

**Московский Государственный Технический Университет
им. Н.Э. Баумана**

Попов А.Ю.

Организация ЭВМ

Методические указания

**для выполнения лабораторных работ
по курсу «Организация ЭВМ»
специальности «Вычислительные машины, комплексы, системы и сети».**

Москва, 2008

<u><i>Работа №1. ПРОЕКТИРОВАНИЕ ЦИФРОВЫХ УСТРОЙСТВ НА</i></u> <u><i>ОСНОВЕ ПЛИС.....</i></u>	<u><i>3</i></u>
Описание проектируемого устройства.....	3
Разработка схемы подавления дребезга.....	5
Разработка схемы управления 7-сегментными индикаторами.....	9
Разработка и отладка основного модуля проекта.....	12
Контрольные вопросы.....	14
<u><i>Работа №2. ПРОЕКТИРОВАНИЕ УСТРОЙСТВ УПРАВЛЕНИЯ НА</i></u> <u><i>ОСНОВЕ ПЛИС.....</i></u>	<u><i>14</i></u>
Описание разрабатываемого устройства.....	15
Порядок выполнения лабораторной работы.....	16
Примеры индивидуальных заданий.....	18
Содержание отчета.....	20
Контрольные вопросы.....	21
<u><i>Работа №3. ОРГАНИЗАЦИЯ ПАМЯТИ КОНВЕЙЕРНЫХ</i></u> <u><i>СУПЕРСКАЛЯРНЫХ ЭЛЕКТРОННЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИН</i></u> <u><i>.....</i></u>	<u><i>21</i></u>
Порядок выполнению лабораторной работы.....	22
Применение программы PCLAB при исследовании производительности ЭВМ.....	23
Описание эксперимента «Исследования расслоения динамической памяти».....	27
Описание эксперимента «Сравнение эффективности ссылочных и векторных структур».....	29
Описание эксперимента «Исследование эффективности программной предвыборки».....	32
Описание эксперимента «Исследование способов эффективного чтения оперативной памяти».....	35
Описание эксперимента «Исследование конфликтов в кэш-памяти»	37
Описание эксперимента «Сравнение алгоритмов сортировки».....	40
Содержание отчета.....	44
Контрольные вопросы.....	44
Список литературы для лабораторных работ 1, 2.....	45
Список литературы для лабораторной работы 3.....	45

Работа №1. ПРОЕКТИРОВАНИЕ ЦИФРОВЫХ УСТРОЙСТВ НА ОСНОВЕ ПЛИС

Цель работы: закрепление на практике теоретических сведений, полученных при изучении методики проектирования цифровых устройств на основе программируемых логических интегральных схем (ПЛИС), получение необходимых навыков работы с системой автоматизированного проектирования ISE WebPack 9.1 устройств на основе ПЛИС фирмы Xilinx, изучение аппаратных и программных средств моделирования, макетирования и отладки устройств на основе ПЛИС.

Для выполнения работы студенту необходимо ознакомиться с архитектурой ПЛИС FPGA Spartan 3 производства фирмы Xilinx, изучить методику проектирования устройств на основе ПЛИС с использованием САПР ISE WebPack 9.1, спроектировать и реализовать с помощью набора XC3S200 устройство управления счетом и индикацией состояния 16-разрядного счетчика.

Описание проектируемого устройства

В данной лабораторной работе осваивается методика проектирования цифровых устройств на примере разработки и реализации на ПЛИС схемы управления счетом и индикацией состояния 16-разрядного счетчика. Отладка устройства производится с помощью набора XC3S200, который содержит матрицу 7-сегментных индикаторов и кнопки, необходимые для управления разрабатываемым устройством. Состав устройства и назначение используемых ресурсов отладочного набора показаны на рисунке 1.

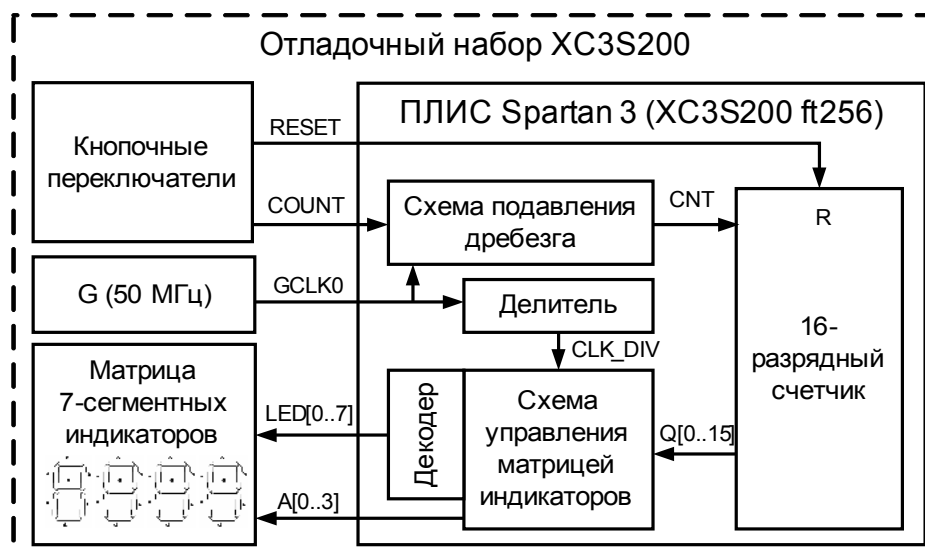


Рисунок 1 – Использование отладочного набора XC3S200 для реализации схемы управления и индикацией 16-ти разрядного счетчика

В устройстве используется синхронный 16-разрядный счетчик с асинхронным сбросом. Для управления счетом и сбросом используются две кнопки, входящие в состав отладочного набора (линии RESET и COUNT). При этом учитывается, что подача внешнего сигнала управления счетом COUNT непосредственно на вход синхронизации счетчика привела бы к многочисленным ложным срабатываниям из-за наличия дребезга при замыкании и размыкании кнопки. Для устранения этого в работе используется схема подавления дребезга, выдающая на счетчик сигнал CNT.

Состояние 16-разрядного счетчика передается на схему управления матрицей индикаторов, которая обеспечивает мультиплексированную передачу тетрад данных на декодер 7-сегментного кода, а также сопровождает выдачу данных сигналами управления анодами ($A[0..3]$). На выходе декодера формируется код активизации сегментов ($LED[0..7]$), передаваемый непосредственно на 4 индикатора, входящие в состав отладочного набора. Делитель частоты должен выдавать сигнал синхронизации CLK_DIV низкой частоты (100-200 Гц) на схему управления матрицей индикаторов.

Разработка схемы подавления дребезга

Схема подавления дребезга представляет собой автомат, воспринимающий входной сигнал COUNT от кнопки и выдающий выходной сигнал CNT в соответствии с приведенной на рисунке 2 диаграммой.

Кнопки, имеющиеся в наличие на плате XC3S200, обладают дребезгом и не снабжены схемами их подавления (триггерами Шмидта и т.д.). Как при нажатии, так и при отпускании кнопки происходит многократное изменение уровня напряжения на линии COUNT, вызванное упругими соударениями. Для предотвращения нежелательных многократных срабатываний устройств, следует построить схему, исключающую возможность прохождения сигналов в момент дребезга. Это можно осуществить с помощью дополнительного счетчика, исполняющего роль схемы задержки на длительность переходных процессов. При подаче на вход данного счетчика сигнала отладочного набора GCLK0, имеющего частоту 50 МГц, в качестве информационного сигнала окончания счета может быть использовано значение двадцатого разряда счетчика(Q[20]). Разрешение работы счетчика задается высоким уровнем сигнала CNT_EN. После окончания счета необходимо выполнить сброс счетчика в исходное нулевое состояние сигналом CNT_CLR.

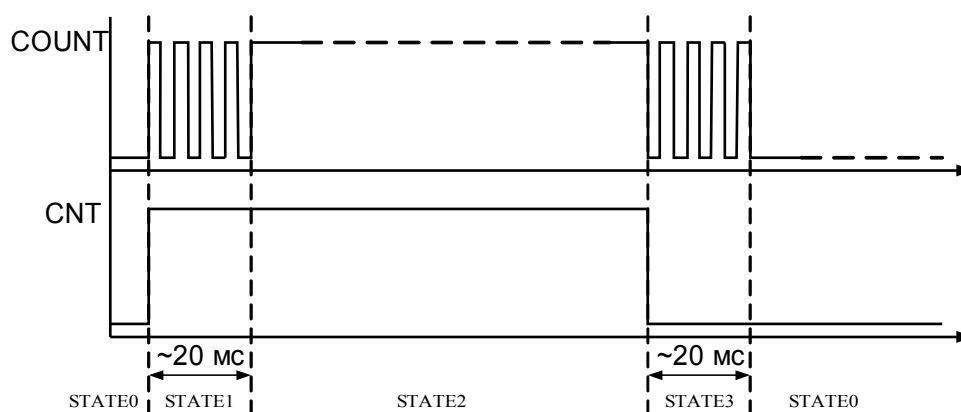


Рисунок 2 – Диаграмма работы схемы подавления дребезга

Автомат, реализующий указанную логику работы, может находиться в одном из четырех состояний: ожидания нажатия (STATE0), счет времени после нажатии (STATE1), ожидание отпускания (STATE2), счет времени после отпускания (STATE3). Диаграмма переходов состояния показана на рисунке 3.

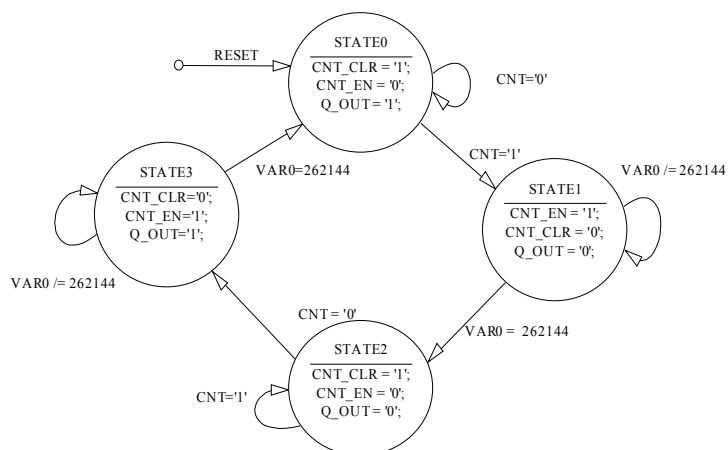


Рисунок 3 – Диаграмма состояния автомата подавления дребезга

Задание 1. Выполнить кодирование состояний автомата, представленного на рисунке 3, в соответствии с индивидуальным вариантом из таблицы 1. Для этого заполнить таблицу 2.

Таблица 1 – Индивидуальные варианты кодирования состояний автомата

Вариант:	Двоичный код состояния (Q_1Q_0)			
	State0	State1	State2	State3
1	00	01	10	11
2	00	01	11	10
3	00	10	01	11
4	00	10	11	01
5	00	11	10	01
6	00	11	01	10
7	01	00	10	11
8	01	00	11	10
9	01	10	00	11
10	01	10	11	00
11	01	11	00	10
12	01	11	10	00
13	10	00	01	11
14	10	00	11	01
15	10	01	00	11
16	10	01	11	00
17	10	11	00	01

Таблица 1 - Индивидуальные варианты кодирования состояний автомата
(окончание)

Вариант:	Двоичный код состояния (Q_1Q_0)			
	State0	State1	State2	State3
18	10	11	01	00
19	11	00	01	10
20	11	00	10	01
21	11	01	00	10

Вариант:	Двоичный код состояния (Q ₁ Q ₀)			
	State0	State1	State2	State3
18	10	11	01	00
22	11	01	10	00
23	11	10	00	01
24	11	10	01	00

По таблице выходов 2 определить функции сигналов управления: CNT = f(Q₁,Q₀), CNT_EN = f(Q₁,Q₀), CNT_CLR = f(Q₁,Q₀). Результаты занести в отчет.

