

ВВЕДЕНИЕ	4
1. ОСНОВЫ РАЗРАБОТКИ ПРОЕКТОВ В VISUAL C++6.0.....	4
2. ОСНОВНЫЕ ОПЕРАТОРЫ C	22
3. ФУНКЦИИ	26
4. ПОНЯТИЕ ОБ ОБЪЕКТНО-ОРИЕНТИРОВАННОМ ПРОГРАММИРОВАНИИ. БАЗОВЫЕ КЛАССЫ.....	32
5. УКАЗАТЕЛИ, ССЫЛКИ И МАССИВЫ	44
6. БЛОК-СХЕМЫ АЛГОРИТМОВ	51
7. ПРОГРАММЫ ИССЛЕДОВАНИЯ ФУНКЦИЙ.....	54
7.1. ОПРЕДЕЛЕНИЕ КОРНЕЙ УРАВНЕНИЯ МЕТОДОМ ДИХОТОМИИ.....	55
7.2. ОПРЕДЕЛЕНИЕ КОЛИЧЕСТВА ЭКСТРЕМУМОВ.....	55
7.3 ВЫЧИСЛЕНИЕ ОПРЕДЕЛЕННЫХ ИНТЕГРАЛОВ	56
7.3.1 Метод прямоугольников	56
7.3.2 Метод трапеций	56
7.3.3 Метод Монте-Карло.....	57
8. МОДЕЛИРОВАНИЕ ЗАДАЧ ТЕПЛОПРОВОДНОСТИ	63
8.1. ОБРАБОТКА ЭКСПЕРИМЕНТАЛЬНЫХ ДАННЫХ	63
8.2 МОДЕЛИРОВАНИЕ ЗАКОНА ВНЕШНЕЙ ТЕПЛОПЕРЕДАЧИ.....	65
8.3 ОБРАБОТКА ДАННЫХ ЭКСПЕРИМЕНТА ЗАКОНА ТЕПЛОПЕРЕДАЧИ	69
9. МОДЕЛИРОВАНИЕ ВЯЗКОГО ТРЕНИЯ	75
10. ОСНОВЫ РАБОТЫ С ГРАФИЧЕСКИМИ СРЕДСТВАМИ C++	84
11. МЕТОД СТАТИСТИЧЕСКОГО МОДЕЛИРОВАНИЯ.....	89
11.1 МОДЕЛИРОВАНИЕ ОТКАЗОВ.....	89
11.2 МОДЕЛИРОВАНИЕ АЛГОРИТМА ИМИТАЦИИ ОТЖИГА.....	95
12. МОДЕЛИРОВАНИЕ СОСТОЯНИЙ ЧАСТИЦЫ В ПОТЕНЦИАЛЬНОЙ ЯМЕ.....	104
13. МОДЕЛИРОВАНИЕ ЦИФРОВОГО ФИЛЬТРА	107
13.1. СОЗДАНИЕ НОВОГО ПРОЕКТА.....	108
13.2 ФАЙЛ СИНТЕЗА ПРОЕКТА	114
13.3 ПОДГОТОВКА И ПРОВЕРКА КОДА	117
13.4 ОПТИМИЗАЦИЯ КОДА.....	127
13.4 ДИРЕКТИВЫ HLS	130
13.5 ЗАДАНИЕ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ.....	133
РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА.....	134

ВВЕДЕНИЕ

Цель настоящего практикума - приобретение навыков работы по созданию проектов приложений с использованием переменных, условий, функций, циклов, классов. В заданиях и примерах изучаются основные синтаксические конструкции языка C++, понятия переменных, использование функций, условий, циклов и классов, как основы объектно-ориентированного программирования. Приобретенные навыки программирования закрепляются при разработке программы исследования функций, разрабатывая которую студенты обучаются на практике применять основные алгоритмические структуры и вычислительные методы определения корней уравнений, численного дифференцирования и интегрирования. Контрольные задания иллюстрируют применение компьютерного моделирования в физике и электронике для определения параметров процессов на основе данных экспериментов и параметров электронных устройств.

1. ОСНОВЫ РАЗРАБОТКИ ПРОЕКТОВ В VISUAL C++6.0

В соответствии с классическим определением, программой называется служащая для выполнения каких-либо действий вычислительных устройств сформулированная с помощью определённых синтаксических конструкций конечная последовательность предписаний. Перевод программы с алгоритмического языка в машинный код осуществляют специальные программы-трансляторы: интерпретаторы и компиляторы. Интерпретатор обрабатывает программу пользователя поэлементно: преобразует в машинный код и сразу же его исполняет, т.е. программа выполняется ЭВМ только в среде интерпретатора. С помощью компилятора создаются файлы программы (exe-файлы), которые могут самостоятельно выполняться в среде операционной системы ЭВМ.

Этапы компиляции:

Препроцессинг

Компиляция

Ассемблирование

Компоновка

Загрузка

Самая первая стадия компиляции программы – препроцессинг, назначением которого является подготовка программы для последующего компилирования. Специальная программа вставляет или убирает в тексте куски кода в соответствии с условиями `#if`, `#ifdef` и `#ifndef`, анализирует препроцессорные директивы, добавляет в код заголовочные фрагменты (`#include`), убирает комментирования, заменяет макросы (`#define`) соответствующим кодом.

После препроцессинга C++ в проект добавляются файлы, которые имеют расширение `.ii`. При запуске компилятора может быть установлен флаг `-E`, который сообщает компилятору, что следует остановиться на стадии препроцессинга:

```
g++ -E driver.cpp -o driver.ii
```

Даже для простой программы, выводящей на консоль сообщение «Hello Word!» размер этого файла `*.ii` будет насчитывать более десяти тысяч строк, что иллюстрируется рисунком 1.1.

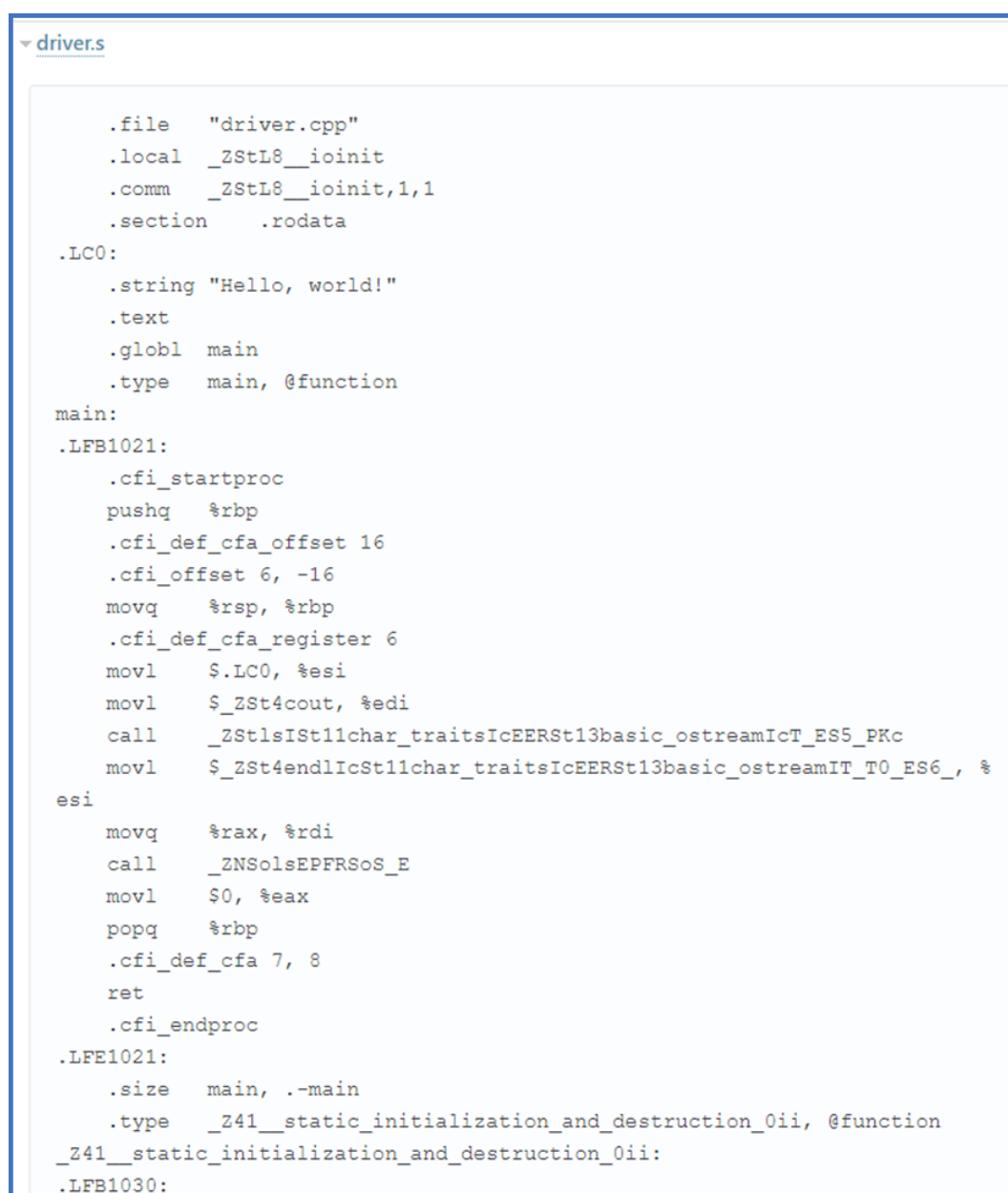
```
1. # 1 "driver.cpp"
2. # 1 "<built-in>"
3. # 1 "<command-line>"
4. # 1 "/usr/include/stdc-predef.h" 1 3 4
5. # 1 "<command-line>" 2
6. # 1 "driver.cpp"
7. # 1 "/usr/include/c++/5/iostream" 1 3
8. # 36 "/usr/include/c++/5/iostream" 3
9.
10. # 37 "/usr/include/c++/5/iostream" 3
11.
12. # 1 "/usr/include/x86_64-linux-gnu/c++/5/bits/c++config.h" 1 3
13. # 194 "/usr/include/x86_64-linux-gnu/c++/5/bits/c++config.h" 3
14.
15. # 194 "/usr/include/x86_64-linux-gnu/c++/5/bits/c++config.h" 3
16. namespace std
17. {
18.     typedef long unsigned int size_t;
19.     typedef long int ptrdiff_t;
20.
21.     18162. int main() {
22.         18163.     cout << "Hello, world!" << endl;
23.         18164.     return 0;
24.     18165. }
25. # 216 "/usr/include/x86_64-linux-gnu/c++/5/bits/c++config.h" 3
26. namespace std
```

Рисунок 1.1 – Фрагмент файла проекта «Hello Word!» после препроцессинга

На данном этапе компиляции (команда **g++**) осуществляется преобразование кода, полученного на этапе препроцессинга и уже не содержащего директив, в *ассемблерный код*. Этот этап является промежуточным шагом между высокоуровневым языком и машинным (бинарным) кодом, и формирует представление машинного кода, доступное для понимания человеком. Чтобы остановиться на этапе формирования ассемблерного кода следует подать команду с ключом **-S**:

```
$ g++ -S driver.cpp -o driver.s
```

Соответствующий ассемблерный код можно проанализировать в выходном файле **driver.s** (см. рис.1.2).



```

▼ driver.s
.file    "driver.cpp"
.local   _ZStL8__ioinit
.comm    _ZStL8__ioinit,1,1
.section    .rodata
.LC0:
.string   "Hello, world!"
.text
.globl    main
.type     main, @function
main:
.LFB1021:
.cfi_startproc
pushq    %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq     %rsp, %rbp
.cfi_def_cfa_register 6
movl     $.LC0, %esi
movl     $_ZSt4cout, %edi
call     _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc
movl     $_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_, %
esi
movq     %rax, %rdi
call     _ZNSolsEPFRSoS_E
movl     $0, %eax
popq     %rbp
.cfi_def_cfa 7, 8
ret
.cfi_endproc
.LFE1021:
.size     main, .-main
.type     _Z41__static_initialization_and_destruction_0ii, @function
_Z41__static_initialization_and_destruction_0ii:
.LFB1030:

```

Рисунок 1.3 - Фрагмент ассемблерного кода файла проекта «Hello Word!»

Поскольку процессоры понимают только двоичные команды, команды, на следующем этапе следует перевести ассемблерный код в бинарный (машинный). Это делается с помощью специальной программы, называемой ассемблером (**as**), которая сопоставляет по специальной таблице текст на языке ассемблера в числовые команды в двоичном представлении.

Команда консоли для ассемблирования кода:

```
$ as driver.s -o driver.o
```

Результат такого преобразования в машинный код сохраняется в *объектном файле* проекта (наименование **driver.o**). Таким образом, объектный файл –это записанный в виде байтов или слов двоичный код, который еще не связан вместе с другими кусками проекта в конечную последовательность команд и данных, представляющих вместе выполняемую программу.

Чтобы не компилировать данный код снова, объектный код может того или иного модуля сохраняться в отдельном файле и может формировать объект так называемой *статической библиотеки* проекта.

Совокупность объектных файлов проекта следует объединить вместе выполняемую последовательность двоичных команд при помощи операции компоновки программы. Соединение объектных файлов в единый исполняемый файл осуществляет специальная программа-компоновщик (редактор связей, линкер):

```
$ g++ driver.o -o driver
```

Таким же путем при необходимости можно добавить в проект другие объектные файлы и библиотеки.

Для понимания процесса формирования исполняемой программ следует рассмотреть способ организации файловой системы на основе специальной структуры данных, представляющей собой таблицу символов, хранящаяся в самих объектных файлах. В этой структуре хранятся имена переменных, функций, классов, объектов и т.д., где каждому идентификатору (символу) соотносится его тип, область видимости. Кроме этого для того, чтобы компоновщик смог в дальнейшем построить связи между данными среди множества других объектных файлов и создать единый исполняемый файл, в таблице символов записаны адреса ссылок на данные и процедуры в других объектных файлах.

На последнем этапе формирования исполняемого машиной кода вызывается другая специальная программа, называемая загрузчиком, которая

также может добавить в код специальные модули сформированных фрагментов служебных программ, хранящихся в файловой системе - *динамических библиотек*.

Результат размещения программы в оперативной памяти (ОЗУ) программ можно просмотреть, применив специальные программы дизассемблирования (например, IDA, Sourcer, Hiew, Beye, HT editor, Hacker Disassembler Engine, CADt, Objdump, Radare2 и другие). Пример просмотра содержимого ОЗУ приведен на рисунке 1.4. На экране слева с шагом в 16 ячеек отображаются адреса памяти, в середине – их содержимое в шестнадцатеричном формате, а справа – расшифровка этих значений, соответствующая представлению содержимого ячеек памяти в коде ASCII.

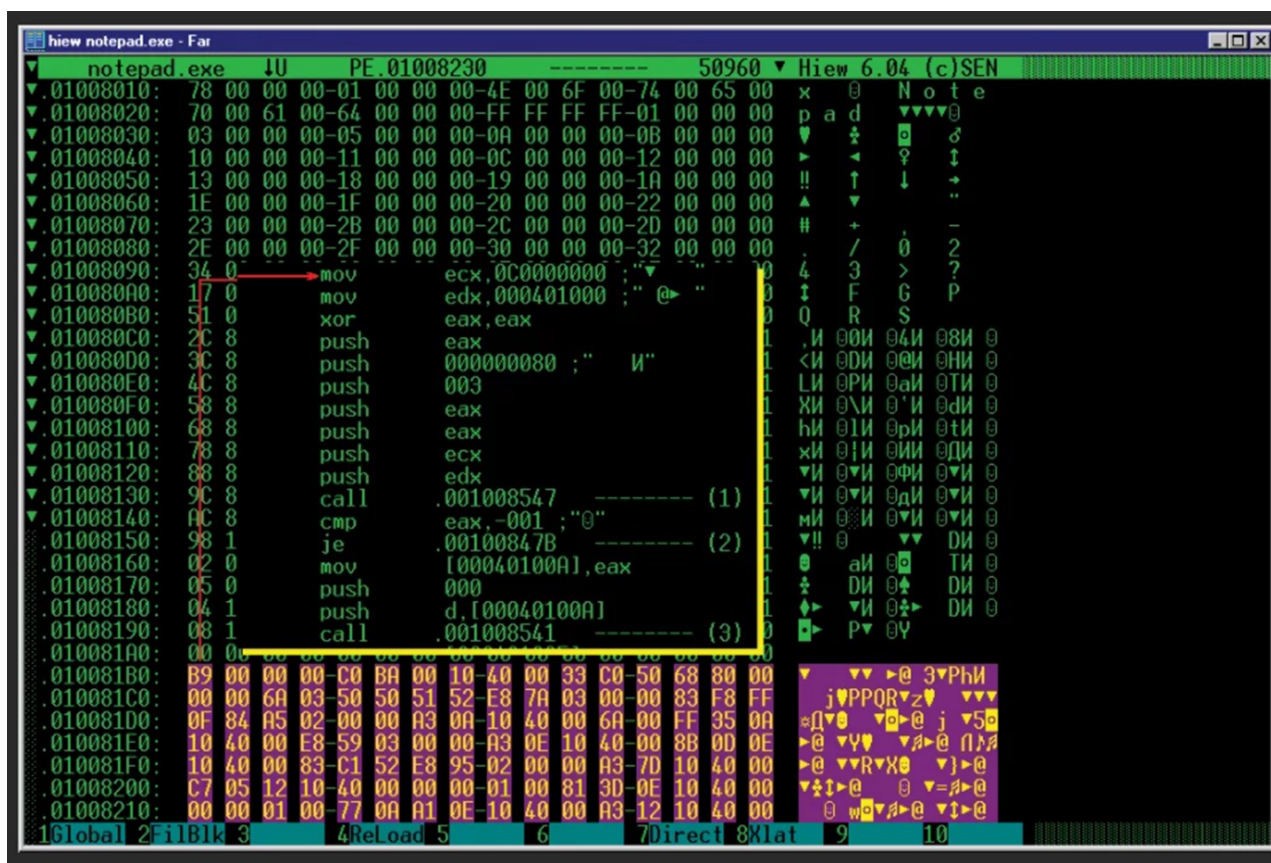


Рисунок 1.4 – Просмотр содержимого оперативной памяти. Выделен участок памяти в режиме дизассемблирования машинного кода

Запуск на исполнение программы «Hello, word!» с консоли:

```
$ ./driver
// Hello, world!
```

Таким образом, весь процесс трансляции кода, написанного в любом текстовом редакторе с соблюдением правил синтаксиса языка С или С++ можно представить в виде последовательности выполнения команд преобразования, что иллюстрируется рисунком 1.6.

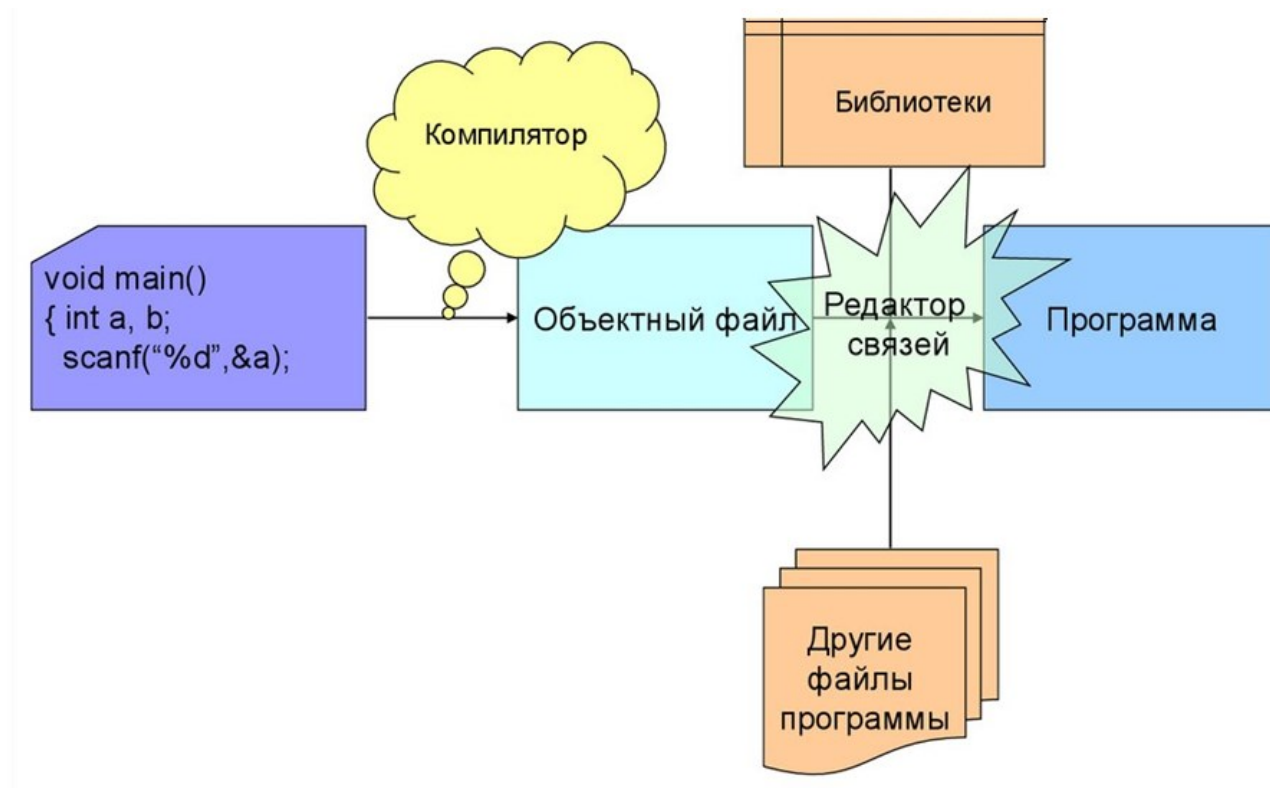


Рисунок 1.6 – Формирование исполняемого файла и кода на С

В общем случае для создания исполняемого файла необходимо подавать каждый раз соответствующие команды, что не удобно при многократной отладке программы. Для облегчения этой процедуры целесообразно сохранить повторяющую последовательность команд в виде специально скриптового командного файла - makefile, представляющего собой также программу, написанную с соблюдением соответствующих правил синтаксиса и интерпретируемой последовательно операционной системой компьютера.

Скриптовый файл с именем «makefile» представляет собой утилиту для автоматизации преобразования файлов из одной формы в другую.

Синтаксис makefile:

```

# каждой команде должен предшествовать отступ
<цели>: <реквизиты>
    <команда #1>
    ...
    
```

<команда #n>

Этот файл должен находиться в корне рабочей директории проекта и его содержание формируется из набора правил, которые в свою очередь описывают выполняемую операцию, реквизиты для выполнения правила и объектов преобразования и команды, выполняющими данные преобразования.

То есть, как правило make это ответы на три вопроса:

{Из чего делаем? (реквизиты)} ---> [Как делаем? (команды)] ---> {Что делаем? (цели)}

Пример. Для компиляции программы «Hello, Word!»:

```
/*
 * main.c
 */
#include <stdio.h>
int main()
{
    printf("Hello World!\n");
    return 0;
}
```

Следует создать написать в текстовом редакторе и сохранить в той же папке makefile:

```
hello: main.c
    gcc -o hello main.c
```

В этом файле записано правило, содержащее цель – **hello**, реквизит – **main.c** и команду – **gcc -o hello main.c**. Для компиляции проекта нужно в консоли рабочего каталога указать команду **main**, которая будет понята операционной системой и выполнена в режиме интерпретации, начиная с первого правила.

Для профессионального программирования на практике часто применяются специальные программы - оболочки представляющие собой разветвленные интегрированные среды разработки программного обеспечения.

В практикуме будет использован компилятор Visual C++ 10.0, входящий с пакет программ Visual Studio от компании Micro\$oft. Visual Studio включает в себя редактор исходного кода и встроенный отладчик, который может работать так с текстовым так и с машинным кодом, а также ряд других полезных инструментов для создания графических интерфейсов, редактирования web-текстов, классов и баз данных.

Все программы данной лабораторной работы запускаются в консольном окне. Программой называется конечная последовательность предписаний, сформулированных с помощью определённых синтаксических конструкций и

служащая для выполнения каких-либо действий ЭВМ. Для перевода программы с алгоритмического языка в машинный код используются специальные программы-трансляторы: интерпретаторы и компиляторы. Интерпретатор обрабатывает программу пользователя поэлементно: преобразует в машинный код и сразу же его исполняет, т.е. программа выполняется ЭВМ только в среде интерпретатора. С помощью компилятора создаются файлы программы (exe-файлы), которые могут самостоятельно выполняться в среде операционной системы ЭВМ. Для запуска среды программирования щёлкните дважды по ярлыку Microsoft Visual Studio 2010.

При первом запуске среды программирования появится окно выбора параметров среды (рис.1.7), в котором следует выбрать пункт «Параметры разработки Visual C++».

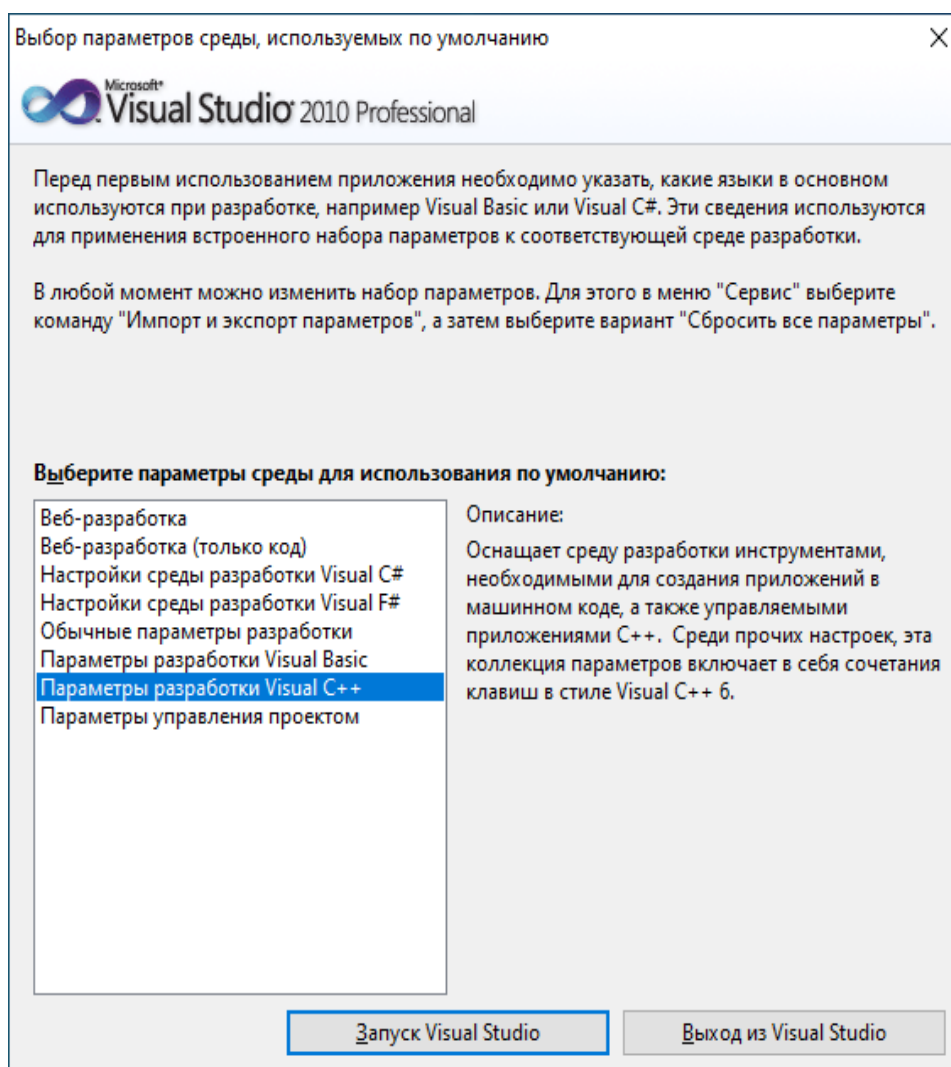


Рисунок 1.7 - Выбор параметров проекта

После запуска программы откроется окно начальной страницы проекта C++ (см. рис.1.8), в котором следует выбрать пункт «Создать проект».

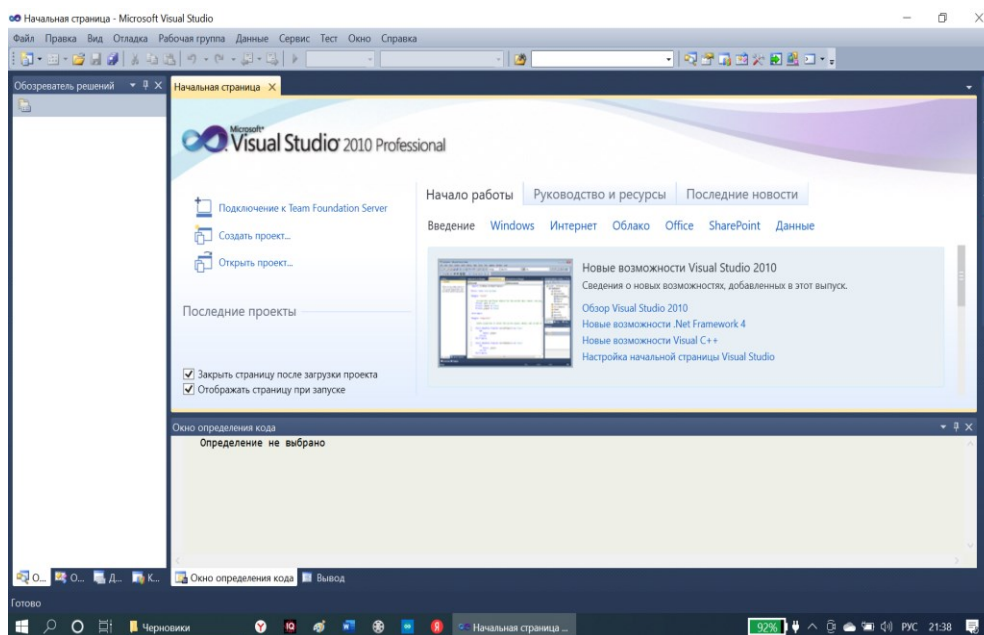


Рисунок 1.8 - Окно начальной страницы проекта

Упражнение 1.1. В качестве примера напомним первую программу Ex_1_HelloWorld, выводящую в консоль сообщение «**Hello World!**» (см. рис.1.9). Выберите опцию «Консольное приложение Win 32» (Win32 Console Application). Укажите путь к папке проекта, в поле Project name наберите имя проекта и щёлкните ОК. Установите флажок «Создать каталог для решения» и нажмите «Ок».

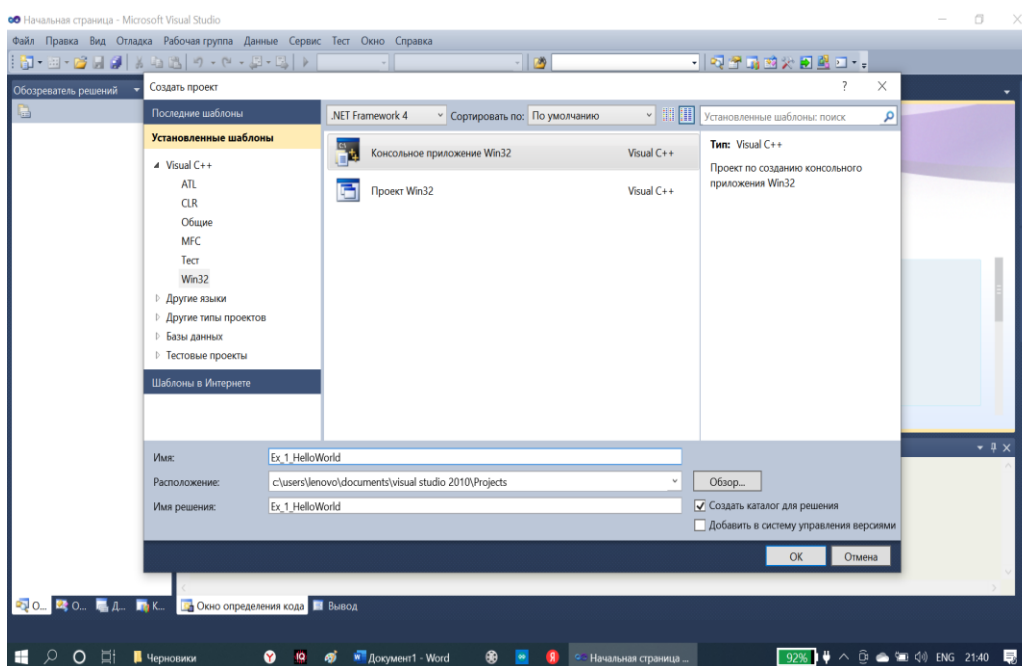


Рисунок 1.8 - Задание имени консольного приложения

В открывшейся вкладке мастера приложений (см. рис.1.9) выберите переключатель «Консольное приложение» и нажмите кнопку «Готово».

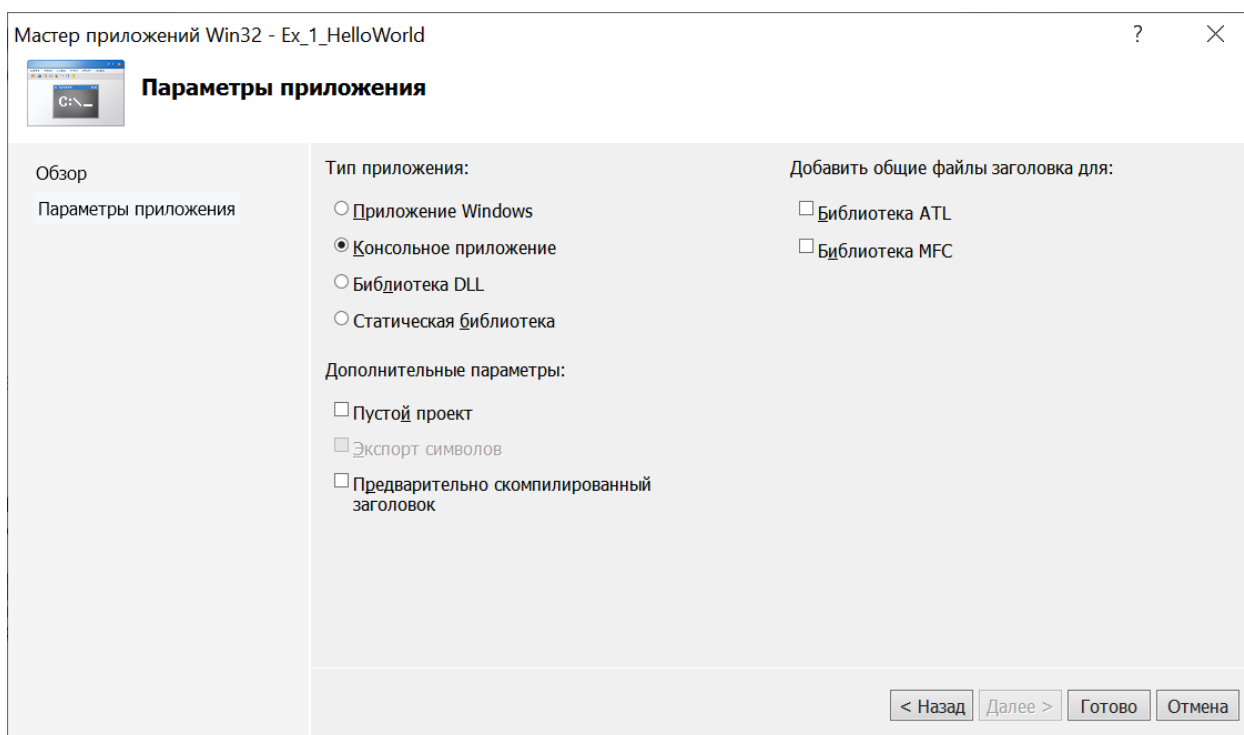


Рисунок 1.9 - Окно мастера приложений

В результате появится главное рабочее окно для редактирования кода и управления проектом (см. рис.1.10).

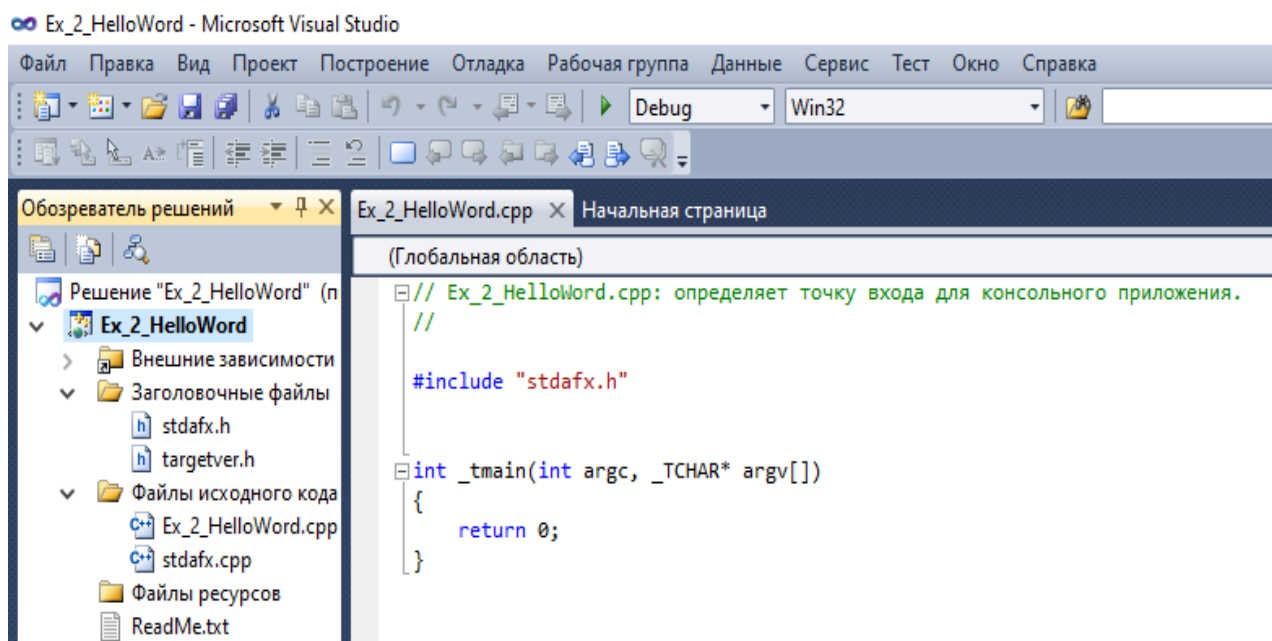


Рисунок 1.10 - Рабочее окно проекта

Изучите назначение функций горизонтального меню, наводя на них курсор мыши. Обратите внимание на открывающиеся выпадающие списки команд и элементов управления.

Введите программный код, изображенный на рис.1.11. и сохраните его.

