство «Лань», 2009. – 592 с.
6. Решение ТОЭ ОТЦ ТЛЭЦ Электротехника [Электронный ресурс]. – URL: <http://rgr-toe.ru> (дата обращения 23.04.17).
7. Getting Started Guide. Chapter 4. MWO: Using the Linear Simulator [Электронный ресурс]. – URL: [awrcorp.com/download/faq/english/docs/Getting\_Started/ch04.html](https://awrcorp.com/download/faq/english/docs/Getting_Started/ch04.html) (дата обращения 23.04.17).
8. Circuit Simulation [Электронный ресурс]. – URL: [literature.cdn.keysight.com/litweb/pdf/ads2003c/cktsim/index.html](http://literature.cdn.keysight.com/litweb/pdf/ads2003c/cktsim/index.html) (дата обращения 23.04.17).
9. Черкашин М.В. Модели и методы анализа проектных решений: учеб. пособие / М.В. Черкашин. – Томск: Томск. гос. ун-т систем упр. и радиоэлектроники, 2012. – 296 с.
10. SPICE (симулятор электронных схем) – Википедия [Электронный ресурс]. – URL: <https://en.wikipedia.org/wiki/SPICE> (дата обращения 27.04.17).
11. Сайт ПАЯЛЬНИК [Электронный ресурс]. – URL: <http://cxem.net/> (дата обращения 27.04.17).
12. Ngspice – Википедия [Электронный ресурс]. – URL: <https://ru.wikipedia.org/wiki/Ngspice> (дата обращения 27.04.17).
13. Qucs – Википедия [Электронный ресурс]. – URL: <https://ru.wikipedia.org/wiki/Qucs> (дата обращения 04.05.17).
14. HSPICE [Электронный ресурс]. – URL: <http://dic.academic.ru/dic.nsf/ruwiki/1700378> (дата обращения 04.05.17).
15. Интеграция Native кода в C# проект [Электронный ресурс]. – URL: <http://itw66.ru/blog/c_sharp/499.html> (дата обращения 30.05.2017).
16. Управляемый код – Википедия [Электронный ресурс]. – URL: <https://ru.wikipedia.org/wiki/Управляемый_код> (дата обращения 30.05.2017).
17. С++/CLI – «клейкий» язык – Хабрахабр [Электронный ресурс]. – URL: <https://habrahabr.ru/post/47732/> (дата обращения 30.05.2017).
18. NET-обёртки нативных библиотек на С++/CLI – Хабрахабр [Электронный ресурс]. – URL: <https://habrahabr.ru/post/318224/> (дата обращения 30.05.2017).
19. Оптимизация приложений .NET – P/Invoke [Электронный ресурс]. – URL: <https://professorweb.ru/my/csharp/optimization/level6/6_2.php> (дата обращения 30.05.2017).
20. Microsoft Developer Network (MSDN) [Электронный ресурс]. – URL: <https://msdn.microsoft.com/ru-ru/library/26thfadc.aspx> (дата обращения 30.05.2017).
21. UML. [Электронный ресурс]. — Режим доступа: <http://www.uml.org/> (дата обращения 27.12.2016).

# Приложение А

(справочное)

**Описание типов лицензий**

Лицензия BSD (англ. BSD license, Berkeley Software Distribution license – Программная лицензия университета Беркли) – это лицензионное соглашение, впервые применённое для распространения UNIX-подобных операционных систем BSD. Права на исходный дистрибутив BSD официально принадлежат «попечителям университета Калифорнии» (англ. Regents of the University of California) – управляющему органу университета Калифорнии.

По сравнению с другими распространёнными лицензиями на свободное программное обеспечение (например, GNU General Public License) лицензия BSD налагает меньше ограничений на пользователя. Лицензия BSD допускает проприетарное коммерческое использование ПО. Для ПО, выпущенного под этой лицензией, допускается встраивание в проприетарные коммерческие продукты. Работы, основанные на таком ПО, даже могут распространяться под проприетарными лицензиями (но всё же обязаны соответствовать требованиям лицензии). Можно применять к распространяемому продукту одновременно лицензию BSD и какую-то другую.

