СОДЕРЖАНИЕ

Перечень используемых сокращений:

АЛУ – Арифметико-логическое устройство
АО – Аппаратное обеспечение
БИС – Большая интегральная схема
ВА – Виртуальный адрес
ВК – Вычислительный комплекс
ВП – Внешняя память
ВС – Вычислительная система
ВТ – Вычислительная техника
ВУ – Внешнее устройство
ГСА – Граф-схема алгоритма
ГТИ – Генератор тактовых импульсов
ЖЛ – Жесткая логика
ЗУ – Запоминающее устройство
ЗЭ – Запоминающий элемент
ЗУПВ – Запоминающее устройство с произвольной выборкой
ИС(ИМС) – Интегральная микросхема
КПДП – Контроллер прямого доступа к памяти
ЛУ – Логическое условие
МК – Микрокоманда
МО – Микрооперация
МП – Микропрограмма
МПК – Микропроцессорный комплект
МПС – Микропроцессорная система
ОА – Операционный автомат
ОЗУ (RAM) – Оперативное запоминающее устройство
ОП – Оперативная память
ОУ – Операционное устройство
ПДП – Прямой доступ к памяти
ПЗС – Прибор с зарядовой связью
ПЗУ (ROM)– Постоянное запоминающее устройство
ПО – Программное обеспечение
ПУ – Периферийное устройство
РОН – Регистр общего назначения
СБИС – Сверхбольшая интегральная схема
СВМО – Среднее время между отказами
СОД – Система обработки данных
УА – Управляющий автомат
УУ – Устройство управления
ФА – Физический адрес
ЦП – Центральный процессор
ЭВМ – Электронная вычислительная машина
DRAM – Динамическое ЗУ с произвольным доступом
PBSRAM – Конвейерно-пакетная статическая память

RAID – Избыточный массив жестких дисков
SBSRAM – Синхронная пакетная статическая память
SDRAM – Синхронная динамическая память
SRAM – Статическое запоминающее устройство с произвольным доступом
UOP – Универсальная операция

Введение в курс «Структурная и функциональная организация ЭВМ»

1. Цели и задачи дисциплины.

Основная цель курса: «Структурная и функциональная организация ЭВМ»: обучение студентов принципам построения узлов и блоков ЭВМ, принципам проектирования вычислительных устройств и микропроцессорных схем, изучение влияния элементной базы на архитектуру компьютера (процессоры с фиксированной архитектурой и разрядно-модульной организацией связей).

В результате изучения дисциплины студенты должны:

знать:

- организацию и принципы построения основных блоков и узлов ЭВМ;
- методы построения ЭВМ;
- принципы работы микропроцессоров, микросхем ПЛИС.

уметь:

- правильно выбирать алгоритмы решения задач и исследование эффективности их реализации в проектируемом компьютере;
- разрабатывать структуры вычислительных устройств и их отдельных узлов и блоков;
- оптимизировать структуры компьютера с учетом переходных процессов в отдельных подсистемах;
- эмуляция узлов ЭВМ в микропроцессорных средах.

2. Структура дисциплины

Согласно учебному плану специальности 1-40.02.01 «Вычислительные машины, системы и сети» курс «Структурная и функциональная организация ЭВМ» изучается студентами на 4 и 5 курсах (8,9 семестры) и включает в себя следующие виды аудиторных занятий:

- 96 часов лекций;
- 16 часов практических занятий;
- 48 часов лабораторных работ.

Ниже представлено распределение курса по видам аудиторных занятий по разделам и темам:

ЛЕКЦИОННЫЙ КУРС (8 СЕМЕСТР)

Наименование разделов и тем лекций и их содержание	Количество часов
Введение в курс «Структурная и функциональная организация ЭВМ»	
Содержание дисциплины и её взаимосвязь с другими дисциплинами.	2
РАЗДЕЛ 1. Основные понятия и определения.	
1.1.Система обработки данных. ВК.	1
1.2.ВС. Структурная и функциональная организация ЭВМ.	1
РАЗДЕЛ 2. Принципы организации ЭВМ.	
2.1.Основные факторы, определяющие принципы организации ЭВМ	1
2.2.Архитектура ЭВМ по Дж. Фон Нейману	1
2.3.Гарвардская архитектура.	1
2.4.Состав устройств, структура и порядок функционирования ЭВМ.	1
2.5.Основные технические характеристики ВК.	1
РАЗДЕЛ 3. Память ЭВМ.	
3.1.Основные понятия. Организация и основные характеристики ЗУ.	1
3.2.Классификация ЗУ.	4
3.3.Иерархия памяти ЭВМ.	4
3.4.Организация ЗУ с произвольным доступом и сверх оперативных ЗУ.	4
3.4.1. Статические ЗУ с произвольным доступом.	4
3.4.2. Асинхронная динамическая память DRAM.	4
3.5.Синхронная динамическая память SDRAM.	6
3.5.1. Синхронная динамическая память DDR SDRAM.	1
3.6. Структура ЗУ с адресной организацией.	1
3.7. Кэш-память.	4
3.8.ПЗУ.	2
3.8.1.Разновидности ПЗУ.	2
3.8.2.Флэш-память.	2
3.9.Компоновка ОЗУ основных моделей СМ ЭВМ.	1
3.10. Регенерация информации в динамических ЗУ. Память RDRAM.	7
3.11.Пакетное ОЗУ SLDRAM.	1
3.12.ОЗУ с виртуальными каналами обмена.	1
3.13. Модули динамических ОЗУ.	2
3.14.Организация стековых ЗУ. ЗУ с ассоциативной организацией.	2
3.15.Организация КЭШ-памяти на основе АЗУ.	1
3.16.ЗУ с двумерной адресацией.	1

ЛЕКЦИОННЫЙ КУРС (9 СЕМЕСТР)

Наименование разделов и тем лекций и их содержание	Количество часов
3.17.Другие типы полупроводниковых ЗУ.	1
3.18.Организация дисковых массивов(RAID).	1
РАЗДЕЛ 4. Проектирование процессоров.	
4.1.Процессоры с фиксированной архитектурой	
4.1.1.Особенности организации современных процессоров.	1
4.1.2.Концепция открытых систем.	1
4.1.3.Структура суперскалярного процессора.	2
4.2.Особенности архитектуры микропроцессора P6 фирмы INTEL	4
4.3.Организация операционных устройств.	
4.3.1.Принцип микропрограммного управления.	1
4.3.2.Средства описания функций ОУ.	1
4.4.Структурная организация операционных устройств	1
4.4.1.Классификация операционных автоматов.	1
4.4.2.Организация операционного автомата	2
4.5.Структура микропроцессорных устройств и систем.	
4.5.1.Интерфейсы микропроцессорных устройств и систем.	1
4.5.2.Управление работой микропроцессорных систем	1
4.6.Микропроцессоры	2
РАЗДЕЛ 5.Устройства управления ЭВМ.	
5.1.Организация управляющего автомата	
5.1.1.Организация УА с программируемой логикой управления.	2
5.1.2.Укрупненная структура УА с программируемой логикой.	1
5.1.3.УА с жесткой логикой.	1
РАЗДЕЛ 6. Устройства ввода/вывода	
6.1.Организация систем ввода/вывода	1
6.2.Основные принципы построения систем ввода/вывода	1
6.3.Описание и структура многофункционального порта ввода/вывода	1
6.4.Принципы построения параллельного порта	1
РАЗДЕЛ 7. Элементы архитектуры компьютерных систем.	
7.1.Форматы команд и машинные операции.	1
7.2.Способы адресации информации в памяти ЭВМ.	1
РАЗДЕЛ 8. Специальные вопросы организации памяти ЭВМ.	
8.1.Виртуальная организация памяти.	1
8.2.Виртуальная память и организация защиты памяти.	2
8.3.Специализированные ЭВМ. Общие понятия.	1

ЛАБОРАТОРНЫЕ ЗАНЯТИЯ

Наименование лабораторной работы	Количество часов
1. Моделирование простейших устройств в пакете программ Max Plus II.	4
2. Реализация графа состояний.	4
3. Разработка устройства на основании имеющейся логической функции.	4
4. Разработка VHDL-кода устройства на основании имеющегося описания логической функции.	4
5. Моделирование передаточных функций.	4
6. Проектирование специализированных устройств микро-ЭВМ.	4
7. Проектирование специализированных устройств микро-ЭВМ.	4
8. Проектирование специализированных устройств микро-ЭВМ.	4
9. Проектирование специализированных устройств микро-ЭВМ.	4
10. Проектирование специализированных устройств микро-ЭВМ.	4
11. Проектирование специализированных устройств микро-ЭВМ.	4
12. Проектирование АЛУ.	4

ПРАКТИЧЕСКИЕ ЗАНЯТИЯ

Наименование практического занятия	Количество часов
1. Разработка функционального состава микро-ЭВМ заданной конфигурации.	2
2. Разработка структуры ПЗУ заданной конфигурации.	2
3. Разработка структуры ОЗУ заданной конфигурации.	2
4. Разработка узла местного управления фазами выполнения команд и структуры микропрограммы устройства управления.	2
5. Разработка устройства управления.	2
6. Построение АЛУ.	2
7. Разработка КПДП.	2
8. Реализация функционального состава микро-ЭВМ.	2

3. Оценка знаний студентов

Для оценки работы и знаний студентов в рамках курса «Структурная и функциональная организация ЭВМ» используется следующая система.

Для получения аттестации необходимо:

- выполнение всех предшествующих аттестации лабораторных работ;
- выполнение всех предшествующих аттестации практических работ;
- написание двух контрольных работ – 10 теоретических вопросов по пройденному материалу со средним баллом – 4(четыре);

Для получения минимальной положительной оценки – 4 балла, студент должен:

- выполнить и защитить все лабораторные и практические работы;
- написать все контрольные работы со средней оценкой не менее 4(четырех) баллов;
- не более 15% пропусков лекционных, лабораторных и практических занятий без уважительных причин.

Для получения максимальной оценки – 10 баллов – студент должен: проявлять самостоятельную творческую работу на практических и лабораторных занятиях; выступать с самостоятельно подготовленными докладами по тематике учебного курса; иметь необходимые знания и способность самостоятельно и творчески решать задачи по учебному курсу в нестандартной ситуации, иметь 100% посещение занятий.

Модуль 1.

ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

Цель модуля: изучение студентами базовых понятий основ структурной и функциональной организации ЭВМ.

В результате изучения модуля студенты должны знать:

- Основные понятия, такие как, вычислительная система, вычислительный комплекс, системы обработки данных; их назначение и различия, виды аппаратного и программного обеспечения;
- Четко определять понятия функции и структуры систем;
- Понимать различия между структурной и функциональной организацией ЭВМ.

Содержание модуля:

1.1. Системы обработки данных. Вычислительный комплекс. Вычислительная система.

1.2. Структурная и функциональная организация системы.

На основе ЭВМ строятся т.н. СОД – системы обработки данных. Определение: **СОД** – это совокупность технических средств и программного обеспечения (ПО), предназначенная для информационного обслуживания пользователей и (или) технических объектов (рисунок 1).

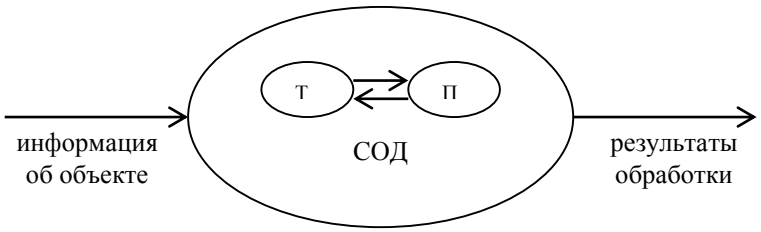


Рис.1.1. – Структура СОД

1.1. Системы обработки данных. Вычислительный комплекс. Вычислительная система.

СОД делятся на два больших класса: СОД общего назначения и автоматизированные СОД (АСОД). АСОД или просто АС – это человеко-машинная система, в которой информация об объекте управления или исследования (изучения) собирается и обрабатывается с помощью ЭВМ, а результаты обработки выдаются человеку–оператору АС и используются им для принятия решения по управлению (исследованию) объектом.

К классу АС относятся: информационно-измерительные системы (ИИС), АС управления технологическими процессами (АСУ ТП) и т.п. АС.

Технические средства СОД строятся на базе ЭВМ (в основном). Следует отметить, что в настоящее время термин ЭВМ (компьютер) трактуется широко – под ним понимается не только аппаратура, но и ПО, т.е. система в целом. Поэтому в ВТ аппаратную часть СОД обычно называют ВК. ВК – это аппаратная основа всех СОД.

Определение вычислительной системы. Под ВС понимается система, состоящая из двух частей (элементов) – АО и ПО, находящихся во взаимодействии (рисунок 1.2). Здесь: АО – аппаратное обеспечение. АО ВС – это технические средства ВС, т.е. ВК. ПО ВС – это системное ПО (СПО) и прикладное ПО (ППО), т.е., если точнее, ВС состоит из трех частей (рисунок 1.3).

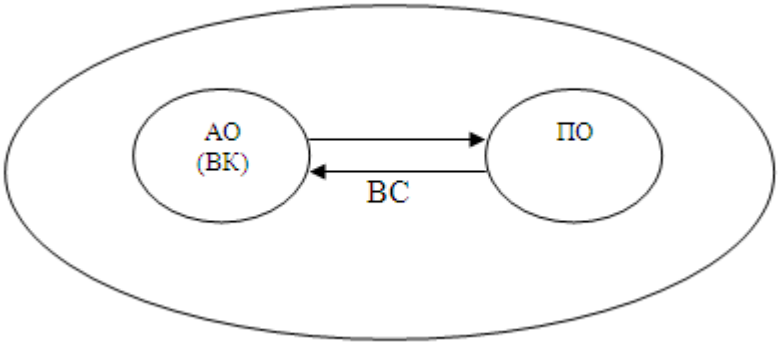


Рис.1.2.– Структура ВС

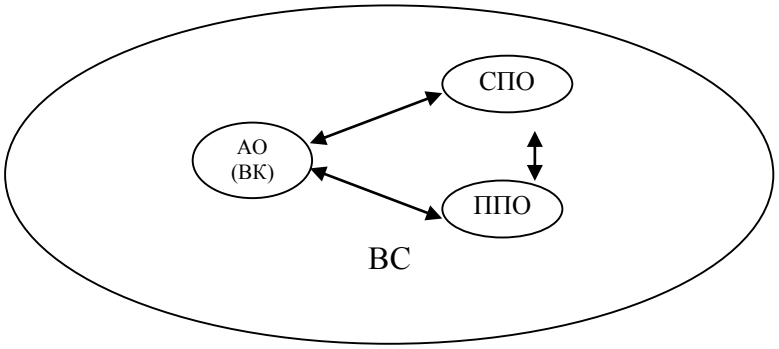


Рис.1.3.– Структура ПО ВС

Понятие ВС по составляющим элементам (АО+ПО) похоже на понятие СОД. Однако это разные понятия. Когда говорят СОД, то имеют в виду назначение системы, т. е. управление конкретным объектом (двигателем, например).

Термин ВС в ВТ используется тогда, когда разработчика СОД интересуют различные характеристики СОД. ВС - это сложная динамическая система, т.е. совокупность элементов системы и связей между ними, рассматриваемая в динамике, во взаимодействии.

Теория ВС состоит из двух разделов: архитектура ВС и метрическая теория ВС.

Архитектура ВС включает общую логическую организацию ВС, режимы работы (т.е. взаимодействие АО и ПО), способы представления данных, способы адресации и т.д.

Метрическая теория ВС занимается вопросами получения количественных оценок показателей, характеризующих организацию и функционирование ВС. В метрической теории исследуется (объясняется) влияние организации ВС на её характеристики: производительность, надёжность, стоимость и др. Здесь ставятся и решаются задачи выбора (определения) оптимальных параметров элементов, входящих в состав проектируемых систем.

Следует отметить, что в популярных источниках вместо терминов СОД, АС, ВК, ВС обычно используют термин ЭВМ (или компьютер), т.е. термином ЭВМ обычно называют СОД, другими словами, ВК, решающий какие-то конкретные задачи.

Известно, что ЭВМ относится к классу сложных систем. Поэтому при изучении принципов организации ЭВМ не обойтись без основных понятий из теории сложных систем.

Система – это совокупность элементов, объединенных в единое целое для достижения определённой цели.

ЭВМ – это система, предназначенная для автоматизации обработки информации на основе алгоритмов.

Сложные системы проектируются по принципу: от функции системы к её структуре, а также по принципу «сверху - вниз». Такой подход к проектированию сложных систем называется **функциональным**. Если проектирование сложных систем осуществляется именно так, то и изучение сложных систем разумно вести по такой же схеме: сверху - вниз, от функции к структуре.

Проектирование – это разработка такого описания проектируемой системы, которое позволяет ответить на вопросы: 1) как система устроена? 2) как функционирует? 3) как её построить (изготовить)? Ответ на третий вопрос важен при производстве систем в заводских условиях. Ответы на первые два вопроса очень важны при изучении системы.

Другими словами, под проектированием системы понимается разработка (получение) такого описания сложной системы, которого достаточно для её изготовления, эксплуатации и изучения.

Отсюда схема изложения материала курса: сверху - вниз, от функции к структуре. Из определения ЭВМ следует функция ЭВМ (рисунок 4): обработка исходных данных D на основе алгоритма A с целью получения результата R. Это первый, верхний уровень в иерархии описаний ЭВМ.

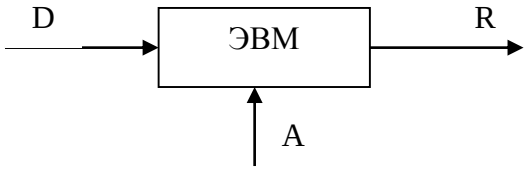


Рис.1.4.– Функция ЭВМ

Далее известно, что ЭВМ состоит из устройств (внутренних элементов): процессоров, запоминающих устройств (ЗУ), устройств ввода-вывода (УВВ). Поэтому на втором уровне иерархии (более детальном) описание ЭВМ можно представить схемой, изображенной на рисунке 5. На этом уровне предстоит ответить на вопрос: откуда взялись и для чего предназначены эти устройства (понятия): центральный процессор (ЦП), оперативное запоминающее устройство (ОЗУ), устройство ввода-вывода (УВВ), общая шина (ОШ).

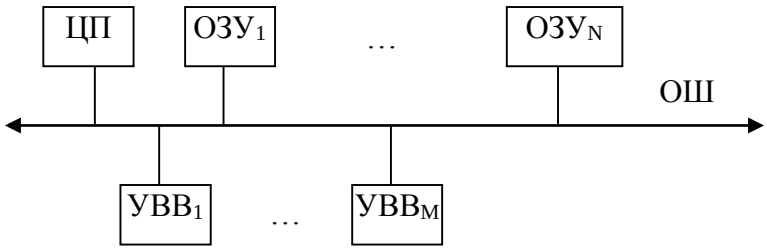


Рис.1.5.– Состав ЭВМ

В свою очередь, на следующем уровне детализации (иерархии описаний) раскрывается внутренняя организация устройств, из которых строится ВК. Например, ЦП строится на базе арифметико-логического устройства (АЛУ), центрального устройства управления (ЦУУ), регистров общего назначения (РОН) и т.д. до известного вам уровня схемотехники.

В результате мы получим иерархию описаний ЭВМ, составленных по принципу «сверху - вниз» (в смысле сложности) и от функции F к структуре S (при обосновании внутренней структуры и организации элементов устройств).

С точки зрения теории сложных систем система считается заданной (т.е. спроектированной), если определены и описаны её функция и структура (схема).

Другие основные понятия из теории сложных систем, которыми мы будем пользоваться: **функция системы, структура системы, организация системы, элемент системы.**

Функция системы - это такое описание системы, из которого ясно, как достигается поставленная перед системой цель. Другими словами, функция системы - это правила получения результатов, вытекающих из назначения системы. Например, назначением АЛУ является выполнение

арифметических и логических операций (АЛО). Отсюда функция АЛУ - правила получения результатов, т.е. правила выполнения арифметических и логических операций. Эти правила задаются путём описания алгоритмов выполнения АЛО: $F_{АЛУ} = \{A_+, A_-, \dots\}$ - перечень алгоритмов операций сложения, вычитания, и т. д.

Структура системы - это фиксированная совокупность элементов и связей между ними (элементами). Структуру системы принято изображать графически, в виде схемы, состоящей из элементов и связей (стрелок, линий) между элементами.

Со схемами, изображающими внутреннюю структуру интегральных схем (ИС), мы хорошо знакомы по курсу «Схемотехника». Пример: АЛУ строится на базе известных вам элементов: сумматоров, регистров, счетчиков, мультиплексоров, демультиплексоров и др.

1.2. Структурная и функциональная организация системы.

Организация – это способ приведения в порядок элементов, целью которой является получение требуемых функций в системах, состоящих из большого числа элементов.

Пример: триггер. Эффект хранения и новые качества зависят от внутренней организации.

Суть понятия организация заключается в ответе на вопрос: как организовать элементы в единое целое, чтобы получить нужный эффект – заданную функцию? В технике этот вопрос обычно формулируется так: организовать элементы в систему наилучшим, оптимальным образом.

В теории сложных систем различают два типа организации – функциональную и структурную.

Функциональная организация – это принципы построения абстрактных систем, то есть систем, заданных только их функциями. Примеры: таблица истинности для логических элементов (ЛЭ), набор алгоритмов операций – для АЛУ.

Структурная организация – это принципы перевода абстрактных систем в материальные (реальные) системы. Другими словами, это методы, приёмы, правила, с помощью которых осуществляется переход от функции F системы к структуре S, её реализующей ($F \rightarrow S$). Примеры: переход от таблицы истинности для ЛЭ к схеме (структуре) ЛЭ, переход от функции АЛУ FАЛУ к структуре АЛУ.

Следует отметить, что если переход от F к S, а также с одного уровня иерархии на другой, более детальный, подробный (сверху - вниз) формализован, то процесс проектирования осуществляется за один шаг, т. к. сводится к добросовестному следованию правилам перехода от F к S.

Однако, к сожалению, чаще всего этот переход не формализован. Как выход используют эвристические, не формальные методы проектирования, которые не гарантируют получение оптимального решения за один шаг.

Поэтому проектирование сложных систем носит итерационный характер. Результат проектирования существенно зависит от опыта и интуиции разработчика.

Элемент – это условное понятие, удобное для описания системы на данном уровне иерархии (детализации). Элемент – неделимая частица лишь на данном уровне иерархии. На других более низких уровнях элемент рассматривается как система, структура которой, в свою очередь, строится на основе более простых элементов и связей между ними.

Пример: АЛУ строится на базе сумматоров, регистров, счётчиков и т.п. элементов. В свою очередь, каждый из них строится на базе элементов другого уровня - логических элементов. Каждый ЛЭ, в свою очередь, состоит из известных вам полупроводниковых (электронных) элементов: транзисторов, резисторов, диодов, конденсаторов и электрических проводников для связи между ними.

Вопросы для самопроверки

1. Что такое система обработки данных?
2. Дайте определение вычислительной системы и вычислительного комплекса. Назовите основные отличия между ними.
3. Что такое структурная и функциональная организация? Какие отличия между этими двумя понятиями?
4. Дайте определения структуры и функции системы.
5. Что такое элемент?

Модуль 2.

ПРИНЦИПЫ ОРГАНИЗАЦИИ ЭЛЕКТРОННЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИН

Цель модуля: изучение студентами основных видов архитектуры ЭВМ; факторов, определяющих принципы организации ЭВМ.

В результате изучения модуля студенты должны знать:

- основные факторы, определяющие принципы организации ЭВМ;
- архитектуры ЭВМ: по Дж. Фон Нейману и Гарвардскую; понимать различия между данными типами архитектур;
- знать состав устройств ЭВМ и порядок их функционирования.

Содержание модуля:

2.1. Основные факторы, определяющие принципы организации электронных вычислительных машин;
2.2. Архитектура ЭВМ по Дж. Фон Нейману;
2.3. Гарвардская архитектура;
2.4. Состав устройств, структура и порядок функционирования электронных вычислительных машин;
2.5. Основные технические характеристики вычислительного комплекса.

Факторов, предопределяющих принципы построения ЭВМ достаточно много. Но если следовать схеме **сверху - вниз,** то есть два основных фактора - назначение ЭВМ и элементная база.

2.1. Основные факторы, определяющие принципы организации электронных вычислительных машин

Влияние элементной базы. Известно, что в ЭВМ используется исключительно двоично-кодированная форма представления информации: во-первых, потому, что при этом предельно упрощается конструкция элементов и машины в целом. Пример: двоичная и десятичная арифметика – отличаются в смысле сложности алгоритмов операций и, как следствие, сложности устройств, реализующих эти операции. Более простой пример: триггер как элемент хранения двоичной цифры и элемент с десятью устойчивыми состояниями как элемент хранения десятичной цифры (существенно сложнее и дороже триггера). Вторая причина - при использовании двоично-кодированной формы существенно возрастает надёжность элементов и ЭВМ в целом.

Второй фактор **назначение ЭВМ.** Из определения ЭВМ (автоматизация обработки информации на основе алгоритмов) следует, что принципы организации ЭВМ неизбежно должны зависеть от свойств алгоритмов.

Наиболее существенное влияние на организацию ЭВМ оказывают следующие три свойства алгоритмов.

1. Детерминированность (однозначность) вычислительных процессов, порождаемых алгоритмами.

2. При описании алгоритмов используется конечный набор элементарных операций. Примеры из начальной школы: правила умножения, деления и т.д.

3. Дискретное представление информации, с которой оперируют алгоритмы

Их влияние на организацию ЭВМ проявляется в следующем.

Детерминированность процессов – это основное свойство алгоритмов, которое позволило Джону фон Нейману использовать алгоритм как основу, источник управления процессом вычислений, процессом обработки информации в ЭВМ. А именно: алгоритм представляется в форме программы, вводится в память машины и используется для управления вычислительным процессом.

Конечный набор элементарных операций – отсюда вытекает, что и аппаратура ЭВМ (т.е. ВК) должна выполнять конечный набор сравнительно простых операций: сложение, вычитание, умножение, деление и др. Следовательно, F={+,–,×,/,...} - список машинных операций конечен и сравнительно прост.

Дискретное представление информации, с которой оперируют алгоритмы. Из этого свойства следует, что информация в ЭВМ представляется исключительно в дискретной форме – числовой, символьной, в форме логических значений. Причём, с учётом фактора элементной базы – не просто числовой, символьной и т.д., а ещё и в двоично-кодированной форме.

Анализируя сказанное, можно сформулировать принципы построения и функционирования современных ЭВМ в виде нескольких основных тезисов. Впервые их сформулировал Джон фон Нейман в 1945 году под названием “Принципы программного управления ЭВМ”. В популярном изложении их можно сформулировать следующим образом.

- Информация, подлежащая обработке с помощью ЭВМ, кодируется в двоичной форме и разделяется на **единицы информации – слова**. Слово – это совокупность двоичных элементов $a_1, a_2, ..., a_k$, где $a_i \in \{0,1\}$, k=8,16,32,64, k=const.
- Перед обработкой слова информации исходные данные размещаются в ячейках памяти ЭВМ. **Ячейка памяти** – это место хранения одного слова информации. Ячейки памяти нумеруются. Номер ячейки памяти называют **адресом**.
- Алгоритм обработки информации представляется в виде последовательности **управляющих слов** – т.е. **команд**. Каждая команда задаёт, предписывает аппаратуре ЭВМ тип выполняемой операции (указывает одну операцию из списка F), т.е. указывает аппаратуре что делать.

Кроме того, команда, в случае необходимости, указывает и местоположение операндов в памяти машины путём указания номера ячейки, т.е. указывает аппаратуре, где взять данные для обработки. Алгоритм, представленный в терминах команд, называют **программой.**

- Команды, как и данные, кодируются в двоичной форме и располагаются в ячейках памяти ЭВМ.
- Выполнение операций, предписанных программой, сводится к поочерёдному выбору команд из памяти и их выполнению (интерпретации) аппаратурой ЭВМ. Порядок, в котором команды извлекаются из памяти, задаётся алгоритмом решения задачи и зависит от исходных данных.

2.2. Архитектура ЭВМ по Дж. Фон Нейману

Универсальность ЭВМ вытекает из анализа сформулированных фон Нейманом принципов программного управления: функция ЭВМ задаётся программой, введённой в память ЭВМ, а не аппаратурой ЭВМ. Аппаратура ЭВМ может выполнять только операции из списка машинных операций F. Именно программа задаёт тот порядок, в котором операции должны выполняться для решения задачи (именно программа обеспечивает аранжировку операций). Таким образом, замена программы в памяти легко приводит к изменению функций ЭВМ, реализуемых аппаратурой ЭВМ.

Достоинства и недостатки фон Неймановских машин. Основные достоинства мы уже обсудили - это универсальность. Свойство универсальности является и основным недостатком. Дело в том, что для решения задачи алгоритм разрабатывается человеком и в форме программы загружается в память ЭВМ. Именно программа и несёт в себе **всю** необходимую для решения задачи информацию. Аппаратура ЭВМ лишь быстро и надёжно (т.е. без ошибок) реализует ее. Следовательно, аппаратура ЭВМ не обладает интеллектом и не может быть помощником человеку при решении интеллектуальных задач.

В связи с этим недостатком уже много лет актуальной является задача пересмотра классических принципов построения ЭВМ и поиск более рациональных. Переход к новым принципам организации ЭВМ специалисты связывают с появлением машин пятого поколения.

К настоящему времени сменилось четыре поколения машин. Все они фон Неймановские по принципу построения: первое поколение – ламповые ЭВМ, второе – ЭВМ на основе полупроводниковых дискретных элементов – транзисторов и интегральных схем (ИС) малой и средней степени интеграции, третье поколение – ЭВМ на основе ИС, четвертое поколение – ЭВМ на основе микропроцессорных больших ИС (БИС). Т.е. смена поколений развивалась по пути совершенствования элементной базы и технологии производства элементов и ЭВМ. Машины пятого поколения должны стать интеллектуальными. Для этого необходимо решить очень сложную проблему – проблему создания искусственного интеллекта. Когда

она будет решена, тогда и появятся теоретические основы для создания новых принципов организации аппаратуры ЭВМ. Одна из современных попыток – попытка реализация языка Пролог на аппаратном уровне. Есть много и других попыток, но не очень успешных пока.

2.3. Гарвардская архитектура

Еще одна архитектурная идея, связанная с преодолением проблемы семантического разрыва, но теперь в части неразличимости программы и данных, основывается на физическом разделении оперативной памяти на два независимых блока с собственными устройствами управления. Предложенная в Гарвардском университете она получила название «Гарвардская архитектура».

Схема такого процессора приведена на рисунке 6:

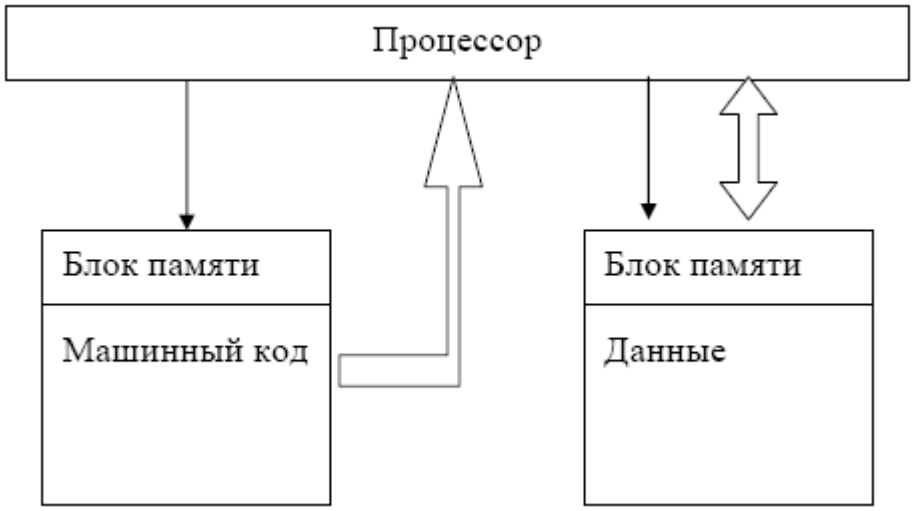


Рис.2.1.– Схема процессора с Гарвардской архитектурой

Два блока оперативной памяти для хранения программы и данных могут работать параллельно, что важно для конвейерной организации самого процессора. Для предотвращения возможности модификации программы во время выполнения (самомодифицируемые программы), что иногда активно использовалось при программировании в фон-неймановских процессорах, аппаратно запрещена операция записи в область машинного кода.

Такой подход к организации памяти широко используется в настоящее время в микропроцессорах, и в процессорах специального назначения, где чрезвычайно важно сохранить целостность программы, даже при возникновении аппаратной ошибки.

2.4. Состав устройств, структура и порядок функционирования электронных вычислительных машин

Состав устройств ЭВМ известен: это устройства типа “процессор”, запоминающее устройство (ЗУ), устройство ввода (УВв), устройство вывода (УВыв).

Известно, что для того, чтобы решить некоторую задачу, сначала необходимо разработать алгоритм ее решения, а затем этот алгоритм выполнить над некоторым набором исходных данных.

Выполнение известного алгоритма – работа механическая. Если работа чисто механическая, рутинная, следовательно, ее может выполнить и какое-то техническое устройство. Ответ Дж. фон Неймана таков: чтобы аппаратура ЭВМ могла выполнять алгоритм автоматически, без участия человека, алгоритм необходимо представить в терминах машинных команд, т. е. в форме программы, а затем заставить аппаратуру эту программу выполнить. Именно программа описывает путь решения задачи, чтобы ее решить – надо по этому пути пройти. Решение задачи – это процесс, протекающий во времени, в динамике. Отсюда следует, что для решения задачи с помощью аппаратуры необходимо в состав ЭВМ, кроме фон Неймановской памяти, состоящей из пронумерованных ячеек, ввести устройство, реализующее процесс выполнения программы. Это устройство естественно названо **процессором**. Он в основном и реализует фон Неймановский принцип программного управления. Процессор выполняет специальный алгоритм управления вычислительным процессом. Этот алгоритм прост и не зависит от конкретных программ (инвариантен по отношению к конкретным задачам). Он и реализуется аппаратурой ЭВМ. Называется он **циклом выполнения команд** и сводится к выполнению следующих действий:

- выборка очередной команды из памяти машины;
- выборка операндов (если необходимо);
- выполнение операции, предписанной командой;
- запись результата операции в память (если необходимо);
- переход к пункту 1.

Какие еще устройства, кроме памяти и процессора необходимы для автоматизации решения задачи? Ответ очевиден: устройство ввода и устройство вывода.

Структура простейшей ЭВМ также является очевидной (рисунок 2.2). Чего в ней нет по сравнению с современной машиной? Нет внешней памяти (ВП).

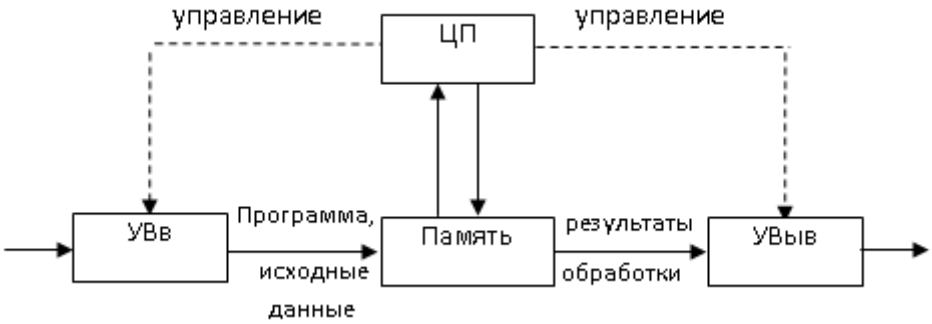


Рис.2.2.– Структура простейшей ЭВМ

Порядок функционирования ЭВМ. Современные ЭВМ работают под управлением операционных систем (ОС). Программы ОС обычно хранятся во ВП (на дисках). В момент включения питания в основной оперативной памяти (ОП) пусто. Для нормальной работы в ОП необходимо сначала ввести основную часть ОС (т.н. резидентную часть) из ВП. Обычно это делается под управлением программы начальной загрузки, которая в современных ЭВМ хранится в части ОП, реализованной на основе БИС ПЗУ. По объему это малая часть ОП.