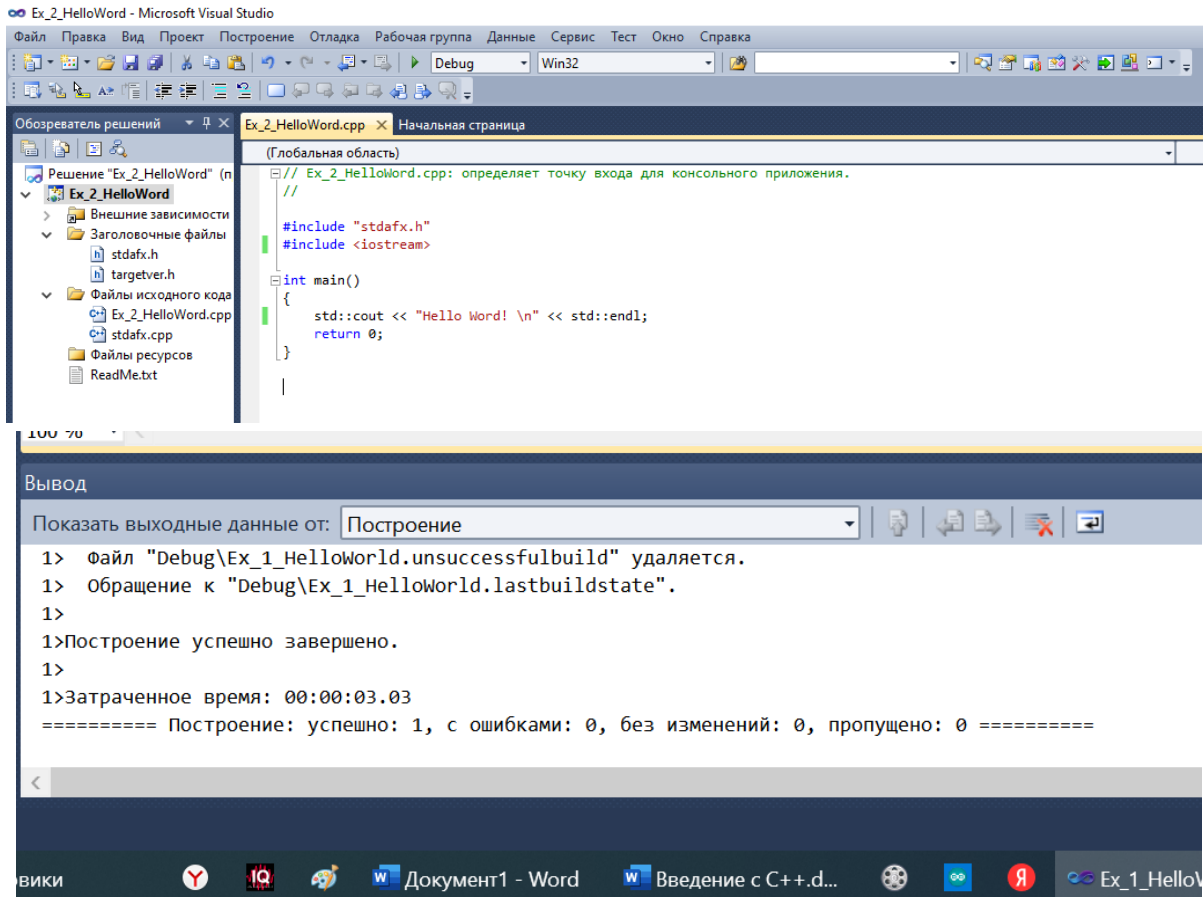


Рисунок 1.11 - Редактирование кода программы «HelloWorld»

В этом программном коде команда **#include <iostream>** обеспечивает подключение файла ввода/вывода сообщений в консольном окне. Функция **main()** вызывается автоматически при запуске программы и должна присутствовать в каждой программе, т.е. в ней содержится основной код программы (внутри фигурных скобок { }).

Команда **cout <<** выводит данные на экран. Управляющий символ **\n** обозначает разрыв строки. Команда **endl** означает конец строки (**end of line**). Кроме разрыва строки существуют следующие управляющие символы, которые можно вставить в выводимый текст: **\t** – табуляция, **\"** – двойная кавычка, **\'** – одинарная кавычка, **\?** – знак вопроса, **** – обратный слеш, **\n** – новая строка (разрыв строки). Примечания в коде обозначаются двойным прямым (правым)

слешем `//` (действует до конца строки), либо текстом в несколько строк, заключенным между правыми слешами со звездочками: `/* текст */`.

```
#include <iostream> /* директива включения файла
ввода/вывода сообщений */

int main() // функция вызова основного содержания
программы

{

std::cout << "Hello World! \n";

return 0; /* значение, возвращаемое функцией. Для main()
- это условность */

}
```

`std` – название пространства имен, в котором определено имя потока стандартного вывода на консоль `cout`.

Для компиляции программы выберите меню «**Построить решение (Build** → пункт **Build название_программы.exe)**» или нажмите **F7** (см. рис. 1.12).

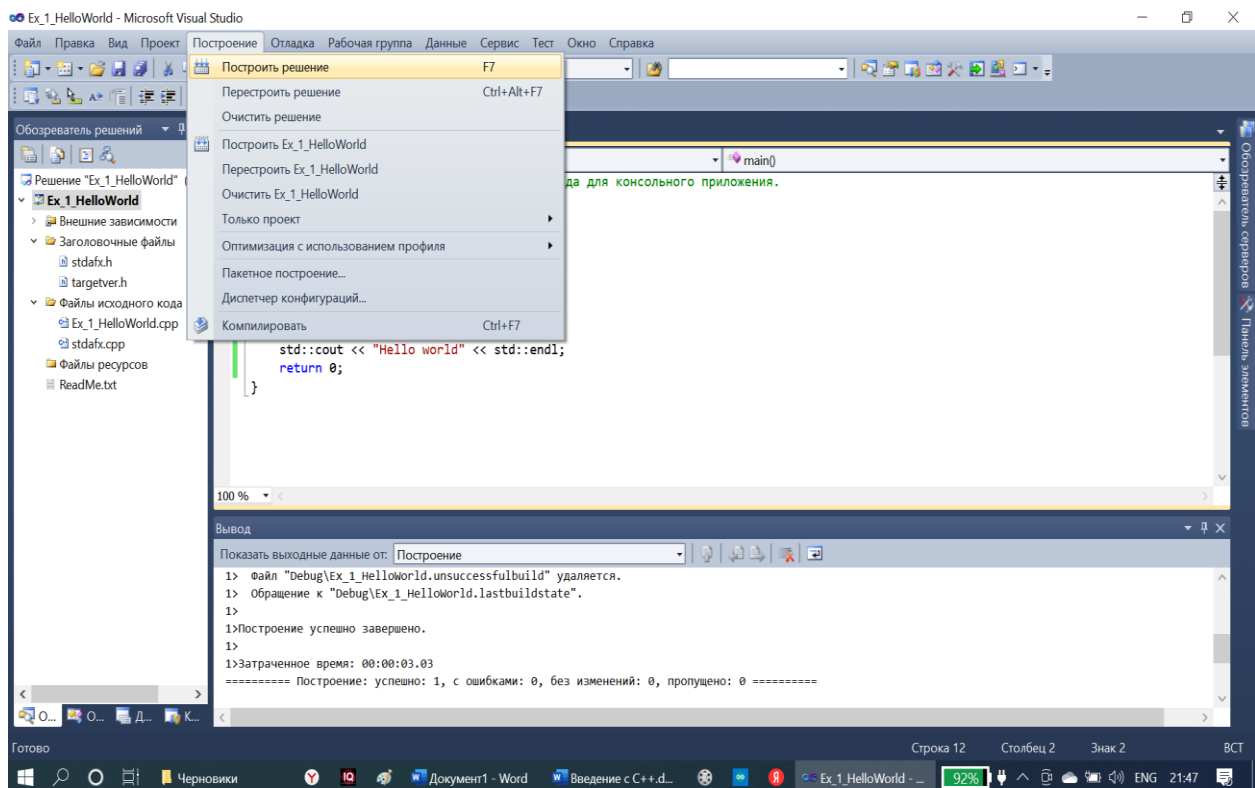


Рисунок 1.12 - Компиляция кода программы

Если имеются ошибки, то исправьте их. При отсутствии ошибок запустите программу в консольном приложении: меню **Build** → пункт **Execute** название_программы.exe (CTRL+F5) (см. рис. 1.12).

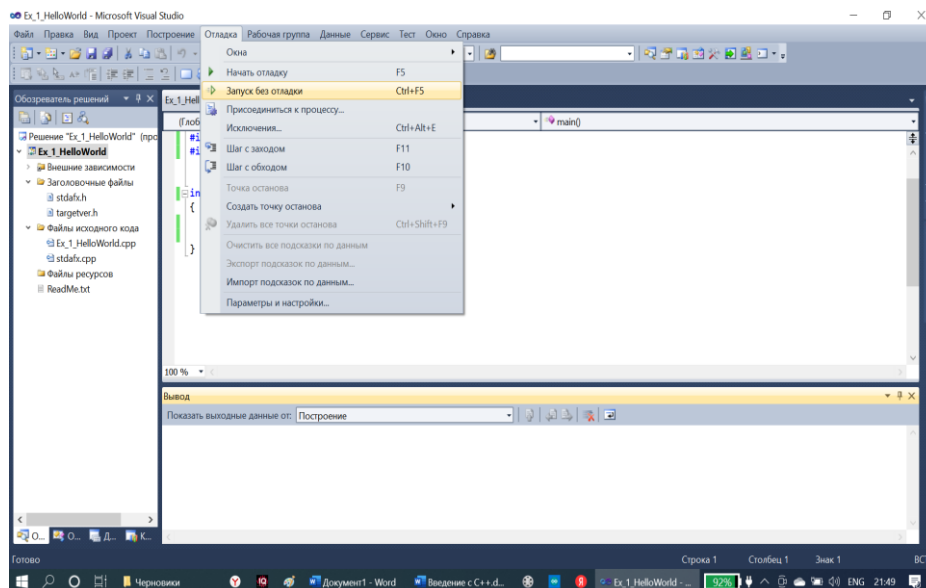


Рисунок 1.12. Запуск программы на выполнение

В результате откроется окно консоли Win32 с результатами выполняемой программы (см. рис.1.13).

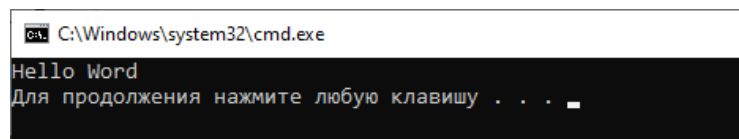


Рисунок 1.13 - Окно консоли Win32 выполнения программы «HelloWorld»

Закройте окно консоли и сохраните код проекта (см. рис. 1.14). По умолчанию проект сохраняется в рабочей папке. Файл с кодом программы имеет расширение *.sln.

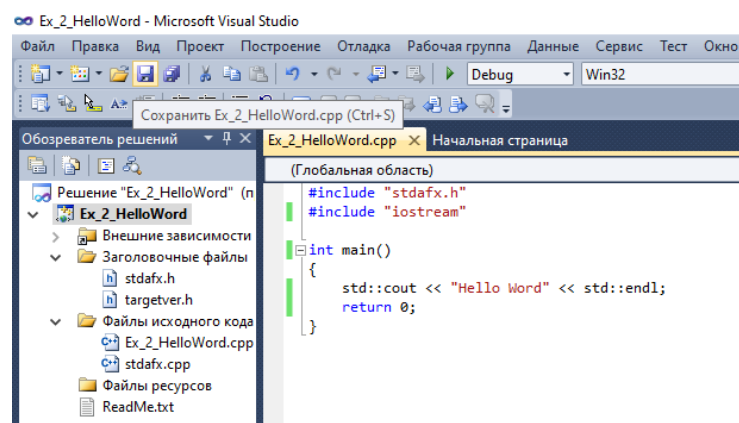


Рисунок 1.14- Сохранение проекта

Если требуется открыть уже существующий проект, то следует в после вызова среды проектирования выбрать пункт «Открытие проекта» и в открывшемся окне выбрать указать файл с кодом проекта так, как это показано на примере рис.1.15.

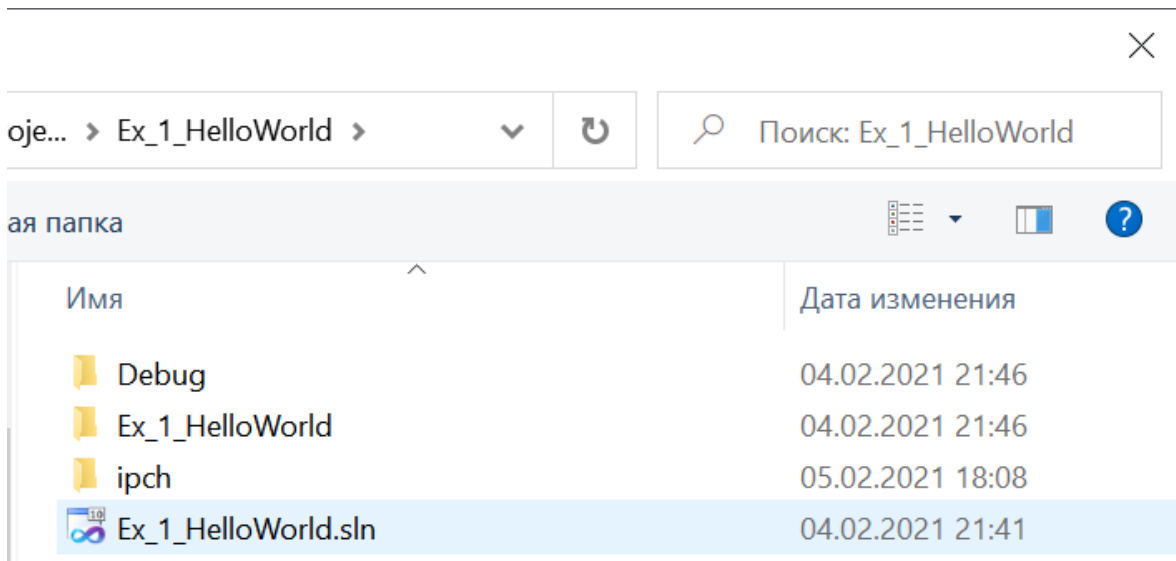


Рисунок 1.15 - Проводник с файлом проекта Ex_1_HelloWorld.sln

Упражнение 1.2. Следующим пунктом работы является ознакомление с типами данных и функциями в C++. Рассмотрите на практике пример работы программы:

```
// ·Ex_2_Demka.cpp: ·определяет ·точку ·входа ·для ·
консольного ·приложения.¶
¶
#include "stdafx.h"¶
#include <iostream>¶
¶
void Demka() ·// ·функция, ·не ·возвращающая ·никакого ·
значения, ·и ·выводящая ·на ·экран ·фразу ·«Demo ·function» ·¶
{ ·¶
→ std::cout << "Demo ·funktion ·\n"; ·// ·тело ·функции ··¶
} ¶
¶
int main() ¶
{ ¶
std::cout << "In ·Main! ·" << 1 << std::endl; ¶
Demka(); ·// ·вызов ·функции ·в ·теле ·основной ·программы ·¶
std::cout << "Back ·in ·main ·\n"; ·¶
return 0; ¶
} ¶
```

Программа выполняется по строкам, пока не встретится вызов функции. Это приводит к передаче управления функции. После выполнения функции управление возвращается строке программы, следующей за строкой вызова

функции. Функция возвращает значение или ничего не возвращает. Для последнего случая есть специальный тип функции **void**, которая не возвращает значение из функции.

Команда `cout << "In Main!" << 1 << endl;` позволяет вывести вслед за фразой «In Main! » значение **1** на экран.

Как можно видеть на данном примере, функция состоит из заголовка и тела. Заголовок содержит тип возвращаемого значения, имя функции и параметры. В языке C++ можно присваивать значения только самим данным и нельзя - типам данных. Например, запись: `int = 5;` неверна, правильно будет записать: `int x = 5`. Параметры передают в функцию значения определённых типов.

Типичной ситуацией при разработке программ, является наличие в коде синтаксических ошибок, что обнаруживаются при построении проекта. Пример сообщения об ошибке приведен на рис.1.16.

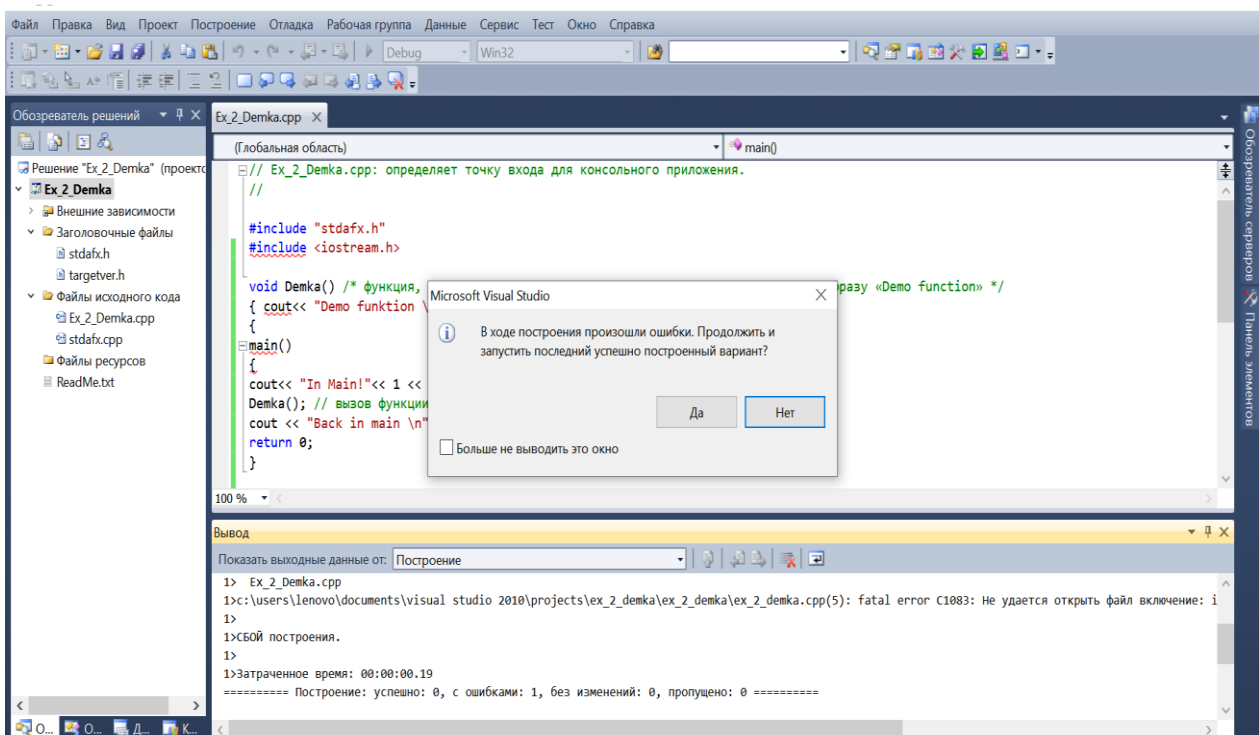


Рисунок 1.16- Окно программы с сообщением об ошибке при компиляции

В этом случае следует вызвать инструмент индикации ошибки (см. рис.1.17) и внимательно проанализировать информацию панели сообщений об ошибках, указывающих тип ошибки и строку кода, которая вызвала эту ошибку.

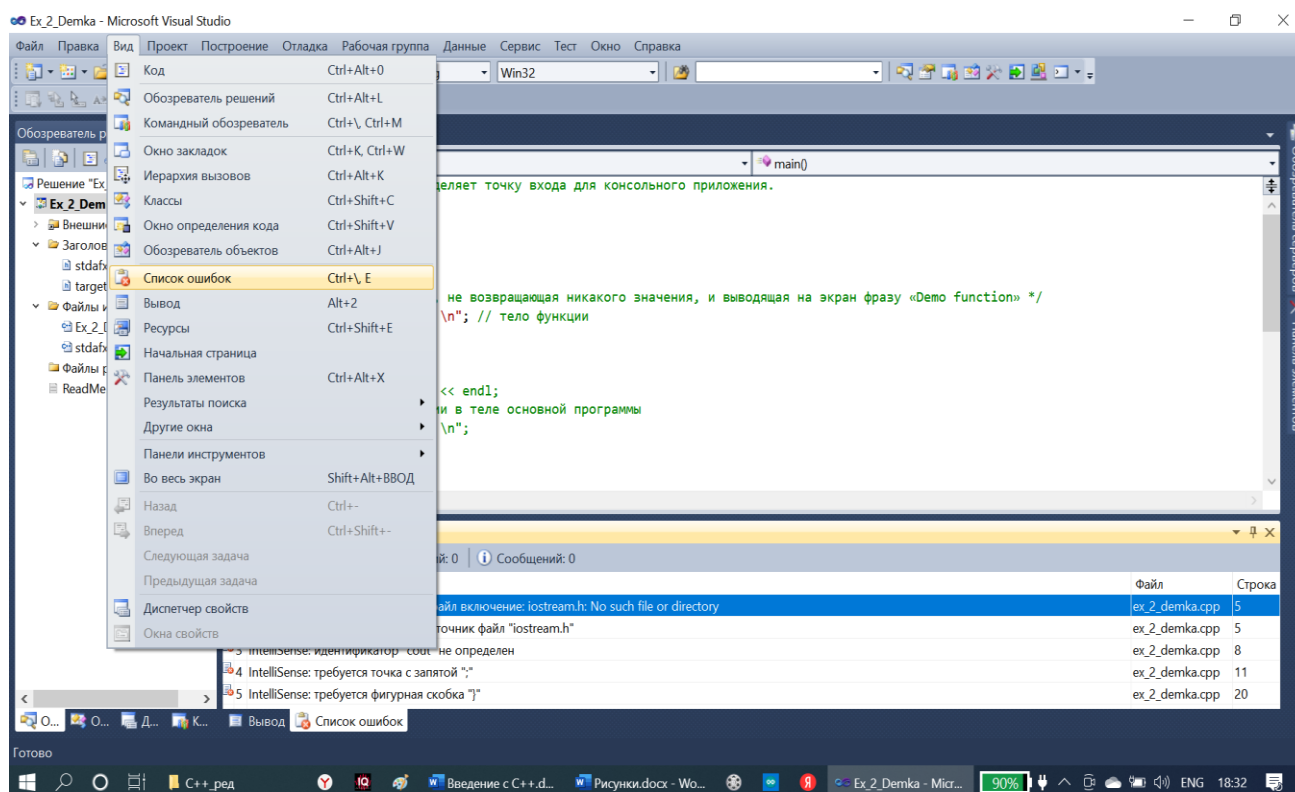


Рисунок 1.17. Вызов списка ошибок

Пример, иллюстрирующий такую ситуацию, приведен на рис.1.18.

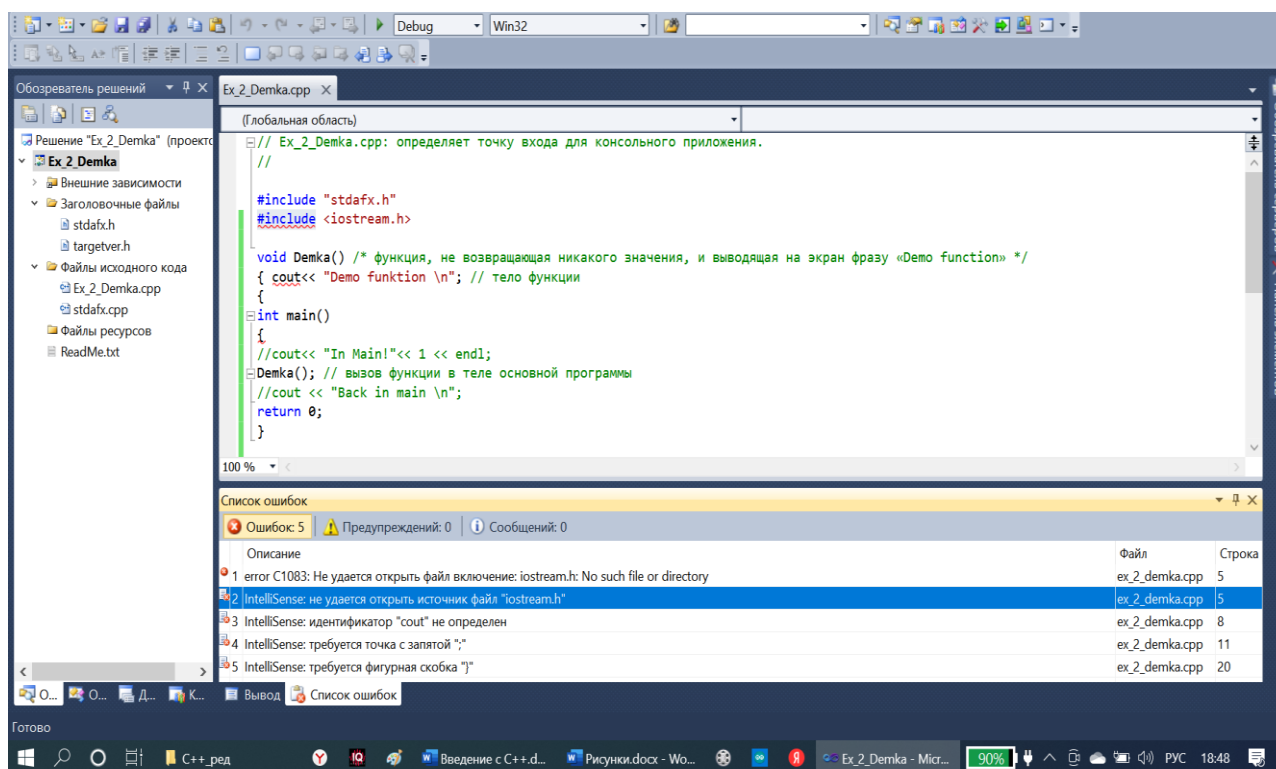


Рисунок 1.18 - Список ошибок и индикация ошибки 2

Упражнение 1.3. Для понимания работы с функциями, выполните пример программы, приведённой ниже:

```
// Ex_3_ABCSum.cpp: определяет точку входа для
консольного приложения.
//

#include "stdafx.h"
#include <iostream>

int Add(int x, int y) // int - целочисленный тип значения
{
    return (x+y); // return - команда возврата значения из
функции
}

int main()
{
    int a,b,c; // создание целочисленных переменных a,b,c
    std::cout << "Enter value of x: \n";
    std::cin >> a; // ввод с клавиатуры значения переменной a
    std::cout << "Enter value of y: \n";
    std::cin >> b;
    c=Add(a,b); // вызов функции Add с аргументами a,b
    std::cout << "Total sum of " <<a<<" and "<< b << " is: "
<< c << std::endl;
    std::cout << "Size of variable c is: "<< sizeof(c) << "
bytes \n";
    // вывод на экран текста и значений
    return 0;
}
```

Заголовок функции -: **int Add(int x, int y)** . Эта функция содержит два параметра **x** и **y** целочисленного типа **int**, и возвращает также значение типа **int**. Согласно синтаксису языка C, описание (тело) функции должно быть описано между фигурными скобками и содержать обязательную команду **return**, которая возвращает во вне функции значение указанного типа. Как видно из текста, в данной программе создаётся три целочисленных переменных, причем двум из них значения присваиваются с клавиатуры (оператор **cin>>**), а третьей переменной присваивается возвращённое из функции значение **c=Add(a,b)**, где **a** и **b** уже являются аргументами, т.е. параметрами, имеющими конкретные значения.

Поименованная ячейка памяти для хранения данных, значения которых могут изменяться, называется переменной. Для ее создания следует указать тип и имя, причем можно сразу присвоить ей значение. Эта операция называется инициализацией.

Пример, в созданы три переменные целочисленного типа, причем первая и третья инициализированы значениями 3 и 40.

```
int MyAge=3, yourAge, hisAge=40;
```

Размер переменной в байтах позволяет определить команда **sizeof()**. Переменные других типов, применяющиеся в синтаксисе С представлены в табл.1.1.

Таблица 1.1 Основные типы данных в С++

Тип данных	Размер в байтах	Значение
char	1	256 символов
float	4	1.2e-38 ... 3.4e38
double	8	2.2e-308 ... 1.8e308
bool	1	true или false
unsigned short int	2	0...65535
short int	2	-32768...32767
unsigned long int	4	0...4294967295
long int	4	-2147483648...2147483647

Чтобы не писать длинные названия типов можно создать псевдоним типа данных командой **typedef**. Пример, в котором переменная с именем **Width** типа **US** инициализирована значением 5:

```
typedef unsigned short int US;
```

и далее в программе создаём переменные с новым типом:

```
US Width=5;
```

Константой называется поименованная ячейка памяти для хранения не изменяющихся данных. Константа вводится ключевым словом **const**.

Пример, в котором создаётся целочисленная константа **students** со значением 15:

```
const int students = 15;
```


Часто в программе требуется создание перечислимого типа данных (константы перечисления). В таком случае используется ключевое слово **enum**.

Пример задания перечислимого типа данных с именем **COLOR**, состоящий из указанных в фигурных скобках значений:

```
enum COLOR {Red, Blue, Green, White, Black};
```

При этом **Red** по умолчанию присваивается значение **0**, **Blue** – **1**, и т.д.

Упражнение 1.4. Напишите и запустите на выполнение программу:

```
// Ex_4_Enum.cpp: определяет точку входа для консольного
// приложения.

#include "stdafx.h"
#include <iostream>

int main()
{
    enum Days {Monday=1, Tuesday, Wednesday, Thursday,
    Friday, Saturday, Sunday};
    int a; // переменная проекта
    std::cout << "Enter a day (1-7): \n";
    std::cin >> a; // ввод с клавиатуры номера дня

    if (a == Sunday || a == Saturday)
        std::cout << "\n It's a WeekEnds!\n";
    else
        std::cout << "\n OK. Job'll not wait \n";

    return 0;
}
```

Для перечислимого типа **Days** данные заданы. Значение переменной проекта вводится с клавиатуры. Соответственно, происходит ветвление программы. Поскольку принято считать понедельник первым днём недели, а не нулевым, значение для **Monday** инициализировано значением **1**. При этом автоматически для других переменных перечислимого типа автоматически сдвинулись на единицу.

2. ОСНОВНЫЕ ОПЕРАТОРЫ C

В C++, как и во всех других языках, действуют правила приоритета одних операторов по отношению к другим, поэтому, чтобы избежать ошибок вычисления и логических операциях, рекомендуется применять блоки и скобки.

Оператор присваивания: =.

Структурная единица – блок (фигурные скобки) { } .

Математические операторы: сложение **+**, вычитание **–**, умножение *****, целочисленное деление **/** и деление по модулю **%**.

Операторы инкремента и декремента :

Инкремент (декремент) бывает префиксный и постфиксный.

Пример префиксного инкремента:

```
int a = ++x; /*если x=5, то сначала увеличиваем x (т.е. x=6), а затем присваиваем её переменной a (т.е. a=6) */
```

Пример постфиксного инкремента:

```
int b = x++; /*если x=5, то сначала присвоить b (т.е. b=5), а затем увеличить на единицу x (т.е. x=6) *
```

Операторы отношения: равенства =, больше >, меньше <, больше или равно >=, меньше или равно <=, не равно !=.

Логические операторы: И &&, ИЛИ ||, НЕ !.

Оператор присваивания: =.

Операторы условия «?» и ветвления if.

Оператор ветвления switch сравнивает выражение, указанное в скобках, со значениями, приводимыми после ключевого слова **case**. В случае совпадения выполняются операторы, следующие после двоеточия до оператора **break** с последующим выходом из **switch**. Это позволяет осуществлять ветвление программы с количеством ветвей больше двух за один раз.

Операторы цикла while, do ... while и for.

Для изучения синтаксиса рекомендуется воспользоваться кратким изложением, приведенным в многочисленных учебниках и справочниках по C и C++, а также пособию <http://www.edamc.mirea.ru/files/kursoviciCpp.pdf>

Для закрепления материала и приобретения опыта программирования следует проделать следующие упражнения.

Упражнение 2.2. Пример программы по заданию условий: программа, которая вычисляет значение функции

$$Y = a + \frac{a}{a+1} + 2,5 * x,$$

где a – параметр, который зависит от аргумента x :

$$a = 3,5 * x \text{ если } x \geq -2 \text{ и } a = x + 1,5 \text{ если } x < -2$$

```

// Ex_5_conditions.cpp: определяет точку входа для
консольного приложения.
//

#include "stdafx.h"
#include <iostream>

int main()
{
    double x, a, answer;
    std::cout << "Enter a value of x: ";
    std::cin >> x;

    if (x >= -2)
    {
        a = 3.5*x;
        answer = a + (a/(a+1)) + 2.5*x;
    }
    else
    {
        a = x + 1.5;
        answer = a + (a/(a+1)) + 2.5*x;
    }

    std::cout << "Value of fuction Y is " << answer <<
    std::endl;
    return 0;
}

```

Упражнение 2.3. Создайте самостоятельно программу вычисления значения функции в соответствии с вводимым значением переменной x в соответствии с вашим вариантом (см. табл. 2.1).

Таблица 2.1. Задания для упражнения по программированию условий.

№	$Y =$	$a =$
1	$Y = \frac{2a^2 + 7a - 2}{x - 2,5} + x^2$	$a = \begin{cases} x + 1,5x^3 + e^{x+1}; (-2 \leq x \leq 5) \\ 1,5 + 4,5x; (5 < x \leq 7,5) \\ x + 1; (x > 7,5) \\ \text{не определено } (x < 2) \end{cases}$
2	$Y = \frac{2}{x} + a^3$	$a = \begin{cases} 2 + 2x; (x \leq 4) \\ 3; (4 < x \leq 5) \\ x + 1; (x > 5) \end{cases}$

Продолжение таблицы 2.1

3	$Y = \frac{2,2a + 3,2a^2 - 2}{x - 1,5}$	$a = \begin{cases} 2 \sin x + 4 \cos^2 x^2 ; (-1 \leq x \leq 1) \\ 3,5 + x ; (1 < x \leq 5) \\ -4,6 ; (x > 5) \\ \text{не определено } (x < -1) \end{cases}$
4	$Y = \frac{x + a^2(2,2 - x)^2}{x + 1}$	$a = \begin{cases} 5 + 2x ; (x \leq 3) \\ 4 ; (3 < x \leq 7) \\ x - 1 ; (x > 7) \end{cases}$
5	$Y = 3x + a^2 - \frac{x}{2ax}$	$a = \begin{cases} \sin x + 2 \cos x ; \left(-\frac{\pi}{2} \leq x \leq \frac{\pi}{2}\right) \\ \frac{x - 2,5}{2 + x} ; \text{для остальных } x \end{cases}$
6	$Y = 7,3 - \frac{a}{1 + x} + ax^3$	$a = \begin{cases} 2x + 1 ; (x \leq 2) \\ \sqrt{x + 3} ; (x > 2) \end{cases}$
7	$Y = a + 2x + \frac{3x^2 + a}{a + x}$	$a = \begin{cases} 3,2x + 1 ; (x < 3,14) \\ x \sin x ; (x = 3,14) \\ x ; (x > 3,14) \end{cases}$
8	$Y = a^3 + 2^x + \frac{2}{x}$	$a = \begin{cases} x^2 + \frac{4}{\sin x + 2} ; \left(-\frac{\pi}{2} \leq x \leq \frac{\pi}{2}\right) \\ 5 ; \left(\frac{\pi}{2} < x \leq 2,5\right) \\ x + 1 ; (x > 2,5) \\ \text{не определено } (x < \frac{\pi}{2}) \end{cases}$
9	$Y = a + 2,8x + \frac{x + a}{3 - x}$	$a = \begin{cases} 4x + 2,5 ; (x \leq 2,5) \\ 1,5x + 8 ; (x > 2,5) \end{cases}$
10	$Y = a + \frac{a}{a + 1} + 2,5x$	$a = \begin{cases} 3,5x ; (x \leq -2) \\ x + 1,5 ; (x > 2) \end{cases}$
11	$Y = 2,5a^2 + \sqrt{a + x}$	$a = \begin{cases} 5 ; (x < 3) \\ 8 + 1,5x ; (x \geq 3) \end{cases}$
12	$Y = a - \frac{x^2}{a}$	$a = \begin{cases} 3,5x ; (x < 4) \\ x - 5 ; (x \geq 4) \end{cases}$
13	$Y = \frac{2 + x}{x} + a^2 - 3x$	$a = \begin{cases} 3,7x ; (-6 < x \leq -3) \\ x^2 + 3x - 3 ; (-3 < x \leq 8) \\ \text{не определено для остальных } x \end{cases}$

Примечания:

1) если в задании имеются тригонометрические функции и возведение в степень, то необходимо в начале программы подключить стандартную библиотеку математических операций, записав **#include <math.h>** после **#include <iostream.h>.**;

2) возведение числа **a** в степень **b** осуществляется оператором **pow(a,b)**.

Упражнение 2.4. Проверьте на компьютере, что выполняет функция, предназначенная для реализации множественной инициализации и приращения:

```
#include <iostream.h>
int main()
{
    for (int i=0, j=0; i<3; i++, j++)
        cout <<
        "i: " << i <<" j: " << j << endl;    return 0;
}
```

3. ФУНКЦИИ

Подпрограмма, которая манипулирует с данными и возвращает некоторое значение формирует специальную структуру, которая называется функцией. Каждая программа на C++ имеет как минимум одну функцию **main()**, которая при запуске вызывается автоматически. Функция **main()** может вызывать другие функции, которые могут вызывать следующие функции и т.д.

Существует два вида функций: функции, определяемые пользователем и встроенные (стандартные), которые являются составной частью пакета компилятора. Для использования функции в программе требуется, чтобы функция сначала была объявлена (прототип функции), а затем определена (тело функции). Существует три способа объявления функции:

1) запись прототипа функции в файл, а затем с помощью команды **#include имя_файла** включить его в свою программу при компиляции;

2) запись прототипа функции в файле, в котором эта функция и используется;

3) определение функции одновременно с её объявлением.

Пример объявления функции:

```
int      FindArea      (int length, int width);
```

сначала указывается тип возвращаемого значения **int**, затем имя функции **FindArea**, затем в скобках типы и имена параметров функции через запятую. Значения параметров можно инициализировать в объявлении прототипа. Например:

```
int Area (int W=25, int H);.
```

Допускается задание прототипа функции без имён параметров, а только указывая их типы, например:

```
long Area (int, int);
```

Определение функции состоит из заголовка и тела функции. В заголовке функции указывается тип возвращаемого значения, имя функции и в скобках - типы и имена параметров через запятую. Обратите внимание что в конце заголовка функции не ставится точка с запятой (;) как это обычно принято в C++. Далее открывается фигурная скобка и записывается тело функции, которое определяет то, что данная функция выполняет. Оператор **return** возвращает значение функции. Необходимо следить, чтобы тип возвращаемого значения соответствовал типу, указанному в объявлении. После закрывающей тело функции фигурной скобки также отсутствует точка с запятой.

Пример:

```
int FindArea (int length, int width  
{  
    return (length*width);  
}
```

В функцию можно не только передавать значения переменных, но и объявлять переменные внутри тела функции. Эти переменные существуют только внутри самой функции и называются локальными. Когда выполнение программы передаётся обратно из функции к основному коду, локальные переменные удаляются из памяти. Глобальные же переменные имеют глобальную область видимости и доступны из любой точки программы.

Возврат значения из функции осуществляется оператором **return**. Например, **return (x>5)** будет возвращать **false**, если **x** не больше или равно 5, или **true**, если **x** больше 5. Функция может содержать несколько операторов **return**.

Пример:

```
int Doubler (int origin)  
{  
    if (origin <= 1000)  
        return origin*2;
```

```
else
return -1;
}
```

Полиморфизм функций или перезагрузка функций в языке C++ подразумевает возможность создания нескольких функций с одинаковым именем. В этом случае функции должны отличаться друг от друга типом параметров и/или их количеством.

Внутри функции нельзя определять другую функцию. В случае, если функция вызывает саму себя, то имеет место **рекурсия**.