Бесплатное программное обеспечение (англ. freeware) – программное обеспечение, лицензионное соглашение которого не требует каких-либо выплат правообладателю. Бесплатное программное обеспечение обычно распространяется в бинарном виде, без исходных кодов и является проприетарным программным обеспечением.

GNU General Public License – лицензия на свободное программное обеспечение, созданная в рамках проекта GNU в 1988 г., по которой автор передаёт программное обеспечение в общественную собственность. Цель GNU GPL – предоставить пользователю права копировать, модифицировать и распространять (в том числе на коммерческой основе) программы, а также гарантировать, что и пользователи всех производных программ получат вышеперечисленные права. GNU GPL не позволяет включать программу в проприетарное ПО.

GPL предоставляет получателям компьютерных программ следующие права, или «свободы»:

- свободу запуска программы с любой целью;

- свободу изучения того, как программа работает, и её модификации (предварительным условием для этого является доступ к исходному коду);

- свободу распространения копий как исходного, так и исполняемого кода;

- свободу улучшения программы, и выпуска улучшений в публичный доступ (предварительным условием для этого является доступ к исходному коду).

В общем случае распространитель программы, полученной на условиях GPL, либо программы, основанной на таковой, обязан предоставить получателю возможность получить соответствующий исходный код.

# Приложение Б

(справочное)

**Листинг заголовочного файла sharedspice.h**

/\* header file for shared ngspice \*/

/\* Copyright 2013 Holger Vogt \*/

/\* Modified BSD license \*/

/\*

Interface between a calling program (caller) and ngspice.dll (ngspice.so)

#ifndef NGSPICE\_DLL\_H

#define NGSPICE\_DLL\_H

#ifdef \_\_cplusplus

extern "C" {

#endif

#if defined(\_\_MINGW32\_\_) || defined(\_MSC\_VER) || defined(\_\_CYGWIN\_\_)

#ifdef SHARED\_MODULE

#define IMPEXP \_\_declspec(dllexport)

#else

#define IMPEXP \_\_declspec(dllimport)

#endif

#else

/\* use with gcc flag -fvisibility=hidden \*/

#if \_\_GNUC\_\_ >= 4

#define IMPEXP \_\_attribute\_\_ ((visibility ("default")))

#define IMPEXPLOCAL \_\_attribute\_\_ ((visibility ("hidden")))

#else

#define IMPEXP

#define IMPEXP\_LOCAL

#endif

#endif

/\* required only if header is used by the caller,

is already defined in ngspice.dll \*/

#ifndef ngspice\_NGSPICE\_H

/\* Complex numbers. \*/

struct ngcomplex {

double cx\_real;

double cx\_imag;

} ;

typedef struct ngcomplex ngcomplex\_t;

#endif

/\* vector info obtained from any vector in ngspice.dll.

Allows direct access to the ngspice internal vector structure,

as defined in include/ngspice/devc.h . \*/

typedef struct vector\_info {

char \*v\_name; /\* Same as so\_vname. \*/

int v\_type; /\* Same as so\_vtype. \*/

short v\_flags; /\* Flags (a combination of VF\_\*). \*/

double \*v\_realdata; /\* Real data. \*/

ngcomplex\_t \*v\_compdata; /\* Complex data. \*/

int v\_length; /\* Length of the vector. \*/

} vector\_info, \*pvector\_info;

typedef struct vecvalues {

char\* name; /\* name of a specific vector \*/

double creal; /\* actual data value \*/

double cimag; /\* actual data value \*/

bool is\_scale; /\* if 'name' is the scale vector \*/

bool is\_complex; /\* if the data are complex numbers \*/

} vecvalues, \*pvecvalues;

typedef struct vecvaluesall {

int veccount; /\* number of vectors in plot \*/

int vecindex; /\* index of actual set of vectors. i.e. the number of accepted data points \*/

pvecvalues \*vecsa; /\* values of actual set of vectors, indexed from 0 to veccount - 1 \*/