Основная часть ОП строится на основе БИС ОЗУ.

После загрузки в ОП ЭВМ операционная система превращается в инструмент, помогающий человеку в автоматическом режиме быстро и без ошибок выполнять машинные программы.

2.5. Основные технические характеристики вычислительного комплекса

К числу основных относятся: операционные ресурсы, емкость памяти, быстродействие устройств, надежность, стоимость.

Операционные ресурсы – это перечень действий (операций), которые может делать (выполнять) аппаратура ВК в плане обработки информации (исходных данных). В этот перечень прежде всего включается **система машинных операций** – список F={+,-,*,/,…}. Кроме того, это порождающая ее (систему операций) система машинных команд К={К1, …, КN}. В понятие операционные ресурсы включаются также способы представления информации в ЭВМ, способы представления чисел, текстов, логических значений. Чем шире перечень действий, чем шире многообразие способов представления данных – тем шире операционные ресурсы ЭВМ и, следовательно, возможности ВК в плане обработки информации.

Емкость памяти – очевидная техническая характеристика, которая характеризует вместимость хранилища программ и данных ВК. Единицы измерения – бит, байт В, килобайт КВ = 2^{10}В, мегабайт МВ = 2^{20}В, гигабайт ГВ = 2^{30}В, терабайт ТВ = 2^{40}В. Емкость памяти Е обычно кратна степени 2: $E = 2^m$, m – длина адреса.

Быстродействие – это характеристика, которая отвечает на вопрос, как быстро действует (работает) аппаратура ЭВМ. Эта характеристика

определяет потенциальные возможности устройств, указывает на верхнюю границу. Относится к отдельным устройствам, а не ВК в целом. Так, быстродействие АЛУ характеризует скорость, с которой это устройство может выполнять операции: $V_{АЛУ}=\{V_{+}, V_{-}, V_{*}, V_{дел}, \ldots\}$. Быстродействие определяется количеством операций в единицу времени и зависит от времени выполнения операции: $V=1/t$ – чем меньше время выполнения операции t, тем выше быстродействие. Быстродействие – это паспортная характеристика, указывается в документе на устройство либо в виде вектора скоростей V, либо в виде набора времен: t_{+}, t_{-}, t_{*}, $t_{/}$, … Быстродействие процессора определяется временем выполнения команд. Следует отметить, что время выполнения команды $t_к$ зависит от многих факторов – быстродействия памяти (т.к. выборка команды и данных осуществляется из памяти, результаты также засылаются в память), от быстродействия АЛУ, а также организации ВК. В простейшем случае

$$t_к = t_{вк} + t_{во} + t_{алу} + t_{зр},$$

где первое слагаемое определяет время выборки команды из памяти, второе – время выборки операнда(ов), третье – время выполнения операции в АЛУ, четвертое – время засылки результата операции.

Быстродействие процессора принято измерять миллионами операций в секунду – MIPS или миллионами операций с плавающей запятой в секунду – MFLPS.

Память ЭВМ предназначена для хранения, записи и чтения информации. Быстродействие памяти принято характеризовать количеством операций чтения/записи в единицу времени. Память ЭВМ строится на базе ЗУ (БИС ОЗУ, ППЗУ). Быстродействие памяти зависит от быстродействия ЗУ и ее внутренней организации.

Т.о. быстродействие устройств ВК характеризует потенциальные возможности отдельных устройств ВК. Быстродействие ВК в целом зависит от многих факторов: от быстродействия устройств, внутренней организации самого комплекса, от операционной системы, под управлением которой работает аппаратура, т.е. от организации вычислительных процессов и др. факторов. Поэтому понятие быстродействие на ВК не распространяется. Вместо него используется понятие **производительность ВК.**

Назначением ЭВМ является обработка информации, т.е. решение различных задач. Поэтому производительность ВК естественно оценивать количеством задач в единицу времени. Но решаемые задачи разные. Оценка производительности ВК – проблема. Простейшее ее решение – смеси операций (Гибсона, например). Для сравнения различных ВК по производительности в ВТ обычно используют один и тот же набор программ, который прогоняют на ВК различных типов. Например, т.н. Бенч-Марковские программы и др.

Надежность ВК – это свойство ВК выполнять возложенные на него функции в течение заданного отрезка времени. Надежность ВК отлична от 100% (т. Е. от абсолютной) ввиду того, что элементы, из которых строится ЭВМ, рано или поздно перестают нормально работать. В результате отказа

элемента работоспособность ВК нарушается. Отказы аппаратуры – случайные события, частоту которых принято характеризовать интенсивностью отказов λ, т.е. количеством отказов в единицу времени. Другая характеристика надежности – т. Н. наработка на отказ: $T=1/\lambda$ - это промежуток времени между двумя соседними (по времени) отказами.

Общий подход к увеличению надежности ВК – резервирование аппаратуры, например, дублирование – двукратное резервирование. Если недостаточно, то трехкратное и т. Д.

Стоимость ВК – интегральная характеристика, определяется всеми перечисленными характеристиками. Чем лучше характеристика, тем выше стоимость.

Вопросы для самопроверки.

1. Назовите основные факторы, предопределяющие принципы построения ЭВМ.
2. Охарактеризуйте архитектуру ЭВМ по Дж. Фон Нейману и Гарвардскую архитектуру. Назовите основные различия между ними, а также достоинства и недостатки данных архитектур.
3. Приведите цикл выполнения команд.
4. Назовите базовый состав устройств ЭВМ.
5. Приведите основные характеристики вычислительного комплекса.

Модуль 3.

ПАМЯТЬ ЭВМ

Цель модуля: изучение студентами видов запоминающих устройств; их структурной и практической реализации.

В результате изучения модуля студенты должны знать:

- структуру ЗУ;
- классификацию ЗУ;
- иерархию ЗУ;
- структуру и принцип работы статических и динамических ОЗУ;
- структуру и принцип работы ПЗУ и флэш-памяти.

Содержание модуля:

3.1 Организация и основные характеристики запоминающих устройств.

3.2 Классификация ЗУ.

3.3 Иерархия памяти ЭВМ.

3.4 Организация ЗУ с произвольным доступом. Организация адресных (сверхоперативных) запоминающих устройств.

3.5 Синхронная динамическая память SDRAM.

3.6 Структура ЗУ с адресной организацией.

3.7 Кэш-память.

3.8 ПЗУ.

3.9 Компоновка ОЗУ основных моделей СМ ЭВМ.

3.10 Регенерация информации в динамических ЗУ. Динамическая память RDRAM.

3.11 Пакетное ОЗУ SLDRAM.

3.12 ОЗУ с виртуальными каналами обмена.

3.13 Модули динамических оперативных ЗУ.

3.14 Организация стековых (магазинных) запоминающих устройств. Запоминающие устройства с ассоциативной организацией.

3.15 Организация кэш-памяти на основе ассоциативного запоминающего устройства (кэш с ассоциативной организацией).

3.16 ЗУ с двумерной адресацией.

3.17 Другие типы полупроводниковых запоминающих устройств.

3.18 Организация дисковых массивов.

К числу основных относятся следующие понятия: запоминающий элемент (ЗЭ), ячейка памяти, запоминающее устройство (ЗУ), память ЭВМ.

Определение 1 памяти (ГОСТ): память – это часть ЭВМ, предназначенная для запоминания и выдачи информации.

Функции памяти: 1) хранение информации;

2) прием информации по запросу – запись;

3) выдача информации по запросу – чтение.

Операции чтения и записи информации в память принято называть **обращением к памяти.**

Запоминающий элемент – это место хранения бита информации. Типичный пример ЗЭ – триггер. На основе ЗЭ организуется хранение более крупных единиц информации – слов.

Ячейка памяти – это фиксированная совокупность ЗЭ, обращение к которым производится одновременно как единому целому. Ячейка памяти – это место хранения слова информации.

Доступ к информации, хранимой в ячейках памяти, обычно организуется по адресному принципу: ячейки памяти нумеруются числами 0, 1, …, E-1, номер ячейки называют ее **адресом**. Количество ячеек памяти E – **емкость памяти** – и длина адреса связаны отношением $E = 2^m$, m – длина адреса в битах. Таким образом, адресуемым элементом памяти является ячейка, а единицей обмена с памятью является слово: за одно обращение к памяти можно прочитать или записать одно слово информации.

Следует отметить, что в памяти третьего уровня (во внешней памяти) адресуются более крупные единицы информации – блоки, состоящие из фиксированного количества слов. Совокупность ЗЭ, предназначенная для хранения блока информации, в этом случае также можно считать ячейкой памяти, так как обращение к нему осуществляется как к единому целому: за одно обращение можно записать или прочитать один блок. Блок является единицей обмена с внешней памятью. Блоки как ячейки памяти нумеруются номерами 0, 1, …, E-1 и рассматриваются как адреса блоков.

Доступ к ячейкам памяти с заданным адресом A обеспечивается схемой селекции, которая выбирает одну из ячеек при обращении с целью записи или чтения слова (блока) информации. В простых случаях схема селекции выполняется на основе дешифратора, в более сложных случаях кроме дешифратора используются и другие схемы и механизмы.

Определение ЗУ. Совокупность ячеек памяти, объединенных в единое целое схемой селекции, называется ЗУ.

Второе определение памяти. Память ЭВМ – это совокупность ЗУ, объединенных в единую систему, управляемую ЦП.

3.1. Организация и основные характеристики запоминающих устройств

Всякое ЗУ состоит из двух частей: запоминающей части ЗЧ и блока управления БУ (рисунок 3.1).

Блок управления БУ обеспечивает выбор ячейки с адресом A, т.е. доступ к этой ячейке, и управление операциями чтения или записи по запросам ЧТ или ЗП: Операция чтения обеспечивает выдачу слова информации из ячейки с адресом A на выходную шину: ЧТ: ВЫХ := [A]. Операция записи обеспечивает прием слова со входной шины и запись его в ячейку с адресом

А: ЗП: [А]: = ВХ. Доступ к ячейкам памяти обеспечивается схемой селекции, входящей в состав БУ

Основные характеристики ЗУ: емкость, быстродействие, надежность, стоимость.

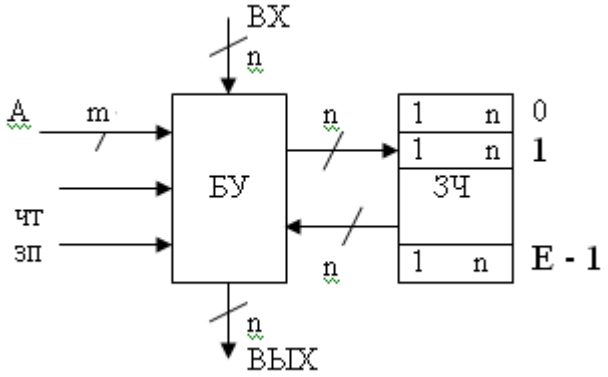


Рис.3.1.– Структура ЗУ

Емкость ЗУ определяется количеством ячеек, т.е. максимальным количеством информации, которая одновременно может храниться в ЗУ.

Быстродействие ЗУ определяется количеством операций обращения в единицу времени и зависит от продолжительности одной операции обращения: Vзу = 1/tобр.

В общем случае время обращения различно при выполнении операций ЧТ и ЗП:

$$t_{обр}^{чт} = \tau_д + \tau_{чт} + {}_{рег},$$
$$t_{обр}^{зп} = \tau_д + \tau_{подг} + \tau_{зп} \qquad (3.1)$$

здесь $t_д$ – время доступа к информации (к ячейке ЗУ);

$\tau_{чт}$ – время выдачи содержимого ячейки на выходную шину ЗУ;

$\tau_{рег}$ – время регенерации – повторной записи в ячейку информации, разрушенной при чтении (это время равно нулю, если чтение производится без разрушения информации);

$\tau_{подг}$ – время на подготовку ячейки к записи новой информации. Подготовка, если она требуется, осуществляется путем стирания старой информации. Время $\tau_{подг}$ =0, если стирания старой информации перед записью новой не требуется делать (например, в ЗУ на триггерах);

$\tau_{зп}$ – время собственно записи слова с входной шины ЗУ в выбранную схемой селекции ячейку ЗУ.

В зависимости от времени обращения (от быстродействия) ЗУ делятся на три класса: сверхоперативные, оперативные и внешние.

Надежность ЗУ обычно характеризуется временем наработки на отказ.

Стоимость ЗУ – интегральная характеристика. Она зависит от емкости, быстродействия, надежности ЗУ. Чем лучше эти технические характеристики, тем выше стоимость.

Применительно к ЗУ используется понятие удельной стоимости: δ=S/E, где S – стоимость ЗУ. Под удельной стоимостью понимается стоимость хранения единицы информации: байта, KB, MB, GB.

Следует отметить, что $\delta_{созу} > \delta_{озу} >> \delta_{взу}$.

3.2. Классификация ЗУ

1 В зависимости от возможности записи и перезаписи данных в памяти последняя подразделяется на:

- Память (ЗУ) с записью-считыванием [read/write memory] – тип памяти, дающей возможность пользователю помимо считывания данных производить их исходную запись, стирание и/или обновление. К этому виду могут быть отнесены оперативная память (ОЗУ, RAM, кэш-память), а также ППЗУ;
- Постоянная память, постоянное ЗУ, ПЗУ [Read Only Memory (ROM)] – тип памяти (ЗУ), предназначенный для хранения и считывания данных, которые никогда не изменяются. Запись данных на ПЗУ производится в процессе его изготовления, поэтому пользователем изменена быть не может. Наиболее распространены ПЗУ, выполненные на интегральных микросхемах (БИС, СБИС) и оптических (компакт) дисках (CD-ROM);
- Программируемая постоянная память, программируемое ПЗУ, ППЗУ [PROM – Programmable Read-Only Memory] – постоянная память или ПЗУ, в которых возможна запись или смена данных путем воздействия на носитель информации электрическими, магнитными и/или электромагнитными (в т.ч. ультрафиолетовыми или другими) полями под управлением специальной программы. Различают ППЗУ с однократной записью и стираемые ППЗУ [EPROM – Erasable PROM], в том числе:
  - Электрически программируемое ПЗУ (ЭППЗУ) [EAROM – Electrically Alterable Read Only Memory];
  - Электрически стираемое программируемое ПЗУ (ЭСПЗУ) [EEPROM – Electrically Erasable Programmable Read-Only Memory]. К стираемым ППЗУ относятся микросхемы флэш-памяти отличающиеся высокой скоростью доступа и возможностью быстрого стирания данных.

2 По признаку зависимости сохранения записи при снятии электропитания различают:

- Энергонезависимая (не разрушаемая) память (ЗУ) [nonvolatile storage] – память или ЗУ, записи в которых не стираются (“не разрушаются”) при снятии электропитания. К этому типу памяти относятся все виды ПЗУ и ППЗУ;
- Энергозависимая (разрушаемая) память (ЗУ) [volatile storage] – память или ЗУ, записи в которых не стираются (“не разрушаются”) при снятии

электропитания. К этому типу памяти относится оперативная память (ОЗУ, RAM и кэш-память);

• Динамическая память [dynamic storage] – разновидность энергозависимой полупроводниковой памяти, в которой хранимая информация с течением времени разрушается, поэтому для сохранения записей необходимо производить их периодическое восстановление (регенерацию), которое выполняется под управлением специальных внешних схемных элементов.

3 По признаку вида физического носителя и способа записи данных различают:

• Акустическая память [acoustic storage] – вид памяти (ЗУ), использующий в качестве среды для записи и хранения данных замкнутые акустические линии задержки;

• Голографическая память [holographic storage] – вид памяти (ЗУ), использующий в качестве среды для записи и хранения графической объемной (пространственной) информации голограмм;

• Емкостная память [capacitor storage] – вид памяти (ЗУ), использующий в качестве среды для записи и хранения данных конденсаторы;

• Криогенная память [cryogenic storage] – вид памяти (ЗУ), использующий в качестве среды для записи и хранения данных материалы, обладающие сверхпроводимостью;

• Лазерная память [laser storage] – вид памяти (ЗУ), в котором запись и считывание данных производится лучом лазера (CD-ROM);

• Магнитная память [magnetic storage] – вид памяти (ЗУ), использующий в качестве среды для записи и хранения данных магнитный материал. Разновидностями этого вида памяти являются: память на магнитной проволоке [plated wire memory], память на магнитной пленке [thin-film memory], наносимой на некоторую подложку, например, стеклянную. Наиболее широко использующимися устройствами реализации магнитной памяти в современных ЭВМ являются накопители на магнитных лентах (НМЛ), магнитных (жестких и гибких) дисках (НЖМД и НГМД);

• Магнитооптическая память [magnetooptics storage] – вид памяти, использующий магнитный материал, запись данных на который возможен только при нагреве до температуры Кюри (порядка 145 ^{0}C), осуществляемого в точке записи лучом лазера (объем записи на стандартные 3.5 и 5,25 дюймовые гибкие диски составляет при этом соответственно до 600 Мбайт и 1,3 Гбайт).

• Молекулярная память [molecular storage] – вид памяти, использующий технологию “атомной туннельной микроскопии”, в соответствии с которой запись и считывание данных производится на молекулярном уровне. Носителями информации являются специальные виды плёнок. Головки, считывающие данные, сканируют поверхность плёнки. Их чувствительность

позволяет определять наличие или отсутствие в молекулах отдельных атомов, на чем, и основан принцип записи/считывания данных. В середине 1999 г. Эта технология была продемонстрирована фирмой Nanochip . В основе архитектуры устройств записи/считывания лежит технология MARE (Molecular Array Read/write Engine). Достигнуты следующие показатели по плотности упаковки: ~ 40 Гбит/см^2 в устройствах чтения/записи и 128 Гбит/см^2 в устройствах с однократной записью, что считается в 6 раз выше, чем у экспериментальных образцов, которые основаны на классической технологии магнитной записи, и более чем в 25 раз превосходит лучшие её образцы, находящиеся в серийном производстве.

- Полупроводниковая память [semicondactor storage] – вид памяти (ЗУ), использующий в качестве средств для записи и хранения данных используются микроэлектронные интегральные схемы (БИС и СБИС). Преимущественное применение этот вид памяти получил в постоянных запоминающих устройствах и, в частности, в качестве оперативной памяти ЭВМ, поскольку он характеризуется высоким быстродействием;
- Ферритовая память [core storage] – вид оперативной памяти на ферритовых сердечниках;
- Фазоинверсная (PCR) память [PCR – Phase Change Rewritable storage] – разновидность лазерной (дисковой) памяти, использующей свойства некоторых полимерных материалов в точке лазерного нагрева в зависимости от температуры изменять фазовое состояние вещества (в частности – кристаллизоваться или плавиться с возвращением в исходное состояние), а вместе с ним – и характеристики отражения. Указанная технология позволяет создавать оптические диски (650 Мбайт) для многократной перезаписи данных. Разработкой данной технологии занимаются ряд фирм, включая Panasonic, Toshiba и Matsushita.
- Электростатическая память [electrostatic storage] – вид памяти (ЗУ), в котором носителями данных являются накопленные заряды статического электричества на поверхности диэлектрика.

4 По назначению, организации памяти и/или доступа к ней различают:

- Автономная память, автономное ЗУ [off-line storage] – вид памяти (ЗУ), не допускающий прямого доступа к ней со стороны центрального процессора: обращение к ней, а также управление ею производится вводом в систему специальных команд и через посредство оперативной памяти;
- Адресуемая память [addressed memory] – вид памяти (ЗУ), к которой может непосредственно обращаться центральный процессор;
- Ассоциативная память, ассоциативное ЗУ (АЗУ) [associative memory, content-addressable memory (CAM)] – вид памяти (ЗУ), в котором адресация осуществляется на основе содержания данных, а не их местоположения, чем обеспечивается ускорение поиска необходимых записей. С указанной целью поиск в ассоциативной памяти производится на основе определения того –

содержится ли в той или иной ее области (“ячейке памяти”) слово, словосочетание, символ и т.п., являющееся поисковым признаком. Существуют различные методы реализации ассоциативной памяти (АЗУ), в том числе – использующие методы поиска, основанные на “точном совпадении”, “близком совпадении”, “маскировании” слова-признака и т.д., а также различные процедуры реализации поиска, например, - “кэширования” с целью производства “наилучшей оценки” истинного адреса, за которой следует проверка содержимого ячейки с вычисленным адресом. Некоторые ассоциативные ЗУ строятся по принципу последовательного, другие – параллельного сравнения признаков поиска (т.н. “ортогональные ЗУ”). Параллельные ассоциативные ЗУ нашли применение в организации кэш-памяти и виртуальной памяти;

- Буферная память, буферное ЗУ [buffer storage] – вид памяти (ЗУ), предназначенный для временного хранения данных при обмене ими между различными устройствами ЭВМ;
- Виртуальная память [virtual memory] – способ организации памяти, в соответствии с которым часть внешней памяти ЭВМ используется для расширения ее внутренней (основной, оперативной) памяти. Например, содержимое некоторой области не используемой в данный момент времени внутренней памяти хранится на жестком диске и возвращается в оперативную память по мере необходимости; область (пространство) памяти, предоставляемая отдельному пользователю или группе пользователей и состоящая из основной и внешней памяти ЭВМ, между которыми организован т. Н. “постраничный” обмен данными. С указанной целью все адресное пространство делится на страницы памяти. Поиск адресов страниц производится в ассоциативной памяти;
- Временная память [temporary storage] – специальное запоминающее устройство или часть оперативной памяти, резервируемые для хранения промежуточных результатов обработки;
- Вспомогательная память [auxiliary storage] – часть памяти ЭВМ, охватывающая внешнюю и нарощенную оперативную память;
- Вторичная память [secondary storage] – вид памяти, который в отличие от основной памяти имеет большее время доступа, основывается на блочном обмене, характеризуется большим объемом и служит для разгрузки основной памяти;
- Гибкая память [elastic storage] – вид памяти, который хранит переменное число данных, пересылает (выдает) их в той же последовательности, в которой принимает и способна варьировать скорость вывода;
- Дополнительная память [add-in memory] – вид устройства памяти (ЗУ), предназначенного для увеличения объема основной оперативной или внешней памяти на жестком магнитном диске (ЖМД), входящих в основной комплект поставки ЭВМ.

- Иерархическая память [hierarchical storage] – вид памяти, имеющей иерархическую структуру, на верхнем уровне которой используется сверхоперативное запоминающее устройство, а на нижнем уровне – архивное ЗУ сверхбольшой емкости;
- Коллективная (массовая) память, память коллективного доступа [shared memory] – память, доступная множеству пользователей, которые могут обращаться к ней одновременно или последовательно; память, связанная одновременно с несколькими процессорами для обеспечения их взаимодействия при совместно решаемых ими задачах и использовании общих для них программных средств;
- Корректирующая память [patch memory] – часть памяти ЭВМ, предназначенная для хранения адресов неисправных ячеек основной памяти;
- Локальная память [local memory] – внутренняя память отдельного устройства ЭВМ (процессора, канала и т.п.), предназначенная для хранения управляющих этим устройством команд, буферируемых данных, а также сведений о состоянии устройства;
- Магазинная (стековая) память [pushdown storage] – вид памяти (ЗУ), являющийся аппаратной реализацией магазинного списка – стека, запись и считывание в котором осуществляются через одну и ту же ячейку – вершину стека;
- Матричная память [matrix storage] – вид памяти, элементы (ячейки) которой имеют такое расположение, что доступ к ним осуществляется по двум или более координатам;
- Многоблочная память [multibunk memory] – вид оперативной памяти, организованной из нескольких независимых блоков, допускающих одновременное обращение к ним, что повышает ее пропускную способность;
- Многовходовая память [multiport storage (memory)] – устройство памяти (ЗУ), допускающее независимое обращение с нескольких направлений (входов), причем обслуживание запросов производится в порядке их приоритета;
- Многоуровневая память [multilevel memory] – организация памяти, состоящая из нескольких уровней запоминающих устройств с различными характеристиками и рассматриваемая со стороны пользователей как единое целое. Для многоуровневой памяти характерна страничная организация, обеспечивающая “прозрачность” обмена данными между ЗУ разных уровней;
- Непосредственно управляемая (оперативно доступная) память [on-line storage] – память, непосредственно доступная в данный момент времени центральному процессору;
- Объектно-ориентированная память [object storage] – память, система управления которой ориентирована на хранение объектов. При этом каждый объект характеризуется типом и размером записи;
- Оверлейная память [overlayable storage] – вид памяти с перекрытием вызываемых в разное время программных модулей;

- Одноуровневая память [one-level storage] – вид памяти с единой адресацией для запоминающих устройств различных типов в одной ЭВМ;
- Память параллельного действия [parallel storage] – вид памяти, в которой все области поиска могут быть доступны одновременно;
- Перезагружаемая управляющая память [reloadable control storage] – вид памяти, предназначенный для хранения микропрограмм управления и допускающий многократную смену содержимого – автоматически или под управлением оператора ЭВМ;
- Перемещаемая память [data-carrier storage] – вид архивной памяти, в которой данные хранятся на перемещаемом носителе. Непосредственный доступ к ним от ЭВМ отсутствует;
- Память последовательного действия [sequential storage] – вид памяти, в которой данные записываются и выбираются последовательно разряд за разрядом;
- Память процессора, процессорная память [processor storage] – память, являющаяся частью процессора и предназначенная для хранения данных, непосредственно участвующих в выполнении операций, реализуемых арифметико-логическим устройством и устройством управления;
- Память (ЗУ) со встроенной логикой, функциональная память [logic-in-memory] – вид памяти (ЗУ), содержащий встроенные средства логической обработки (преобразования) данных, например, - их масштабирования, преобразования кодов, наложения полей и др.;
- Рабочая (промежуточная) память [working (intermediate) storage] – часть памяти ЭВМ, предназначенная для размещения временных наборов данных;
- Реальная память [real storage] – вся физическая память ЭВМ, включая основную и внешнюю память, доступная для центрального процессора и предназначенная для размещения программ и данных;
- Регистровая память [register storage] – вид памяти, состоящей из регистров общего назначения и регистров с плавающей запятой;
- Свободная (доступная) память [free space] – область или пространство памяти (ЗУ), которая в данный момент может быть выделена для загрузки программы или записи данных;
- Семантическая память, семантическое ЗУ [semantic storage] – вид памяти (ЗУ), в котором данные размещаются и из которой списываются в соответствии с некоторой структурой понятийных признаков;
- Совместно используемая (разделяемая) память (ЗУ) [shareable storage] – вид памяти (ЗУ), допускающий одновременное использование его несколькими процессорами ;
- Память с защитой, защищенное ЗУ [protected storage] – вид памяти (ЗУ), имеющий встроенные средства защиты от несанкционированного доступа к любой из его ячеек;
- Память (ЗУ) с последовательным доступом [sequential access storage] – вид памяти (ЗУ), в котором последовательность обращенных к ним входных сообщений и выборок данных соответствует последовательности, в которой

организованы их записи. Основной метод поиска данных в этом виде памяти – последовательный перебор записей;

- Память с прямым доступом, ЗУ прямого доступа, ЗУ с произвольной выборкой (ЗУПВ) [Random (direct) Access Memory (storage), RAM] – вид памяти (ЗУ), в котором последовательность обращенных к ним входных сообщений и выборок данных не зависит от последовательности, в которой организованы их записи или их местоположения;
- Память с пословной организацией [word-organized memory] – вид памяти, в которой адресация, запись и выборка данных производится не побайтно, а пословно;
- Статическая память (ЗУ) [static storage] – вид памяти, в котором положение данных и их значение не изменяются в процессе хранения и считывания. Разновидностью этого вида памяти является статическое ЗУПВ [static RAM];
- Страничная память [page memory] – память, разбитая на одинаковые области – “страницы”. Обмен с такой памятью осуществляется страницами;
- Управляющая память [control storage] – память, содержащая управляющие программы или микропрограммы. Обычно реализуется в виде постоянного запоминающего устройства.
- Флэш-память, ЭСППЗУ
- [EEPROM – Electrically Erasable PROM, flash memory] – “электрически стираемое перепрограммируемое ПЗУ”: полупроводниковое ЗУ, выполненное в виде микросхемы (чипа), в которую можно записывать данные и хранить их сколь угодно долго. Стирание производится электрическим разрядом, после чего на него можно записывать новые данные. Некоторые виды ЭСППЗУ для стирания и перепрограммирования требуют использования специальных устройств. Другие, собственно относящиеся к категории флэш-памяти, могут стираться и перепрограммироваться непосредственно в ЭВМ. Конструктивно флэш-память часто выполняется в виде т.н. “флэш-карт” [Flash-cards] или модулей памяти, которые используются в различных устройствах, например мобильных ПК, цифровых фотокамерах, сотовых телефонах, пейджерах, плейерах, портативных навигационных приборах и т.д. Одним из наиболее популярных вариантов реализации флэш-карт является Memory Stick (MS) фирмы Sony, выпуск которых начат с осени 1998 г. Эти карточки прочнее SmartMedia (также SSFDC – Solid State Floppy Disk Card) и компактнее CompactFlash .
- Многоуровневая флэш-память [StrataFlash] – усовершенствованная микросхема флэш-памяти, основанная на использовании ячеек не с двумя уровнями состояния (“включено” и “выключено”), как это имеет место в первых поколениях схем, а с четырьмя (два положения “включено” и два положения “выключено”). Таким образом, каждая ячейка может хранить в два раза больше данных, чем в обычной микросхеме. Технология разработана фирмой Intel, в 1997 г. Основная область предполагаемого применения: в

цифровых фотокамерах, карманных ПК, сотовых телефонах, цифровых автоответчиках и магнитофонах.

• Флэш-память двойной плотности [Double Density Flash Mamory] – усовершенствованная микросхема флэш-памяти, выпускаемая фирмой ScanDisk с ноября 1996 г. Относится к категории “многоуровневой флэш-памяти”.

3.3. Иерархия памяти ЭВМ

При разделении ЗУ по функциональному назначению иногда рассматривают два класса: внутренние и внешние ЗУ ЭВМ. Такое деление первоначально основывалось на различном конструктивном расположении их в ЭВМ. В настоящее время, например, накопители на жестких магнитных дисках, традиционно относимые к внешним ЗУ, конструктивно располагаются непосредственно в основном блоке компьютера. Поэтому разделение на внешние и внутренние ЗУ имеет в ряде случаев относительный, условный характер. Обычно к внутренним ЗУ относят устройства, непосредственно доступные процессору, а к внешним – такие, обмен информацией которых с процессором происходит через внутренние ЗУ.

Общий вид иерархии памяти ЭВМ представлен на рис. 3.20. На нем показаны различные типы ЗУ, причем, поскольку рисунок обобщенный, то не все из представленных на нем ЗУ обязательно входят в состав ЭВМ, а характер связей между устройствами может отличаться от показанного на рисунке.

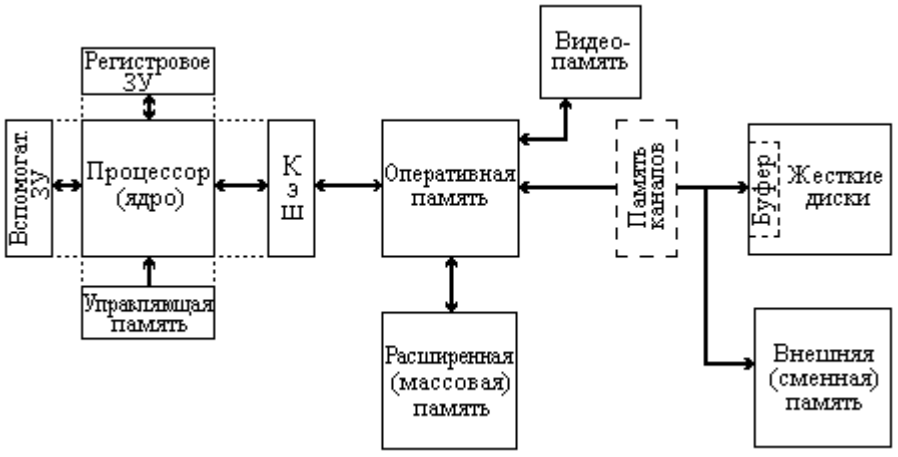


Рис.3.2.– Возможный состав системы памяти ЭВМ

Верхнее место в иерархии памяти занимают ***регистровые ЗУ***, которые входят в состав процессора и часто рассматриваются не как самостоятельный блок ЗУ, а просто как набор регистров процессора. Такие ЗУ в большинстве

случаев реализованы на том же кристалле, что и процессор, и предназначены для хранения небольшого количества информации (до нескольких десятков слов, а в RISC-архитектурах – до сотни), которая обрабатывается в текущий момент времени или часто используется процессором. Это позволяет сократить время выполнения программы за счет использования команд типа регистр-регистр и уменьшить частоту обменов информацией с более медленными ЗУ ЭВМ. Обращение к этим ЗУ производится непосредственно по командам процессора.

Следующую позицию в иерархии занимают ***буферные ЗУ***. Их назначение состоит в сокращении времени передачи информации между процессором и более медленными уровнями памяти компьютера. Буферная память может устанавливаться на различных уровнях, но здесь речь идет именно об указанном ее местоположении. Ранее такие буферные ЗУ в отечественной литературе называли сверхоперативными, сейчас это название практически полностью вытеснил термин «кэш-память» или просто ***кэш***.

Принцип использования буферной памяти во всех случаях сводится к одному и тому же. Буфер представляет собой более быстрое (а значит, и более дорогое), но менее емкое ЗУ, чем то, для ускорения работы которого он предназначен. При этом в буфере размещается только та часть информации из более медленного ЗУ, которая используется в настоящий момент. Если доля h обращений к памяти со стороны процессора, удовлетворяемых непосредственно буфером (37ЭШа37) высока (0,9 и более), то среднее время для всех обращений оказывается близким ко времени обращения к КЭШу, а не к более медленному ЗУ.

Пусть двухуровневая память состоит из кэш и оперативной памяти, как показано на рисунке 3.3. И пусть, например, время обращения к КЭШу t_c = 1 нс (10-9 с), время t_m обращения к более медленной памяти в десять раз больше – t_m = 10 нс, а доля обращений, удовлетворяемых 37ЭШа37, h = 0,95. Тогда среднее время обращения к такой двухуровневой памяти $T_{ср}$ составит $T_{ср}$ = 1 * 0.95 + 10 * (1 – 0.95) = 1.45 нс, т.е. всего на 45% больше времени обращения к 37ЭШа. Значение h зависит от размера КЭШа и характера выполняемых программ и иногда называется отношением успехов или попаданий (*hit ratio*).