Упражнение 3.1. Создание функции, переводящей температуру из шкалы по Фаренгейту в градусы Цельсия.

```
// Ex_6_Fareng.cpp: определяет точку входа для
консольного приложения.
//

#include "stdafx.h"
#include <iostream>

float Convert(float); // задание прототипа функции

int main()
{
float TempFar;
float TempCel;
std::cout<< "Enter the Temperature in Fahrenheit: ";
std::cin >> TempFar;
TempCel = Convert (TempFar);
std::cout << "\nThe Temperature in Celsius is: ";
std::cout << TempCel << std::endl;
return 0;
}

float Convert(float Far) /* заголовок функции. Обратите
внимание на отсутствие точки с запятой в заголовке
функции ;*/
{
float Cel; // тело функции
Cel = ((Far - 32) * 5)/9; // тело функции
return Cel; // тело функции
} //в конце строки тоже нет точки с запятой
```

Упражнение 3.2. Напишите программу, использующую рекурсию для вычисления чисел, образующих последовательность чисел 1, 1, 2, 3, 5, 8, 13, 21, 34 ... и т.д. В этом ряде, называемом рядом Фибоначчи первые два числа ряда равны единице, а каждый последующий член ряда является суммой двух предыдущих. : 1, 1, 2, 3, 5, 8, 13, 21, 34 ... и т.д.

```
// Ex_7_Fibonacci.cpp: определяет точку входа для
консольного приложения.
//

#include "stdafx.h"
#include <iostream> // пример рекурсии

int fib(int n); /* прототип функции вычисления члена ряда
Фибоначчи */

int main()
{
    int n, answer;
    std::cout << "Enter number to find: ";
    std::cin >> n;
    std::cout << "\n\n";
    answer = fib(n);
    std::cout << answer << " is the " << n << "-th member of
Fibonacci series\n";
    return 0;
}

int fib(int n) /*заголовок функции вычисления члена ряда
Фибоначчи */
{
    if(n < 3)
        return 1;
    else
        return( fib(n-2) + fib(n-1) );
}
```

Упражнение 3.3. Пример программирования циклов (вариант 1 задания к упражнению 8): используя циклы, функции и рекурсию вычислить с точностью $\epsilon = 0,0001$ сумму

$$\sum_{i=1}^{\infty} (-1)^{i+1} \frac{x^{2(i-1)}}{2(i-1)!},$$

которая является разложением в ряд Тейлора функции $\cos(x)$. Положить, что x изменяется в диапазоне $[0.1, 1]$ с шагом 0.1.

Соответствующая программа может выглядеть так:

```
// Ex_8_Loops.cpp: определяет точку входа для консольного
приложения.
//

#include "stdafx.h"
#include <iostream>
#include <math.h>

double factorial(int); /* прототип функции вычисления
факториала */

int main()
{
    double a,b,h,eps,S;
    a=0.1; b=1; h=0.1; eps=1e-4;

    for (double i=a; i<=b; i+=h)
    {
        int j=1;
        double st,f;
        S=0;
        do
        {
            f = factorial(2*(j-1));
            st = pow(-1.0, j + 1) * pow(i, 2*(j-1)) / f;
            j++;
            S += st; //тоже, что S=S+st
        } while(st >= eps);

        std::cout << "Sum of series is: " << S << " Value of
cos(" << i << ") is: " << cos(i) << std::endl;
    }

    --

return 0;
}

double factorial(int a)
{
    if(a == 0 || a == 1) /* заметьте, что здесь стоит знак
равенства ==, а не присвоить = */
        return 1;
    else
        return (a*factorial(a-1)); /* для вычисления
использована рекурсия */
}
```


Задания к упражнению 3.3.

Вычислить сумму S при изменении аргумента x в указанном диапазоне $[a,b]$ с шагом h ., прекращая суммирование, когда очередной член станет меньше 0.0001. Для проверки в каждой точке вычислите также функцию $y=f(x)$, представляющую собой аналитическое выражение ряда. Варианты заданий приведены в табл.3.1.

Таблица 3.1 Варианты заданий к упражнению 3.3.

№ вар.	$S=$	$y=$	$a=$	$b=$	$h=$
1	$\sum_{i=1}^{\infty} i^2 \cdot x^{i-1}$	$\frac{1+x}{(1-x)^3}$	0	0,9	0,1
2	$\sum_{i=1}^{\infty} (2i-1)^2 \cdot x^{i-1}$	$\frac{1+6x+x^2}{(1-x)^3}$	0	0,9	0,1
3	$\sum_{i=1}^{\infty} (-1)^{i+1} \cdot \frac{x^{2i-1}}{(2i-1)!}$	$\sin x$	0,1	0,9	0,1
4	$\sum_{i=1}^{\infty} (-1)^{i+1} \cdot \frac{x^{2(i-1)}}{2(i-1)!}$	$\cos x$	0,1	1	0,1
5	$\sum_{i=0}^{\infty} \frac{(2x)^i}{i!}$	e^{2x}	0,1	1	0,05
6	$\sum_{i=0}^{\infty} \frac{x^{2i}}{(2i)!}$	$chx = \frac{e^x + e^{-x}}{2}$	0,1	1	0,05
7	$\sum_{i=0}^{\infty} \frac{x^{(2i+1)}}{(2i+1)!}$	$shx = \frac{e^x - e^{-x}}{2}$	0,1	1	0,05
8	$\sum_{i=0}^{\infty} (-1)^i \frac{x^{(2i+1)}}{(2i+1)}$	$arctgx$	0,1	0,5	0,05
9	$\sum_{i=0}^{\infty} (2i+1) \cdot x^i$	$\frac{1+x}{(1-x)^2}$	0	0,9	0,05
10	$\sum_{i=1}^{\infty} (-1)^i \frac{\cos(ix)}{i^2}$	$\frac{x^2 - \frac{\pi^2}{3}}{4}$	-0,5	0,5	0,1

Продолжение таблицы 3.1.

11	$\sum_{i=1}^{\infty} (-1)^{i+1} \cdot \frac{x^i}{i}$	$\ln(1+x)$	-0,5	0,5	0,1
12	$\sum_{i=1}^{\infty} (-1)^{i+1} \cdot \frac{x^{2i}}{2i(2i-1)}$	$x \cdot \arctg(x) - \ln\sqrt{1+x^2}$	0,1	0,8	0,05
13	$\sum_{i=0}^{\infty} \frac{(2i+1)x^{2i}}{i!}$	$(1+2x^2) \cdot e^{x^2}$	0,1	1	0,1
14	$\sum_{i=0}^{\infty} \frac{\left(\frac{x-1}{x+1}\right)^{2i+1}}{2i+1}$	$\frac{\ln x}{2}$	0,2	1	0,08

4. ПОНЯТИЕ ОБ ОБЪЕКТНО-ОРИЕНТИРОВАННОМ ПРОГРАММИРОВАНИИ. БАЗОВЫЕ КЛАССЫ

Класс – это набор переменных различных типов, скомбинированный с набором связанных функций. Инкапсуляцией называется соединение переменных и функций в один объект класса. Части программы, работающие с классом, могут использовать ваш объект, не беспокоясь о том, что находится в нём или как оно работает.

Переменные в классе принадлежат только своему объекту и называются ещё переменными-членами или данными класса.

Функции в классе выполняют действия с переменными-членами и называются методами класса.

Например, объявление класса **Cat** делается так:

```

Class Cat
{
    int Age;    // Данные класса
    int Weight; // Данные класса
    void Meaw(); // Метод класса
};

```

Объект нового типа определяется также, как и любая переменная.

Пример:

```

int Skill;
Cat Murzik;

```

После этой записи **Murzik** является объектом класса **Cat**.

Доступ к членам объекта (как переменным, так и методам) осуществляется оператором прямого доступа (**.**). Чтобы присвоить число переменной-члену **Weight** объекта **Murzik** нужно записать:

```
Murzik.Weight = 12;
```

Аналогично, чтобы вызвать метод **Meaw**, запишем: **Murzik.Meaw()** ;

Определенные значения должны присваиваются объектам, а не классам.

Поэтому запись: **Cat Age=5;** - ошибочна. Синтаксически правильно будет:

```
Cat Murzik;
```

```
Murzik.Age = 5;
```

При объявлении класса используются ключевые слова **public** (открытый) и **private** (закрытый), определяющие порядок доступа к членам класса. Все члены класса (переменные и методы) являются закрытыми по умолчанию. К закрытым членам можно получить доступ только с помощью методов самого класса. Открытые члены доступны для всех других функций программы. Например, при записи:

```
Cat Barsik;
```

```
Barsik.Age = 3;
```

компилятор выдаст ошибку, т.к. нельзя обращаться к закрытым переменным объекта. Выходом из положения является объявление класса **Cat** открытым:

```
Class Cat  
{  
public:  
int Age;  
int Weight;  
void Meaw() ;  
}
```

Общая стратегия использования классов состоит в том, чтобы оставлять переменные-члены класса закрытыми. Это позволяет инкапсулировать данные внутри класса. Доступ следует открывать только методам класса, которые обеспечивают доступ к закрытым данным. Эти методы можно вызвать из любого места в программе для возвращения или установления значения закрытых данных. Такая стратегия позволяет скрыть детали хранения данных в объектах, снабжая пользователя методами их использования. Можно модернизировать способы хранения и обработки данных внутри класса, не переписывая при этом методы доступа и вызова их во внешнем программном коде.

Пример:

```
class Cat  
{
```

```
public:
    int GetAge();
    void SetAge(int Age);
    void Meow();
private:
    int itsAge;
};
```

Чтобы установить возраст кота Мурзика теперь нужно записать:

```
Cat Murzik;
Murzik.SetAge(5);
```

Как и для функций, после объявления класса нужно дать определение класса, т.е. определить, что он выполняет.

Упражнение 4.1. Пример определения класса.

```
// Ex_9_class.cpp: определяет точку входа для консольного
приложения.
//

#include "stdafx.h"
#include <iostream> // пример класса

class Cat // объявляем класс Cat
{
public:
    int GetAge(); // метод доступа
    void SetAge(int age); // метод доступа
    void Meow(); // обычный метод
private:
    int itsAge; // переменная-член (данные)
метода
};

int Cat::GetAge() /* определение метода
GetAge, который возвращает значение переменной-члена
метода */
{
    return itsAge;
}

void Cat::SetAge(int age) /* определение метода
SetAge, который инициализирует переменную-член метода */
{
    itsAge = age;
}
```

```
void Cat::Meow()           // метод выводит на экран тест
сообщения
{
    std::cout << "Meow\n";
}

int main()
{
    Cat Murzik;           // создаём объект Мурзик
    Murzik.SetAge(5);      // устанавливаем его возраст
    Murzik.Meow();
    /* заставляем его мяукнуть и выводим сообщение: */
    Murzik.Meow();
    std::cout << "Murzik has a " << Murzik.GetAge() << "
years old.\n";
    return 0;
}
```

Для инициализации и разрушения переменных-членов класса используются специальные функции: конструктор и деструктор. Существуют специальные методы класса, имена которых совпадают с именем самого класса и которые обязательно должны присутствовать в коде и добавляются компилятором по умолчанию. Это конструктор и деструктор класса. Причём конструктор будет пустым, если инициализация переменных – членов класса не проводится. Деструктор служит для удаления из памяти отработавших объектов, а также других действий при модификации объекта или класса. Он не принимает никаких аргументов и не возвращает никаких значений. Объявляется с помощью значка тильда (~), например: **~Cat()** .

Упражнение 4.2. Пример, дополняющий предыдущий и позволяющий понять роль конструктора:

```
// Ex_10_class2.cpp: определяет точку входа для
консольного приложения.
//

#include "stdafx.h"
#include <iostream> // классы и конструктор/деструктор

class Cat           // объявляем класс Cat
{
```

```

class Cat // объявляем класс Cat
{
public:
    Cat(int initAge); // конструктор
    ~Cat(); // деструктор
    int GetAge();
    void SetAge(int age);
    void Meow();
private:
    int itsAge; // переменная-член метода
};

Cat::Cat(int initAge) /* конструктор инициализирует
переменную-член значением */
{
    itsAge = initAge;
}

Cat::~~Cat() // деструктор не выполняет действия
{
}

int Cat::GetAge()
{
    return itsAge;
}

void Cat::SetAge(int age)
{
    itsAge = age;
}

void Cat::Meow()
{
    std::cout<< "Meow\n";
}

int main()
{
    Cat Murzik(5); /* создаём объект Мурзик с инициали-
зированной возрастом 5 */
    Murzik.Meow(); /*заставляем его мяукнуть и выводим
сообщение*/
    std::cout<< "Murzik has a " << Murzik.GetAge() <<"
years old.\n";
}

```

```

Murzik.Meow();
Murzik.SetAge(7); /* переустанавливаем его возраст и
выводим сообщение: */
std::cout<< "Now Murzik has a " << Murzik.GetAge()
<<" years new.\n";
return 0;
}

```

В языке C++ предусмотрена возможность объявить метод класса таким образом, что такому методу будет запрещено изменять значение переменных-членов класса. Для этого используется ключевое слово **const** после круглых скобок, но перед точкой с запятой. Пример:

```
void Function() const;
```

Использование **const** везде в объявлениях методов, если они не изменяют переменные объекта, позволяет лучше отслеживать ошибки. Если метод объявлен как **const**, а в его выполнении происходит изменение переменной-члена объекта, то компилятор выдаст ошибку.

Т.к. функция **GetAge()** приведенной выше программы просто возвращает текущее значение переменной **itsAge**, то правильнее будет записать: **int GetAge() const;**

Можно также помещать определение методов класса непосредственно в объявлении класса. Обратите внимание на отсутствие точки с запятой в этом случае после определения.

Пример:

```

class Cat
{
public:
int GetAge() {return itsAge;} /* определение в объявлении
void SetAge(int age); */
};

```

Можно строить сложные классы путём объявления более простых классов и последующего их включения в объявление сложного класса. Т.е. классы могут содержать другие классы в качестве переменных-членов.

Очень близкими родственниками ключевого слова **class** являются структуры, объявляемые ключевым словом **struct**. В C++ структура – это тот же класс, но с открытыми по умолчанию членами. Структуры перешли в C++ по наследству из языка C.

Механизм, позволяющий строить иерархии, в которых производные (последующие) классы получают элементы базовых (родительских) классов и могут их дополнять или изменять их свойства называется наследованием. Чтобы создать класс, используется ключевое слово **class**, после которого указывается имя нового класса, двоеточие тип объявления класса, а затем имя базового класса. Базовый класс должен быть объявлен ранее.

Пример:

```
class Grey : public Cat  
{ тело класса }
```

Члены класса, объявленные как **private**, недоступны для наследования. Если же их объявить как **protected**, то данные-члены класса станут доступными для всех производных классов, но недоступными для всех внешних классов.

Если создаётся объект производного класса, то сначала вызывается конструктор базового класса, а затем конструктор производного класса, который завершает создание объекта. При удалении объекта из памяти сначала вызывается деструктор производного класса, например: **~Grey()**, а затем деструктор базового класса, например: **~Cat()**.

Объект производного класса имеет доступ ко всем методам базового класса, а также ко всем своим методам. Кроме того, методы базового класса могут быть замещены. Под замещением понимается изменение действий функции, обладающей тем же именем и тем же типом и набором аргументов. В случае замещения одного из методов остальные варианты этого метода (т.е. имеющие то же имя, но другие типы и количество аргументов) оказываются скрытыми. Если вы хотите их использовать в производном классе, то их также нужно будет заместить в этом классе.

Обратиться напрямую к методу базового класса можно так:

```
Murzik.Cat :: Meow( );
```

т.е.: имя объекта производного класса, *точка*, имя базового класса, *два двоеточия* и имя вызываемого метода.

Упражнение 4.3. Пример, иллюстрирующий изученные выше понятия языка C++: программа, вычисления объёма полого цилиндра при разных исходных данных.

```
// Ex_10_Cylinder.cpp: определяет точку входа для  
консольного приложения.  
//  
  
#include "stdafx.h"
```



```
#include <iostream> /* подключение стандартной библиотеки
ввода/вывода */
#include <math.h> /* подключение стандартной библиотеки
математических операций для функции возведения в степень
b числа a: pow(a,b) */

enum choice {Size=1, Volume, Thickness, Quit}; /*
перечислимый тип переменной */

class Cylinder // объявление класса
{
public:
    Cylinder(float R1, float R2, float H); // конструктор
    ~Cylinder() {} // деструктор
    float GetExtRadius() const {return itsR1;} /* метод
доступа класса сразу с его определением */
    float GetIntRadius() const {return itsR2;} /* метод
доступа класса сразу с его определением */
    float GetHeight() const {return itsH;} /* метод доступа
класса сразу с его определением */
    float GetVolume() const; // метод расчета объёма
цилиндра
    float GetThickness() const {return (itsR1 - itsR2);} /*
метод класса сразу с его определением */
    void SetSize (float newR1, float newR2, float newH); /*
метод задания размеров цилиндра */
private: // закрытые члены класса (переменные)
    float itsR1;
    float itsR2;
    float itsH;
};

// определение методов класса:
Cylinder::Cylinder (float R1, float R2, float H)
/* определение конструктора*/
{
    itsR1 = R1;
```

```
    itsR2 = R2;
    itsH = H;
}

void Cylinder::SetSize(float newR1, float newR2, float
newH)
/* определение метода класса */
{
    itsR1 = newR1;
    itsR2 = newR2;
    itsH = newH;
}

float Cylinder::GetVolume() const
// определение метода класса
{
    float V;
    // формула расчёта объёма цилиндра
    V = 3.1415 * (pow(itsR1,2) - pow(itsR2,2)) * itsH;
    return V;
}

// прототипы используемых функций
int DoMenu(); // функция вывода меню на экран

// функция вызова метода расчёта объёма
void DoVolume(Cylinder);

// функция вызова метода расчёта толщины стенки^
void DoThickness(Cylinder);

int main() // основное содержание программы, функция main
{
    //инициализация конструктором параметров цилиндра
    Cylinder theCyl(0,0,0);
    int choice = Size;
    int fQuit = false;
```

```
while(!fQuit)
{
    choice = DoMenu(); /* функция DoMenu выводит Меню и
возвращает сделанный там выбор пункта */
    if (choice < Size || choice > Quit)
    {
        std::cout<<"\nInvalid choice. Try again.\n\n";
        continue;
    } /* переход на следующую итерацию тела цикла без
выполнения тела цикла */

    switch(choice)
    {
        case Size:
            float L1,L2,Hi;
            std::cout<<"\nEnter external radius of Cylinder:
";

            std::cin>> L1;
            std::cout<<"Enter internal radius of Cylinder: ";
            std::cin>> L2;
            std::cout<<"Enter height of Cylinder: ";
            std::cin>> Hi;
            theCyl.SetSize(L1, L2, Hi);
            break; /* выход из выполнения оператора switch */
        case Volume:
            DoVolume(theCyl);
            break;
        case Thickness:
            DoThickness(theCyl);
            break;
        case Quit:
            fQuit = true;
            std::cout<<"\nExiting...\n\n";
            break;
        default:
            std::cout<< "Error in choice!\n";
            fQuit = true;
    }
}
```

```
        break;
    } //конец switch
} // конец while
return 0;
}

// определение тел функций
int DoMenu() // тело функции вывода меню и выбора его
пункта
{
    int choice;
    std::cout<< "\n ***Menu***\n";
    std::cout<< "(1) Set Size\n";
    std::cout<< "(2) Count Volume\n";
    std::cout<< "(3) Count Thickness\n";
    std::cout<< "(4) Quit\n";
    std::cout<< "Choice a number from Menu: ";

    std::cin>> choice;
    return choice;
}

void DoVolume(Cylinder theCyl) /* тело функции вызова
метода расчёта объёма */
{
    std::cout <<"The Volume of Cylinder: " <<
theCyl.GetVolume() << std::endl;
}

void DoThickness(Cylinder theCyl) /* тело функции вызова
метода расчёта толщины стенки */
{
    std::cout << "The Thickness of Cylinder Wall: " <<
theCyl.GetThickness() << std::endl;
}
```

Упражнение 4.4. Самостоятельно разработать программу, позволяющую многократно вычислять некоторую физическую величину по заданной формуле и одновременно вычислять значения каких-либо параметров формулы. Варианты заданий приведены в табл. 1.4.

Таблица 4.1. Задания к упражнению 4.4.

№	Наименование	Формула	Параметр
	Объем:		
1	тора	$V = 2\pi^2 R r^2$	r
2	усеченного цилиндра	$V = \pi R^2 \frac{h_1 + h_2}{2}$	R
3	усеченного прямого конуса	$V = \frac{\pi h}{3} (R^2 + Rr + r^2)$	h
4	шарового сектора	$V = \frac{2\pi R^2 h}{3}$	R
5	шарового сегмента	$V = \frac{1}{3} \pi h (3R - h)$	h
	Центральный момент инерции		
6	полого цилиндра	$J_z = \frac{1}{2} (R^2 + r^2)$	$R - r$
7	полого шара	$J = \frac{2}{5} \frac{(R^5 - r^5)}{(R^3 - r^3)}$	$R - r$
8	шарового сектора	$J = \frac{h}{5} (3R - h)$	h
9	тора	$J_z = \left(R^2 + \frac{3}{4} r^2 \right)$	r
10	Импеданс последовательной цепи LRC	$Z_{\Sigma} = \sqrt{R^2 + \left(\omega L - \frac{1}{\omega C} \right)^2}$	R
11	Импеданс параллельной цепи LRC	$Z_{\parallel} = \frac{1}{\sqrt{\frac{1}{R^2} + \left(\omega C - \frac{1}{\omega L} \right)^2}}$	R
12	Емкость плоского конденсатора ($\epsilon_0 = 8,85 \cdot 10^{-12} \text{ Ф/м}$)	$C = \frac{\epsilon_0 \epsilon S}{d}$	ϵ
13	Сила отталкивания двух элементарных зарядов $e = 1,6 \cdot 10^{-29} \text{ Кл}$	$F = \frac{1}{4\pi\epsilon_0} \frac{e^2}{r^2}$	r

В конце упражнения следует оформить отчёт в виде текстового документа Word, который должен включать следующие пункты:

1. Формулировка задания с указанием номера варианта и исходных данных.
2. Блок-схема алгоритма программы.
3. Полный программный код с комментариями.
4. План тестирования программы.
5. Результаты тестирования и результаты работы программы.

5. УКАЗАТЕЛИ, ССЫЛКИ И МАССИВЫ

Переменная, в которой записан адрес ячейки памяти компьютера имеет специальное назначение, называется указателем (**pointer**) и широко применяется при программировании. Для получения адреса ячейки памяти применяется оператор «&».

Например, в результате выполнения кода:

```
int i = 10;  
cout << &i << endl;
```

на экран будет выведен адрес ячейки памяти, в которой хранится значение переменной **i**, равное 10.

В указателе содержится *адрес* переменной, а для получения доступа к *значению* переменной используется оператор разыменования *****.

Пример:

```
int yourAge;  
yourAge = *pAge;
```

в результате переменной **yourAge** присвоится значение, хранимое по адресу, содержащемуся в указателе **pAge**.

Если после наименования типа стоит символ *****, то это означает, что эта переменная является указателем.

Пример:

```
int howOld = 50;  
int* pAge = &howOld;
```

Т.е. символ ***** указывает, что операция должна производиться не над самим *адресом*, а над *значением*, сохранённым по адресу, который хранится в указателе. Компилятор по контексту программы определяет, какой именно оператор используется в данном случае.

Таким образом следует различать два написания:

тип * имя – это *объявление* указателя, а просто: *** имя** – это *значение*, хранящееся по адресу, содержащемуся в памяти под данным именем имя.

Инициализируйте указатель нулевым значением при объявлении, если заранее не известно для указания на какую переменную будет использоваться указатель, то его следует инициализировать нулевым значением. В противном случае могут возникнуть проблемы при работе программы.

Указатели используются:

- для размещения данных в свободных областях памяти и доступа к ним;
- для доступа к переменным и функциям классов;
- для передачи параметров в функции по ссылке;

Локальные переменные и параметры функций размещаются в стековой памяти. По завершению работы функции стек очищается. В результате все локальные переменные оказываются вне области видимости и их значения уничтожаются. В отличие от стека, динамическая память не очищается до завершения работы программы, поэтому её очищением должен заниматься сам программист. Выделенная память в динамической области не может использоваться, пока явно не будет освобождена. Доступ к данным в динамической памяти можно получить только из функций, в которых есть доступ к указателю, хранящему нужный адрес. Это позволяет жестко контролировать доступ к данным в динамической памяти.

Для выделения памяти в динамической области используется ключевое слово **new**. Затем указывают тип объекта, который будет размещаться в памяти. В качестве результата оператор **new** возвращает адрес выделенного фрагмента памяти. Этот адрес должен присваиваться указателю.

Один указатель можно вычитать из другого. Это может понадобиться при вычислении количества элементов массива, если эти указатели указывают на разные элементы массива.

Ссылка – это альтернативное имя данного объекта (по сути - псевдоним). Для объявления ссылки нужно указать тип объекта адресата, за которым следует оператор ссылки «&», а за ним – имя ссылки.

Оператор ссылки выглядит так же, как оператор взятия адреса, который используется для возвращения адреса при работе с указателями. Однако это *разные* операторы.

Если использовать ссылку для получения адреса, то она будет показывать тот же адрес, что и объект. Т.е. ссылка – полный синоним адресата (объекта), даже если применяется оператор адреса.

Упражнение 5.1. В качестве упражнения, иллюстрирующего сказанное, выполните программу:

```
// Ex_11_pointer1.cpp: определяет точку входа для консольного
приложения.
//

#include "stdafx.h"
#include <iostream>

int main()
{
    int intOne;
    int &rRef=intOne; // создание ссылки
    intOne=5;
    std::cout<< "intOne: " << intOne<< std::endl;
    std::cout<< "rRef: " << rRef<< std::endl;
    rRef=7;
    std::cout<< "intOne: " << intOne<< std::endl;
    std::cout<< "rRef: " << rRef<< std::endl;
    std::cout<< "&intOne: " << &intOne<< std::endl;
    std::cout<< "&rRef: " << &rRef<< std::endl;
    return 0;
}
```

Упражнение 5.2. Чтобы лучше понять, для чего нужны ссылки и указатели, выполните пример:

```
// Ex_12_pointer2.cpp: определяет точку входа для консольного
приложения.
//

#include "stdafx.h"
#include <iostream>

void swap (int x, int y); // прототип функции

int main()
{
```



```

int x=5, y=10;
std::cout << "Main.Before swap. x=" << x << " y="<< y <<"\n";
swap(x, y); // вызов функции в основной программе
std::cout << "Main.After swap. x=" << x << " y="<< y<<"\n";
return 0;
}

void swap(int x, int y)
{
    int temp; // создание промежуточной переменной
    std::cout << "Swap.Before swap. x=" << x << " y="<< y << std::endl;
    temp = x;
    x=y;
    y=temp;
    std::cout << "Swap.After swap. x=" << x << " y="<< y << std::endl;
}

```

Результаты работы программы будут такими:

```

Main. Before swap.  x=5  y=10
    Swap. Before swap.  x=5  y=10
    Swap. After swap.  x=10 y=5
Main. After swap.  x=5  y=10

```

В данном примере в функции значения *x* и *y*, поменялись местами, но их исходные значения не изменились. Таким образом в функцию отправляются копии значений аргументов, а сами аргументы защищены от изменений. Такая передача аргументов в функцию называется передачей **по значению**. Альтернативным способом работы с функцией является применение ссылок и указателей. В этом случае в стек помещается не значение объекта, а его адрес и все изменения в функции отражаются на объекте.

Упражнение 5.3. Пример использования указателей:

```

// Ex_13_pointer3.cpp: определяет точку входа для консольного
приложения.
//

#include "stdafx.h"
#include <iostream>

void swap (int *, int *); // аргументы - это указатели

```

```
int main()
{
    int x=5, y=10;
    std::cout << "Main.Before swap. x=" << x << " y=" << y << std::endl;
    swap(&x,&y); //передаются адреса переменных
    std::cout << "Main.After swap. x=" << x << " y=" << y << std::endl;
    return 0;
}

void swap(int* px, int* py)
{
    int temp;
    std::cout << "Swap.Before swap. *px=" << *px << " *py=" << *py <<
std::endl;
    temp = *px;          /* значение по адресу px (а в данном случае px
= &x, т.е. адрес переменной x ) присваивается temp */
    *px=*py;
    *py=temp;
    std::cout << "Swap.After swap. *px=" << *px << " *py=" << *py <<
std::endl;
}
```

Упражнение 5.4. Пример использования ссылок в функциях:

```
// Ex_2_4_pointer4.cpp: определяет точку входа для консольного
приложения.
//

#include "stdafx.h"
#include <iostream>

void swap (int &, int &); // аргументы - это ссылки

int main()
{
    int x=5, y=10;
    std::cout << "Main.Before swap. x=" << x << " y=" << y << std::endl;
```

```

swap(x,y); // передаются переменные x и y
std::cout << "Main.After swap. x=" << x << " y=" << y << std::endl;
return 0;
}

void swap(int& rx, int& ry)
{
    int temp;
    std::cout << "Swap.Before swap. rx=" << rx << " ry=" << ry << std::endl;
    temp = rx; // ссылка rx присваивается temp
    rx=ry;
    ry=temp;
    std::cout << "Swap.After swap. rx=" << rx << " ry=" << ry << std::endl;
}

```

Для указателей передаются в функцию адреса переменных (**&x**, **&y**), а для ссылок - сами переменные **x** и **y**. Таким образом, благодаря использованию ссылок функция может изменять исходные данные. При этом сам вызов функции ничем не отличается от обычного.

Массив – это последовательность элементов одного типа, имеющая общее имя. Доступ к каждому элементу осуществляется указанием индекса (номера) данного элемента. Имя массива является указателем на нулевой элемент данного массива.

Объявление массива:

тип имя [количество элементов];

Пример:

int Mas [30];

string Names [15];

Размерность массива, указанная в квадратных скобках – целое число, причем если объявлено [n] элементов, то они будут иметь номера начиная с нулевого, т.е. 0, 1, 2 ... n-1. Если при обращении к элементу массива указать номер вне промежутка 0, 1, ... n-1, то произойдёт выход за пределы массива. Компилятор этого не отслеживает, и это может приводить к ошибкам в работе программы.

Инициализация массива осуществляется списком значений:

int mas [3] = {82, 3, 18};

Можно не указывать размер массива при инициализации:

```
int mas [ ] = {100, 8, 36,1,0};
```

в этом случае компилятор сам присвоит размерность массиву исходя из списка инициализации.

Упражнение 5.5. Программа, которая заменяет в массиве из 10 элементов все элементы, большие 7, - на нули.

```
// Ex_15_array1.cpp: определяет точку входа для консольного
приложения.
//

#include "stdafx.h"
#include <iostream>

int main()
{
    int mas [10]; // объявляем массив из 10 элементов

    // Вводим данные в массив (целые величины)
    for (int i=0; i<10; i++)
    {
        std::cout << "Enter a value of " << i << " element: ";
        std::cin >> mas[i];
    }

    for (int j=0; j<10; j++)
    {
        if(mas[j]>7)
            mas [j] = 0;
    }

    std::cout<< "Result is: " << std::endl;

    for (int k=0; k<10; k++)
        std::cout << "Element number "<<k<<" has a value " <<mas[k]<<
std::endl;

    return 0;
}
```

6. БЛОК-СХЕМЫ АЛГОРИТМОВ

Алгоритм – последовательность действий, которую нужно выполнить, чтобы достичь требуемого результата. Более строго: **алгоритм** – конечная совокупность точно заданных правил решения произвольного класса задач или набор инструкций, описывающих порядок действий исполнителя для решения некоторой задачи. и систем.

Блок- схемы алгоритмов - см. ГОСТ 19.701-90 (ИСО 5807-85) Единая система программной документации (ЕСПД). Обозначения условные и правила выполнения.

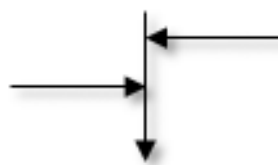
Некоторые правила и рекомендации построения схем

1. Допускается зеркально отображать символы и поворачивать их вокруг оси. В частности, запоминающие устройства с прямым доступом (таблицы на жестком диске) на схемах, как правило, поворачивают на 90° против часовой стрелки.

2. Большинство символов допускают задание внутри них текстовых пояснений. Если текст не помещается внутри символа, то лучше его приводить, используя комментарии.

3. Количество пересечений линий следует минимизировать. При этом считается, что пересекающиеся линии не имеют логической связи друг с другом. Другими словами, потоки данных или управления в местах пересечений не меняют своего направления.

4. Если две или более линий объединяются в одну, то место объединения должно быть смещено. Пример объединения потоков данных или управления:



5. Несколько выходов из символа решения следует показывать одним из следующих способов:

- несколькими линиями от данного символа к другим символам;
- одной линией от данного символа, которая затем разветвляется в соответствующее число потоков.

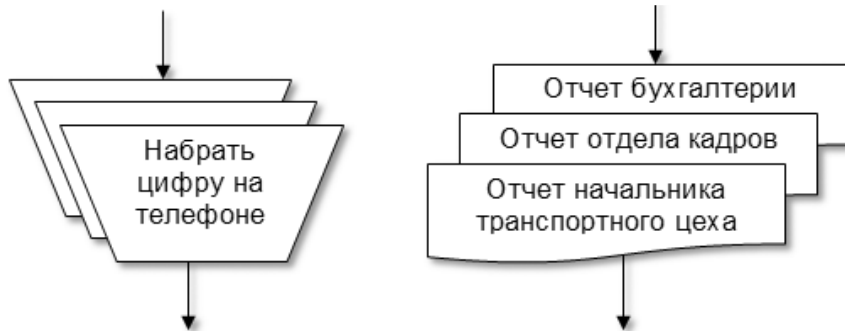
Примеры ветвления линий:



В случае ветвления каждый выход из символа должен сопровождаться либо записью условия (например, условие "Сравнить А и В", 3 выхода: $A > B$, $A < B$ и $A = B$), либо результата проверки условия, записанного внутри символа (например, условие " $A = B$ ", 2 выхода: Да и Нет).

Символ решения допускает более двух исходящих потоков.

6. Вместо одного символа с соответствующим текстом могут быть использованы несколько символов с перекрытием изображения, каждый из которых может содержать дополнительный текст (например, запись или посылка нескольким получателям, подготовка нескольких копий документа и т. д.). Примеры множественного обозначения символов:



Программа – непустое множество записанных в памяти (не обязательно последовательно) машинных команд (в двоичном коде), задающих действия, описывающих шаги работы алгоритма. Таким образом, программа – это запись алгоритма на языке машины.

Язык машины – набор всех возможных операций, выполняемых командами и допустимые форматы этих команд. Набор команд в общем случае может быть произвольным для данного типа ЭВМ (в современном понимании – процессора) и понимается специальным устройством – устройством управления с дешифратором команд, обеспечивающим функционирование ЭВМ как конечного автомата с ограниченным числом состояний. Каждая команда однозначно определяется своим кодом операции (в простейшем случае это просто номер команды).

Упражнение 6.1. Составьте блок-схему алгоритма решения и разработайте соответствующую программу по вариантам в табл. 6.1.

Замечание: для вычисления модуля используйте функцию **fabs(double)**; а для округления до целого числа функцию **ceil(double)**. Обе эти функции

возвращают значения типа **double** и для их использования необходимо подключить библиотеку **<math.h>**.

Таблица 6.1. Варианты для выполнения упражнения 6.1. Сгенерируйте целочисленную прямоугольную матрицу и определите:

1	Количество строк, не содержащих ни одного нулевого элемента.
2	Максимальное из чисел, встречающихся в заданной матрице более одного раза.
3	Количество столбцов, не содержащих ни одного нулевого элемента.
4	Переставляя строки матрицы, расположите их в соответствии с ростом характеристик считая, что характеристикой строки называется сумма её положительных чётных элементов.
5	Количество столбцов, содержащих хотя бы один нулевой элемент.
6	Номер строки, в которой находится самая длинная серия одинаковых элементов.
7	Произведение элементов в тех строках, которые не содержат отрицательных элементов.
8	Максимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы.
9	Сумму элементов в тех столбцах, которые не содержат отрицательных элементов.
10	Минимум среди сумм модулей элементов диагоналей, параллельных побочной диагонали матрицы.
11	Сумму элементов в тех строках, которые не содержат хотя бы один отрицательный элемент
12	Привести матрицу к треугольному виду.
13	Найти количество строк, среднее арифметическое элементов которых меньше заданной величины.
14	Найти номер первой из строк, содержащих хотя бы один положительный элемент.
15	Номер первого из столбцов, содержащих хотя бы один нулевой элемент.
16	Упорядочить строки по возрастанию количества одинаковых элементов в каждой строке.
17	Номер первого из столбцов, не содержащих ни одного отрицательного элемента.
18	Номер столбца, в котором находится самая длинная серия одинаковых элементов.

Продолжение таблицы 6.1.

19	Сумму элементов в тех строках, которые не содержат отрицательных элементов.
20	Минимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы.
21	Количество локальных минимумов заданной матрицы размером 10 на 10. Элемент матрицы называется локальным минимумом, если он строго меньше всех имеющихся у него соседей.
22	Сумму модулей элементов, расположенных выше главной диагонали.
23	Осуществить циклический сдвиг элементов прямоугольной матрицы на n элементов вправо или вниз (в зависимости от введенного режима). n может быть больше количества элементов в строке или столбце.
24	Соседями элемента A_{ij} в матрице назовём элементы A_{kl} , где $i-1 \leq k \leq i+1, j-1 \leq l \leq j+1, (k,l) \neq (i,j)$. Операция сглаживания матрицы даёт новую матрицу того же размера, каждый элемент которой получается, как среднее арифметическое имеющихся соседей соответствующего элемента исходной матрицы. Построить результат сглаживания заданной вещественной матрицы размером 10 на 10.
	В одномерном массиве, состоящем из n вещественных чисел, вычислить:
25	Сумму отрицательных элементов массива.
26	Произведение элементов массива, расположенных между максимальным и минимальным элементами.
27	Упорядочить элементы массива по убыванию.
20	Упорядочить элементы массива по возрастанию.

7. ПРОГРАММЫ ИССЛЕДОВАНИЯ ФУНКЦИЙ

Контрольное задание. Разработайте и протестируйте программу, выводящую в консоли меню, при выборе одного из пунктов которого над функцией в заданном интервале значений аргумента выполняются следующие операции:

1. Вывод значений аргумента и заданной функции $F(x)$ с шагом h , начиная от начального значения a до конечного значения аргумента b .
2. Вычисление корней уравнения $F(x)=0$ методом дихотомии с точностью 0,0001.
3. Вывод на экран количества экстремумов функции $F(x)$.
4. Вычисление интеграла функции методом прямоугольников с точностью 0,001 на отрезке между вторым и третьим корнем функции.

5. Вычисление интеграла функции методом трапеций с точностью 0,001 на отрезке между вторым и третьим корнем функции.
6. Вычисление методом Монте-Карло интеграла функции на отрезке между вторым и третьим корнем функции, задав не менее 100 точек для усреднения значения интеграла.

Указания по выполнению задания.

7.1. ОПРЕДЕЛЕНИЕ КОРНЕЙ УРАВНЕНИЯ МЕТОДОМ ДИХОТОМИИ

Пусть задано уравнение

$$F(x) = 0,$$

причём функция $F(x)$ непрерывна на отрезке $[a, b]$ и $F(a) * F(b) < 0$. Из этого следует, что на этом отрезке она пересекает ось абсцисс нечётное число раз. Пусть корень только один. Разделим отрезок $[a, b]$ пополам: $x = (a + b) / 2$. Далее возможны три случая:

- если $F(x) = 0$, тогда x есть искомый корень
- если $F(a) * F(x) > 0$, то перемена знака имеет место в правой половине отрезка и следует изменить левую границу отрезка $[a, b]$: $a = x$ и продолжить процесс деления пополам;
- если $F(a) * F(x) < 0$, то перемена знака функции имеет место в левой половине отрезка и следует изменить правую границу отрезка $[a, b]$, задав $b = x$, опять продолжить процесс деления пополам.