Таблица 2 – Таблица выходов

Состояние	State0	State1	State2	State3
Двоичный код состояния (Q ₁ Q ₀)				
CNT	0	1	1	0
CNT_EN	0	1	0	1
CNT_CLR	1	0	1	0

Синтезировать схему автомата с использованием динамических синхронных D-триггеров с асинхронным сбросом и установкой (элементы FDC и FDP библиотеки) и шестнадцатиразрядных счетчиков (элемент CR16CE), управляемых фронтом сигнала синхронизации CLK. Для определения функций D0=f(Q[20],CNT,Q₁,Q₀) и D1=f(Q[20],CNT,Q₁,Q₀) заполнить таблицу 3. Результаты занести в отчет.

Таблица 3 – Сигналы D1 и D2

CNT	Q[20]	Q1(t)	Q0(t)	Q1(t+1)	Q0(t+1)	D1	D0	Описание события
0	x							Ожидание нажатия кнопки
1	x							Нажатие кнопки
x	0							Ожидание окончания счета
x	1							Конец счета
1	x							Ожидание отпускания
0	x							Отпускание кнопки
x	0							Ожидание окончания счета
x	1							Конец счета

Задание 2. Собрать модель полученного устройства в редакторе схем САПР ISE WebPack 9.1. Для этого выполнить следующие действия:

- Запустить САПР ISE WebPack 9.1.
- В меню File выбрать пункт New Project.
- Указать название и путь к файлу создаваемого проекта.
- Указать тип модуля верхнего уровня: Schematic. Нажать кнопку Next.
- В поле Product Category указать: All.
- В поле Family указать: Spartan3.
- В поле Device указать: XC3S200.
- В поле Package указать: FT256.
- В поле Speed указать: -4.
- В поле Synthesis Tool указать: XST(VHDL/Verilog).
- В поле Simulator указать: ISE Simulator (VHDL/Verilog).
- В поле Preferred Language: VHDL. Трижды нажать кнопку Next, Нажать кнопку Finish.
- Создать новое схемотехническое описание проекта, выбрав в меню Project пункт New Source. Далее выбрать тип описания (Schematic) и указать имя и путь к создаваемому файлу описания. Нажать кнопки Next и Finish.

В итоге будет создан и открыт для редактирования файл схемотехнического описания. Для описания схемы необходимо поместить на рабочее поле компоненты, выбрав их во вкладке Symbols окна Sources. В данном задании потребуются компоненты FDC, FDP, CR16CE, AND, OR, VCC, GND.

Соединение компонентов выполняется с помощью цепей и шин. Для создания цепи или шины необходимо над рабочим полем вызвать контекстное меню с помощью правой кнопки мыши. Далее, в контекстном меню выбрать пункт Add и подпункт Wire. Если порт компонента является шиной, то цепь также будет иметь такой тип.

Для подключения цепи или группы цепей к существующей шине используется разветвитель Bus Tap. Для его добавления в контекстном меню выбрать пункты Add и Bus Tap, после чего необходимо поместить разветвитель на шине. Указание соответствия цепей в шине с цепями, подведенными к

разветвителю, выполняется с помощью имен. Например, если шина носит имя XLXN_1(15:0), то для выбора старшей цепи следует назвать подведенную к разветвителю цепь, как XLXN_1(15). Порты схемы подключаются к цепям с помощью компонента I/O Marker. Для этого в контекстном меню необходимо выбрать пункты Add и I/O Marker.

Задание 3. В интегрированном редакторе тестов САПР ISE WebPack 9.1 разработать тест для полученного устройства и выполнить моделирование его работы в ISE Simulator.

Для этого необходимо создать новое описание теста проекта, выбрав в меню Project пункт New Source. Далее выбрать тип описания (Test Bench WaveForm) и указать имя и путь к создаваемому файлу теста. Далее нажать на кнопку Next и в открывшемся диалоге выбрать тестируемое описание, после чего нажать на кнопки Next и Finish. В результате будет создан файл графического тестового воздействия для выбранного описания и будет вызван диалог его настройки. В этом диалоге необходимо указать сигнал синхронизации (например, CLK) и время тестирования в поле Initial Length of Test Bench (остальные параметры можно оставить неизменными).

После создания теста и его сохранения можно приступить к моделированию, для чего активизировать вкладку Sources в окне Sources. В выпадающем списке Sources For выбрать строку Behavioral Simulation и выбрать запускаемый тест в дереве описаний проекта. После этого во вкладке Processes окна Processes выбрать пункт Xilinx ISE Simulator и пункт Simulate Behavioral Model.

Схему устройства и результаты моделирования занести в отчет.

Разработка схемы управления 7-сегментными индикаторами

Для индикации информационных сигналов большой разрядности при отладке устройств с помощью набора XC3S200 целесообразно использовать имеющиеся четыре 7-сегментных индикатора. Управление их работой осуществляется благодаря подаче восьми общих сигналов управления сегментами LED[0..7] одновременно с установкой в активный низкий уровень сигнала выборки анода A[0..3]. Таким образом, состояние линий LED должно устанавливаться схемой управления в соответствии с активным в данный

момент 7-сегментным индикатором. Диаграмма работы устройства управления четырьмя 7-сегментными индикаторами показана на рисунке 4. Четыре тетрады сигналов данных ($Q[0..3]$, $Q[4..7]$, $Q[8..11]$, $Q[12..15]$) должны быть преобразованы в код активизации индикаторов с помощью декодера. Темп активизации индикаторов должен обеспечивать отсутствие видимого мерцания сегментов (100-200 Гц). Для задания частоты активизации CLK_DIV целесообразно использовать дополнительный 16-разрядный счетчик – делитель частоты CB16CE, соединенный по выходу CEO с буфером разрешения синхросигнала BUFGCE.

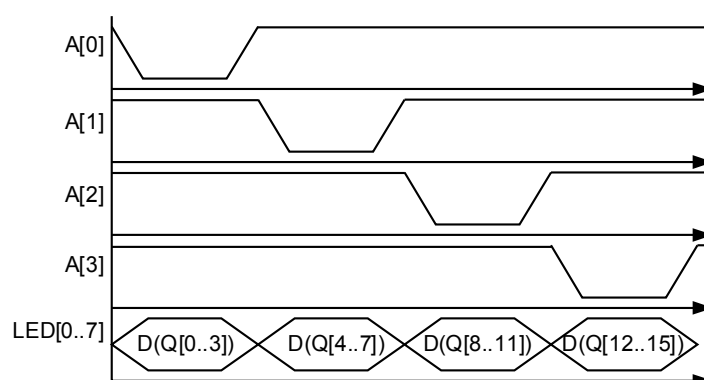


Рисунок 4 – Диаграмма работы устройства управления четырьмя 7-сегментными индикаторами

Для преобразования четырех информационных сигналов в код активизации восьми светодиодов используется табличное перекодирование, реализуемое программой синтеза при помощи LUT-таблиц.

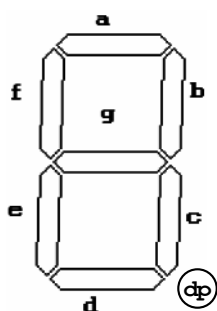


Рисунок 5 - Назначение сегментов индикатора

Расположение сегментов индикатора показано на рисунке 5, а коды их активизации для различных значений входного четырехразрядного слова $D[0..3]$ указаны в таблице 4 (нулевой разряд означает активизацию сегмента; сегмент DP не активизируется).

Таблица 4 – Значения информационных входов активизации сегментов 7-сегментного индикатора для возможных значений информационного слова $D[0..3]$

D[0..3]	DP	A	B	C	D	E	F	G
0000	1	0	0	0	0	0	0	1
0001	1	1	0	0	1	1	1	1
0010	1	0	0	1	0	0	1	0
0011	1	0	0	0	0	1	1	0
0100	1	1	0	0	1	1	0	0
0101	1	0	1	0	0	1	0	0
0110	1	0	1	0	0	0	0	0
0111	1	0	0	0	1	1	1	1
1000	1	0	0	0	0	0	0	0
1001	1	0	0	0	0	1	0	0
1010	1	0	0	0	1	0	0	0
1011	1	1	1	0	0	0	0	0
1100	1	0	1	1	0	0	0	1
1101	1	1	0	0	0	0	1	0
1110	1	0	1	1	0	0	0	0
1111	1	0	1	1	1	0	0	0

Задание 4. Разработать устройство управления, принимающее 16-разрядное информационное слово Q[0..15] и управляющее их последовательной выдачей по шине D[0..3] на декодер 7-сегментных индикаторов в соответствии с показанной на рисунке 4 диаграммой. Для этого создать файл VHDL, содержащий следующий текст описания:

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
ENTITY Seven_Segment_Driver IS
  PORT (
    clk_div   : IN           std_logic;
    Q         : IN           std_logic_vector(15 DOWNTO 0);
    rst       : IN           std_logic;
    D         : OUT          std_logic_vector(3 DOWNTO 0);
    A         : INOUT        std_logic_vector(3 DOWNTO 0)  );
END ENTITY Seven_Segment_Driver;
ARCHITECTURE Struct OF Seven_Segment_Driver IS
BEGIN
  A_drive: PROCESS (clk_div, rst)
    BEGIN
      IF (rst = '1') THEN
        A <= "1110";
      ELSIF (clk_div'EVENT AND clk_div='1') THEN
        A(3) <= A(2);
        A(2) <= A(1);
        A(1) <= A(0);
        A(0) <= A(3);
      END IF;
    END PROCESS A_drive;
  D(0) <= (Q(0) AND NOT(A(0)))
          OR (Q(4) AND NOT(A(1)))
          OR (Q(8) AND NOT(A(2)))
          OR (Q(12) AND NOT(A(3)));

```