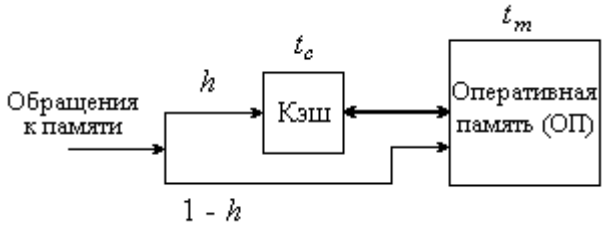


Рис.3.3.– К расчету среднего времени обращения
t_c – время обращения к кэш-памяти,
t_m – время обращения к ОП,
h – доля обращения, обслуживаемых кэш-памятью,
1 – h – доля обращений, обслуживаемых ОП

Размеры кэш-памяти существенно изменяются с развитием технологий. Так, если в первых ЭВМ, где была установлена кэш-память, во второй половине 1960-х годов (большие ЭВМ семейства IBM-360) ее емкость составляла всего от 8 до 16 Кбайт, то уже во второй половине 1990-х годов емкость КЭШа рядовых персональных ЭВМ составляла 512 Кбайт. Причем сама кэш-память может состоять из двух (а в серверных системах – даже трех) уровней: первого (L1) и второго (L2), также отличающихся своей емкостью и временем обращения.

Конструктивно кэш уровня L1 входит в состав процессора (поэтому его иногда называют внутренним). Кэш уровня L2 либо также входит в микросхему процессора, либо может быть реализован в виде отдельной памяти. Как правило, на параметры быстродействия процессора большее влияние оказывают характеристики кэш-памяти первого уровня.

Время обращения к кэш-памяти, которая обычно работает на частоте процессора, составляет от десятых долей до единиц наносекунд, т.е. не превышает длительности одного цикла процессора.

Обмен информацией между кэш-памятью и более медленными ЗУ для улучшения временных характеристик выполняется блоками, а не байтами или словами. Управляют этим обменом аппаратные средства процессора и операционная система, и вмешательство прикладной программы не требуется. Причем непосредственно командам процессора кэш-память недоступна, т.е. программа не может явно указать чтение или запись в кэш-памяти, которая является для нее, как иногда говорят, «прозрачной» (прямой перевод используемого в англоязычной литературе слова *transparent*).

Еще одним (внутренним) уровнем памяти являются *служебные ЗУ*. Они могут иметь различное назначение.

Одним из примеров таких устройств являются ЗУ микропрограмм, которые иногда называют управляющей памятью. Другим – вспомогательные ЗУ, используемые для управления многоуровневой памятью.

В управляющей памяти, использующейся в ЭВМ с микропрограммным управлением, хранятся микропрограммы выполнения команд процессора, а также различных служебных операций.

Вспомогательные ЗУ для управления памятью (например, теговая память, используемая для управления кэш-памятью, буфер переадресации *TLB – (translation location buffer*) представляют собой различные таблицы, используемые для быстрого поиска информации в разных ступенях памяти, отображения ее свойств, очередности перемещения между ступенями и пр.

Емкости и времена обращения к таким ЗУ зависят от их назначения. Обычно – это небольшие (до нескольких Кбайт), но быстродействующие ЗУ. Специфика назначения предполагает недоступность их командам процессора.

Следующим уровнем иерархии памяти является *оперативная память*. Оперативное ЗУ (ОЗУ) является основным запоминающим устройством ЭВМ, в котором хранятся выполняемые в настоящий момент процессором программы и обрабатываемые данные, резидентные программы, модули операционной системы и т.п. Название оперативной памяти также несколько изменялось во времени. В некоторых семействах ЭВМ ее называли основной памятью, основной оперативной памятью и пр. В англоязычной литературе также используется термин *RAM* (*random access memory*), означающий память с произвольным доступом.

Эта память используется в качестве основного запоминающего устройства ЭВМ для хранения программ, выполняемых или готовых к выполнению в текущий момент времени, и относящихся к ним данных. В оперативной памяти располагаются и компоненты операционной системы, необходимые для ее нормальной работы. Информация, находящаяся в ОЗУ, непосредственно доступна командам процессора, при условии соблюдения требований защиты.

Оперативная память реализуется на полупроводниках (интегральных схемах), стандартные объемы ее составляют (в начале 2000-х годов) сотни мегабайт – единицы гигабайт, а времена обращения – единицы÷десятки наносекунд.

Еще одним уровнем иерархии ЗУ может являться *дополнительная память*, которую иногда называли расширенной или массовой. Первоначально (1970-е годы) эта ступень использовалась для наращивания емкости оперативной памяти до величины, соответствующей адресному пространству (например, 24-битного адреса) команд, с помощью подключения более дешевого и емкого, чем ОЗУ, запоминающего устройства.

Это могла быть ферритовая память или даже память на магнитных дисках. Конечно, она была более медленной, а хранимая в ней информация сперва передавалась в оперативную память и только оттуда попадала в процессор. При записи путь был обратный.

Затем, в ранних моделях ПЭВМ, дополнительная память также использовалась для наращивания емкости ОЗУ и представляла собой отдельную плату с микросхемами памяти. А еще позже термин дополнительная память (*extended* или *expanded memory*) стал обозначать область оперативного ЗУ с адресами выше одного мегабайта. Конечно, этот термин применим только к IBM PC совместимым ПЭВМ.

В состав памяти ЭВМ входят также ЗУ, *принадлежащие отдельным функциональным блокам* компьютера. Формально эти устройства непосредственно не обслуживают основные потоки данных и команд, проходящие через процессор. Их назначение обычно сводится к буферизации данных, извлекаемых из каких-либо устройств и поступающих в них.

Типичным примером такой памяти является *видеопамять* графического адаптера, которая используется в качестве буферной памяти для снижения нагрузки на основную память и системную шину процессора.

Другими примерами таких устройств могут служить ***буферная память*** контроллеров жестких дисков, а также память, использовавшаяся в каналах (процессорах) ввода-вывода для организации одновременной работы нескольких внешних устройств.

Емкости и быстродействие этих видов памяти зависят от конкретного функционального назначения обслуживаемых ими устройств. Для видеопамяти, например, объем может достигать величин, сравнимых с оперативными ЗУ, а быстродействие – даже превосходить быстродействие последних.

Следующей ступенью памяти, ставшей фактически стандартом для любых ЭВМ, являются ***жесткие диски***. В этих ЗУ хранится практически вся информация, которая используется более или менее активно, начиная от операционной системы и основных прикладных программ и кончая редко используемыми пакетами и справочными данными.

Емкость этой ступени памяти, которая может включать в свой состав до десятков дисков, обеспечивая хранение очень большого количества данных, зависит от области применения ЭВМ. Типовая емкость жесткого диска, составляющая на начало 2000-х годов десятки гигабайт, удваивается примерно каждые полтора года.

Со временами обращения дело обстоит несколько иначе: компоненты этого времени, обусловленные перемещением блока головок чтения-записи уменьшаются сравнительно медленно (примерно вдвое за 10 лет). Компонента, обусловленная временем подвода сектора и зависящая от скорости вращения шпинделя диска, также уменьшается с ростом этой скорости примерно такими же темпами. А скорость передачи данных растет значительно быстрее, что связано с увеличением плотности записи информации на диски.

Все остальные запоминающие устройства можно объединить с точки зрения функционального назначения в одну общую группу, охарактеризовав ее как группу *внешних ЗУ*. Под словом «внешние» следует подразумевать то,

что информация, хранимая в этих ЗУ, в общем случае расположена на носителях не являющихся частью собственно ЭВМ. Под это определение подпадают гибкие диски, компакт диски, накопители на сменных магнитных дисках и магнитооптические диски, твердотельные (флэш) диски и флэш-карты, стримеры, внешние винчестеры и др. Естественно, что параметры этих устройств достаточно различны. Функциональное назначение их обычно сводится либо к архивному хранению информации, либо к переносу ее од одного компьютера к другому.

3.4. Организация ЗУ с произвольным доступом. Организация адресных (сверхоперативных) запоминающих устройств

Полупроводниковые ЗУ в настоящее время представляют собой большой класс запоминающих устройств, различных по своим функциональным и техническим характеристикам, широко используемых в качестве внутренних ЗУ ЭВМ. Но этим их использование не ограничивается. Подавляющее большинство электронной и бытовой техники переходит на цифровые методы представления данных (не только текстовых, но и аудио, графических и видео) и управления (использование микроконтроллеров).

Различные сферы применения накладывают свои особенности на реализацию полупроводниковых ЗУ, однако это чаще касается их конструктивных особенностей, а принципы построения одинаковы.

Высокое быстродействие полупроводниковых ЗУ обусловливает то, что большинство из них имеет организацию с произвольным доступом. Хотя такие ЗУ, как флэш-память и ЗУ с переносом зарядов (используемые, например, в фото- и видеокамерах), организованы несколько иначе.

Это же высокое быстродействие определяет и основные области применения полупроводниковых ЗУ в ЭВМ: кэш-память и оперативная память.

Причем надо отметить, что термин “ЗУ с произвольным доступом” (Random Access Memory – RAM) не соответствует в точности термину “оперативная память”, поскольку первый из них указывает на способ доступа, а второй – на функциональное назначение. И действительно, кэш-память и постоянные ЗУ также являются ЗУ с произвольным доступом. Однако, в соответствии с принятой в русскоязычной литературе терминологией, термин “оперативные ЗУ” ниже иногда используется как синоним ЗУ с произвольным доступом.

В данном разделе рассматриваются основные виды полупроводниковых ЗУ: статическая и динамическая память с произвольным доступом, постоянная и флэш-память, а также приводятся некоторые сведения по другим видам памяти.

Запоминающие устройства (ЗУ) характеризуются рядом параметров, определяющих возможные области применения различных типов таких устройств. К основным параметрам, по которым производится наиболее

общая оценка ЗУ, относятся их информационная емкость (Е), время обращения (Т) и стоимость (С).

Состав и структура микросхем оперативных ЗУ в процессе совершенствования технологий их изготовления подверглись определенным изменениям.

Первые полупроводниковые оперативные ЗУ строились на схемах малой и средней степени интеграции и включали в себя несколько различных типов микросхем: собственно матрицы элементов памяти, усилители чтения-записи, дешифраторы и, при необходимости, регистры (адреса и данных).

Позднее, с появлением больших интегральных схем (БИС) и повышением частоты их работы, использование раздельных типов микросхем перестало быть оправданным по следующим причинам. *Во-первых*, количество элементов памяти в матрицах возросло настолько, что число выводов, требующееся для выбора элемента памяти и равное сумме количества строк и количества столбцов матрицы, стало очень большим (несколько тысяч). *Во-вторых*, длина соединений между микросхемами больше, чем длина соединений внутри микросхемы, что увеличивает время прохождения сигнала и реактивные составляющие (емкость и индуктивность, или перекрестные помехи), а следовательно, уменьшает быстродействие памяти.

Поэтому микросхемы памяти стали включать в себя не только элементы памяти, но и всю остальную электронику управления: дешифраторы, усилители, буферные регистры, схемы управления. Такой состав БИС памяти приводил к известной аппаратной избыточности строящихся на их основе модулей памяти.

Действительно, первые БИС памяти имели логическую организацию вида N одноразрядных слов, или $N \times 1$, где N – количество адресов (одноразрядных слов) микросхемы. Следовательно, каждый разряд модуля памяти, построенного на таких микросхемах, включал свои собственные дешифраторы, буферные регистры и схемы управления, одного комплекта которых при традиционной организации было достаточно для целого модуля. Однако такое дублирование оправдывалось достигаемыми характеристиками памяти, а его стоимость не была чрезмерной (электроника обрамления составляла не более 5-15% от общей площади кристалла микросхемы).

Впоследствии разрядность хранимых в микросхеме слов была увеличена и составляет на сегодня от 4-х до 16-ти разрядов, т.е. $N \times 4$, $N \times 8$, $N \times 16$. Понятно, что относительная доля избыточных схем обрамления при этом падает.

На функциональных схемах микросхема памяти изображается обычным прямоугольником с левым и правым полями, как показано на рисунке 3.4.

Микросхема имеет три группы входов: адресные входы, вход(ы) данных и управляющие входы.

Количество адресных входов ($A_0 \div A_k$) определяется емкостью и организацией микросхемы памяти, а также способом подачи адреса.

Нетрудно видеть, что емкость микросхемы E_{Cx}, равная произведению количества адресов (слов) N на разрядность хранимых слов n, не определяет однозначно требуемое число адресных входов. Для адресации любого из N слов требуется адрес разрядностью $\log_2 N$. Например, для адресации микросхемы емкостью E_{Cx} = 128 Мбит, имеющей организацию 16М × 8 (адресов × бит), достаточно $\log_2 16\text{M} = \log_2 (2^4 \times 2^{20}) = 24$ разряда.

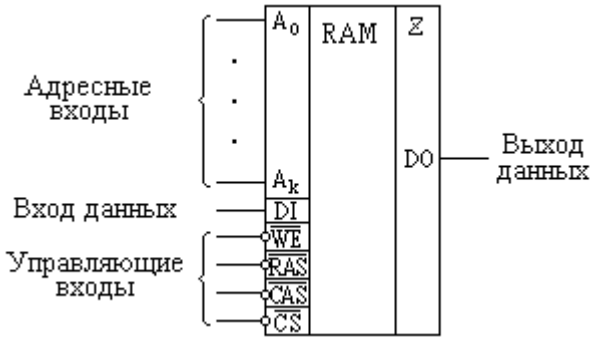


Рис.3.4.– Условное изображение ОЗУ на функциональных схемах

Способ подачи адреса также оказывает влияние на количество адресных входов микросхемы. Так, распространенный в динамических оперативных ЗУ прием мультиплексирования адресных входов, состоящий в поочередной подаче на одни и те же адресные входы сначала старшей части (половины) адреса – адреса строки (*Row Address*), а затем – младшей части – адреса столбца (*Column Address*), позволяет уменьшить вдвое количество требующихся адресных входов. Конечно, это несколько увеличивает время обращения к памяти, но оказывается экономически (да и схемотехнически) оправданным.

В статических ЗУ все разряды адреса подаются на адресные входы одновременно.

Количество входов данных (*DI – Data Input*) равно разрядности хранимых слов. Количество выходов данных (*DO – Data Output*) также равно разрядности хранимых слов. Однако во многих случаях входы и выходы данных объединяются, что позволяет уменьшить вдвое количество выводов данных у микросхем памяти, а также упростить их подключение к шинам данных.

Для этого выходы микросхем памяти (или объединенные входы/выходы) обычно имеют специальный выходной каскад, позволяющий подключать к одной шине выходы нескольких микросхем без использования дополнительных сборок ИЛИ. Есть два варианта организации таких выходов: выход с тремя устойчивыми состояниями (или z-выход) и выход с открытым коллектором. Тип выхода отмечается специальным значком в верхней части правого поля изображения микросхемы. На рисунке 3.4 показан z-выход.

Выход данных, реализованный по схеме с открытым коллектором, как правило, инверсный.

Управляющие входы могут заметно различаться как по назначению, так и по обозначениям для разных типов микросхем памяти.

Во всех случаях присутствует вход управления режимом обращения: чтение или запись. Частым его обозначением является ***WE***# (*Write Enable* – разрешение записи). Вход этот обычно инверсный (это и обозначает символ #), т.е. режим записи включается при нулевом значении сигнала на данном входе, а при единице на входе производится чтение.

Другим общим сигналом, имеющимся почти во всех микросхемах, является сигнал выбора микросхемы – ***CS***# (*Chip Select*). Этот вход также обычно является инверсным и при единичном значении на нем микросхема переходит в «выключенное» состояние (выход данных микросхемы переходит в состояние высокого выходного сопротивления, если он является z-выходом, или в состояние «1», если это инверсный выход с открытым коллектором). При нулевом значении сигнала на входе ***CS***# микросхема находится в активном состоянии.

В динамических ОЗУ при мультиплексировании адресных входов используются два управляющих входа сигналов строба: ***RAS***# (*Row Address Strobe* – строб адреса строки) и ***CAS***# (*Column Address Strobe* – строб адреса столбца, или колонки). Сигналы на этих входах переводятся в активное состояние (в «0») в тот момент, когда на адресных входах установлен адрес строки или адрес столбца соответственно.

Структурная схема БИС динамического ОЗУ показана на рисунке 3.5. Основными ее компонентами являются четыре банка памяти, представляющих собой матрицы элементов памяти с дешифраторами строк и столбцов и усилителями чтения-записи. Кроме собственно банков памяти в состав ОЗУ входят:

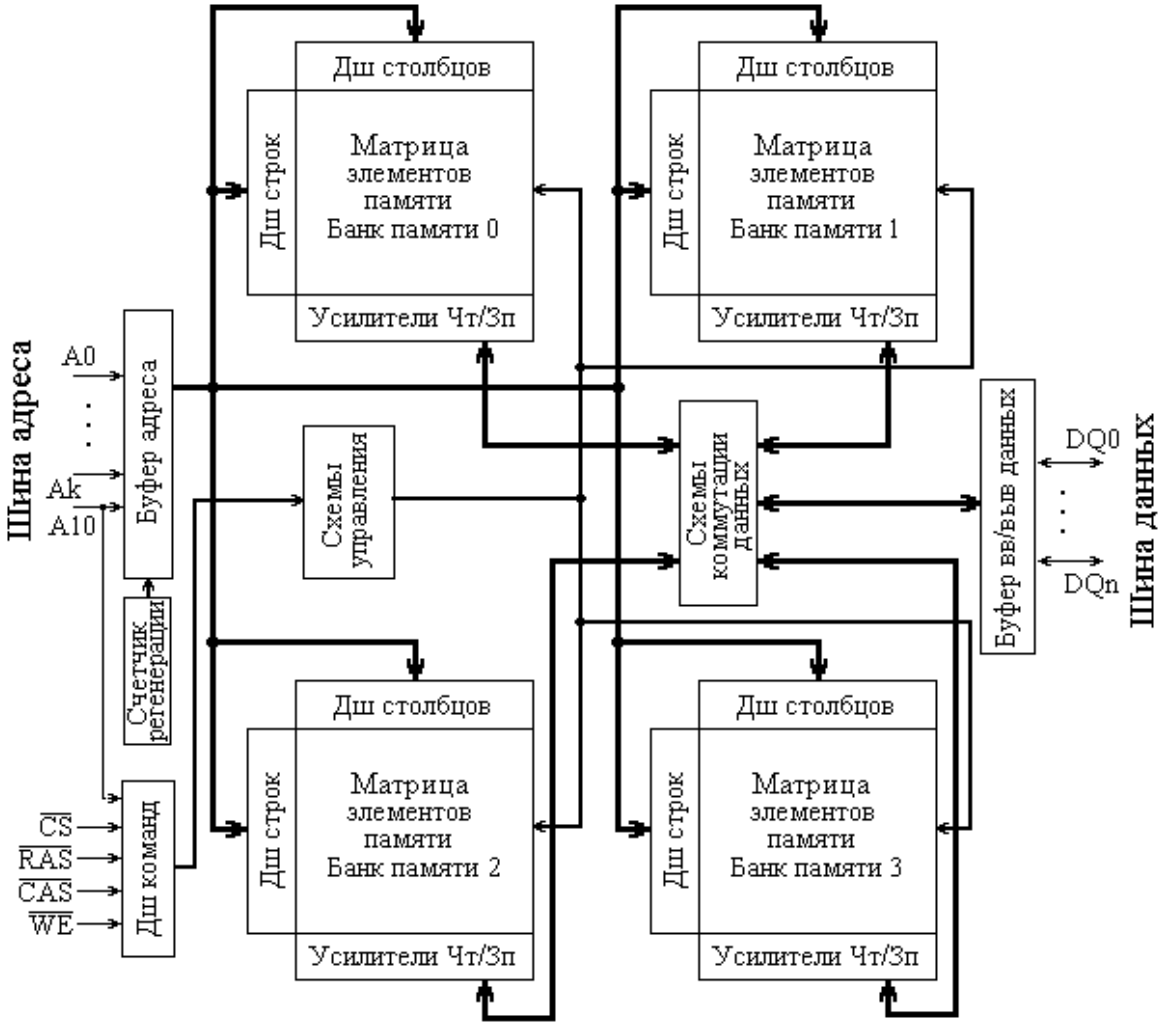


Рис.3.5.– Структурная схема БИС ОЗУ

- буфер адреса, фиксирующий адреса строки и столбца;
- счетчик регенерации, формирующий адрес строки, в которой должна выполняться очередная регенерация;
- дешифратор команд, определяющий, какое действие (команду) должна выполнить микросхема в соответствии с поданными управляющими сигналами (и сигналомA_{10});
- схемы управления, формирующие управляющие сигналы для остальных узлов микросхемы;
- схемы коммутации данных, передающие читаемые или записываемые данные из/в банки памяти;
- буфер ввода/вывода данных, обеспечивающий связь микросхемы памяти с шиной данных.

Адресный сигнал A_{10} выделен среди других адресных линий, так как он имеет специальное назначение: при подаче адреса столбца этот сигнал указывает на особенности выполнения последующего (пакетного) чтения или

записи, задавая (при единичном значении) режим так называемого автоматического подзаряда банка памяти.

3.4.1. Статические ЗУ с произвольным доступом

В статических ЗУ (*Static Random Access Memory* – SRAM) в качестве элемента памяти используется триггер, что, конечно, сложнее, чем конденсатор с транзисторным ключом динамического ЗУ. Поэтому статические ЗУ обладают меньшей плотностью хранения информации: емкость типовых микросхем статических ЗУ начала 2000-х годов не превосходила 16 Мбит.

Однако триггер со времен первых компьютеров был и остается самым быстродействующим элементом памяти. Поэтому статическая память позволяет достичь наибольшего быстродействия, обеспечивая время доступа в единицы и даже десятые доли наносекунд, что и обусловливает ее использование в ЭВМ, главным образом, в высших ступенях памяти – кэш-памяти всех уровней.

Главными недостатками статической памяти являются ее относительно высокие стоимость и энергопотребление.

Конечно, в зависимости от используемой технологии, память будет обладать различным сочетанием параметров быстродействия и потребляемой мощности. Например, статическая память, изготовленная по КМОП-технологии (CMOS память), имеет низкую скорость доступа, со временем порядка 100 нс, но зато отличается очень малым энергопотреблением. В ПЭВМ такую память применяют для хранения конфигурационной информации компьютера при выключенном напряжении сети (в этой же микросхеме размещают и часы, отсчитывающие реальное время). Питание такой памяти осуществляется от небольшой батарейки, которая может служить несколько лет.

Основными разновидностями статической памяти (SRAM) с точки зрения организации ее функционирования являются асинхронная (*Asynchronous*), синхронная пакетная (*Synchronous Burst*) и синхронная конвейерно-пакетная (*Pipeline Burst*) память.

Первой появилась асинхронная память, Интерфейс этой памяти включает шины данных, адреса и управления. В состав сигналов последней входят:

- ***CS***# (*Chip Select*) – сигнал выбора микросхемы;
- ***WE***# (*Write Enable*) – сигнал разрешения записи;
- ***OE***# (*Output Enable*) – сигнал включения выходов для выдачи данных.

Все сигналы управления инверсные, т.е. их активный (вызывающий соответствующее действие) уровень низкий. При единичном значении

сигнала ***OE***# выход микросхемы переходит в состояние высокого выходного сопротивления.

Временные диаграммы циклов чтения и записи приведены на рисунке 3.6 и не требуют особых пояснений. Цикл записи может быть организован и несколько иначе, чем показано на рисунке 3.6 *б*), в случае удержания во время цикла высокого уровня сигнала ***OE***#.

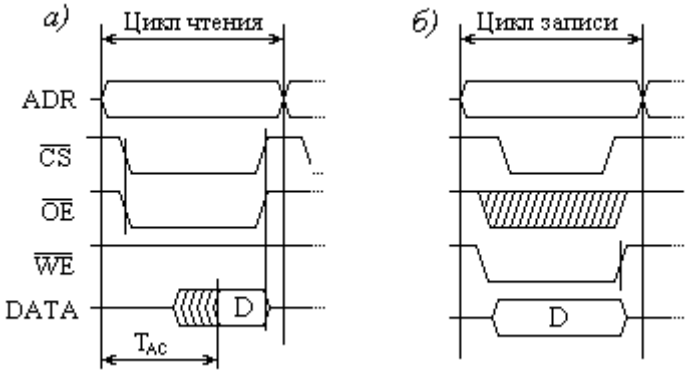


Рис.3.6.– Временная диаграмма простых циклов асинхронной статической памяти: а) чтения, б) записи

Время доступа t_{AC} у типовых микросхем составляет порядка 10 нс. Поэтому реально такие микросхемы могут работать на частотах, близких к частоте системной шины, только если эти частоты не превышают 66 МГц.

Несколько позже появилась синхронная пакетная статическая память (SBSRAM), ориентированная на выполнение пакетного обмена информацией, который характерен для кэш-памяти. Эта память включает в себя внутренний счетчик адреса, предназначенный для перебора адресов пакета, и использует сигналы синхронизации ***CLK,*** как и синхронная DRAM память.

Для организации пакетного обмена, помимо имеющихся у асинхронной памяти управляющих сигналов ***CS#, OE***# и ***WE***#, в синхронную память также введены сигналы ***ADSP***# (*Address Status of Processor*) и ***CADS***# (*Cache Address Strobe*), сопровождающие передачу адреса нового пакета, а также сигнал ***ADV***# (*Advance*) продвижения на следующий адрес пакета. Пакетный цикл всегда предусматривает передачу четырех элементов, так как внутренний счетчик имеет всего 2 бита, причем перебор адресов в пределах пакета может быть последовательным или с расслоением (чередованием) по банкам (при использовании процессоров семейства x86).

Временные диаграммы пакетных циклов чтения и записи приведены на рисунке 3.7. Обращения к синхронной памяти могут быть и одиночными. В этом случае низкому уровню сигнала ***ADSP***#, указывающему на передачу адреса, соответствует высокий уровень сигнала ***CADS***#, а не низкий, как при

пакетном цикле. Параметр T_{QK} характеризует время задержки данных относительно синхронизирующего сигнала.

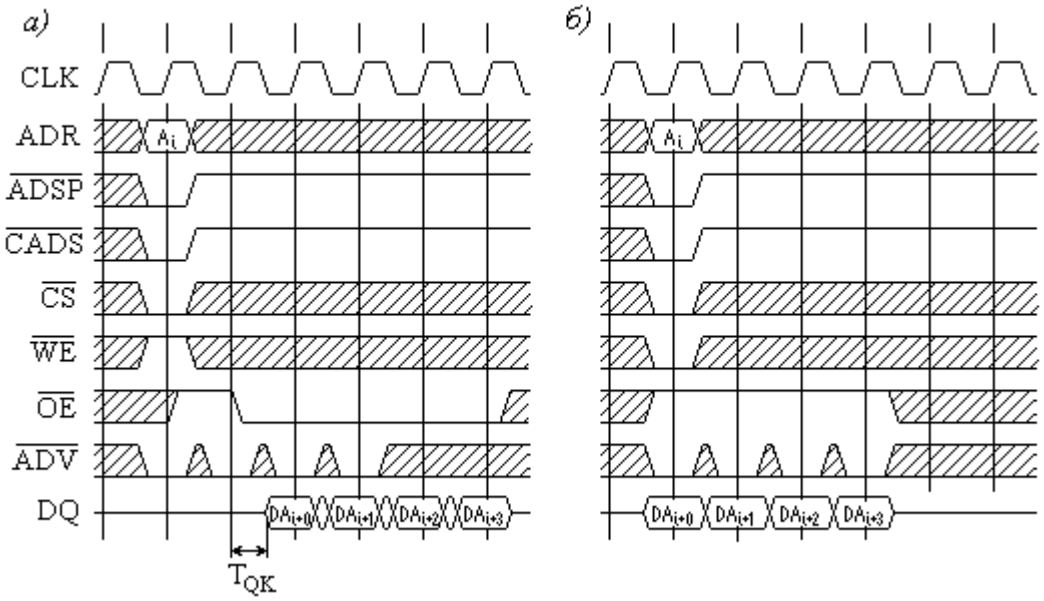


Рис.3.7.–Пакетные циклы синхронной пакетной статической памяти (SBSRAM): а) чтения, б) записи

Следующим шагом в развитии статической памяти явилась конвейерно-пакетная память PBSRAM, обеспечивающая более высокое быстродействие, чем SBSRAM. В нее были введены дополнительные внутренние буферные регистры данных (здесь можно провести аналогию с EDO DRAM памятью) адреса, а в ряде модификаций предусмотрена возможность передачи данных на двойной скорости по переднему и заднему фронтам синхросигнала и используются сдвоенные внутренние тракты записи и чтения. Это позволило получить время обращения порядка 2-3 нс и обеспечить передачу данных пакета без задержек на частотах шины более 400 Мгц.

Внутренняя логика позволяет переключаться с циклов чтения на циклы записи и наоборот без дополнительных задержек, кроме того, анализируется совпадение адресов записи и чтения для исключения избыточных операций.

Структурная схема такой памяти приведена на рисунке 3.8, где ФАП – блок формирования адресов пакета, МП – мультиплексоры, переключающие внутренние тракты чтения и записи в соответствии со значением младшего разряда адреса A0.

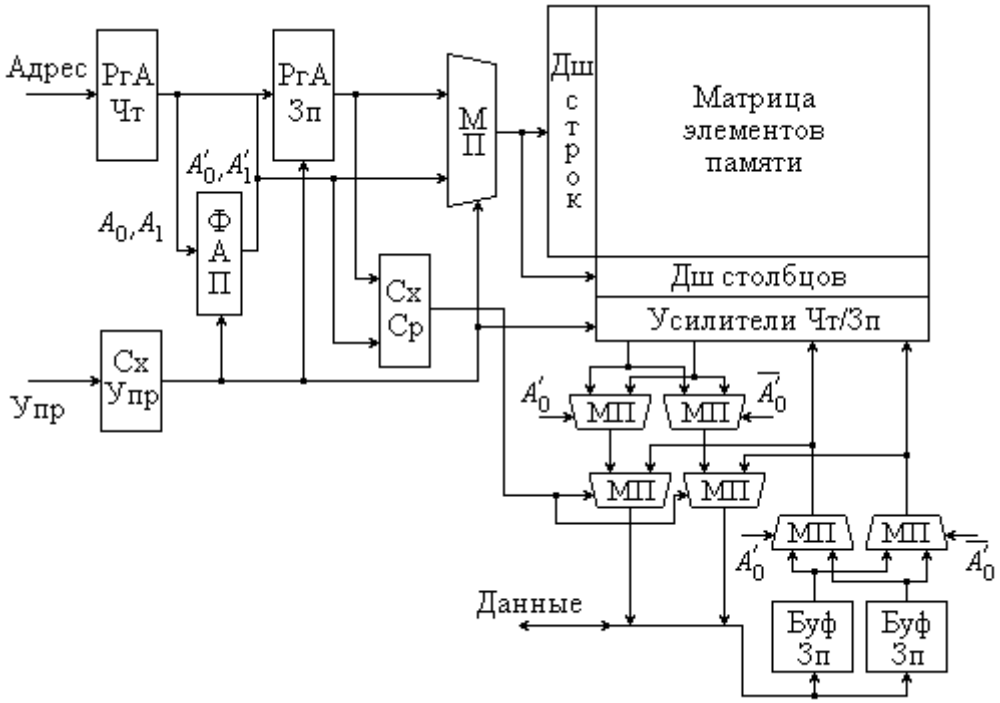


Рис.3.8.– Структурная схема синхронной конвейерно-пакетной статической памяти (PBSRAM)

Временная диаграмма, приведенная на рисунке 3.9, показывает запись и чтение как на одиночной (SDR – *Single Data Rate*), так и на двойной скорости (DDR – *Double Data Rate*) передачи. Сигналы ***CQ*** и ***CQ#*** – дифференциальные выходные сигналы синхронизации, близкие по времени к моменту появления данных на шине при чтении и используемые для синхронизации принимающих устройств. Сигналы ***SA*** и ***B*** – адресные и управляющие сигналы соответственно, причем последние используются для задания типа цикла. В режиме чтения с двойной скоростью передачи формируются два набора данных, начиная со второго переднего и заднего фронтов синхросигнала, если по его первому переднему фронту передается начальный адрес пакета. Первый набор данных (DOUT-A) формируется для заданного адреса, а второй (DOUT-A') – для следующего адреса пакета, в соответствии с определенным для пакета порядком.

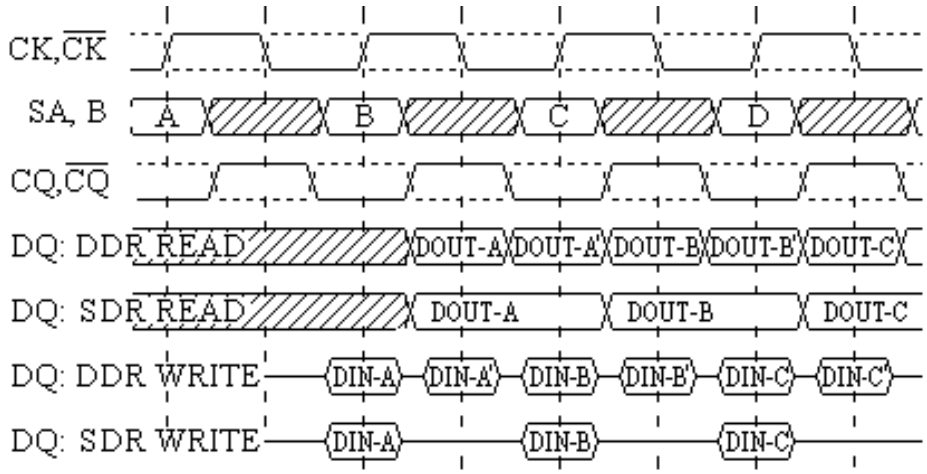


Рис. 3.9.– Общий вид диаграмм чтения и записи с одиночной и двойной скоростью передачи в синхронной конвейерно-пакетной статической памяти (PBSRAM)

Аналогичным образом запись с двойной скоростью передачи требует установки данных для записи, начиная со второго переднего и заднего фронтов сигнала синхронизации.

В режиме чтения с одиночной скоростью формируется только один набор данных по переднему фронту синхросигнала, начиная со второго сигнала, а при записи с такой же скоростью данные должны выставляться на шину, начиная с переднего фронта второго синхросигнала после передачи адреса.

Переключение из режима двойной скорости в режим одиночной (и наоборот) производится при подаче соответствующего управляющего сигнала.

Как отмечалось выше, в качестве оперативных ЗУ в настоящее время чаще используются динамические ЗУ с произвольным доступом (DRAM). Такое положение обусловлено тем, что недостатки, связанные с необходимостью регенерации информации в таких ЗУ и относительно невысоким их быстродействием, с лихвой компенсируются другими показателями: малыми размерами элементов памяти и, следовательно, большим объемом микросхем этих ЗУ, а также низкой их стоимостью.

Широкое распространение ЗУ этого типа проявилось также и в разработке многих его разновидностей: асинхронной, синхронной, RAMBUS и других. Основные из них рассматриваются далее.

3.4.2. Асинхронная динамическая память DRAM

В процессе совершенствования технологии производства изменялась и логика функционирования динамических ОЗУ.

Первые такие ЗУ, которые впоследствии стали называть асинхронными динамическими ОЗУ, выполняли операции чтения и записи, получив лишь

запускающий сигнал (обычно, сигнал строба адреса) независимо от каких-либо внешних синхронизирующих сигналов. Диаграмма простых (не пакетных) циклов чтения и записи для таких ЗУ представлена на рисунке 3.10, *а*) и 3.10, *б*) соответственно. Любой цикл (чтения или записи) начинается по спаду (фронту “1” →“0”) сигнала ***RAS***#.

