МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Физические методы контроля»

ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Методические рекомендации к лабораторным работам для студентов специальности 1-54 01 02 «Методы и приборы контроля качества и диагностики состояния объектов» очной формы обучения

Могилев 2023

УДК 004.43
ББК 32.973-018.1
Я41

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Физические методы контроля» «1» сентября 2023 г., протокол № 1

Составитель канд. техн. наук, доц. А. В. Кушнер

Рецензент канд. техн. наук, доц. В. М. Ковальчук

Методические рекомендации предназначены для выполнения лабораторных работ по курсу «Языки программирования» студентами специальности 1-54 01 02 «Методы и приборы контроля качества и диагностики состояния объектов» очной формы обучения.

Учебное издание

ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Ответственный за выпуск	А. В. Хомченко
Корректор	А. А. Подошевко
Компьютерная верстка	М. М. Дударева

Подписано в печать 29.11.2023. Формат 60×84/16. Бумага офсетная. Гарнитура Таймс. Печать трафаретная. Усл. печ. л. 2,09. Уч.-изд. л. 2,06 . Тираж 16 экз. Заказ № 1274.

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

Содержание

1 Изучение среды программирования MASM

Цель работы: ознакомиться с созданием ассемблерных программ в среде MASM.

1.1 Основные теоретические положения

Основная концепция пакета MASM 6.1x – совместить удобства программирования, свойственные языкам высокого уровня, с традиционными достоинствами машинно-ориентированных языков.

В состав пакета ассемблера фирмы Microsoft входят следующие программы:

masm.exe – ассемблер;

ml.exe (Masm and Link) – ассемблер и компоновщик;

link.exe – компоновщик;

pwb.exe (Programmer's WorkBench) – интегрированная среда разработки ассемблерных программ, позволяющая редактировать, формировать, отлаживать и выполнять ассемблерную программу. Среда обладает макроязыком, с помощью которого существенно повышается гибкость управления ее работой;

cv.exe (CodeView) – отладчик ассемблерных программ, предназначенный для обнаружения ошибок в ассемблерных программах. Существует два варианта отладчика – один для MS DOS, другой для Microsoft Windows; вспомогательные утилиты lib.exe, nmake.exe и др.

Пакет MASM позволяет производить разработку программ двумя способами:

1) традиционным для ассемблера способом – запуском отдельных программ трансляции, компоновки и отладки;

2) с использованием интегрированной среды, запускаемой программой pwb.exe.

При традиционном способе разработки программы используются следующие средства пакета MASM: masm.exe, ml.exe, link.exe и cv.exe.

Программа **masm.exe** Командная строка masm.exe имеет вид:

Masm[ключи]исх_файл[.[объектный_файл][.[файл_листинга][.[файл_перекрестных_ссылок]]]]

Ключи (некоторые из них).

/A – сегменты должны быть упорядочены в алфавитном порядке.

/H – вывести справку по ключам командной строки.

/HELP – вызвать QuickHelp (qh.exe)для справки по MASM.

/L – создать обычный листинг.

/LA – в листинг помещается вся возможная информация.

/ZI – включение в объектный файл информации о идентификаторах для отладчика CodeView.

Программа **ml.exe.**

Транслирует и связывает один или более исходных ассемблерных файлов.

Командная строка ml.exe имеет следующий вид:

ml [ключи]исх_файл_1[[ключи]исх_файл_2]…[/link ключи_link].

Ключи командной строки чувствительны к регистру (ниже приведены некоторые из них).

/c – только трансляция исходного файла.

/Fe имя_файла – задать имя исполняемого файла.

/Fl имя_файла – генерировать листинг.

/Fm имя_файла – создать файл карты компоновщика.

/Fo имя_файла – имя объектного файла.

/Sa – включить в листинг всю доступную информацию.

/Sg – включить в листинг текст сгенерированного кода.

/Si – включение в объектный файл информации для отладчика CodeView.

/? – справка о синтаксисе командной строки ml.

Программа **link.exe.**

Компоновщик link компонует (объединяет) объектные файлы и библиотеки в исполняемый файл или динамически компонуемую библиотеку (DLL). Командная строка link.exe имеет вид:

Link[ключи]объект_файлы[.[исполн_файл][.[файл_карты][файлы_библиотек][.[def_файл]]]]][;].

Некоторые ключи приведены ниже.

/T – создать исполняемый файл для MS DOS формата.com.

/? – вывод информации о синтаксисе link.exe.

Программа **cv.exe.**

Отладчик Microsoft CodeView Debugger – предназначен для выполнения исполняемого файла программы при одновременном отображении ее исходного текста, состояния переменных, регистров процессора, занимаемой памяти и другой сопутствующей информации.

Существует две версии отладчика CodeView – для MS DOS (cv.exe) и Windows (cvw.exe).

Отладчик cv.exe запускается следующей командной строкой:

Cv[ключи]исполн_файл[параментры].

Некоторые значения ключей приведены ниже.

/2 – разрешает использование двух мониторов.

/25 – запуск отладчика с 25 строками.

/43 – запуск отладчика в режиме с 43 строками.

/50 – запуск отладчика в режиме с 50 строками.

/M – запретить CodeView использовать мышь.

Программа **lib.exe.**

Программа lib.exe является библиотечным менеджером, который создает и обслуживает стандартные библиотеки объектных модулей. С его помощью программист может выполнить все действия с файлом библиотеки: создание, добавление, удаление и замену модулей.

Программа lib.exe запускается следующей командной строкой:

Lib библиотека[ключи][команды][.[файл_листинга][.[вых_библиотека]]][;]

Значения команд приведены ниже.

+name – добавить объектный код или библиотечный файл в конец библиотечного файла.

-name – удалить модуль.

-+name – заменить модуль, удаляя его и добавляя в конец с тем же самым именем.

*name – копировать модуль в новый объектный файл.

-*name – переместить модуль (удаляя его) из библиотеки в новый объектный файл.

Работа с данными, находящимися в оперативной памяти, это целиком забота программиста. Для определения адреса данных необходимо хорошо знать способы адресации оперативной памяти. Адрес оперативной памяти состоит из двух частей: адреса сегмента и смещения относительно начала сегмента. Адрес начала сегмента находится в регистре сегмента, значение смещения в одном из регистров, как правило, это регистры BX, DI, SI, SP, BP.

1.2 Порядок выполнения работы

1 Создать в текстовом редакторе файл с названием example.asm.

2 Набрать в этом файле следующую программу:

```
.MODEL  SMALL                    ; Модель памяти-SMALL(Малая)
.STACK  100h                ; Отвести под Стек 256 байт
.DATA                            ; Начало сегмента данных
Grt  DB 'Вас приветствует ст. - Иванов И.И.',13,10,'$'
                                 ; Текст приветствия

.CODE                ; Начало сегмента кода
 Main:   mov  ax,@data           ; Загрузка в DS адреса начала
mov  ds,ax           ; сегмента данных
mov  dx,OFFSET Grt               ; Загрузка в dx смещения
                     ; адреса текста приветствия
DisplayGreeting:
mov  ah,9            ; # функции ДОС печати строки
int  21h             ; вывод на экран  приветствия
mov  ah,4ch          ; # функции ДОС завершения программы
int  21h             ; завершение программы и выход в ДОС
END main
```

3 Протранслировать программу с помощью команды

> masm имя_файла.asm

с созданием объектного файла и файла диагностических сообщений (файла листинга). Объяснить и исправить синтаксические ошибки, если они будут обнаружены транслятором.

Повторить трансляцию программы до получения объектного модуля.

4 Скомпоновать загрузочный модуль с помощью строки

> link имя_файла.obj

5 Выполнить программу в автоматическом режиме путем набора строки

> имя_файла.exe

и убедиться в ее работоспособности.

6 Создать собственную программу на ассемблере по заданию преподавателя.

Содержание отчета

1 Цель работы.
2 Постановка задачи и исходные данные.
3 Схема алгоритма и программа.

Контрольные вопросы

1 Что такое адрес оперативной памяти?
2 Из каких составных частей состоит адрес?
3 Какие методы адресации предоставляет ассемблер?
4 Для чего используются флаги и какими средствами они проверяются?
5 Как организуется цикл?
6 Как извлечь элемент массива?
7 Что такое условный переход?
8 Каково назначение индексного регистра?

2 Разработка программ на языке Ассемблер

Цель работы: изучить способы адресации изучить операции ввода/вывода.

2.1 Основные теоретические положения

Способы адресации операндов.

В программах на **Assembler** применяются следующие типы адресации операндов: регистровая, прямая, непосредственная, косвенная, базовая, индексная, базово-индексная.