Процесс деления отрезка пополам следует завершить, когда выполнится одно из условий:

$$F(x) < eps, \text{ либо } b - a < eps,$$

где eps – заданная погрешность вычислений.

Отыскание корней уравнения следует оформить отдельной функцией.

Для определения интервалов изоляции корней необходимо просмотреть последовательно пары соседних значений функции при заданном шаге разбиения области задания аргумента функции. При выполнении условия отрицательности произведения соседних значений функции, следует обратиться к функции отыскания корня методом дихотомии.

7.2. ОПРЕДЕЛЕНИЕ КОЛИЧЕСТВА ЭКСТРЕМУМОВ

Экстремумом функции называют точку локального минимума или максимума функции. В этих точка производная функции обращается в ноль. Таким образом задача сводится к предыдущей: следует найти количество нулей

производной функции. Величину самой производной следует в данном случае вычислять приближенно как отношение приращения значения функции к приращению аргумента в данной точке при достаточно малом шаге приращения аргумента (не более 1% от интервала задания функции).

7.3 ВЫЧИСЛЕНИЕ ОПРЕДЕЛЕННЫХ ИНТЕГРАЛОВ

7.3.1 Метод прямоугольников

Отрезок интегрирования делится на n равных частей длины $h = (b - a)/n$. Обозначим:

$$\begin{aligned} x_0 &= a, \\ x_1 &= a + h, \\ &\dots \\ x_i &= a + i * h, \\ x_n &= b, \\ F(x_i) &= F_i. \end{aligned}$$

В каждом интервале разбиения площадь под функцией заменяется на прямоугольный отрезок $F(x_i) * h$. Чем меньше шаг h , тем точнее выражение для интеграла. Затем все площади суммируются, и получается значение интеграла функции:

$$\int_a^b F(x) dx = h \left(\sum_{i=1}^n F_i \right)$$

7.3.2 Метод трапеций

Отрезок интегрирования делится на n равных частей длины $h = (b - a)/n$. Обозначим аналогично предыдущему: $x_0 = a$, $x_1 = a + h$, ... $x_i = a + i * h$, $x_n = b$, $F(x_i) = F_i$. Заменяя на каждом отрезке разбиения длиной h , подынтегральную функцию $F(x)$ отрезком прямой, проходящей через крайние точки отрезка разбиения x_i и x_{i+1} получаем формулу трапеций:

$$\int_a^b F(x) dx = \frac{h}{2} (F_0 + 2 \sum_{i=1}^{n-1} F_i + F_n)$$

Замечание. О точности вычислений определенных интегралов. Оценка погрешности вычисления определенного интеграла делается по формулам Рунге. На практике требуется выбрать значение шага разбиения h . Алгоритм его вычисления сводится к следующему. Выбираются шаги h_1 и h_2 , причем $h_2 < h_1$.

Используя эти шаги, вычисляются каким-либо вышеописанным методом значения интегралов $A_1=A(h_1)$ и $A_2=A(h_2)$, которые являются приближёнными значениями истинного значения величины интеграла A . Если A_1 и A_2 оказываются близкими по некоторому критерию точности, то за искомое значение принимается A_2 . В противном случае выбирается новый шаг $h_3 < h_2$ и снова вычисляется значение $A_3=A(h_3)$, которое сравнивается с A_2 . Удобно выбирать $h_{i+1}=h_i/2$ (метод двойного пересчета). Значение A считается найденным, если выполняется одно из условий:

$$|A_{i+1} - A_i| \leq \varepsilon \text{ при } |A_{i+1}| \leq I, \text{ или}$$

$$\left| \frac{A_{i+1} - A_i}{A_{i+1}} \right| \leq \varepsilon \text{ при } |A_{i+1}| > I.$$

Если эти условия не выполняются, то процесс уменьшения шага продолжается. Таким образом, реализация указанного метода сводится к вычислению определённых интегралов при заданных шагах разбиения h_1 и $h_2=h_1/2$, что удобно оформить в виде отдельной функции. Полученные значения сравниваются и, если условия достижения заданной точности не выполняются, то цикл **do { } while (условие)** продолжается.

7.3.3 Метод Монте-Карло

Данный метод относится к статистическим методам. В качестве узла разбиения выбирается случайное число, выбранное из участка интегрирования $[a, b]$. Проводится N вычислений со случайными узлами x_i , результат усредняется и принимается за приближённое значение интеграла:

$$\int_a^b F(x) dx = \frac{b-a}{N} \left(\sum_{i=1}^N F(x_i) \right)$$

Погрешность вычисления интеграла зависит от числа испытаний N и пропорциональна $N^{1/2}$.

Генерирование псевдослучайных чисел осуществляется командой **rand()**. Для этого необходимо подключить библиотеки **stdlib** и **time**:

```
#include <stdlib.h>
#include <time.h>
srand ( time ( NULL ) );
double j = ( (double) rand() / RAND_MAX);
```

Данные строки позволяют сгенерировать псевдослучайное вещественное число из диапазона от 0 до 1. Его затем следует перевести в случайное число заданного диапазона, умножив на соответствующий нормирующий множитель.

Варианты заданий приведены в табл. 7.1.

Рекомендации: для проверки разработанной программы следует выполнить построение графиков функции, первообразной и производной, воспользовавшись специализированными программами MathCad, Matlab или их аналогами.

По результатам выполнения работы необходимо оформить отчёт в виде текстового документа Word, который должен включать следующие пункты:

1. Формулировка задания с указанием номера варианта и исходных данных.
2. Копии экранов с графиками функции, производной и интеграла функции и вычисленными значениями корней и определенного интеграла, выполненным заданием в любой программе построения графиков.
3. Полный программный код на Visual C++ с комментариями.
4. Копия экрана консоли с работающей программой.
5. Результаты тестирования и результаты работы программы.

Таблица 7.1 - Варианты заданий контрольной работы по исследованию функций

№	Аргумент	Функция (N=1,2...5)
1.	$x := -10, -9.9 \dots 10$	$y(x) := N - \frac{((2 - 18 \cdot N \cdot x) + 16 \cdot ((N \cdot x)^2))}{1 + N \cdot (x)^4}$
2.	$x := -3, -2.99 \dots 1$	$y(x) := \frac{\sqrt{N}}{2} - \left[\frac{\left[\left[1 + 2 \cdot (x)^4 - 8 \cdot (x)^2 \right]^2 - 4 \cdot N \cdot x^3 + N^2 \cdot x^4 \right]}{(1 - N \cdot x)^4} \right]$
3.	$x := -5, -4.99 \dots 2$	$y(x) := 5 + \frac{\left[\left[(-5 + 2 \cdot x + 1 \cdot x^2) + 4 \cdot N \cdot x^3 \right] + N \cdot x^4 \right]}{(1 - N \cdot x)^{\frac{4}{5}}}$
4.	$x := -1, -0.99 \dots 4$	$y(x) := \left[(((2 \cdot N \cdot x - 1) \cdot (2 \cdot x - 2)) \cdot (x - 3))^2 \right]^{\frac{2}{3}} - N$
5.	$x := -4, -3.99 \dots 2$	$y(x) := -2 \cdot \sqrt{N} + \frac{\left[(0.5 \cdot x^5 - 8 \cdot x^3 - 8 \cdot x^2) - N \cdot x + 10 \right]^3}{100 \cdot (x - 1)^4}$

Продолжение таблицы 7.1.

6.	$x := 0, 0.01 .. 2$	$y(x) := \sin\left(N \cdot x^{\frac{3}{4}} + 4 \cdot x^2\right) + 2 \cdot x^{\frac{3}{4}} - 2$
7.	$x := -1.9, -1.89 .. 3$	$y(x) := e^{-2x+1} \cdot \sqrt{N} \cdot (\sin(4 \cdot x - 1))^2 - 8 \cdot (x)^2$
8.	$x := 0.2, 0.21 .. 10$	$y(x) := -3 \cdot \sqrt{N} + \left[-\frac{[x^5 - (3x)^3 - N^2 \cdot x + 10]}{(x+1)^4} \right]^2$
9.	$x := 0, 0.01 .. 10$	$y(x) := 0.2 \cdot (x+2) + \sqrt{N} \cdot \sin(x) + \frac{4 \cdot \sin(3 \cdot x)}{3}$
10.	$x := 0, 0.01 .. 2$	$y(x) := \frac{\ln\left(8 \cdot \cos\left(4x^2\right)^2 + 1\right) - x^3 + 0.4}{N \cdot (x)^{\frac{1}{5}} - 4}$
11.	$x := -4, -3.99 .. 4$	$y(x) := 0.25 \cdot x^2 - 0.8 \cdot \sqrt{N} + (\sin(2 \cdot x - 1))^4 + N \cdot (\cos(x))^4$
12.	$x := -3, -2.99 .. 3$	$y(x) := 0.25 \cdot x^2 + 0.1 \cdot x - 0.8 \cdot \sqrt{N} + N \cdot (\sin((2 \cdot x)))^4 + N \cdot (\cos(x))^4$
13.	$x := 2, 2.01 .. 18$	$y(x) := \left \left(x^3 - 2x^2 + 1 \right)^{\frac{(2 \cdot N \cdot \sin(x))}{6}} \right - \frac{x \cdot N}{3}$

Продолжение таблицы 7.1

14.	$x := -2, -1.99.. 5$	$y(x) := \left[\left[\cos \left[\frac{x^3 - N \cdot (x)^2 + x - 1}{20} \right] \right] \right]^4 + N \cdot \left(\frac{x - 1}{10} \right)^2 - 1 + \frac{x}{20}$
15.	$x := 0, 0.01.. 3$	$y(x) := -6 + \left[10 \cdot (N \cdot (e)) \left[\left(x^5 + 6 \cdot x \right)^{\frac{1}{2}} - x^2 - x - 1 \right] - (x)^2 \right]$
16.	$x := 0, 0.01.. 6$	$y(x) := N \cdot (e)^{\frac{x-4}{8}} - N \cdot x + \left[4 \cdot \sin \left[\left(\frac{x}{4} \right)^2 + x \right] \right]^2$
17.	$x := -4, -3.99.. 5$	$y(x) := N \cdot (e)^{\frac{x}{8}} - x^2 + \left[4 \cdot \sin \left[\left(\frac{x}{4} \right)^2 + \sqrt{N \cdot x} \right] \right]^2$
18.	$x := -2, -1.99.. 16$	$y(x) := \left[\left[x^5 - N \cdot (x)^4 - 4 \cdot x^3 + (N \cdot x)^2 - 6 \cdot N \cdot x \right] - 10 \right] \cdot e^{-(x)} - 1$
19.	$x := -10, -9.99.. 1$	$y(x) := 0.2 \cdot (x + 1) + N \cdot \sin(x) + \frac{2 \cdot \sin(3 \cdot x - 1)}{3}$
20.	$x := -4, -3.99.. 4$	$y(x) := (2 \cdot x)^2 - N^2 \cdot (\sin((2 \cdot x)))^4 + 4 \cdot x + \left(\cos \left(\frac{x^2}{4} \right) \right)^4$

Окончание таблицы 7.1.

21.	$x := -4, -3.99 .. 4$	$y(x) := -(2 \cdot x)^2 \cdot N \cdot \sin((2 \cdot x))^4 + 4 \cdot N \cdot x + \left(\cos\left(\frac{x^2}{4}\right) \right)^4$
22.	$x := 0, 0.01 .. 10$	$y(x) := 0.2 \cdot (x + 2) + \sin(x) + \frac{4 \cdot \sqrt{N} \cdot \sin(3 \cdot x)}{3}$
23.	$x := -4, -3.99 .. 4$	$y(x) := 0.25 \cdot x^3 - 0.8 \cdot \sqrt{N} + (\sin(2 \cdot x - 1))^4 + N^2 \cdot (\cos(x))^4$
24.	$x := 0, 0.01 .. 6$	$y(x) := e^{\frac{x-4}{8}} - N \cdot x + \left[4 \cdot \sin\left[\sqrt{N} \cdot \left(\frac{x}{4}\right)^2 + x \right] \right]^2$

8. МОДЕЛИРОВАНИЕ ЗАДАЧ ТЕПЛОПРОВОДНОСТИ

8.1. ОБРАБОТКА ЭКСПЕРИМЕНТАЛЬНЫХ ДАННЫХ

При обработке результатов экспериментов, целью которых является определение параметров моделей, описывающих физические законы, эффективным оказывается применение метода наименьших квадратов (м.н.к), основные положения которого разработаны К. Гауссом (1795) и А. Лежандром (1806). Суть м.н.к. заключается в минимизации ошибки, возникающей при замене истинного (неизвестного) значения физической величины X её приближённым значением X_{exp} , определенным по результатам наблюдений. М.н.к обосновывает, что наилучшее приближение соответствует минимуму квадрата ошибки $(X - X_{exp})^2$. Таким образом, алгоритм наилучшего приближения модели с параметром сводится к поиску в заданных пределах его искомого значения по минимуму суммы квадратов разностей между наблюдаемыми в эксперименте данными и вычисленными значениями функции, моделирующей данное явление.

Рассмотрим частный пример, когда исследуемая физическая величина y линейно зависит от другой величины t :

$$y(t) = at + b, \quad (8.1)$$

где a и b – параметры зависимости.

Пусть в результате эксперимента определено, что ряду значений t_i соответствуют значения $y_i(t_i)$ (где $i=1, \dots, N$). Поскольку в любом эксперименте всегда существуют погрешности, то, как это показано на рис.8.1, наблюдаемое значение отличается от истинного на величину $\Delta y_i = y(t_i) - y_i(t_i) \neq 0$.



Рисунок 8.1 – Иллюстрация метода наименьших квадратов

Согласно м.н.к. задача выбора параметров a и b наилучшего приближения соотношения (7.1) сводится к определению минимума функции:

$$S(a, b) = \sum_{i=1}^N (\Delta y_i)^2 = \sum_{i=1}^N (at_i + b - y_i)^2 \quad (8.2)$$

Из курса математического анализа известно, что точки минимума этой функции двух переменных определяются из соотношений:

$$\begin{cases} \frac{\partial S(a,b)}{\partial a} = 0 \\ \frac{\partial S(a,b)}{\partial b} = 0 \end{cases} \quad (8.3)$$

Подставив (8.2) в (8.3), получаем систему линейных неоднородных уравнений для определения значений a и b :

$$\begin{cases} \sum_{i=1}^N (at_i + b - y_i) \cdot t_i = (\sum_{i=1}^N t_i^2) \cdot a + (\sum_{i=1}^N t_i) \cdot b - \sum_{i=1}^N (y_i \cdot t_i) = 0 \\ \sum_{i=1}^N (at_i + b - y_i) = (\sum_{i=1}^N t_i) \cdot a + N \cdot b - \sum_{i=1}^N y_i = 0 \end{cases} \quad (8.4)$$

Решение этой системы имеет вид:

$$\begin{cases} a = \frac{C \cdot N - B \cdot D}{A \cdot N - B^2} \\ b = \frac{A \cdot D - B \cdot C}{A \cdot N - B^2} \end{cases}, \quad (8.5)$$

где обозначено: $A = \sum_{i=1}^N t_i^2$, $B = \sum_{i=1}^N t_i$, $C = \sum_{i=1}^N (y_i \cdot t_i)$ и $D = \sum_{i=1}^N y_i$.

Полученные по соотношениям (8.5) значения параметров a и b задают прямую, наилучшим образом описывающую полученные экспериментальные данные, что иллюстрируется рисунком 1. В общем случае часто требуется определить наиболее подходящее значение параметров a_i , если заранее известен вид зависимости одной физической величины y от другой t : $y = f(a_1, \dots, a_n, t)$, где a_i – параметры зависимости ($i=1, \dots, n$). В частности, если в

результате эксперимента получено, что каждому из значений физической величины t_i соответствуют значения другой физической величины y_i , то оптимальной оценкой при определении зависимости $y=f(a_1, \dots, a_n, t)$, естественно признать такую кривую, для которой значение суммы квадратов отклонений $(y(t_i)-y_i(t_i))$ в каждой точке наблюдений минимально. Используя алгоритм поиска минимальной суммы среди сумм квадратов отклонений при разных значениях параметров a_i , можно определить их наиболее оптимальные значения.

8.2 МОДЕЛИРОВАНИЕ ЗАКОНА ВНЕШНЕЙ ТЕПЛОПЕРЕДАЧИ

Закон теплопередачи был предложен Ньютоном и Рихманом для описания процесса остывания тела путем теплообмена через границу раздела двух сред, не находящихся в тепловом равновесии друг с другом. Если теплопроводность рассматриваемого тела достаточно велика, то уместно положить, что в каждый момент времени все точки тела имеют одинаковую температуру. Это же относится и к внешней среде. Считается, что поток тепла на границе двух тел распространяется перпендикулярно поверхности раздела от более нагретого тела к менее нагретому (внешней среде), и его значение пропорционально разности их температур. В таком случае для описания процесса теплопередачи справедливо следующее соотношение (закон теплопередачи Ньютона):

$$\frac{dT}{dt} = -r(T - T_S) \quad T > T_S \quad (8.6)$$

где T – температура тела, T_S – температура окружающей среды, r – эмпирический коэффициент, характеризующий скорость остывания (интегральный «коэффициент остывания», коэффициент теплоотдачи) постоянный в определенном диапазоне температур и зависящий от физических свойств тела и его геометрических параметров.

В простейшем случае, если коэффициент r – константа, уравнение (8.6) можно решить аналитически:

$$\begin{aligned} \int \frac{d(T - T_S)}{T - T_S} &= -r \int dt \\ \ln(T - T_S) &= -rt + Const = -rt + \ln(T_0 - T_S) \\ T(t) &= T_S - (T_S - T_0) \cdot e^{-rt}, \end{aligned} \quad (8.7)$$

где T_0 – температура тела в момент времени $t = 0$.

В общем случае коэффициент теплоотдачи не постоянен и может даже зависеть от разности температур, т.е. дифференциальное уравнение, описывающее процесс теплопередачи, имеет вид:

$$\frac{dT}{dt} = g(t, T - T_s) \quad (8.8)$$

где правая часть уравнения зависит от t и разности температур $T - T_s$ нелинейно и выражается достаточно сложной функцией $g(t, T - T_s)$.

В этом общем случае уравнение (7.8) не решается в квадратурах и требуется применение численных методов. Один из методов численного решения дифференциальных уравнений предложенный Эйлером, состоит в замене производных конечными разностями (метод конечных разностей). Такая замена дифференциальных коэффициентов уравнения (7.8) позволяет построить его конечно-разностную схему и свести его решение к решению алгебраического разностного аналога.

Пусть:

$$\frac{dT}{dt} = g(t, T) \quad (8.9)$$

причем известны начальные значения (т.е. при $t = t_0$, $T(t_0) = T_0$). Тогда, согласно определению производной, значение функции в точке $t + \Delta t$, близкой к t_0 будет приближенно равно:

$$T_1 = T(t_0) + \Delta T \approx T_0 + g(t_0) \cdot \Delta t \quad (8.10)$$

Обобщая это соотношение для n -го шага разбиения интервала моделирования, получаем приближенное равенство:

$$T_n \approx T_{n-1} + g(t_{n-1}) \cdot \Delta t \quad (8.11)$$

где $n = 1, 2, \dots, N$.

Рис.8.2 показывает, что согласно методу Эйлера, для определения значения функции в каждой последующей точке используется аппроксимация нелинейной функции прямой с наклоном, определяемым соотношением (8.11).

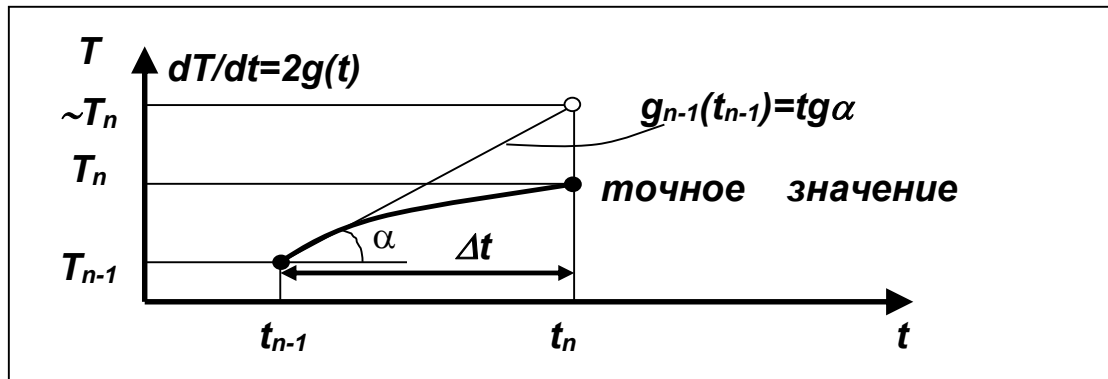


Рисунок 8.2 – Иллюстрация численного решения дифференциального уравнения первого порядка

Рассмотрим конкретный пример. Пусть $dT/dt = 2t$ и $t_0=1$, $T_0=1$. Требуется определить значение функции $T(t)$ в точке $t = 2$: т.е. $T(2)$. Реализация алгоритма Эйлера при этих начальных условиях иллюстрируется рис.8.3.

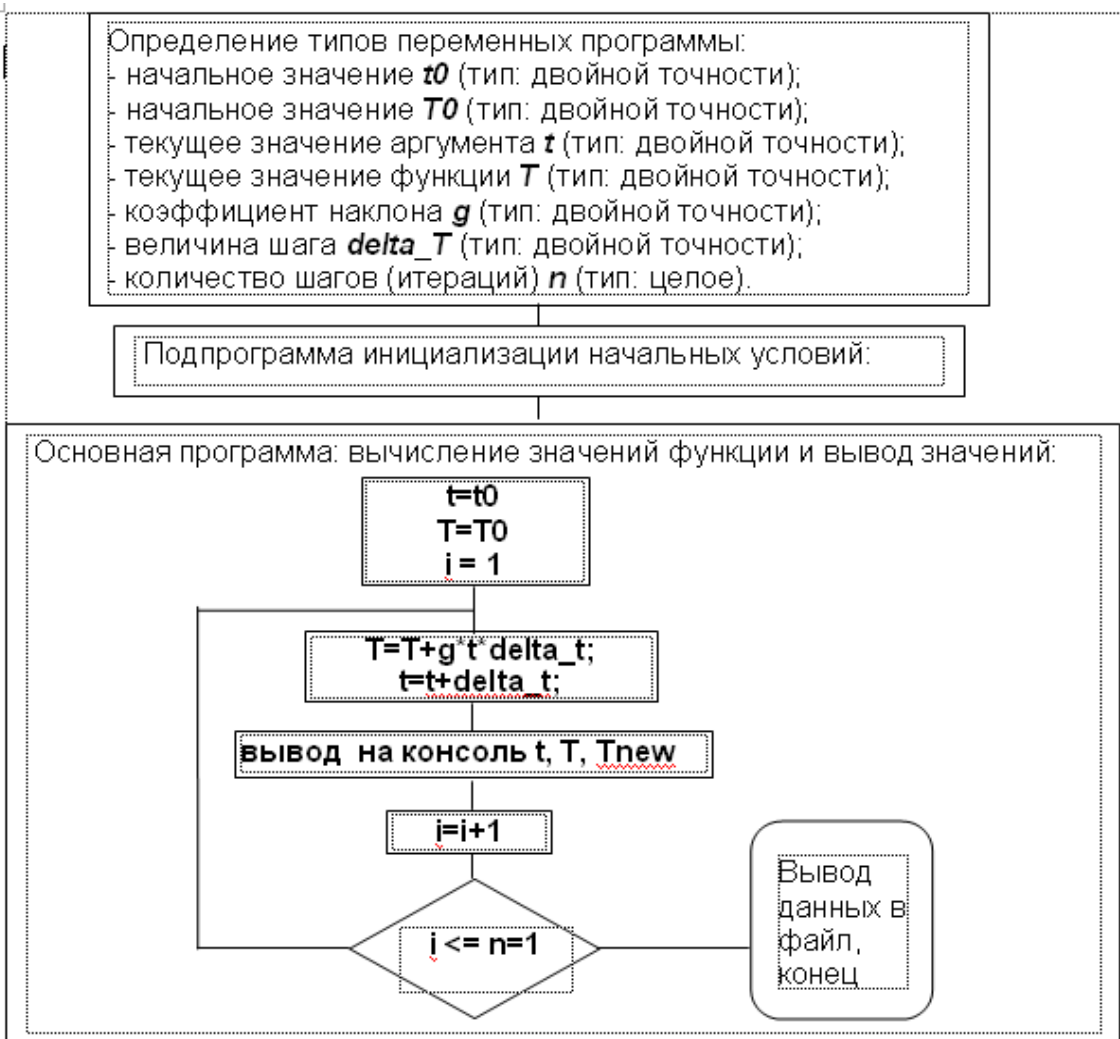


Рисунок 8.3 – Блок-схема программы численного решения дифференциального уравнения $dT/dt = 2t$

Ниже приведен пример кода программы реализующей этот алгоритм, использующий простейшие конструкции языка C++ :

```
//Euler_0_v1

#include <stdio.h>

double t0;    //нач. значение аргумента
double T0;    //нач. значение функции
double t;     //текущее значение аргумента
double T;     //текущее значение функции
double delta_t; //шаг аргумента
double g;     //коэффициент наклона
int n;        //количество шагов

void init_func (void)
{
    t0=1.0;    //начальные значения
    T0=1.0;
    g=2.0;
    delta_t=0.1;
    n=10;
}

int main()
{
    init_func (); // t0, T0, g, dt, n

    t=t0;
    T=T0;
    double Change_T = 0.0; //изменение функции
    for (int i=1; i <= n+1; i++)
    {

        Change_T=g*t*delta_t;
        //Вывод на консоль
        printf ("i=%02d  t=%4.2f T=%4.2f Tnew=%4.2f\n", i, t, T, T+Change_T);
```

```

    T=T+Change_T;
    t=t+delta_t;
}
}

```

Очевидно, что погрешность вычисления каждого следующего значения функции, для которой справедливо соотношение (8.11) зависит от того, насколько мало значение шага разбиения.

8.3 ОБРАБОТКА ДАННЫХ ЭКСПЕРИМЕНТА ЗАКОНА ТЕПЛОПЕРЕДАЧИ

Задание. Пусть проводятся серии экспериментов, в которых в определенные моменты времени фиксируется температура кофе, налитого в керамическую чашку, и остывающего на воздухе (см. таблицу 8.2).

Таблица 8.2 – Данные эксперимента об остывании чашки кофе

Время, мин.	N варианта					
	0	1	2	3	4	5
0	80.5	89.5	88.5	89.5	89.5	88.5
1	77.5	86.5	84.5	86.5	83.5	79.5
2	75.0	83.0	82.5	83.0	79.0	76.5
3	73.0	81.5	80.5	81.5	74.0	72.5
4	71.0	79.0	77.5	79.0	71.5	68.0
5	70.0	78.0	76.0	78.5	65.5	65.0
6	68.0	75.0	74.5	75.0	63.5	61.0
7	66.5	73.5	72.5	73.5	61.5	60.0
8	64.5	71.0	70.5	71.5	58.0	55.5
9	63.5	70.0	68.0	70.0	55.0	54.0
10	62.0	67.5	67.0	67.5	52.0	50.0
11	61.0	65.5	64.5	65.5	51.5	48.0
12	60.0	63.5	63.5	63.5	49.0	47.0
13	58.5	63.0	63.0	63.0	47.5	46.0
14	58.0	61.0	60.0	61.0	44.5	44.0
15	56.5	60.5	60.0	60.5	43.0	43.0

Продолжение таблицы 8.2.

Время, мин.	N варианта					
	6	7	8	9	10	11
0	88.5	77.0	86.0	78.5	66.5	51.5
1	84.5	74.5	82.5	77.5	63.0	48.5
2	82.5	71.0	82.0	76.5	60.0	48.0
3	80.5	68.0	80.5	72.5	56.5	46.5
4	77.5	65.5	78.5	72.0	53.5	44.5
5	76.0	61.0	77.5	68.5	49.5	42.5
6	74.5	59.0	77.0	68.0	49.0	41.5
7	72.5	56.0	76.5	67.5	48.0	41.0
8	70.5	54.5	76.0	64.0	45.5	38.0
9	68.0	50.5	74.0	62.0	44.0	37.0
10	67.0	50.0	73.0	61.5	41.5	34.5
11	64.5	48.0	72.0	61.0	41.0	34.0
12	63.5	46.0	71.5	58.5	40.5	33.0
13	63.0	45.0	69.5	58.0	38.0	32.5
14	60.5	42.5	69.0	56.5	37.0	33.0
15	60.0	39.5	68.5	55.0	34.0	32.0
Время, мин.	N варианта					
	12	13	14	15	16	17
0	76.5	41.5	70.0	68.0	59.5	43.0
1	72.5	41.0	68.0	67.5	59.0	41.5
2	72.0	39.0	65.5	64.0	57.0	40.5
3	68.5	39.0	61.0	62.0	56.0	39.0
4	67.5	35.5	60.0	61.5	55.0	37.5
5	67.0	36.0	60.0	61.0	53.0	36.0
6	64.0	35.0	59.5	59.0	51.5	35.5
7	62.5	34.0	58.0	58.5	51.0	34.5
8	61.5	31.5	55.0	56.5	50.0	33.5
9	61.0	32.0	54.0	55.0	49.5	32.5
10	58.5	33.0	53.0	54.0	48.0	32.0
11	58.0	31.0	52.0	53.0	47.0	30.0
12	56.5	31.5	52.5	51.0	46.0	31.5
13	55.0	30.0	50.0	50.5	45.0	41.5
14	54.0	30.0	50.0	49.0	44.5	30.0
15	53.0	28.0	49.0	47.5	43.5	29.0

Окончание таблицы 8.2.

Время, мин.	N варианта					
	18	19	20	21	22	23
0	61.0	82.5	79.0	79.5	51.0	72.0
1	58.5	82.0	75.5	76.0	48.0	68.0
2	58.0	79.0	75.0	70.5	47.0	64.5
3	56.5	78.5	75.0	68.0	45.0	61.0
4	55.0	77.5	74.5	65.0	43.0	58.5
5	54.0	76.0	74.0	61.5	42.0	56.0
6	53.0	75.5	71.0	59.0	40.0	54.0
7	51.5	75.0	70.0	56.0	39.5	50.5
8	50.5	74.5	69.0	54.0	37.0	49.0
9	49.0	73.0	68.5	51.0	36.5	47.0
10	47.5	72.5	68.0	48.5	35.5	44.5
11	47.0	72.0	67.5	47.0	34.0	43.5
12	45.5	71.0	67.0	44.5	33.5	42.0
13	44.5	70.5	66.0	43.5	33.0	40.5
14	44.0	67.5	64.0	41.0	32.0	39.5
15	43.5	67.0	63.5	40.5	31.5	38.5

Кофе находится в стакане из материала высокой теплопроводностью, помещенной в керамический стакан (температура кофе регистрировалась с точностью до 0.5°C , температура окружающего воздуха равна 22°C). Требуется на основе компьютерного моделирования проверить справедливость применимости закона теплопередачи (8.6) к этому процессу и определить экспериментальное значение коэффициента r .

Указания: для решения этой задачи рекомендуется следовать следующему алгоритму.

1) Полагая, что данный эксперимент описывается моделью (8.6), и используя соотношения (8.7), напишите программу, позволяющую по полученным экспериментальным данным, приведенным в таблице 8.2 (варианты задания), определить значение параметра r закона теплообмена Ньютона. При выполнении этого задания воспользуйтесь методом наименьших квадратов, применив его к логарифму разности $T(t) - T_S$ из соотношения (8.7). Предложите, как исходя из соотношения (8.7) и данных эксперимента оценить предполагаемое начальное значение параметра r , и в соответствии с вариантом задания выполните эту оценку. Затем напишите код программы, которая,

варьируя параметр r , определяет при каком его значении сумма квадратов отклонений данных эксперимента от $\ln(T(t)-T_s)$ минимальна. Полученное таким образом значение r используйте затем для моделирования процесса остывания кофе.

2) С целью определения справедливости модели (8.6) для полученного значения параметра r напишите программу, моделирующую процесс теплопередачи (8.6) путем численного решения дифференциального уравнения первого порядка.

Для этого используйте код программы, приведенный на рисунке 8.4, внося необходимые изменения в обозначения переменных и код программы. Задайте следующие переменные и начальные условия:

$t = 0$ – начальное время (мин),
 $T = 83$ – начальная температура кофе ($^{\circ}\text{C}$),
 $room_T$ – комнатная температура ($^{\circ}\text{C}$),
 $r = -$ коэффициент остывания (1/мин), определенный в п.1.
 $dt = 0.01$ – шаг по времени (мин),
 $tmax$ – длительность моделирования (мин),
 n – общее количество шагов.

3) Модифицируйте программу так, чтобы исходное значение параметра r и начальные условия можно было вводить с клавиатуры.

4) Один из методов определения точности численного решения заключается в повторении вычислений с меньшим шагом и сравнении результатов (метод Рунге-Кутты).

Если решения в обоих результатах совпадают с заданной точностью, то можно ограничиться достигнутым шагом разбиения. Исходя из этих соображений, убедитесь, что выбранное значение величины шага по времени достаточно мало и не оказывает влияние на получаемую зависимость температуры от времени. Предложите способы тестирования правильности работы программы.

5) Используя дополнительный вложенный цикл, модернизируйте программу, так чтобы полученные результаты выводились в консоль и сохранялись в текстовом файле.

6) Вывод в текстовый файл можно реализовать, используя следующие операторы:

```
//открытие //файла с данными моделирования:
```

```
FILE* f=fopen ("Nevton_r.txt","w");
```

```
//запись в //файл значения параметра:
```

```
fprintf (f, "%5.3f\n", r);
```

```
//запись результатов вычисленных значений в цикле:

{
...
fprintf (f, "%02d %4.2f %4.2f %4.2f\n", i, t, T, T+change_T);
...
}
fclose (f);
```

7) Модифицируйте программу так, чтобы значения параметров и начальных условий можно было считывать из текстового файла инициализации.

8) Модернизируйте разработанный программный код так, чтобы значение интегрального коэффициента теплоотдачи определялось автоматически. Для этого используйте код программы, полученной в п.1) как подпрограмму.

9) С целью исследования применимости модели сравните полученные результаты моделирования с аналитическим решением. Для этого импортируйте данные численного моделирования п.2) в текстовый файл, а затем в электронных таблицах Excel постройте графики данных опыта из таблицы 8.2, а также численного и аналитического решения задачи об остывании кофе. Оформите надлежащим образом график в Excel и импортируйте его в документ Word. Пример оформления: см. рис.8.4 и 8.5.

r	t_i	$T(t_{i+1})$	$T(t_i)$
0.036			
01	0.00	80.50	80.29
02	0.10	80.29	80.08
03	0.20	80.08	79.87
04	0.30	79.87	79.66
05	0.40	79.66	79.45
06	0.50	79.45	79.25
07	0.60	79.25	79.04
08	0.70	79.04	78.84
09	0.80	78.84	78.63
10	0.90	78.63	78.43
11	1.00	78.43	78.22
12	1.10	78.22	78.02
13	1.20	78.02	77.82
14	1.30	77.82	77.62
15	1.40	77.62	77.42
16	1.50	77.42	77.22

Рисунок 8.4 – Фрагмент текстового файла с результатами работы программы моделирования задачи об остывании кофе

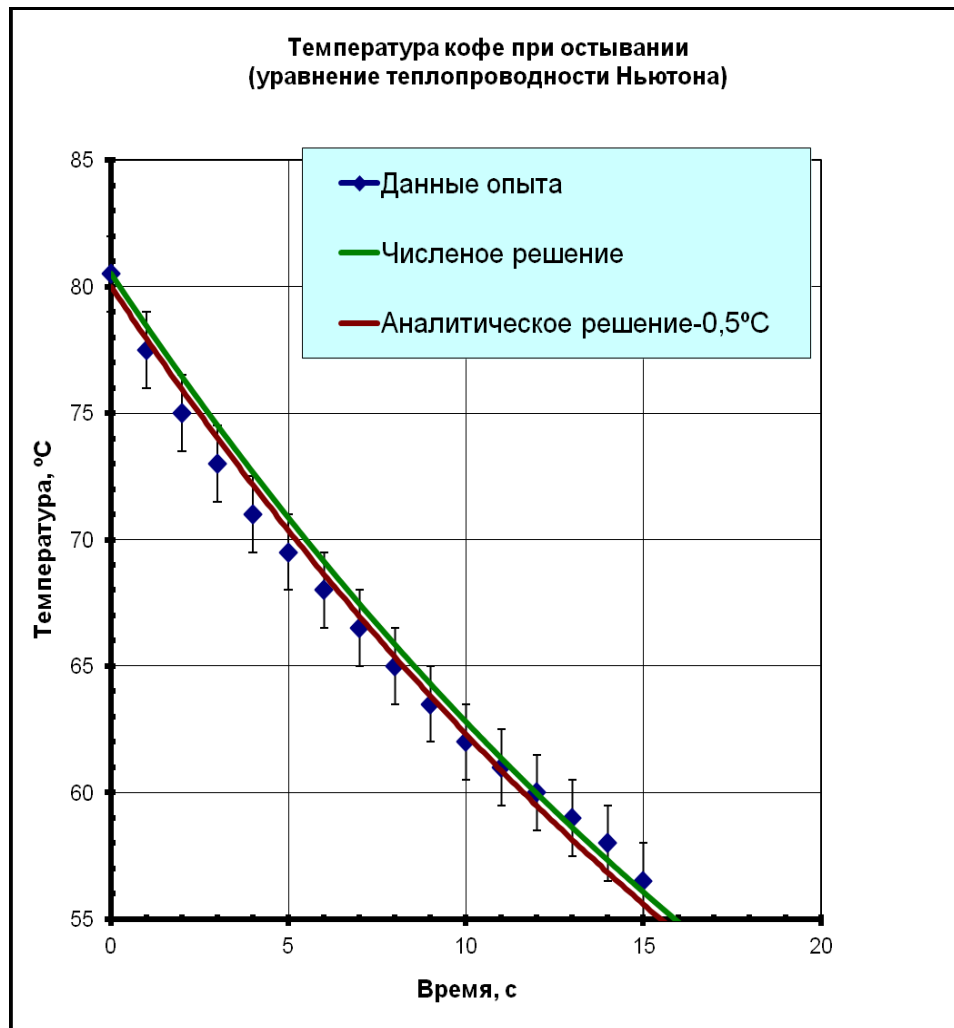


Рисунок 8.5 – Пример оформления графика с данными эксперимента и расчетными данными задачи об остывании кофе. Для наглядности график аналитического решения смещен на $0,5^{\circ}\text{C}$ вниз

Задания повышенной сложности:

I* Начальная разность температур между кофе и окружающей средой равна 61°C . Сколько надо остужать кофе, чтобы эта разность составила $61/2=30,5^{\circ}\text{C}$? Через какое время разность температур уменьшится до $61/4^{\circ}\text{C}$ и до $61/8^{\circ}\text{C}$? Прежде, чем проводить моделирование, попытайтесь из простых соображений предугадать результаты.

II* Пусть добавление молока понижает температуру кофе на 5°C . Используя разработанную вами программу моделирования и выбранное значение коэффициента остывания, решите, как быстрее охладить кофе до температуры 75°C : 1) добавить сначала молоко или 2) дожидаться, когда температура понизится до 80 градусов, а потом добавить молоко?