```

D(1)  <= (Q(1)  AND NOT(A(0)))
        OR (Q(5)  AND NOT(A(1)))
        OR (Q(9)  AND NOT(A(2)))
        OR (Q(13) AND NOT(A(3)));
D(2)  <= (Q(2)  AND NOT(A(0)))
        OR (Q(6)  AND NOT(A(1)))
        OR (Q(10) AND NOT(A(2)))
        OR (Q(14) AND NOT(A(3)));
D(3)  <= (Q(3)  AND NOT(A(0)))
        OR (Q(7)  AND NOT(A(1)))
        OR (Q(11) AND NOT(A(2)))
        OR (Q(15) AND NOT(A(3)));
END ARCHITECTURE Struct;

```

В интегрированном редакторе тестов САПР ISE WebPack 9.1 разработать тест для полученного устройства и выполнить моделирование его работы в ISE Simulator. Описание устройства и результаты моделирования занести в отчет. Подключить устройство к текущему проекту, выбрав в меню Project пункт Add Source.

Задание 5. Разработать поведенческое VHDL описание схемы преобразования четырехразрядного информационного кода D[0..3] в код активизации 7-сегментного индикатора LED[0..7] в соответствии с таблицей 4. Подключить устройство к библиотеке компонентов текущего проекта. Текст описания занести в отчет.

Разработка и отладка основного модуля проекта

Для реализации частей разработанного устройства и последующей их отладки необходимо создать файл ограничений, содержащий назначение контактов целевой микросхемы. Вариант назначения сигналам контактов микросхемы, расположенной на плате набора XC3S200, представлены в таблице 5.

Таблица 5 – Вариант назначения контактов микросхемы сигналам схемы

Сигнал	Номер контакта	Назначение
CLK	T9	Глобальный сигнал GCLK0(50 МГц)
COUNT	M13	Сигнал от кнопки 3
RESET	L14	Сигнал от кнопки 0
LED[0]	N16	Сигнал управления сегментом G
LED[1]	F13	Сигнал управления сегментом F
LED[2]	R16	Сигнал управления сегментом E
LED[3]	P15	Сигнал управления сегментом D
LED[4]	N15	Сигнал управления сегментом C
LED[5]	G13	Сигнал управления сегментом B
LED[6]	E14	Сигнал управления сегментом A

Сигнал	Номер контакта	Назначение
LED[7]	P16	Сигнал управления сегментом DP
A[0]	D14	Сигнал управления анодом 0
A[1]	G14	Сигнал управления анодом 1
A[2]	F14	Сигнал управления анодом 2
A[3]	E13	Сигнал управления анодом 3

Задание 6. В редакторе схем САПР ISE WebPack 9.1. собрать модель полученного устройства, включающего схему подавления дребезга, схему управления 7-сегментными индикаторами, 16-разрядный счетчик и схему деления частоты для активизации 7-сегментных индикаторов (библиотечные элементы CB16CE и BUFGCE). Для использования описаний в составе описаний более высокого уровня необходимо создать для них символы, благодаря чему описания включается в библиотеку. Для этого следует во вкладке Sources окна Sources выбрать описание проекта, после чего во вкладке Processes окна Processes выбрать ветвь Design Utilities и пункт Create Schematic Symbol.

Созданное описание следует указать в качестве модуля верхнего уровня, для чего необходимо выбрать его в дереве описаний проекта во вкладке Sources окна Sources, после чего в меню Sources выбрать пункт Set as Top Module. Схему основного модуля занести в отчет.

Задание 7. В программе Xilinx PACE создать файл ограничений *.ucf, в котором назначить внешние выводы сигналам разрабатываемого устройства в соответствии с таблицей 5. Для этого выбрать модуль верхнего уровня в дереве описаний проекта во вкладке Sources окна Sources, после чего во вкладке Processes окна Processes выбрать ветвь User Constraints и пункт Assign Package Pins. В окне редактора PACE назначение контактов выполняется в поле LOC.

Задание 8. В САПР ISE WebPack 9.1 выполнить автоматический синтез технологической схемы, размещение и трассировку полученного устройства на кристалле Spartan3 XC3S200 ft256, генерировать файл конфигурации ПЛИС (*.bin). Для этого в окне Sources выбрать вкладку Sources и в списке Sources For выбрать строку Synthesis/Implementation. После этого в дереве описаний проекта выбрать описание верхнего уровня, а во вкладке Processes окна Processes

выбрать ветвь Generate Programming File и пункт Programming File Generation Report.

Занести в отчет общие сведения о результатах проектирования устройства с вкладки Design Summary. Сделать выводы о быстродействии полученного устройства, используя отчет Static Timing Report.

Задание 8. Выполнить программирование макетной ПЛИС Spartan3 отладочного набора XC3S200. Для этого в окне Sources выбрать вкладку Sources и в списке Sources For выбрать строку Synthesis/Implementation. После этого в дереве описаний проекта выбрать описание верхнего уровня, а во вкладке Processes окна Processes выбрать ветвь Generate Programming File и пункт Configure Device. В результате будет запущен модуль iMPACT. В открывшемся диалоге выбора способа программирования отметить пункт Configure devices using Boundary-Scan и выбрать автоматический способ идентификации. В результате будет определена цепочка, состоящая из ПЛИС и Flash ПЗУ.

После автоматического определения цепочки граничного сканирования по JTAG интерфейсу необходимо запрограммировать ПЛИС XC3S200. Для этого необходимо указать файл конфигурации ПЛИС (*.bit), полученный в задании 7, после чего в окне iMPACT Processes меню выбрать пункт Program.

Провести тестирование разработанного устройства. Результаты тестирования представить в отчете.

Контрольные вопросы

1. Назовите основные этапы проектирования цифровых устройств на основе ПЛИС.
2. Перечислите устройства, входящие в состав отладочного набора XC3S200.
3. Какую информацию содержит файл ограничений *.ucf.
4. Какой стиль описания на VHDL использован в примере описания драйвера 7-сегментных индикаторов.

Работа №2. ПРОЕКТИРОВАНИЕ УСТРОЙСТВ УПРАВЛЕНИЯ НА ОСНОВЕ ПЛИС

Цель работы: закрепление на практике теоретических знаний о способах реализации устройств управления, исследование способов организации узлов ЭВМ, освоение принципов проектирования цифровых устройств на основе ПЛИС.

В ходе работы студенту необходимо разработать, реализовать и отладить устройство управления на основе ПЛИС Spartan3 XC3S200.

Описание разрабатываемого устройства

В лабораторной работе необходимо разработать и реализовать на ПЛИС XC3S200 управляющий автомат схемного типа, обрабатывающий входное командное слово $C=\{C_0, \dots, C_{i-1}\}$ и осведомительные сигналы $U=\{U_0, \dots, U_{j-1}\}$, выдающий сигналы управления $V=\{V_0, \dots, V_{k-1}\}$ операционному блоку в соответствии с приведенной ниже логикой работы.

При отладке устройства в работе используется набор Xilinx XC3S200, содержащий макетную ПЛИС Spartan 3 типа XC3S200 ft256, средства индикации, кнопочные и ползунковые переключатели, энергонезависимую память конфигурации ПЛИС и другие отладочные средства. Для автоматизации процесса проектирования в работе используется свободно распространяемая САПР Xilinx ISE 9.1 Web Pack.

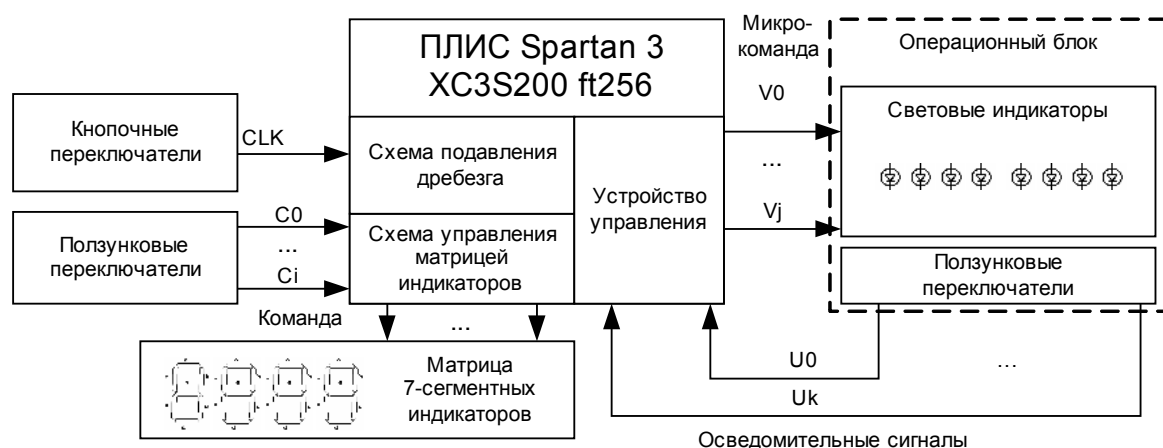


Рисунок 6 – Общая схема отладки устройств управления с помощью набора XILINX XC3S200.

На рисунке 6 показана схема отладки устройства управления с помощью отладочного набора XC3S200. Для задания входных сигналов применяются переключатели, имеющиеся на плате. Состояние осведомительных сигналов и

командное слово удобно задавать с помощью восьми имеющихся ползунковых переключателей. В качестве синхросигнала целесообразно использовать выходной сигнал кнопочного переключателя. Индикация управляющих сигналов может быть осуществлена с помощью восьми имеющихся световых индикаторов и четырех 7-сегментных светодиодных индикаторов.

К моменту нажатия на кнопку, используемую для задания синхросигнала, ползунковые переключатели должны находиться в одном из двух устойчивых состояний, что исключает искажение сигналов. При этом следует учитыватьдребезг, появляющийся при работе кнопочного переключателя. Для исключения многократных срабатываний следует использовать стандартную схему подавлениядребезга контактов, разработанную студентом по индивидуальному заданию в лабораторной работе 1.

При разработке устройства управления не следует учитывать разрядность операционных блоков. Времена задержек сигналов во всех комбинационных схемах считать меньшими периода синхросигнала.

Порядок выполнения лабораторной работы

Задание 1. В САПР Xilinx ISE 9.1 Web Pack создать проект описания устройства на основе ПЛИС Spartan 3 типа XC3S200 ft256.

Задание 2. По текстовому описанию индивидуального задания (выдаваемому преподавателем) разработать алгоритм функционирования устройства управления, выбрать тип управляющего автомата (синхронный или асинхронный автомат; автомат Мили, Мура, комбинированный; автомат с синхронным входом, синхронным выходом). Произвести кодирование состояний управляющего автомата. Разработать описание устройства управления на языке VHDL и включить его в проект.

Задание 3. Включить в проект VHDL описания устройств подавлениядребезга и управления матрицей 7-сегментных индикаторов, разработанные в лабораторной работе №1.

Задание 4. Разработать тестовые воздействия для моделирования работы устройства. Выполнить функциональное моделирование устройства микропрограммного управления в модуле Xilinx ISE Simulator. Результаты

моделирования занести в отчет. Синтезировать RTL-модель и технологическую модель устройства.

Задание 5. Для реализации частей разработанного устройства и последующей их отладки необходимо создать файл ограничений, содержащий назначение контактов целевой микросхемы. Назначение ресурсов отладочного набора выводам микросхемы приведено в таблице 6.

Таблица 6 – Назначения контактов микросхемы ресурсам отладочного набора

Номер контакта	Назначение
T9	Глобальный тактовый сигнал GCLK0(50 МГц)
M13	Сигнал от кнопки 3
L14	Сигнал от кнопки 0
F12	Ползунковый переключатель 0
G12	Ползунковый переключатель 1
H14	Ползунковый переключатель 2
H13	Ползунковый переключатель 3
J14	Ползунковый переключатель 4
J13	Ползунковый переключатель 5
K14	Ползунковый переключатель 6
K13	Ползунковый переключатель 7
K12	Светодиод 0
P14	Светодиод 1
L12	Светодиод 2
N14	Светодиод 3
P13	Светодиод 4
N12	Светодиод 5
P12	Светодиод 6
P11	Светодиод 7

Задание 6. Выполнить автоматическое размещение и трассировку модели на кристалле ПЛИС типа XC3S200 ft256. Выполнить моделирование управляющего устройства с учетом результатов размещения и трассировки в Xilinx ISE Simulator. Результаты моделирования занести в отчет. По результатам размещения и трассировки определить: максимальную частоту работы устройства управления, максимальную задержку от контакта до входа, максимальную задержку от входа до контакта.

Задание 6. С помощью отладочного набора XC3S200 и программы iMPACT выполнить программирование макетной ПЛИС. Провести тестирование разработанного устройства с использованием отладочного набор Xilinx XC3S200.

Примеры индивидуальных заданий

Номер варианта задания определяется по журналу подгруппы.

Вариант 1. Разработать устройство управления циклической FIFO памятью.

Операционное устройство состоит из следующих блоков (Рисунок 7):

- регистрового файла;
- счетчика номера первого занятого регистра;
- счетчика номера первого свободного регистра;
- мультиплексора адреса;
- декодера выборки регистра;
- схемы контроля переполнения;
- схемы контроля опустошения.
- шинного формирователя.

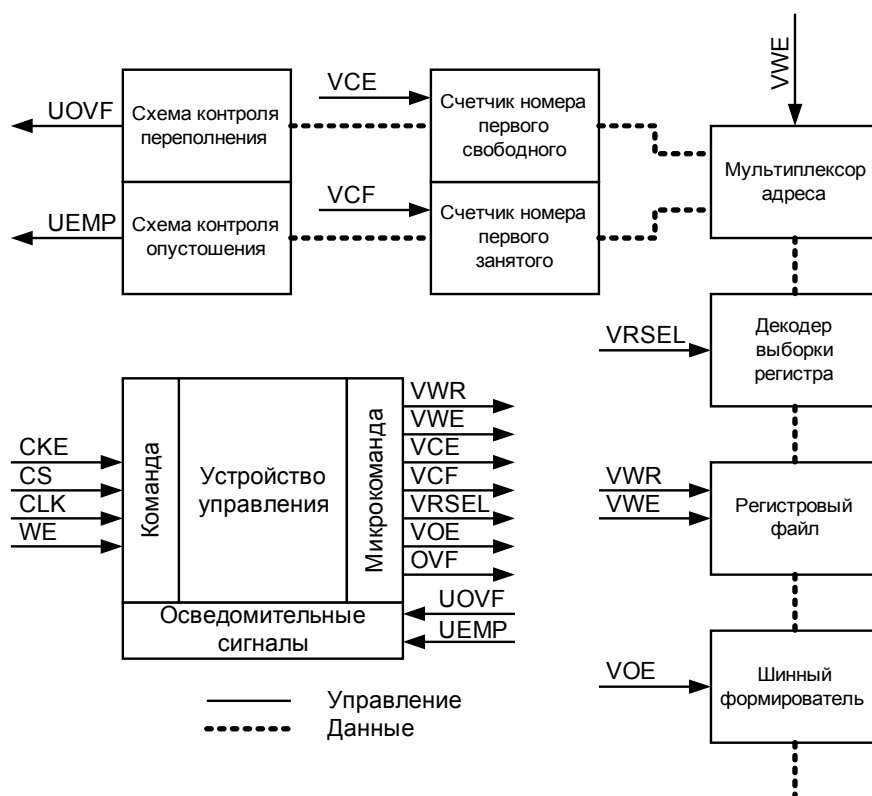


Рисунок 6. Пример организации циклической FIFO памяти.

Назначение линий в группах C,U и V поясняются в таблице:

Сигнал	Назначение
<i>Командное слово C</i>	
WE	Тип операции (чтение, запись)
CS	Сигнал выборки кристалла, инициализирующий операции

Сигнал	Назначение
CKE	Сигнал разрешения синхросигнала/управление режимом энергосбережения
CLK	Сигнал синхронизации
<i>Осведомительные сигналы U</i>	
UOVF	Регистровый файл полон
UEMP	Регистровый файл пуст.
<i>Сигналы управления V</i>	
VWR	Разрешение записи данных
VWE	Тип операции
VCE	Строб увеличения адреса первого свободного
VCF	Строб увеличения адреса первого занятого
VRSEL	Сигнал разрешения выборки регистра
VOE	Сигнал управления шинным формирователем
OVF	Сигнал уведомления об ошибке

Вариант 2. Разработать устройство управления блокируемой наборно-ассоциативной кэш-памятью со сквозной записью и FIFO алгоритмом замещения.

Операционное устройство состоит из следующих блоков:

- двух регистровых файлов (банков);
- схемы сравнения тегов и выборки первого совпавшего;
- триггера, хранящего номер последнего использованного банка;
- регистра тега;
- регистра номера набора;
- дешифратора набора;
- регистра данных;
- шинного формирователя.

Назначение линий в группах C,U и V поясняются в таблице:

Сигнал	Назначение
<i>Командное слово C</i>	
WE	Тип операции (чтение, запись)
CS	Сигнал выборки кристалла, инициализирующий операции
CLK	Сигнал синхронизации
<i>Осведомительные сигналы U</i>	
UNIT1	Найдено совпадение тега в банке 1
UNIT2	Найдено совпадение тега в банке 2