Регистровая адресация подразумевает использование в качестве операнда регистра, например:

PUSH DS
MOV BP,SP

При ***прямой*** адресации один операнд представляет собой адрес памяти,

второй – регистр:

MOV DATA, AX

Непосредственная адресация применяется, когда операнд длиной в байт или слово находится в ассемблерной команде:

MOV AX, 4Ch

При использовании ***косвенной*** адресации исполнительный адрес формируется исходя из сегментного адреса в одном из сегментных регистров и смещения в регистрах BX, BP, SI или DI, например:

MOV AL, [BX]; база находится в регистре DS, смещение в регистре BX

MOV AH, [SI]; база − в DS, смещение – в SI

MOV AX,[DI]; база в DS, смещение – в DI

MOV AX, ES: [DI]; база − в ES, смещение – в DI

MOV DX, [BP]; база − в SS, смещение – в BP

В случае применения ***базовой*** адресации исполнительный адрес является суммой значения смещения и содержимого регистра **BP** или **BX**, например:

MOV AX, [BP+6]; база – SS, смещение − содержимое BP, которое складывается

MOV [BX+Delta], AX; база – DS, смещение – содержимое BX+смещение Delta

MOV AX, [BP]+4; база – SS, смещение – содержимое ***BP+4***

MOV DX, 8[BX]; база − DS, смещение – содержимое BX+8

При ***индексной*** адресации исполнительный адрес определяется как сумма значений указанного смещения и содержимого регистра SI или DI так же, как и при базовой адресации, например:

MOV DX, [SI+5]; база – DS, смещение – SI+5

MOV ES:[DI]+6,AL ; база – ES, смещение – DI+6

Базово-индексная адресация подразумевает использование для вычисления исполнительного адреса суммы содержимого базового и индексного регистров, а также смещения, находящегося в операторе, например:

MOV BX, [BP][SI]

MOV ES:[BX+DI],AX

MOV Array[BX][SI],12h

MOV AX,[BP+6+DI]

MOV Array [BP+BX]; ошибка − два базовых регистра

MOV Array [DI+SI]; ошибка − два индексных регистра

Флаги.

Девять флагов размещены в регистре флагов. Флаги имеют следующие значения.

1 Бит 0, флаг переноса **CF (carry flag)**. Изменяется во многих операциях.

2 Бит 2, флаг четности **PF (parity flag)**.

3 Бит 4, вспомогательный флаг переноса **AF (auxiliary carry flag**).

4 Бит 6, флаг нуля **ZF (zero flag)**, равен 1, если результат операции равен 0.

5 Бит 7, флаг знака **SF (sign flag)**. SF равен 1, если результат отрицательный.

6 Бит 8, флаг трассировки **TF (trap flag)**,

7 Бит 9, флаг прерывания **IF (interrupt enable flag)**, разрешает прерывания от внешних устройств. Если IF=0 прерывания запрещены.

8 Бит 10, флаг направления **DF (direction flag)**.

9 Бит 11, флаг переполнения **OF (overflow flag)**.

Обработка массивов.

Обработка массивов обычно выполняется в цикле. Кроме того, в программе часто необходимо выполнять многократно некоторые фрагменты программы. Для этого в Ассемблере имеется удобное решение организации циклов. Количество повторений цикла заносится в регистр **CX,** а команда **LOOP** обеспечивает выполнение заданного количества повторений. Как правило, команда **LOOP** ставится в конце повторяющейся последовательности команд программы.

Loop-метка. Команда вычитает единицу из содержимого регистра **CX** и, если содержимое неравно нулю, передает управление на метку. Если регистр **CX** становится равен нулю, выполняется следующая за **LOOP** команда. В регистр **CX** заносится количество повторений цикла.

Например, сложить десять чисел, которые находятся в таблице с именем **TAB**.

```
Mydata segment
 Tab DB 1, 2, 3, 4, 5, 6, 7, 8, 9, 10; значения элементов массива
Mydata ends
Mycode segment
     MOV CX, 10          ; количество повторений
     MOV SI,0            ; ноль в индексный регистр
     MOV BL,0            сумма будет в BL
BEGIN: ADD BL, TAB[SI]   ; добавляем элемент к сумме
     INC SI              ; переходим к следующему элементу
     LOOP BEGIN          ; возврат на начало цикла
. . .
Mycode ends
```

Команда сравнения.

CMP операнд1, операнд2. Команда вычитает операнд2 из операнда1 и устанавливает регистры флагов. Операнды не меняются. За ней обычно идет команда условного перехода.

Команды условного перехода.

В Ассемблере имеется группа команд условного перехода, которые проверяют состояние флагов в регистре флагов.

Команда проверки состояния флага переноса.

JC – перейти, если флаг CF установлен в единицу (например, при сдвиге бита из регистра).

JZ – перейти, если флаг ZF установлен в единицу (например, результат операции равен нулю).

JS – перейти, если флаг SF установлен в единицу (проверяет только старший бит байта или слова).

JP – перейти, если флаг PF установлен в единицу (в результате операции количество единиц четно).

JO – перейти, если флаг OF установлен в единицу (операция завершилась переполнением).

Команда безусловного перехода JMP.

JMP-метка. Передает управление на метку.

Команды логических операций.

Устанавливают флаги **PF, SF, ZF.**

Команда AND источник, приемник (логическое умножение).

Побитно выполняет операцию логического умножения над источником и приемником. Если оба обрабатываемых бита равны 1, результат 1, иначе 0. Как правило, операция применяется для обнуления заданных в операнде битов или для выделения части битов кода.

Например, необходимо убрать признак кода ASCII в введенной с клавиатуры цифре.

Тогда

MOV AH, 1 Введем с клавиатуры

INT 21H Цифру по прерыванию 21H в регистр AL

AND AL, 00001111B Удалим признак кода ASCII (т. е. старшие 4 бита).

Команда OR источник, приемник (логическое сложение).

Побитно складывает биты в операндах. Если хотя бы один из пары обрабатываемых битов равен 1, то результат 1, иначе 0. Обычно используется для установки заданных битов в 1.

Например, надо добавить признак кода ASCII в выводимую на экран цифру.

Тогда

MOV DL, 7 В регистре DL получаем двоичную семерку

OR DL, 00110000B и добавляем к ней признак ASCII (т. е. в старшие 4 бита).

Команда XOR источник, приемник (сложение по модулю 2).

Побитно складывает биты операндов (но переноса в старший бит нет). Если оба бита одинаковые, результат 0, иначе 1. Обычно используется для изменения заданных битов в обратное состояние.

Например, изменим значения старших 4 битов на обратные.

Тогда

MOV AL, 7BH В **AL** комбинация 01111011

XOR AL, 11110000B После операции в **AL** 10001011.

Команда TEST источник, приемник.

Аналогична команде **AND**, но не изменяет операнды, а только флаги **ZF,**

SF, PF. Идущая после этой команды команда условного перехода определяет каков был результат.

Команды сдвига.

Команда SHR dest, 1 или SHR dest, CL (сдвиг вправо). Последний сдвигаемый бит помещается во флаг **CF.**

Команда SHL dest, 1 или SHL dest, CL (сдвиг влево). Последний сдвигаемый бит помещается во флаг **CF.**

Команды сдвига обычно используется для анализа состояния байта или слова.

2.2 Порядок выполнения работы

1 Создать собственную программу на языке Ассемблер по заданию преподавателя.

2 Протранслировать программу с помощью команды

> masm имя_файла.asm

с созданием объектного файла и файла диагностических сообщений (файла листинга). Объяснить и исправить синтаксические ошибки, если они будут обнаружены транслятором.

Повторить трансляцию программы до получения объектного модуля.

3 Скомпоновать загрузочный модуль с помощью строки

> link имя_файла.obj

4 Выполнить программу в автоматическом режиме путем набора строки

> имя_файла.exe

и убедиться в ее работоспособности.

Содержание отчета

1 Цель работы.

2 Постановка задачи и исходные данные.

3 Схема алгоритма и программа.

4 Результат выполнения программы.

Отчет оформляется в соответствии с ГОСТ 2.105–95 *Общие требования к текстовым документам.*

Контрольные вопросы

1 Что такое адрес оперативной памяти?

2 Из каких составных частей состоит адрес?

3 Какие методы адресации предоставляет ассемблер?

4 Для чего используются флаги и какими средствами они проверяются?

5 Как организуется цикл?
6 Как извлечь элемент массива?
7 Что такое условный переход?
8 Каково назначение индексного регистра?

3 Изучение среды программирования Visual C++

Цель работы: изучить основные операторы и конструкции языка программирования *C++* для программирования разветвляющихся алгоритмов, а также основные операторы и конструкции языка программирования *C++* для программирования алгоритмов циклической структуры.

3.1 Основные теоретические положения

1 Разветвления в программе возникают при необходимости выбора одного из нескольких возможных путей в решении задачи, который может зависеть от значений исходных данных или промежуточных результатов. Для организации разветвлений в программах используются операторы перехода и условный оператор.

Условный оператор обеспечивает выполнение или невыполнение некоторого оператора, группы операторов или блока в зависимости от заданных условий.

Вид условного оператора:

1) if (выражение) инструкция;

2) if (выражение) инструкция; else инструкция.

Принцип работы: если выражение принимает целое ненулевое значение (истина), то выполняется инструкция, следующая за выражением, в противном случае выполняется инструкция, стоящая после else. Если выражение ложно (принимает значение 0) и else отсутствует, то выполняются инструкции, следующие сразу за оператором if.