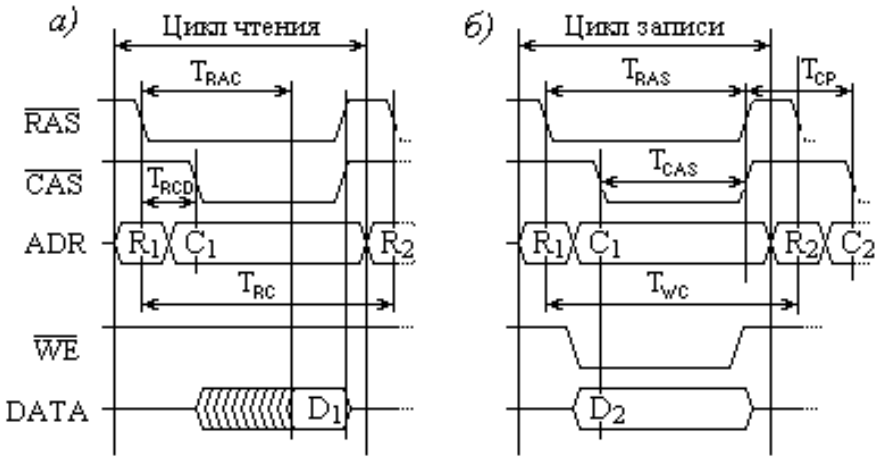


Рис. 3.10.– Временные диаграммы простых циклов чтения а) и записи б) асинхронной динамической памяти

Как видно из диаграмм, адрес на шины адреса поступает двумя частями: адрес строки (обозначенный как R_1 или R_2) и адрес столбца (C_1 и C_2). В момент, когда на адресной шине установилось требуемое значение части адреса, соответствующий сигнал строба (***RAS***# или ***CAS***#) переводится в активное (нулевое) состояние.

В цикле чтения (сигнал ***WE***# во время этого цикла удерживается в единичном состоянии) после подачи адреса строки и перевода сигнала ***CAS***# в нулевое состояние начинается извлечение данных из адресованных элементов памяти, что показано на диаграмме сигнала ***DATA*** как заштрихованная часть. По истечении времени доступа T_{RAC} (*RAS Access Time* – задержка появления данных на выходе ***DATA*** по отношению к моменту спада сигнала ***RAS***#) на шине данных устанавливаются считанные из памяти данные. Теперь после удержания данных на шине в течение времени, достаточного для их фиксации, сигналы ***RAS***# и ***CAS***# переводятся в единичное состояние, что указывает на окончание цикла обращения к памяти.

Цикл записи начинается так же, как и цикл чтения, по спаду сигнала ***RAS***# после подачи адреса строки. Записываемые данные выставляются на шину данных одновременно с подачей адреса столбца, а сигнал разрешения записи ***WE***# при этом переводится в нулевое состояние (известен и несколько иной цикл “задержанной” записи). По истечении времени, достаточного для записи данных в элементы памяти, сигналы данных, ***WE***#, ***RAS***# и ***CAS***# снимаются, что говорит об окончании цикла записи.

Помимо названного параметра T_{RAC} – времени доступа по отношению к сигналу ***RAS***# (его значение для микросхем второй половины 90-х годов XX столетия составляло от 40 нс до 80 нс), - на диаграмме на рис.15 указаны еще несколько времен:

- T_{RCD} – минимальное время задержки между подачей сигналов ***RAS***# и ***CAS***# (*RAS-to-CAS Delay*);
- T_{RAS} и T_{CAS} – длительности (активного уровня) сигналов ***RAS***# и ***CAS***#;
- T_{RC} и T_{WC} – длительности циклов чтения и записи соответственно;
- T_{RP} и $_{TCP}$ – времена подзаряда строки и столбца соответственно (время подзаряда определяет минимальную задержку, необходимую перед подачей очередного сигнала ***RAS***# или ***CAS***# после снятия (подъема в “1”) текущего).

Значения времен T_{RC} и T_{WC} для памяти (90-х годов) составляли порядка 50 – 100 нс, так что на одно (полное) обращение уходило от 5 до 7 циклов системной шины в зависимости от ее частоты, особенностей используемого чипсета и, собственно, быстродействия памяти. Так, для системной шины с частотой 66 Мгц длительность цикла составляет порядка 15 нс, что для 5 – 7 циклов дает диапазон 75 – 100 нс, если же частота системной шины составляла 100 Мгц, то 5 циклов занимают 50 нс.

Подача адреса двумя частями удлиняет цикл обращения к памяти. Вместе с тем большинство обращений непосредственно к оперативной памяти производится по последовательным адресам.

Действительно, как отмечалось выше, до 90 и более процентов обращений процессора к памяти удовлетворяются кэш-памятью. Те обращения, которые не могут быть удовлетворены КЭШем, вызывают обмен информацией между ОП и КЭШем. При этом передачи выполняются блоками, по 32 байта (4 цикла по 8 байт, в процессорах Intel 486 это были строки по 16 байт – 4 цикла по 4 байта), расположенными в последовательных адресах и называемыми строками КЭШа. Обмен информацией между оперативной памятью и внешними устройствами обычно выполняется целыми блоками, что также предполагает обращения по последовательным адресам.

Поскольку адрес строки является старшей частью адреса, то для последовательных адресов памяти адрес строки одинаков (исключение составляет переход через границу строки). Это позволяет в (пакетном) цикле обращений по таким адресам задать адрес строки только для обращения по первому адресу, а для всех последующих задавать только адрес столбца. Такой способ получил название FPM (*Fast Page Mode* – быстрый страничный режим) и мог реализовываться обычными микросхемами памяти при поддержке контроллера памяти чипсета, обеспечивая сокращение времени обращения к памяти для всех циклов пакета, кроме первого. Получающаяся

при этом временная диаграмма пакетного цикла чтения представлена на рисунке 3.11.

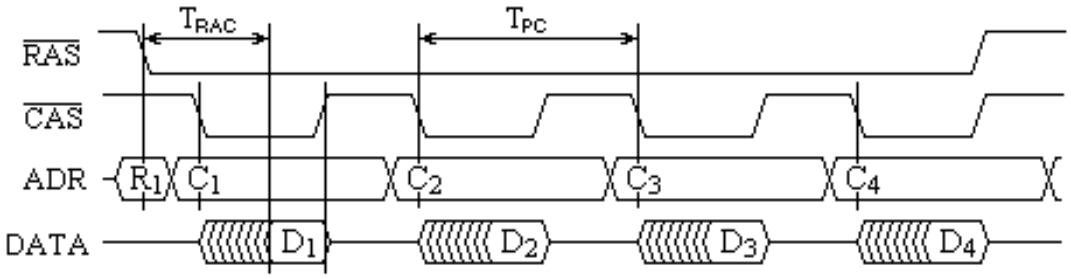


Рис.3.11.– Временная диаграмма цикла чтения последовательных адресов динамической памяти DRAM в режиме FPM

Как видно из рисунка, цикл чтения первого слова пакета выполняется так же, как и одиночное обращение. Второй и последующие циклы чтения оказываются короче первого из-за отсутствия фазы подачи адреса строки, и их длительность определяется минимально допустимым периодом следования импульсов ***CAS#*** – T_{PC} (*Page CAS Time*). Соотношение длительностей первого и последующих циклов при частоте системной шины может достигать 5:3, откуда и обозначение 5-3-3-3, используемое как характеристика памяти (и чипсета) и указывающее, что первый из циклов пакета занимает по времени 5 циклов системной шины, а последующие – по 3 цикла.

Длительность (низкого уровня) импульса ***CAS#*** определяется не только временем извлечения данных из памяти, но и временем удержания их на выходе микросхемы памяти. Последнее необходимо для фиксации прочитанных данных (контроллером памяти), так как данные присутствуют на выходе только до подъема сигнала ***CAS#***. Поэтому следующей модификацией асинхронной динамической памяти стала память EDO (*Extended Data Output* – растянутый выход данных). В микросхеме EDO памяти на выходе был установлен буфер-защелка, фиксирующий данные после их извлечения из матрицы памяти при подъеме сигнала ***CAS#*** и удерживающий их на выходе до следующего его спада. Это позволило сократить длительность сигнала ***CAS#*** и соответственно цикла памяти, доведя пакетный цикл до соотношения с циклами системной шины 5-2-2-2 (т.е. сократить длительность второго и последующих циклов в 1,5 раза только за счет выходного регистра-буфера). Временная диаграмма для режима EDO показана на рисунке 3.12, а сам этот режим иногда называют гиперстраничным (*Hyper Page Mode*).

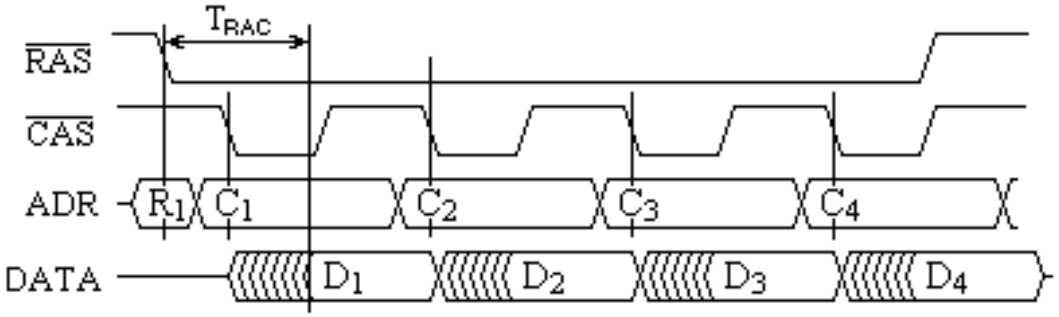


Рис.3.12.– Временная диаграмма цикла чтения последовательных адресов динамической памяти DRAM в режиме EDO

Впоследствии появилась и еще одна (последняя) модификация асинхронной DRAM – BEDO (*Burst EDO* – пакетная EDO память), в которой не только адрес строки, но и адрес столбца подавался лишь в первом цикле пакета, а в последующих циклах адреса столбцов формировались с помощью внутреннего счетчика. Это позволило еще повысить производительность памяти и получить для пакетного цикла соотношение 5-1-1-1.

Однако у отечественных поставщиков этот тип памяти не получил широкого распространения, так как на смену асинхронной памяти пришла синхронная – SDRAM.

3.5. Синхронная динамическая память SDRAM

Синхронная динамическая память обеспечивает большее быстродействие, чем асинхронная, при использовании аналогичных элементов памяти. Это позволяет реализовать пакетный цикл типа 5-1-1-1 при частоте системной шины 100 МГц и выше.

Основные сигналы интерфейса SDRAM схожи с сигналами интерфейса асинхронной памяти. Главные их отличия сводятся к появлению ряда новых сигналов:

- У памяти SDRAM присутствует синхросигнал ***CLK***, по переднему фронту которого производятся все переключения в микросхеме. Кроме этого сигнала имеется также сигнал ***CKE*** (*Clock Enable*), разрешающий работу микросхемы при высоком уровне, а при низком – переводящий ее в один из режимов энергосбережения.
- В интерфейсе SDRAM имеются сигналы выбора банка ***BS***0 и ***BS***1 (*Bank Select*), позволяющие адресовать конкретные обращения в один из четырех имеющихся в микросхемах SDRAM банков (массивов элементов) памяти.
- Присутствуют сигналы ***DQM*** маски линий данных, позволяющие блокировать запись данных в цикле записи или переключать шину данных в состояние высокого выходного сопротивления (z-состояние) при чтении.

- Имеет место специфическое использование одной из адресных линий (A_{10}) в момент подачи сигнала ***CAS***#. Значение сигнала на этой линии задает способ подзаряда строки банка.

Кроме того, SDRAM память сразу ориентирована на выполнение пакетных передач данных, причем длина пакета задается при инициализации микросхем памяти.

Временные диаграммы простых пакетных циклов чтения и записи приведены на рисунках 3.13 и 3.14.

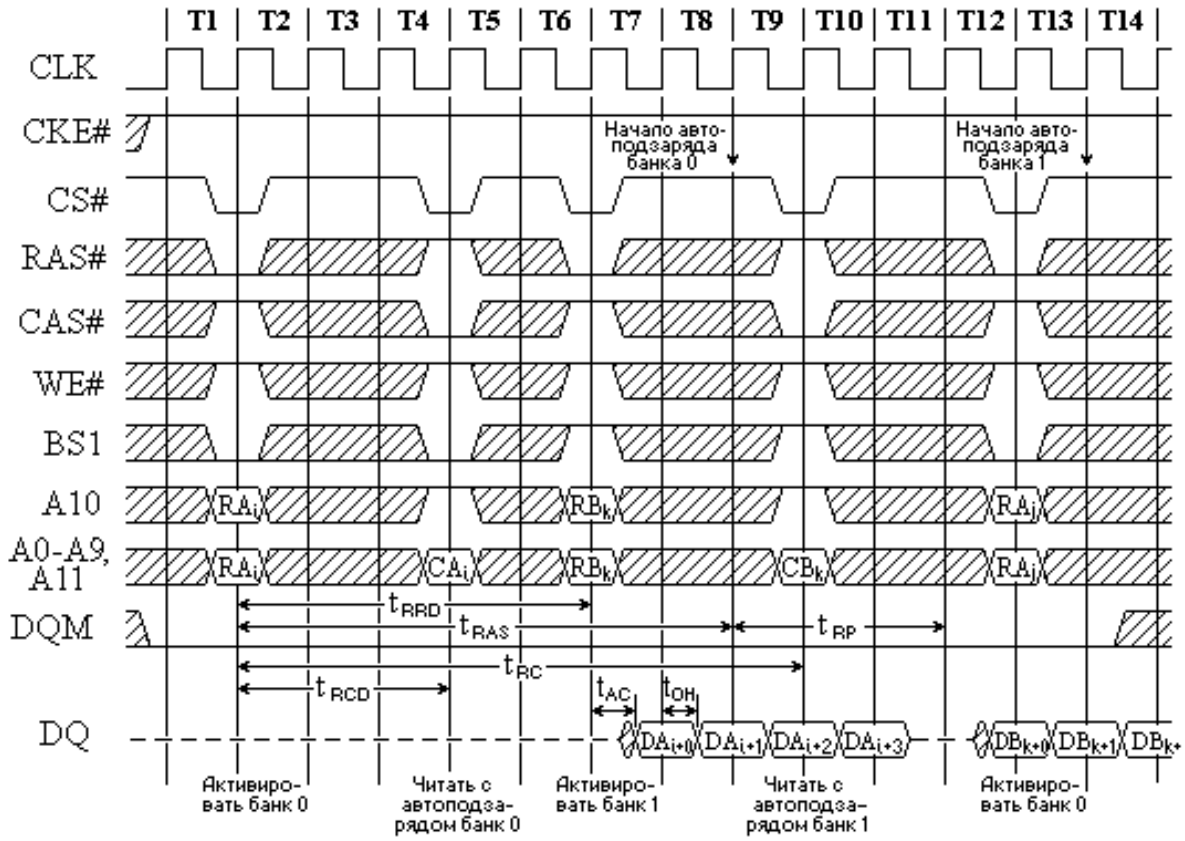


Рис.3.13.– Временная диаграмма пакетного чтения из SDRAM (длина пакета = 4, задержка появления данных CAS Latency = 3)

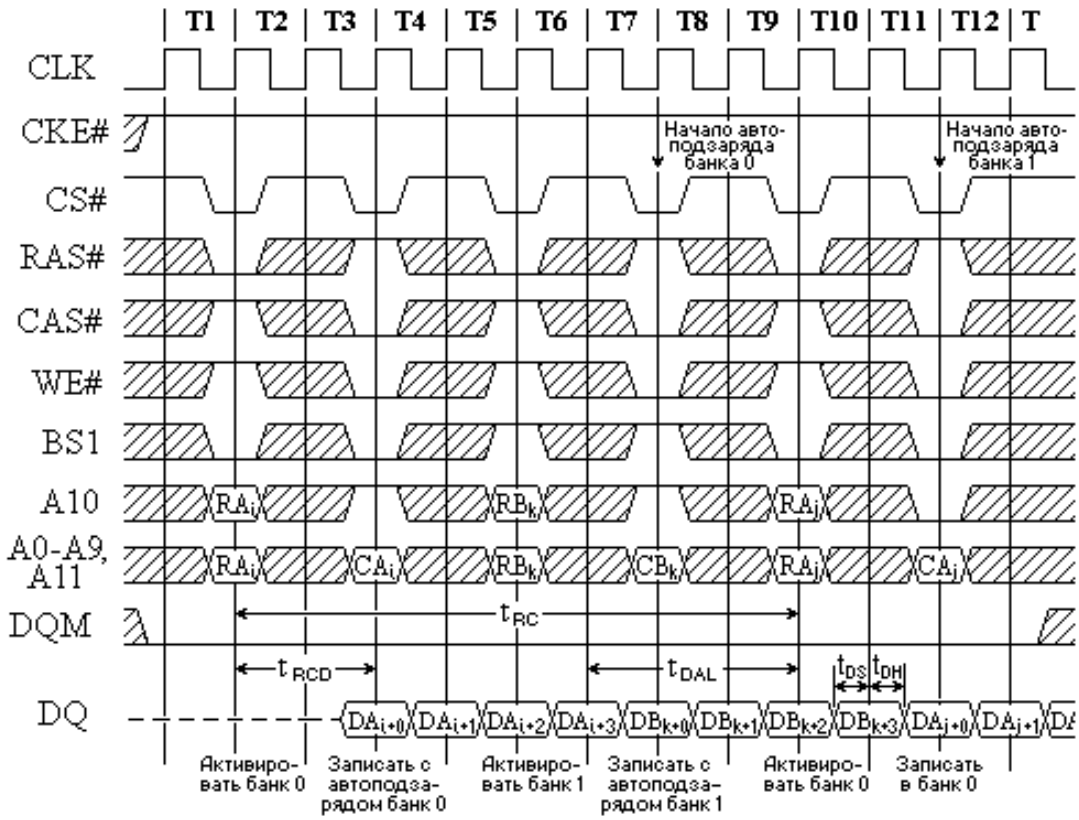


Рис.3.14.– Временная диаграмма пакетной записи в SDRAM (длина пакета = 4, задержка появления данных CAS Latency = 3)

Назначение основных сигналов приведено в таблице 3.1. Длина пакетов в SDRAM программируется, и на диаграммах показаны операции для пакетов длиной 4 цикла (32 байта). Временные параметры, указанные на диаграммах, имеют следующий смысл:

• t_{CK} – задержка данных по отношению к сигналу ***CAS***# (*CAS Latency*): минимально время (в тактах синхроимпульсов) между подачей сигнала ***CAS***# и появлением считанных данных на шине DQ;

• t_{RAS} – минимальное время (в тактах синхроимпульсов) активизации банка (*Row Active State*): минимально допустимое время удержания строки открытой;

• t_{RC} – время цикла строки (*Row Cycle*): минимальный временной интервал (в тактах синхроимпульсов) между двумя последовательными командами активации одного и того же банка ($t_{RC} = t_{RAS} + t_{RP}$);

• t_{RCD} – задержка между сигналами ***RAS***# и ***CAS***# (*RAS to CAS Delay*): минимально допустимое время (в тактах синхроимпульсов) между подачей сигналов ***RAS***# и ***CAS***#;

• t_{RP} – время подзаряда строки (*Row Precharge*): минимальное время (в тактах синхроимпульсов), необходимое для подзаряда строки после обращения к ней перед тем, как можно будет обратиться к другой строке того же банка;

• t_{RRD} – задержка между двумя последовательными сигналами ***RAS***# (*RAS to RAS Delay*): минимально допустимое время (в тактах синхроимпульсов) между командами активации различных банков;

• t_{AC} – время доступа (*Clock Access*): временной интервал (измеряется в наносекундах) от подачи сигнала синхронизации (того такта, в котором должны появиться данные чтения – второй или третий по отношению к сигналу ***CAS***# в зависимости от значения *CAS Latency*) до появления данных на шине DQ;

• t_{OH} – время удержания выходных данных (*Data Out Hold*): временной интервал (измеряется в наносекундах) от подачи сигнала синхронизации до снятия прочитанных в предыдущем такте данных с шины DQ;

• t_{DAL} – время задержки активации/регенерации (*Data in to Activate/Refresh Latency*): минимальный временной интервал (в тактах синхроимпульсов) от подачи последней порции данных пакета записи до команды активации/регенерации того же банка;

• t_{DS} – время установки входных данных (*Data in Set-up*): минимальный временной интервал (измеряется в наносекундах) от подачи записываемых данных на шину DQ до подачи команды записи (по переднему фронту синхроимпульса);

• t_{DH} – время удержания входных данных (*Data in Hold*): минимальный временной интервал (измеряется в наносекундах) от подачи команды записи до снятия записываемых данных с шины DQ.

Важно также учитывать, что различные микросхемы памяти могут иметь разные значения перечисленных параметров. Важными параметрами, влияющими на производительность памяти, являются параметры t_{CK}, t_{RCD} и t_{RP}. Значения этих времен для обычной памяти SDRAM составляют 2 или 3 такта синхроимпульсов и непосредственно определяют длительность основных операций памяти. Очевидно, что память, у которой все эти три параметра имеют значение 2, при пакетных циклах по 4 передачи потребует 10 тактов на полный пакет (t_{RCD} + t_{CK} + 4 такта передачи + t_{RP}), а в случае равенства этих параметров 3 потребуется 13 тактов. Время t_{RP} нужно учитывать с определенными оговорками, например, при обращении к разным строкам одного банка.

Приведенные на рисунках 3.13 и 3.14 диаграммы внешне аналогичны диаграммам асинхронной памяти в режиме BEDO. Действительно, в обоих случаях имеет место пакетный режим. Выигрыш в производительности SDRAM достигается за счет более гибкого управления процессами чтения и записи, возможности задания параметров и лучших алгоритмов работы контроллера памяти.

Во-первых, в SDRAM памяти процессы синхронны, что позволяет более жестко упорядочить их во времени.

Во-вторых, микросхемы SDRAM имеют внутреннюю мультибанковую организацию (на рисунке приведена структура именно такой микросхемы). Это позволяет применять приемы, повышающие пропускную способность памяти. В частности, можно прибегнуть к чередованию, или расслоению, (interleave) адресов, обсуждавшемуся ранее. Кроме того, оказывается возможным в ряде случаев так спланировать порядок обработки обращений к памяти, чтобы уменьшить их времена. Однако это не совсем простая задача, так как необходимо учитывать временные ограничения на различные сочетания следующих друг за другом операций, длину пакетных циклов, особенности выполнения подзаряда. Поэтому чипсеты различных производителей обеспечивают разную пропускную способность памяти. Лучше всего решение этой задачи для ПЭВМ удается фирме *Intel*.

Действительно, в общем случае процедура записи или чтения в SDRAM памяти выполняется в три этапа:

- Сначала при подаче сигнала ***RAS***# происходит выбор нужной строки, или в терминах, принятых для этой памяти, выполняется команда активации банка.
- Затем выполняются требуемые операции записи или чтения и передачи данных.
- После записи или чтения строку, к которой выполнялось обращение, надо закрыть (выполнить подзаряд банка), иначе нельзя будет обратиться к новой строке этого же банка (вновь его активировать).

Именно за счет первого и последнего этапов и можно добиться ускорения работы памяти. Если очередное обращение к данному банку будет адресоваться к той же строке, то ее можно не закрывать, что позволит не выполнять заново команду активации банка.

В таблице 3.1 перечислены сигналы микросхемы SDRAM, а ниже – выполняемые ею команды (функции).

Таблица 3. 1

Сигнал	Тип	Полярность	Функция
CLK	Вход	Положит. Фронт	Вход синхронизации системы. Все входы SDRAM срабатывают по положительному фронту CLK
CKE	Вход	Активный уровень – высокий	Активирует сигнал CLK высоким уровнем и отключает – низким. Отключение CLK низким уровнем CKE инициирует режимы энергосбережения и саморегенерации
CS#	Вход	Активный – низкий	Низкий уровень включает дешифратор команд, высокий –

			выключает. Когда дешифратор команд выключен, новые команды игнорируются, и продолжается предыдущая операция
Продолжение таблицы 3.1			
RAS#, CAS#, WE#	Вход	Активный – низкий	Состояние RAS, CAS и WE на положительном фронте CLK определяет операцию, которую будет выполнять SDRAM
BS0, BS1	Вход		Выбирает банк, который должен быть активным
A0 – A11	Вход		Во время цикла команды активации банка по переднему фронту CLK определяют адрес строки (RA0 – RA11) Во время цикла команды чтения или записи по переднему фронту CLK A0 – A9 и A11 определяют адрес строки (CA0 – CA9 и CA11). A10 используется для задания операции автоподзаряда в конце пакетного цикла чтения или записи. При высоком уровне A10 задан автоподзаряд, а BS0 и BS1 определяют банк для подзаряда. При низком уровне A10 подзаряд отключен. В цикле подзаряда A10 в сочетании с BS0 и BS1 задает банк (и), в котором выполняется подзаряд. При высоком уровне A10 подзаряд выполняется во всех банках, независимо от состояния BS0 и BS1. При низком уровне A10 банк для подзаряда определяется BS0 и BS1.
DQ0 – DQ15	Вход/Выход		Входы/выходы данных работают так же, как в обычной динамической памяти
DQM LDQM UDQM	Вход	Активный уровень – высокий	Маска ввода/вывода данных переводит буферы линий DQ в третье состояние высоким уровнем. В 16-разрядных модулях LDQM и UDQM

			управляют буферами ввода/вывода младшего и старшего байтов соответственно. В режиме чтения маска DQM имеет задержку в два цикла и управляет выходными буферами как сигнал “Разрешение выхода”. Низкий уровень DQM включает выходные буферы, а высокий – отключает. В режиме записи DQM имеет нулевую задержку и действует, как маска слова, разрешая запись входных данных низким уровнем и блокируя ее высоким уровнем DQM.
V_{DD}, V_{SS}	Вход		Питание и земля для входных буферов и логических схем ядра
V_{DDQ}, V_{SSQ}	Вход		Отдельные изолированные шины питания и земли для выходных буферов, обеспечивающие улучшенную помехоустойчивость к шумам

Команды, выполняемые микросхемой (DDR) SDRAM (Таблица 3.2):

Таблица 3.2

- установка регистра режимов - авторегенерация (CBR) - вход в режим саморегенерации - выход из режима саморегенерации - подзаряд одного банка - подзаряд всех банков - активация банка - запись - запись с автоподзарядом - чтение	- чтение с автоподзарядом - завершение пакета - нет операции - снятие выборки устройства - вход в режим приостановки синхронизации - выход из режима приостановки синхронизации - запись/включение выхода - маска/выключение выхода - вход в режим пониженного энергопотребления - выход из режима пониженного энергопотребления

Все выполняемые команды определяются состоянием сигналов ***CS#***, ***WE#***, ***RAS#***, ***CAS#*** и DQM по положительному фронту сигнала синхронизации.

3.5.1. Синхронная динамическая память DDR SDRAM

Следующим шагом в развитии SDRAM стала память DDR SDRAM, обеспечивающая двойную скорость передачи данных (*DDR – Double* или *Dual Data Rate*), в которой за один такт осуществляются две передачи данных – по переднему и заднему фронтам каждого синхроимпульса. Во всем остальном эта память работает аналогично обычной SDRAM памяти (которую стали иногда называть SDR SDRAM – *Single Data Rate*). Времена задержек *CAS Latency* для DDR SDRAM могут быть 2 и 2,5 такта.

Из рассмотрения временных диаграмм понятно, что производительность DDR SDRAM вовсе не в 2 раза выше, чем производительность обычной SDRAM, так как ускорение касается только собственно передачи данных, основные задержки остались теми же. Т.е. при задержках (t_{CK}, t_{RCD} и t_{RP}) в 2 цикла и пакетах длиной в 4 передачи времена передачи для DDR и обычной памяти составят 8 и 10 тактов соответственно.

Коммерческие названия SDR и DDR типов памяти SDRAM несколько различаются. Для обычной памяти используют для указания скоростных характеристик рабочую частоту системной шины: PC100, PC133, что соответствует времени такта синхроимпульсов 10 нс и 7,5 нс. Тогда как для DDR SDRAM указывают скорость передачи данных, что с учетом передачи за один раз 8 байтов данных дает скорости (при двух передачах за такт) при частоте шины 133 МГц – 2 × 133 × 8 = 2128 Мбайт/с, при частоте 166 МГц – 2 × 166 × 8 = 2656 и при частоте 200 МГц – 2 × 200 × 8 = 3200. Такую память маркируют PC2100, PC2700 и PC3200 соответственно, причем этот ряд постоянно растет.

Дальнейшим развитием SDRAM является стандарт DDR2. В нем обеспечивается учетверенная скорость передачи данных по отношению к частоте работы самих элементов памяти. Авторы этого стандарта отмечают его эволюционный характер. Микросхемы такого типа изготавливаются в других корпусах, а модули памяти имеют 240 контактов.

3.6. Структура ЗУ с адресной организацией

Простейшая структура ЗУ с адресной организацией представлена на рисунке 3.15.

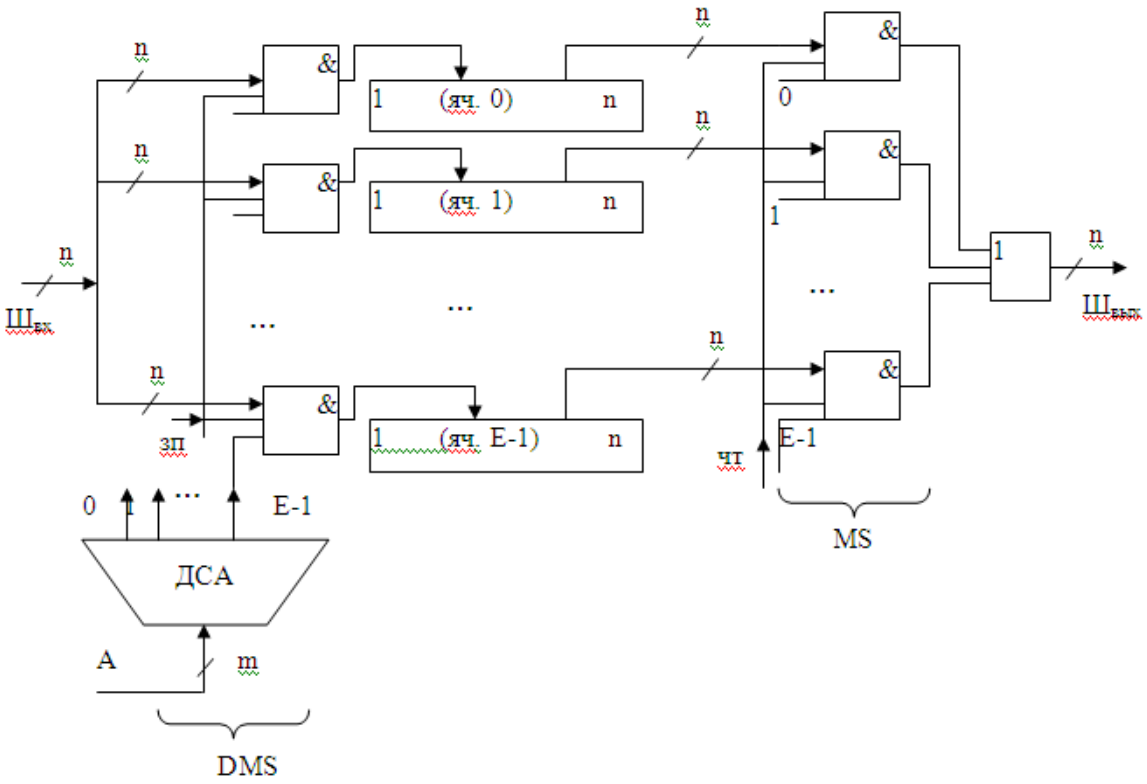


Рис.3.15.– Структура ЗУ с адресной организацией

Запоминающая часть ЗУ организована как линейная последовательность ячеек, обращение к каждой из которых обеспечивается схемой селекции, построенной на основе дешифратора адреса DCA, демультиплексора DMS и мультиплексора MS. Выбор ячейки при выполнении операции записи осуществляется дешифратором DCA и демультиплексором DMS, а при чтении – DCA и MS.

Такого рода ЗУ с адресной организацией принято называть ЗУ типа 2D, т.е. ЗУ с двумя измерениями (координатами): первая координата при обращении – адрес A, вторая – направление обмена – чтение или запись. Основой построения ЗУ типа 2D являются запоминающие элементы следующего вида (рисунок 3.16):

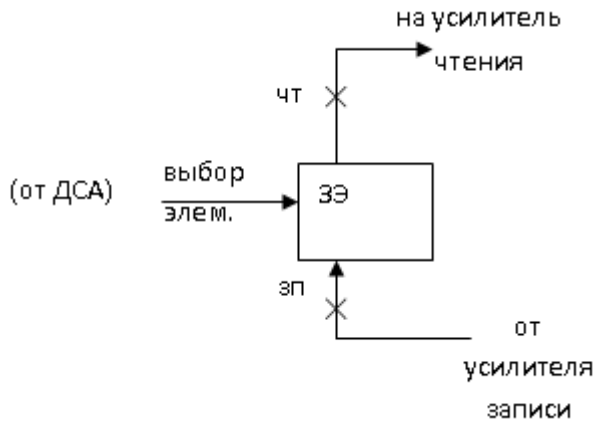


Рис. 3.16.– Вид запоминающего элемента

Каждый ЗЭ адресуемой ячейки выбирается сигналом выбора от DCA. Основной недостаток ЗУ типа 2D – сложная схема селекции и, следовательно, большие затраты оборудования на её реализацию. Сложность селектора пропорциональна емкости ЗУ 2^m=E (пример: m=16, 2^{16}=64K), т.е. пропорциональна количеству выходов DCA (выходов DMS, входов MS). В силу этого недостатка ЗУ типа 2D используются лишь в тех случаях, когда количество ячеек памяти невелико, что как раз и характерно для РОН: m=3 (4, 5) обычно.

3.7. Кэш-память

Под кэш-памятью (кэш) понимается буферное оперативное, более высокого быстродействия, чем основное ОЗУ, запоминающее устройство, имеющее в своем составе схему быстрого поиска информации. Различают кэш первого уровня (L1), или внутренний, и второго уровня (L2), или внешний, подключаемый к шине процессора. Внутренний кэш работает с быстродействием МП, внешний – синхронно с устройствами шины процессора. Внутренний кэш является буфером между регистрами МП и ОЗУ. Если имеется внешний кэш, то он является буфером между внутренним кэш и ОЗУ. Кэш-2 может быть линейным, подключаемым к одной системной шине МП, или тыльным, выполняющим обмен по двум шинам, одна из которых осуществляет связь с МП на более высокой частоте, чем другая процессорная с ОЗУ. Иногда ЭВМ имеет в составе и кэш-3, которая является буфером между кэш-2 и ОЗУ. Емкость памяти внутреннего кэш определяется типом МП (8, 16 Кб и более), а память внешнего кэш может наращиваться при использовании Windows 95 до 512 Кб или до 2 Мб с 32 разрядными ОС, как OS/2, Windows NT и UNIX. Кэш строится на элементах SRAM с временем доступа 4.5 – 12 нс и схемах сравнения, и поэтому ощутимо дороже динамических ОЗУ.

Для обмена информацией между кэш и ОЗУ используют 3 способа:

- со сквозной записью;
- со сквозной буферной записью;
- с обратной записью.

При сквозной записи результат операции передается МП одновременно в кэш и ОЗУ. При низком быстродействии ОЗУ МП простаивает, ожидая весь цикл записи. Увеличить быстродействие обмена удается при использовании сквозной буферной записи, когда МП ждет записи только в кэш, а для записи в ОЗУ информация передается в буферные регистры шинного интерфейса, а при свободной шине процессора затем передается в ОЗУ. Приоритет отдается операциям записи из МП в ОЗУ, и этот способ часто используются в алгоритмах, требующих синхронной смены информации в ОЗУ.