<i>Сигналы управления V</i>	
VLD	Разрешение записи данных
VLA	Разрешение записи тега и набора

Сигнал	Назначение
VWE	Тип операции
VBN	Сигнал управления триггером банка
VRSEL	Сигнал разрешения выборки регистра
VHIT	Сигнал уведомления об нахождении данных
VOE	Сигнал управления шинным формирователем

Вариант 3. Разработать устройство управления аппаратным стеком.

Операционное устройство состоит из следующих блоков:

- регистрового файла;
- счетчика указателя стека;
- счетчика номера первого свободного регистра;
- декодера выборки регистра;
- схемы контроля переполнения;
- схемы контроля опустошения.
- шинного формирователя

Назначение линий в группах C,U и V поясняются в таблице:

Сигнал	Назначение
<i>Командное слово C</i>	
WE	Тип операции (чтение, запись)
CS	Сигнал выборки кристалла, инициализирующий операции
CKE	Сигнал разрешения синхросигнала
CLK	Сигнал синхронизации
<i>Осведомительные сигналы U</i>	
UOVF	Стек полон
UEMP	Стек пуст
<i>Сигналы управления V</i>	
VWR	Разрешение записи данных
VWE	Тип операции
VPUSH	Строб увеличения указателя стека
VPOP	Строб уменьшения указателя стека
VLRSEL	Сигнал разрешения выборки регистра
VOE	Сигнал управления шинным формирователем
OVF	Сигнал уведомления об ошибке

Содержание отчета

- Отчет о выполнении домашнего задания, содержащий:
 1. Исходное задание для проведения лабораторной работы.

2. Схема алгоритма работы устройства.
 3. Диаграмма переходов состояний управляющего автомата.
 4. Листинг VHDL описания управляющего автомата.
- Диаграмму функционального тестирования автомата.
 - Таблица с результатами тестирования макета устройства.

Контрольные вопросы

1. Назовите этапы проектирования цифровых устройств с использованием ПЛИС.
2. Какие способы описания устройств используются в САПР Xilinx ISE 9.1.
3. Перечислите стадии конструкторско-технологического этапа проектирования с использованием ПЛИС.
4. На каких стадиях проектирования с использованием ПЛИС используется моделирование.

Работа №3. ОРГАНИЗАЦИЯ ПАМЯТИ КОНВЕЙЕРНЫХ СУПЕРСКАЛЯРНЫХ ЭЛЕКТРОННЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИН

Цель работы - освоение принципов эффективного использования подсистемы памяти современных универсальных ЭВМ, обеспечивающей хранение и своевременную выдачу команд и данных в центральное процессорное устройство. Работа проводится с использованием программы для сбора и анализа производительности PCLAB.

В ходе работы необходимо ознакомиться с теоретическим материалом, касающимся особенностей функционирования подсистемы памяти современных конвейерных суперскалярных ЭВМ, изучить возможности программы PCLAB, изучить средства идентификации микропроцессоров, провести исследования времени выполнения тестовых программ, сделать выводы о архитектурных особенностях используемых ЭВМ.

Порядок выполнению лабораторной работы

Задание 1. Ознакомиться с возможностями программы PCLAB в Разделе 2 методических указаний. Запустить программу PCLAB 1.0. Изучить идентификационную информацию на вкладке «Идентификация процессора».

Задание 2. На основании идентификационной информации о микропроцессоре ЭВМ, используемой при проведении лабораторной работы, определить следующие параметры: размер линейки кэш-памяти верхнего уровня и объем физической памяти. Результаты занести в отчет.

Задание 3. Ознакомиться с описанием эксперимента «Исследование расслоения динамической памяти» на вкладке «Описание эксперимента». Провести эксперимент. По результатам эксперимента определить: количество банков динамической памяти; размер одной страницы динамической памяти; количество страниц в динамической памяти. Сделать выводы о использованном способе наращивания динамической памяти. Результаты занести в отчет.

Задание 4. Ознакомиться с описанием эксперимента «Сравнение эффективности ссылочных и векторных структур данных». Провести эксперимент. По результатам эксперимента определить: отношение времени работы алгоритма, использующего зависимые данные, ко времени обработки аналогичного алгоритма обработки независимых данных. Сделать выводы об эффективности ссылочных и векторных структур данных и способах ее повышения. Результаты занести в отчет.

Задание 5. Для ЭВМ, используемой при проведении лабораторной работы определить следующие параметры: степень ассоциативности и размер TLB данных. Ознакомиться с описанием и провести эксперимент «Исследование эффективности программной предвыборки». По результатам эксперимента определить: отношение времени последовательной обработки блока данных ко времени обработки блока с применением предвыборки; время и количество тактов первого обращения к странице данных. Сделать выводы об эффективности предвыборки и способах ее повышения. Результаты занести в отчет.

Задание 6. Ознакомиться с описанием и провести эксперимент «Исследование способов эффективного чтения оперативной памяти». По

результатам эксперимента определить: отношение времени обработки блока памяти неоптимизированной структуры ко времени обработки блока структуры, обеспечивающей эффективную загрузку и параллельную обработку данных. Сделать выводы о способах повышения эффективности чтения оперативной памяти.

Задание 7. Для ЭВМ, используемой при проведении лабораторной работы определить следующие параметры: размер банка кэш-памяти данных первого и второго уровня, степень ассоциативности кэш-памяти первого и второго уровня, размер линейки кэш-памяти первого и второго уровня. Ознакомиться с описанием и провести эксперимент «Исследование конфликтов в кэш-памяти». По результатам эксперимента определить: отношение времени обработки массива с конфликтами в кэш-памяти ко времени обработки массива без конфликтов. Сделать выводы о способах устранения конфликтов в кэш-памяти.

Задание 8. Ознакомиться с описанием и провести эксперимент «Исследование алгоритмов сортировки». По результатам эксперимента определить: отношение времени сортировки массивов алгоритмом QuickSort ко времени сортировки алгоритмом Counting-Radix, а также ко времени сортировки Counting-Radix алгоритмом, оптимизированным под 8-процессорную вычислительную систему. Сделать выводы о наиболее эффективном алгоритме сортировки.

Применение программы PCLAB при исследовании производительности ЭВМ

Программа PCLAB предназначена для исследования производительности x86 совместимых ЭВМ с IA32 архитектурой, работающих под управлением операционной системы Windows (версий 95 и старше). Исследование организации ЭВМ заключается в проведении ряда экспериментов, направленных на построение зависимостей времени обработки критических участков кода от изменяемых параметров. Набор реализуемых программой экспериментов позволяет исследовать особенности построения современных

подсистем памяти ЭВМ и процессорных устройств, выявить конструктивные параметры конкретных моделей ЭВМ.

Процесс сбора и анализа экспериментальных данных в PCLAB основан на процедуре профилировки критического кода, т.е. в измерении времени его обработки центральным процессорным устройством. При исследовании конвейерных суперскалярных процессорных устройств, таких как 32-х разрядные процессоры фирмы Intel или AMD, способных выполнять переупорядоченную обработку последовательности команд программы, требуется использовать специальные средства измерения временных интервалов и запрещения переупорядочивания микрокоманд.

Для измерения времени работы циклов в PCLAB используется методика, описанная в [2]:

- Длительность обработки участка профилируемой программы характеризуется изменением величины счетчика тактов процессора, произошедшим за время его работы.
- Для предотвращения влияния соседних участков кода на результаты измерений, перед началом замера и после его окончания необходимо выдать команду упорядоченного выполнения CPUID, препятствующую переупорядочиванию потока команд на конвейере процессора.
- Замеры количества тактов процессора необходимо повторить несколько раз.
- Взаимное влияние последовательных повторов экспериментального участка программы исключается благодаря очищению кэш-памяти и буферов процессора.
- Часть граничных результатов отбрасывается (как наибольших, так и наименьших). По оставшимся результатам замеров определяется средняя величина.

На рисунке 8 показан внешний вид программы PCLAB 1.0.

Пользователю представляется многооконный интерфейс со свободной навигацией. Основное окно содержит главное меню и кнопки быстрого доступа к функциям программы. При запуске программы автоматически открывается окно «Описание лабораторной работы», содержащее две вкладки: «Методические указания» и «Идентификация процессора». На вкладке

«Методические указания» пользователь может ознакомиться с теоретическим материалом, порядком выполнения лабораторной работы, содержанием отчета и контрольными вопросами. Вкладка «Идентификация процессора» позволяет ознакомиться с результатами выполнения команды идентификации процессора CPUID и с их расшифровкой. Данная вкладка позволяет определить следующие архитектурные параметры исследуемой ЭВМ:

1. Идентификационную информацию о фирме-производителе, номере модели и версии процессора.
2. Набор поддерживаемых команд.
3. Наличие специализированных подсистем, таких как: поддержка машинного контроля, автоматического мониторинга температуры, Hyper-Threading и других.
4. Информацию о размере и ассоциативности кэш-памяти всех уровней.
5. Информацию о размерах буферов быстрого страничного преобразования.
6. Некоторую дополнительную информацию.

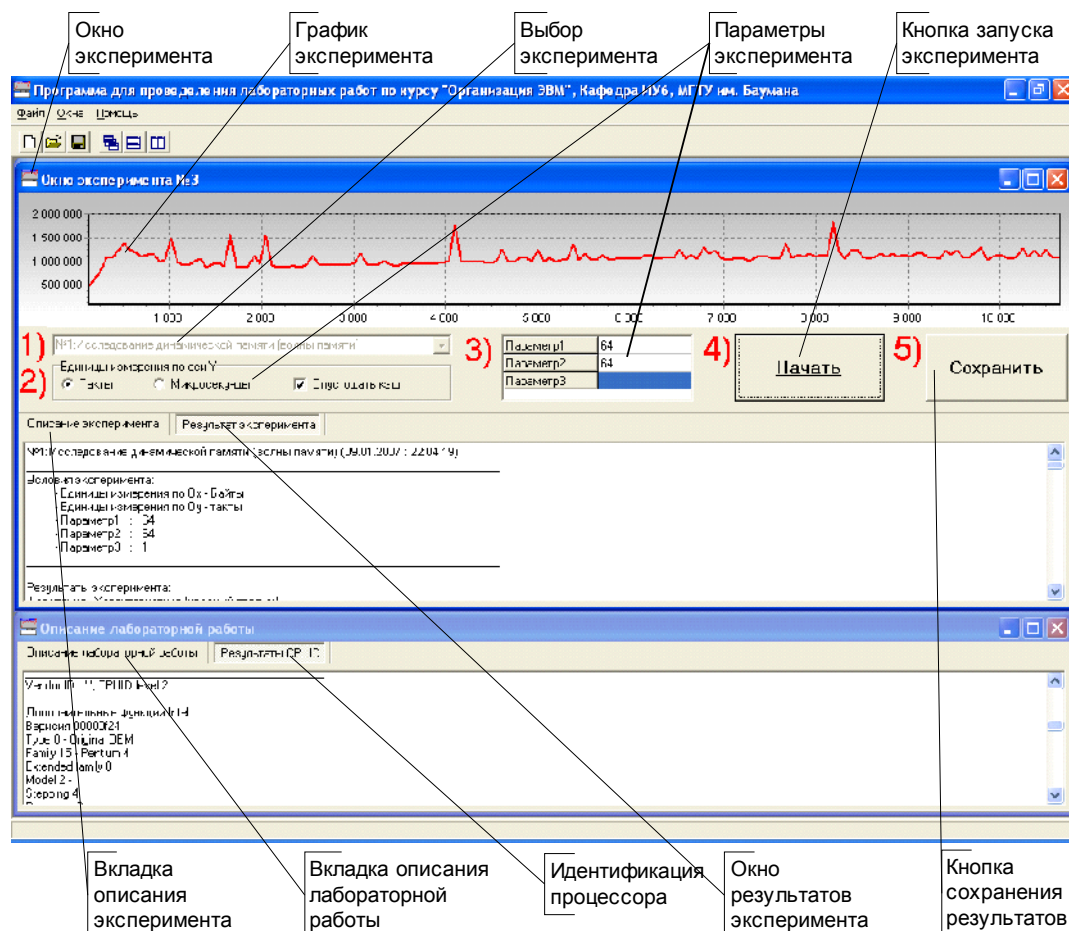


Рисунок 8 – Внешний вид программы PCLAB

Несмотря на большое количество выдаваемых процессором данных, для полной его идентификации и определения всех необходимых параметров оказывается достаточным определить информацию, относящуюся к первому пункту.

Для проведения экспериментов пользователю необходимо в меню «Файл» выбрать пункт «Новый эксперимент». После этого будет открыто новое дочернее окно эксперимента, в котором пользователю будет предложено выбрать один из доступных экспериментов.

После данного действия в нижней части окна появляется вкладка с описанием выбранного эксперимента, в которой содержится следующая информация:

1. Название и назначение эксперимента.
2. Список настраиваемых параметров эксперимента и единицы их измерения.

3. Допустимые диапазоны изменения настраиваемых параметров эксперимента.
4. Дополнительные указания по проведению и анализу полученных экспериментальных зависимостей.

После задания настраиваемых параметров пользователь может начать эксперимент по кнопке «Начать». В итоге будет построен экспериментально-полученная зависимость, характеризующая конструктивные параметры исследуемой ЭВМ. Численные результаты в виде читаемого текста доступны на вкладке «Результаты эксперимента».

После проведения эксперимента пользователь может сохранить числовые и графические данные, нажав на кнопку «Сохранить». При этом будет открыт стандартный диалог сохранения, в котором представляется возможность указать путь и название файлов типа *.bmp и *.rtf, в которые будут сохранены графические и текстовые результаты.

Описание эксперимента «Исследования расслоения динамической памяти»

Цель эксперимента: определение способа трансляции физического адреса, используемого при обращении к динамической памяти.

Исходные данные: размер линейки кэш-памяти верхнего уровня; объем физической памяти.

Результаты эксперимента: количество банков динамической памяти; размер одной страницы динамической памяти; количество страниц в динамической памяти.

Описание проблемы. В связи с конструктивной неоднородностью оперативной памяти, обращение к последовательно расположенным данным требует различного времени. В связи с этим, для создания эффективных программ необходимо учитывать расслоение памяти, характеризуемое способом трансляции физического адреса.

Суть эксперимента. Для определения способа трансляции физического адреса при формировании сигналов выборки банка, выборки строки и столбца запоминающего массива применяется процедура замера времени обращения к динамической памяти по последовательным адресам с изменяющимся шагом

чтения. Для сравнения времен используется обращение к одинаковому количеству различных ячеек, отстоящих друг от друга на определенный шаг. Результат эксперимента представляется зависимостью времени (или количества тактов процессора), потраченного на чтение ячеек, от шага чтения.

Для проведения эксперимента необходимо задать изменяемые параметры:

Параметр	Диапазон	Масштаб	Описание
№ 1	1..128	К	Максимальное расстояния между читаемыми блоками
№ 2	4..64	Б	Шаг увеличения расстояния между читаемыми 4-х байтовыми ячейками.
№ 3	1..16	М	Размер массива

Код профилируемой программы на языке С представлен ниже.