Примеры записи операторов:

```
if (x<0.1) v=exp(x);
if (x<a) v=exp(x); else z=cos(x);
```

Если инструкции состоят из нескольких операций, то они заключаются в фигурные скобки:

```
if (x<0.1){x=exp(y); i++;} else {z=cos(x);i --}
```

2 В случае когда при выполнение какого либо условия (или невыполнения) необходимо выполнять группу операторов повторно, используются циклы. Существует три основных вида циклов: цикл while (с предусловием), цикл do-while (с послеусловием) и цикл for (с подготовкой).

Цикл с предусловием while используется для проверки некоторых условий в начале цикла.

Формат оператора цикла while:

while (выражения) инструкция;

В цикле while вычисляется выражение. Если оно имеет результат «истина» (не ноль), выполняется инструкция. В противном случае выполнение цикла завершается.

Цикл с послеусловием – do-while.

Формат цикла:

do

инструкция

while(выражение);

Операторы в цикле do-while выполняются хотя бы один раз, потому что проверка выражения осуществляется в конце тела цикла. Цикл do-while выполняется, пока выражение не станет ложно (нулевое). Если инструкции состоят из нескольких операций, то они заключаются в фигурные скобки.

Пример 1

```
s=0;
x=1;
do{
s+=x;
x+=0.1;}
while(x<5);
```

Цикл с подготовкой – for.

Формат для цикла for:

for (выражение 1; выражение 2; выражение 3) инструкция;

где выражение 1 – задание начального значения параметра цикла;

выражение 2 – условие продолжения цикла;

выражение 3 – изменение параметра цикла.

Цикл for работает следующим образом: сначала выполняется выражение 1, затем выражение 2, если оно истинно, выполняется инструкция и выражение 3, затем снова проверяется выражение 2. Цикл прекращает работу, если выражение 2 становится ложным (нулевым);

Пример 2

```
s=0;
for(x=1;x<5;x+=0.1) s+=x;
```

3.2 Порядок выполнения работы

1 Составить схему алгоритма и написать в соответствии со схемой алгоритма программу для таблицы вычисления значений функции на алгоритмическом языке *C*.

Разработать программу по заданию преподавателя.

Пример 3 − Разработать программу с ветвлением для вычисления выражения вида:

$$D=\begin{cases} ln(a+b)^2,\ a+b<0; \\ ln(a+b),\ a+b>0; \\ 0,\ a+b=0. \end{cases}$$

```
#include<stdio.h>
#include<math.h>
void main(void) /*Программа с ветвлением*/
{
  float a, b, c,d;
      puts(“\n\tВведите исходные данные A и B”);
      scanf(“%f%f”,&a,&b);
      c=a+b;
      d=0;
      if (c<0)
         d=log(pow((a+b),2));
      else
         d=log(a+b);
      printf(“\tD=%f”,d);
}
```

2 Вычислить таблицу значений функций. Вывод на экран должен быть оформлен в виде таблицы значений. Данные выдает преподаватель.

Пример 4 − Разработать программу для вычисления таблицы значений функций y=x при x=-3, -2.5, …, 3.

Один из вариантов программы:

```
#include<stdio.h>
#include<math.h>
viod main(void)
{
  float x0=-3, x1=-2.5,xk=3, h, x, y;
  h=x1-x0;
  x=x0;
  puts(“\n_____”);
  puts(“I      X       I    Y    I”);
  puts(“_____”);
      do
      {
         y=x*x;
            printf(“I%3.2f\tI%3.2f\tI\n”,x,y);
            x+=h;
      }while(x<=xk);
      puts(“_____”);
}
```

Содержание отчета

1 Цель работы.
2 Постановка задачи и исходные данные.
3 Схема алгоритма и программа.
4 Результат выполнения программы.
Отчет оформляется в соответствии с ГОСТ 2.105–95 *Общие требования к текстовым документам.*

Контрольные вопросы

1 Что называется полным и неполным ветвлением?
2 Перечислить действия, реализуемые при выполнении условного оператора.
3 Как организовать разветвление вычислений на две ветви?
4 Какие типовые конструкции алгоритмов используются при решении циклических задач?
5 Какие операторы цикла используются в языке *C*?
6 Разница между циклами с предусловием и послеусловием.
7 Опишите формат цикла for.

4 Разработка программ на языке C++

Цель работы: изучить основные алгоритмы обработки массивов:
– алгоритм поиска максимального (минимального) значения элемента массива;
– алгоритмы вычисления суммы и произведения элементов массива;
– операции с матрицами (суммирование, скалярное произведение, транспонирование);
– алгоритм сортировки элементов массива.
Приобрести практические навыки программной реализации основных алгоритмов обработки одномерных и многомерных массивов.

4.1 Основные теоретические положения

Массив – это структурированный тип данных языка программирования *C*. Он представляет упорядоченное множество элементов одинакового типа. Каждый элемент массива имеет свой номер (индекс), при помощи которого обеспечивается доступ к его значению.
Объявление массива производится в следующей форме:
Базовый_тип имя массива [количество элементов];
Массивы в общем случае бывают двух типов:
1) одномерные;
2) многомерные.

Многомерные массивы представляются как массивы, каждым элементом которых является массив.

Пример объявление массивов:

char m1[10]; (одномерный массив из 10 элементов);

int m2[10][10]; (двухмерный массив).

Доступ к значениям переменных элементов массива производится с помощью идентификатора переменной массива, имеющего в конце указание индекса элемента в квадратных скобках (m[2], m[0][9]). Индекс первого элемента массива 0.

4.2 Порядок выполнения работы

Разработать алгоритм и программу обработки матрицы. Вариант задается преподавателем. Печать значений исходных данных, а также промежуточных и окончательных результатов счета, должна сопровождаться пояснительным текстом, например «Исходная матрица A(3x4):», «Результирующий вектор B(5):» и т. п. Элементы массивов при выводе располагать в виде, удобном для восприятия.

Содержание отчета

1 Цель работы.

2 Постановка задачи и исходные данные.

3 Схема алгоритма и программа.

4 Результат выполнения программы.

Отчет оформляется в соответствии с ГОСТ 2.105–95 *Общие требования к текстовым документам.*

Контрольные вопросы

1 Как определяются массивы в языке *C*?

2 Назовите основные алгоритмы обработки массивов.

3 Как организовать ввод/вывод значений элементов массива?

5 Изучение среды программирования Turbo Prolog

Цель работы: приобрести практические навыки работы в среде Турбо-Пролога.

5.1 Основные теоретические положения

После запуска системы Турбо-Пролог на экране дисплея появляются четыре окна:

1) окно редактирования **Editor** – для ввода и редактирования исходной программы;

2) окно диалога **Dialog** – для ввода запросов и выдачи результатов;

3) окно сообщений **Message** – для выдачи сообщений;

4) окно трассировки **Trace** – для выдачи информации о выполнении программы.

Главное меню содержит шесть команд:

1) **Files** – команды управления файлами;

2) **Edit** – редактирование программы;

3) **Run** – запуск программы;

4) **Compile** – компиляция программы;

5) **Options** – изменение параметров компиляции;

6) **Setup** – изменение системных параметров.

При помощи клавиши ESC можно выйти из любого меню или подменю и вернуться в главное меню.

Команда **Files** содержит девять подкоманд обработки файлов:

1) **Load** − загружает файл для обработки;

2) **Pick** − позволяет загрузить один из семи последних файлов, с которыми работал пользователь;

3) **New file** − открывает пустое окно для создания нового файла;

4) **Save** − запись на диск рабочего файла;

5) **Write to** − сохраняет файл с именем, заданным пользователем;

6) **Directory** − позволяет перейти к файлам другого каталога, не изменяя текущего каталога;

7) **Change dir** − изменяет текущий каталог;

8) **OS shell** − временный выход в MS-DOS, возврат в систему Турбо-Пролога по команде exit;

9) **Quit** − выход из среды Турбо-Пролога.

Команды Edit и Run не имеют подкоманд.

В команде Compile указываются пять подпунктов, которые определяют результаты последующей компиляции:

1) **Memory** (устанавливается по умолчанию) − скомпилированная программа помещается в оперативную память;

2) **OBJ file** − скомпилированная программа имеет формат объектного файла, который должен компоноваться совместно с другими объектными файлами;

3) **EXE file** − скомпилированная программа имеет формат автономного загрузочного файла, который может запускаться вне системы Турбо-Пролог;

4) **Project** – используется при модульном программировании;

5) **Link only** – используется при модульном программировании.

Команда **Options** позволяет задавать опции компоновки (**Link options**), редактировать файл проекта (**Edit PRJ file**) при модульном программировании, устанавливать директивы компилятора (**Compiler directives**).

Команда **Setup** позволяет изменять следующие системные параметры:

Colors – изменение цветов переднего плана и фона выбранного окна;

Windows size – изменение размера и расположения выбранного окна;

Directories – задание имен директорий, к которым Турбо-Пролог будет обращаться по умолчанию;

Miscellaneous – настройка расширенного графического адаптера, установка режимов экрана, конфигурации клавиатуры, содержания строк подсказки;

Load SYS file – загрузка файла типа sys (по умолчанию Prolog.sys) со значениями параметров конфигурации;

Save SYS file – запоминание текущих значений параметров конфигурации в указанном файле типа sys (по умолчанию Prolog.sys).

Использование редактора Турбо-Пролога.