III* С помощью программы, разработанной в пп. 6 и 7 задания, вычислите температуры в момент времени 1 мин, изменяя значения шага Δt по времени: 0.1 ,

0.05, 0.025, 0.01 и 0.005 мин. Постройте таблицу, содержащую разность между точным и численным решением уравнения как функцию шага по времени. Будет ли эта функция убывающей? Постройте график функции. Если шаг уменьшить в два раза, как изменится разность? В случае, если разность между аналитическим и численным решениями пропорциональна $(\Delta x)^n$, численный метод называется методом n -го порядка точности. Какой порядок точности метода Эйлера?

IV* Какой нужно выбрать величину шага, чтобы достигалась точность 0,1% в момент времени 1 мин? В момент времени 5 мин?

V* Используя электронные таблицы Excel, найдите время, необходимое для того, чтобы разность температур между температурой кофе и комнатной температурой составила $1/e \approx 0,37$ от начальной (время релаксации). Проверьте, зависит ли это время от начальной температуры кофе или комнатной температуры? Попробуйте взять различные значения коэффициента r и определите зависимость времени релаксации от r .

9. МОДЕЛИРОВАНИЕ ВЯЗКОГО ТРЕНИЯ

Общее выражение для описания одномерного движения вдоль оси y материальной точки (частицы) массой m под действием равнодействующей силы $F_y(y, v)$:

$$m \frac{d^2 y}{dt^2} = F_y, \quad (9.1)$$

где $v = dy/dt$ – скорость частицы.

Рассмотрим частный случай. Пусть тело падает вертикально вниз с высоты y_0 без начальной скорости под действием сил тяжести и тормозящей силы, причем тормозящая сила зависит от скорости тела. Пусть ось y направлена вертикально вверх, и начало отсчета находится на поверхности земли (см. рис.9.1).

В общем случае вид зависимости силы сопротивления от скорости обусловлен многими факторами и определяется из данных опыта. Поскольку вычисление наклона кривых, необходимое для нахождения скорости и ускорения сопряжено с большими ошибками, то лучшим является метод, когда заранее предполагается какой-то определенный вид зависимости силы сопротивления от скорости $F_d(v)$.

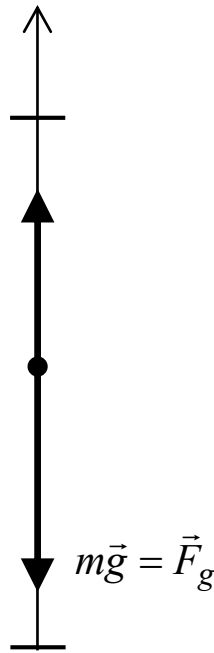


Рисунок 9.1 – Силы тяжести и сопротивления воздуха

В общем случае вид зависимости силы сопротивления от скорости обусловлен многими факторами и определяется из данных опыта. Поскольку вычисление наклона кривых, необходимое для нахождения скорости и ускорения сопряжено с большими ошибками, то лучшим является метод, когда заранее предполагается какой-то определенный вид зависимости силы сопротивления от скорости $F_d(v)$.

Обычно зависимость величины силы сопротивления от скорости задается феноменологическим выражением (т.е. выражением, которое не является точным законом физики, а используется для приближенного описания в ограниченных диапазонах аргументов и параметров):

$$F_d(v) = k_1 \cdot v \quad \text{или} \quad F_d(v) = k_2 \cdot v^2 \quad (9.2)$$

где параметры k_1 и k_2 зависят от свойств среды.

Поскольку в данном случае предполагается, что сопротивление растет со скоростью, то существует предельная или установившаяся скорость v_1 или v_2 , соответствующая моменту, когда равнодействующая сил тяжести F_g и сопротивления F_d , будет равна нулю, т.е. будет выполняться условие $F_d = F_g = mg$ (g – ускорение свободного падения, стандартное значение $g = 9.80665$ м/сек²). Соответствующие выражения для величин установившейся скорости:

$$v_1 = \frac{mg}{k_1} \quad \text{или} \quad v_2 = \left(\frac{mg}{k_2} \right)^{1/2} \quad (9.3)$$

Отсюда, переходя для удобства численных расчетов к безразмерной скорости, получаем выражения для сил сопротивления F_{1d} и F_{2d} и равнодействующих сил F_1 и F_2 :

$$F_{1d} = k_1 v_1 \left(\frac{v}{v_1} \right) = mg \left(\frac{v}{v_1} \right) \quad \text{или} \quad F_{2d} = k_2 v_2^2 \left(\frac{v}{v_2} \right)^2 = mg \left(\frac{v}{v_2} \right)^2 \quad (9.4)$$

$$F_1(v) = -mg \left(1 - \frac{v}{v_1} \right) \quad \text{или} \quad F_2(v) = -mg \left(1 - \frac{v^2}{v_2^2} \right), \quad (9.5)$$

где F_1 и F_2 – проекции результирующей силы на ось y (эти выражения рекомендуется получить самостоятельно).

Обобщение метода Эйлера на случай решения дифференциальных уравнений второго порядка (т.е. уравнений вида (9.1)) состоит в том, чтобы, используя разбиение по времени с шагом Δt , выразить значения скорости и координаты точки через значения скорости, координаты и ускорения в предыдущий момент времени. Таким образом, решение дифференциального уравнения второго порядка сводится к решению системы линейных алгебраических уравнений вида:

$$v_{n+1} = v_n + a_n \Delta t \quad (9.6)$$

$$y_{n+1} = y_n + v_n \Delta t \quad (9.7)$$

Здесь аналогично тому, как это делается в методе Эйлера для дифференциального уравнения первого порядка, скорость в конечной точке интервала v_{n+1} вычисляется через производную скорости (скорость изменения скорости, т.е. ускорение) a_n в начальной точке этого интервала. Также, координата в конечной точке интервала y_{n+1} вычисляется через скорость v_n в начальной точке этого интервала.

Предложенный алгоритм численного решения дифференциального уравнения второго порядка не является единственным. В частности, можно

вычислять координату через скорость не в начальной точке интервала, а в конечной (аппроксимации по последней точке):

$$v_{n+1} = v_n + a_n \Delta t \quad (9.8)$$

$$y_{n+1} = y_n + v_{n+1} \Delta t \quad (9.9)$$

Этот метод численного решения дифференциального уравнения был проанализирован Кроммером и носит название метода Эйлера-Кромера.

Задание

1) Для проверки правильности кода программы численного решения сначала получите данные моделирования, позволяющие сравнить их с известными аналитическими решениями. Для этого решите численно задачу о свободном падении тела без сопротивления воздуха. Пример кода программы для решения приведен ниже:

```
//Euler_2_1.1

# include <stdio.h>

/* эти глобальные переменные доступны их всех функций и могут
ими изменяться */
double Mass_HeightE[1000]; //изменение координаты тела
относительно начального положения (моделирование), м
double Mass_HeightA[1000]; //изменение текущей координаты
(точное решение), м
double Mass_VitessE[1000]; //текущая скорость (моделирование)
м/сек
double Mass_VitessA[1000]; //текущая скорость (точное решение)
м/сек
double Mass_Accel[1000]; // текущее ускорение м/сек2
double Mass_Time[1000]; // текущее время, сек
double Vitess_0=0.0; // начальная скорость, м/сек
double g=9.8; // ускорение свободного падения = 9,8 м/сек2
double Height=15; // высота от поверхности земли, м
double Period_T=1.6; // время падения, сек
double dt=0.1; // шаг по времени, сек

int n = int(Period_T/dt); // число шагов

void init_F () // инициализация массивов
{
```



```

    for (int i = 0; i < 1000; i++)
    {
        Mass_HeightE[i] = 0.0;
        Mass_HeightA[i] = 0.0;
        Mass_VitessE[i] = 0.0;
        Mass_VitessA[i] = 0.0;
        Mass_Accel[i] = g;
        Mass_Time[i] = 0.0;
    }

    Mass_HeightA[0] = Height;
}

void Euler_diff_2 ()
{
    for (int i = 0; i < n; i++)
    {
        Mass_Accel[i+1] = g;
        Mass_HeightE[i+1] = Mass_HeightE[i] + Mass_VitessE[i] * dt;
        Mass_VitessE[i+1] = Mass_VitessE[i] + Mass_Accel[i] * dt;
        Mass_Time[i+1] = Mass_Time[i] + dt;
        Mass_VitessA[i+1] = Vitess_0 + g * Mass_Time[i+1];
        Mass_HeightA[i+1] = Height - (Vitess_0 * Mass_Time[i+1] +
+ 0.5 * g * Mass_Time[i+1] * Mass_Time[i+1]);
    }
}

void Euler_Kromer ()
{
    for (int i = 0; i < n; i++)
    {
        Mass_Accel[i+1] = g;
        Mass_VitessE[i+1] = Mass_VitessE[i] + Mass_Accel[i] * dt;
        Mass_HeightE[i+1] = Mass_HeightE[i] + Mass_VitessE[i+1] *
dt;

        Mass_Time[i+1] = Mass_Time[i] + dt;
        Mass_VitessA[i+1] = Vitess_0 + g * Mass_Time[i+1];
        Mass_HeightA[i+1] = Height - (Vitess_0 * Mass_Time[i+1] +
+ 0.5 * g * Mass_Time[i+1] * Mass_Time[i+1]);
    }
}

void Print_results_Euler_2 ()

```

```

{
    FILE* f = fopen("res_dif_2_E.txt", "w");
    for (int i = 0; i < n; i++)
    {
        printf (" T[%2d]= %5.3f %5.3f %5.3f %5.3f %5.3f\n", i,
Mass_Time[i], Height - Mass_HeightE[i], Mass_HeightA[i],
Mass_VitesseE[i], Mass_VitesseA[i]);
        fprintf (f, "%u %5.3f %5.3f %5.3f %5.3f %5.3f\n", i,
Mass_Time[i], Height - Mass_HeightE[i], Mass_HeightA[i],
Mass_VitesseE[i], Mass_VitesseA[i]);
    }
    fclose(f);
}

void Print_results_Kromer ()
{
    FILE* f = fopen("res_dif_2_K.txt", "w");
    for (int i = 0; i < n; i++)
    {
        printf (" T[%2d]= %5.3f %5.3f %5.3f %5.3f %5.3f\n", i,
Mass_Time[i], Height - Mass_HeightE[i], Mass_HeightA[i],
Mass_VitesseE[i], Mass_VitesseA[i]);
        fprintf (f, "%u %5.3f %5.3f %5.3f %5.3f %5.3f\n", i,
Mass_Time[i], Height - Mass_HeightE[i], Mass_HeightA[i],
Mass_VitesseE[i], Mass_VitesseA[i]);
    }
    fclose(f);
}

int main ()
{
    init_F ();
    Euler_diff_2 ();
    Print_results_Euler_2 ();

    printf("\n");

    init_F ();
    Euler_Kromer ();
    Print_results_Kromer ();

    return 0;
}

```

Результаты импорта расчетных данных в электронные таблицы и соответствующие графики приведены на рис.9.2.

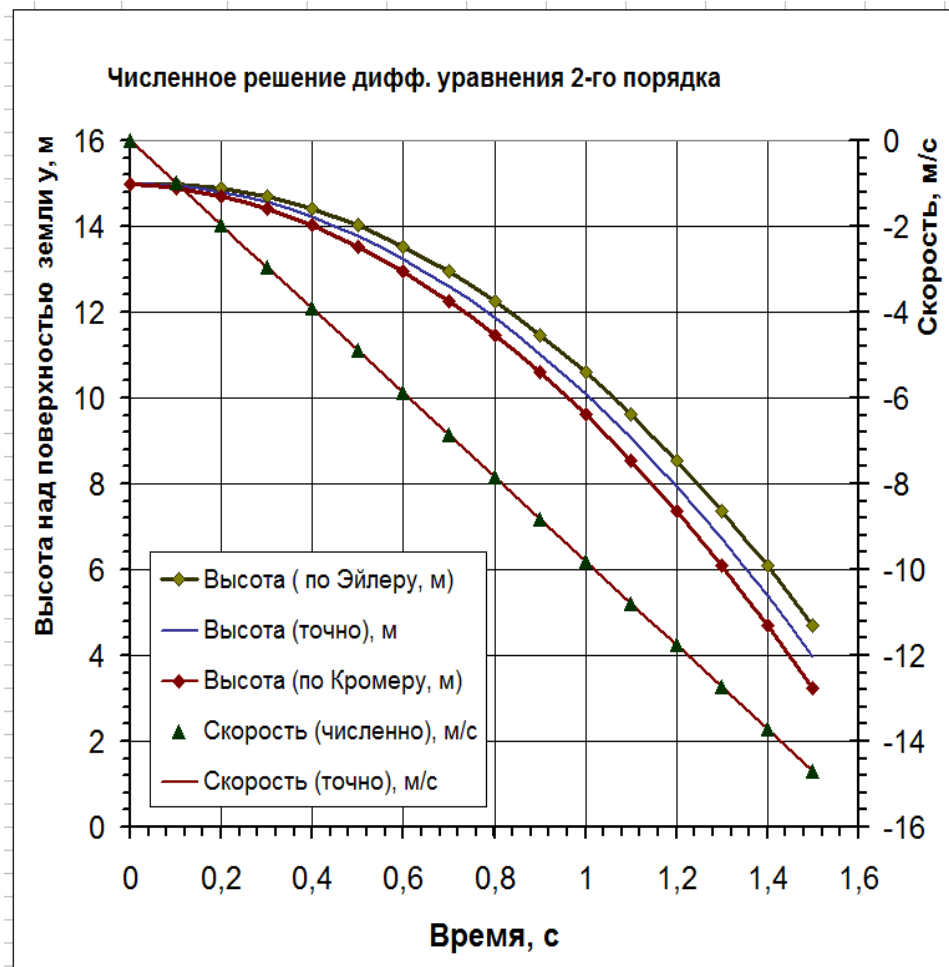


Рисунок 9.2 – Результаты решения задачи о свободном падении тела

Указание 1. Поскольку данные о координатах в разные моменты времени имеют одинаковый тип, то целесообразно выделить для их сохранения в программе упорядоченный по номерам объем памяти, т.е. сформировать массив. Аналогично следует поступить для сохранения значений скорости и текущего времени.

Указание 2. Чтобы облегчить возможность модернизации программы, процедуру определения координаты и скорости в определенный момент времени по алгоритму Эйлера или Эйлера-Кроммера целесообразно оформить как отдельную функцию.

Контрольное задание. Модифицируйте программу предыдущего задания так, чтобы равнодействующая сила задавалась формулами (9.4 и 9.5). Определите, какая зависимость лучше всего согласуется с данными опыта, приведенными в таблице 9.1 и определите значение установившейся скорости. Импортируйте

данные расчетов в электронные таблицы и постройте графики с данными расчетов и эксперимента (см. рисунок 9.3).

Таблица 9.1 – Экспериментальные данные о вязком падении тел

Время, мс	131	231	331	431	531	631	731	831	931
№ вар.	Величина смещения тела от начального положения, м								
0.	0.084	0.258	0.519	0.857	1.264	1.726	2.234	2.780	3.354
1.	0.07	0.18	0.30	0.42	0.55	0.67	0.79	0.92	1.04
2.	0.08	0.20	0.34	0.49	0.64	0.78	0.93	1.08	1.23
3.	0.08	0.24	0.45	0.67	0.97	1.24	1.52	1.81	2.10
4.	0.08	0.25	0.48	0.76	1.07	1.40	1.75	2.109	2.46
5.	0.08	0.25	0.49	0.79	1.12	1.49	1.87	2.26	2.66
6.	0.08	0.26	0.51	0.83	1.21	1.63	2.08	2.55	3.04
7.	0.08	0.26	0.51	0.85	1.24	1.69	2.17	2.68	3.22
8.	0.08	0.26	0.52	0.87	1.29	1.78	2.32	2.91	3.53
9.	0.08	0.26	0.53	0.87	1.30	1.79	2.34	2.93	3.57
10.	0.08	0.26	0.53	0.88	1.31	1.81	2.37	2.99	3.65
11.	0.08	0.26	0.53	0.88	1.32	1.82	2.39	3.02	3.69
12.	0.08	0.26	0.53	0.89	1.33	1.84	2.42	3.06	3.76
13.	0.06	0.15	0.26	0.38	0.50	0.62	0.75	0.87	0.99
14.	0.07	0.20	0.36	0.54	0.75	0.96	1.18	1.41	1.63
15.	0.08	0.21	0.40	0.63	0.89	1.17	1.46	1.77	2.09
16.	0.08	0.22	0.41	0.69	0.98	1.31	1.66	2.03	2.41
17.	0.08	0.23	0.45	0.72	1.04	1.40	1.80	2.22	2.66
18.	0.08	0.24	0.46	0.75	1.09	1.48	1.90	2.36	2.85
19.	0.08	0.24	0.47	0.77	1.12	1.53	1.98	2.47	3.00
20.	0.08	0.24	0.48	0.78	1.15	1.57	2.04	2.55	3.12
21.	0.08	0.24	0.48	0.80	1.17	1.60	2.09	2.62	3.20
22.	0.08	0.24	0.48	0.78	1.15	1.57	2.03	2.54	3.09
23.	0.08	0.24	0.47	0.77	1.12	1.52	1.97	2.45	2.97
24.	0.08	0.24	0.46	0.74	1.08	1.46	1.88	2.33	2.81
25.	0.08	0.23	0.45	0.72	1.03	1.38	1.77	2.18	2.61

Указание 1. Предложите, каким образом по данным эксперимента получить значение начального приближения установившейся скорости. В соответствии с вариантом задания выполните оценку этого значения и используйте его как начальное значение параметра.

Указание 2. Для подбора параметра моделирования (установившейся скорости) используйте метод наименьших квадратов, т.е. определите, при каком его значении результаты моделирования уравнения вертикального падения с сопротивлением пропорциональным скорости или квадрату скорости наилучшим образом согласуются с данными эксперимента.

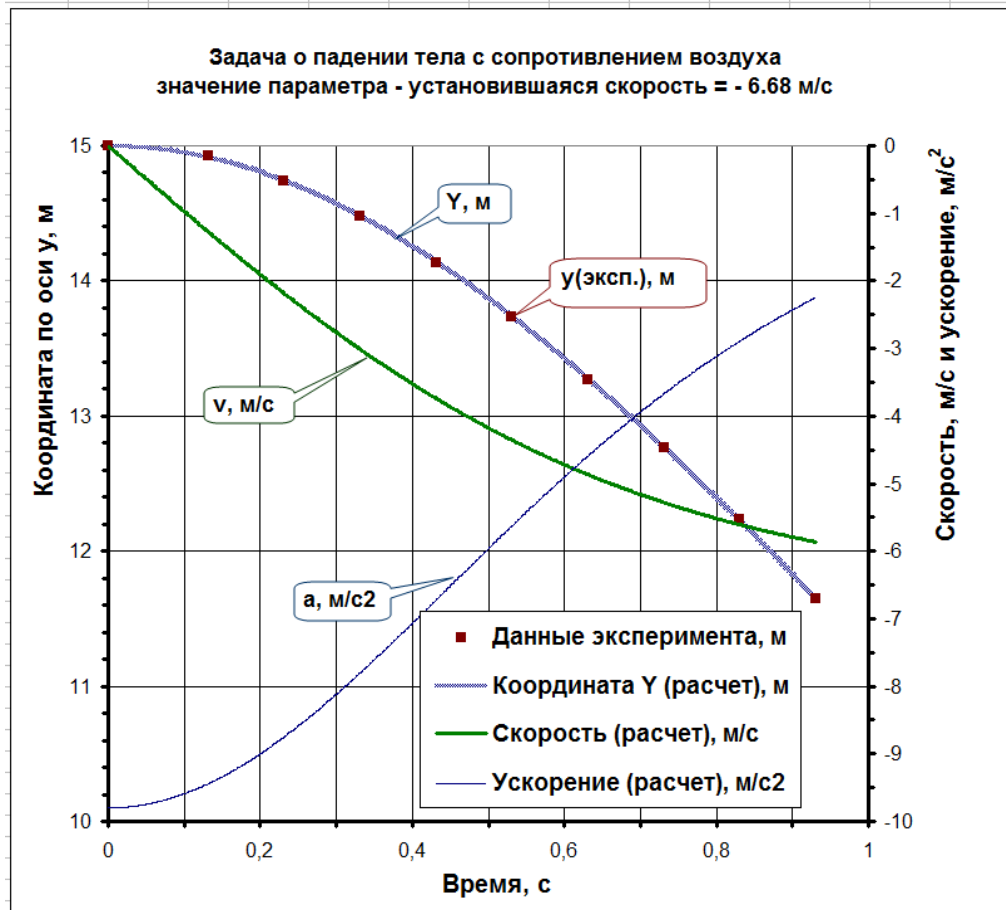


Рисунок 9.3 – Электронные таблицы с результатами расчетов и экспериментов о падении тела с сопротивлением воздуха.

Контрольные вопросы и задания повышенной сложности.

I* Используя программы вычислений из задания, проверьте, как влияет сопротивление воздуха на движение камня. Вычислите, с какой скоростью упадет в реальных условиях с высоты 50 м камень, брошенный без вертикальной начальной скорости? Примите, что тормозящая сила пропорциональна v^2 , а установившаяся скорость равна 30 м/с. Построив графики изменения координат со временем, сравните полученные результаты со свободным падением тела при тех же условиях.

II* Предположим, что тело брошено вертикально вверх с начальной скоростью v_0 . Известно, что если пренебречь сопротивлением воздуха, то максимальная высота, на которую поднимется тело равна $v_0^2/2g$, скорость, с которой тело упадет на землю равна v_0 , время подъема и время падения одинаковы, а общее время движения равно v_0/g . Прежде, чем проводить численное моделирование задачи о движении с учетом сопротивления воздуха, проведите качественное объяснение того, как изменятся в этом случае указанные величины. Проведите численные расчеты и проверьте правильность своих качественных выводов. При моделировании положите, что сила сопротивления пропорциональна квадрату скорости, а установившаяся скорость равна 30 м/с.

10. ОСНОВЫ РАБОТЫ С ГРАФИЧЕСКИМИ СРЕДСТВАМИ C++

Упражнение 10.1. Построение графика функции выполните, используя встроенные шаблоны и библиотечные функции Visual C++. Для этого выполните следующие действия.

Создайте новый проект, выбрав меню: **File->New...** и выбрав затем во вкладке **Project** обычное приложение **Win32 Application**. Задайте имя проекта, в данном случае это имя **GraficFunction** и далее нажмите **OK** и выберите **Typical "Hello World" application**. Затем нажмите **Finish** и **OK**. Далее щёлкните по любому из классов папки **Globals** во вкладке **ClassView** и увидите содержание файла **GraficFunction.cpp**, текст которого приводится ниже. Жирным шрифтом в программном коде выделены те изменения и дополнения, которые необходимо внести в шаблон (исходный код графика) для построения графика функции. Комментарии в тексте кода проясняют суть операций.

```
// GraficFunction.cpp : Определяет точку входа для приложения.
#include "stdafx.h"
#include "resource.h"
#include "Math.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;           // текущий экземпляр
TCHAR szTitle[MAX_LOADSTRING]; // текст заголовка
TCHAR szWindowClass[MAX_LOADSTRING]; // текст заголовка
// Прямые объявления функций, включенных в этот модуль кода :
ATOM MyRegisterClass(HINSTANCE hInstance);
BOOL InitInstance(HINSTANCE, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
```

```

LRESULT CALLBACK About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine,
                    int nCmdShow)
{
    // Размещаем код здесь
    MSG msg;
    HACCEL hAccelTable;
    // Initialize global strings
    LoadString(hInstance, IDS_APP_TITLE, szTitle,
MAX_LOADSTRING);
    LoadString(hInstance, IDC_GRAFICTION, szWindowClass,
MAX_LOADSTRING);
    MyRegisterClass(hInstance);
    // // Выполнить инициализацию приложения:
    if (!InitInstance (hInstance, nCmdShow))
    {
        return FALSE;
    }
    hAccelTable = LoadAccelerators(hInstance,
(LPCTSTR)IDC_GRAFICTION);
    // Основной цикл сообщений:
    while (GetMessage(&msg, NULL, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable,
&msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }
    return msg.wParam;
}

// ФУНКЦИЯ: MyRegisterClass()
// НАЗНАЧЕНИЕ: Регистрирует класс окна.
// КОММЕНТАРИИ:
// Эта функция и ее использование необходимы только в том
// случае, если вам нужен этот код, чтобы быть совместимым с
// системами Win32 до "RegisterClassEx"
// функция, которая была добавлена в Windows 95.
// Важно вызвать эту функцию, чтобы приложение получало //"хорошо сформированные"
// маленькие значки

ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEX wcx;

```

```

    wcx.cbSize = sizeof(WNDCLASSEX);
    wcx.style = CS_HREDRAW | CS_VREDRAW;
    wcx.lpfnWndProc = (WNDPROC)WndProc;
    wcx.cbClsExtra = 0;
    wcx.cbWndExtra = 0;
    wcx.hInstance = hInstance;
    wcx.hIcon = LoadIcon(hInstance,
(LPCTSTR)IDI_GRAFICFUNCTION);
    wcx.hCursor = LoadCursor(NULL, IDC_ARROW);
    wcx.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
    wcx.lpszMenuName =
(LPCTSTR)IDC_GRAFICFUNCTION;
    wcx.lpszClassName = szWindowClass;
    wcx.hIconSm = LoadIcon(wcx.hInstance,
(LPCTSTR)IDI_SMALL);
    return RegisterClassEx(&wcx);
}
//ФУНКЦИЯ: ИНИЦИАЛИЗАЦИЯ (ДЕСКРИПТОР, int)
//НАЗНАЧЕНИЕ: Сохраняет дескриптор экземпляра и создает
// главное окно
// КОММЕНТАРИИ: В этой функции мы сохраняем дескриптор
// экземпляра в глобальной переменной и создаем и отображаем // главное окно
// программы.
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    HWND hWnd;
    hInst = hInstance;
    // сохраняем дескриптор экземпляра в глобальной переменной
    hWnd = CreateWindow(szWindowClass, szTitle,
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0,
    NULL, NULL, hInstance, NULL);
    if (!hWnd)
    {
        return FALSE;
    }
    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);
    return TRUE;
}
// Вычисляемая функция
double y(double x)
{
    return sin(x)*x;
}
// ФУНКЦИЯ: WndProc (HWND - беззнаковое длинное слово)

```



```
//НАЗНАЧЕНИЕ: Обрабатывает сообщения для главного окна.
// WM_COMMAND - обработать меню приложения
//WM_PAINT - Нарисовать главное окно
//WM_DESTROY - отправить сообщение о выходе и вернуться
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
    int wmId, wmEvent;
    PAINTSTRUCT ps;
    HDC hdc;
    TCHAR szHello[MAX_LOADSTRING];
    LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);
#define xLow -5
// Нижняя граница аргумента функции
#define xHigh 5
// Верхняя граница аргумента функции
#define Step 0.1 // Шаг
#define Scale 100 // Приближение графика
    static int CenterX, CenterY; // Центр экрана
    double NumberOfPoint = (xHigh - xLow)/Step;
// вычисляем количество точек
    double x;
    POINT * pPt;
    DWORD i;
    switch (message)
    {
        case WM_COMMAND:
            wmId = LOWORD(wParam);
            wmEvent = HIWORD(wParam);
            // Parse the menu selections:
            switch (wmId)
            {
                case IDM_ABOUT:
                    DialogBox(hInst,
(LPCTSTR)IDD_ABOUTBOX, hWnd, (DLGPROC)About);
                    break;
                case IDM_EXIT:
                    DestroyWindow(hWnd);
                    break;
                default:
                    return DefWindowProc(hWnd, message, wParam,
lParam);
            }
            break;
// необходимо добавить для задания центра
case WM_SIZE:
```

```

        CenterX = LOWORD(lParam) / 2;
        CenterY = HIWORD(lParam) / 2;
        break;
    case WM_PAINT:
        // Выделяем память
        pPt = (POINT*) malloc(NumberOfPoint * sizeof(POINT));
        // просчитываем значение функции в точках
        x=xLow;
        for (i = 0; i< NumberOfPoint; i++)
        {
            pPt[i].x = (long) (CenterX + x * Scale);
            pPt[i].y = (long) (CenterY + y(x) * Scale);
            x+=Step;
        }
        hdc = BeginPaint(hWnd, &ps);
        // Рисуем оси координат
        MoveToEx(hdc, CenterX, 0, NULL);
        LineTo(hdc, CenterX, CenterY*2);
        MoveToEx(hdc, 0, CenterY, NULL);
        LineTo(hdc, CenterX*2, CenterY);
        // Рисуем функцию
        Polyline(hdc, pPt, NumberOfPoint);
        EndPaint(hWnd, &ps);
        // Очищаем память
        free(pPt);
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

// обработчик сообщений о выполнении программы
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
        case WM_INITDIALOG:
            return TRUE;
        case WM_COMMAND:
            if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
            {

```

```
EndDialog(hDlg, LOWORD(wParam));  
return TRUE;  
}  
break;  
}  
return FALSE;  
}
```

11. МЕТОД СТАТИСТИЧЕСКОГО МОДЕЛИРОВАНИЯ

Одним из мощных методов компьютерного моделирования, нашедшим широкое применение в электронике является метод статистического моделирования (метод Монте-Карло). Суть метода состоит в том, что определенные количественные характеристики модели исследуемых процессов варьируются по заданному закону вероятностного распределения. в результате формируются случайные реализации моделируемого явления. Процесс варьирования повторяется многократно, после чего осуществляется статистическая обработка результатов моделирования.

11.1 МОДЕЛИРОВАНИЕ ОТКАЗОВ

Типичный пример применения метода Монте-Карло – анализ вероятности отказа сложной системы. Такой метод может применяться для анализа влияния технологических операций изготовления электронного устройства, в частности, для расчета процента выхода годных при производстве СБИС и других элементов электронной компонентной базы.

Для иллюстрации метода рассмотрим задачу из теории массового обслуживания.

Пусть требуется найти вероятность отказа системы, состоящей из трех последовательно соединённых модулей, причем вероятность отказа каждого за заданное время работы T независима от других и составляет $1/6$. В таком случае из элементарных соображений теории вероятности получаем, что вероятность безотказной работы одного прибора составляет $5/6$, и вероятность безотказной работы трех приборов – $(5/6)^3$. Следовательно, вероятность события отказа системы: $1 - (5/6)^3 = 0,42$.

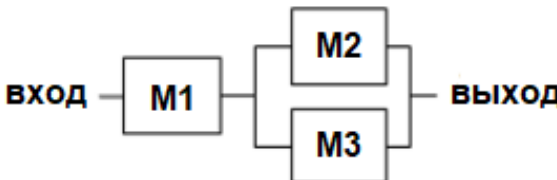
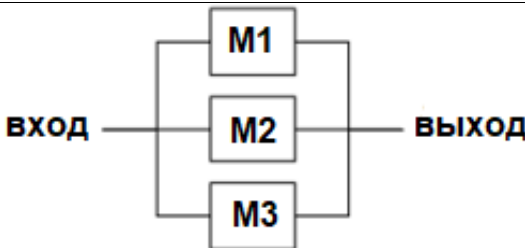
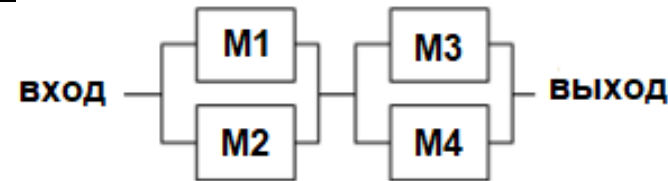
Применение метода статистического моделирования для этой задачи приводит к простой модели, отвечающей условиям данного процесса, которую можно реализовать, например, наблюдая статистически результаты бросания

трех кубиков, если каждая грань одного кубика помечена одним числом от 1 до 6 (игральные кости). При этом реализация считается отказом, если выпадут одновременно грани с числом 1. Статистический набор большого числа таких реализаций будет соответствовать вероятности отказа системы.

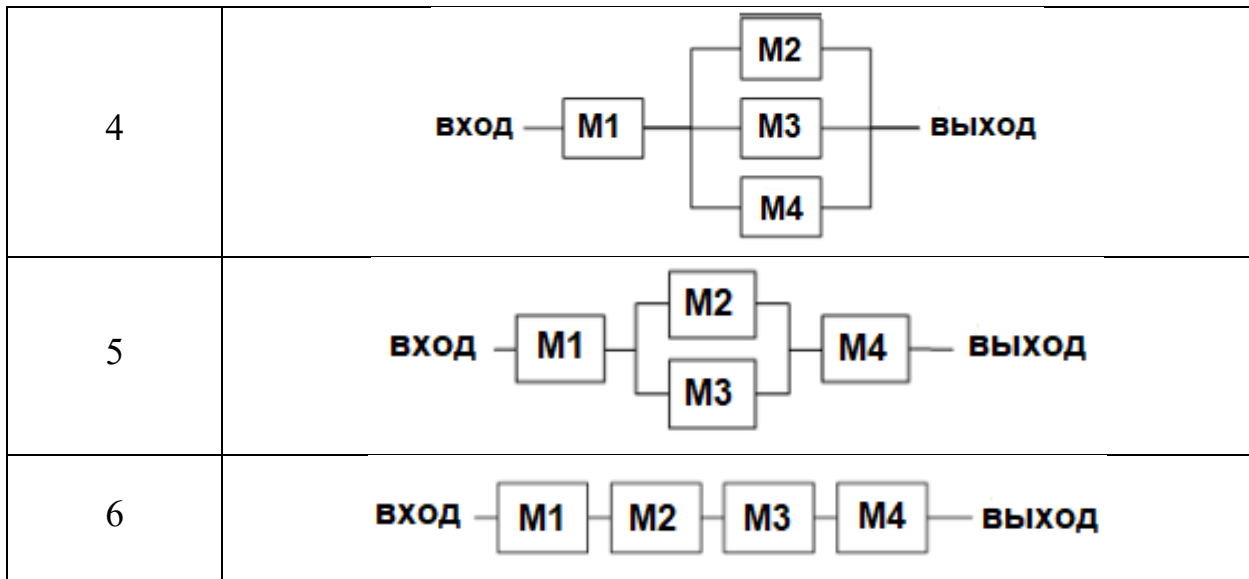
Упражнение 11.1. Разработайте и протестируйте программу, реализующую описанный алгоритм метода статистических испытаний, где входным параметром, задаваемым пользователем с консоли, является количество испытаний, а выходным параметром - вероятность отказа.

Контрольное задание. Для заданной конфигурация соединений системы из трех модулей, каждый из которых имеет определенную вероятность отказа за конкретный промежуток времени (пусть 0.1, 0.2 и 0.3 соответственно), разработать программу имитационного моделирования, определяющую вероятность отказа всей системы. Сравнить полученные результаты с аналитически расчётом, выполненным путем решения соответствующей задачи теории вероятности. Варианты задания приведены в табл.11.1

Таблица 11.1 – Варианты заданий для моделирования методом Монте-Карло

№ варианта	Конфигурация модулей
1	 <pre> graph LR ВХОД --> M1 M1 --> M2 M1 --> M3 M2 --> ВЫХОД M3 --> ВЫХОД </pre>
2	 <pre> graph LR ВХОД --> M1 ВХОД --> M2 ВХОД --> M3 M1 --> ВЫХОД M2 --> ВЫХОД M3 --> ВЫХОД </pre>
3	 <pre> graph LR ВХОД --> M1 ВХОД --> M2 M1 --> M3 M2 --> M3 M1 --> M4 M2 --> M4 M3 --> ВЫХОД M4 --> ВЫХОД </pre>

Продолжение таблицы 11.1.



Пример кода на языке C++ для выполнения задания:

```
#include <cstdio>
#include <cstdlib>
#include <ctime>
#include <iostream>

int main(int argc, char* argv[])
{
    double threshold = 0.1;

    double k1[2000] = { 0.0 };
    double k2[2000] = { 0.0 };
    double k3[2000] = { 0.0 };

    srand(time(NULL));

    int numExp = 0;
    while (numExp <= 0 || numExp > 1000)
    {
        std::cout << "Please enter number of experiments (1 - 1000):
";
        std::cin >> numExp;
    }

    int numTries = 0;
    while (numTries < 100 || numTries > 2000)
    {
```

```
std::cout << "Please enter number of tries (100 - 2000): ";
std::cin >> numTries;
}

std::cout << std::endl;

double sum = 0.0;

for (int j = 1; j <= numExp; ++j)
{
    int failureCount = 0;

    for (int i = 1; i <= numTries; ++i)
    {
        k1[i] = (double) rand() / RAND_MAX;
        k2[i] = (double) rand() / RAND_MAX;
        k3[i] = (double) rand() / RAND_MAX;

        if (k1[i] < threshold || k2[i] < threshold || k3[i] <
threshold)
        {
            ++failureCount;
        }
    }

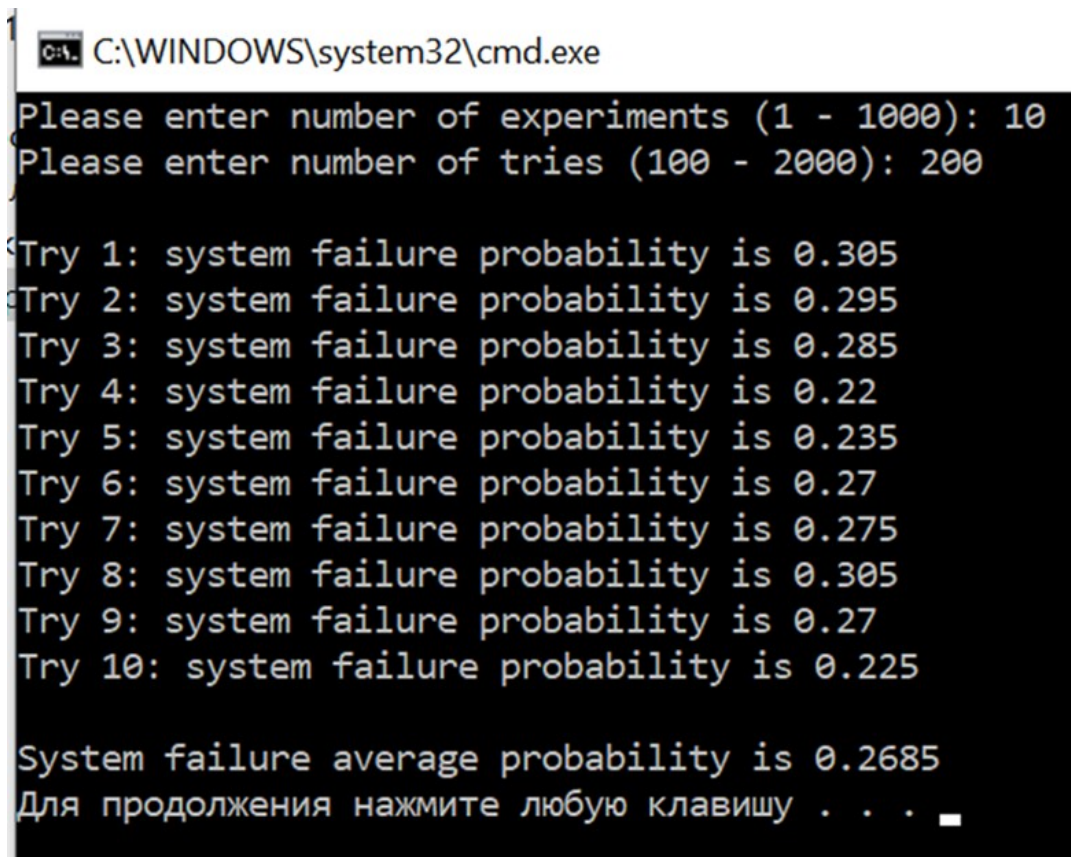
    double probFail = (double) failureCount / numTries;
    std::cout << "Try " << j << ": system failure probability is "
<< probFail << std::endl;

    sum += probFail;
}

double probFailAvg = sum / numExp;
std::cout << std::endl << "System failure average probability is "
<< probFailAvg << std::endl;

return 0;
}
```

Пример, иллюстрирующий работу программы приведен на рис. 11.1



```
C:\WINDOWS\system32\cmd.exe
Please enter number of experiments (1 - 1000): 10
Please enter number of tries (100 - 2000): 200

Try 1: system failure probability is 0.305
Try 2: system failure probability is 0.295
Try 3: system failure probability is 0.285
Try 4: system failure probability is 0.22
Try 5: system failure probability is 0.235
Try 6: system failure probability is 0.27
Try 7: system failure probability is 0.275
Try 8: system failure probability is 0.305
Try 9: system failure probability is 0.27
Try 10: system failure probability is 0.225

System failure average probability is 0.2685
Для продолжения нажмите любую клавишу . . .
```

Рисунок 11.1 – Результаты расчета отказов

Иллюстрацией сказанного является анализ допусков на элементы конструкции микроэлектромеханического гироскопа (см. рис.11.2), выполненный в соответствующем модуле САПР Coventor. Подробное описание маршрута моделирования микроэлектромеханических систем в САПР Coventor приведено в пособии Певцов Е.Ф., Деменкова Т.А., Аль-Натах Р.И. Основы моделирования и проектирования МЭМС в САПР CoventorWare: Учебное пособие [Электронный ресурс] // Московский технологический университет (МИРЭА) – М., 2016. – 98 с. ISBN 978-5-9909518-0-8.