```
// ВЫДЕЛЕНИЕ ПАМЯТИ
p = (int*)_malloc64(Param_[3]);      // АДРЕС КРАТЕН 64

for (int pg_size = Param_[2]; pg_size <= Param_[1]; pg_size += Param_[2])
{
    Start_Count(); // Начало замера времени
    volatile int x = 0;
    for (int b = 0; b < pg_size; b += Param_[2])
        for (int a = b; a < Param_[3]; a += pg_size)
            x += *(int *) (int(p) + a);
    Finish_Count(); // Конец замера времени
    // Результат:
    // По оси X: pg_size
    // По оси Y: Время (Количество тактов)
}
```

Пример полученного графика показан на рисунке 9.

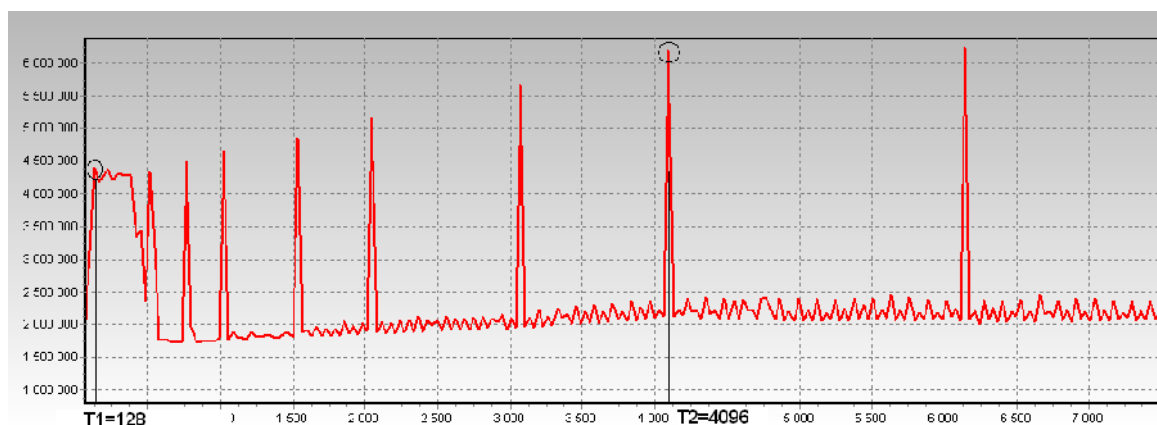


Рисунок 9 – Пример результата исследования расслоения динамической памяти

График показывает время или количество тактов работы алгоритма. Ось абсцисс отражает шаг приращения адреса читаемых данных. Ось ординат

отображает время в микросекундах или количество тактов (в зависимости от заданного параметра «Единицы измерения по оси Y»).

По графику можно определить следующие параметры:

1. Минимальный шаг чтения динамической памяти, при котором происходит постоянное обращение к одному и тому же банку. При наличии нескольких банков памяти данный параметр соответствует первому локальному экстремуму полученной функции (точка T1). Отсутствие характерного ступенеобразного графика говорит об одном независимом банке динамической памяти. По полученному значению шага T1 можно определить количество банков памяти: $B = T1/\Pi$, где Π – объем данных, являющийся минимальной порцией обмена кэш-памяти верхнего уровня с оперативной памятью и соответствует размеру линейки кэш-памяти верхнего уровня.
2. При достижении глобального экстремума, после которого рост локальных экстремумов не происходит, определяется характерная точка T2. Соответствующий данной точке шаг чтения является наихудшим при обращении к динамической памяти, т.к. приводит к постоянному закрытию и открытию страниц динамической памяти. Таким образом шаг T2 соответствует расстоянию (в байтах) между началом двух последовательных страниц одного банка. Зная количество банков, определяем размер одной страницы: $PC = T2/B$.
3. Зная параметры PC и B, а также полный объем памяти O определяем количество страниц физической оперативной памяти: $C = O/(PC*B*\Pi)$.

Описание эксперимента «Сравнение эффективности ссылочных и векторных структур»

Цель эксперимента: оценка влияния зависимости команд по данным на эффективность вычислений.

Результаты эксперимента: отношение времени работы алгоритма, использующего зависимые данные, ко времени обработки аналогичного алгоритма обработки независимых данных.

Описание проблемы. Обработка зависимых данных происходит в тех случаях, когда результат работы одной команды используется в качестве адреса

операнда другой. При программировании на языках высокого уровня такими операндами являются указатели, активно используемые при обработке ссылочных структур данных: списков, деревьев, графов. Обработка данных структур процессорами с длинными конвейерами команд приводит к заметному увеличению времени работы алгоритмов: адрес загружаемого операнда становится известным только после обработки предыдущей команды. В противоположность этому, обработка векторных структур, таких как массивы, позволяет эффективно использовать аппаратные возможности ЭВМ.

Суть эксперимента. Для сравнения эффективности векторных и списковых структур в эксперименте применяется профилировка кода двух алгоритмов поиска минимального значения. Первый алгоритм использует для хранения данных список, в то время как во втором применяется массив. Очевидно, что время работы алгоритма поиска минимального значения в списке зависит от его фрагментации, т.е. от среднего расстояния между элементами списка. Таким образом для проведения эксперимента требуется задать следующие настраиваемые параметры:

Параметр	Диапазон	Масштаб	Описание
№ 1	1..20	М	Количество элементов в списке
№ 2	4..500	К	Максимальная фрагментации списка
№ 3	1..10	К	Шаг увеличения фрагментации

Код профилируемой программы на языке С представлен ниже.