Управление курсором:

Ctrl-F или Ctrl → – на одно слово вправо;

Ctrl-A или Ctrl ← – на одно слово влево;

Ctrl-Home – в начало текста на экране;

Ctrl-End – в конец текста на экране;

Ctrl-QR или Ctrl-PgUp – в начало программы;

Ctrl-QC или Ctrl-PgDn – в конец программы.

Поместить курсор в любую строку программы можно при помощи комбинации клавиш Ctrl-F2. При этом запрашивается номер строки, после введения которого и повторного нажатия Ctrl-F2 курсор попадает в начало строки с указанным номером.

Удаление текста:

Ctrl-T – удаление слова (перед удалением курсор подгоняется к началу слова);

Ctrl-Y – удаление строки (курсор перед удалением может размещаться в любой позиции строки);

Ctrl-QY – удаление символов до конца строки, начиная с позиции размещения курсора.

Выбор режима вставки (**Insert**) или режима замены (**Overwrite**): нажимается клавиша Ins или комбинация Ctrl-V.

Автоматическое выравнивание строк. Установка режима автоматического выравнивания (**Indent** в строке статуса) или отмена его производится комбинацией Ctrl-QI. Автоматическое выравнивание функционирует только при включенном режиме вставки.

Изменение размеров окна редактора:

F5 – расширение окна до полного экрана. Возврат к начальным размерам окна осуществляется повторным нажатием F5.

Команды работы с фрагментами текста:

Ctrl-K+B – отметка начала фрагмента;

Ctrl-K+K – отметка конца фрагмента;

Ctrl-K+H – отмена выделения фрагмента;

Ctrl-K+C – копирование фрагмента в место размещения курсора;

Ctrl-K+V – перемещение фрагмента;

Ctrl-K+Y – удаление фрагмента.

Команды поиска и поиска-замены:

Ctrl-Q+F или F3 – поиск заданной последовательности символов;
Shift-F3 – повтор последнего поиска;
Ctrl-Q+A или F4 – поиск и замена;
Ctrl-L или Shift-F4 – повтор последнего поиска и замены.

Использование вспомогательного редактора – F8. После нажатия F8 появляется окно с запросом имени дополнительного требующего редактирования файла. После ввода имени файла появляется окно, в котором можно отредактировать этот файл. Выход из вспомогательного редактора – F10.

Копирование текста из другого файла – F7. Курсор перемещается в то место, куда нужно вставить фрагмент из другого файла, и нажимается F7. После ввода имени файла, из которого будет копироваться текст, в дополнительном окне редактора появится содержимое этого файла. Поместив курсор в начало копируемого фрагмента, нажимается F7, затем курсор сдвигается в конец фрагмента и снова нажимается F7. Дополнительное окно при этом исчезает, а отмеченный фрагмент скопируется в рабочий файл.

5.2 Порядок выполнения работы

1 Ознакомиться с работой в Turbo Prolog.
Набрать следующий текст программы:

```
  domains
n,f=integer
  predicates
     factorial(n,f)
  clauses
     factorial(1,1).
     factorial(N,F):-N>0,N1=N-1,factorial(N1,F1),F=F1*N.
```

Откомпилировать программу и запустить на выполнение.
В окне диалога ввести запросы:

```
factorial(5,X)  и
factorial(8,X).
```

Проанализировать правильность полученных результатов (вычислялись значения 5! и 8!).

Сохранить текст программы в файле, очистить окно редактора, снова загрузить в окно редактора файл и с помощью редактора Турбо-Пролога преобразовать загруженную программу к следующему виду:

```
predicates
factorial(integer,real)
clauses
factorial(1,1).
factorial(N,F):-N>0,N1=N-1, factorial(N1,F1),F=F1*N.
```

Сохранить полученный текст в файле. Повторить при выполнении программы запросы для вычисления 5! и 8!.

Опробовать возможности редактора Турбо-Пролога.

Содержание отчета

1 Цель работы.
2 Постановка задачи и исходные данные.
3 Схема алгоритма и программа.
4 Результат выполнения программы.
Отчет оформляется в соответствии с ГОСТ 2.105–95 *Общие требования к текстовым документам.*

Контрольные вопросы

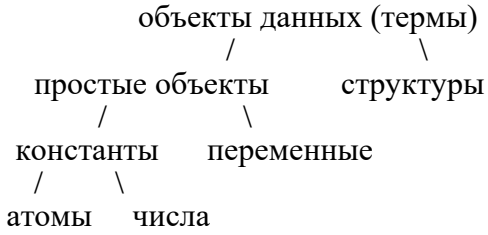
1 Назовите особенности декларативного программирования.
2 В чем особенности программирования на языке Пролог?
3 Для чего используется язык Пролог?

6 Написание программ на языке Пролог

Цель работы: приобрести навыки составления и отладки программ в среде Турбо-Пролога.

6.1 Основные теоретические положения

Объекты данных в Прологе называются термами. Терм может быть: константой, переменной, структурой (составной терм). Классификация объектов данных приведена на схеме:



Атом – это синтаксически неделимый терм. Константы относятся к одному из шести стандартных типов данных (доменов) (таблица 6.1).

Таблица 6.1 – Стандартные типы данных

Тип	Ключевое слово	Диапазон значений	Примеры
Символы	char	Все возможные одиночные символы	’a’,’B’,’?’
Целые числа	integer	-32768 ... 32767	-15, 1235, 9
Действительные числа	real	1E-307 ... 1E308	48, 2.45E-8
Строки	string	Последовательность символов (до 250)	“Слово”
Символические имена	symbol	Последовательность букв, цифр, знака подчеркивания последовательно заключенных в кавычки символов	read_data4 “Delete r1”
Файлы	file	Имя файла	Example.txt

Переменная – имя, начинающееся с большой (прописной) буквы или знака подчеркивания. Когда значение переменной неважно, то в качестве имени переменной используется знак подчеркивания. Такая переменная называется анонимной.

Структуры (сложные термы) – это объекты, которые состоят из нескольких компонент. Структура записывается с помощью указания ее функтора и компонент. Компоненты заключаются в круглые скобки и разделяются запятыми. Число компонент в структуре называется арностью структуры.

Пример структуры: data_r(12, mart,1962).

Программа на Турбо-Прологе состоит из нескольких разделов, каждому из которых предшествует ключевое слово. Типичная структура программы представлена ниже:

```
/*     комментарии      */
domains
<описание типов данных>
database
<описание предикатов динамической базы данных>
predicates
<описание предикатов>
goal
<целевое утверждение>
clauses
<утверждения>
```

В программе не обязательно наличие всех разделов. Обычно в программе должны быть по крайней мере разделы predicates и clauses.

Наиболее простым способом описания типа данных в разделе domains является следующий:

name=d,

где name − имена объектов стандартного типа;

d – один из типов: integer, real, char, string, symbol.

Предикат (отношение) в общем случае – это структура вида:

predname(komp1,komp2,...),

где predname – имя предиката;

komp1,... – типы компонентов, описанных в разделе domains или стандартные типы.

Например,

```
domains
fio=string
den,god=integer
mes=symbol
predicates
anketa(fio,den,mes,god)
```

Если в предикатах используются только стандартные типы данных, то раздел domains мажет отсутствовать:

predicates

anketa(string,integer,symbol,integer)

Предикат может состоять только из одного имени, например,

predicates

result

В разделе clauses размещаются предложения (утверждения). Предложение представляет собой факт или правило, соответствующее одному из объявленных предикатов.

Факт – простейший вид утверждения, которое устанавливает отношение между объектами.

Пример факта:

anketa(“Иванов”,5,august,1950).

Этот факт содержит атом anketa, который является именем предиката, и в скобках после него – список термов, соответствующих компонентам этого предиката. Факт всегда заканчивается точкой. Факты содержат утверждения, которые всегда являются безусловно верными.

Все предложения раздела clauses, описывающие один и тот же предикат, должны записываться друг за другом.

В разделе goal записывается третий тип предложения – вопрос, состоящий из одного или нескольких целевых утверждений (целей), разделенных запятыми и оканчивающихся точкой. Пролог-система рассматривает вопросы как цели, к достижению которых нужно стремиться. Ответ на вопрос может оказаться или положительным или отрицательным в зависимости от того, может ли быть цель достигнута или нет.

В разделе goal может быть только один вопрос, на который будет выдано только одно решение.

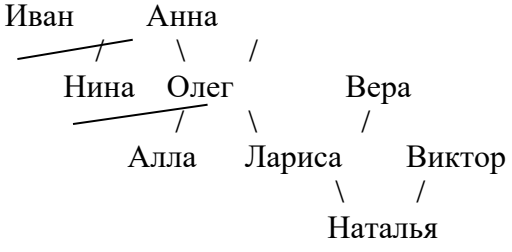
Для получения всех решений можно удалить цель из программы и задать цель на подсказку в окне DIALOG.

В вопросах могут использоваться переменные.

Применение внешних целей бывает полезно при записи коротких вопросов, а также для получения всего набора допустимых решений. Другое преимущество внешних вопросов – возможность адресовать базе данных совершенно произвольные вопросы.