На рис.11.3 представлены результаты, полученные методом статистического анализа, которые показывают гистограмму распределения значения собственной частоты колебаний подвеса для партии из 500 гироскопов при данных технологических допусках. Результаты полученного анализа показали, что 81,6% устройств будет иметь рабочую частоту в диапазоне ± 500 Гц от резонансной частоты.

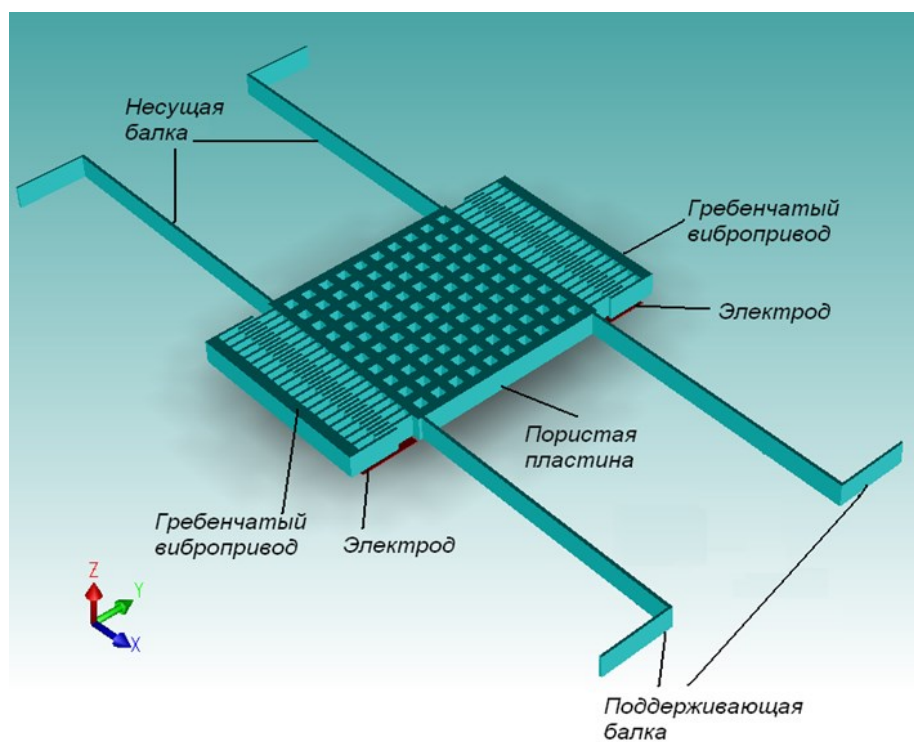


Рисунок 11.2 – 3D модель микроэлектромеханического гироскопа LL типа

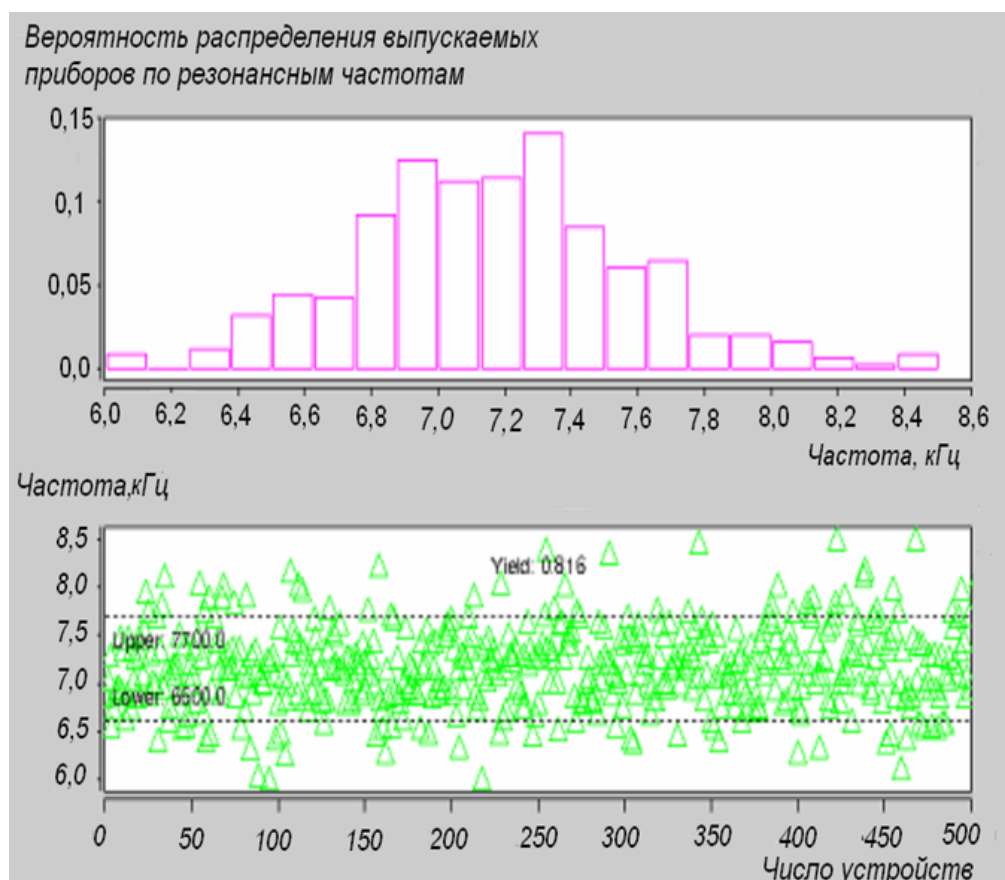


Рисунок 11.3 – Гистограмма распределения собственных частот колебаний LL гироскопа

11.2 МОДЕЛИРОВАНИЕ АЛГОРИТМА ИМИТАЦИИ ОТЖИГА

Одним из многочисленных вариантов применения метода Монте-Карло, также широко применяющимся в электронике, в частности, при проектировании топологии интегральных схем, является алгоритм имитации отжига.

Согласно этому алгоритму, система имеющая множество S всех состояний (решений) и характеризующаяся некоторым обобщенным критерием, описывающим ее состояние \underline{s}_i , например, энергией E_i , статистически подвергается изменению, приводящему к изменению состояния с новой энергией E_{i+1} . Если энергия нового состояния меньше E_i , оно становится новым текущим состоянием \underline{s}_i , в противном случае переход к состоянию \underline{s}_i реализуется с определенной степенью вероятности, характеризующей протекание моделируемого процесса. В частности, такой переход может осуществляться по экспоненциальному закону с показателем экспоненты, зависящей от текущего состояния.

Симулирование отжига в переборных задачах может быть использовано для приближённого нахождения глобального минимума функций с большим количеством свободных переменных, в частности, для размещения вентилях стандартных элементов внутри интегральной схемы в программах трассировки топологии (см. рис.11.4).

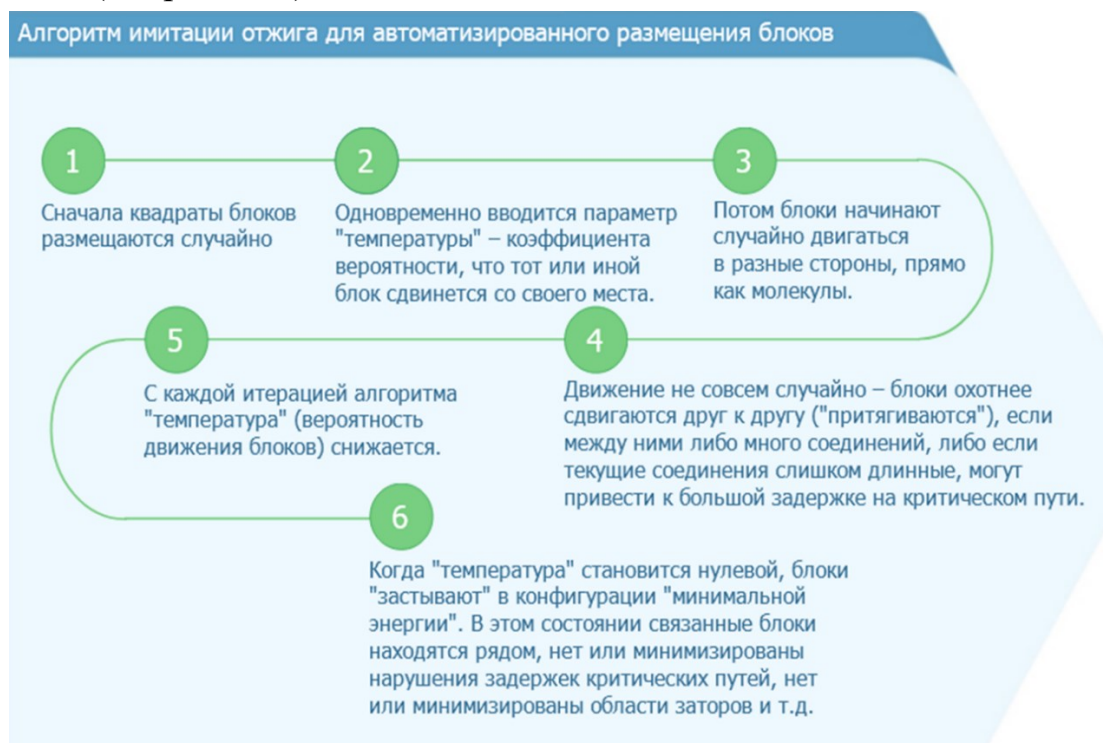


Рисунок 11.4 - Иллюстрация алгоритма имитации отжига для автоматизированного размещения стандартных ячеек СБИС

Пример реализации алгоритма отжига после размещения стандартных ячеек в чертеже топологии СБИС представлен на рис. 11.5.

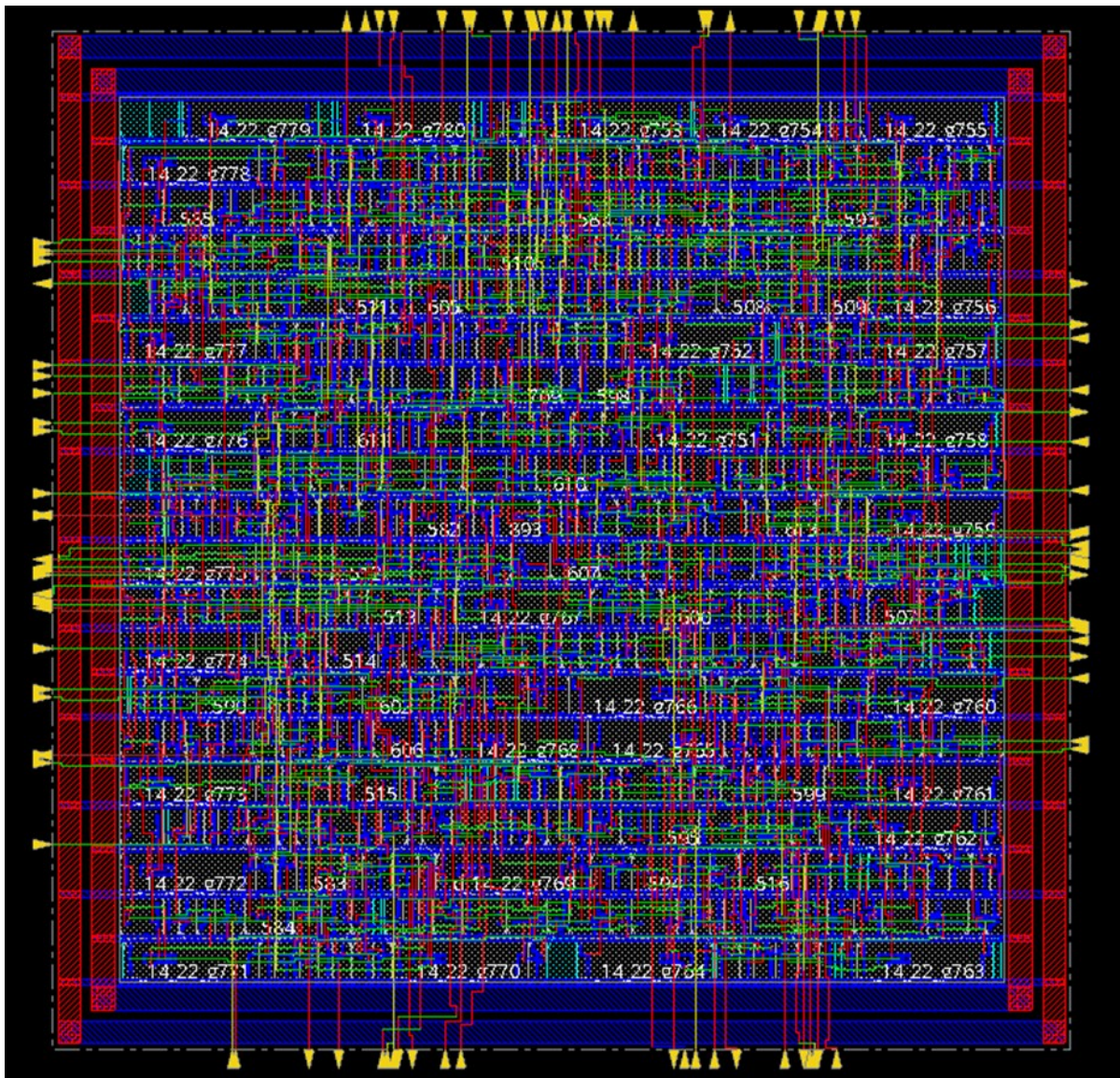


Рисунок 11.5 - Вид кристалла АЛУ после окончания синтеза и размещения стандартных ячеек

Стандартная ячейка состоит из группы транзисторов и соединений между ними, которые реализуют некоторую логическую функцию (AND, OR, XOR, XNOR, инвертор), представляющую собой законченный элемент на вентиляном уровне. Простейшие ячейки являются прямым представлением элементарных булевых функций NAND, NOR, XOR (см. пример на рис. 11.6); часто используются более сложные ячейки такие как, полный двухбитовый сумматор или мультиплексированный D-триггер.

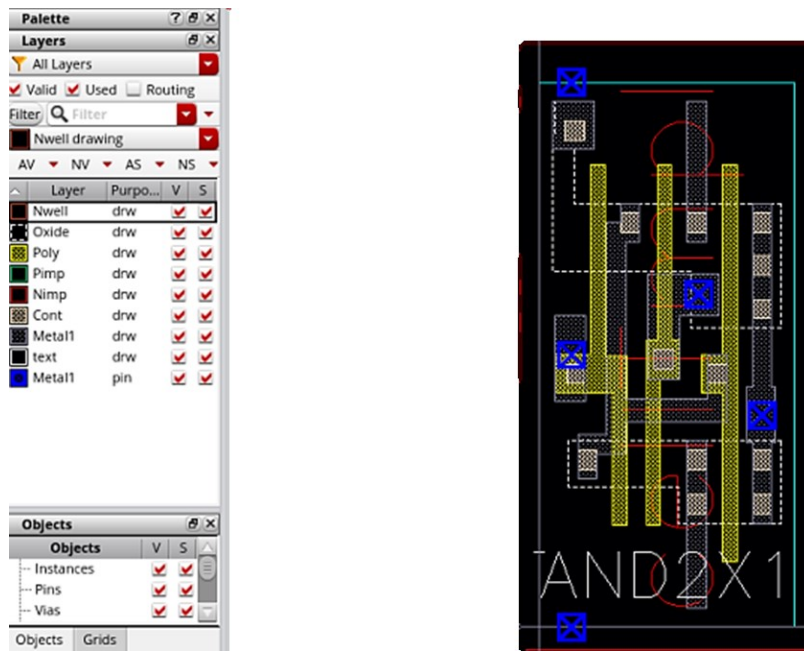


Рисунок 11.6 – Пример элементарной AND2X1 ячейки из библиотеки стандартных элементов

Классическим примером, иллюстрирующим алгоритм имитации отжига, служит задача коммивояжера.

Замечание. Для реализации автоматизированного исполнения размещения ячеек на заготовки кристалла интегральной схемы применяется специализированный скриптовый язык, представляющий собой последовательность команд для системы автоматизированного проектирования при выполнении операций синтеза топологии.

Упражнение 11.2. Задача коммивояжера. Пусть в квадрате 10x10 случайным образом разбросаны 100 точек (см. рис. 11.7).

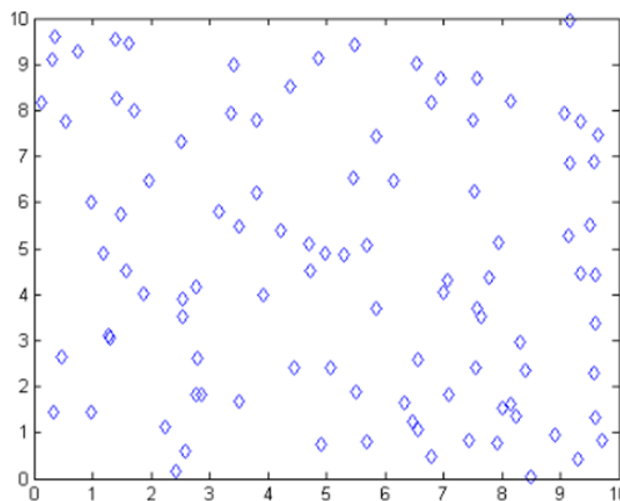


Рисунок 11.7 – Случайное распределение точек в квадрате 10x10

Каждая точка, соответственно, представлена парой координат, присвоим этим точкам соответствующие порядковые номера. Требуется найти оптимальный путь, соединяющий все точки, случайно расположенные на плоскости заданной площади, если критерием оптимизации служит порядок обхода точек, а количественным параметром для описания служит сумма всех расстояний между обходимыми точками.

Указание. Алгоритм решения состоит в следующем. Генерируем случайные координаты точек (x_i, y_i) , лежащие в заданной области. Функция, которую требуется минимизировать, представляет собой сумму:

$$E(s_i) = \sum_1^{|C|-1} \sqrt{(x_{k+1} - x_k)^2 + (y_{k+1} - y_k)^2} + \sqrt{(x_{|C|} - x_1)^2 + (y_{|C|} - y_1)^2}, \quad (10.1)$$

которая варьируется в зависимости от порядка обхода всех точек.

Здесь через $|C|$ обозначено множество всех точек маршрута обхода, а последнее слагаемое замыкает маршрут в исходной точке.

В качестве базового решения выберем случайную комбинацию последовательности из 100 чисел, обозначающих порядок обхода точек. В процессе моделирования будем пытаться улучшить эту комбинацию путем случайной перестановки пар чисел, инвертирующую порядок прохождения точек. Каждый шаг оставляем выполненную перестановку, если она приводит к уменьшению значения $E(s_i)$. Например, для маршрута (1,2,3,4,5,6,7), случайно выбираем точки с номерами 2 и 7, и, если результат проверки условия уменьшения пути, оказался отрицательным, выполняем процедуру инверсии пути и получаем новый порядок обхода (1,7,6,5,4,3,2).

Пример программы, реализующей решение задачи коммивояжера:

```
#include <stdio>
#include <stdlib>
#include <cmath>
#include <ctime>
#include <fstream>
#include <iostream>

#define CITI_COUNT 100
#define FIELD_MAX 10
```

```
#define MAX_ITERATIONS 100000

#define INITIAL_TEMPERATURE 10.0
#define MIN_TEMPERATURE 0.00001

#define EPSILON 0.000001

struct Point
{
    double x;
    double y;
};

void logPath(const char* fname, const Point* theMap, const int* path, int
size, double energy)
{
    std::ofstream fState(fname);
    for (int i = 0; i < size; ++i)
    {
        int idx = path[i];
        fState << theMap[idx].x << " ; " << theMap[idx].y << std::endl;
    }

    fState << std::endl << "Energy: " << energy << std::endl;
}

void printPath(const int* path, int size)
{
    for (int j = 0; j < size; ++j)
    {
        std::cout << path[j] << ",";
    }
}

void initializeMap(Point* theMap, int* path, int size)
{
    // generate
    for (int i = 0; i < size; ++i)
    {
        theMap[i].x = FIELD_MAX * ((double) rand() / RAND_MAX);
        theMap[i].y = FIELD_MAX * ((double) rand() / RAND_MAX);

        path[i] = i;
    }

    // shuffle
    for (int i = size - 1; i >= 0; --i)
```

```
{
    int idx = rand() % (i + 1);
    int tmp = path[i];
    path[i] = path[idx];
    path[idx] = tmp;
}
}

double calculateDistance(const Point& ptA, const Point& ptB)
{
    return sqrt((ptA.x - ptB.x) * (ptA.x - ptB.x) + (ptA.y - ptB.y) *
(ptA.y - ptB.y));
}

double calculateEnergy(Point* theMap, int* path, int size)
{
    double res = 0;
    for (int i = 0; i < size - 1; ++i)
    {
        res += calculateDistance(theMap[path[i]], theMap[path[i + 1]]);
    }

    res += calculateDistance(theMap[path[size - 1]], theMap[path[0]]);

    return res;
}

void nextPath(int* currentPath, int* nextPath, int size)
{
    for (int i = 0; i < size; ++i)
    {
        nextPath[i] = currentPath[i];
    }

    int n = rand() % size;
    int m = rand() % size;

    if (m > n)
    {
        for (int i = n; i <= m; ++i)
        {
            nextPath[i] = currentPath[m - (i - n)];
        }
    }
    else
    {
        for (int i = m; i <= n; ++i)
```

```
    {
        nextPath[i] = currentPath[n - (i - m)];
    }
}

double decreaseTemperature(double initialT, int nIteration)
{
    return initialT * 0.1 / nIteration;
}

double calculateTransitionProbability(double deltaEnergy, double
temperature)
{
    return exp(-1 * deltaEnergy / temperature);
}

bool isTransition(double deltaEnergy, double temperature)
{
    double pTran = calculateTransitionProbability(deltaEnergy,
temperature);

    double var = ((double) rand()) / RAND_MAX;

    return (var <= pTran);
}

int main(int argc, char* argv[])
{
    Point theMap[CITI_COUNT];
    int pathRg[CITI_COUNT];
    int pathCandidateRg[CITI_COUNT];

    int* path = &(pathRg[0]);
    int* pathCandidate = &(pathCandidateRg[0]);

    double temperature = INITIAL_TEMPERATURE;

    srand(time(NULL));

    initializeMap(theMap, path, CITI_COUNT);

    double minEnergy = calculateEnergy(theMap, path, CITI_COUNT);

    logPath("initial.txt", theMap, path, CITI_COUNT, minEnergy);

    for (int i = 0; i < MAX_ITERATIONS; ++i)
```

```

{
    nextPath(path, pathCandidate, CITI_COUNT);
    double nextPathEnergy = calculateEnergy(theMap, pathCandidate,
CITI_COUNT);

    if (nextPathEnergy < minEnergy && fabs(nextPathEnergy - minEnergy) >
EPSILON)
    {
        std::cout << "New min on iteration " << i << " : " << minEnergy <<
" -> " << nextPathEnergy << ", new path : ";
        printPath(pathCandidate, CITI_COUNT);
        std::cout << std::endl;

        minEnergy = nextPathEnergy;

        int* tmp = path;
        path = pathCandidate;
        pathCandidate = tmp;
    }
    else
    {
        if (isTransition(nextPathEnergy - minEnergy, temperature))
        {
            if (fabs(nextPathEnergy - minEnergy) > EPSILON)
            {
                std::cout << "Transition on iteration " << i << " : " <<
minEnergy << " -> " << nextPathEnergy << ", new path : ";
                printPath(pathCandidate, CITI_COUNT);
                std::cout << std::endl;
            }

            minEnergy = nextPathEnergy;

            int* tmp = path;
            path = pathCandidate;
            pathCandidate = tmp;
        }
    }

    temperature = decreaseTemperature(INITIAL_TEMPERATURE, i + 1);
    if (temperature <= MIN_TEMPERATURE)
    {
        break;
    }
}

logPath("final.txt", theMap, path, CITI_COUNT, minEnergy);

```



```
return 0;
}
```

Результат работы программы, выводящей в текстовый файл исходные координаты точек в случайном начальном варианте обхода и координаты точек после оптимизированного обхода приведены на рис.11.8. Внизу и указаны соответствующие суммарные пути до и после реализации алгоритма.

initial.txt – Блокнот				final.txt – Блокнот			
Файл	Правка	Формат	Вид	Файл	Правка	Формат	Ви,
2.28736 ; 6.85263				8.60622 ; 1.17588			
7.09372 ; 2.42409				7.55821 ; 1.73437			
8.05597 ; 6.94205				8.70296 ; 2.86172			
8.31416 ; 7.2454				8.95535 ; 3.03507			
9.44182 ; 2.99142				9.71648 ; 4.2732			
6.05487 ; 8.62545				8.04529 ; 6.65853			
0.434889 ; 6.3918				9.15098 ; 6.62923			
3.58928 ; 0.329905				9.46989 ; 6.27949			
4.12091 ; 5.21653				9.58312 ; 6.74673			
6.08448 ; 6.09241				9.80285 ; 7.05496			
5.55162 ; 8.4756				9.60082 ; 7.61071			
0.663167 ; 6.49342				8.52931 ; 7.94244			
4.35194 ; 4.74075				8.80184 ; 8.44295			
0.18952 ; 6.22486				8.72921 ; 8.82992			
4.63149 ; 7.9339				8.50764 ; 8.72738			
5.49577 ; 4.10566				8.51161 ; 9.03073			
7.44407 ; 5.49913				8.03308 ; 8.97702			
5.92486 ; 4.57595				8.08832 ; 8.16004			
5.53056 ; 0.865505				6.9097 ; 7.6867			
2.17719 ; 0.00366222				5.37309 ; 7.27378			
6.91885 ; 2.47841				4.73128 ; 7.24082			
5.4326 ; 3.39061				4.45418 ; 7.75414			
6.30421 ; 9.75677				4.45265 ; 8.80734			
4.01929 ; 9.71496				5.0737 ; 8.37916			
4.45784 ; 6.07013				5.75732 ; 8.6639			
9.95788 ; 2.37037				6.00848 ; 9.32798			
9.62706 ; 3.85449				5.76342 ; 9.32432			
1.04251 ; 6.58071				5.05325 ; 9.90753			
7.90948 ; 5.90869				3.08359 ; 9.79095			
Energy: 489.491				Energy: 82.1292			

Рисунок 11.8 – Результаты работы программы «Задача коммивояжера»

12. МОДЕЛИРОВАНИЕ СОСТОЯНИЙ ЧАСТИЦЫ В ПОТЕНЦИАЛЬНОЙ ЯМЕ

Задание. Разработать программы моделирования состояния частицы в потенциальной яме, определив вероятность его нахождения в различных точках ямы и его энергию при условии, что энергетическая высота ямы намного больше диапазона энергий электрона.

Варианты задания приведены в табл. 12.1

Таблица 12.1 Варианты задания для моделирования состояния частицы в потенциальной яме ($m_e = 9.1 \cdot 10^{-31}$ кг, масса электрона).

	0	1	2	3	4	5	6	7	8	9
Ширина ямы L ($l_0=10^{-9}$ м)	l_0	$2 l_0$	$4 l_0$	$8 l_0$	$10 l_0$	$0,1 l_0$	$0,2 l_0$	$0,4 l_0$	$0,8 l_0$	$0,01 l_0$
Масса m_q	m_e	$2 m_e$	$4 m_e$	$8 m_e$	$10 m_e$	$0,1 m_e$	$0,2 m_e$	$0,4 m_e$	$0,5 m_e$	$0,01 m_e$
Вид эксперимента	$E(L)$	$E(m_q)$	$E(L)$	$E(m_q)$	$E(L)$	$E(m_q)$	$E(L)$	$E(m_q)$	$E(L)$	$E(m_q)$

Указания для выполнения задания.

Положить, что потенциальная яма имеет вертикальные стенки неограниченной высоты. Для описания системы применить стационарное уравнение Шредингера и следующую модель расчета.

Уравнение Шредингера:

$$-\frac{\hbar^2}{2m} \frac{d^2\psi}{dx^2} = E\psi \quad (12.1)$$

где ψ – функция координаты x ; m – масса частицы; E – энергия.

Откуда

$$\frac{d^2\psi}{dx^2} = -\frac{2mE}{\hbar^2} \psi = -k^2\psi \quad (12.2)$$

где $k^2 = \frac{2mE}{\hbar^2}$

Решение однородного уравнения

$$\frac{d^2\psi}{dx^2} + k^2\psi = 0$$

имеет вид:

$$\psi = A'e^{\chi_1 x} + B'e^{\chi_2 x} \quad (12.3)$$

где A' и B' - константы, а χ_1 и χ_2 решения характеристического уравнения

$$\chi^2 + k^2 = 0$$

В результате получаем, что $\chi_{1,2} = \pm ik$

Т.е. применяя формулу Эйлера:

$$\psi(x) = A'e^{kx} + B'e^{-kx} = A \sin(kx) + B \cos(kx) \quad (12.4)$$

где A и B – новые константы, значения которых определяются из граничных условий: $\psi(0) = 0$ и $\psi(L) = 0$:

$$\begin{aligned} \psi(0) &= A \sin(0) + B \cos(0) = B = 0 \\ \psi(L) &= A \sin(kL) = 0 \end{aligned}$$

Откуда получаем, что $k = \frac{\pi}{L}N$, где $N=1, 2, 3, \dots$ и каждому значению N соответствует своя волновая функция

$$\psi_N = A_N \sin\left(\frac{N\pi x}{L}\right) \quad (12.5)$$

Применим условие нормировки:

$$1 = \int_{-\infty}^{\infty} \psi^2 dx = \int_{-\infty}^{\infty} A_N^2 \sin^2\left(\frac{N\pi x}{L}\right) dx = A_N^2 \frac{L}{N\pi} \int_0^{N\pi} \sin^2 \theta d\theta = A_N^2 \frac{L}{2}$$

Получим: $A_N = \sqrt{\frac{2}{L}}$ и

$$\psi_N = \sqrt{\frac{2}{L}} \sin\left(\frac{N\pi x}{L}\right) \quad (12.6)$$

$$E_N = \frac{\hbar^2 k_N^2}{2m} = N^2 \frac{h^2}{8mL^2} \quad (12.7)$$

В этих уравнениях: $\hat{h} = \frac{6,626 \cdot 10^{-34}}{2\pi}$ Дж с

Для моделирования состояния частицы разработайте программы зависимости энергии от квантового числа в соответствии с выражением (12.7) и программу, рассчитывающую волновые функции по формуле (12.6) при разных значениях параметров из таблицы 12.1

Примеры результатов моделирования, полученные путем комбинирования кода динамической библиотеки на С, встроенной в модуль виртуального прибора LabVIEW приведены на рисунках 12.1 и 12.2.

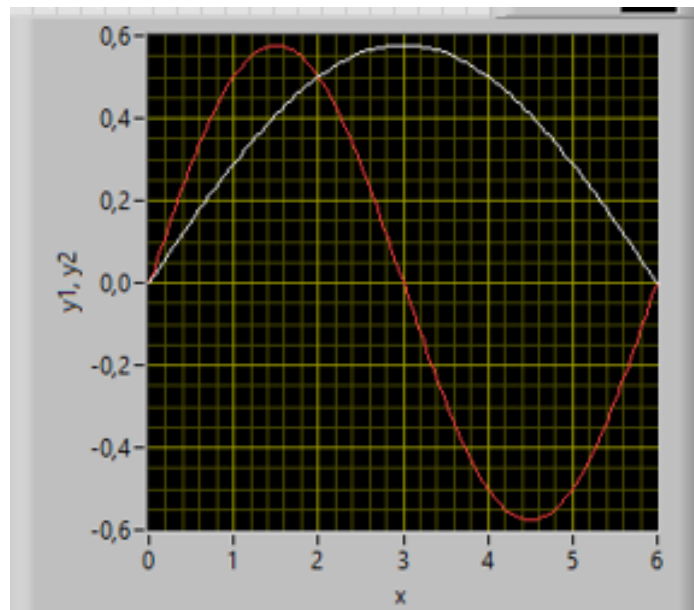


Рисунок 12.1 – Волновые функции электрона $\psi(x)$, $\text{нм}^{-1/2}$ в потенциальной яме

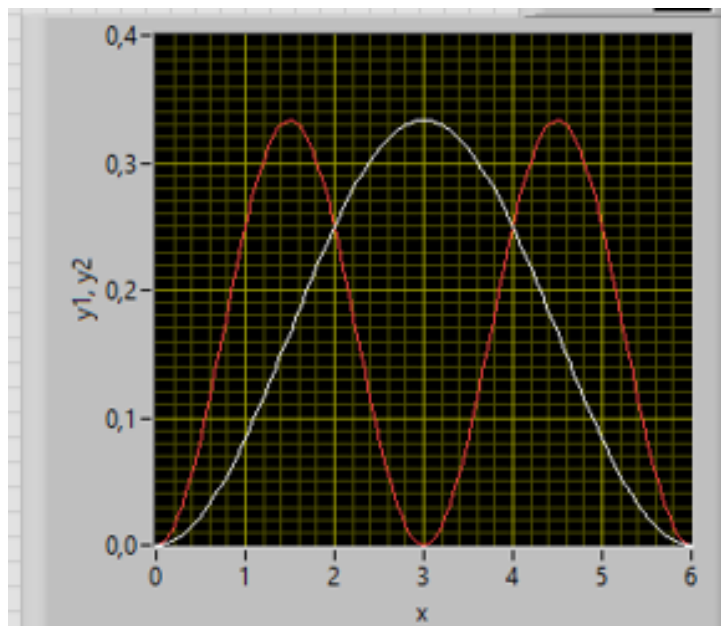


Рисунок 12.2 – Распределение плотности вероятности электрона $\psi^2(x)$, нм^{-1} нахождения электрона в потенциальной яме

Примените разработанные программы для реализации компьютерного эксперимента, показывающего влияние размера потенциальной ямы на энергию основного состояния электрона при варьировании ширины ямы в диапазоне 0,1...1000 нм.

13. МОДЕЛИРОВАНИЕ ЦИФРОВОГО ФИЛЬТРА

Задание. Выполните моделирование цифрового рекурсивного фильтра первого порядка с использованием IP ядра инструмента Vitis HLS, реализующего как ФНЧ так и ФВЧ с заданным параметром рекурсии. В качестве высокоуровневого языка программирования использовать C, C++ или System C.

Указания. Для линейных рекурсивных фильтров связь между входной последовательностью $x(n)$ и реакцией фильтра $y(n)$ записывается в виде разностного уравнения:

$$y(n) = - \sum_{v=1}^V a_{V-v} y(n-v) + \sum_{m=1}^M b_{M-m} x(n-m)$$

Согласно этому уравнению текущий отсчет $y(n)$ определяется текущим значением входной последовательности $x(n)$ и предшествующим значением выходной последовательности $y(n)$.

Для фильтра первого порядка формула преобразуется к следующему виду

$$y(n) = -a_0 y(n-1) + b_1 x(n) + b_0 x(n-1)$$

При этом должно выполняться условие $|a_0| < 1$.

Для формирования ФНЧ (фильтр нижних частот): $b_0 = b_1 = (1 + a_0)/2$

Для формирования ФВЧ (фильтр высоких частот): $b_1 = -b_0 = (1 - a_0)/2$

Vitis HLS как и Vivado HLS это инструмент высокоуровневого синтеза, который позволяет преобразовать алгоритм, описанный с помощью высокоуровневого языка программирования в структуру модулей, описанных на языке Verilog/VHDL, реализующих этот алгоритм.

Однако при работе с этим инструментом нужно помнить, что есть некоторые ограничения на входной код. В частности, нельзя применять динамическое выделение памяти (malloc и т.д.), рекурсивный вызов функций, функции операционной системы (sleep(), printf() и т.д.).

Основное применение этого инструмента – реализация IP ядер математической обработки данных, однако можно реализовать и медленные интерфейсы, такие как UART, SPI, I2C, ISO7816 и т.д.

Рекомендуемая последовательность операций для выполнения задания:

13.1. СОЗДАНИЕ НОВОГО ПРОЕКТА

Для начала необходимо запустить инструмент Vitis HLS одним из способов:



- Найти на рабочем столе иконку Vitis HLS (в некоторых случаях может потребоваться открыть папку Xilinx на рабочем столе)
- Используя командную строку ввести команду `vitis_hls` (данный метод может не сработать, если в переменные операционной системы не добавлен путь к исполняемому файлу, уточняйте у системного администратора).

Внимание Запуск инструмента может занимать длительное время, после нажатия на иконку, подождите порядка 2-3 минут, если окно загрузки инструмента не появится, повторите попытку по запуску.

После запуска, Вы увидите окно приветствия как на рисунке 13.1.