```
// ВЫДЕЛЕНИЕ ПАМЯТИ
struct list{
    struct list    *next;
    int            val;
};
int a,b,cur,prev,FF,max_list,max_arr;
struct list *list_,*list_t;
int *arr;
// АДРЕС КРАТЕН 64
list_ = (struct list*) _malloc64(Param_[1]*sizeof(struct list));
arr = (int*) _malloc64(Param_[1]*sizeof(int));

// СПИСОК

for (FF = 1; FF <= Param_[2]+1; FF=FF+Param_[3])
{
    for (a = 0; a < Param_[1]; a++)
    {
        list_[a].next = 0;
        list_[a].val = 0;
    };
    prev = 0;
```

```

    for (a = 0; a < Param_[1]; a++)
    {
        cur = prev + FF;
        if (cur>=Param_[1]) {cur=cur-Param_[1];};
        while (list_[cur].next)
        {
            if (++cur>=Param_[1]) {cur=cur-Param_[1];};
        };
        list_[prev].next = list_ + cur;
        list_[prev].val = a;
        prev = cur;
    }
    list_[cur].next = 0;
    list_[cur].val = a;
    Start_Count(); // Начало замера времени
    max_list=list_[0].val;
    list_t=list_;
    while (list_t = list_t[0].next)
    {
        if (max_list<list_t[0].val)
        {
            max_list=list_t[0].val;
        }
    };
    Finish_Count();// Конец замера времени
    // Результат:
        // По оси X: FF
        // По оси Y: Время (Количество тактов)

// МАССИВ
    for (a = 0; a < Param_[1]; a++)
    {
        arr[a]= a;
    }
    max_arr=arr[0];
    Start_Count(); // Начало замера времени
    for (a = 0; a < Param_[1]; a++)
    {
        if (max_arr<arr[a])
        max_arr=arr[a];
    };
    Finish_Count();// Конец замера времени
    // Результат:
        // По оси X: FF
        // По оси Y: Время (Количество тактов)

```

Пример полученного графика показан на рисунке 10.

Красный график (верхний) показывает время или количество тактов работы алгоритма, использующего список. Зеленый график (нижний) показывает время или количество тактов работы алгоритма, использующего массив. Ось абсцисс отражает фрагментацию списка. Ось ординат отображает время в микросекундах или количество тактов (в зависимости от заданного параметра «Единицы измерения по оси Y»).

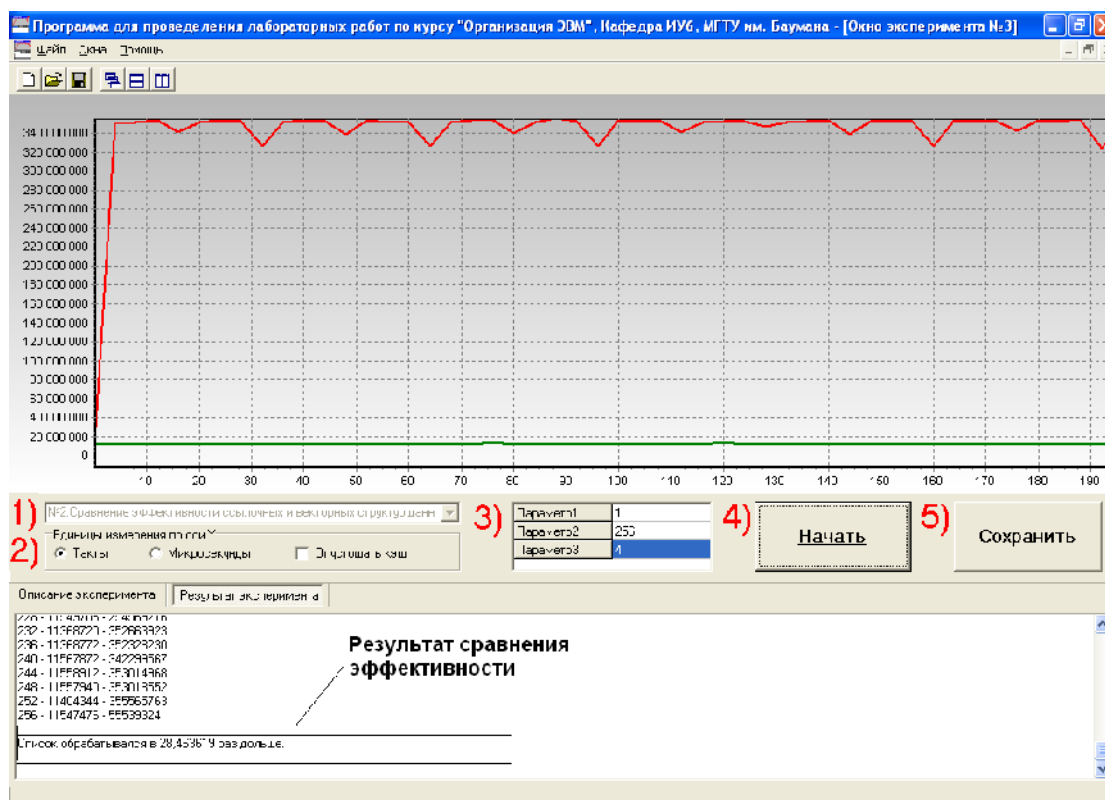


Рисунок 10 – Пример исследования эффективности ссылочных и векторных структур

После проведения эксперимента на вкладке «Результаты эксперимента» отображается отношение суммарного времени обработки списков с различной степенью фрагментации к суммарному времени обработки массивов.

Описание эксперимента «Исследование эффективности программной предвыборки»

Цель эксперимента: выявление способов ускорения вычислений благодаря применению предвыборки данных.

Исходные данные: степень ассоциативности и размер TLB данных.

Результаты эксперимента: отношение времени последовательной обработки блока данных ко времени обработки блока с применением предвыборки; время и количество тактов первого обращения к странице данных.

Описание проблемы. Обработка больших массивов информации сопряжена с открытием большого количества физических страниц памяти. При первом обращении к странице памяти наблюдается увеличенное время доступа к данным. Это связано с необходимостью преобразования логического адреса в

физический адрес памяти, а также с открытием страницы динамической памяти и сохранения данных в кэш-памяти. Преобразование выполняется на основе информации о использованных ранее страницах, содержащейся в TLB буфере процессора. Первое обращение к странице при отсутствии информации в TLB вызывает двойное обращение к оперативной памяти: сначала за информацией из таблицы страниц, а далее за востребованными данными. Предвыборка заключается в заблаговременном проведении всех указанных действий благодаря дополнительному запросу небольшого количества данных из оперативной памяти.

Суть эксперимента. Эксперимент основан на замере времени двух вариантов подпрограмм последовательного чтения страниц оперативной памяти. В первом варианте выполняется последовательное чтение без дополнительной оптимизации, что приводит к дополнительным двойным обращениям. Во втором варианте перед циклом чтения страниц используется дополнительный цикл предвыборки, обеспечивающий своевременную загрузку информации в TLB данных.

Для проведения эксперимента необходимо задать изменяемые параметры:

Параметр	Диапазон	Масштаб	Описание
№ 1	1..4096	Б	Шаг увеличения расстояния между читаемыми данными
№ 2	4..8192	К	Размер массива

Сокращение времени работы алгоритма, использующего предвыборку происходит в том случае, когда информация о востребованных страницах умещается в TLB. Поэтому необходимо определить размер и степень ассоциативности TLB и учитывать ее в алгоритме. Код профилируемой программы на языке С представлен ниже.

```
// ВЫДЕЛЕНИЕ ПАМЯТИ
p = (int*)_malloc64(Param_[2]);          // АДРЕС КРАТЕН 64

// БЕЗ ПРЕДВЫБОРКИ
for (int a = 0; a < Param_[2]; a += Param_[1])
{
    Start_Count(); // Начало замера времени
    x += *(int *) (int(p) + a);
    Finish_Count(); // Конец замера времени
    // Результат:
    // По оси X: a
```

```

        // По оси Y: Время (Количество тактов)
    }

    // С ПРЕДВЫБОРКОЙ
    for (int i = 0; i < Param_[2]; i += 4*K) x += *(int *)((int)p + i);
    // Считаем, что вся информация поместится в TLB
    for (int a = 0; a < Param_[2]; a += Param_[1])
    {
        Start_Count(); // Начало замера времени
        x += *(int *) (int(p) + a);
        Finish_Count(); // Конец замера времени
        // Результат:
        // По оси X: a
        // По оси Y: Время (Количество тактов)
    }
}

```

Пример полученного графика показан на рисунке 11.

Красный график (верхний с острыми пиками) показывает время или количество тактов работы алгоритма без предвыборки. Зеленый график (нижний без значимых пиков) показывает время или количество тактов работы алгоритма с использованием предвыборки. Ось абсцисс отражает смещение читаемых данных от начала блока. Ось ординат отображает время в микросекундах или количество тактов (в зависимости от заданного параметра «Единицы измерения по оси Y»).

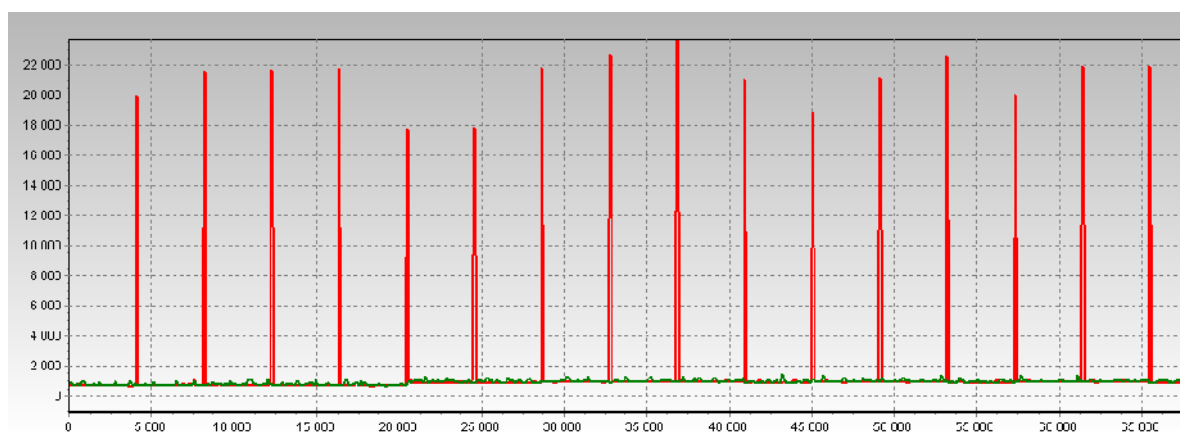


Рисунок 11 – Пример исследования эффективности предвыборки

По окончании эксперимента на вкладке «Результаты эксперимента» отображается информация об эффективности предвыборки. Например: «Обработка без загрузки таблицы страниц в TLB производилась в 1,54 раза дольше».

Конкретные значения задержек, характерных для первого обращения к странице, определяются по числовым результатам экспериментов на вкладке «Результаты эксперимента».

Описание эксперимента «Исследование способов эффективного чтения оперативной памяти»

Цель эксперимента: исследование возможности ускорения вычислений благодаря использованию структур данных, оптимизирующих механизм чтения оперативной памяти.

Исходные данные: Адресное расстояние между банками памяти, размер буфера чтения.

Результаты эксперимента: отношение времени обработки блока памяти неоптимизированной структуры ко времени обработки блока структуры, обеспечивающей эффективную загрузку и параллельную обработку данных.

Описание проблемы. При обработке информации, находящейся в нескольких страницах и банках оперативной памяти возникают задержки, связанные с необходимостью открытия и закрытия страниц DRAM памяти. При программировании на языках высокого уровня такая ситуация наблюдается при интенсивной обработке нескольких массивов данных или обработке многомерных массивов. При этом процессоры, в которых реализованы механизмы аппаратной предвыборки, часто не могут организовать эффективную загрузку данных. Кроме этого, объемы запрошенных данных оказываются заметно меньше размера пакета, передаваемого из оперативной памяти. Таким образом, эффективная обработка нескольких векторных структур данных без их дополнительной оптимизации не использует в должной степени возможности аппаратных ресурсов.

Для создания структур данных, оптимизирующих их обработку современными процессорами, требуется максимально исключить несвоевременную передачу данных, т.е. передавать в каждом пакете только востребованную для вычислений информацию. В результате такой оптимизации снижается количество кэш-промахов, сокращается количество открытий и закрытий страниц DRAM-памяти, обеспечивается параллельная обработка данных и выполнение операций загрузки и выгрузки.

Суть эксперимента. Для сравнения производительности алгоритмов, использующих оптимизированные и неоптимизированные структуры данных используется профилировка кода двух подпрограмм, каждая из которых должна выполнить обработку нескольких блоков оперативной памяти. В алгоритмах обрабатываются двойные слова данных (4 байта), что существенно меньше размера пакета (32 – 128 байт). Неоптимизированный вариант структуры данных представляет собой несколько массивов в оперативной памяти, в то время как оптимизированная структура состоит из чередующихся данных каждого массива.

Для проведения эксперимента необходимо задать изменяемые параметры:

Параметр	Диапазон	Масштаб	Описание
№ 1	1..4	M	Размер массива
№ 2	1..128	1	Количество потоков данных

Код профилируемой программы на языке C представлен ниже.