6.2 Порядок выполнения работы

Описать средствами Турбо-Пролога (с помощью фактов) дерево родственных отношений, используя предикат roditel с двумя параметрами: имя родителя и имя ребенка.



В окне диалога сформировать следующие вопросы.
1 Является ли Иван родителем Нины?
2 Является ли Иван родителем Аллы?
3 Кто родители Ларисы?
4 Как зовут детей Олега?
5 Кто родитель родителя Натальи?
6 Кто чей родитель?
7 Есть ли у Нины и Олега общий родитель?
8 Как зовут жену Ивана?
9 Кто у Анны внуки?
10 Есть ли у Ларисы брат или сестра?

Содержание отчета

1 Цель работы.
2 Постановка задачи и исходные данные.
3 Схема алгоритма и программа.
4 Результат выполнения программы.
Отчет оформляется в соответствии с ГОСТ 2.105–95 *Общие требования к текстовым документам.*

Контрольные вопросы

1 Назовите особенности декларативного программирования.
2 В чем особенности программирования на языке Пролог?
3 Для чего используется язык Пролог?

7 Разработка описания на языке VHDL

Цель работы: ознакомиться с интерфейсом и возможностями сред ActiveHDL, WebPack ISE и получить практические навыки в создании и моделировании простейших комбинационных схем.

7.1 Основные теоретические положения

Создание проекта.
Выбрать пункт меню File >> New >> Design для создания нового проекта.
В появившемся мастере создания проекта выбрать Create an empty design для создания пустого проекта.
В следующем окне в качестве Synthesis tool (Инструмент синтеза) выбрать Xilinx ISE 6.x XST VHDL/Verilog. В качестве Implementation tool (Инструмент реализации) выбрать Xilinx WebPack 6.x. Выбрать семейство ПЛИС (Default Family), например Xilinx6x VIRTEXE. Убедиться, что язык моделирования по умолчанию (Default HDL language) установлен в VHDL.

В следующем окне указать имя проекта и его месторасположение (рисунки 7.1 и 7.2).

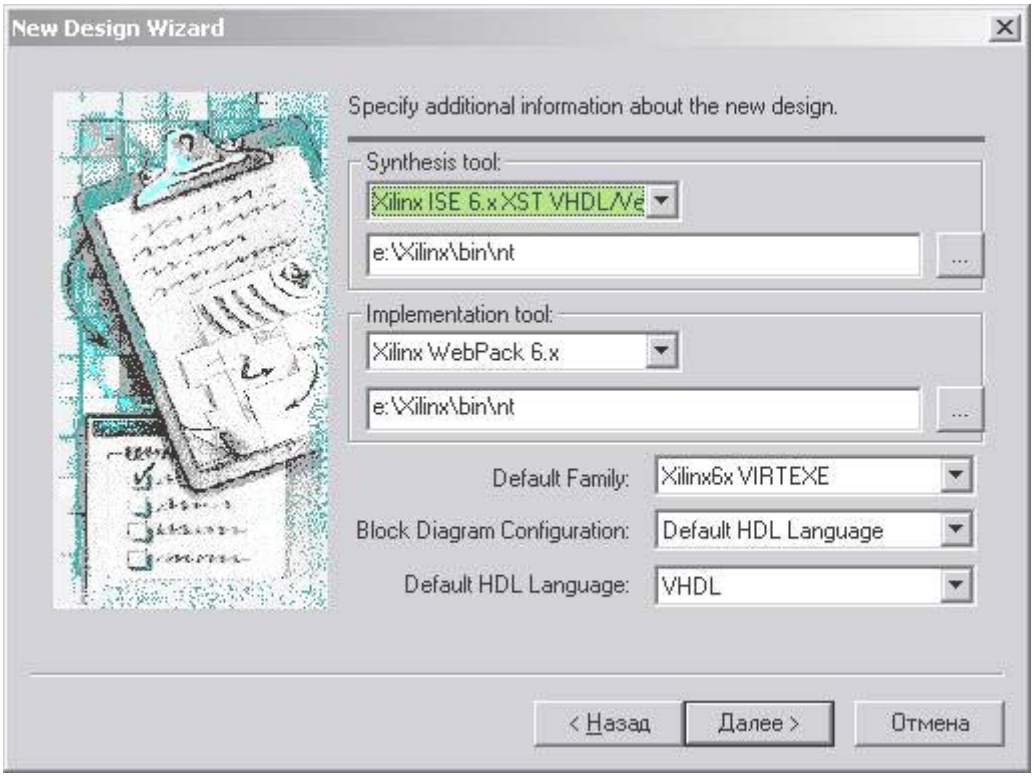


Рисунок 7.1 – Окно New Design Wizard

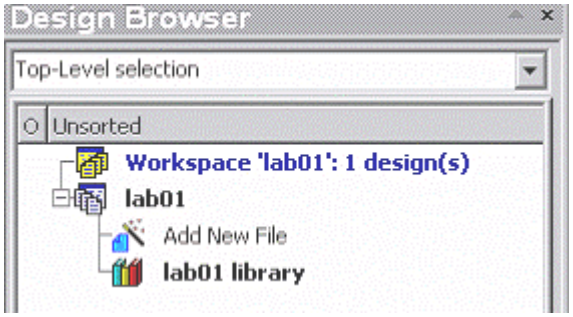


Рисунок 7.2 – Окно местоположения файлов

Создание описания устройства на VHDL.

Когда проект создан, дважды щелкнуть по пункту проекта Add New File (Добавить новый файл, рисунок 7.3).

Переключиться в режим мастера Wizards и выбрать пункт VHDL Source Code Wizard (Мастер исходного кода VHDL).

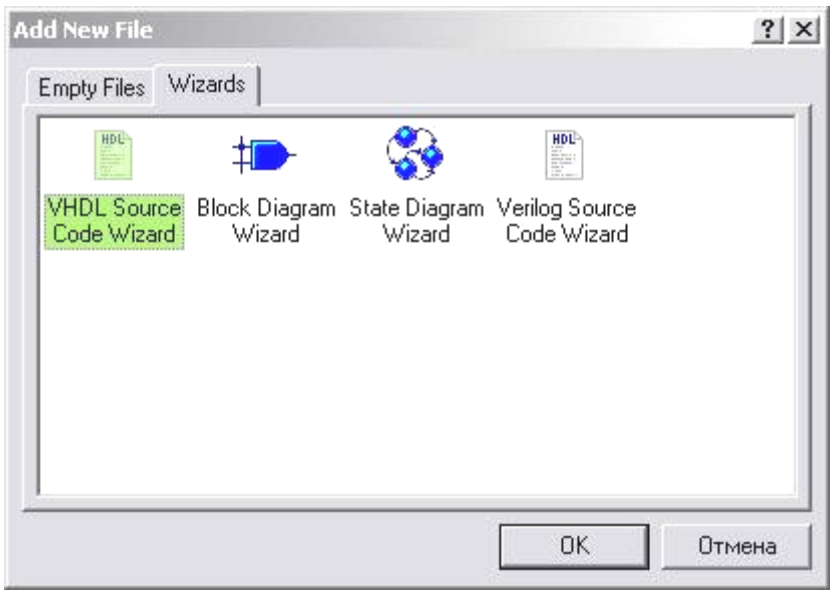


Рисунок 7.3 – Окно добавления нового файла

В первом окне мастера установить галочку Add the generated file to the design (Добавить сгенерированный файл в проект).

В следующем окне мастера указать имя файла с исходным кодом.

В следующем окне добавить необходимое количество входных и выходных портов для устройства. Добавление выполняется кнопкой New, при этом должно быть указано имя порта и его направление (вход/выход) (рисунок 7.4).

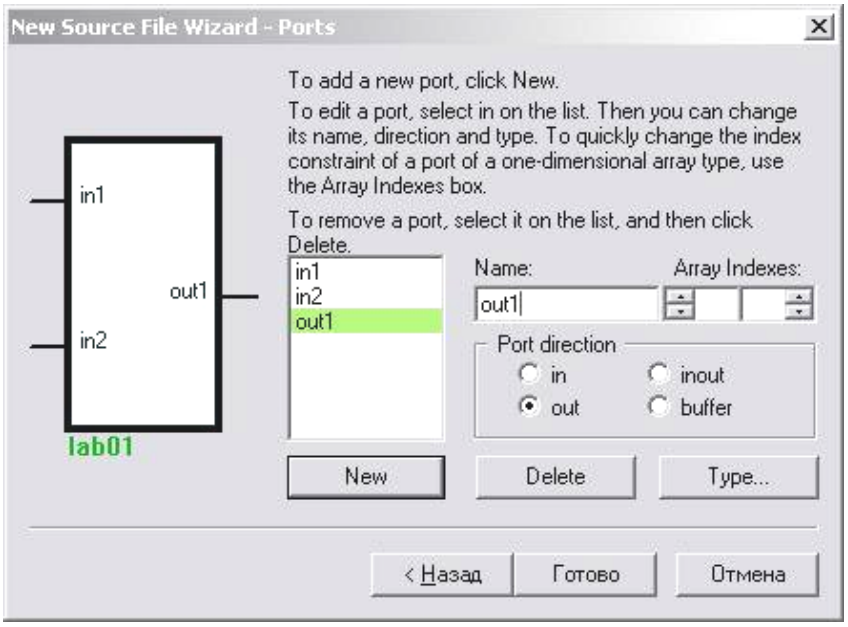


Рисунок 7.4 – Окно добавления портов

После этого шаблон VHDL-кода для устройства готов. Остается лишь добавить необходимый код архитектуры (рисунок 7.5).

```
40  architecture lab1_vhdl of lab1_vhdl is
41  begin
42
43      -- enter your statements here --
44      out1 <= in1 and in2;
45
46  end lab1_vhdl;
```

Рисунок 7.5 – Код архитектуры

Создание описания устройства при помощи редактора схем.