Для многих задач адреса данных и следующих команд расположены рядом, при этом частично данные являются промежуточными и располагаются в одних и тех же ячейках памяти при выполнении различных операций. В таком случае большее быстродействие обмена обеспечивает способ с обратной записью. При таком способе данные из строк кэш передаются в ОЗУ только при изменении информации на новую запись из ОЗУ, когда стираемая строка в кэш обновляется МП при выполнении программы. Недостатком способа является ″старение″ информации в ОЗУ в процессе вычислений. Полное соответствие информации с кэш достигается только после решения задачи, когда кэш копируется в ОЗУ. При считывании в кэш все способы обмена работают одинаково.

Для повышения быстродействия поиска информации в кэш используется ассоциативная или адресно-ассоциативная адресация. При ассоциативной адресации в качестве признака поиска (ключа, тэга) используется весь физический адрес. При адресно-ассоциативной адресации старшие разряды физического разряда используются для тэга, а младшие являются адресом ячейки внутри множества адресуемых ячеек. На рисунке 3.17 приведена структура ассоциативной памяти. Физический адрес (ФА) с ША в виде, например, кода 011…10 подается как тэг на схемы сравнения *D*1 … *DN*+1. Наличие идентичности тэга с содержимым ячейки блока тэга (БТ) (например, с номером 0) включает соответствующую схему сравнения, выходной сигнал которой становится равен 1. Выходной сигнал 1 схемы сравнения (1 с D1) коммутирует соответствующую строку массива (0) на шину данных (ШД). Недостатками схемы являются значительные затраты памяти, требуемые для построения блока тэгов, и многочисленные поразрядные схемы сравнения.

Число схем сравнения обычно равно числу строк. Преимущество – высокое быстродействие поиска из-за отсутствия дешифратора адреса.

Снизить аппаратные затраты можно при помощи способа адресно-ассоциативной адресации (кэш прямого отображения), показанной на рис. 3.8. Кэш содержит массив данных из 0, 1 …, *N* множеств. Каждое множество состоит из 0, 1, …, (2^{n-m}-1) строк (ячеек).

В такой структуре кэш младшие разряды (смещение в команде или странице) используются для определения номера множества БТ с помощью DCA и номера строки с помощью дешифратора строк (DCC). Сигнал с одного из выходов DCA, равный 1, подключает тэг, обслуживающий множество, к схеме сравнения D1, на другие входы которой подаются старшие *k* разрядов ключа физического адреса с ША.

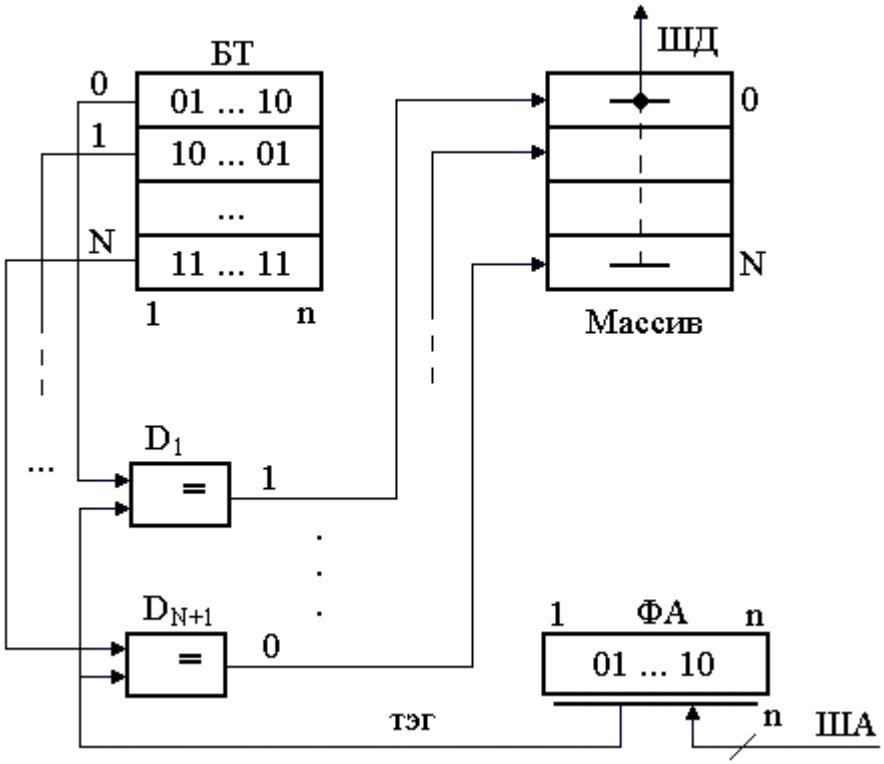


Рис.3.17.– Ассоциативный кэш

При срабатывании D1 происходит коммутация содержимого возбужденной строки соответствующего множества массива на ШД. Строка 2^{n-m}-1 множества 1 передается на шину данных, т.к. в подключенном DCA множестве 1 тэг равен старшим разрядам ФА и выход DCC 2^{n-m}-1 равен 1. Недостатком такого 65ЭШа является невозможность одновременного размещения информации из ячеек, имеющих разные тэги и одинаковое смещение в разрядах *k*+1, …, *n* ФА. Чтобы исключить этот недостаток, на практике используют несколько множеств (направлений) и соответствующее им число блоков тэгов и схем сравнения.

На рис. 3.18 приведена структура кэш *i*486, выполненная по схеме четырех направлений 0, 1, 2, 3. Каждое направление содержит множество из

128 строк размерностью по 16 байт. Запись в кэш осуществляется через буферные регистры интерфейса процессора. Для этого из МП в ОЗУ по ША передается адрес младшей ячейки в строке и осуществляется ее чтение в буфер интерфейса через шину (ШД) процессора, затем трижды выполняется инкремент адреса A(31,2) + 1 и трижды – чтение выше адресуемых ячеек. По окончании такой пакетной передачи содержимое буферного регистра записывается в строку множества, объявляемую недостоверной. Номер строки множества определяет адрес A(10,4), направление вычисляется в блоке LRU. По адресу A(10,4) и направлению, указанному блоком LRU, в блок тэгов записывается <A(31,11)> – старшие 21 бит физического адреса данных. Так как информация читается 32-битными ячейками по адресам, где A(1,0) = 00, то МП поддерживает при чтении только выровненные данные. Для повышения быстродействия чтения пакетный обмен производится по 16 байт.

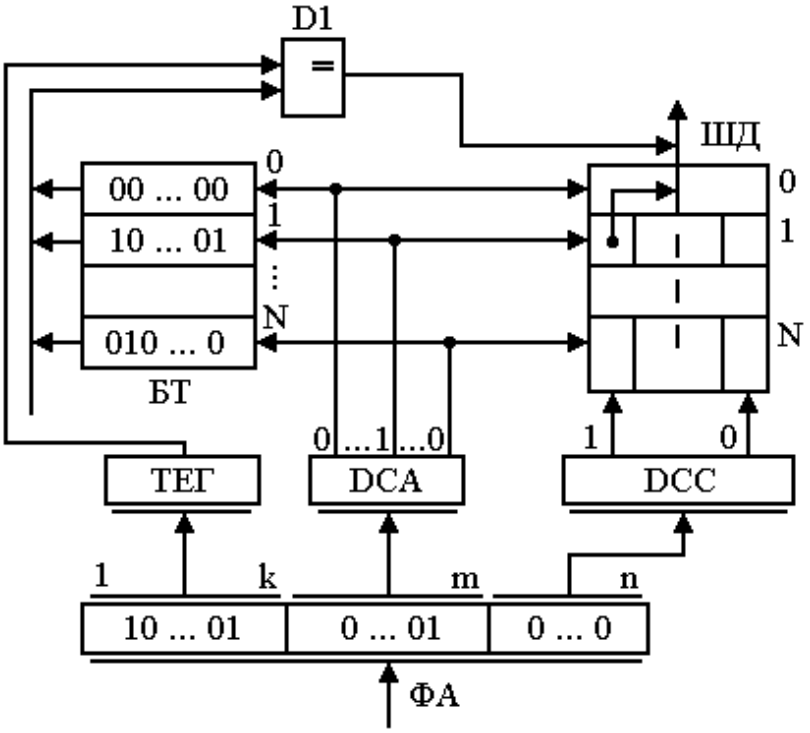


Рис.3.18.– Кэш прямого отображения

Чтение из кэш осуществляется по физическому адресу <ФА(31,0)>, поступающему из устройства сегментации или страничного преобразования по ША. Он позволяет извлекать информацию с точностью до байта, используя дешифратор строки DCC, на вход которого подаются младшие четыре бита адреса <ФА(3,0)>. При попадании, когда выход DCA подключает строки БТ к схемам сравнения D1 ÷ D4, под действием одного из сигналов *а, б, в, г* строка одного из направлений массива передается в буферный регистр, из которого, с использованием DCC, требуемый байт (слово, двойное слово) передается на внутреннюю ШД процессора. Часто

строка из буферного регистра передаётся как 16-байтный код в устройство предвыборки команд, минуя ВШД. При промахе, если не срабатывает одна из схем сравнения D1 ÷ D4, МП включает режим чтения внешнего кэш или ОЗУ по данному ФА.

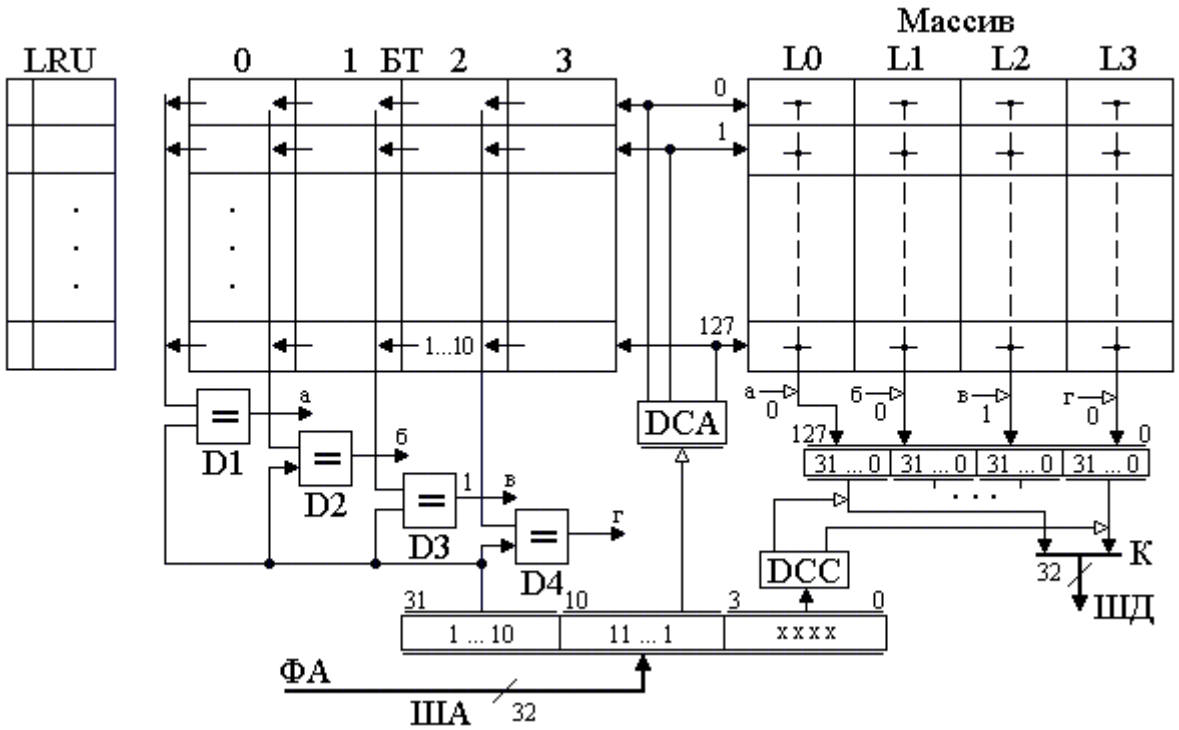


Рис. 3.19.– Внутренний кэш *i*486

Блок LRU содержит память также из 128 строк, по 7 бит в строке. Три бита каждой строки LRU <B0, B1, B2> служат для определения правила замены *Lij* *j*-й строки *i*-го направления при записи в кэш. Эти биты изменяются в одной из соответствующих строк при каждой записи или попадании в кэш следующим образом:

- если при чтении было обращение к строке *L*0*j* или *L*1*j*, то *B*0*j* = 0;
- если данные извлекались из *L*0*j*, *L*1*j* и оказались в *L*0*j*, то *B*1*j* = 1, а если из *L*1*j*, то *B*1*j* = 0;
- если данные извлекались (записывались) из *L*2*j*, *L*3*j* и оказались в *L*2*j*, то *B*2*j* = 1, а если из *L*3*j*, *B*2*j* = 0.

Состояния <B0, B1, B2> используются в алгоритме псевдо-LRU для определения номера направления и замены данных при промахе или опережающей выборке: <B0, B1, B2> = <0, 0, X> заменяется строка L0; <0, 1, X> – L1; <1, X, 0> – L2, <1, X, 1> – L3, где X ∈ {0, 1}.

Четыре оставшиеся бита памяти строки LRU используются для определения достоверности данных, хранимых в кэш. При очистке кэш или сбросе МП все биты достоверности устанавливаются в ″0″. При промахе и новой записи бит, соответствующий направлению и строке, устанавливается

в ″1″. При вычислениях МП может объявить часть строк недостоверными и соответствующие биты B3-B6 LRU установить в ″0″. Эти недостоверные строки впоследствии используются в первую очередь для записи новых данных в кэш. Если все биты B3-B6 = 1, то замена данных в кэш определяется в соответствии с состоянием <B0, B1, B2>. Если данные в кэш становятся частично достоверными, то соответствующие биты достоверности не устанавливаются в ″1″.

3.8. ПЗУ

Постоянные запоминающие устройства (ПЗУ или Read Only Memory – ROM), которые также часто называют энергонезависимыми (или Non Volatile Storage), обеспечивают сохранение записанной в них информации и при отсутствии напряжения питания. Конечно, под такое определение подпадают и память на жестких и гибких дисках, и компакт диски, и некоторые другие виды ЗУ.

Однако, говоря о постоянных ЗУ, обычно подразумевают устройства памяти с произвольным адресным доступом. Такие ЗУ могут строиться на различных физических принципах и обладать различными характеристиками не только по емкости и времени обращения к ним, но и по возможности замены записанной в них информации.

3.8.1. Разновидности постоянных ЗУ

К началу 2000-х годов наибольшее распространение получили полупроводниковые ПЗУ, элементы памяти которых используют различные модификации диодов и транзисторов и изготавливаются по интегральной технологии.

Непосредственными предшественниками таких ЗУ были магнитные (трансформаторные) ПЗУ, информация в которые записывалась соответствующей прокладкой (прошивкой) проводников ферритовых сердечников, что обеспечивало при требовавшихся в то время емкостях высокую надежность этих ЗУ в самых тяжелых (в электромагнитном отношении) условиях.

Известны также емкостные и индуктивные ПЗУ, в которых использовались проводники специальной формы, образующие емкостные или индуктивные связи.

В настоящее время исследуются и другие принципы реализации постоянных ЗУ, в некотором смысле возвращающиеся к магнитным и конденсаторным схемам, но на другом уровне развития технологий.

Запись информации в постоянные ЗУ, как правило, существенно отличается от считывания по способу и времени выполнения. Процесс записи для полупроводниковых постоянных ЗУ получил также название “прожига”

или программирования, первое из которых связано со способом записи, сводящимся к разрушению (расплавлению, прожигу) соединительных перемычек в чистом ЗУ.

В полупроводниковых ПЗУ в качестве элементов памяти, точнее, в качестве нелинейных коммутирующих и усилительных элементов обычно используются транзисторы. Они объединены в матрицу, выборка данных из которой производится по строкам и столбцам, соответствующим указанному адресу, так же, как и в других ЗУ с произвольным доступом. Один из возможных вариантов структурной схемы полупроводникового ПЗУ, представлен на рис. 3.21. Строго говоря, непосредственно запоминание информации в этом ПЗУ осуществляется плавкой перемычкой, а транзисторы выполняют роль ключей-усилителей. Плавкая перемычка может быть изготовлена из нихрома, поликристаллического кремния или других материалов. В зависимости от того, как именно работает усилитель считывания (в режиме повторителя или инвертора), наличие перемычки соответствует записи “1” или “0”. Разрушение перемычки (импульсом сильного тока) приводит к записи значения, обратного исходному.

Постоянные и полупостоянные ЗУ используются в ЭВМ как долговременная память для хранения констант, программ BIOS, POST, конфигурации ЭВМ и параметров устройств. ПЗУ и ППЗУ (флэш-память) могут строиться по структуре 2D емкостью до нескольких десятков Кб по схеме [1], показанной на рис. 3.20. При подаче двоичного кода физического адреса <ФА (1,n)> на ША и сигнала чтения дешифратором адреса DCA возбуждается одна из адресных (горизонтальных) шин A_i с номером $i \in \{1, 2, \ldots, N\}$, равным адресу ячейки. Шина A_i «подключает» элементы памяти $Э_{i1}$, $Э_{i2}$, …, $Э_{iR}$ к информационным (вертикальным) шинам массива $\{Э_{i,j}\}$. Если j-й элемент памяти хранит «1», то он возбуждает вертикальную шину D_j и устанавливает в «1» j-й разряд информационного регистра RGИ (j) = 1. Если он хранит «0», то j-й разряд RGИ обнуляется. С появлением сигнала чтения W = 1 содержимое регистра передается на шину данных, т.е. выполняется микрооперация ШД (1,R) = RGИ (1,R).

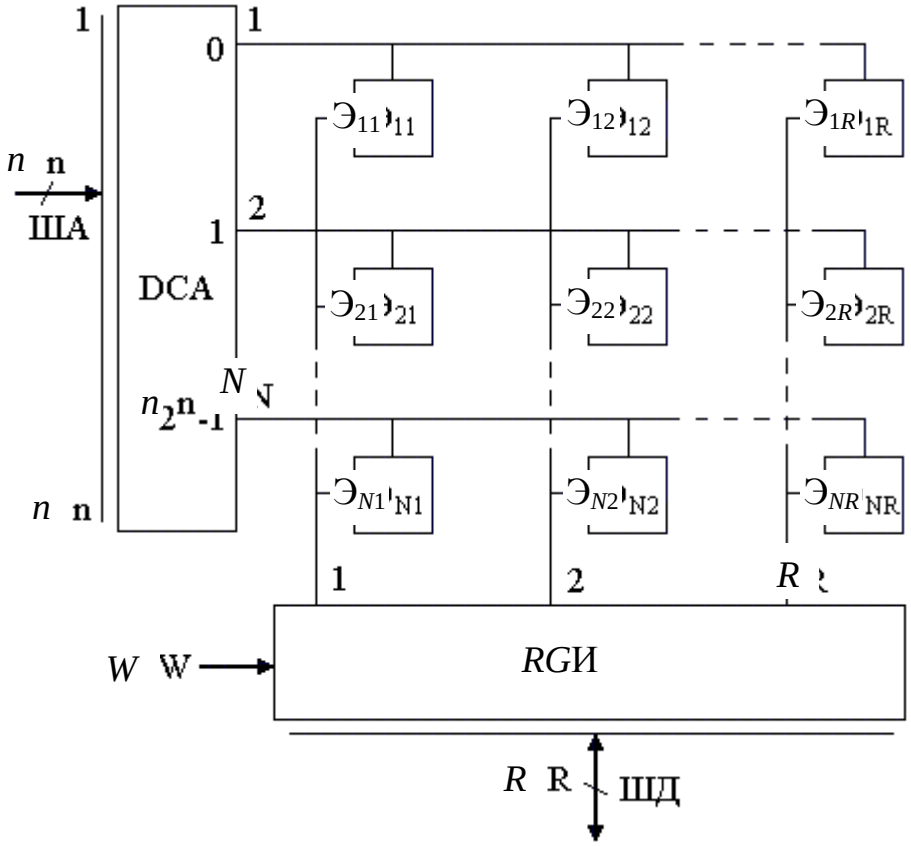
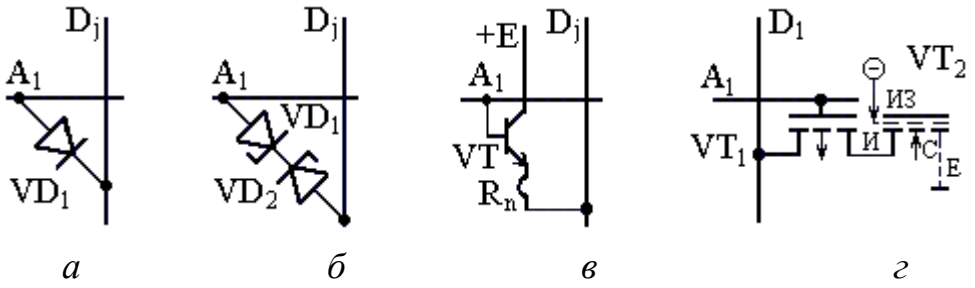


Рис. 3.20.– Схема ЗУ типа 2*D*

Режим чтения характерен для ПЗУ, элементы памяти которого не изменяют своего состояния в процессе эксплуатации и программируются один раз. Наиболее просто программирование можно осуществить распайкой диодов (рисунок 3.21, *а*). Наличие диода VD_1 между шинами A_i и D_j создает однонаправленную гальваническую связь от A_i к D_j и кодируется как «1» в *j*-м разряде *i*-й ячейки. Отсутствие диода (разрыв связи) программируется как «0». Недостатком способа ручной распайки является значительная трудоемкость «программирования», и поэтому он уже нигде не применяется.

Для автоматизации «прошивки» ПЗУ используются элементы памяти, показанные на рисунке 3.21 *б, в*.

Эти элементы при изготовлении ИС имеются на всех пересечениях адресных и информационных шин. В процессе программирования на адресной шине A_i и информационных шинах D_j последовательно устанавливается такое напряжение, когда пробивается и закорачивается один из стабилитронов (состояние логической «1») или сгорает, образуя разрыв, плавкая перемычка R_n (состояние «0»). Режим программирования путем пережигания плавких перемычек используется в ИС серии К 556 РТ, где, например, К 556 РТ5 имеет емкость 4096 бит с организацией $N \times R = 512 \times 8$ и временем выборки 70 нс.



а *б* *в* *г*

Рис. 3.21.–Элементы памяти ПЗУ и ППЗУ:
а – диодное ПЗУ,
б – ПЗУ на стабилитронах,
в – ПЗУ с плавкими перемычками,
г – ПЗУ на транзисторах с изолированным затвором

Отличие ИС ППЗУ от ПЗУ заключается в том, что они позволяют изменять в процессе эксплуатации состояния элементов памяти от нескольких десятков до нескольких тысяч раз, а в новых БИС от 100 тысяч до 1 млн циклов перезаписи. Программирование осуществляется занесением объемного заряда электронов (-) в область плавающего затвора диэлектрика, находящуюся около изолированного затвора ИЗ. Этот процесс можно пояснить с помощью рис. 3.21, *г*.

При подаче высокого напряжения E к p-n переходу стока (С) либо истока (И) происходит инжекция электронов в область плавающего затвора. После снятия потенциала E этот заряд из-за отсутствия проводимости в диэлектрике остается длительное время и притягивает дырки, создавая проводящим слой между И-С. При отсутствии заряда связи между И-С не будет. Если после программирования все стоки транзисторов VT_2 подключить к нулевому потенциалу ($\perp$), то с возбуждением шины A_i транзисторы VT_1 откроются и передадут состояние (потенциал) истока транзисторов VT_2 (при наличии заряда на D_j будет «0», при отсутствии заряда – «1»).

Для повторного программирования ППЗУ стирание зарядов осуществляется либо электрически (ИС КР 558 РР), подачей высоких напряжений обратной полярности, либо ультрафиолетовым облучением диэлектрика ИС через кварцевую крышку в корпусе (ИС КР 573 РФ). Процесс электрического стирания вызывает значительный ток от утекания заряда из области плавающего затвора под действием повышенного обратного напряжения более 10 В. Чтобы кристалл не разогревался выше допустимой температуры, ток ограничивают уменьшая число одновременно стираемых ячеек. Основной причиной ограничения числа циклов перезаписи и уменьшения времени эксплуатации ППЗУ является аккумулируемый после каждой перезаписи износ плавающего затвора транзистора, который происходит от многократного воздействия повышенного напряжения более 10 В при стирании информации. При этом происходит нарушение оксидного слоя либо накопление электронов в плавающем затворе. Стирание

информации выполняют в несколько циклов. При этом переходы И-С всех транзисторов VT_2 становятся непроводящими и все ячейки содержат «1».

Перепрограммирование ИС ведется последовательно отдельно для каждой ячейки. Сначала передается ее адрес, затем передается код записываемых данных в RGИ (1, R). Затем возбуждается шина *E* в разрядах, где кодируется «0», и пропускаются информационные разряды, где кодируются «1».

Так как накопление заряда идет медленно, запись осуществляется циклами. Длительность каждого цикла около 0.5 мс, число циклов (около 100) определяется необходимым временем занесения заряда в ИС. Принцип с ультрафиолетовым стиранием и электрической записью информации используется в ИС КР 573 РФ2 емкостью 2 Кб (N × R = 2048 × 8 (бит)) с временем выборки 0.9 мкс, временем хранения заряда 10^4 час и возможностью до 10 раз перезаписи данных.

Функционально ППЗУ (ПЗУ) представляет собой шифратор, который преобразует унитарный код с адресной шины в позиционный, подключаемый к информационной шине. В зависимости от способа объединения элементов памяти, показанных на рисунке 3.21, *г,* шифратор может реализовываться на схемах И – НЕ или ИЛИ – НЕ ($D_j = \overline{(A_1 x_{1j} \; V A_2 x_{2j} \; V \ldots V \; A_N x_{Nj})}$, j=1, R; где x_{ij} – содержимое j-го разряда, i-й ячейки). Схемы ИЛИ – НЕ обеспечивают высокое быстродействие, но имеют меньшую емкость, чем ППЗУ на И – НЕ.

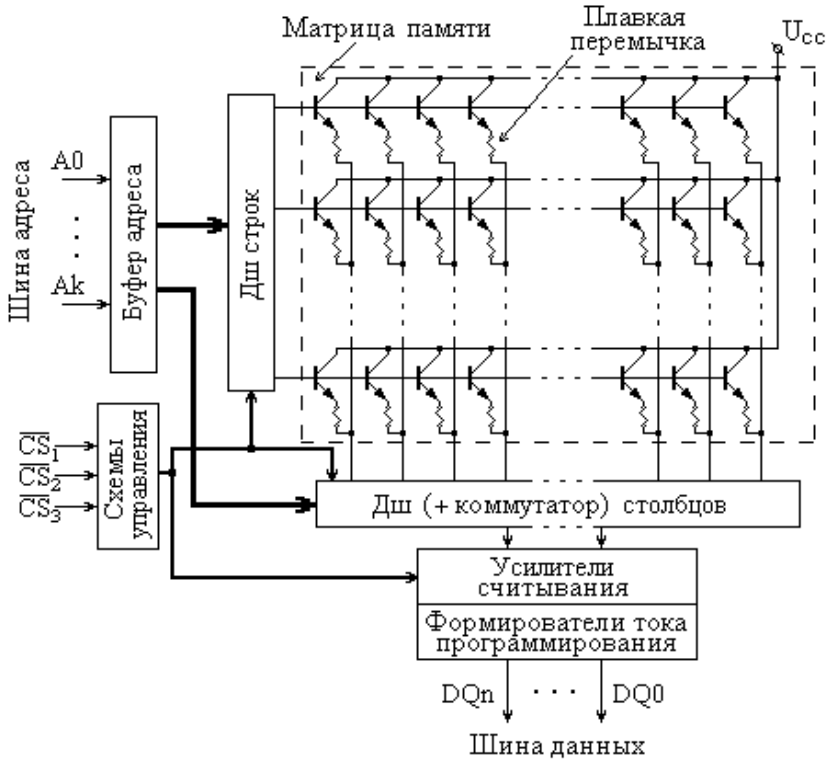


Рис. 3.22.– Вариант структурной схемы ПЗУ с однократным программированием

Различают две большие группы ПЗУ: *программируемые изготовителем* и *программируемые пользователем*.

ЗУ первой группы, называемые иначе масочными, обычно выпускаются большими партиями. Информация в них заносится в процессе изготовления этих ЗУ на заводах: с помощью специальной маски в конце технологического цикла на кристалле формируется соответствующая конфигурация соединений. Такие ЗУ оказываются наиболее дешевыми при массовом изготовлении. Их обычно используют для хранения различных постоянных программ и подпрограмм, кодов, физических констант, постоянных коэффициентов и пр.

В ПЗУ, программируемые пользователем, информация записывается после их изготовления самими пользователями. При этом существуют два основных типа таких ЗУ: *однократно программируемые* и *перепрограммируемые*.

Нетрудно вспомнить, что аналогичные разновидности имеются и у CDROM, которые, по существу, являются ПЗУ (ROM), изготавливаемыми на основе другого физического принципа.

Наиболее простыми являются однократно программируемые ПЗУ. В этих ЗУ запись как раз и производится посредством разрушения соединительных перемычек между выводами транзисторов и шинами матрицы (хотя есть и несколько иные технологии).

Изображение программируемого ПЗУ на функциональной схеме показано на рисунке 3.23.

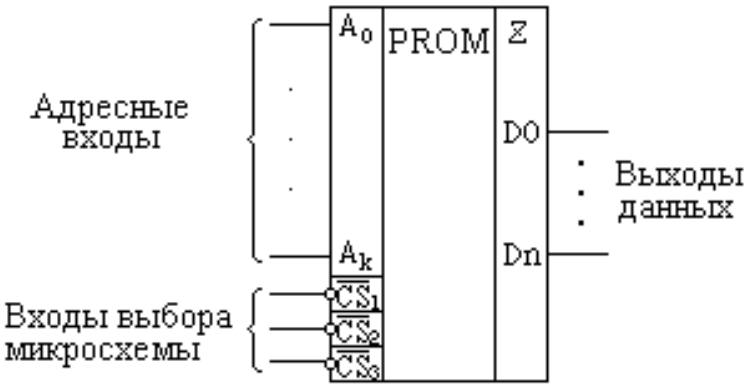


Рис.3.23.– Условное изображение программируемого ПЗУ на функциональных схемах

Перепрограммируемые ПЗУ позволяют производить в них запись информации многократно. Конечно, в таких ЗУ должен использоваться иной принцип, чем разрушение перемычек в процессе записи. Распространенные технологические варианты используют МОП-транзисторы со сложным затвором (составным или “плавающим”), который способен накапливать заряд, снижающий пороговое напряжение отпирания транзистора, и сохранять этот заряд при выключенном питании. Программирование таких ПЗУ и состоит в создании зарядов на затворах тех транзисторов, где должны

быть записаны данные (обычно “0”, так как в исходном состоянии в таких микросхемах записаны все “1”).

Перед повторной записью требуется произвести стирание ранее записанной информации. Оно производится либо электрически, подачей напряжения обратной полярности, либо с помощью ультрафиолетового света. У микросхем последнего типа имелось круглое окошечко из кварцевого стекла, через которое и освещался кристалл при стирании.

Параметры постоянных ЗУ соответствуют технологическим нормам своего времени. В начале 2000-х годов типовые емкости микросхем постоянной памяти с масочным программированием составляли порядка 32-128 Мбит, а времена обращения превышали аналогичные показатели оперативной памяти и для различных модификаций достигали до 100 нс.

3.8.2. Флэш-память

Флэш-память, появившаяся в конце 1980-х годов (*Intel*), является представителем класса перепрограммируемых постоянных ЗУ с электрическим стиранием. Однако стирание в ней осуществляется сразу целой области ячеек: блока или всей микросхемы. Это обеспечивает более быструю запись информации или, как иначе называют данную процедуру, программирование ЗУ. Для упрощения этой процедуры в микросхему включаются специальные блоки, делающие запись “прозрачной” (подобной записи в обычное ЗУ) для аппаратного и программного окружения.

Флэш-память строится на однотранзисторных элементах памяти (с “плавающим” затвором), что обеспечивает плотность хранения информации даже несколько выше, чем в динамической оперативной памяти. Существуют различные технологии построения базовых элементов флэш-памяти, разработанные ее основными производителями. Эти технологии отличаются количеством слоев, методами стирания и записи данных, а также структурной организацией, что отражается в их названии. Наиболее широко известны NOR и NAND типы флэш-памяти, запоминающие транзисторы в которых подключены к разрядным шинам, соответственно, параллельно и последовательно.

Первый тип имеет относительно большие размеры ячеек и быстрый произвольный доступ (порядка 70 нс), что позволяет выполнять программы непосредственно из этой памяти. Второй тип имеет меньшие размеры ячеек и быстрый последовательный доступ, что более пригодно для построения устройств блочного типа, например “твердотельных дисков”.

Способность сохранять информацию при выключенном питании, малые размеры, высокая надежность и приемлемая цена привели к широкому ее распространению. Этот вид памяти применяется для хранения BIOS, построения так называемых “твердотельных” дисков (*memory stick, memory drive* и др.), карт памяти различного назначения и т.п. Причем устройства на

основе флэш-памяти используются не только в ЭВМ, но и во многих других применениях.

К минусам данного вида памяти можно отнести относительно невысокую скорость передачи данных, средний объем и дороговизну устройств с большой емкостью (свыше 512 Мбайт и более).

Элементы памяти флэш-ЗУ организованы в матрицы, как и в других видах полупроводниковой памяти. Разрядность данных для микросхем составляет 1-2 байта.

Операция чтения из флэш-памяти выполняется как в обычных ЗУ с произвольным доступом (оперативных ЗУ или кэш). Однако запись сохраняет в себе некоторые особенности, аналогичные свойствам постоянных ЗУ.

Перед записью данных в ЗУ ячейки, в которые будет производиться запись, должны быть очищены (стерты). Стирание заключается в переводе элементов памяти в состояние единицы и возможно только сразу для целого блока ячеек (в первых микросхемах предусматривалось стирание только для всей матрицы сразу). Выборочное стирание невозможно.

В процессе записи информации соответствующие элементы памяти переключаются в нулевое состояние. Также, как и в ПЗУ, без стирания можно дозаписать нули в уже запрограммированные ячейки, однако необходимость в такой операции относительно редка.

Фактически при операции записи производится два действия: запись и считывание, но управление этими операциями производится внутренним автоматом и “прозрачно” для процессора.

Разбиение адресного пространства микросхемы флэш-памяти на блоки обычно бывает двух видов: симметричное и асимметричное.

В первом случае, называемом также *Flash File*, все блоки (стирание в пределах каждого из которых производится только для всего блока сразу) имеют одинаковый размер, например 64 Кбайт или 128 Кбайт. Количество блоков зависит от емкости микросхемы. Например, в микросхеме *28F128J3* (*Intel Strata Flash*) емкостью 128 Мбит (16 Мбайт) имеется 128 блоков по 128 Кбайт.

В случае асимметричной архитектуры, называемой иначе *Boot Block*, один из блоков, на которые разбито адресное пространство микросхемы, дополнительно разбивается на меньшие блоки. Например, в микросхеме *28F640C3* (*Intel Advanced+ Boot Block*) емкостью 64 Мбит выделен один загрузочный (*Boot*) блок размером 64 Кбайт, разбитый на 8 блоков параметров (*parameter blocks*) по 8 Кбайт, и 127 основных (*main*) блоков по 64 Кбайт. Причем загрузочный блок может размещаться либо в начале, либо в конце адресного пространства микросхемы.