} vecvaluesall, \*pvecvaluesall;

/\* info for a specific vector \*/

typedef struct vecinfo

{

int number; /\* number of vector, as postion in the linked list of vectors, starts with 0 \*/

char \*vecname; /\* name of the actual vector \*/

bool is\_real; /\* TRUE if the actual vector has real data \*/

void \*pdvec; /\* a void pointer to struct dvec \*d, the actual vector \*/

void \*pdvecscale; /\* a void pointer to struct dvec \*ds, the scale vector \*/

} vecinfo, \*pvecinfo;

/\* info for the current plot \*/

typedef struct vecinfoall

{

/\* the plot \*/

char \*name;

char \*title;

char \*date;

char \*type;

int veccount;

/\* the data as an array of vecinfo with length equal to the number of vectors in the plot \*/

pvecinfo \*vecs;

} vecinfoall, \*pvecinfoall;

/\* callback functions

addresses received from caller with ngSpice\_Init() function

\*/

/\* sending output from stdout, stderr to caller \*/

typedef int (SendChar)(char\*, int, void\*);

/\*

char\* string to be sent to caller output

int identification number of calling ngspice shared lib

void\* return pointer received from caller, e.g. pointer to object having sent the request

\*/

/\* sending simulation status to caller \*/

typedef int (SendStat)(char\*, int, void\*);

/\*

char\* simulation status and value (in percent) to be sent to caller

int identification number of calling ngspice shared lib

void\* return pointer received from caller

\*/

/\* asking for controlled exit \*/

typedef int (ControlledExit)(int, bool, bool, int, void\*);

/\*

int exit status

bool if true: immediate unloading dll, if false: just set flag, unload is done when function has returned

bool if true: exit upon 'quit', if false: exit due to ngspice.dll error

int identification number of calling ngspice shared lib

void\* return pointer received from caller

\*/

/\* send back actual vector data \*/

typedef int (SendData)(pvecvaluesall, int, int, void\*);

/\*

vecvaluesall\* pointer to array of structs containing actual values from all vectors

int number of structs (one per vector)

int identification number of calling ngspice shared lib

void\* return pointer received from caller

\*/

/\* send back initailization vector data \*/

typedef int (SendInitData)(pvecinfoall, int, void\*);

/\*

vecinfoall\* pointer to array of structs containing data from all vectors right after initialization

int identification number of calling ngspice shared lib

void\* return pointer received from caller

\*/

/\* indicate if background thread is running \*/

typedef int (BGThreadRunning)(bool, int, void\*);

/\*

bool true if background thread is running

int identification number of calling ngspice shared lib

void\* return pointer received from caller

\*/

/\* callback functions

addresses received from caller with ngSpice\_Init\_Sync() function

\*/

/\* ask for VSRC EXTERNAL value \*/

typedef int (GetVSRCData)(double\*, double, char\*, int, void\*);

/\*

double\* return voltage value

double actual time

char\* node name

int identification number of calling ngspice shared lib

void\* return pointer received from caller

\*/

/\* ask for ISRC EXTERNAL value \*/

typedef int (GetISRCData)(double\*, double, char\*, int, void\*);

/\*

double\* return current value

double actual time

char\* node name

int identification number of calling ngspice shared lib

void\* return pointer received from caller

\*/

/\* ask for new delta time depending on synchronization requirements \*/

typedef int (GetSyncData)(double, double\*, double, int, int, int, void\*);

/\*

double actual time (ckt->CKTtime)

double\* delta time (ckt->CKTdelta)

double old delta time (olddelta)

int redostep (as set by ngspice)

int identification number of calling ngspice shared lib

int location of call for synchronization in dctran.c

void\* return pointer received from caller

\*/

/\* ngspice initialization,

printfcn: pointer to callback function for reading printf, fprintf

statfcn: pointer to callback function for the status string and percent value

ControlledExit: pointer to callback function for setting a 'quit' signal in caller

SendData: pointer to callback function for returning data values of all current output vectors