Переключиться в режим мастера Wizards и выбрать пункт Block Diagram Wizard (Мастер исходного кода VHDL).

В первом окне мастера установить галочку Add the generated file to the design (Добавить сгенерированный файл в проект).

Выбрать язык для генерирования исходного кода схемы (VHDL).

Задать имя файла для схемы.

В следующем окне добавить необходимое количество входных и выходных портов для устройства. Добавление выполняется кнопкой New, при этом должно быть указано имя порта и его направление (вход/выход).

После завершения работы мастера появляется окно схемы с отображением входных и выходных портов.

Для добавления в схему элементарных логических элементов, необходимо выбрать пункт меню View >> Symbols Toolbox, после чего появится окно с набором элементов. В появившемся окне Symbols Toolbox развернуть список базового набора элементов (Built-in symbols) и перетащить оттуда на схему нужные элементы (рисунок 7.6).

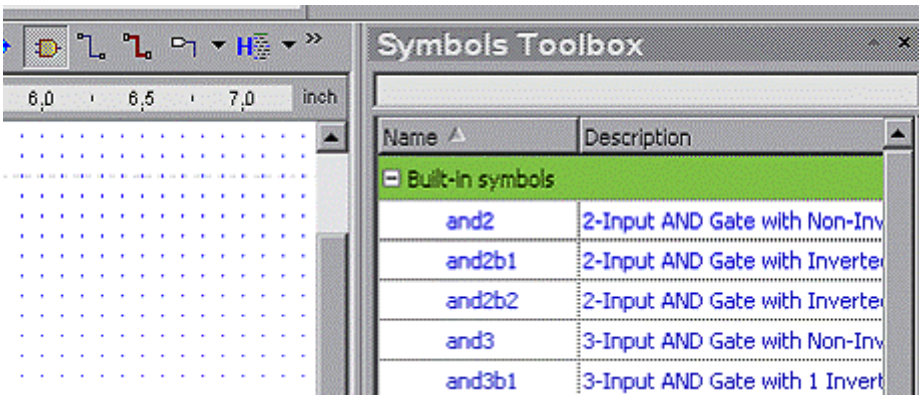


Рисунок 7.6 – Окно базового набора элементов

Основная функциональность для создания схем находится в меню Diagram и на соответствующей панели кнопок. В частности, для соединения элементов используется пункт меню Diagram >> Wire или соответствующая кнопка (рисунок 7.7).

После создания схемы развернуть её иерархию в окне проекта и сравнить соответствующий ей VHDL-код с созданным вручную VHDL-кодом.

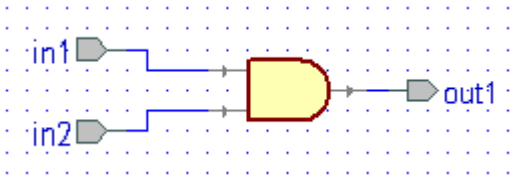


Рисунок 7.7 – Соединение элементов проводами

Задание тестовых воздействий.

Исходный код на языке VHDL сохраняется в файле с расширением VHD, схема устройства сохраняется в файле с расширением BDE. Перед проведением моделирования файлы должны быть откомпилированы (пункт Compile в контекстном меню каждого из файлов или пункт Compile All для компиляции всех файлов проекта). После успешной компиляции напротив имени файла отображается галочка. При наличии ошибок в файле отображается крестик.

Для выбора моделируемого файла необходимо развернуть иерархию для соответствующего файла в окне проекта и в контекстном меню соответствующей пары Entity-Architecture выбрать команду Set at Top-Level (рисунок 7.8).

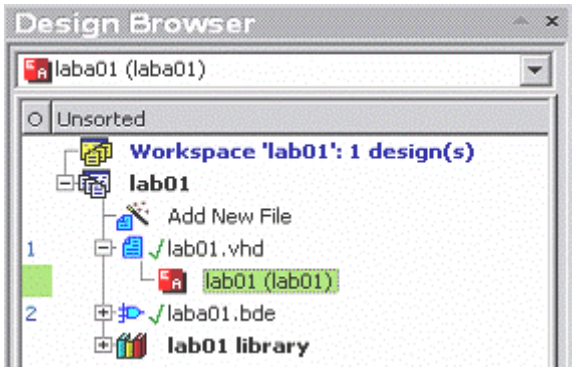


Рисунок 7.8 – Выбор моделируемого файла

Выбрать пункт меню File >> New >> Waveform, либо нажать соответствующую кнопку.

Выбрать пункт меню Waveform >> Add Signals и добавить все порты (кнопка Add) (рисунок 7.9).

Рисунок 7.9 – Добавление портов

Дважды кликнуть на строке каждого из входных сигналов в столбце Stimulator. Выбрать тип Custom (Пользовательский) или Clock (Генератор) и отметить сигнал галочкой. Для типа Clock можно задать начальное значение сигнала, начальное смещение, частоту или период и скважность (рисунок 7.10).

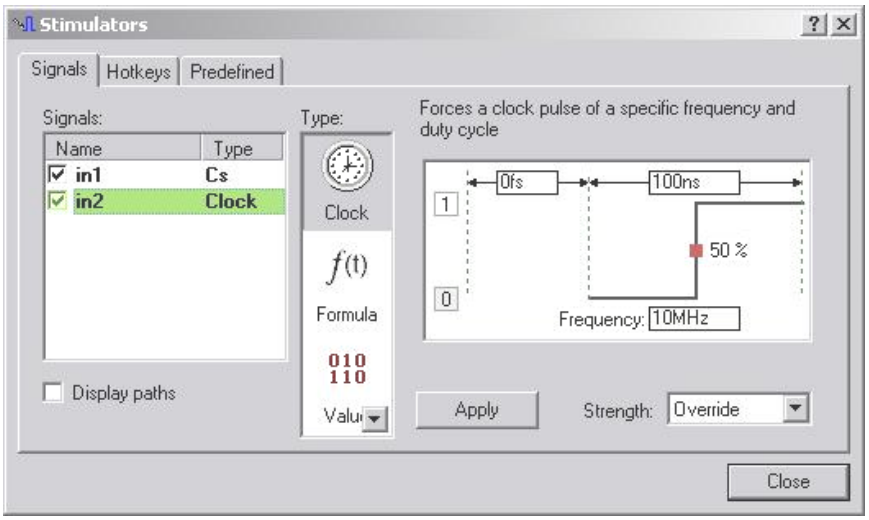


Рисунок 7.10 – Диалоговое окно Simulators

Перед тем как задавать значения входных сигналов нужно проинициализировать симуляцию проекта, в противном случае не удастся выполнить редактирование сигналов. Для этого необходимо вызвать пункты меню Simulation >> Initialize Simulation, а затем сразу же Simulation >> End Simualtion.

Перейти в режим редактирования сигналов (Waveform >> Edit Mode).

В режиме редактирования можно выделить любой участок времени для одного или нескольких портов, заданного как Custom, и нажатием клавиши «1» или «0» установить на этом промежутке соответствующее логическое значение.

Сохранить полученный Waveform.

Моделирование и просмотр результатов.

Инициализировать процесс моделирования (Simulation >> Initialize Simulation).

Запустить процесс моделирования командой Simulation >> Run Until. В этом случае понадобится ввести длину промежутка времени, для которого будут выполнены моделирования. При вызове команды Simulation >> Run Until будет выполнено моделирование для промежутка времени, на котором заданы входные воздействия.

Результат моделирования можно будет увидеть в строке, соответствующей выходному порту (рисунок 7.11).

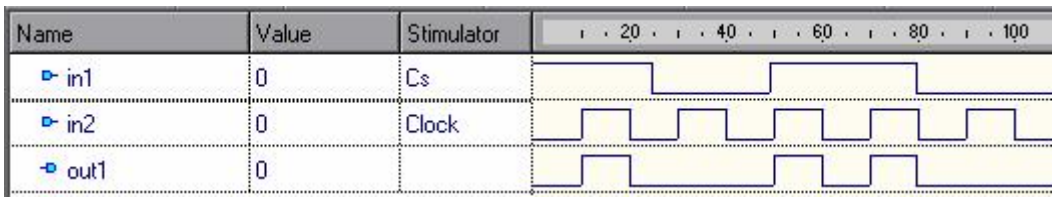


Рисунок 7.11 – Сигналы на входных и выходных портах

Для моделирования после изменения входных воздействий необходимо завершить моделирование и инициализировать его заново.

Для просмотра результатов моделирования в форме таблицы выполнить File >> New >> List.