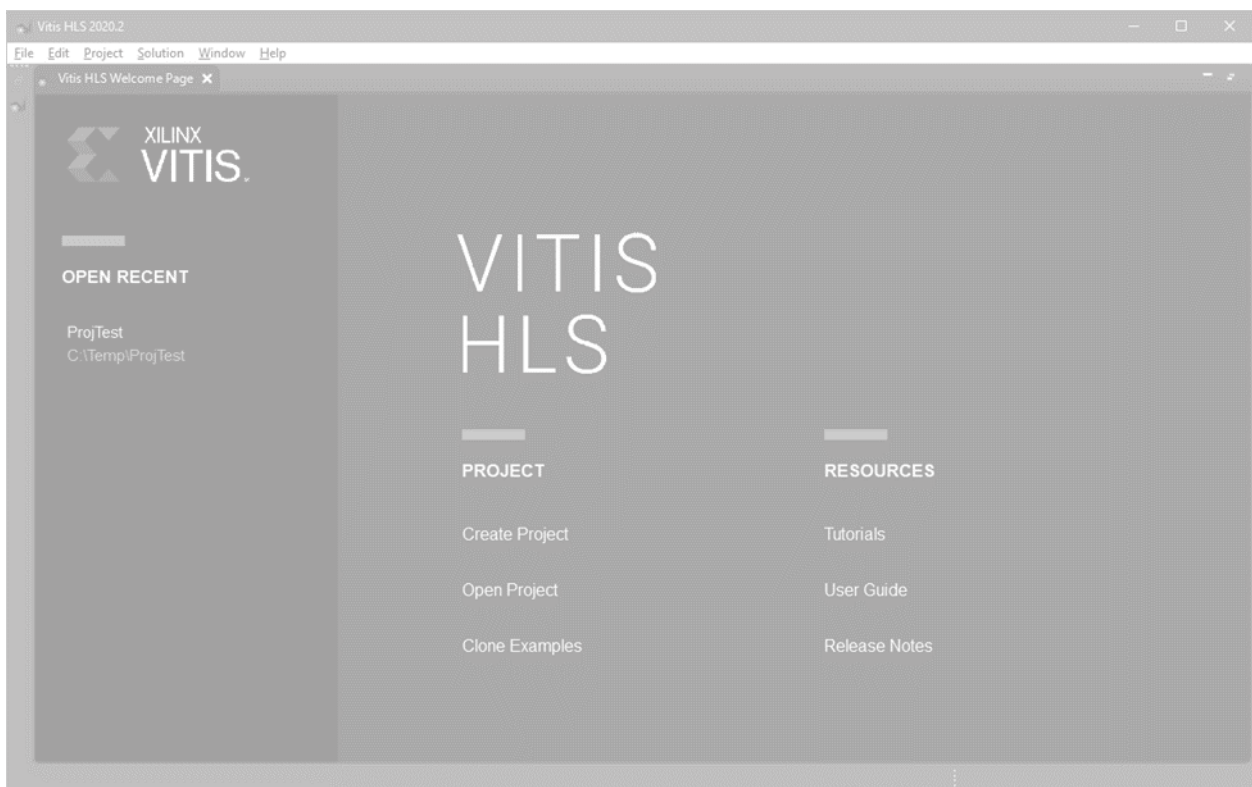


Рисунок 13.1. Окно приветствия инструмента Vivado

Левая часть окна содержит список проектов, которые были ранее открыты, в правой части список команд для быстрого перехода к созданию проекта, открытию проекта и т.д.

Создайте новый проект, для этого найдите строчку *Create Project* и кликните на неё (рис. 13.2)

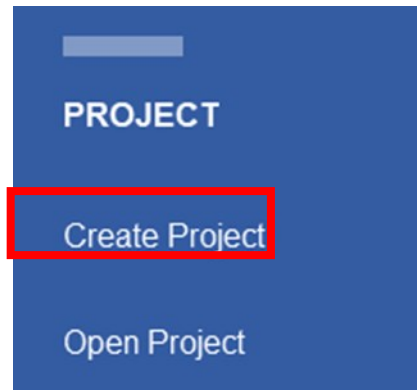


Рисунок 13.2 – Создание нового проекта

В открывшемся окне укажите название проекта (RejFilt) и путь, где будет располагаться проект (так же для этого можно воспользоваться кнопкой Browse), при этом имейте ввиду, что по указанному пути будет создана папка с названием проекта, затем нажимаем кнопку **Next** (см. рис. 13.3).

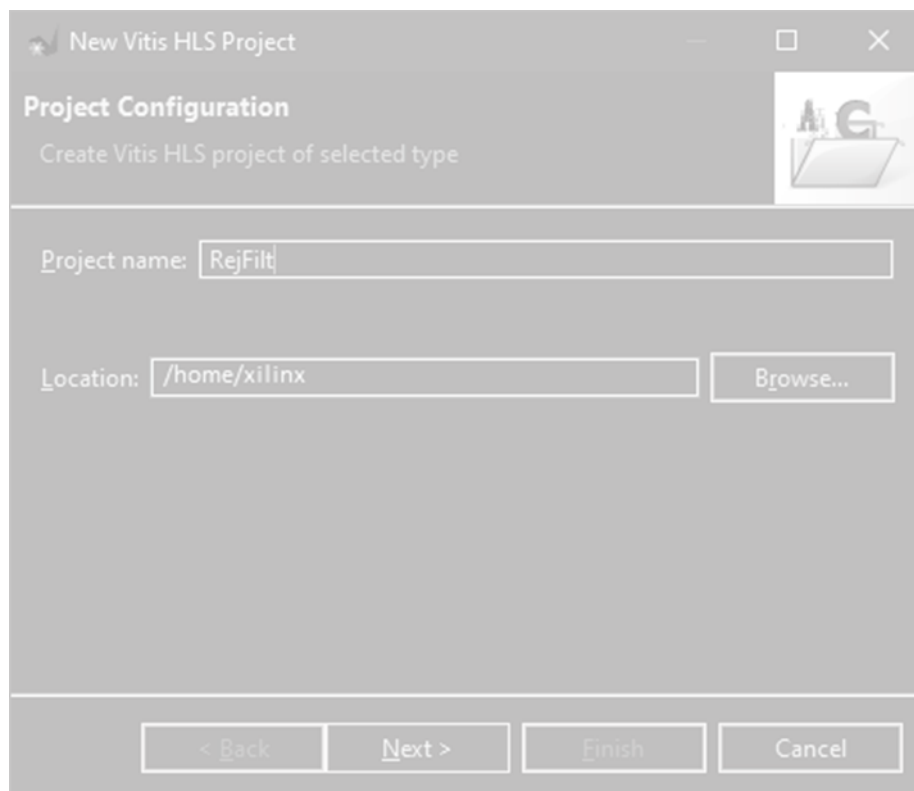


Рисунок 13.3 – Задание локации проекта

В следующем окне (рис.13.4) необходимо указать имя для основной функции, которая будет синтезироваться (аналог функции main для обычного

ПО), однако это имя не должно быть `main`, так как оно используется при запуске симуляции и со-симуляции.

Введите название функции `RejectFilter` и нажмите кнопку **Next**

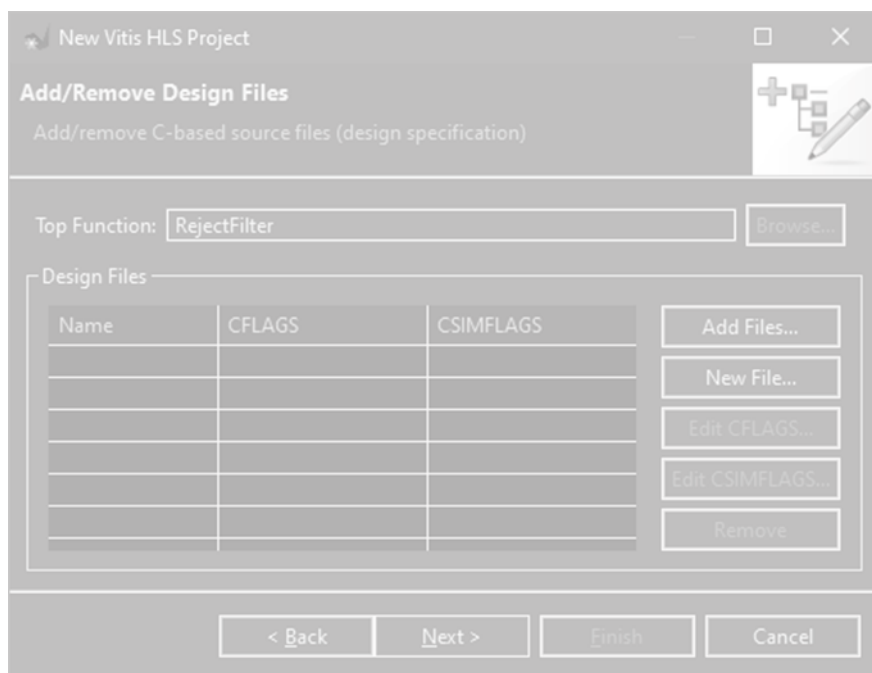


Рисунок 13.4 – Задание имени проекта

Нажимаем кнопку **Next**, и переходим к следующему окну задани файла проекта (см. рис. 13.5).

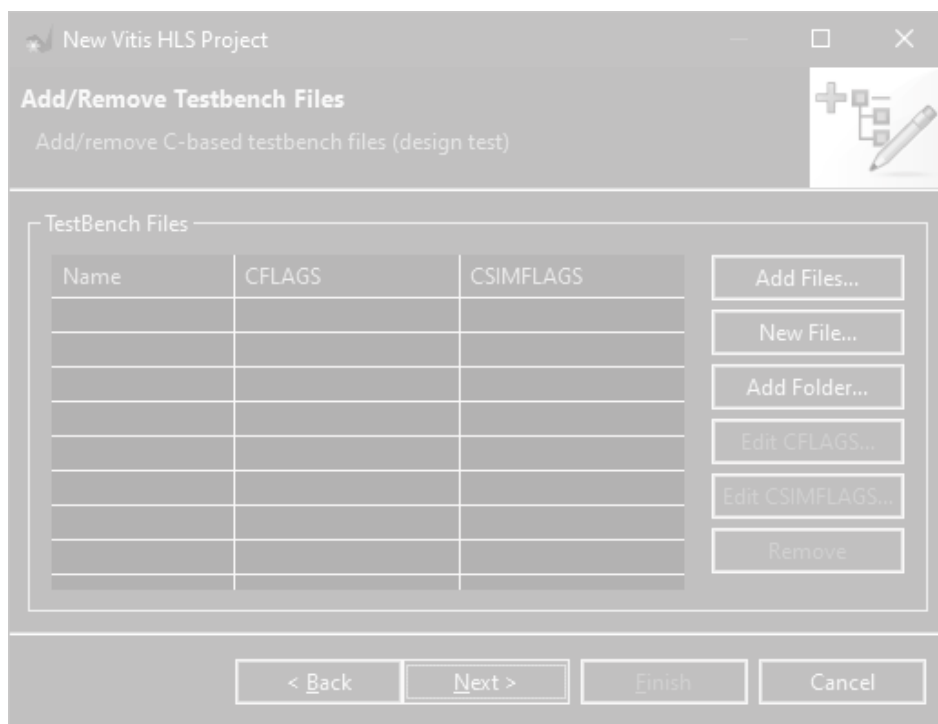


Рисунок 13.5 – Задание файла проекта

Последнее окно (рис. 13.6) позволяет настроить параметры проекта.

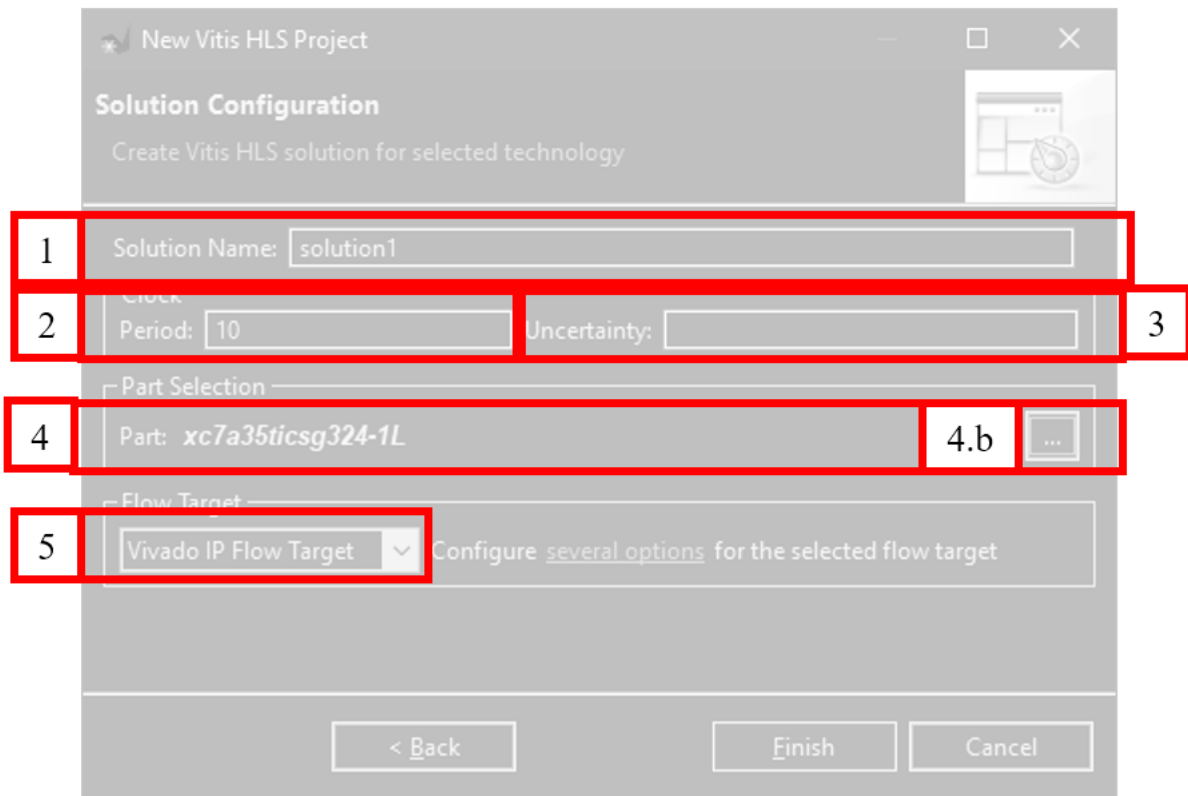


Рисунок 13.6 – Настройка параметров проекта

А именно:

1. Название первого файла Solution (в одном проекте их может быть несколько).
2. Указать желаемую максимальную частоту (в формате <значение>MHz) или минимальный период (указывается в наносекундах, приставки не требуется) для тактового сигнала, которым планируется тактировать будущий модуль. Эта информация используется для распределения операций по конечному автомату. Таким образом, чем выше максимальная частота, тем больше тактов может потребоваться для “безопасной” реализации. Таким образом этот параметр нужно выбирать аккуратно.
3. Нестабильность – этот параметр позволяет уточнить насколько “нестабильным” планируется входной тактовый сигнал. Задается в % от тактового сигнала, значение по-умолчанию (если поле пустое) составляет 12.5%.
4. Указать целевой кристалл для эффективного применения примитивов выбранного кристалла. В случае, если Вы создаете проект из TCL скрипта, то Вам будет доступна возможность указать семейство, а не конкретный кристалл.

5. Назначение – этот параметр позволяет определить, как Вы планируете использовать результат синтеза, как отдельное IP ядро, или как систему ускорения расчетов для проектов Vitis.

Все настройки, кроме (4), нужно оставить без изменений. Выбор кристалла нужно задать, вызвав кнопкой с многоточием (4.b) контекстное меню. В результате откроется окно выбора кристалла (рис. 13.7).

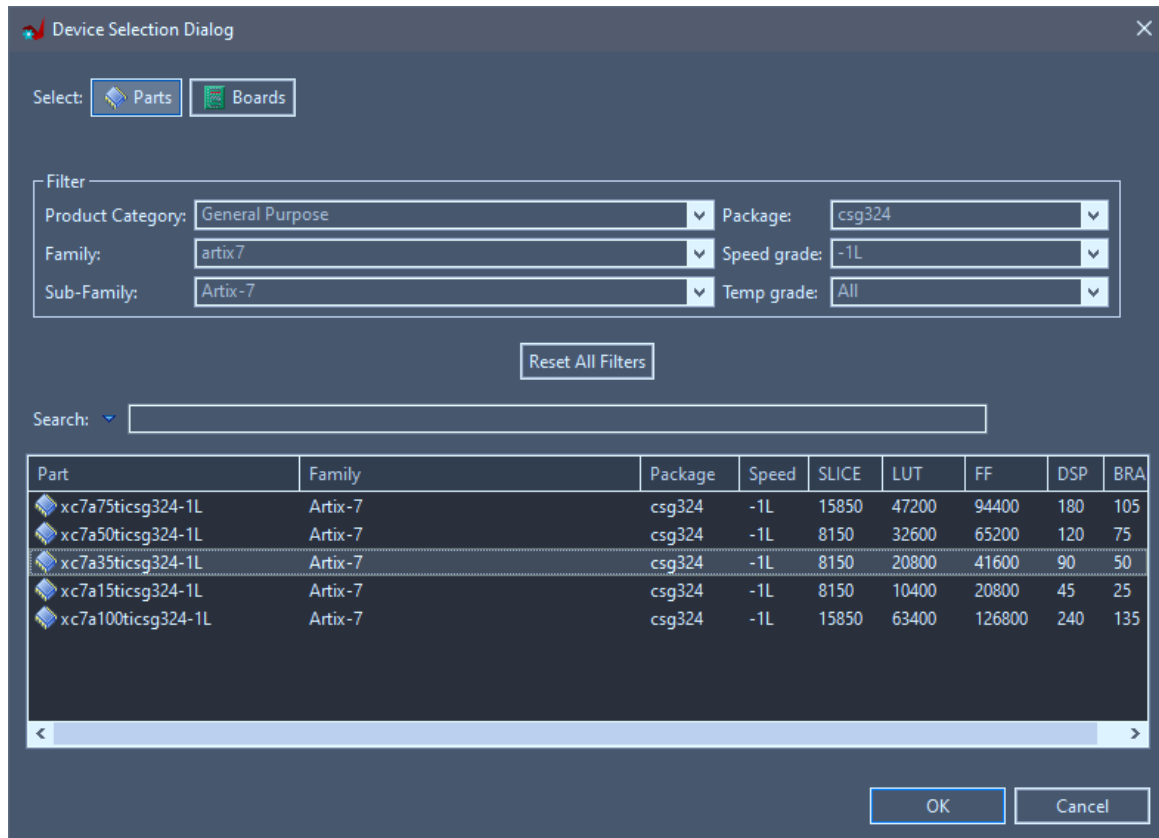


Рисунок 13.7 – Выбор типа микросхемы ПЛИС

Пусть на плате установлена микросхема ПЛИС **XC7A35TICSG324-1L**. Для того, что бы проще было найти нужный кристалл можно скрыть лишние кристаллы, используя один из следующих методов:

- воспользоваться полем поиска (**Search**) введя в него нужную маркировку
- уточнить информацию по кристаллу используя выпадающие списки:
 - a. **Product Category** -> General Purpose
 - b. **Family** -> artix7
 - c. **Sub-Family** -> Artix-7
 - d. **Package** -> csg324
 - e. **Speed grade**-> -1L

После выполнения всех настроек нажмите кнопку **Ok** и кнопку **Finish**. Откроется основное окно проекта в режиме Synthesis (см. рис.13.8). В левой части располагается дерево проекта, там же мы можем найти все доступные Solution для данного проекта. В нижней части окна вкладки с логированием, информацией о найденных ошибках и т.д.

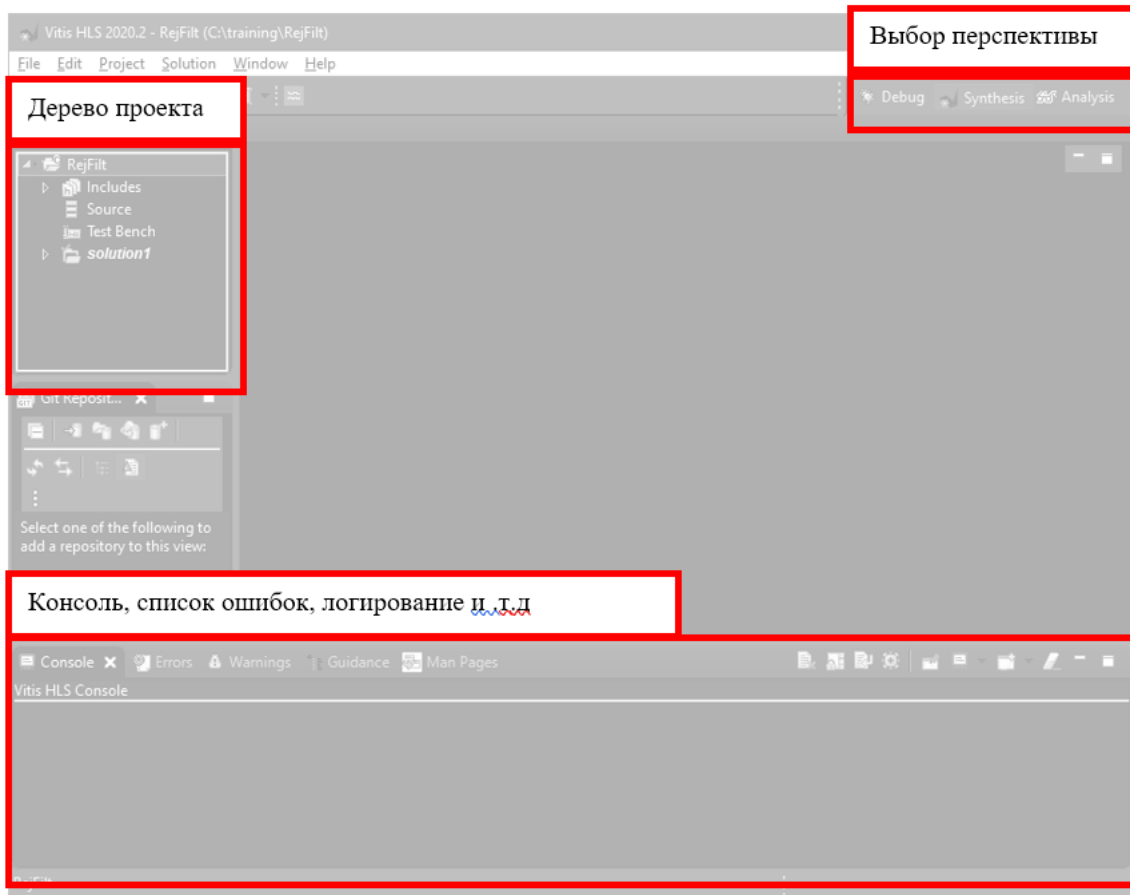


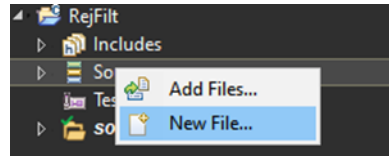
Рисунок 13.8 – Основное окно проекта

Особенность Vitis HLS заключается в том, что GUI (Graphical User Interface) основан на базе Eclipse IDE. Этому отвечает идеология построения рабочего окна, которое состоит из вложенных окон (в рамках данного IDE они называются View). Компоновка этих вложенных окон образует перспективу (Perspective). В самом инструменте содержится три предустановленные компоновки. Для просмотра полного списка, а так же дополнительных окон следует воспользоваться меню **Window**. Так же через это меню можно восстановить компоновку вложенных окон, для этого следует выбрать нужную нам компоновку и перейти в меню **Window->Reset Perspective**.

13.2 ФАЙЛ СИНТЕЗА ПРОЕКТА

Добавим файлы, которые будут использоваться для синтеза в проект. Для этого необходимо сделать одно из следующих действий:

- Вызвать контекстное меню для элемента *Source* в дереве проекта и выбрать **New File...**



- Через меню **Project->New Source...**

Откроется окно для выбора места сохранения для нового файла (см.рис.13.9).

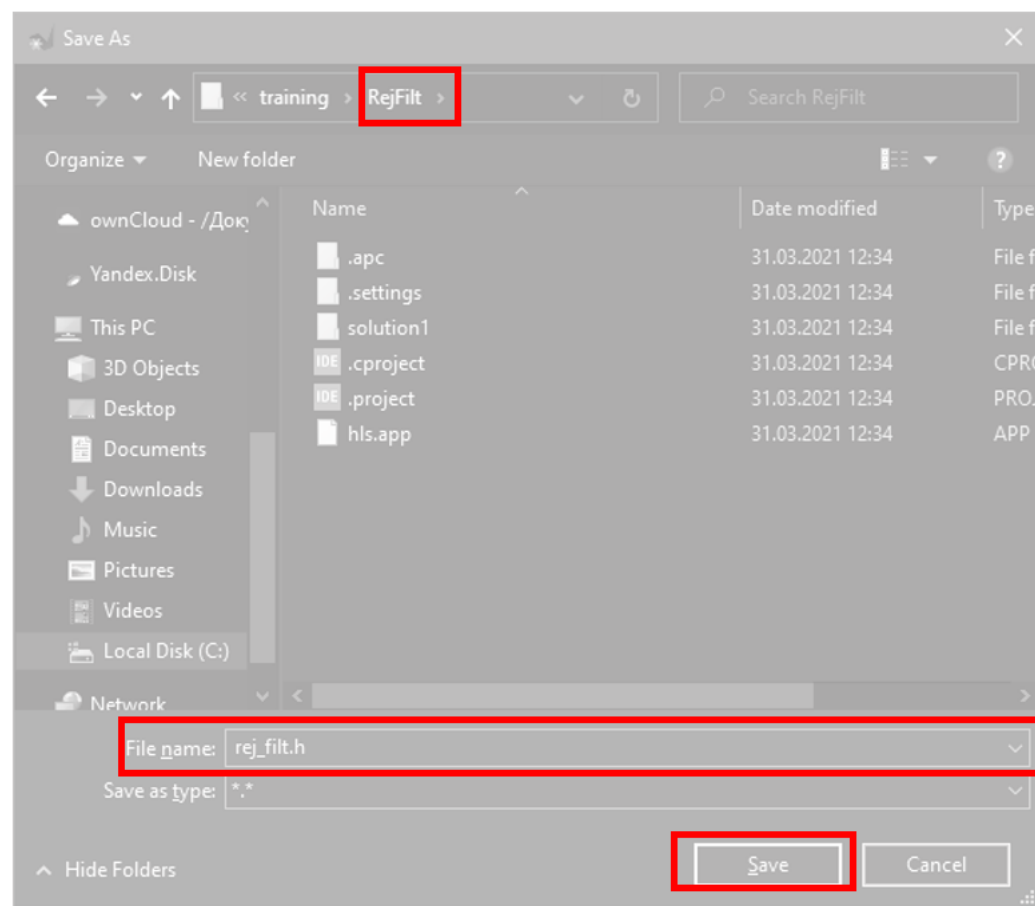


Рисунок 13.9 - окно для выбора места сохранения для нового файла

Откройте папку с названием проекта (*RejFilt*) и в поле File name впишите *rej_filt.h* и нажмите кнопку **Save**. После этого вы увидите пустой файл *rej_filt.h* открытый для редактирования. Также появится два окна в правой части (Outline и

Directive). Обратите внимание, что файл появился в дереве проекта в разделе Source.

Запишите в файл:

```

1 #ifndef _REJ_FILT_H_
2 #define _REJ_FILT_H_
3
4 // Precision type for integer with width of bit from 1 to 1024
5 #include <ap_int.h>
6 // Fixed-point type and calculation
7 #include <ap_fixed.h>
8 // Stream interface description
9 #include <hls_stream.h>
10
11 #define BIT_WIDTH 14
12 #define A -0.8
13
14 typedef ap_int<BIT_WIDTH> iItm;
15 typedef ap_fixed<BIT_WIDTH*2, BIT_WIDTH> fxItm;
16 typedef hls::stream<iItm> tStream;
17
18 void RejectFilter (tStream *in, bool hFn1F, tStream *out);
19
20 #endif

```

Примечание: Для простоты ввода, не забывайте использовать комбинацию клавиш Ctrl+Space. Данная комбинация активирует автозаполнение, если вариант однозначно определен, либо предлагает выпадающий список, из которого можно выбрать, доступный вариант. Работает для ключевых слов, переменных и функций входящих в библиотеку, определенных в ранее включенном файле и т.д.

Нужно отметить, что в состав Vitis HLS включены файлы, оптимизированные для синтеза.

В частности, заголовочный файл *ap_int.h* позволяет использовать целочисленные переменные, с указанным количеством бит (диапазон от 1 до 1024).

Файл *ap_fixed.h* позволяет работать с числами с фиксированной точкой. Это числа, являются альтернативной для чисел с плавающей точкой, и рекомендуется применять их, для повышения быстродействия и снижения требуемых для реализации ресурсов.

Файл *hls_stream.h* позволяет реализовывать потоковые интерфейсы. Без дополнительных настроек, создается порт для взаимодействия с FIFO.

Макрос **BIT_WIDTH** позволяет определять ширину входной и выходной шин данных.

В строке кода 15 объявлен пользовательский тип на основе числа с фиксированной точкой (см. рис.13.10).

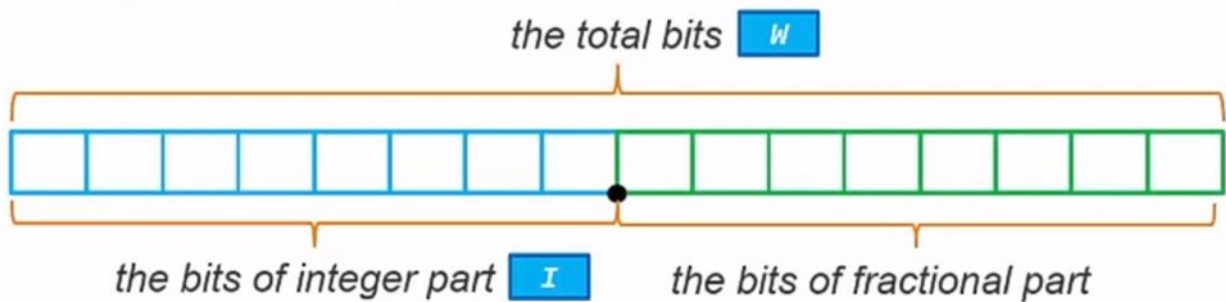


Рисунок 13.10 – Формат числа с фиксированной точкой

Этот формат в общем случае в данной среде проектирования выглядит так:

Для знаковых чисел: `ap_fixed<W, I, Q, O, N>`.

Для без знаковых чисел: `ap_ufixed<W, I, Q, O, N>`.

где W – общее количество бит; I – количество бит для целой части; Q – режим квантизации (определяет правило округления и отбрасывания чисел); O – поведение при переполнении (определяет что будет происходить, когда результат требует больше бит, чем доступно); N – количество бит сатурации (используется совместно с предыдущим параметром).

Все параметры кроме первых двух имеют значение по умолчанию. Таким образом, в некоторых случаях достаточно указать только общее количество бит, и положение двоичной точки.

Сохраните изменения:

- Комбинацией клавиш `Ctrl + s`
- Меню **File->Save**

Создайте еще один файл `rej_filt.cpp` с реализацией фильтров. Обратите внимание, что диалоговое окно сохранения файла будет открыто в той папке, которую ранее использовали, поэтому теперь достаточно указать имя файла и нажать на кнопку **Save**.

Впишите в него следующий код:

```
1 #include "rej_filt.h"
2
3 void RejectFilter (tStream *in, bool hFnlf, tStream *out)
4 {
5     const fxItm cA = A;
6     const fxItm cB0_hf = ((A - 1.0)/2.0);
7     const fxItm cB1_hf = -cB0_hf;
8     const fxItm cB0_lf = ((1.0 + A)/2.0);
9     const fxItm cB1_lf = cB0_lf;
```

```

10
11     static fxItm y0 = 0.0;
12     static fxItm x0 = 0.0;
13
14     fxItm tmp, tmp1, B0, B1;
15
16     // Read new data and convert from integer to fixed point
17     tmp (BIT_WIDTH*2-1, BIT_WIDTH) = in->read();
18
19     if (hFn1F)
20     {
21         B0 = cB0_hf;
22         B1 = cB1_hf;
23     }
24     else
25     {
26         B0 = cB0_lf;
27         B1 = cB1_lf;
28     }
29
30     // Do filtration
31     tmp1 = -cA*y0 + B1*tmp + B0*x0;
32
33     // Save data for use in next step
34     x0=tmp;
35     y0=tmp1;
36
37     // Write result out as integer
38     out->write(tmp1 (BIT_WIDTH*2-1, BIT_WIDTH));
39 }

```

В строках с 5 по 9 объявлены коэффициенты фильтров. Так как коэффициенты не должны высчитываться динамически, они объявляются с ключевым словом `const`.

В 17 строке происходит считывание данных из входного интерфейса, в старшие биты переменной `tmp`, а именно в целую часть. Затем происходит проверка, какой тип фильтра необходим (аргумент `hFn1F`), в зависимости от его значения, коэффициенты `B0` и `B1` инициализируются для реализации ФВЧ или ФНЧ. Далее в коде происходит непосредственно расчёт нового значения, сохранение текущего входного значения и отклика, для следующего расчёта, и передача данных.

13.3 ПОДГОТОВКА И ПРОВЕРКА КОДА

Vitis HLS позволяет реализовать проверку для результат работы синтезируемой функции. При этом эта проверка может выполняться как до синтеза (что равносильно простой сборке исполняемого файла для ПК) так и после синтеза. В последнем случае, синтезируемый модуль запускается в

симуляторе RTL кода, а для передачи тестовых воздействий и получения откликов используются специальные адаптеры между тестовым кодом и симулятором RTL. Это режим называется Co-Simulation (см. рис. 13.11).

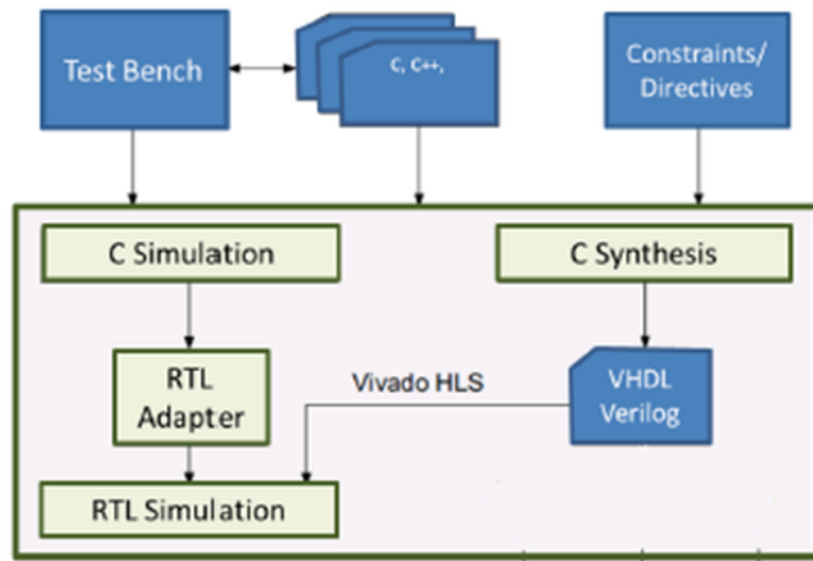


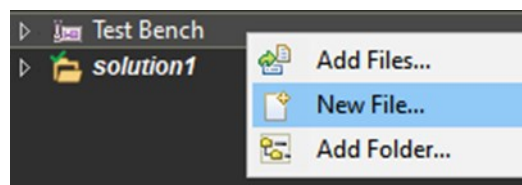
Рисунок 13.11 – Структура проекта

Как отмечено выше, основной функцией для запуска тестирования является функция **main**. При этом эта функция должна возвращать целочисленные значения, т.е. объявлена как **int main()**. Возвращаемое значение, как и в случае простых приложений являет собой код ошибки. Если функция возвращает 0, то симуляция считается завершенной успешно, если нет, то проверка считается проваленной.

Так как используемый для тестирования не синтезируется, то на него не накладывается никаких ограничений.

Для добавления файла тестирования необходимо сделать одно из двух:

- Вызвать контекстное меню для элемента *Test Bench* в дереве проекта и выбрать **New File...**



- Через меню **Project->New Test Bench...**

Далее откроется окно, для сохранения файла, такое же как и в предыдущих случаях. Убедитесь, что Вы будете сохранять файл в папке с проектом **RejFilt**, задайте имя файла **main.cpp** и нажмите кнопку Save.

Исходный код для тестирования:

```

#include <random>

#include "rej_filt.h"

#define RND_GEN_MIN 0
#define RND_GEN_MAX 10

int main()
{
    double fTmp;
    // Random generator
    std::uniform_real_distribution<double> unif(RND_GEN_MIN, RND_GEN_MAX);
    std::default_random_engine re;

    tStream in, out;
    iItm tmp;

    // Init random generator
    srand(static_cast<int>(time(0)));

    printf("Test low-frequency filter\n");
    for(int i=0; i<100; i++)
    {
        fTmp = unif(re);
        fTmp += i;
        in.write(fTmp);
        printf("Input %lf, ", fTmp);
        RejectFilter(&in, false, &out);
        fTmp = out.read();
        printf(" Output %lf\n", fTmp);
    }

    printf("Test high-frequency filter\n");
    for(int i=0; i<100; i++)
    {
        fTmp = unif(re);
        fTmp += i;
        in.write(fTmp);
        printf("Input %lf, ", fTmp);
        RejectFilter(&in, true, &out);
        fTmp = out.read();
        printf(" Output %lf\n", fTmp);
    }

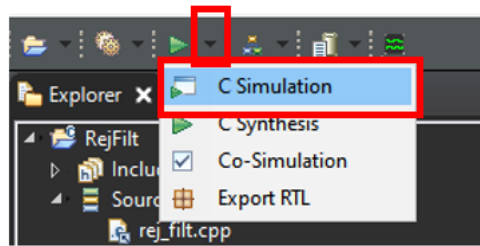
    return 0;
}

```

После завершения редактирования кода, запустите симуляцию до синтеза.

Для этого можно выполнять одно из следующих действий:

- Выберите в меню **Project-> Run C Simulation**.
- На панели инструментов, нажмите на черную стрелку рядом с зеленой и выберите C Simulation.



После чего появится окно настройки симуляции (рис.13.12):

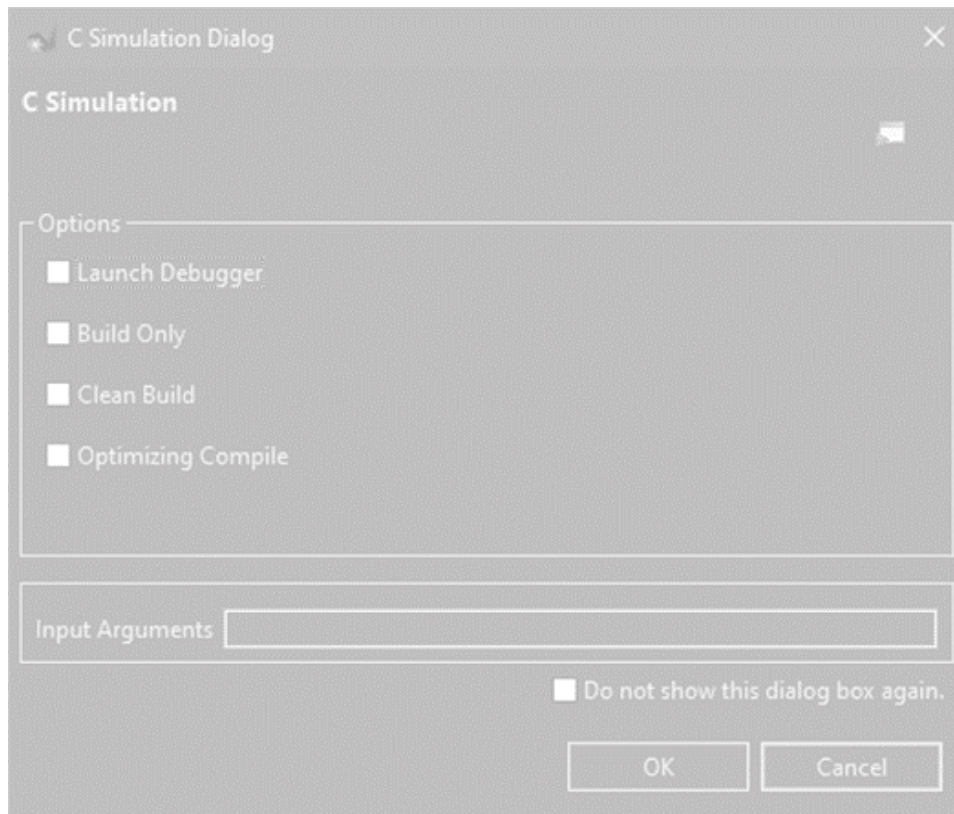


Рисунок 13.12 – Окно настройки симуляции

В этом окне можно отметить запуск в режиме отладки (опция Launch Debugger), очистка ранее собранных файлов т.е., произвести сборку проекта “с нуля”,- опция Clean Build и так далее. Так же тут можно указать аргументы для функции *main*. Их значения можно получить, если функция *main* объявлена так: `int main(int argc, char* argv[])`. Вы можете поставить отметку Do not show the dialog box again, и тогда при следующем запуске, это окно не появится.