```
// ВЫДЕЛЕНИЕ ПАМЯТИ
int *p, *px[32];
for (a = 0; a < Param_[1]; a++)
    px[a] = (int *) _malloc64(Param_[1]);           // АДРЕС КРАТЕН 64
p = (int*) _malloc64(Param_[1]*Param_[2]);        // АДРЕС КРАТЕН 64

// НЕСКОЛЬКО МАССИВОВ
for (r=1; r <= Param_[2]; r++)
{
    x=0;
    Start_Count(); // Начало замера времени
    for (int a = 0; a < Param_[1]; a += sizeof(int))
        for(b=0; b < r; b++)
            x += *(int *)((int)px[b] + a );
    Finish_Count(); // Конец замера времени
    // Результат:
    // По оси X: r
    // По оси Y: Время (Количество тактов)
}

// ОДИН ОПТИМИЗИРОВАННЫЙ МАССИВ
for (r=1; r <= Param_[2]; r++)
{
    x=0;
    Start_Count(); // Начало замера времени
    for (int a = 0; a < Param_[1]*r; a += (sizeof(int)*r))
        for(b=0; b < r; b++)
            x += *(int *)((int)p + a + b*sizeof(int));
    Finish_Count(); // Конец замера времени
    // Результат:
    // По оси X: r
    // По оси Y: Время (Количество тактов)
```

}

Пример полученного графика показан на рисунке 12.

Красный график (верхний) показывает время или количество тактов работы алгоритма, использующего неоптимизированную структуру. Зеленый график (нижний) показывает время (или количество тактов) работы алгоритма с использованием оптимизированной структуры. Ось абсцисс отражает количество одновременно обрабатываемых массивов. Ось ординат отображает время в микросекундах или количество тактов (в зависимости от заданного параметра «Единицы измерения по оси Y»).

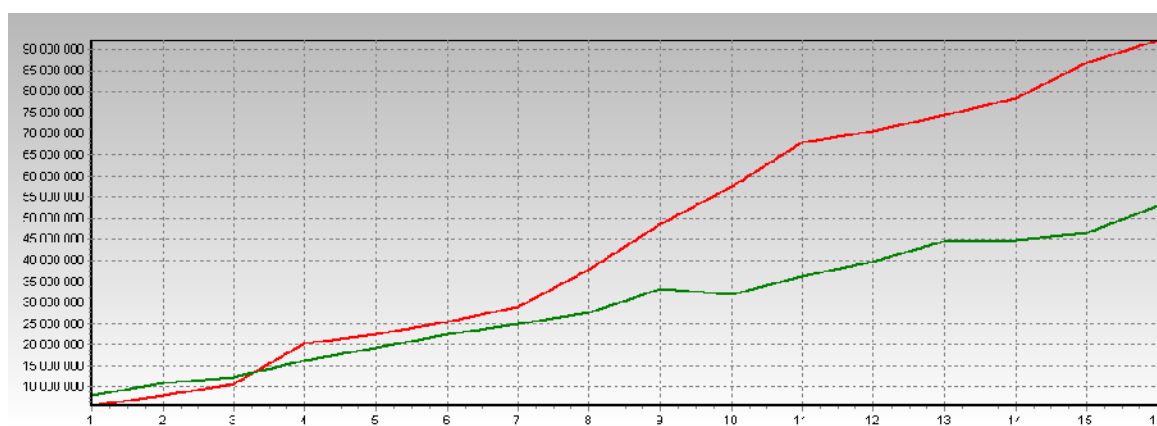


Рисунок 12 – Пример исследования оптимизирующих структур данных

По окончании эксперимента на вкладке «Результаты эксперимента» отображается информация об эффективности использования оптимизирующих структур данных. Например: «Неоптимизированная структура обрабатывалась в 5,14 раза дольше».

Описание эксперимента «Исследование конфликтов в кэш-памяти»

Цель эксперимента: исследование влияния конфликтов кэш-памяти на эффективность вычислений.

Исходные данные: Размер банка кэш-памяти данных первого и второго уровня, степень ассоциативности кэш-памяти первого и второго уровня, размер линейки кэш-памяти первого и второго уровня.

Результаты эксперимента: отношение времени обработки массива с конфликтами в кэш-памяти ко времени обработки массива без конфликтов.

Описание проблемы. Наборно-ассоциативная кэш-память состоит из линеек данных, организованных в несколько независимых банков. Выбор банка для каждой порции кэшируемых данных выполняется по ассоциативному принципу, т.е. из условия улучшения представительности выборки, в то время как целевая линейка в каждом из банков жестко определяется по младшей части физического адреса. Совокупность таких линеек всех банков принято называть набором. Таким образом, попытка читать данные из оперативной памяти с шагом, кратным размеру банка, приводит к их помещению в один и тот же набор. Если же количество запросов превосходит степень ассоциативности кэш-памяти, т.е. количество банков или количество линеек в наборе, то наблюдается постоянное вытеснение данных из кэш-памяти, причем больший ее объем остается незадействованным.

Суть эксперимента. Для определения степени влияния конфликтов в кэш-памяти на эффективность вычислений используется профилировка двух процедур чтения и обработки данных. Первая процедура построена таким образом, что чтение данных выполняется с шагом, кратным размеру банка. Это порождает постоянные конфликты в кэш-памяти. Вторая процедура оптимизирует размещение данных в кэш с помощью задания смещения востребованных данных на некоторый шаг, достаточный для выбора другого набора. Этот шаг соответствует размеру линейки.

Для проведения эксперимента необходимо задать изменяемые параметры:

Параметр	Диапазон	Масштаб	Описание
№ 1	1..256	К	Размер банка кэш-памяти
№ 2	1..128	б	Размер линейки кэш-памяти
№ 3	2..512	1	Количество читаемых линеек

Код профилируемой программы на языке С представлен ниже.

```
// ВЫДЕЛЕНИЕ ПАМЯТИ
// АДРЕС КРАТЕН 64
p = (int *)_malloc64((Param_[1]+Param_[2])*Param_[3]);

// ЧТЕНИЕ КЭШ-ПАМЯТИ С КОНФЛИКТАМИ
for(int a=0; a < Param_[3]; a++)
```

```

    {
        Start_Count(); // Начало замера времени
        x+=*(int *)((int)p + a*Param_[1]);
        Finish_Count(); // Конец замера времени
        // Результат:
        // По оси X: a*Param_[1]
        // По оси Y: Время (Количество тактов)
    }
// ЧТЕНИЕ КЭШ-ПАМЯТИ БЕЗ КОНФЛИКТОВ

for(int a=0; a < Param_[3]; a++)
{
    Start_Count(); // Начало замера времени
    x+=*(int *)((int)p + a*(Param_[1]+Param_[2]));
    Finish_Count(); // Конец замера времени
    // Результат:
    // По оси X: a*(Param_[1]+Param_[2])
    // По оси Y: Время (Количество тактов)
}

```

Пример полученного графика показан на рисунке 13.

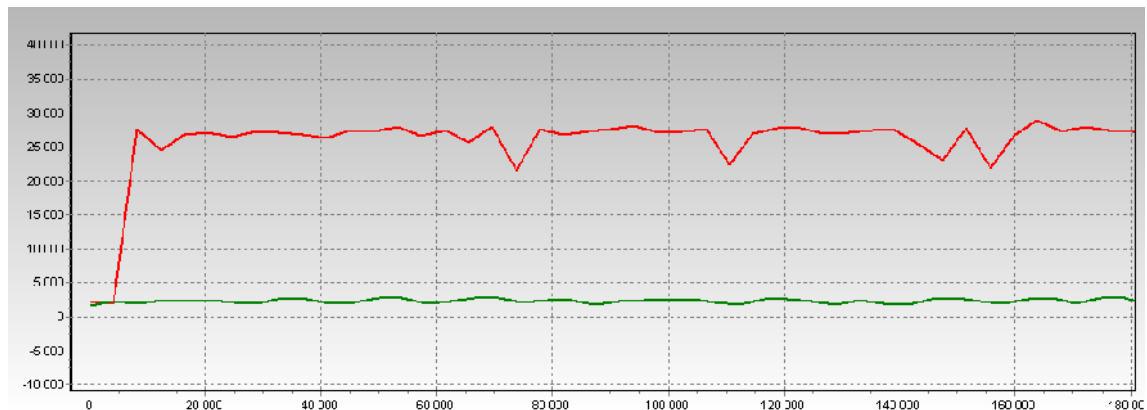


Рисунок 13 – Пример исследования конфликтов в кэш-памяти

Красный график (верхний) показывает время или количество тактов работы процедуры, читающей данные с конфликтами в кэш-памяти. Зеленый график (нижний) показывает время или количество тактов работы процедуры, не вызывающей конфликтов в кэш-памяти. Ось абсцисс отражает смещение читаемой ячейки от начала блока данных. Ось ординат отображает время в микросекундах или количество тактов (в зависимости от заданного параметра «Единицы измерения по оси Y»).

По окончании эксперимента на вкладке «Результаты эксперимента» отображается информация об эффективности чтения без конфликтов кэш-памяти. Например: «Чтение с конфликтом банков производилось в 12,8 раза дольше».

Описание эксперимента «Сравнение алгоритмов сортировки»

Цель эксперимента: исследование способов эффективного использования памяти и выявление наиболее эффективных алгоритмов сортировки, применимых в вычислительных системах.

Исходные данные: количество процессоров вычислительной системы, размер пакета, количество элементов в массиве, разрядность элементов массива.

Результаты эксперимента: отношение времени сортировки массива алгоритмом QuickSort ко времени сортировки алгоритмом Radix-Counting Sort и ко времени сортировки Radix-Counting Sort, оптимизированной под 8-процессорную вычислительную систему.

Описание проблемы. Существует несколько десятков алгоритмов сортировки. Их можно классифицировать по таким критериям, как: назначение (внутренняя и внешняя сортировки), вычислительная сложность (алгоритмы с вычислительными сложностями $O(n^2)$, $O(n \cdot \log(n))$, $O(n)$, $O(n/\log(n))$), емкостная сложность (алгоритмы, требующие и не требующие дополнительного массива), возможность распараллеливания (не распараллеливаемые, ограниченно распараллеливаемые, полностью распараллеливаемые), принцип определения порядка (алгоритмы, использующие парные сравнения и не использующие парные сравнения).

Среди известных алгоритмов не выделено однозначного лидера, выполняющего сортировку чисел любой разрядности за минимальное время с минимальной емкостной сложностью. Считается [5,6], что хорошие результаты обеспечивает алгоритм внутренней сортировки QuickSort, имеющий вычислительную сложность в среднем $O(n \cdot \log(n))$ и $O(n^2)$ в худшем. Теоретически доказано, что алгоритмы, использующие парные сравнения не могут иметь вычислительную сложность меньше, чем $O(n \cdot \log(n))$.

Особый интерес представляют алгоритмы внутренней сортировки, не использующие парные сравнения: Counting Sort, Radix Sort и другие. В работе [4] предложен алгоритм, являющийся сочетанием алгоритмов Counting Sort (сортировка подсчетом) и Radix Sort (поразрядная сортировка), лишенный некоторых присущих им недостатков. В [4] получены оценки вычислительной сложности Radix-Counting алгоритма ($O(n/\log(n))$), т.е. она менее чем линейна.

Однако, вычислительная сложность алгоритма существенно зависит от его настройки на определенную разрядность чисел и размерность массивов. Для выполнения сортировки на многопроцессорных ЭВМ требуется использовать оптимизированную стратегию Radix-Counting сортировки, учитывающую количество процессоров, а также пакетный режим обмена процессора и оперативной памяти.