В контекстом меню появившегося списка выбрать Add Signals (Добавить сигналы) и добавить все сигналы.

Запустить моделирование. В таблице появятся значения всех добавленных сигналов в момент смены их значений.

Размещение на кристалле.

Для настройки этапа синтеза нажать соответствующую кнопку Options в окне Design Flow (Ход разработки) (рисунок 7.12).

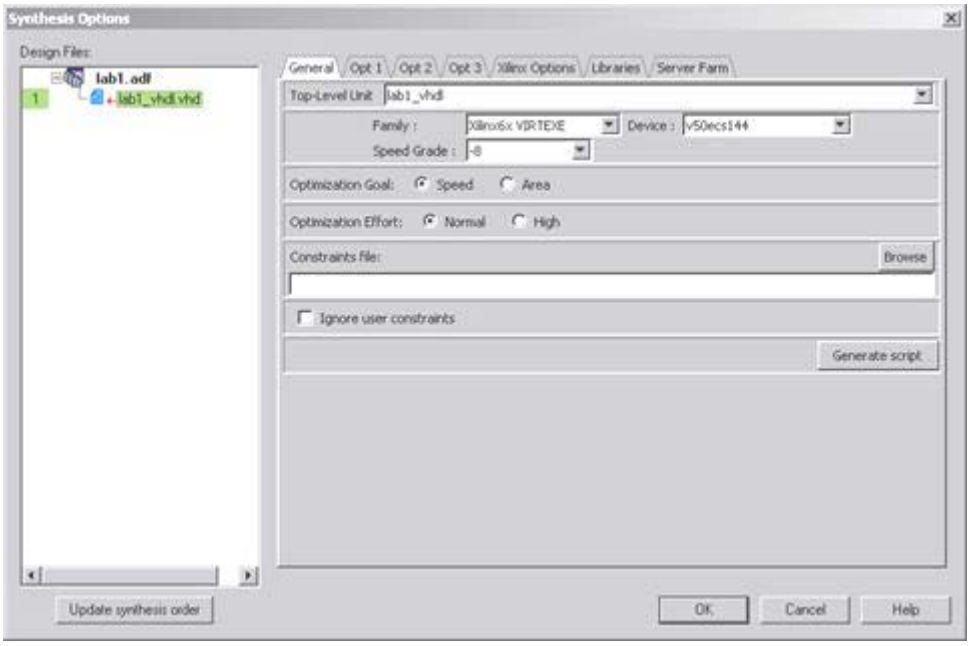


Рисунок 7.12 – Окно Options

В появившемся окне задать настройки этапа синтеза.

В контекстном меню файла, для которого нужно выполнить синтез, выбрать пункт Include to synthesis (Добавить в синтез).

В Top-Level Unit (Компонент верхнего уровня) задать этот же компонент.

Optimization Goal (Цель оптимизации) позволяет оптимизировать проект на быстродействие (Speed) или на минимизацию используемого пространства кристалла (Area).

Optimization Effort (Объем оптимизации). Установка High (Высокий) позволяет достичь большего уровня оптимизации за счет увеличения времени затрачиваемого на неё.

Для выполнения синтеза кликнуть на элемент Synthesis. О завершении синтеза будет свидетельствовать окно результатов (рисунок 7.13).

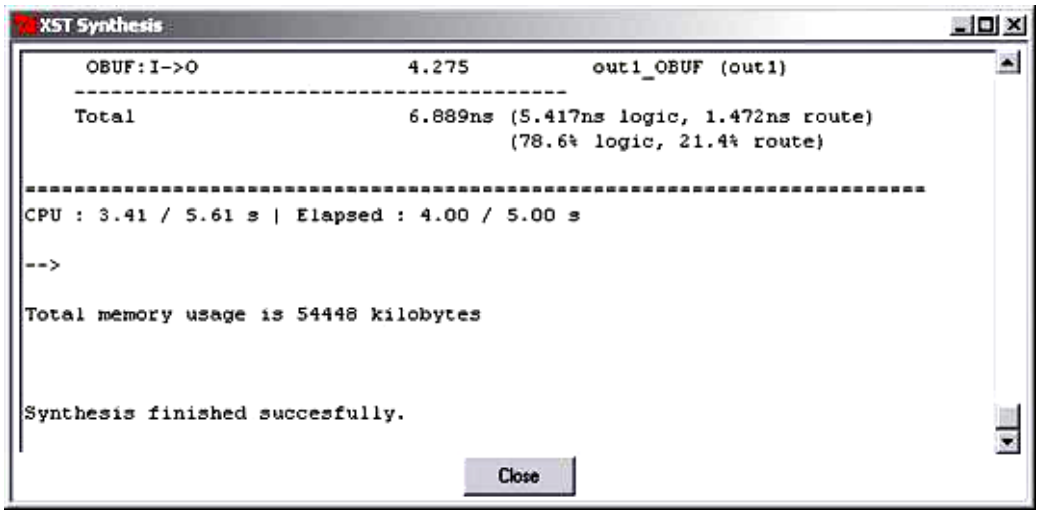


Рисунок 7.13 – Окно результатов

Для настройки этапа реализации нажать соответствующую кнопку Options. В появившемся окне задать настройки этапа синтеза (рисунок 7.14).

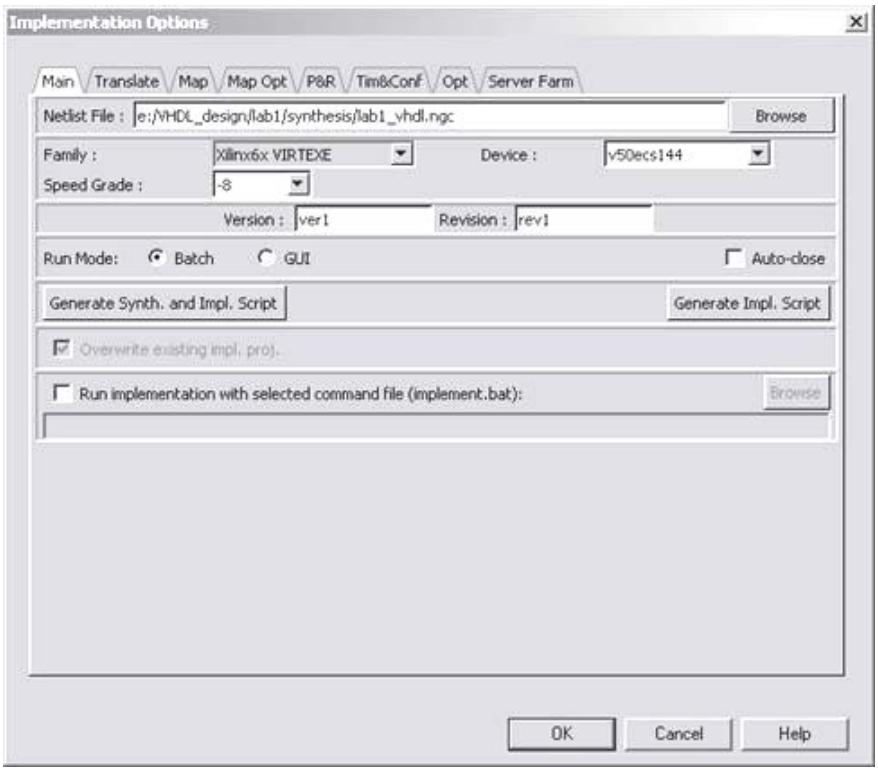


Рисунок 7.14 – Окно настройки результатов этапов синтеза

Run Mode (Режим запуска) для пакетного режима выбрать Batch, для пошагового GUI.

Для выполнения синтеза кликнуть на элемент Implementation. О завершении реализации будет свидетельствовать окно результатов (рисунок 7.15).

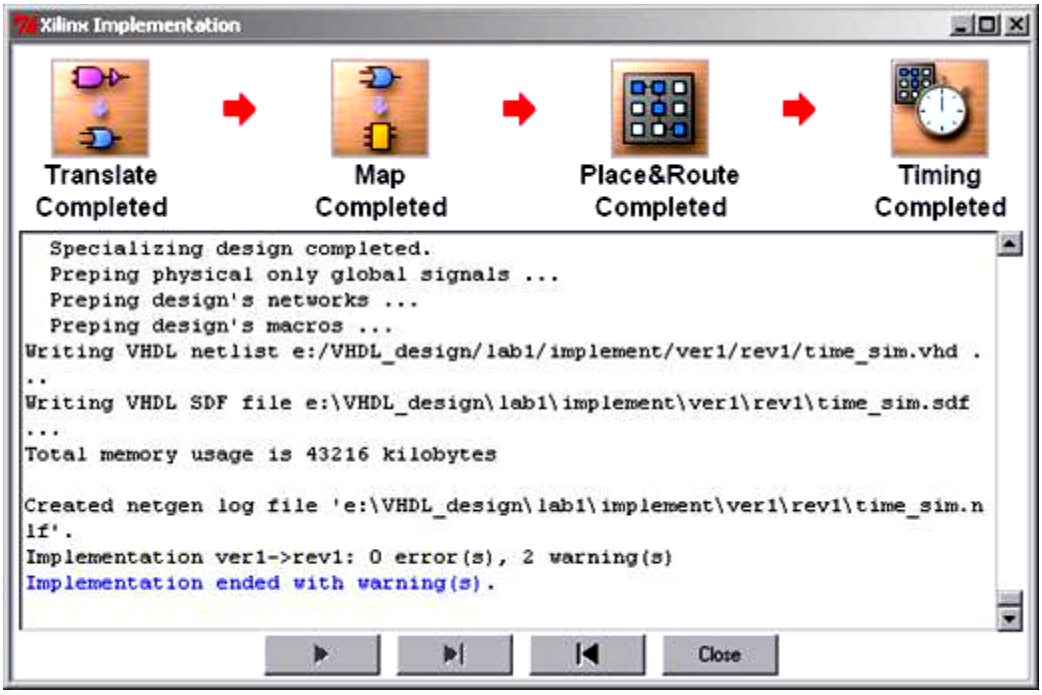


Рисунок 7.15 – Окно результатов выполнения синтеза

Для пошагового режима реализации будет запущен Xilinx Project Navigator. В окне Process View будут выведены все этапы реализации (рисунок 7.16).