Структурная схема флэш-памяти с асимметричной архитектурой приведена на рисунке 3.24. В этой схеме управляющий сигнал ***WP***# (*Write Protect*) используется для исключения возможности случайной записи по командам программы, а сигнал ***RP***# (*Reset/Deep Power Down*) также

применяется для управления записью, закрывая все блоки для записи при единичном уровне. Остальные управляющие сигналы аналогичны одноименным сигналам в других типах памяти. На вход $\boldsymbol{V}_{PP}$ подается напряжение, необходимое для ускорения операций стирания и записи данных.

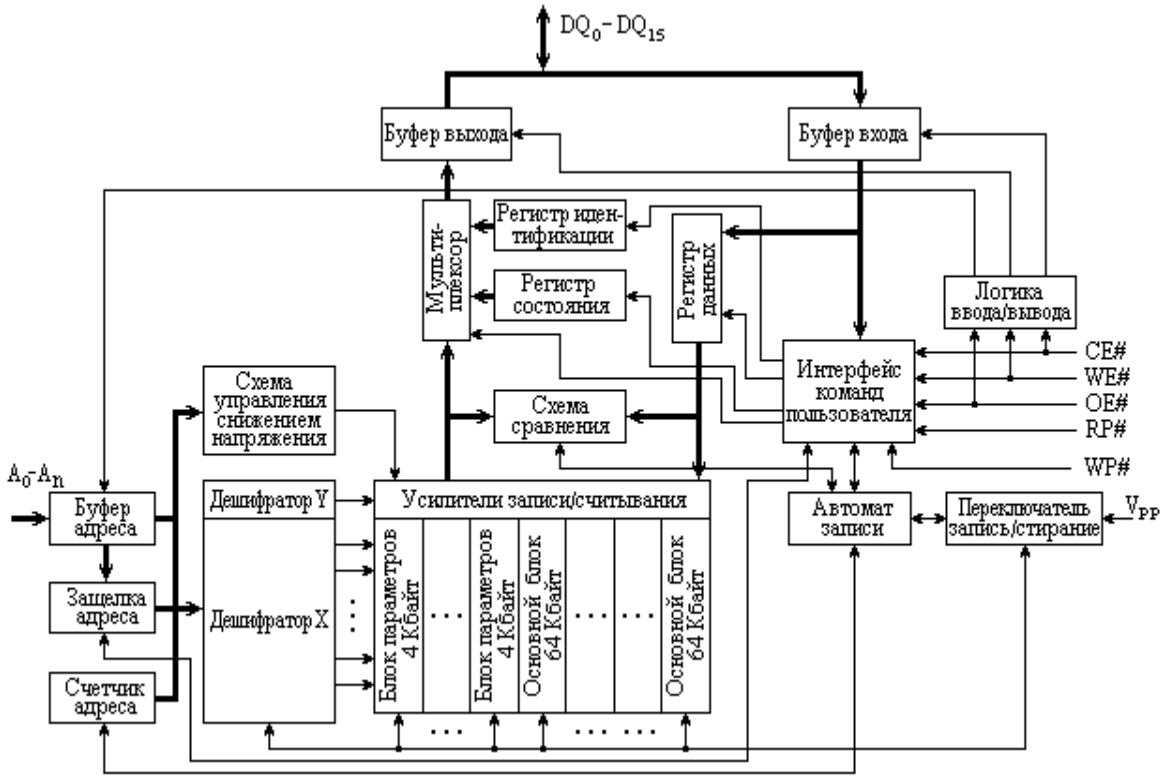


Рис.3.24.– Структурная схема флэш-памяти

Флэш-память используется для различных целей. Непосредственно в самой ЭВМ эту память применяют для хранения BIOS (базовой системы ввода-вывода), что позволяет при необходимости производить обновление последней, прямо на рабочей машине. (Надо отметить, что без особой необходимости и при отсутствии соответствующего опыта, производить такие операции не рекомендуется.)

Другим применением флэш-памяти, получившим достаточно широкое распространение, являются так называемые “твердотельные диски” (*solid-state disks*), эмулирующие работу внешних винчестеров. Такое устройство имеет габариты порядка 70×20×10 мм, подключается обычно к шине USB и состоит из собственно флэш-памяти, эмулятора контроллера дисковода и контроллера шины USB. При включении его в систему (допускается “горячее” подключение и отключение) устройство с точки зрения пользователя ведет себя как обычный (съемный) жесткий диск. Конечно, производительность его меньше, чем у жесткого диска.

Более общее применение флэш-память находит в различных модификациях карт памяти, которые используются не только в компьютерах разных классов, но и в цифровых видео- и фотокамерах, плеерах, телефонах,

музыкальных центрах и другой медиатехнике. Причем такая карта может также быть и сменной картой в твердотельном диске.

3.9. Компоновка ОЗУ основных моделей СМ ЭВМ.

Данный вопрос рассмотрим на примере модулей памяти модели СМ 1810. Модуль оперативный запоминающий МОЗ 256 СМ. Он предназначен для приема, хранения и выдачи оперативной информации в качестве встроенной оперативной памяти в составе микроЭВМ СМ 1810. Модуль имеет следующие технические характеристики:

Объем – 256 Кб

Разрядность – 8 и 16 бит

Порядок обращения – произвольный

Выполняемые операции – запись слова (ЗПС), чтение слова (ЧТС), запись байта (ЗПБ), чтение байта (ЧТБ)

Цикл обращения – при операциях ЧТС, ЗПС, ЧТБ не более 0,7 мкс, при операции ЗПБ не более 1,4 мкс

Модуль обеспечивает коррекцию одинарной и обнаружение двойной ошибки. На рис. Показана структурная схема модуля. Узел приема осуществляет формирование адреса обращения к требуемой ячейке памяти при обращении к модулю со стороны интерфейса И41, узел обработки данных осуществляет прием и выдачу данных на (из) интерфейса И41. В его состав входит корректор, обеспечивающий при операциях записи формирование контрольных разрядов накопителя. При операциях чтения корректор формирует признаки одинарной и двойной ошибки и в случае одинарной ошибки производит коррекцию данных и выдачу их через соответствующий буферы на интерфейс И41. Узел управления формирует сигналы управления другими узлами модуля и соответствующую диаграмму. В его состав входит контроллер памяти КМ1810ВТ03, осуществляющий формирование управляющих сигналов для динамических микросхем памяти, прием и мультиплексирование адресов строки и столбца, а также формирует режим регенерации.

Узел накопителя предназначен для записи, хранения и выдачи информации представляет собой матрицу микросхем памяти К565РУ5 (64К х 1). Матрица содержит два ряда по 22 микросхемы. Разряд данных включает в себя по одной микросхеме из каждого ряда; таким образом, в матрице всего по 16 информационных и 6 контрольных разрядов. Полная емкость накопителя 128К х 22 бит, где К=1024бит

Узел портов диагностики осуществляет прием и выдачу информации о диагностики модуля и состоит из портов ввода-вывода, в которых хранится информация о работоспособности модуля.

В состав СМ1810 входит еще один модуль оперативной памяти МОЗ 4М, который отличается от МОЗ256 большей емкостью (до 4 Мбайт). Остальные

параметры МОЗ 4М аналогичны МОЗ 256. МОЗ 4М состоит из пяти плат, которые устанавливаются в соответствующие места 1810.40 и 1810.41. Из них одна плата выполняет функции контроллера памяти, остальные четыре платы – функции накопителя. Платы накопителя полностью взаимозаменяемы и служат для наращивания накопителя блоками по 1 Мбайт до 4х. Минимальная емкость МОЗ 4М – 1 Мбайт.

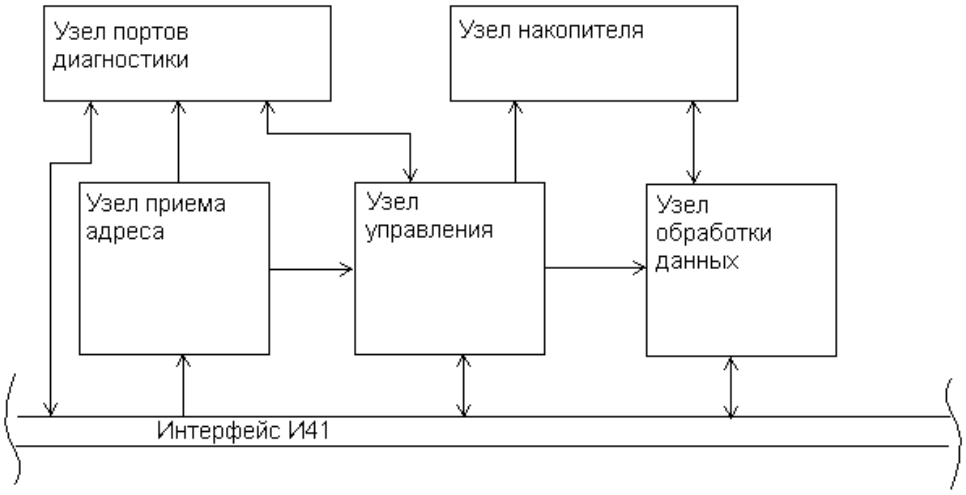


Рис. 3.25.– Компоновка ОЗУ

3.10. Регенерация информации в динамических ЗУ. Динамическая память RDRAM

Независимо от того, какова конкретная модификация динамической памяти, запоминающие конденсаторы ее запоминающих элементов разряжаются из-за наличия токов утечки. Постоянная разряда, как известно, зависит от емкости запоминающего конденсатора и сопротивления цепи тока утечки и может различаться для разных модификаций. Время, в течение которого информация сохраняется в элементе памяти, составляет до нескольких десятков миллисекунд.

Это приводит к необходимости периодического (с периодом не больше, чем время сохранения информации) восстановления зарядов емкостей. Такая процедура и получила название регенерации (*refresh*) динамической памяти. Выполняется она одновременно для целой строки матрицы (банка) элементов памяти, поскольку регенерировать информацию по элементам или по словам (по 8 байт) слишком долго.

Действительно, если даже считать, что регенерация выполняется одним длинным пакетным циклом, то для микросхем памяти PC2700, в которых на одну передачу данных приходится 3 нс (165 МГц × 2), при емкости 256 Мбит на поэлементную регенерацию всех элементов потребуется $3 \times 10^{-9} \times 228 =$ 0,8 с, а на регенерацию по словам, точнее, ячейкам, разрядность которых не может быть выше разрядности микросхемы (как правило, не более 16 бит),

потребуется в 16 раз меньше времени, т.е. 50 мс. Учитывая, что для таких микросхем максимальный период регенерации T_{REF} = 64 мс, поэлементная регенерация оказывается принципиально невозможной, а регенерация по словам будет занимать более 75 % времени работы памяти.

Количество же строк в одном банке в данной микросхеме составит 8192, а построчная регенерация в таком случае будет занимать всего 24,5 мкс или около 0,5 % времени.

Конечно, если ко всем строкам памяти за время, не превышающее T_{REF}, были бы выполнены обращения, то обновлять информацию не было бы нужно, так как при обращении (не только записи, но и чтении) заряд на запоминающих емкостях полностью восстанавливается.

Распределить циклы регенерации строк по полному периоду регенерации можно различными способами. Рассмотренный выше вариант *пакетной* регенерации, при котором все циклы регенерации строк группируются в начале или в конце периода, хотя и может быть более экономичным по времени из-за отсутствия дополнительных переключений, не является достаточно удобным, так как блокирует работу памяти на относительно длительный интервал времени. Поэтому чаще применяют так называемую *распределенную* регенерацию (иногда используется термин “синхронная”), при которой циклы регенерации строк равномерно распределяются по периоду.

Длительность периода регенерации не обязательно устанавливается равной максимальному значению. Например, в ряде ПЭВМ для управления регенерацией часто используют сигналы одного из счетчиков (счетчик 1) системного таймера, на который поступают сигналы от кварцевого генератора (частотой 14.31818 МГц), установленного на системной плате. Этот счетчик вырабатывает импульсы примерно каждые 15 мкс, и эти импульсы могут использоваться для запуска регенерации. Однако для памяти большого объема такая частота оказывается недостаточной.

Возможны и более сложные схемы регенерации: пакетная с возможностью прерывания пакета или скрытая – во время свободных циклов памяти, если таковые имеются.

За последовательностью циклов регенерации строк следит контроллер памяти, который может организовывать *очередь* из этих циклов. Этот же контроллер может формировать адреса строк для регенерации, но часто эти адреса формируются внутренним счетчиком, имеющимся в самой микросхеме динамической памяти. Известна также квазистатическая память, в которой регенерация полностью контролируется внутренней логикой микросхемы и не требует никаких внешних сигналов.

Собственно циклы регенерации также могут различаться по порядку выполнения и запускающим их управляющим сигналам.

Наиболее распространенным вариантом является цикл, при котором изменяется порядок подачи адресных стробов. Если в обычном цикле чтения или записи сигнал строба адреса строки ***RAS***# предшествует стробу адреса

столбца ***CAS***#, то рассматриваемый цикл регенерации запускается при одновременной подаче низких уровней обоих этих сигналов для синхронной памяти, а для асинхронной – спадающий фронт сигнала ***CAS***# предшествует отрицательному фронту сигнала ***RAS***#. Соответствующие временные диаграммы показаны на рисунке 3.26. В обоих случаях этот способ регенерации называется CBR (*CAS Before RAS*), хотя в синхронной памяти он также называется командой автоматической регенерации (*Auto Refresh Command*).

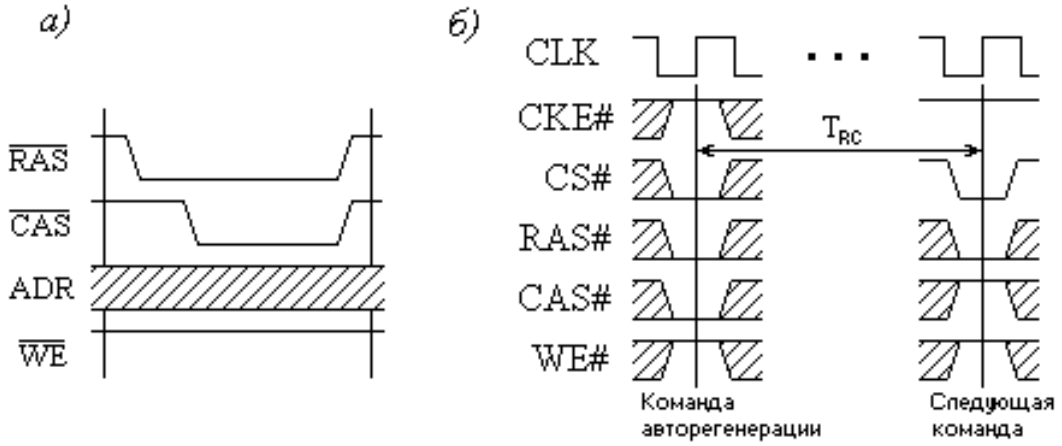


Рис.3.26.– Временные диаграммы регенерации типа CBR в асинхронной а) и синхронной б) динамической памяти

Другим вариантом запуска регенерации является подача только одного сигнала ***RAS***# без последующего ***CAS***#. Такой вариант называется ROR (*RAS Only Refresh*), а адреса регенерируемых строк формируются контроллером памяти и подаются на адресные входы микросхемы.

Память RDRAM (*Rambus DRAM*) построена на таких же элементах памяти, как и рассмотренные выше виды динамической памяти. Она относится к тому направлению разработок, в котором производительность памяти (ее пропускная способность, вычисляемая как произведение разрядности шины данных и частоты передачи по ней) достигается за счет увеличения частоты, при уменьшении разрядности шины данных. Последнее обстоятельство способствует снижению взаимных помех от проводников шины, особенно сказывающихся на высоких частотах.

Таким образом, для этого типа памяти характерен свой интерфейс, существенно отличный как логически, так и электрически от интерфейса асинхронных и синхронных DRAM. Шина данных RDRAM имеет 16 разрядов и работает на частоте 400 Мгц и выше, используя сдвоенные передачи данных по обоим фронтам синхроимпульсов (как и DDR память). С учетом этого RDRAM обеспечивает пропускную способность 1600 Мбайт/с, что, по сравнению с DDR SDRAM, не так уж и много.

Определенное противостояние фирмы разработчика памяти RDRAM с разработчиками и изготовителями памяти SDRAM, имевшее место в конце 1990-х годов, привело к отказу многих производителей от использования RDRAM.

Память RDRAM, структура которой показана на рисунке 3.27, включает в себя контроллер (RMC — *Rambus Memory Controller*), собственно микросхемы памяти, генератор синхросигналов (DRCG — *Direct Rambus Clock Generator*), источник питания и терминаторы, исключающие отражение сигналов на концах шин.

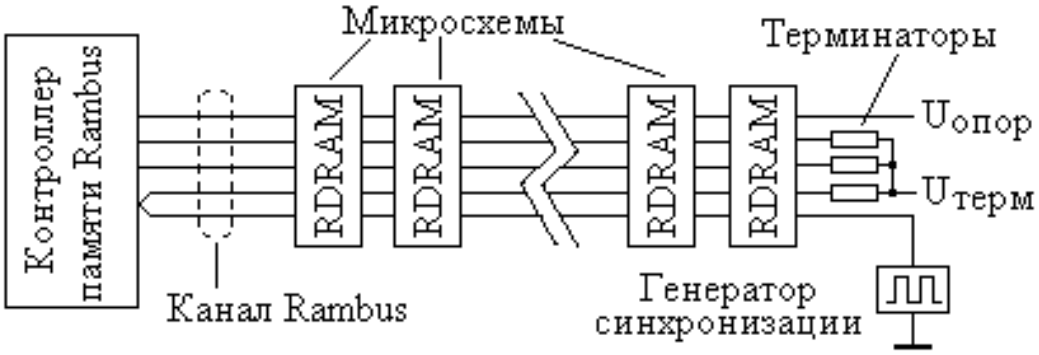


Рис.3.27.– Структура интерфейса памяти Rambus

Контроллер позволяет использовать различные микросхемы в одном канале, суммируя общую емкость и банки памяти по всем микросхемам. Причем все микросхемы имеют многобанковую организацию (до 32 банков в 256-Мбитных микросхемах). Однако их архитектура может быть различной: со сдвоенными банками (*doubled*), с разделенными банками (*splitted*) и с независимыми банками (*independent*). Эти различия определяют особенности параллельной работы банков.

Во всех случаях разрядность данных микросхемы – 16 бит. Ядро (матрица) элементов памяти, разделенное на банки, имеет построчную организацию, в которой каждая строка разделена на так называемые “двойные восьмерки” (*dualocts* в терминологии Rambus) , состоящие из 16 байтов каждая. Например, микросхема памяти емкостью 256 Мбит может разделяться на 16 банков по 2 Мбайта, каждый их которых имеет 1024 строки, содержащих по 128 16-байтных двойных восьмерок. Такие двойные восьмерки представляют собой физически минимально адресуемые (внутри микросхемы) единицы данных.

Учитывая высокую частоту работы интерфейса Direct Rambus (именно таково его полное название, но слово Direct часто для краткости опускают), к его физической реализации предъявляются довольно жесткие требования. В частности, его линии должны идти строго параллельно и заканчиваться терминаторами (для подавления отраженных сигналов). Количество слотов для установки модулей памяти в каждом канале не может превышать трех,

причем незадействованные слоты должны быть заполнены модулями-заглушками (*dummy modules*). В свою очередь, один модуль памяти может включать в себя от одной (на практике – от четырех) до 16 микросхем памяти при общей полной нагрузке в 32 устройства на канал. Регламентируется даже то, сколько раз модуль можно вставлять и вынимать из слота – до 25 раз.

Канал памяти имеет три шины: 3-битную шину строк ROW, 5-битную шину столбцов COL и двухбайтовую шину данных, состоящую из двух половинок – DQA и DQB. Кроме того, имеются также линии синхронизации, управляющих сигналов и напряжений питания. Интерфейс предусматривает параллельно-последовательную передачу данных пакетами из восьми посылок (передаваемых за 4 такта шины, т.е. за 10 нс при частоте 400 Мгц). Пакет строк, таким образом, состоит из 24 бит, столбцов – из 40 бит, а данных – из 16 байт (по 8 или 9, в случае контроля, бит каждый). Следует помнить, что это не те пакетные передачи, которые осуществляются в интерфейсе BEDO или SDRAM.

Допускается наращивание количества каналов, или, как говорят, масштабируемость памяти. Ведь разрядность данных системной шины данных, например, в ПЭВМ с процессорами семейства P6 – 64 разряда. К одному контроллеру можно подключать до 4-х каналов. Поэтому известны и модули с разрядностью более 16.

Временные диаграммы операций чтения и записи, называемых в RDRAM *транзакциями*, представлены на рисунках 3.28 и 3.29, где CFM и CTM означают сигналы синхронизации от микросхем и от контроллера.

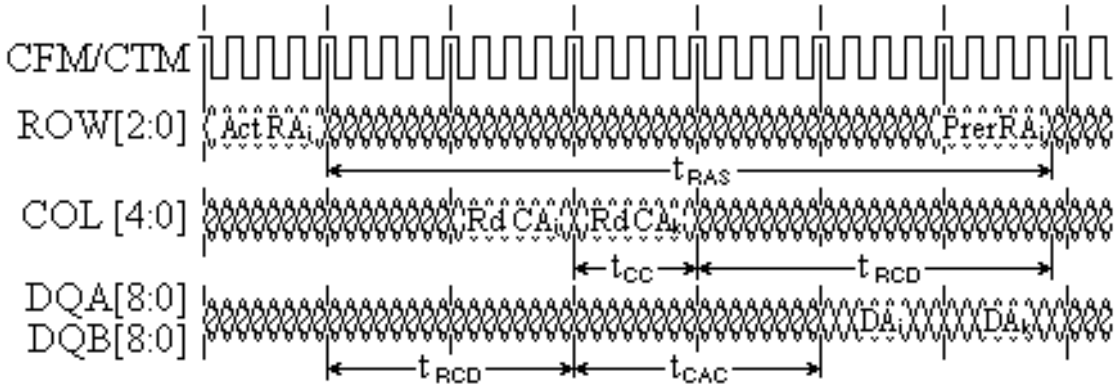


Рис.3.28.– Временная диаграмма чтения двух слов из памяти RDRAM

Как видно из рисунка 3.28, транзакция чтения начинается с команды активации (*Act*) банка, содержащейся в пакете ROW со строкой адреса a_i. По прошествии времени задержки t_{RCD} пакетом COL с колонкой адреса a_i выдается команда чтения (*Rd*) колонки. Адрес, передаваемый в пакете ROW, указывает микросхему, банк и строку, а адрес, передаваемый в пакете COW, указывает микросхему, банк и колонку.

После этого с задержкой t_{CAC} микросхема выдает прочитанные данные. Отсчет времени на линиях ROW и COL производится по отношению к

моментам окончания пакета, а на линиях данных – по отношению к началу пакета. Далее с задержкой t_{CC} выдается вторая команда чтения (колонки) с адресом a_k.

Затем подается команда подзаряда (*Prer*) банка, которая должна быть подана не ранее времени t_{RAS} после команды активации банка (команда активации в любой синхронной динамической памяти, не только RDRAM, разрушает информацию, записанную в активируемой строке, разряжая все конденсаторы, поэтому их и приходится восстанавливать командой подзаряда банка). Кроме того, команда подзаряда не должна выдаваться ранее времени t_{RPD} после предшествующей команды чтения.

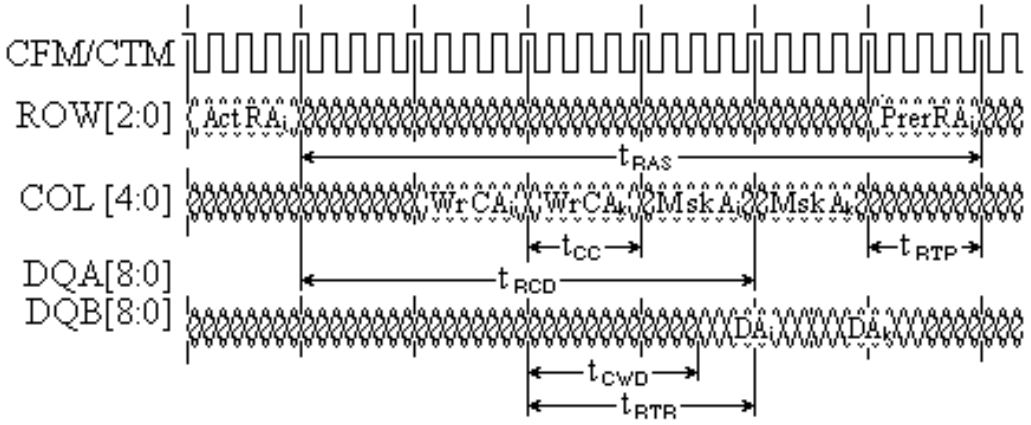


Рис. 3.29.– Временная диаграмма записи двух слов в память RDRAM

Транзакция записи, представленная на рисунке 3.29, начинается также, как и транзакция чтения, командой активации банка. За ней следует команда записи (*Wr*) колонки, подаваемая через интервал времени, равный $t_{RCD} - t_{RTR}$ (времена отсчитываются от концов пакетов), данные для которой поступают на шины данных с задержкой t_{CWD}, что отличается от SDRAM памяти, в которой такой задержки нет. Далее с задержкой t_{CC} по отношению к первой команде выдается вторая команда записи колонки с адресом a_k, а затем и данные для нее.

Спустя время задержки записи t_{RTR} может быть подана необязательная команда маски записи (*Msk*), позволяющая производить побитное маскирование записи данных. Если команда маски не подается, то в этих же тактах данные записываются полностью. Наконец, спустя время t_{RTP} после последней записи подается команда подзаряда строки банка.

Как видно, эти транзакции выполняются, в целом, аналогично тому, как и соответствующие операции в SDRAM. Однако имеется и ряд отличий.

Во-первых, транзакции RDRAM обрабатывают только одну передачу, т.е. 2 байта, а не пакет (от 8 до 64 байт как в SDRAM). Это позволяет упростить протокол шины, обеспечивая ее производительность за счет высокой частоты шины.

Во-вторых, передачи адресов и данных выполняются, как указано выше, параллельно-последовательно (занимая по четыре такта шины каждая), что показано на рис. 22 и 23 соответствующими переключениями сигналов. Пакеты, передаваемые по адресным шинам (ROW и COL), могут иметь различное назначение, задавая либо собственно адрес, либо команду. Сами команды (активация (строки) банка, запись, чтение, подзаряд и др.) аналогичны командам памяти SDRAM типа.

В-третьих, имеются особенности синхронизации для транзакций чтения, которые должны компенсировать различную физическую удаленность модулей памяти от контроллера и обеспечить одновременное поступление данных к контроллеру. Для этого приходится устанавливать различную задержку выдачи данных относительно адреса столбца для микросхем, находящихся на разном расстоянии от контроллера.

Также, как и SDRAM, память RDRAM допускает конвейерную обработку различных обращений. При полной занятости шины на ней может присутствовать до четырех транзакций, что при обращениях по последовательным адресам может обеспечить до 100% использования полосы пропускания шины данных.

3.11. Пакетное ОЗУ SLDRAM

В начале 1997 г. 12 производителей различных фирм объединились, чтобы создать более дешевую, чем Direct RDRAM, быстродействующую память емкостью 16, 64, 256 Мб для использования в ПК. Память нового типа с протоколом передачи данных на тактовой частоте более 400 Мгц получила название SLDRAM (Sync Link DRAM). Технология SLDRAM представляет собой следующий эволюционный шаг в развитии классического ядра DRAM.

В SLDRAM используется 16-разрядная шина данных, работающая на тактовой частоте 400 МГц. Аналогично DDR SDRAM и RDRAM, передача данных осуществляется на обоих фронтах тактового сигнала и пропускная способность SLDRAM составляет 16 бит × 400 МГц × 2 = 1.6 Гб/с. Она может быть увеличена за счет повышения тактовой частоты и разрядности (до 64 бит) системной шины памяти.

В SLDRAM использована основательно переделанная архитектура синхронной памяти: адреса, команды, сигналы управления передаются в пакетном режиме по однонаправленной шине. Одновременно с ними по другой двунаправленной шине DataLink в пакетном режиме передаются данные. Величина всего пакета может равняться странице. Так как пропускная способность обеих шин одинакова, то переключение на любую страницу ОЗУ осуществляется без потери производительности.

Набор команд у SLDRAM увеличен, что облегчает работу контроллера. Команда представляет собой четыре 10-битных пакета и содержит всю

информацию о массиве данных для проведения следующей операции в ОЗУ, что повышает производительность SLDRAM.

3.12. ОЗУ с виртуальными каналами обмена

Кроме увеличения частоты обмена, существуют другие способы повысить интенсивность работы SDRAM. Один из них – использование технологии обмена с виртуальным каналом VCM, разработанной фирмой NEC. В основе VCM лежит принцип снижения длительности задержки в обмене данными за счет увеличения числа каналов обмена с разными областями ОЗУ, а также снижение энергопотребления модулей памяти. Добиться выполнения этой задачи удалось следующим образом.

В обычном ОЗУ для обмена данными по ШД любым системным устройствам (контроллеры PCI, IDE или AGP, кэш МП) менеджер памяти делает запрос, включающий адрес, размер блока данных и т.д., как показано на рис. 3.30. Причем если все эти действия ведутся одновременно несколькими устройствами, то выполнить обмен данными по этим запросам невозможно из-за наличия только одного буфера обмена ОЗУ с системной шиной.

В VCM каждому устройству назначается свой высокоскоростной виртуальный канал, учитывающий специфические характеристики его запросов, в том числе и обеспечивающей функции кэширования. Менеджер памяти закрепляет устройства обмена ЭВМ за конкретными каналами, которым выделяется область ОЗУ и индивидуальные или смежные банки памяти. Затем он посылает каналу команду на запись или чтение, а каждый канал, включающий все необходимые микрооперации, такие как задержки между циклами, вычисление адресов блоков и т.д., занимается индивидуальным обменом. В результате внешние и внутренние операции оказываются независимыми друг от друга и могут исполняться параллельно. Эффективность доступа к памяти значительно повышается. Причем при необходимости можно увеличить число виртуальных каналов, обслуживающих одно устройство ЭВМ. Так, AGP может использовать каналы для загрузки текстур, расчета фигур, цвета и др.

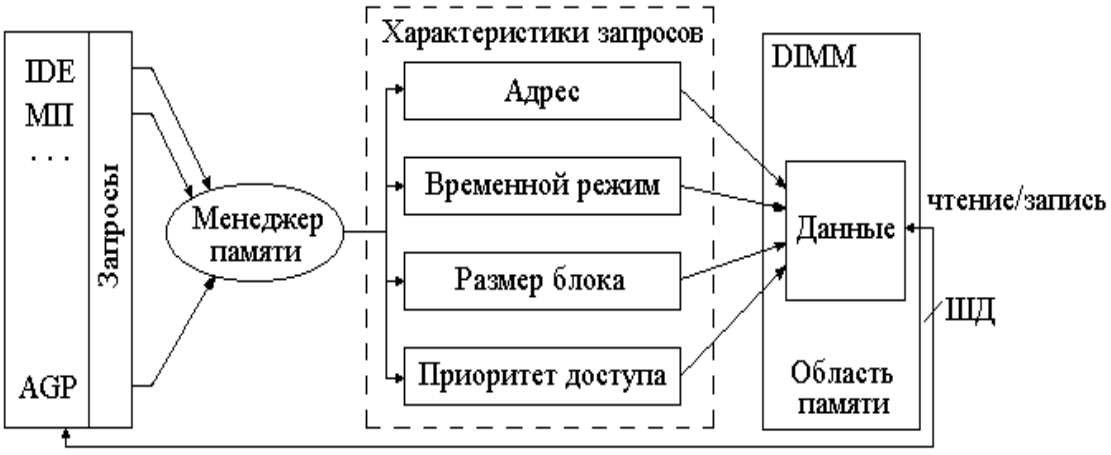


Рис. 3.30.–Структура обмена устройств ЭВМ с ОЗУ

Конструктивно, как показано на рис. 3.31, каждый виртуальный канал (ВК) включает ИС в виде буферной памяти SRAM емкостью 1 024 бит и располагается между линией ввода/вывода памяти и матрицей ячеек памяти. В буферной памяти осуществляется временное хранение данных, что позволяет с учетом разных банков осуществлять регенерацию памяти и циклы чтения/записи данных одновременно. Обмен данными между буфером виртуального канала и ячейками банка ОЗУ происходит блоками емкостью по 1 024 бита. Содержимое буфера канала затем участвует в интенсивных передачах системной шины.

По данным NEC, увеличение эффективности может достичь 90 %, а по тестам VCM133 SDRAM превосходит PC133 на 10 – 30 %. Кроме того, уменьшается энергопотребление примерно на 30 % за счет того, что в тот момент, когда происходит передача результатов приказа системному устройству, участвующему в обмене, вся фоновая активность по другую сторону виртуального канала может быть заморожена.

По выводам чипы VCM аналогичны обычным чипам SDRAM, совместимы с ними и по используемому интерфейсу. BIOS может легко распознать модули VCM. Модули, взаимозаменяемые с обычными SDRAM DIMM, причем все последние чипсеты от SiS и VIA поддерживают VCM. Увеличение площади чипа по сравнению SDRAM составляет всего 1 – 3 %, незначительно отличается и себестоимость PC133 SDRAM.

VCM не критичен к типу памяти и легко мо86ЭШа быть в дальнейшем встроен, например, в DDR SDRAM.

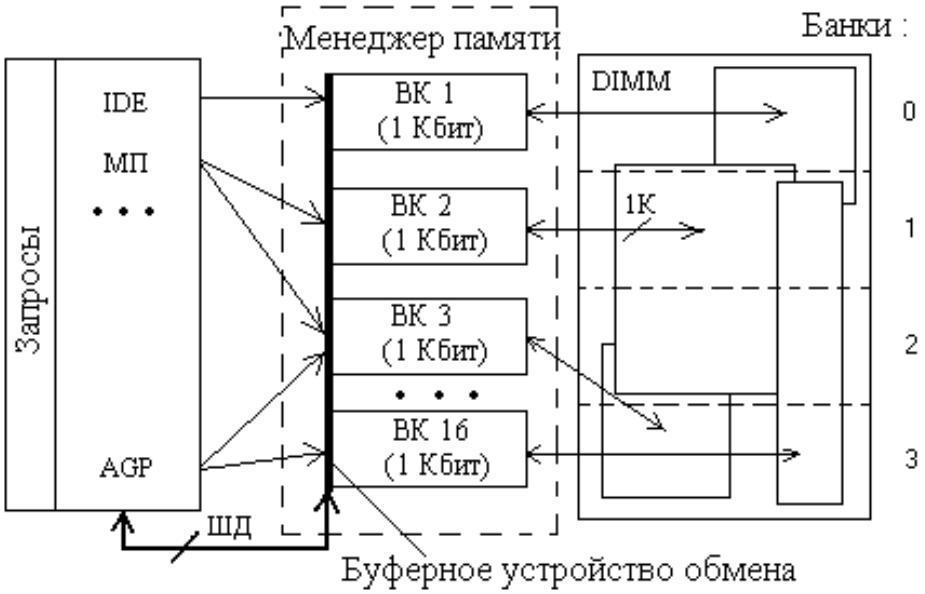


Рис.3.31.–Архитектура виртуального канала

3.13. Модули динамических оперативных ЗУ

Оперативные запоминающие устройства всегда были ресурсом, допускающим увеличение емкости, а иногда и сокращение времени обращения, с целью повышения общей производительности ЭВМ. Совершенствование технологий изготовления оперативных ЗУ привело к тому, что стойки памяти середины 1970-х годов емкостью 512 Кбайт, размером с двустворчатый платяной шкаф, сменили маленькие платы с микросхемами размером с зажигалку. Более того, если модернизация ЭВМ тех времен, связанная с наращиванием или заменой оперативной памяти, предполагала проведение достаточно серьезных монтажных работ, то теперь, в стандартной ситуации, замену памяти в течение 5 – 10 минут может провести даже пользователь, не являющийся специалистом по компьютерной технике.