Примечание

Если отключено появление диалогового окна настройки симуляции, для внесения изменений необходимо будет:

1. Выбрать меню **Project->Project Settings**
2. В новом окне выбрать раздел Simulation (см. рис. 13.13).

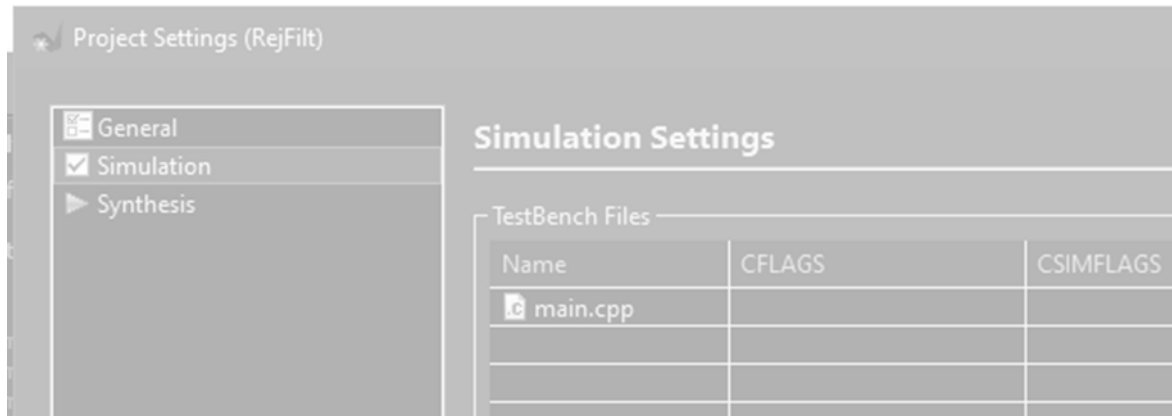


Рисунок 13.13 – Окно настройки симуляции для внесения изменений

3. Внести необходимые изменения

4. Нажать на кнопку Ok

Оставляем все настройки без изменений, и нажимаем на кнопку **Ok**. Спустя некоторое время, должен открыться файл *RejectFilter_csim.log*, следующего содержания (входные данные могут незначительно отличаться, так как для их формирования используется генератор случайных чисел):

```
INFO: [SIM 2] ***** CSIM start *****
INFO: [SIM 4] CSIM will launch GCC as the compiler.
  Compiling ../../main.cpp in debug mode
  Compiling ../../rej_filt.cpp in debug mode
  Generating csim.exe
Test low-frequency filter
Input 1.315378, Output 0.000000
Input 5.586501, Output 0.000000
Input 4.189592, Output 1.000000
Input 9.788647, Output 2.000000
Input 13.346929, Output 4.000000
Input 10.194164, Output 5.000000
Input 6.345721, Output 6.000000
Input 12.297002, Output 7.000000
Input 8.076982, Output 7.000000
Input 9.668422, Output 7.000000
Input 16.867727, Output 8.000000
Input 20.304365, Output 10.000000
Input 17.269288, Output 12.000000
Input 19.539190, Output 13.000000

...

Input 90.653390, Output 87.000000
Input 89.351094, Output 87.000000
Input 93.553073, Output 88.000000
Input 94.523002, Output 89.000000
Input 100.316744, Output 91.000000
Input 94.152484, Output 92.000000
```

```

Input 102.089219, Output 93.000000
Input 102.608598, Output 95.000000
Input 100.059559, Output 96.000000
Input 104.175615, Output 97.000000
Input 101.622450, Output 98.000000
Input 104.327387, Output 99.000000
Input 107.246974, Output 100.000000
Test high-frequency filter
Input 7.022066, Output -10.000000
Input 10.544149, Output -5.000000
Input 4.893162, Output -10.000000
Input 8.144347, Output -4.000000
Input 8.140285, Output -3.000000
Input 13.765657, Output 2.000000
Input 13.297477, Output 1.000000
Input 14.156423, Output 2.000000
Input 15.065355, Output 2.000000
Input 9.190924, Output -4.000000

...

Input 103.440227, Output 5.000000
Input 101.158259, Output 2.000000
The maximum depth reached by any of the 2 ::stream() instances in the design is 1
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] ***** CSIM finish *****

```

Если будет отображено что-то совсем другое, посмотрите внимательно на содержимое файла, в нем будет отмечена ошибка. Для поиска ошибки и быстрого перехода к предполагаемому месту в коде можно воспользоваться вкладкой Console (рис. 13.14).

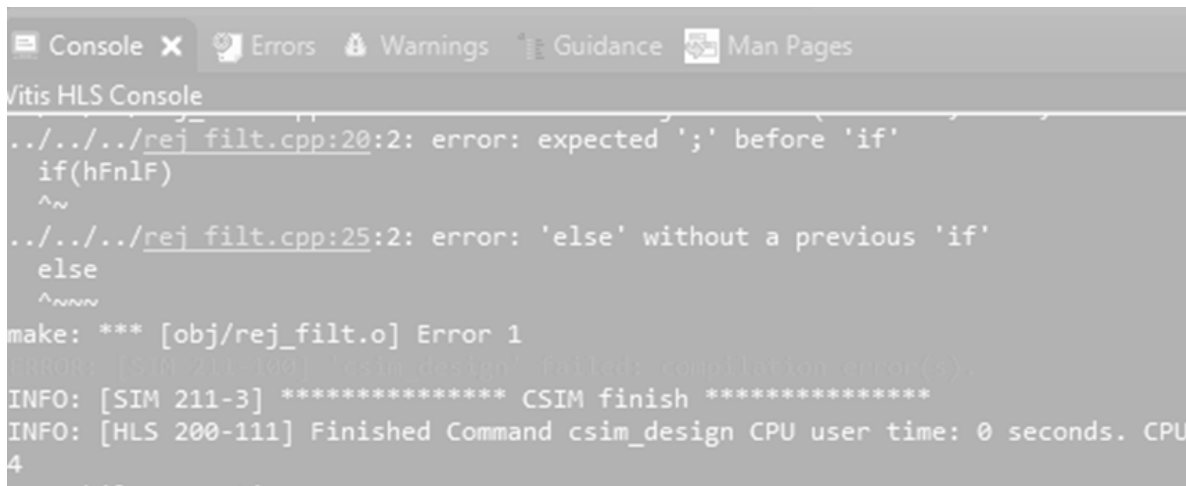
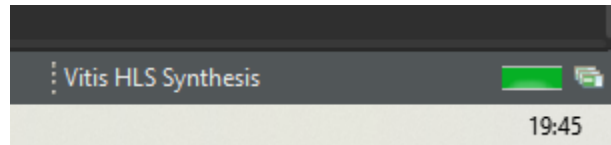


Рисунок 13.14 – Поиск ошибки в коде

Далее синтезируем RTL код, для этого нажмите в панели инструментов на зеленую стрелку, или выберите в меню **Solution->Run C Synthesis->C Synthesis**

Если синтез прошел успешно, то будет открыт файл отчета, если нет, то не будет открыт.

Для проверки того, что синтез еще идет, проверьте нижний правый угол окна:



В случае если синтез не прошел, перейдите на вкладку Console и изучите ее, в поисках ошибок. Сообщение об окончании синтеза приведено на рис. 13.14.

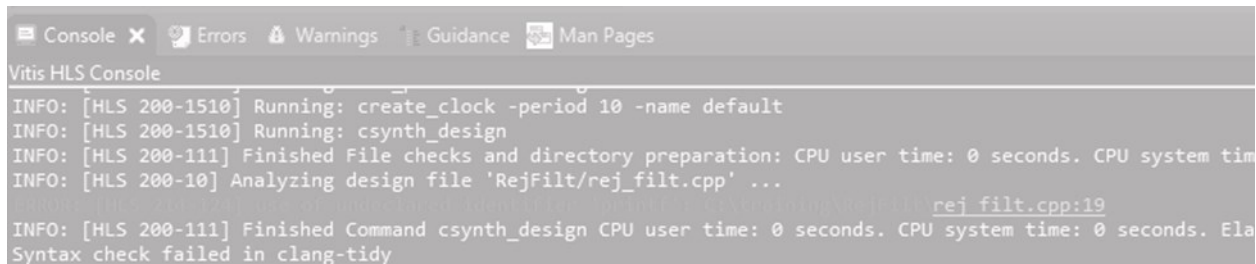


Рисунок 13.14 – Сообщение об окончании синтеза

Из окна отчета о результатах синтеза можно получить информацию о минимальном расчётном периоде, на котором будет работать синтезированная схема, задержках получения результата вычислений (параметр *Latency* в количестве тактовых сигналов и время), интервал считывания новых данных (параметр *Interval* в количестве тактов), ресурсов необходимых для реализации, количестве и типах интерфейсов и т.д. Отчет о синтезе проекта находится в дереве проекта в папке **solution 1->syn->report**. (см. рис. 13.15). Чтобы его открыть, кликните два раза на любом файле отчетов.

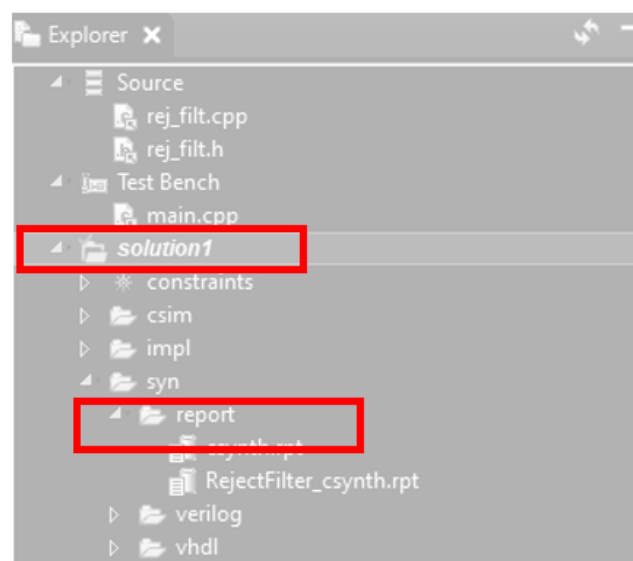
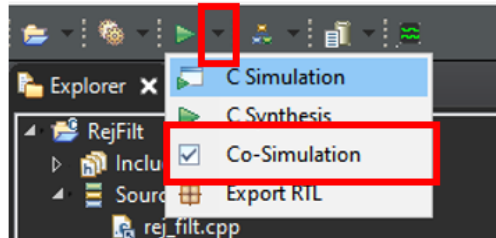


Рисунок 13.15 – Место файла отчета о синтезе в проводнике проекта

Синтезированный RTL код, следует просмотреть, открыв в папке **syn** подпапку **verilog** или **vhdl**.

Запустите Co-Simulation. Для этого выполните одно из двух:

- Выберите в меню **Solution-> Run C/RTL Cosimulation**
- На панели инструментов, нажмите на черную стрелку рядом с зеленой и выберите Co-Simulation.



Для того, чтобы открыть и изучить тактовую диаграмму работы модуля в окне конфигурации симуляции (рис.13.16), в списке **Dump trace** выберите **port**.

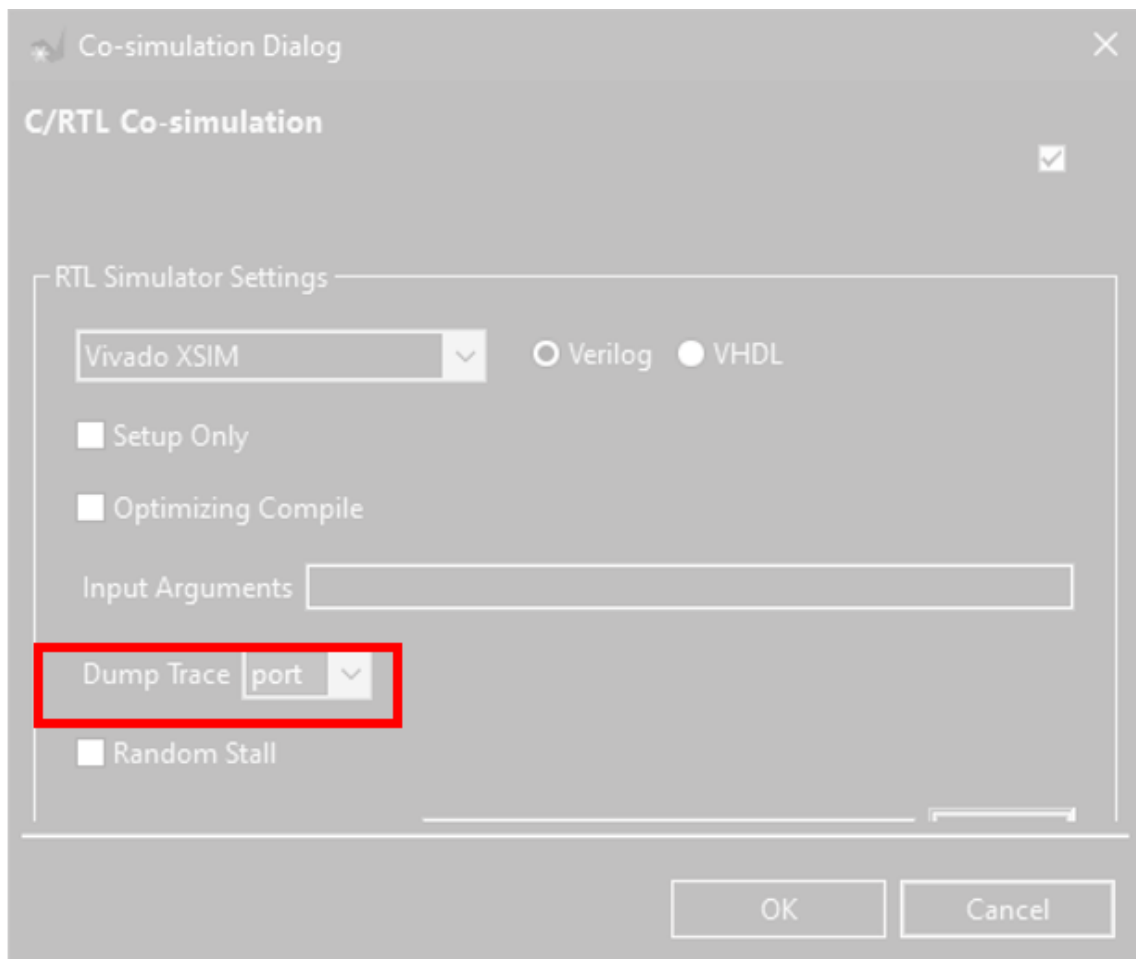


Рисунок 13.16 - Окно конфигурации симуляции

Запустите процесс, нажатием на кнопку **ОК**. Ход выполнения симуляции можно контролировать через вкладку Console.

Если симуляция пройдет успешно, то по завершению откроется окно отчета (рис. 13.17).

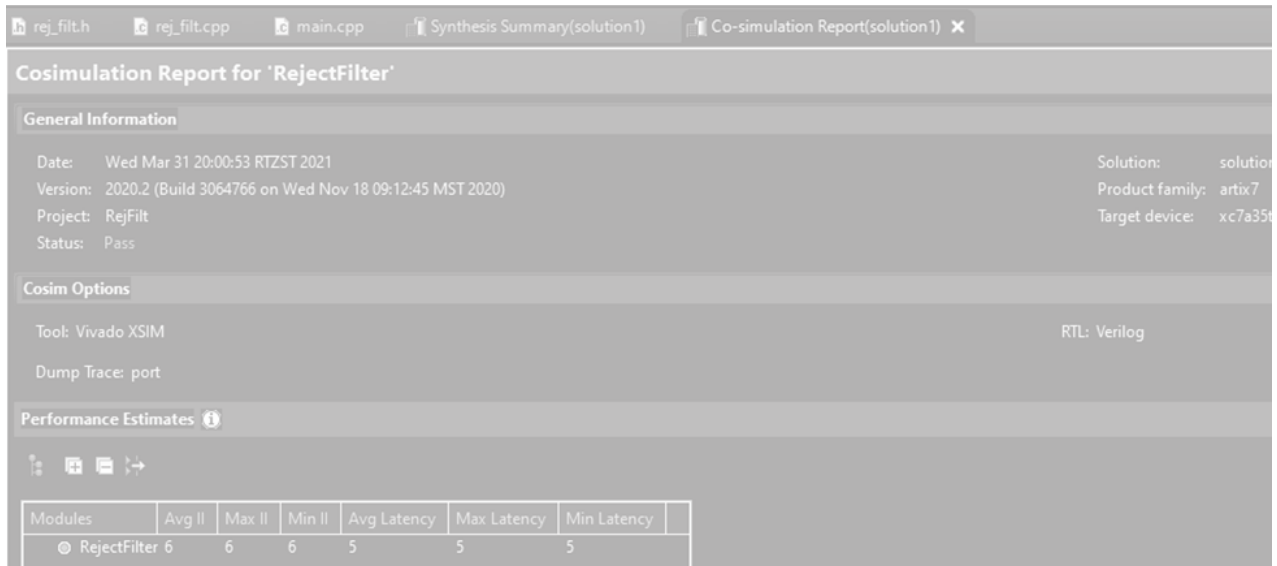


Рисунок 13.17 - Окно отчета о результатах симуляции

Обратите внимание на поле **Status**. В зависимости от значения, полученного от функции main тут будет указано **Pass** или **Fail**. Так как в нашем случае функция main всегда возвращает 0, то мы в любом случае получим значение Pass.

Для открытия отчета, перейдите в дереве проекта в папку **solution 1-> sim -> report**.

Если при конфигурировании проекта была отмечена опцию сбора данных, то кнопка просмотра тактовой диаграммы должна стать активной:



Нажмите на нее, и спустя некоторое время откроется окно Vivado с результатом симуляции работы модуля (рис. 13.18).

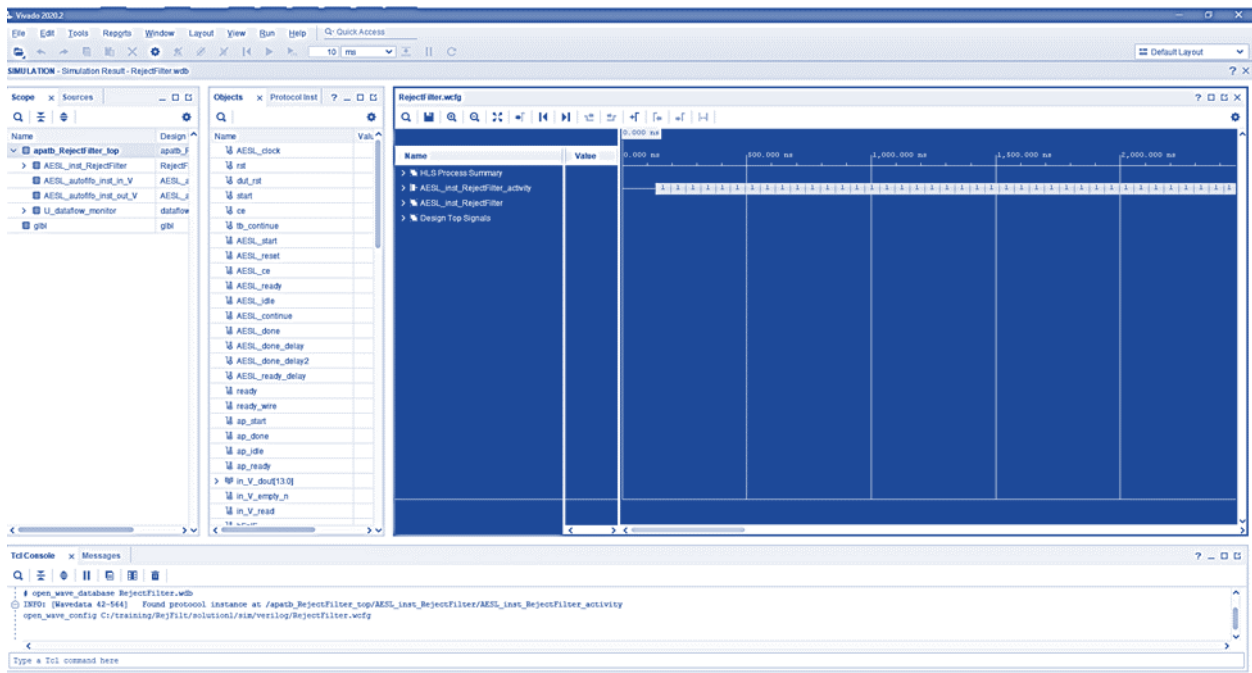


Рисунок 13.18 - Окно Vivado с результатом симуляции работы модуля

Выполните следующие действия:

1. Раскройте сигналы, как показано на рис. 13.19:

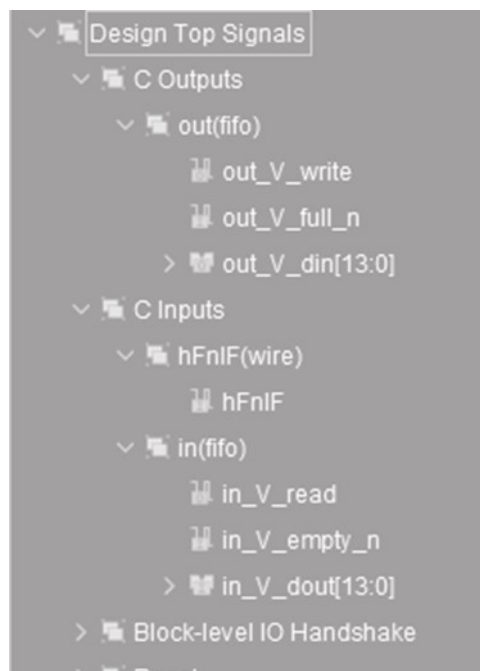


Рисунок 13.19 – Окно назначения сигналов для просмотра

2. Вызовете контекстное меню на шине `out_V_din[13:0]` и выберите Radix->Signed Decimal

3. Вызовите контекстное меню на шине out_V_din[13:0] и выберите Waveform Style->Analog
4. Вызовите контекстное меню на шине in_V_dout[13:0] и выберите Radix->Signed Decimal
5. Вызовите контекстное меню на шине in_V_dout[13:0] и выберите Waveform Style->Analog
6. Измените масштаб, чтобы на экране поместилась вся тактовая диаграмма как это показано на рис. 13.20.

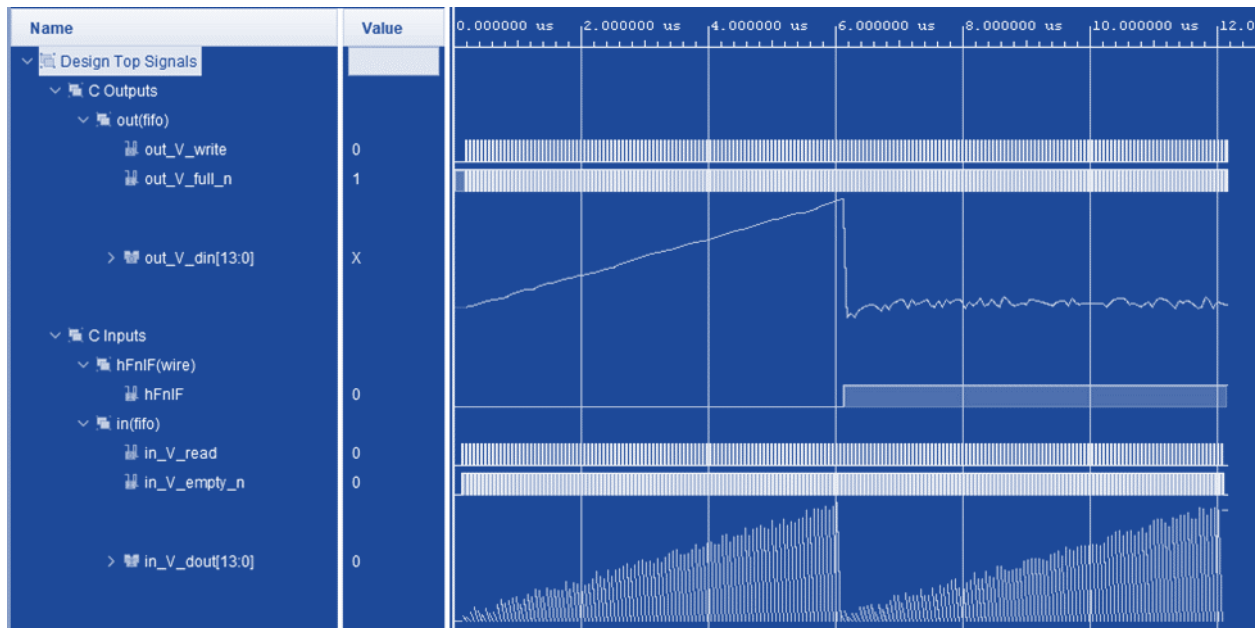


Рисунок 13.20 - Тактовая диаграмма с результатами моделирования

Сигнал out_V_din демонстрирует нам результат работы фильтра в режиме фильтрации низких частот (первая половина тактовой диаграммы) и в режиме работы фильтра высоких частот.

Обратите внимание на in_V_dout. Он имеет специфичный вид, так как во входное FIFO записывается каждый отсчет отдельно, из-за чего появляются нулевые значения, и итоговый аналоговый сигнал имеет пилообразную форму. Связано это с тем, что наш модуль обрабатывает только один отсчет из FIFO за один запуск. Исправим этот недочет.

Закройте окно симуляции.

13.4 ОПТИМИЗАЦИЯ КОДА

Тип **stream** который используется для входных и выходных данных может считывать/записывать данные в блокирующем режиме и не блокирующем. Блокирующий режим предполагает, что система остановиться, на этапе

считывание, если данные не готовы. Например, если модуль запущен на выполнении, а входное FIFO пустое, то наш модуль остановится на этапе получения отсчета, до тех пор, пока в FIFO не появятся данные. В неблокирующем режиме есть возможность выбрать модель поведения в случае, если на момент запроса данные есть, и даже если их нет.

В данном случае используется блокирующий режим как для считывания входных данных, так и для записи результатов работы модуля. Перепишите реализацию функции **RejectFilter**:

```
#include "rej_filt.h"

void RejectFilter (tStream *in, bool hFn1F, tStream *out)
{
    const fxItm cA = A;
    const fxItm cB0_hf = ((A - 1.0)/2.0);
    const fxItm cB1_hf = -cB0_hf;
    const fxItm cB0_lf = ((1.0 + A)/2.0);
    const fxItm cB1_lf = cB0_lf;

    static fxItm y0 = 0.0;
    static fxItm x0 = 0.0;

    fxItm tmp, tmp1, B0, B1;
    iItm curIn;

    if(hFn1F)
    {
        B0 = cB0_hf;
        B1 = cB1_hf;
    }
    else
    {
        B0 = cB0_lf;
        B1 = cB1_lf;
    }

    // Read new data and convert from integer to fixed point
    while(in->read_nb(curIn))
    {
        tmp = curIn;

        // Do filtration
        tmp1 = -cA*y0 + B1*tmp + B0*x0;

        // Save data for use in next step
        x0=tmp;
        y0=tmp1;

        // Write result out as integer
        out->write_nb(tmp1(BIT_WIDTH*2-1, BIT_WIDTH));
    }
}
```

}

Внесите изменения в функцию main():

```
int main()
{
    double fTmp;
    // Random generator
    std::uniform_real_distribution<double> unif(RND_GEN_MIN, RND_GEN_MAX);
    std::default_random_engine re;

    tStream in, out;
    iItm tmp;

    // Init random generator
    srand(static_cast<int>(time(0)));

    printf("Test low-frequency filter\n");
    for(int i=0; i<100; i++)
    {
        fTmp = unif(re);
        fTmp += i;
        in.write(fTmp);
    }

    RejectFilter(&in, false, &out);

    for(int i=0; i<100; i++)
    {
        fTmp = out.read();
        printf(" Output %lf\n", fTmp);
    }

    printf("Test high-frequency filter\n");
    for(int i=0; i<100; i++)
    {
        fTmp = unif(re);
        fTmp += i;
        in.write(fTmp);
        printf("Input %lf, ", fTmp);
        RejectFilter(&in, true, &out);
        fTmp = out.read();
        printf(" Output %lf\n", fTmp);
    }

    return 0;
}
```

Запустите синтез, а затем косимуляцию. Откройте тактовую диаграмму, и приведите ее к удобному виду, как это делали выше (см. рис. 13.21).

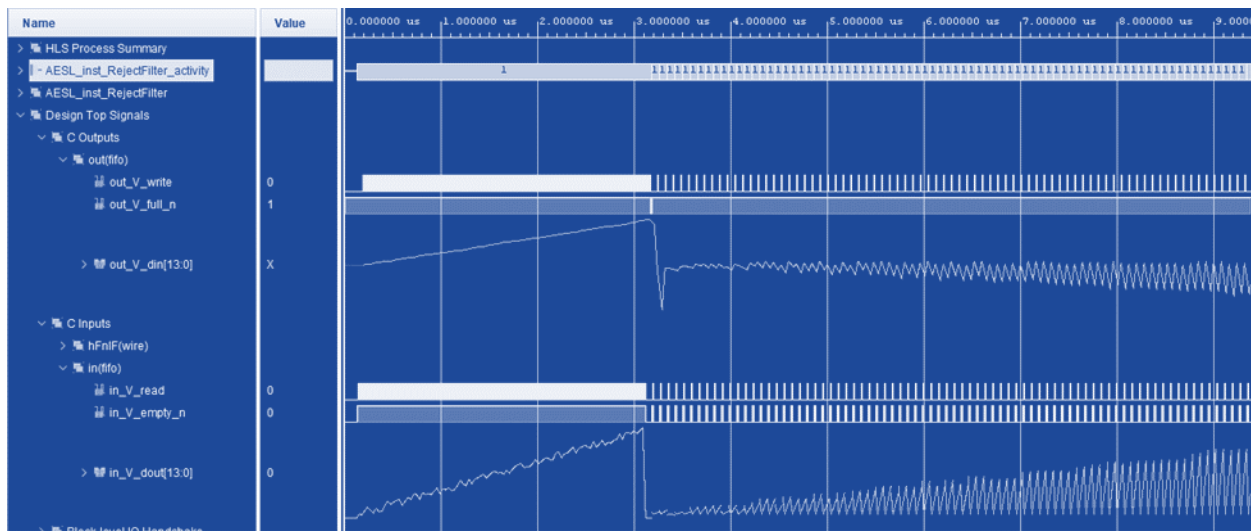


Рисунок 13.21 – Результаты моделирования после редактирования кода

Как видно, теперь модуль стал работать быстрее, однако на то, как происходит взаимодействие с модулем, влияет не только код самого модуля, но и код, реализующий проверку работы этого кода. Пилообразные выбросы в выходном сигнале блока связаны с тем, что теперь корректное значение между выдачей данных не сохраняются, в этом Вы можете убедиться, переключив обратно на цифровой вид, и изучив тактовую диаграмму канала **out(fifo)**.

Закройте симуляцию.

13.4 ДИРЕКТИВЫ HLS

Запустите синтез, и посмотрите внимательно на отчет.

Modules && Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
RejectFilter	II Violation	-	-	-	-	-	-	no	0	3	667	496	0
VITIS_LOOP_29_1	II Violation	-	-	-	0	3	-	yes	-	-	-	-	-

Обратите внимание, что после внесенных изменений, стало возможным получить информацию о задержке и интервале инициализации для одной итерации цикла, но для всей функции стоит знак “-“. Это связано с тем, что из кода явным образом нельзя узнать сколько итераций будет выполнено. Для того что бы решить эту проблему, воспользуемся директивой.

Директивы HLS - это специальные параметры, влияющие на архитектуру выходного модуля, количество ресурсов необходимых для его реализации, быстродействие и т.д. Директивы могут быть добавлены в исходный код, используя ключевое слово `#pragma` или в tcl файл выбранного Solution'a. В

последнем случае мы сможем воспользоваться разным набором директив создав несколько Solution`ов и сравнить результаты (что мы сделаем чуть позже).

Добавьте директиву LOOP_TRIPCOUNT, которая позволяет указать в ситуациях, когда количество итераций явно не определено, прогнозируемое количество итераций, что позволит инструменту рассчитать временные характеристики.

Для того, чтобы добавить директиву, откройте файл rej_filt.cpp и откройте вкладку Directive (рис. 13.22).

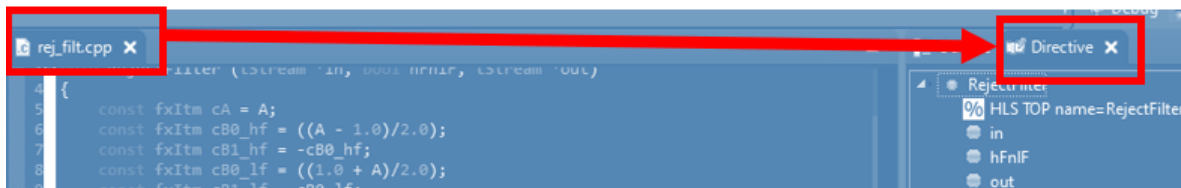


Рисунок 13.22 – Открытие вкладки Directive

Затем найдите в списке запись **whileStatement**, вызовите левой клавишей её контекстное меню и выберите Insert Directive:



Откроется окно управления директивами (рис. 13.23).

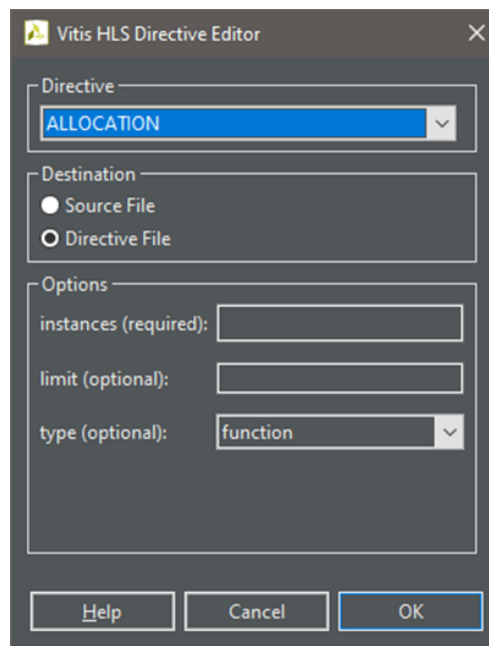


Рисунок 13.23 - Окно управления директивами

Доступный набор директив Вы можете увидеть, раскрыв список **Directive**. Этот набор будет различным, в зависимости от того, для какого элемента кода вызвали это окно. Раскройте список, и выберите директиву **LOOP_TRIPCOUNT**.

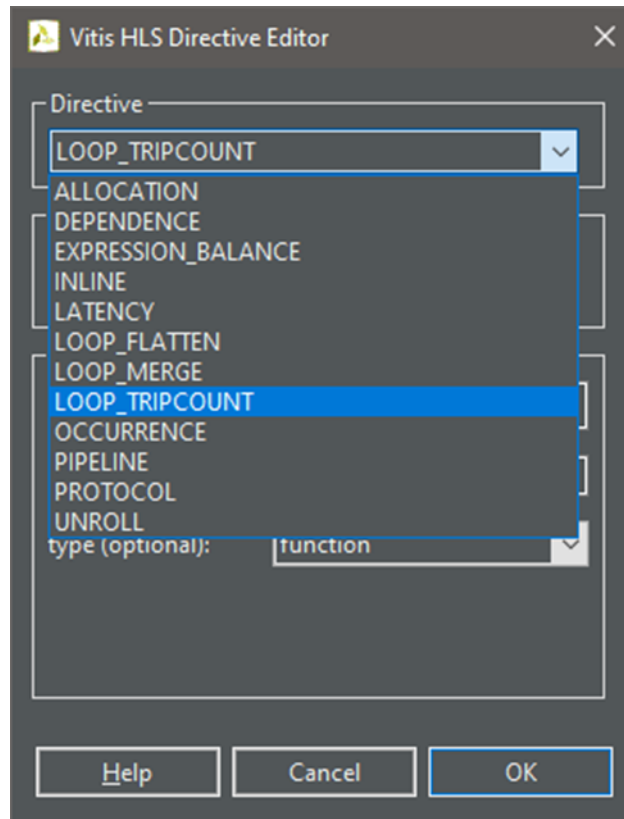


Рисунок 13.24 - Выбор директивы **LOOP_TRIPCOUNT**

Далее следует выбрать, где сохранить запись об этой директиве, в **tcl** файле текущего **Solution** или в исходном коде. Выберите сохранение в исходном коде (Source File):



Затем установите расчётные значения количества циклов.

- Avg (среднее количество) 100
- Max (максимальное количество) 256
- Min (минимальное количество) 1

Нажмите ОК, и посмотрите на первую строчку в теле цикла **while()**. Там появилась запись директивы, с установленными опциями. Вы может изменить эти опции, переписав данную строчку. Сохраните файл и вновь запустите синтез.

Теперь для всего модуля появились расчётные значения времени выполнения и инициализации. Можно узнать подробнее эти значения, если открыть отчет в дереве проекта: **solution 1 -> syn -> report -> RejectFilter_csynth.rpt.**

13.5 ЗАДАНИЕ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ

Создайте проект для реализации рекурсивного фильтра второго порядка (ФНЧ и ФВЧ):

$$y(n) = -a_1y(n-1) + a_0y(n-2) + b_2x(n) + b_1x(n-1) + b_0x(n-2)$$

Для формирования ФНЧ (фильтр нижних частот): $b_1 = b_0 + b_2 = (1 + a_1 + a_0)/2$

Для формирования ФВЧ (фильтр высоких частот): $-b_1 = b_0 + b_2 = (1 - a_1 + a_0)/2$

При $a_0 = 0.02$, $a_1 = 0.55$

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Майк МакГрат Программирование на С для начинающих / Майк МакГрат. [Пер. с англ. М. Райтмана] . – М.: Эксмо, 2016. – 192 с.
2. Васильев А. Самоучитель С++ с примерами и задачами. 4-е издание (переработанное). Книга + виртуальный CD. — СПб.: Наука и Техника, 2016. — 480 с.: ил. (+ виртуальный CD).
3. Полный справочник по С++. // 4-е издание. Герберт Шилдт [Электронный ресурс] URL: https://codernet.ru/books/c_plus/polnyj_spravochnik_po_c_4-e_izdanie/ Дата обращения 02.12.2021.
4. 44 ресурса для изучения С и С++: от новичка до профессионала [Электронный ресурс] URL: <https://proglib.io/p/44-resursa-dlya-izucheniya-c-i-c-ot-novichka-do-professional-a-2021-08-28/>. Дата обращения 02.12.2021.
5. А.Ю. Осташенко, Е.Ф. Певцов ИНФОРМАТИКА. ПРАКТИКУМ ПО ПРОГРАММИРОВАНИЮ «Основы программирования на VISUAL С++ 6.0» Методические указания по выполнению курсовой работы «Разработка программы исследования функций» МИРЭА, 2009 [Электронный ресурс] URL: <http://www.edamc.mirea.ru/files/kursovicCpp.pdf>. Дата обращения 02.12.2021.
6. Магницкий Б.В., Певцов Е.Ф. Компьютерное моделирование физических процессов с использованием С++ // Методические указания по выполнению лабораторных работ. Федеральное государственное бюджетное учреждение высшего профессионального образования «Московский государственный университет радиотехники, электроники и автоматики». М.: МГТУ МИРЭА 2012, 36 с. [Электронный ресурс] URL: http://www.edamc.mirea.ru/files/Curs_C++_2012.pdf. Дата обращения 02.12.2021.