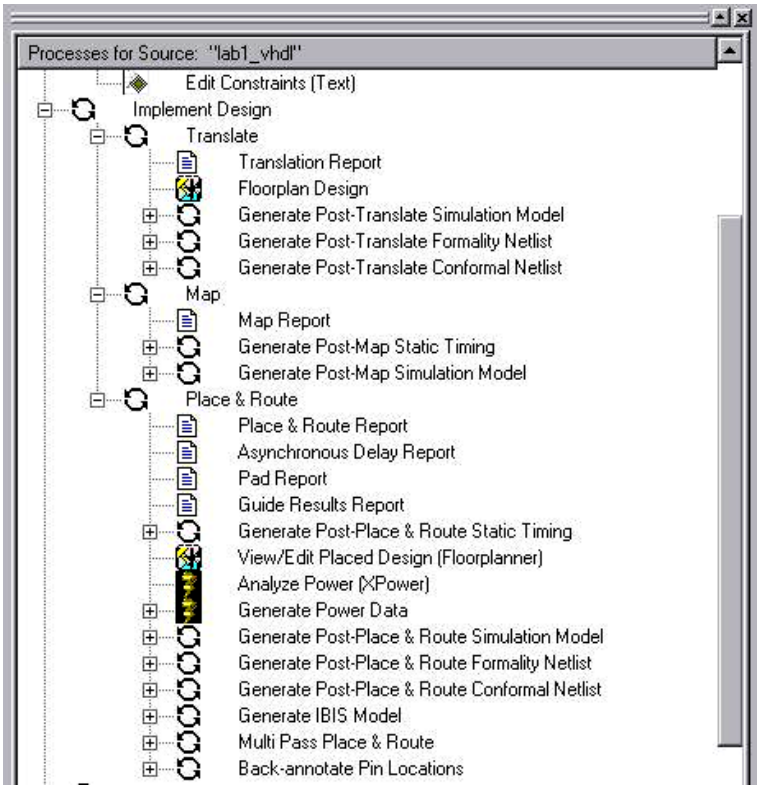


Рисунок 7.16 – Окно Process View

Запуск пункта Floorplan Design вызовет редактор, который позволит самостоятельно разместить блоки на кристалле. Слева в редакторе находится список блоков и выводов. Кликнув по элементу в списке, следует затем указать его размещение на кристалле (рисунок 7.17).

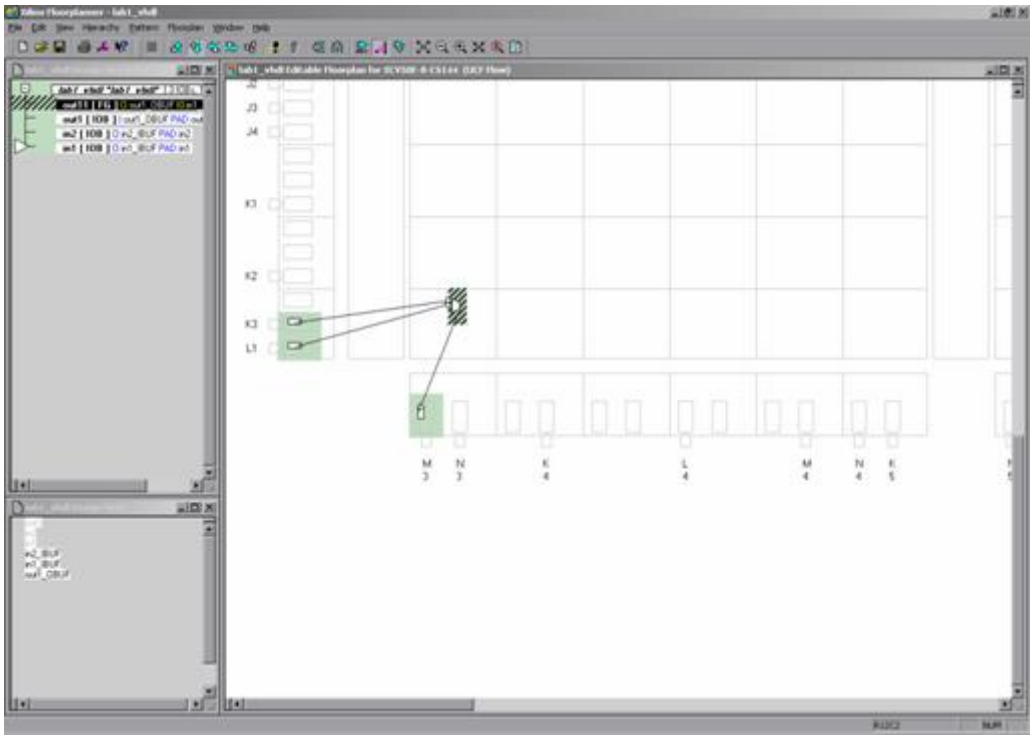


Рисунок 7.17 – Окно размещения блоков на кристалле

Запуск пункта Place & Route вызовет окончательное размещение блоков и трассировку соединений между ними.

Для просмотра отчетов о выполненных шагах можно запустить двойным щелчком пункты Translation Report, Map Report, Place & Route Report и т.д.

Для создания результирующего файла, который может быть запрограммирован в ПЛИС, следует выполнить пункт Generate Programming File (Генерировать Программируемый Файл). Для просмотра отчета выбрать Programming File Generation Report (Отчет по Генерации Программируемого Файла).

7.2 Порядок выполнения работы

1 Ознакомиться с требованиями методических рекомендаций по выполнению лабораторной работы.

2 Изучить среды ActiveHDL и WebPack ISE.

3 Рассчитать таблицу истинности для индивидуального задания.

4 Реализовать индивидуальное задание на языке VHDL в среде ActiveHDL.

5 Реализовать индивидуальное задание в схемном редакторе среды ActiveHDL.

6 Протестировать работу полученного кода.

7 Выполнить размещение полученного решения на кристалле.

Содержание отчета

1 Цель работы.

2 Постановка задачи и исходные данные.

3 Схема алгоритма и программа.

4 Результат выполнения программы.

Отчет оформляется в соответствии с ГОСТ 2.105–95 *Общие требования к текстовым документам*.

Контрольные вопросы

1 Маршрут проектирования цифровых устройств с использованием ПЛИС и языка VHDL.

2 Проект. Структурное описание. Поведенческое описание. Дерево проекта.

3 Структура программы на языке VHDL.

4 Лексические элементы языка VHDL: разделители, комментарии, идентификаторы.

5 Ключевые слова языка VHDL. Литералы. Классификация типов. Примеры.

6 Операции в выражениях языка VHDL.

7 Константы. Сигналы. Переменные. Декларации констант, сигналов, переменных.

8 Логические, арифметические, символьные типы и подтипы языка VHDL.

Список литературы

1 **Немцова, Т. И.** Программирование на языке высокого уровня. Программирование на языке С++ : учебное пособие / Т. И. Немцова, С. Ю. Голова, А. И. Терентьев; под ред. Л. Г. Гагариной. − Москва: ФОРУМ; ИНФРА-М, 2023. – 512 с.

2 **Лисицин, Д. В.** Программирование на языке ассемблера: учебное пособие / Д. В. Лисицин. – Новосибирск: НГТУ, 2018. – 100 с.

3 **Ефимова, Е. А.** Программирование на языке Пролог для задач искусственного интеллекта. Введение в логическое программирование: учебник / Е. А. Ефимова. – 2-е изд. – Москва: Рос. гос. гуманит. ун-т, 2020. – 411 с.

4 VHDL: справочное пособие по основам языка / В. П. Бабак [и др.]. – Москва: ДМК Пресс, 2020. – 224 с.

5 **Скляров, В. А.** Программирование на языке Ассемблера: учебное пособие / В. А. Скляров. – Москва: Высшая школа, 1999. – 152 с.

6 **Суворова, Е. А.** Проектирование цифровых систем на VHDL / Е. А. Суворова, Ю. Е. Шейнин. – Санкт-Петербург: БХВ-Петербург, 2003. – 576 с.