SendInitData: pointer to callback function for returning information of all output vectors just initialized

BGThreadRunning: pointer to callback function indicating if workrt thread is running

userData: pointer to user-defined data, will not be modified, but

handed over back to caller during Callback, e.g. address of calling object \*/

IMPEXP

int ngSpice\_Init(SendChar\* printfcn, SendStat\* statfcn, ControlledExit\* ngexit,

SendData\* sdata, SendInitData\* sinitdata, BGThreadRunning\* bgtrun, void\* userData);

/\* initialization of synchronizing functions

vsrcdat: pointer to callback function for retrieving a voltage source value from caller

isrcdat: pointer to callback function for retrieving a current source value from caller

syncdat: pointer to callback function for synchronization

ident: pointer to integer unique to this shared library (defaults to 0)

userData: pointer to user-defined data, will not be modified, but

handed over back to caller during Callback, e.g. address of calling object.

If NULL is sent here, userdata info from ngSpice\_Init() will be kept, otherwise

userdata will be overridden by new value from here.

\*/

IMPEXP

int ngSpice\_Init\_Sync(GetVSRCData \*vsrcdat, GetISRCData \*isrcdat, GetSyncData \*syncdat, int \*ident, void \*userData);

/\* Caller may send ngspice commands to ngspice.dll.

Commands are executed immediately \*/

IMPEXP

int ngSpice\_Command(char\* command);

/\* get info about a vector \*/

IMPEXP

pvector\_info ngGet\_Vec\_Info(char\* vecname);

/\* send a circuit to ngspice.dll

The circuit description is a dynamic array

of char\*. Each char\* corresponds to a single circuit

line. The last entry of the array has to be a NULL \*/

IMPEXP

int ngSpice\_Circ(char\*\* circarray);

/\* return to the caller a pointer to the name of the current plot \*/

IMPEXP

char\* ngSpice\_CurPlot(void);

/\* return to the caller a pointer to an array of all plots created

so far by ngspice.dll \*/

IMPEXP

char\*\* ngSpice\_AllPlots(void);

/\* return to the caller a pointer to an array of vector names in the plot

named by plotname \*/

IMPEXP

char\*\* ngSpice\_AllVecs(char\* plotname);

/\* returns TRUE if ngspice is running in a second (background) thread \*/

IMPEXP

bool ngSpice\_running(void);

/\* set a breakpoint in ngspice \*/

IMPEXP

bool ngSpice\_SetBkpt(double time);

#ifdef \_\_cplusplus

}

#endif

#endif

# Приложение В

(справочное)

**Входные параметры модульных тестов**

Модульное тестирование проводится для метода расчета комплексного сопротивления элемента у классов Capacitor, Inductor, Resistor. Ниже перечислены тестовые случаи.

[TestCase(double.MinValue, double.MinValue, 0, 0, TestName = "Couple MinValue")]

[TestCase(double.MaxValue, double.MaxValue, 0, 0, TestName = "Couple MaxValue")]

[TestCase(double.NaN, double.NaN, 0, 0, TestName = "Couple NaN")]

[TestCase(double.NegativeInfinity, double.NegativeInfinity, 0, 0, TestName = "Couple NegativeInfinity")]

[TestCase(double.PositiveInfinity, double.PositiveInfinity, 0, 0, TestName = "Couple PositiveInfinity")]

[TestCase(double.NegativeInfinity, double.PositiveInfinity, 0, 0, TestName = "NegativeInfinity frequency and PositiveInfinity value")]

[TestCase(0, 0, 0, 0, TestName = "Couple 0")]

[TestCase(10 / Math.PI, 10, 10, 0, TestName = "1 / Math.PI, 10")]

[TestCase(100 / Math.PI, 100, 100, 0, TestName = "100 / Math.PI, 100,")]

[TestCase(1000 / Math.PI, 1000, 1000, 0, TestName = "1000 / Math.PI, 1000")]

[TestCase(50000 / Math.PI, 50000, 50000, 0, TestName = "50000 / Math.PI, 50000,")]