Стандартизация интерфейсов оперативной памяти сделала возможным использование модулей памяти одних производителей в системах, собранных из компонент других производителей. Хотя, конечно, параметры и качество их могут быть различны.

Модули динамической полупроводниковой памяти прошли эволюцию от набора микросхем, устанавливаемых на системной плате и заметных по своему регулярному расположению (несколько смежных рядов одинаковых микросхем) до отдельных небольших плат, вставляемых в стандартный разъем (слот) системной платы. Первенство в создании таких модулей памяти обычно относят к фирме IBM.

Основными разновидностями модулей динамических оперативных ЗУ с момента их оформления в виде самостоятельных единиц были:

- 30-контактные однобайтные модули SIMM (DRAM)
- 72-контактные четырехбайтные модули SIMM (DRAM)
- 168-контактные восьмибайтные модули DIMM (SDRAM)
- 184-контактные восьмибайтные модули DIMM (DDR SDRAM)
- 184-контактные (20 из них не заняты) двухбайтные модули RIMM RDDRAM).

Сокращение SIMM означает *Single In-Line Memory Module* – модуль памяти с одним рядом контактов, так как контакты краевого разъема модуля, расположенные в одинаковых позициях с двух сторон платы, электрически соединены.

Соответственно DIMM значит *Dual In-Line Memory Module* – модуль памяти с двумя рядами контактов.

А вот RIMM означает *Rambus Memory Module* – модуль памяти типа Rambus.

Кроме этих модулей имеются также варианты для малогабаритных компьютеров, для графических карт и некоторые другие.

Если микросхемы памяти физически располагаются только с одной стороны платы, то такой модуль называют *односторонним*, а если с двух сторон – то *двухсторонним*. При равной емкости модулей, у двухстороннего модуля количество микросхем больше, поэтому на каждую линию шины данных приходится большая нагрузка, чем при использовании одностороннего. С этой точки зрения односторонние модули предпочтительней двусторонних. Однако количество банков в двусторонних модулях вдвое больше, чем в односторонних, поэтому при определенных условиях и хорошем контроллере памяти двусторонний модуль может обеспечить несколько большую производительность.

На рисунке 3.31 изображен односторонний 168-контактный 8-байтный модуль DIMM памяти SDRAM. Размеры этого модуля 133x35 мм. Микросхема, обозначенная звездочкой и пунктиром, используется в модулях, имеющих контрольные разряды и схемы контроля (см. далее). Размеры модулей SIMM несколько меньше. Модули RIMM имеют также металлическую пластину радиатора, закрывающую микросхемы.

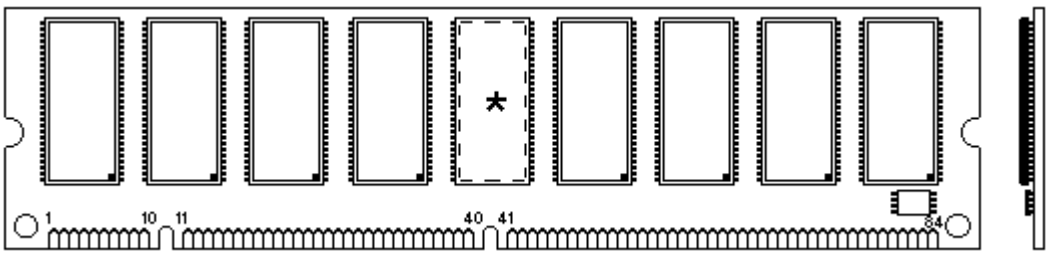


Рис.3.32– Модуль оперативной памяти DIMM с двумя рядами контактов

Помимо собственно конструктивной организации и типа памяти, модули имеют также и некоторые другие различия. Одним из таких различий является возможность (или ее отсутствие) ***контроля хранимых данных.***

Контроль может основываться на использовании дополнительных (по одному на каждый хранимый байт) битов четности (*Parity bits*), т.е. в этом случае каждый байт занимает в памяти по 9 бит. Такой контроль позволяет выявить ошибки при считывании хранимой информации из памяти, но не исправить их. Причем, строго говоря, нельзя и установить, когда возникла ошибка: при записи или при чтении. Кроме того, парная ошибка, хотя ее вероятность и очень мала, обнаружена не будет.

Более сложный контроль предполагает использование кодов, корректирующих ошибки – ECC (*Error Correcting Codes*), которые позволяют обнаруживать ошибки большей кратности, чем одиночные, а одиночные ошибки могут быть исправлены. Подобные схемы используются в серверных конфигурациях, когда требуется повышенная надежность.

Память, устанавливаемая в настольные ПЭВМ, обычно не имеет никакого контроля.

Кроме того, известны также различные модификации схем контроля, вплоть до просто имитирующих контрольные функции, но не осуществляющие их, например, с генерацией всегда верного бита четности.

Модули DIMM также различаются по наличию или отсутствию в них буферных схем на шинах адреса и управляющих сигналов. Небуферизованные (*unbuffered*) модули больше нагружают эти шины, но более быстродействующие и дешевые. Их обычно применяют в настольных ПЭВМ. Буферизованные (*registered*) имеют буферные регистры и, обеспечивая меньшую нагрузку на шины, позволяют подключить к ней большее количество модулей. Однако эти регистры несколько снижают быстродействие памяти, требуя лишнего такта задержки. Применяют буферизованные модули обычно в серверных системах.

Еще одной особенностью, различающей модули динамической памяти, является способ, посредством которого после включения компьютера определяется объем и тип установленной в нем памяти.

В первых ПЭВМ объем и быстродействие установленной памяти задавались переключателями (джамперами – *jumpers*), расположенными на системной плате.

С появлением модулей SIMM (существовали также похожие на них модули SIPP) стал использоваться так называемый параллельный метод идентификации (*parallel presence detect*), при котором краевой разъем модуля имел дополнительные контакты, используемые только для целей указания присутствия модуля в том слоте, где он установлен, его объема и времени обращения. В самых первых (30-контактных) модулях таких дополнительных контактов было только два, в 72-контактных модулях их стало четыре: два указывали на объем модуля и два – на время обращения. Эти контакты могли

заземляться непосредственно на модуле, что позволяло различить четыре вида модулей по объему и четыре – по времени доступа.

Попытки использовать этот же прием в последующих модулях потребовали увеличения количества таких контактов, но решить все проблемы идентификации не смогли. Поэтому, начиная с модулей DIMM, используют так называемый последовательный способ идентификации (*Serial Presence Detect* – SPD), при котором на плату модуля устанавливается специальная дополнительная микросхема, так называемый SPD-чип, представляющая собой небольшую постоянную память на 128 или 256 байт с последовательным (I^2C) интерфейсом доступа. На рисунке 3.31 эта микросхема показана в правом нижнем углу, хотя она может располагаться и иначе. В этой микросхеме записана основная информация об изготовителе микросхемы и ее параметрах. Формат этих данных стандартный, определенный советом JEDEC (*Joint Electron Devices Engineering Council*), стандартов которого придерживаются все изготовители полупроводниковой памяти.

3.14. Организация стековых (магазинных) запоминающих устройств. Запоминающие устройства с ассоциативной организацией

ЗУ со стековой организацией широко используется при построении системы прерываний ЭВМ, а также при программировании алгоритмов, связанных с обработкой данных типа вектор, массив (переменных с индексами). Стековые ЗУ обеспечивают запись, чтение информации в соответствии с правилом: последним пришёл, первым вышел (LIFO). В них при обращении доступна только одна ячейка – т.н. вершина стека. При записи в стек слово сначала записывается в вершину стека, а затем проталкивается внутрь ЗУ и т.д. при записи очередных слов. При выполнении операции чтения слово сначала выталкивается в вершину стека, а затем подаётся на выходную шину ЗУ.

Технически такая память может быть реализована на основе сдвиговых регистров в количестве n – штук: количество сдвиговых регистров определяется разрядностью ячеек – n. Разрядность регистров сдвига определяется ёмкостью стека.

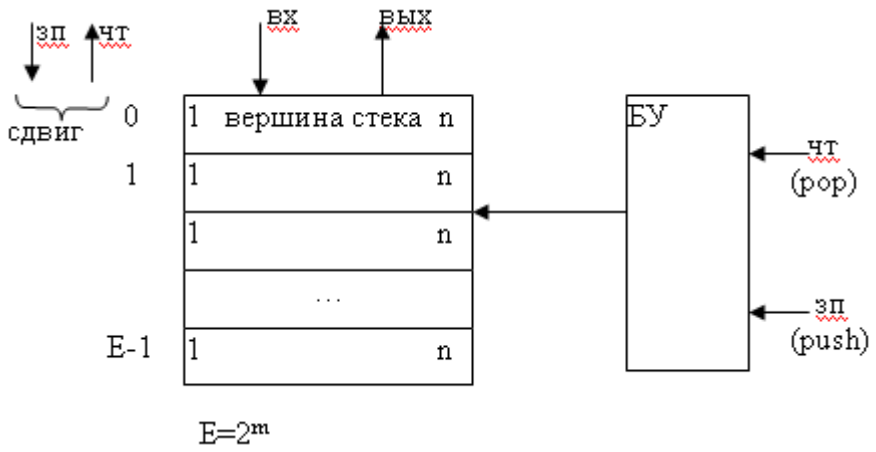


Рисунок 3.33.– Структура магазинных ЗУ

Такая организация стека называется **магазинной памятью** (рисунок 3.33). Одна из основных проблем магазинных ЗУ – переполнение стека, которое ведёт к потере информации, поэтому не допустимо. Следить за возможным переполнением должен сам программист.

Недостаток ЗУ магазинного типа – большие затраты оборудования и, следовательно, высокая удельная стоимость: сдвиговые регистры сложнее обычных.

Стековые ЗУ по этой причине (с целью экономии оборудования) обычно организуются другим способом: вместо сдвига информации в них используется подвижный указатель вершины стека (УС). Структура стекового ЗУ представлена на рисунке 3.34. Операция записи осуществляется по сигналу ЗП: 1) ЗП: [УС]:=ВХ; 2) УС:=УС+1, т.е. сначала производится запись слова в вершину стека (в ячейку, на которую указывает УС), а затем УС инкрементируется. Операция чтения реализуется по сигналу чтения ЧТ: 1) УС:=УС-1; 2) ВЫХ:= [УС].

Технически УС реализуется на основе реверсивного счетчика.

Следует отметить, что запоминающая часть стековых ЗУ обычно располагается в адресном пространстве ОП: часть ячеек ОП отводится под стек.

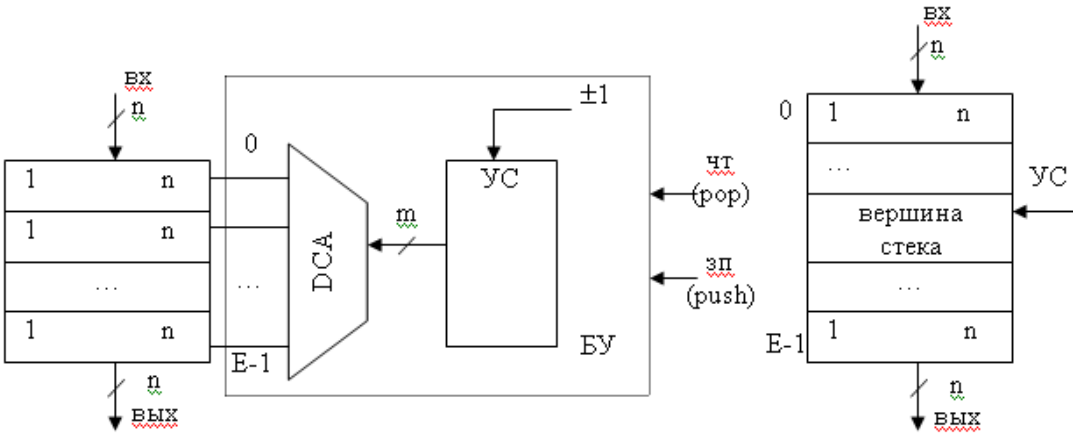


Рис.3.34.– Структура стекового ЗУ

Ассоциативные ЗУ (АЗУ) относятся к ЗУ безадресного типа. В них доступ к ячейкам памяти осуществляется не по адресу, а по ассоциативному признаку. В качестве ассоциативного признака (АП) используются содержимое ячейки или её части. АЗУ состоит из трех частей: 1) запоминающей части, которая организована в виде совокупности ячеек с номерами 0, 1, …, E-1; 2) блока ассоциативного поиска (доступа), т.е. схемы ассоциативного селектора и 3) блока замещения слов, который используется для записи информации в условиях, когда все ячейки АЗУ заняты. Схема АЗУ представлена на рисунке 3.35.

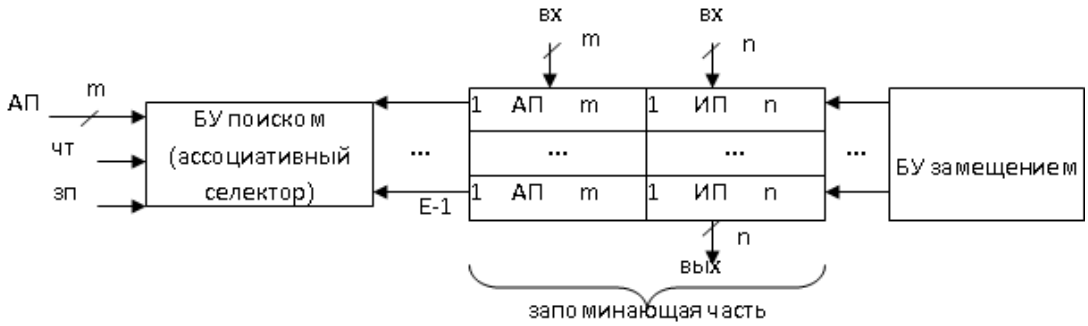


Рис.3.35.– Структура ассоциативного ЗУ

Ячейки запоминающей части состоят из двух полей: поля ассоциативного признака АП и информационного поля ИП.

Поиск информации (т.е. доступ к ячейкам ИП) осуществляется по ассоциативному признаку АП, который подается на вход АП ассоциативного селектора, и ведется путем сравнения входного признака АП со всеми АП, которые хранятся в полях АП всех ячеек запоминающей части:

$$АП=АП_i,\ i=0,\ 1,\ \ldots E\text{-}1. \qquad (3.2)$$

Выбранной считается та (или те) ячейка (и), для которой (ых) совпали ассоциативные признаки в (3.2). Если совпадений нет ни для одной ячейки, то это означает отсутствие в АЗУ той информации, обращение к которой производится по данному АП.

Таким образом, схема селекции в АЗУ организована на основе схем сравнения, а не дешифраторов, как в ЗУ с адресной организацией.

Блок управления замещением начинает работать в случае не сравнения в (3.2) и отсутствия свободных ячеек для записи. В этом случае блок назначает кандидата на удаление – указывает номер ячейки, содержимое которой следует удалить из памяти и на освободившееся место записать новую информацию, включая новый ассоциативный признак АП.

Если АЗУ используется в качестве кэш–буфера (между ЦП и ОП), то в этом случае в качестве ассоциативных признаков используются адреса ячеек (блоков) ОП, информация из которых дублируется в 92ЭШа. ’Скрытость’ 92ЭШа обеспечивается работой блока управления замещением, который реализует автоматический обмена с ОП. Наличие скрытой кэш–памяти с

точки зрения программиста делает память как бы одноуровневой (виртуально одноуровневой): уровень СОП скрыт от пользователя.

Основной недостаток АЗУ – большие (колоссальные) затраты оборудования на реализацию ассоциативного селектора. В случае, если запоминающая часть устройства организована по принципу 2D, то сложность селектора определяется количеством схем сравнения: $2^k=E_{АЗУ}$ (количество которых совпадает с количеством ячеек устройства).

Сложность одной схемы сравнения m элементов сравнения (однобитных). Общая сложность селектора: $N_{2D}=m2^k$ (3.3)

Пример: m=26, k=16, $N_{2D}=26*2^{16}$

Как уменьшить сложность селектора? Использовать ЗУ типа 3D:

$$N_{3D}=m*2^{k/2} \qquad (3.4)$$

В нашем примере: $N_{3D}=26*2^8$, т.е. в 256 раз меньше (в общем случае в $2^{k/2}$ раз меньше).

3.15. Организация кэш–памяти на основе ассоциативного запоминающего устройства (кэш с ассоциативной организацией)

В этом случае всё пространство ОП разделяется на блоки (строки КЭШа) размером $E_{стр}=2^n$, n<<m, которые принимаются за единицу обмена 93ЭШа с ОП. Пример: m=26, n=5 (строка длиной 32 байта).

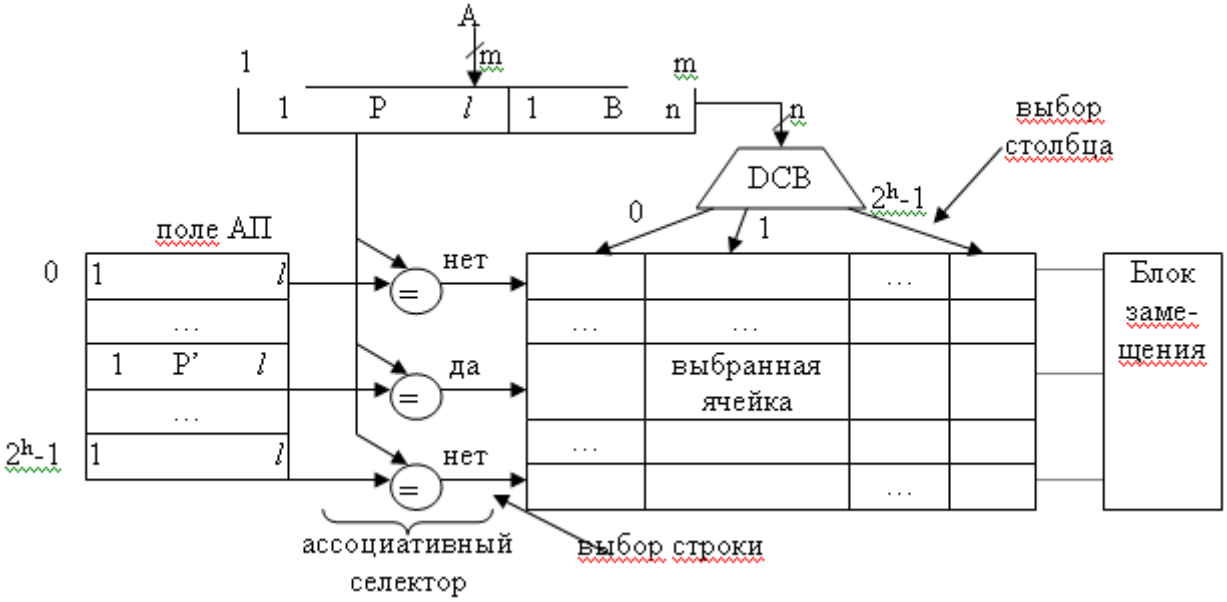
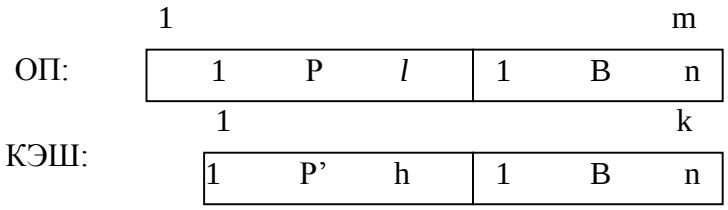


Рис.3.36.– Структура кэш с ассоциативной организацией

Всё пространство кэш-памяти также разделяется на строки ёмкостью 2^n. В результате адреса ячеек ОП и КЭШа разделяются на поля:



m=n+*l*, E=2^l – количество строк в ОП

k=n+h, E=2^h – количество строк в кэше

Пример: m=36, n=5, l=21, k=16, h=11.

Структура кэш с ассоциативной организацией типа 3D представлена на рисунке 3.36.

Если ассоциативный признак P=P', то обращение осуществляется к выбранной по 'да' строке КЭШа. Выбор ячейки в строке осуществляется при помощи дешифратора DCB (в примере B=1), т.е. по адресному принципу.

Если P≠P', то производится замещение: адресуемый блок P извлекается из ОП и переписывается в свободную или специально освобождённую строку КЭШа. Освобождение строки КЭШа осуществляется путём её переписи обратно в ОП по старому адресу P.

Основной недостаток – большие затраты оборудования на селектор. Пример: m=26, n=5, l=21, k=16, h=11, $N_{сел}$=l*2^h=21*2^{11}=42K (K=2^{10}) элементов сравнения.

В силу указанного недостатка кэш с чисто ассоциативной организацией обычно не применяется. С целью экономии оборудования используется более простая организация – кэш с наборно-ассоциативной организацией (архитектурой) (рисунок 3.37). В этом случае адрес A ячейки ОП делится на 3 поля: в третьем (старшем) поле G указывается номер группы строк (в нашем

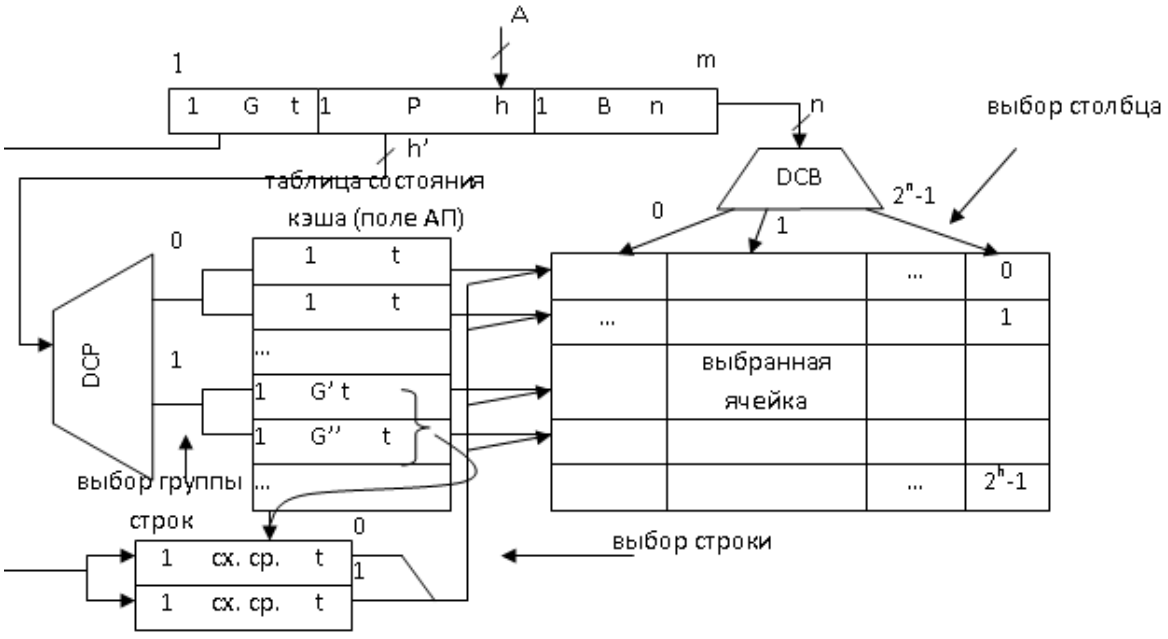


Рис.3.37.– Наборно-ассоциативная архитектура КЭШ

примере t=10). В группу объединяются две (четыре, …) строки.

Экономия оборудования достигается за счёт того, что ассоциативный поиск остаётся только в группе из r строк (в примере r = 2). Выбор же группы строк осуществляется по адресному принципу при помощи дешифратора DCP. Таким образом, количество схем сравнения уменьшается до r, т.е. до количества строк в группе, а разрядность схем сравнения уменьшается до t . В результате затраты оборудования на ассоциативный поиск сокращаются до величины: $N_{acc} = rt$ (пример: r=2, t=10, N=20).

Кроме того, сокращается длина ячеек поля АП до t разрядов, что также экономит оборудование. Правда, появляется дополнительный дешифратор DCP с количеством входов $2^{h'}$, где $h'=h-\log_2 r$ (в примере h'=h-1=10). Количество строк в группе определяется уровнем мультипрограммирования: r = M.

Кэш с наборно-ассоциативной архитектурой широко используется в IBM PC. Другие особенности организации кэш–буфера в IBM PC. Двухуровневая организация КЭШа: кэш первого уровня встраивается в процессор (конструктивно), а кэш второго уровня – вне ЦП, причём кэш второго уровня имеет большую ёмкость, чем кэш первого уровня. Кэш первого уровня работает на более высокой частоте – частоте ЦП, а второго – на частоте интерфейса ЭВМ (например, PCI). Вторая особенность (начиная с процессора Pentium): внутренний кэш делится на две части по назначению – кэш команд и кэш данных (т.н. Гарвардская архитектура ЭВМ).

3.16. ЗУ с двумерной адресацией

При необходимости побитовой записи-считывания информации применяют структуру памяти с двумерной адресацией (рисунок 3.38).

Данная структура содержит матрицу ЭЗЭ, статический регистр адреса, дешифраторы строки и столбца, усилители записи и считывания, входной и выходной буферные триггеры. Однако, каждый ЭЗЭ матрицы содержит не один, а два вывода разрешения работы (CS1 и CS2). При этом информационные выводы p1 и p2 являются обратимыми, то есть позволяют как записывать, так и считывать информацию. Для выбора нужной ячейки на оба входа CS необходимо подать активные логические уровни.

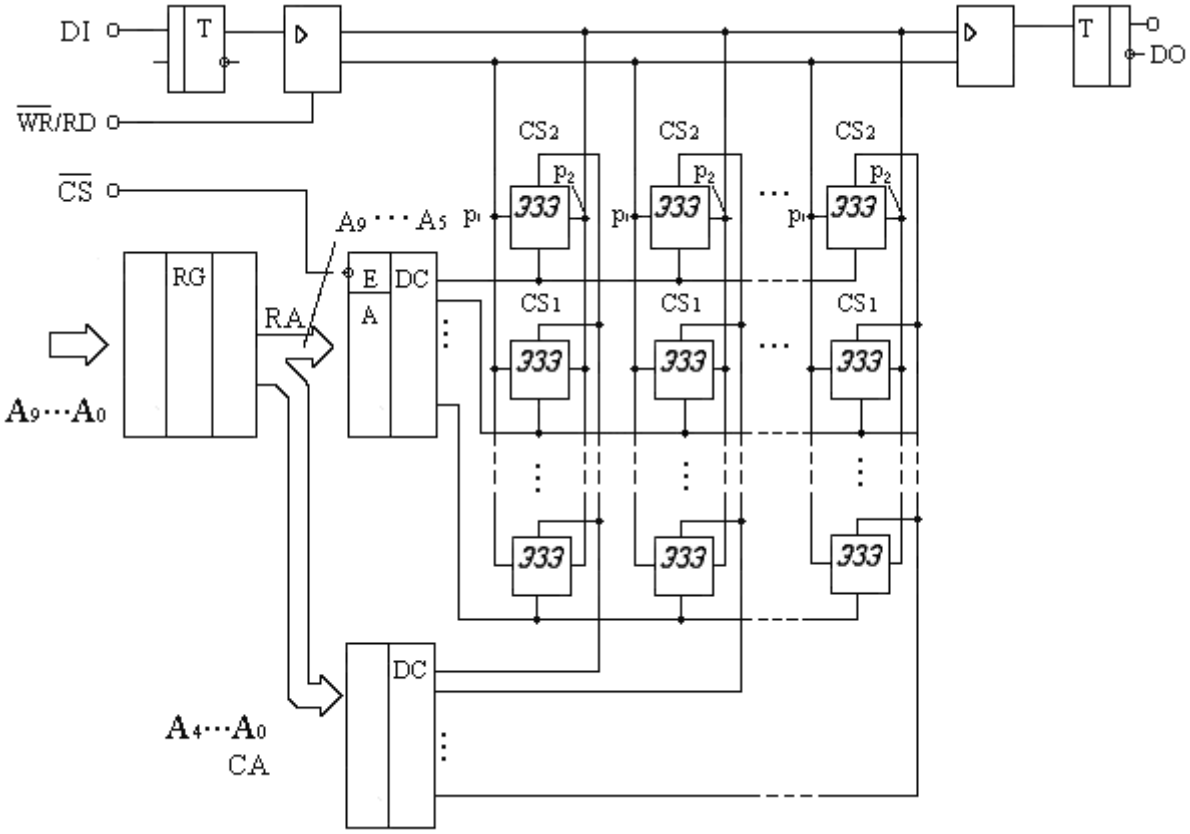


Рис.3.38.– Структурная схема ЗУ с двумерной адресацией

Цепи управления матрицей ЭЗЭ обеспечивают реализацию одного из трех режимов работы:

- хранения, при котором ЭЗЭ отключены от входа и выхода ИС;
- чтения, при котором информация из ЭЗЭ, выбранного по соответствующему адресу, выдается на выход ИС;
- записи, при которой информация со входа ИС записывается по указанному адресу.

Каждому ЭЗЭ матрицы присваивается определенный адрес, поиск которого производится указанием номеров соответствующих строки и столбца. Эти номера формируются на выходах дешифраторов. Адрес ЭЗЭ в виде двоичного числа принимается по адресной шине регистром адреса. Число разрядов регистра адреса однозначно связано с объемом памяти ИС. Число строк и столбцов матрицы ЭЗЭ обычно выбирается равным целой степени числа 2.

Разряды регистра адреса делятся на две группы: одна определяет двоичный адрес строки, другая – двоичный адрес столбца. Каждая группа разрядов адреса подается на соответствующий дешифратор (строк и столбца). Выходные сигналы дешифраторов выбирают требуемый ЭЗЭ из матрицы.

При чтении (WR(с инверсией)/RD= 1) содержимое этой ячейки через усилитель считывания выводится в выходной триггер.

Режим записи устанавливается путем подачи в усилитель записи сигнала разрешения записи (WR(с инверсией)/RD= 1). Этот сигнал открывает усилитель записи, и бит входной информации поступает на внутреннюю шину ИС, с которой переписывается в выбранный по соответствующему адресу ЭЗЭ.

Указанные процессы считывания-записи могут осуществляться только в случае, если на вход CS подан разрешающий сигнал. Обычно это сигнал лог. 0. При отсутствии этого сигнала работа дешифратора строки блокируется, что эквивалентно запрещению выборки ЭЗЭ по указанному адресу. В этом случае ИС находится в режиме хранения информации и ее выходы отключены от матрицы ЭЗЭ.

Рассмотренная организация памяти обеспечивает хранение 2^nx1 кодовых слов, то есть заданному адресу соответствует один бит информации. Использование метода двумерной адресации позволяет максимально упростить схему ИС, что при заданной площади кристалла является предпосылкой получения максимально больших объемов памяти.

3.17. Другие типы полупроводниковых запоминающих устройств

Семейство полупроводниковых ЗУ не ограничивается рассмотренными в предыдущих разделах разновидностями. Существуют различные ЗУ, либо представляющие собой определенные модификации описанных выше типов, целью которых является улучшение каких-либо характеристик, либо использующие иные технологические и физические принципы организации.

К модификациям типовых ЗУ можно отнести следующие разновидности:

- видео ОЗУ (*Video RAM*)
- оконное ОЗУ (*Window RAM*)
- синхронная графическая память (*SGRAM*)
- память с виртуальными каналами (*Virtual Channel Memory – VCM*) и др.

Видео ОЗУ, оконное ЗУ и синхронная графическая память – устройства, предназначенные для использования в видеоадаптерах.

Первые два из них представляют собой двухпортовую память. Один из портов является обычным интерфейсом динамической памяти, а другой – последовательным трактом считывания данных, используемым для регенерации изображения на экране монитора. Поскольку такая память реализуется на обычных динамических элементах, то наличие двух портов предполагает разделение доступа к матрице памяти со стороны портов, т.е. общая пропускная способность собственно памяти не увеличивается. Однако

за счет использования буферных регистров портов можно добиться их одновременной работы. *Window RAM* обеспечивает более высокую производительность при работе с графикой, чем *Video RAM*, за счет структурных решений, а также, так как имеет некоторые дополнительные аппаратные функции, ориентированные специально на задачи видеоадаптера.

Синхронная графическая память SGRAM представляет собой улучшенный вариант SDRAM, ориентированный на работу в видеоадаптерах, особенности которой проявляются при записи и чтении больших объемов информации. Как правило, эта информация пересылается большими блоками ячеек с последовательными адресами. Для оптимизации такого рода операций введены специальные режимы блочной записи – *Block Write* и побитовой записи – *Write-per-Bit*. Первый из них позволяет записать значение, предварительно занесенное в специальный регистр SGRAM (*Color register* – регистр цвета), в восемь ячеек одновременно. Второй режим (*Write-per-Bit*) также предназначен для заполнения ячеек определенными данными с возможностью маскирования (блокировки) отдельных ячеек.

Память с виртуальными каналами VCM, в отличие от перечисленных выше видов, предполагает усовершенствование не интерфейсов, а внутреннего доступа к матрице элементов памяти. В обычной памяти имеется один канал, по которому данные считываются из ядра и записываются в него. Фирма NEC предложила использовать несколько независимых каналов чтения/записи информации. Каждый из них имеет свои собственные буферные регистры для адреса строки, счетчики адресов пакетного режима и промежуточные регистры хранения информации, что позволяет считывать и записывать данные в разные области матрицы элементов памяти. При этом каждый канал может быть назначен независимому устройству, которое обращается к памяти, процессору, контроллеру ПДП или AGP-контроллеру. При обращении к памяти сразу нескольких устройств промежуточные данные временно хранятся в отдельных каналах и записываются в банки памяти по мере их освобождения от выполнения предыдущих операций. Такой подход позволяет увеличить производительность системы в среднем на 20%.

Известны и некоторые другие типы модификаций основных вариантов динамической памяти, например EDRAM (улучшенная динамическая память), MDRAM (многобанковая память), 3DRAM (“трехмерная” память) и др.

К запоминающим устройствам, использующим иные технологические и/или физические принципы, относятся:

- ЗУ на приборах с зарядовой связью;
- ферроэлектрические ЗУ.

ЗУ на приборах с зарядовой связью (иначе, с переносом зарядов) ПЗС являются представителями класса ЗУ с циклическим (прямым) доступом. Эти

ЗУ построены на принципе перемещения электрического заряда, для реализации которого используется МОП-транзистор, имеющий много затворов. Информация в таком транзисторе хранится в виде заряда (“1”) или его отсутствия (“0”) под затвором.

Подавая на соседние затворы сдвинутые по фазе потенциалы (обычно используется две или три серии сигналов), можно заставить заряды перемещаться в требуемом направлении от затвора к затвору, что показано на рисунке 3.39. Разместив такие транзисторы и затворы на них соответствующим образом, можно построить логические аналоги сдвигающих регистров с циклическим сдвигом (или, что то же, аналоги дорожек диска).