```
//Алгоритм QUICK-SORT
//Выделение памяти
QMAS = (unsigned __int64 *) _malloc128(Param_[0]*sizeof(unsigned __int64));
RMAS = (unsigned __int8 *) _malloc128(Param_[0]*sizeof(unsigned __int64));
TMP = (unsigned __int8 *) _malloc128(Param_[0]*sizeof(unsigned __int64));
TMP_A64 = (unsigned __int64 *)RMAS;
TMP_B64 = (unsigned __int64 *)QMAS;

for (b = Param_[1]; b <= Param_[0]; b += Param_[1])
{
    Start_Count(); // Начало замера времени
    QuickSort(QMAS,0,b-1);
    Finish_Count(); // Конец замера времени
    // Результат:
    // По оси X: b*Param_[1]
    // По оси Y: Время (Количество тактов)
}

void QuickSort (unsigned __int64* A, int iLo, int iHi)
{
    int Lo, Hi;
    unsigned __int64 Mid,T;
    Lo = iLo;
    Hi = iHi;
    Mid = A[(Lo + Hi)>>1];
    do
    {
        while (A[Lo] < Mid) {Lo++;};
        while (A[Hi] > Mid) {Hi--};
        if (Lo <= Hi)
        {
            T = A[Lo];
            A[Lo] = A[Hi];
            A[Hi] = T;
            Lo++;
            Hi--;
        }
    } while (Lo < Hi);
    if (Hi > iLo) {QuickSort(A, iLo, Hi);};
    if (Lo < iHi) {QuickSort(A, Lo, iHi);};
}

//Алгоритм RADIX-COUNTING (размерность всех групп разрядов: 8 бит).
for (b = Param_[1]; b <= Param_[0]; b += Param_[1])
{
    TMP_A64 = (unsigned __int64 *)RMAS;
    TMP_B64 = (unsigned __int64 *)TMP;
    Start_Count(); // Начало замера времени
    for (i=0; i<=7; i++) {
```

```

//Заполнить массив с нулями
    for (j=0; j<=255; j++) {c[j]=0;};
//Накопить в массиве С количества повторений разрядов
    for (j=0; j<b; j++) {c[RMAS[i+j*8]]++;};
//Сохранить в массиве С количество чисел, меньших данного
    for (j=1; j<=255; j++) {c[j] += c[j-1]};
    for (j=b-1; j>=0; j--) {
//Модифицировать массив С для данного значения разряда
        c[RMAS[i+j*8]]--;
//Записать число из исходного массива во временный массив
        temp = TMP_A64[j];
        TMP_B64[c[RMAS[i+j*8]]] = temp; }
//Подготовиться к сортировке по следующему разряду
    p=TMP;
    TMP=RMAS;
    RMAS=p;
    TMP_A64 = (unsigned __int64 *)RMAS;
    TMP_B64 = (unsigned __int64 *)TMP;
}
Finish_Count();// Конец замера времени
// Результат:
// По оси X: b*Param_[1]
// По оси Y: Время (Количество тактов)
}
//Алгоритм RADIX-COUNTING, оптимизированной под 8 процессоров
for (b = Param_[1]; b <= Param_[0]; b += Param_[1])
{
    TMP_A64 = (unsigned __int64 *)RMAS;
    TMP_B64 = (unsigned __int64 *)TMP;
    Start_Count(); // Начало замера времени
    for (i=0; i<256*8; i++) {copt[i]=0;};
    j=0;
    for (k=0; k<b; k++) {
        //Параллельное обращение к 8 частям сорт
        copt[RMAS[j+0]*8+0]++;
        copt[RMAS[j+1]*8+1]++;
        copt[RMAS[j+2]*8+2]++;
        copt[RMAS[j+3]*8+3]++;
        copt[RMAS[j+4]*8+4]++;
        copt[RMAS[j+5]*8+5]++;
        copt[RMAS[j+6]*8+6]++;
        copt[RMAS[j+7]*8+7]++;
        j+=8;
    }
    for (j=1; j<256; j++) {
        //Параллельное обращение к 8 частям сорт
        copt[j*8+0] += copt[j*8-8+0];
        copt[j*8+1] += copt[j*8-8+1];
        copt[j*8+2] += copt[j*8-8+2];
        copt[j*8+3] += copt[j*8-8+3];
        copt[j*8+4] += copt[j*8-8+4];
        copt[j*8+5] += copt[j*8-8+5];
        copt[j*8+6] += copt[j*8-8+6];
        copt[j*8+7] += copt[j*8-8+7];
    }
    for (i=0; i<=7; i++)
    {
        for (j=b-1; j>=0; j--)
        {

```

```

        ii=(RMAS[j*8+i])*8+i;
        jj=--copt[ii];
        temp = TMP_A64[j];
        TMP_B64[jj] = temp;
    }
    p=TMP;
    TMP=RMAS;
    RMAS=p;
    TMP_A64 = (unsigned __int64 *)RMAS;
    TMP_B64 = (unsigned __int64 *)TMP;
}
Finish_Count();// Конец замера времени
// Результат:
// По оси X: b*Param_[1]
// По оси Y: Время (Количество тактов)
}

```

Суть эксперимента. Эксперимент основан на замерах времени трех вариантов алгоритмов сортировки (Quick Sort, Radix-Counting Sort, Оптимизированный Radix-Counting Sort).

Для проведения эксперимента необходимо задать изменяемые параметры:

Параметр	Диапазон	Масштаб	Описание
№ 1	1..20	М	Количество 64-х разрядных элементов массивов
№ 2	4..1024	К	Шаг увеличения размера массива

Пример полученного графика показан на рисунке 14.

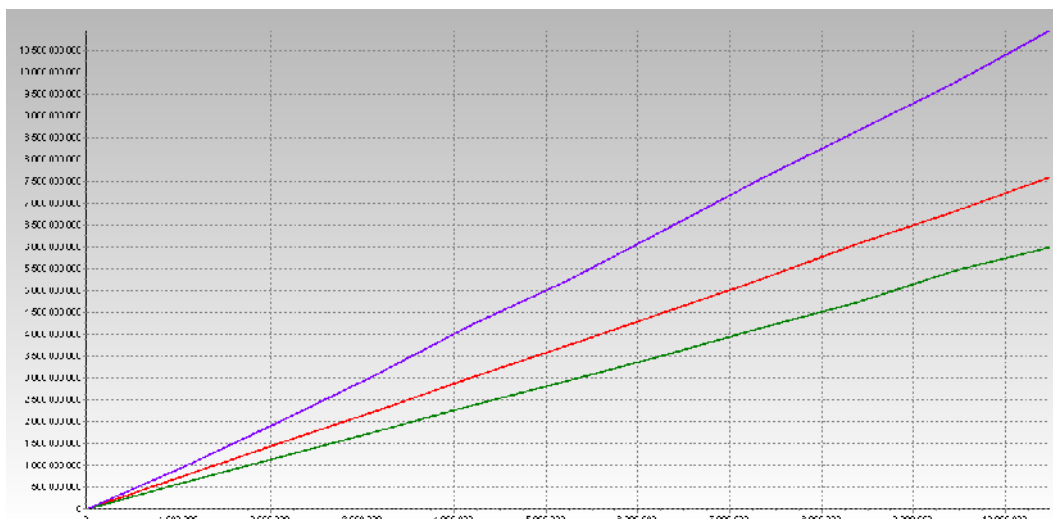


Рисунок 14 – Пример исследования алгоритмов сортировки

Фиолетовый график (верхний) показывает время или количество тактов работы алгоритма QuickSort. Красный график (средний) показывает время или

количество тактов работы неоптимизированного алгоритма Radix-Counting. Зеленый график (нижний) показывает время или количество тактов работы оптимизированного под 8-процессорную вычислительную систему алгоритма Radix-Counting. Ось абсцисс отражает количество 64-разрядных элементов сортируемых массивов. Ось ординат отображает время в микросекундах или количество тактов (в зависимости от заданного параметра «Единицы измерения по оси Y»).

По окончании эксперимента на вкладке «Результаты эксперимента» отображается информация об эффективности алгоритмов сортировки. Например: «QuickSort работал в 1,4161678 раз дольше Radix-Counting Sort. QuickSort работал в 1,7967811 раз дольше Radix-Counting Sort, оптимизированного под 8-процессорную ЭВМ.».

Содержание отчета

В отчете, помимо названия лабораторной работы, номера группы, фамилии и инициалов студента, должна содержаться следующая информация: для каждого проведенного эксперимента указываются исходные данные, настраиваемые параметры, графики полученных характеристик и результаты эксперимента. По каждому эксперименту должны быть сделаны выводы. В конце отчета приводится информация об идентификации процессора, использованного при проведении экспериментов.

Контрольные вопросы

1. Назовите причины расслоения оперативной памяти.
2. Как в современных процессорах реализована аппаратная предвыборка.
3. Какая информация храниться в TLB.
4. Какой тип ассоциативной памяти используется в кэш-памяти второго уровня современных ЭВМ и почему.
5. Приведите пример программной предвыборки.

Список литературы для лабораторных работ 1, 2

1. Угрюмов Е. П. Цифровая схемотехника: Учеб. Пособие для вузов. – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2004. – 800 с.: ил.
2. Грушвицкий Р. И., Мурсаев А. Х., Угрюмов Е. П. Проектирование систем на микросхемах с программируемой структурой, БХВ-Петербург, 2006, 708 с.
3. Spartan-3 FPGA Family: Complete Data Sheet. Xilinx Inc.
4. Xilinx ISE Guide (HTML Book). Xilinx Inc.
5. Xilinx ISE 9 Software Manuals
6. Spartan-3 Starter Kit Board User Guide
7. В. Зотов Инструментальный комплект Spartan3 Starter Kit

Список литературы для лабораторной работы 3

1. Цилькер Б.Я., Орлов С.А. Организация ЭВМ и систем: Учебник для вузов. – СПб.: Питер, 2004. – 668 с.: ил.
2. Касперски К. Техника оптимизации программ. Эффективное использование памяти. – СПб.: БХВ-Петербург, 2003. – 464 с.
3. IA-32 Intel® Architecture Software Developer's Manual
4. Кокотов В.З. Поразрядная сортировка менее чем линейной сложности. – Информационные технологии, 1998, №10, с. 14-21
5. Кнут Д. Искусство программирования. Т. 1.: Основные алгоритмы. 2-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2001. – 720 с.
6. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. – М.: МЦНМО, 2000. – 960 с.