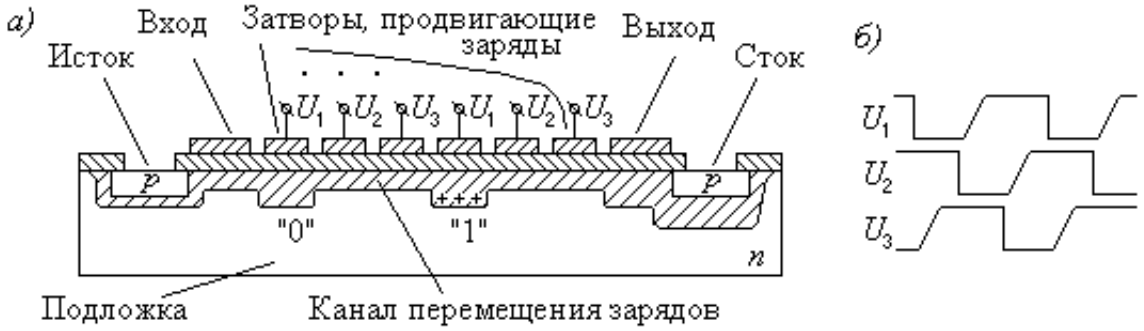


Рис. 3.39– ПЗС элемент на основе многозатворного МОП-транзистора, а) и диаграмма управляющих напряжений, б)

Знакомые с устройством электродвигателей могут увидеть в схеме продвижения зарядов аналогию с трехфазными электродвигателями. Подобного рода схемы применялись и в другом классе ЗУ с циклическим доступом: ЗУ на цилиндрических магнитных доменах.

Для построения собственно ЗУ на основе ПЗС используется архитектура, представляющая собой, по существу, набор сдвигающих регистров и дешифраторов адреса – мультиплексоров, коммутирующих связи этих регистров с шинами данных.

Однако наибольший интерес в настоящем представляет не использование ПЗС в запоминающих устройствах, а возможность построения светочувствительных ПЗС и применение матриц таких приборов в цифровых фотоаппаратах и видеокамерах. В этих матрицах инжекция зарядов в проводящую область осуществляется под действием света.

В ферроэлектрических ЗУ элементы памяти подобны элементам памяти динамических ЗУ, но вместо обычных конденсаторов в них используются конденсаторы на ферроэлектриках, обладающие свойством сохранения остаточной поляризации. Характеристика состояния таких элементов имеет вид гистерезисной петли, подобной той, которую имели элементы в запоминающих устройствах на ферритовых сердечниках. Этот тип памяти рассматривается как один из кандидатов на место оперативной памяти, позволяющей также сохранять информацию при выключении питания.

3.18. Организация дисковых массивов (RAID)

Одним из способов повышения производительности дисковой памяти стало построение дисковых массивов, и он нацелен не только (и не столько) на достижение более высокой производительности, но и большей надежности работы запоминающих устройств на дисках.

Технология RAID (*Redundant Array of Independent Disks* – избыточный массив независимых дисков) задумывалась как объединение нескольких недорогих жестких дисков в один массив дисков для увеличения производительности, объема и надежности, по сравнению с одиночным диском. При этом ЭВМ должна видеть такой массив как один логический диск.

Если просто объединить несколько дисков в (неизбыточный) массив, то среднее время между отказами (СВМО) будет равно СВМО одного диска, деленному на количество дисков. Такой показатель слишком мал для приложений, критичных к аппаратным сбоям. Улучшить его можно применяя реализуемую различным образом избыточность при хранении информации.

В RAID системах для повышения надежности и производительности используются комбинации трех основных механизмов, каждый из которых хорошо известен и по отдельности:

- организация “зеркальных” дисков, т.е. полное дублирование хранимой информации;
- подсчет контрольных кодов (четность, коды Хэмминга), позволяющих восстановить информацию при сбое;
- распределение информации по различным дискам массива так, как это делается при чередовании обращений по блокам памяти, что повышает возможности параллельной работы дисков при операциях над хранимой информацией. При описании RAID этот прием называют “stripped disks”, что буквально означает “разделенные на полоски диски”, или просто «полосатые диски».

Эти полоски (далее называемые блоками) данных с каждого диска чередуются, как показано на рисунке 3.40 для случая массива из четырех дисководов, образуя единый логический диск.

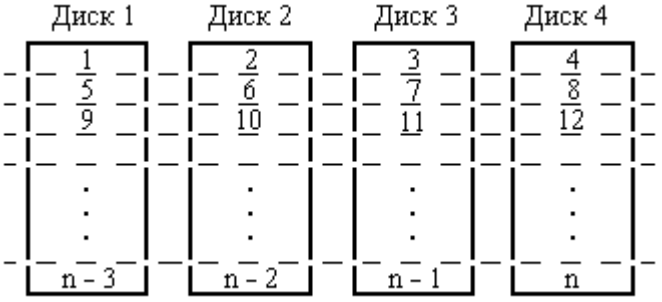


Рис. 3.40.– Разбиение дисков на чередующиеся блоки – “полоски”.

Изначально было определено пять типов дисковых массивов, обозначаемых RAID 1 – RAID 5, различающихся по своим особенностям и производительности. Каждый из этих типов за счет определенной избыточности записываемой информации обеспечивал повышенную отказоустойчивость по сравнению с одиночным дисководом. Кроме того, массив дисков, не обладающих избыточностью, но позволяющий повысить производительность (за счет расслоения обращений), стали часто называть RAID 0.

Основные типы RAID массивов можно кратко охарактеризовать следующим образом.

•***RAID 0***. Обычно этот тип массива определяется как группа дисков с чередованием (stripped) расположения информации без контроля четности и без избыточности данных. Размеры чередующихся областей (stripes – “полосок”, или блоков) могут быть большими в многопользовательском окружении или малыми в однопользовательской системе при последовательном доступе к длинным записям.

Организация RAID 0 как раз и соответствует той, которая показана на рисунке 3.39. Операции записи и чтения могут выполняться одновременно на каждом дисководе. Минимальное количество дисководов для RAID 0 – два.

Для этого типа характерны высокая производительность и наиболее эффективное использование дискового пространства, однако выход из строя одного из дисков приводит к невозможности работы со всем массивом.

•***RAID 1***. Этот тип дискового массива (рисунок 3.41, *а*) известен также как зеркальные диски и представляет собой просто пары дисководов, дублирующих хранимые данные, но представляющиеся компьютеру как один диск. И хотя в рамках одной пары зеркальных дисков разбиение на полоски не производится, чередование блоков может быть организовано для нескольких массивов RAID 1, образующих вместе один большой массив из нескольких зеркальных пар дисков. Такой вариант организации получил название RAID 1 + 0. Существует и обратный вариант.

Все операции записи производятся одновременно в оба диска зеркальной пары, чтобы информация в них была идентична. Но при чтении каждый из дисков пары может работать независимо, что позволяет выполнять одновременно две операции чтения, удваивая тем самым производительность при чтении. В этом смысле RAID 1 обеспечивает наилучшую производительность среди всех вариантов дисковых массивов.

•***RAID 2***. В этих дисковых массивах блоки-сектора данных чередуются по группе дисков, часть из которых используется только для хранения контрольной информации – ECC (error correcting codes) кодов. Но поскольку во всех современных дисках имеется встроенный контроль с помощью ECC

кодов, то RAID 2 мало что дает, по сравнению с другими типами RAID, и сейчас редко используется.

•***RAID 3***. Как и в RAID 2 в этом типе дискового массива (рисунок 3.41, *б*) блоки-сектора чередуются по группе дисков, но один из дисков группы отведен для хранения информации о четности. В случае выхода дисковода из строя восстановление данных осуществляется на основе вычисления значений функции «исключающее ИЛИ» (XOR) от данных, записанных на оставшихся дисках. Записи обычно занимают все диски (так как полоски короткие), что повышает общую скорость передачи данных. Так как каждая операция ввода-вывода требует доступа к каждому диску, массив RAID 3 может обслужить в каждый момент времени только один запрос. Поэтому данный тип обеспечивает наилучшую производительность для одного пользователя в однозадачном окружении с длинными записями. При работе с короткими записями во избежание снижения производительности требуется синхронизация шпинделей дисководов. По своим характеристикам RAID 3 близок к RAID 5 (см. ниже).

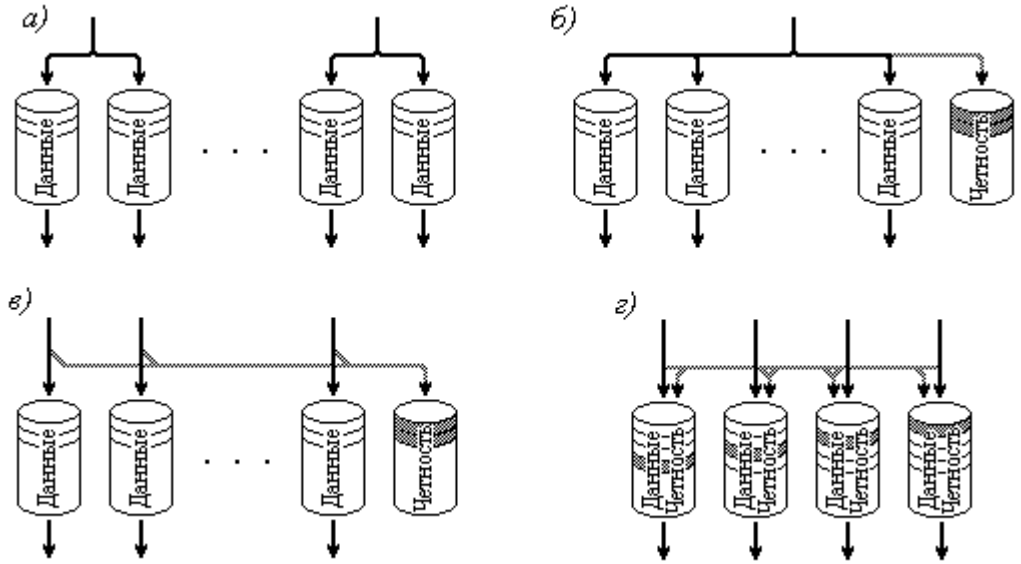


Рис. 3.41.– Варианты организации дисковых массивов RAID:
а) RAID 1,
б) RAID 3,
в) RAID 4,
г) RAID 5
(дисководы и блоки, используемые для хранения кодов четности показаны затененными).

•***RAID 4***. Эта организация, показанная на рис. 3.41, *в*), похожа на RAID 3 с той лишь разницей, что в нем используются блоки (полоски) большого размера, так что записи можно читать с любого диска массива (кроме диска, хранящего коды четности). Это позволяет совмещать операции чтения на

разных дисках. При операциях записи всегда происходит обновление диска четности, поэтому их совмещение невозможно. В целом, данная архитектура не имеет особых преимуществ перед другими вариантами RAID.

•***RAID 5***. Этот тип дискового массива похож на RAID 4, но хранение кодов четности в нем осуществляется не на специально выделенном диске, а блоками, располагающимися поочередно на всех дисках. Эту организацию даже иногда называют массив с “вращающейся четностью” (можно отметить некую аналогию с назначением линий прерываний для слотов шины PCI или с циклическим приоритетом контроллера прерываний в процессорах линии x86). Такое распределение позволяет избежать ограничения возможности одновременной записи из-за хранения кодов четности только на одном диске, характерного для RAID 4. На рис. 3.41, *г*) показан массив, состоящий из четырех дисководов, причем для каждых трех блоков данных имеется один блок четности (эти блоки заштрихованы), местоположение которого для каждой тройки блоков данных изменяется, перемещаясь циклически по всем четырем дисководам.

Операции чтения могут выполняться параллельно для всех дисков. Операции записи, требующие участия двух дисководов (для данных и для четности) обычно также могут совмещаться, так как коды четности распределены по всем дискам.

Сравнение различных вариантов организации дисковых массивов показывает следующее:

•Организация RAID 0 – это наиболее быстрый и эффективный вариант, но не обеспечивающий устойчивости к сбоям. Он требует минимум 2 дисковода. Операции записи и чтения могут выполняться одновременно на каждом дисководе.

•Архитектура RAID 1 наиболее пригодна для высокопроизводительных высоконадежных приложений, но и наиболее дорогая. Кроме того, это единственный вариант, устойчивый к сбоям, если используются только два дисковода. Операции чтения могут выполняться одновременно для каждого дисковода, операции записи всегда дублируются для зеркальной пары дисководов.

•Архитектура RAID 2 используется редко.

•Дисковый массив типа RAID 3 можно использовать для ускорения передачи данных и повышения устойчивости к сбоям в однопользовательской среде при последовательном доступе к длинным записям. Но он не позволяет совмещать операции и требует синхронизации вращения шпинделей дисководов. Для него нужно, как минимум, три дисковода: 2 для данных и один для кодов четности.

•Архитектура RAID 4 не поддерживает одновременные операции и не имеет преимуществ, по сравнению с RAID 5.

•Организацию RAID 5 характеризует эффективность, устойчивость к сбоям и хорошая производительность. Но производительность при записи и в

случае отказа дисковода хуже, чем у RAID 1. В частности, поскольку блок кодов четности относится ко всему записываемому блоку, то, если пишется только часть его, необходимо сперва считать ранее записанные данные, затем вычислить новые значения кодов четности и только после этого записать новые данные (и четность). Операции перестройки также требуют больше времени из-за необходимости формирования кодов четности. Для данного типа RAID нужно, как минимум, три дисковода.

Кроме того, на основе наиболее распространенных вариантов RAID: 0, 1 и 5 могут формироваться так называемые двухуровневые архитектуры, в которых сочетаются принципы организации различных типов массивов. Например, несколько RAID массивов одного и того же типа можно объединить в одну группу массивов данных или массив четности.

За счет такой двухуровневой организации можно достичь требуемого баланса между увеличением надежности хранения данных, характерным для массивов RAID 1 и RAID 5 и высокой скоростью чтения, присущей чередованию блоков на дисках в массиве типа RAID 0. Такие двухуровневые схемы иногда называют RAID 0+1 или 10 и 0+5 или 50.

Управление работой RAID массивов может осуществляться не только аппаратно, но и программно, возможность чего предусматривается в некоторых серверных вариантах операционных систем. Хотя понятно, что такая реализация будет иметь существенно худшие характеристики производительности.

Вопросы для самопроверки.

1. Дайте определение памяти по ГОСТу, а также второе определение памяти.
2. Что такое запоминающее устройство? Какие оно имеет характеристики?
3. Приведите общую структуру ЗУ.
4. Приведите классификацию ЗУ в зависимости от возможности записи и перезаписи данных в памяти.
5. Дайте схему иерархии памяти ЭВМ.
6. Объясните принцип работы БИС ОЗУ по структурной схеме.
7. Что такое SRAM? Какие основные сигналы присутствуют у данного вида памяти?
8. Приведите временные диаграммы памяти SRAM. Объясните принцип работы памяти по временной диаграмме.
9. Назовите разновидности пакетной статической памяти? Приведите их основные характеристики.
10. Назовите отличия между сигналами ADSP# и CADS#.
11. Приведите пакетные циклы чтения и записи (временную диаграмму) памяти SBSRAM. Объясните полученную диаграмму.

12. Приведите схему конвейерно-пакетной статической памяти.
13. По временной диаграмме объясните принцип работы конвейерно-статической памяти.
14. Приведите временные диаграммы асинхронной динамической памяти DRAM.
15. Что такое режимы FPM, EDO, BEDO?
16. Назовите основные свойства памяти SDRAM.
17. По временной диаграмме проведите анализ процессов чтения и записи.
18. Приведите структуру ЗУ с адресной организацией.
19. Назовите основные виды КЭШ-памяти? Приведите основные различия между ними.
20. Приведите схему ЗУ типа 2D. Объясните принцип работы.
21. Приведите схему ПЗУ с однократным программированием.
22. Назовите основные сигналы, присущие флэш-памяти.
23. Какие различия между памятью SDRAM и RDRAM?
24. Приведите основные модули ЗУ.
25. Какими основными характеристиками обладают стековые ЗУ?
26. Приведите структурную схему ЗУ с двумерной адресацией.
27. Какие вы знаете принципы организации RAID-массивов?

Модуль 4.

ПРОЕКТИРОВАНИЕ ПРОЦЕССОРОВ

Цель модуля: изучение студентами микропроцессоров и различных особенностей их архитектуры.

В результате изучения модуля студенты должны знать:

- особенности организации современных процессоров;
- процессоры с фиксированной и разнесенной архитектурой;
- организацию операционных устройств;
- структуру микропроцессорных устройств и систем

Содержание модуля:

4.1 Процессоры с фиксированной архитектурой;
4.2 Особенности архитектуры микропроцессора P6 фирмы INTEL.
4.3 Организация операционных устройств.
4.4 Структурная организация операционных устройств.
4.5 Структура микропроцессорных устройств и систем.
4.6 Микропроцессоры.

Структурная организация процессоров вытекает из принципа программного управления Дж. фон Неймана: процессор – это устройство для реализации процесса выполнения программы. Процесс выполнения программы сводится к выполнению действий, известных как цикл выполнения команд:

1. Выборка команды из памяти.
2. Выборка операндов.
3. Выполнение операции.
4. Запись результата.
5. Переход к п.1.

4.1. Процессоры с фиксированной архитектурой

4.1.1. Особенности организации современных процессоров

Элементной базой современных компьютеров являются сверхбольшие ИС (СБИС). СБИС выпускаются трех типов:

- СБИС с аппаратной реализацией алгоритмов обработки данных. Это микропроцессоры различных типов: универсальные, сигнальные, а также микроконтроллеры;
- микросхемы памяти: статистической (SRAM) и динамической (DRAM);

- программируемые логические ИС (ПЛИС).

Быстродействие процессоров можно повысить:

- за счет увеличения тактовой частоты (т.е. за счет уменьшения продолжительности такта Т: $t_{ком}=n*T$);
- за счет параллельной (в частности, конвейерной) обработки (организации работы), при которой как бы уменьшается количество тактов n на выполнение команд программы;
- за счет уменьшения времени обращения к памяти.

Увеличение тактовой частоты достигается в основном за счет совершенствования технологии СБИС, т.е. за счет уменьшения размеров электронных элементов, длины проводников, напряжения питания.

Повышение быстродействия процессоров на уровне системы команд обеспечивается различными способами.

Операции (и команды) процессора делятся на: скалярные и векторные. Скалярные операции (и команды, их возбуждающие) – это операции, в которых операнды и результаты являются скалярами (числами).

Для обработки массивов используются векторные операции (и команды, их возбуждающие). В них операнды и, возможно, результат являются массивом (вектором) чисел. Пример – умножение элементов двух массивов (векторов): очередные два элемента из двух массивов перемножаются, их произведение суммируется с содержимым заданного регистра. Затем модифицируются адреса ячеек памяти для извлечения двух других очередных элементов массивов и т. д. Эта последовательность действий (умножений, сложений, модификаций адресов элементов) повторяется заданное число раз по счетчику, определенному в теле команды.

Внедрение векторных команд в систему команд процессора повышает скорость обработки за счет уменьшения затрат времени на адресацию, выборку и дешифрацию команд: вместо нескольких обращений к памяти за командами – одно (за одной векторной командой) на весь массив исходных данных.

Следует отметить, что использование векторных команд требует от программиста написания векторной программы, что равносильно разработке параллельной программы (параллельному программированию).

При сохранении последовательного программирования для ускорения обработки используются суперскалярные процессоры. В них увеличение быстродействия достигается за счет применения конвейерной обработки, при которой в одном такте вырабатывается несколько скалярных результатов.

Повышение быстродействия при конвейерной организации обеспечивается не только за счет совершенствования самих конвейеров, но и путем их эффективной загрузки, т.е. за счет уменьшения простоев.

Эффективная загрузка параллельно работающих конвейеров обеспечивается либо аппаратурой процессора, либо компилятором, либо совместно аппаратурой и компилятором.

При компиляции из последовательной программы извлекается информация о параллелизме. Эта информация в явном виде указывается процессору и используется им для эффективного управления вычислительным процессом. В случае аппаратного выделения параллелизма специальные средства процессора сами выделяют элементы параллелизма из последовательной программы в процессе ее выполнения – в динамике. При этом в силу ограниченных возможностей аппаратуры (по сравнению с компилятором) используются более простые формы параллелизма – в основном естественного. Пример простейшего естественного параллелизма – это целочисленная обработка и обработка с плавающей запятой. Использование этого параллелизма привело к появлению процессоров с так называемой разнесенной структурой (архитектурой) – decoupled architecture (рисунок 4.10). Процессор с разнесенной архитектурой состоит из двух субпроцессоров, каждый из которых управляется собственным потоком команд. Условно их называют адресным А-процессором и исполнительным Е-процессором.

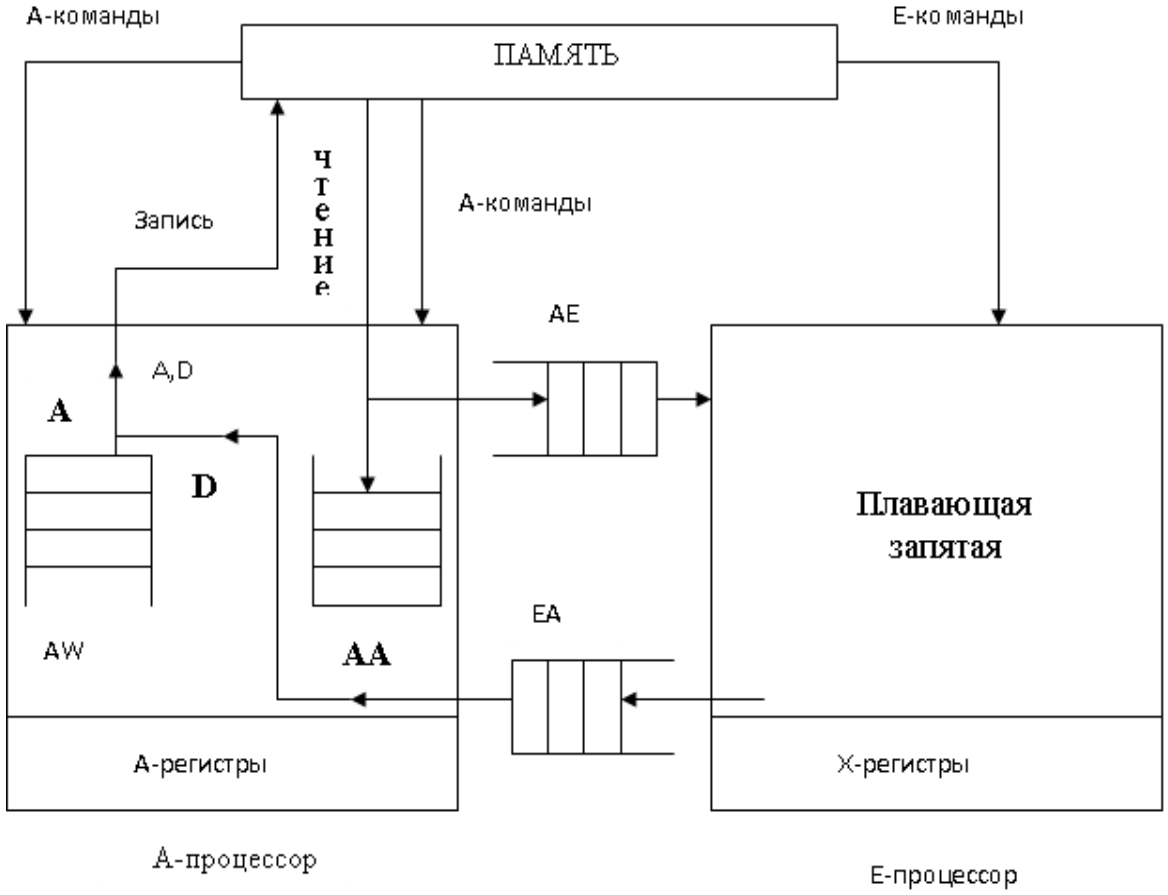


Рис.4.1– Процессор с разнесенной архитектурой

А-процессор выполняет все целочисленные (прежде всего адресные) вычисления и формирует обращения к памяти по чтению и записи. Е-процессор реализует вычисления с плавающей запятой.

Данные, извлекаемые из памяти, разделяются по типам и помещаются в соответствующую очередь (FIFO-очередь) адресов AA (целочисленные данные) или AE - очередь данных с плавающей запятой. Если очередь AE пуста, то Е-процессор ждет поступления данных в очередь AE от А-процессора. Результаты обработки из Е-процессора помещаются в очередь результатов EA для записи в память по адресу из очереди адресов записи AW. Адрес для записи результата вычисляется А-процессором и отправляется в очередь адресов для записи AW, не дожидаясь, пока результат обработки поступит в очередь EA. А-процессор, выбирая первые элементы из очередей AW, EA, организует их в пары A,D и отправляет в память (на запись). Если одна из очередей пуста (или обе вместе), то запись в память приостанавливается.

При чтении из памяти А-процессор отправляет данные либо в очередь AA (целочисленные данные), либо в очередь EA (плавающая запятая).

Расщепление последовательной программы на потоки команд для А- и Е-процессоров осуществляется либо на уровне компилятора, либо в процессоре специальным блоком–расщепителем.

Обмен с памятью в процессорах с расщепленной архитектурой организуется путем посылки транзакций – входных сообщений, несущих информацию не только об адресах ячеек и направлении обмена (ЧТ, ЗП), но и сами данные, их назначение (тип данных).

Интерфейс с памятью посредством транзакций чтения и записи позволяет поставить между процессором и памятью коммутационную среду, с помощью которой можно переходить к построению (организации) многопроцессорных ВК.

Способы уменьшения времени обращения к памяти: Первый способ – многоуровневая иерархическая организация памяти:

- регистры (десятки, сотни), время обращения – 1 такт ЦП;
- КЭШ первого уровня (L1) (десятки КВ), время обращения – 1-2 такта ЦП;
- КЭШ второго уровня (L2) (сотни КВ), время обращения – 3-5 такта ЦП, и т.д.
- Основная память (десятки, сотни МВ, ГВ), время обращения – десятки тактов ЦП.

Использование буферной памяти небольшого объема и высокого быстродействия позволяет существенно сократить количество медленных обращений к ОП, подменяя часть из них обращениями к быстрой буферной памяти. Это происходит за счет размещения активных фрагментов информации (команд и данных) в буфере на разных уровнях. Такая

организация памяти существенно уменьшает простои процессора в ожидании данных. Пока процессор обрабатывает блоки данных, размещенные в буфере, производится обмен блоками (программ и данных) между уровнями памяти, т. е. Обеспечивается совмещение во времени обработки в процессоре с пересылкой блоков между уровнями памяти.

Эффект уменьшения времени обращения к памяти будет тем больше, чем больше время обработки информации, расположенной в буфере, по сравнению с временем пересылки между ОП и буфером. Этот эффект базируется на свойстве локальности вычислительных процессоров, которое заключается в том, что процессор обычно многократно использует одни и те же ячейки для выработки некоторого результата (пример – итерационный цикл).

Второй способ – расслоение памяти. Этот способ также базируется на свойстве локальности адресов: при обращении к памяти процессор обычно генерирует естественную последовательность соседних адресов А, А+1, …, А+К-1 (К – длина цепочки адресов). В этом случае возможно К параллельных одновременных обращений в память по адресам, принадлежащим различным блокам памяти, состоящей из К блоков.

Очередной шаг эволюции – использование сокращенного набора команд (RISC). Дело в том, что практика применения компиляторов показала, что в процессе трансляции чаще всего используются простые команды типа R,R $\rightarrow$R и R память. Это означает, что компиляторы (их авторы) не в состоянии эффективно использовать сложные команды. Поэтому созрела идея использования сокращенного набора простых команд, которая и была реализована в RISC-процессорах. К такому решению разработчиков подтолкнуло также появление конвейерных процессоров (суперскалярных). Оказалось, что для RISC-команд конвейер организуется проще и эффективнее, чем для CISC-команд.

Конвейерная организация (обработка) породила проблемы, связанные с эффективной загрузкой конвейеров (с уменьшением простоев). Эта проблема (эффективной загрузки конвейеров) решается либо компилятором, устанавливающим очередность запуска команд в конвейер и вставляющим пустые такты (команды NOP) при невозможности запуска очередной команды, либо специальной аппаратурой процессора, отслеживающей зависимости между командами и устраняющей конфликты.

CISC-процессоры отличаются сложностью форматов и переменной длиной команд и, следовательно, сложным устройством управления, что препятствует повышению тактовой частоты. Поэтому переход на RISC-команды позволяет увеличивать тактовую частоту: блоки конвейеров проще, время их работы меньше, такт работы конвейера меньше, частота процессора выше.

Недостаток RISC-процессоров – большой объем (длина) машинных программ.

Развитие архитектуры процессоров происходит при постоянном стремлении обеспечить преемственность ПО. Сохранение преемственности и повышение быстродействия процессоров противоречат друг другу.

Проблема повышения быстродействия в рамках CISC-процессоров решается путем использования в них RISC-ядра и аппаратного транслятора CISC-команд в RISC-команды. Другими словами, в CISC-процессор встраивается RISC-ядро и аппаратный транслятор, превращающий CISC-команды в команды (в последовательности команд) RISC-ядра. Такой подход используется, в частности, в современных процессорах фирмы Intel.

Повышение производительности путем переключения контекста. Современные компьютеры обычно используют мультипрограммный режим работы. Время переключения процессора с одной задачи на другую, естественно, должно быть минимальным. Современные операционные системы при обработке прерываний, вызове подпрограмм, при переключении задач активно используют переключение контекста процессора. Уменьшение времени переключения контекста можно обеспечить либо за счет уменьшения количества регистров, сохраняемых в памяти, либо за счет аппаратной поддержки сохранения (вместо программной). В этом случае каждой активизируемой программе предоставляется свое подмножество регистров.

В процессорах SPARC компании SUN, например, используются 192 регистра, разделенных на 8 групп по 32 регистра в каждой (с перекрытием окон): регистры с номерами 24-31 одной группы одновременно являются регистрами с номерами 0-7 другой группы (окна). В этом случае сохранения регистров в памяти фактически не требуется.

4.1.2. Концепция открытых систем

Со временем постоянно растет сложность решаемых задач и, как следствие, растет сложность программных систем. Разнообразие типов (архитектур) процессоров ограничивает область применения сложных программных систем, т. к. для процессоров других типов их приходится разрабатывать заново. Поэтому с целью расширения области применимости сложных и дорогостоящих программных систем предпринимались попытки стандартизации (унификации) архитектуры процессоров. Однако на уровне машинных команд этого сделать так и не удалось (в силу конкуренции фирм и различного назначения процессоров). Аналогичные попытки были сделаны на уровне ассемблера, языков высокого уровня, интерфейса прикладных программ с операционными системами.

Отсутствие стандартизации не позволяет создавать новые программные системы простым способом – путем конструирования из существующих программ, прошедших апробацию в различных условиях применения

большим количеством независимых пользователей. На пути к такой стандартизации необходимо пройти два обязательных этапа:

- решение проблемы адекватного описания системы,
- решение проблемы тестирования системы на соответствие этому описанию. Тестирование должно быть доказательным.

Обе эти проблемы до сих пор не решены. Свидетельством тому являются известные примеры ошибок в микропроцессорах известных компаний (Intel в том числе), а также не декларированные возможности, как микропроцессоров, так и операционных систем.

Один из подходов комплексного решения проблемы комплексного решения проблемы стандартизации средств сводится к формулировке концепции открытых систем.

Открытая система – это система, разработанная с использованием стандартных средств: интерфейсов, протоколов и форматов данных, переносимых по принципам построения.

Переносимость – это свойство, заключающееся в возможности исполнения программы в исходных кодах на различных аппаратных средствах (платформах) под управлением (в среде) различных ОС.

Кроме того, открытые системы должны обладать свойством взаимодействия.

Взаимодействие – это свойство (открытой) системы, обеспечивающее способность обмена информацией с автоматическим восприятием форматов и семантики данных (например, в сети Internet).

Третье свойство открытых систем – **масштабируемость**, которая обеспечивает возможность исполнения программ на различных ресурсах (объем памяти, количество и быстродействие процессоров) с пропорциональным изменению ресурсов значением показателей эффективности.

Одна из известных попыток реализации концепции открытых систем предпринята в проекте стандарта ANDF (Architecture Nentral Distribution Format), который направлен на обеспечение переносимости программ. Для этого компиляцию предлагается делать в два этапа. На первом этапе исходный текст программы вместе с обобщенными описаниями типов данных транслируется в обобщенные декларации интерфейсов открытых систем – API. Результатом является программа в терминах абстрактной алгебры. Текст такой программы может быть подвергнут формальной проверке (тестированию) и, если необходимо, преобразованию. Тем самым обеспечивается адекватное описание системы и исчерпывающее ее тестирование. На втором этапе трансляции генерируется программа под конкретную архитектуру процессора.

Второй подход – использование Java-технологии (компания SUN). В основу положена так называемая виртуальная Java-машина со стандартными спецификациями (архитектурой).

Java-процессор имеет стековую архитектуру, поэтому большинство его команд имеют длину в один байт. Это позволяет иметь минимальную длину программ. Технология виртуальных Java-процессоров активно используется в сети Internet. Открытые программные системы на основе Java-процессоров, накапливаемые в недрах сети Internet, позволяют конструировать системы из уже существующих программных продуктов.

Недостаток Java-машин – низкая производительность, которая компенсируется использованием RISC-ядра в современных процессорах. В этом случае программа на первом этапе транслируется (компилируется) в язык Java-процессора, а на втором этапе в процессе выполнения Java-программы транслируется (интерпретируется аппаратурой процессора) в совокупность команд RISC-ядра.

4.1.3. Структура суперскалярного процессора

В настоящем параграфе для иллюстрации особенностей построения суперскалярных систем в качестве базовой используется архитектура суперскалярного процессора Пентиум фирмы Intel с системой команд типа CISC. Вызвано это следующими причинами:

- *суперскалярная организация процессора* Пентиум достаточно проста для изучения, однако в ней отражены все особенности построения суперскалярных систем;
- система команд процессора Пентиум практически совпадает с системой команд широко известного семейства микропроцессоров i80x86, поэтому примеры программ легко воспринимаются;
- наконец, в процессоре Пентиум на программном и аппаратном уровне принято ряд решений, приближающих его архитектуру к архитектуре RISC-процессоров, позволяющих, однако, сохранить программную совместимость с большим объемом ранее написанного прикладного матобеспечения.

Появление в структуре процессора более одного конвейера делает этот процессор суперскалярным.

Как правило, в суперскалярных процессорах в первую очередь увеличивают количество целочисленных конвейеров, так как статистика показывает, что в обычных пакетах прикладных программ для ПЭВМ около 80% команд - целочисленные, 15% - команды условных переходов, и только небольшой процент команд является командами с плавающей запятой.

Немедленно после введения второго или большего числа конвейеров возникают принципиально важные для суперскалярной структуры вопросы по организации и технической реализации вычислений:

- Как организовать запуск параллельных команд во множественных конвейерах? Нетрудно видеть, что здесь решается и вопрос о распараллеливании последовательных программ.
- Как уменьшить отрицательное влияние на работу конвейеров зависимости по данным для смежных команд и потери быстродействия из-за появления в программе условных переходов? Планирование межконвейерного параллелизма должно производиться с учетом указанных эффектов.
- Как обеспечить повышенную пропускную способность всех видов памяти в суперскалярном процессоре, особенно если число одновременно выполняемых операций составляет 20-30?
- Каковы в перспективе предельные возможности, размеры суперскалярных процессоров?

Организация параллельного запуска команд предусматривает выполнение трех этапов:

- распараллеливание последовательной программы;
- планирование порядка выполнения команд для заданных ресурсов;
- представление спланированной программы в форме, пригодной для исполнения аппаратурой.

В зависимости от способа реализации этих этапов методы формирования и запуска параллельных команд можно разделить на: статические, динамические и смешанные.

Считается, что статические способы предпочтительнее, поскольку анализ программы в процессе компиляции обеспечивает более глубокое выделение и планирование параллелизма, кроме того, исключает служебные команды управления параллелизмом из вычислительного процесса. Динамический же запуск позволяет более полно учитывать текущее состояние программы и ресурсов при исполнении программы.

Возможны следующие способы статического формирования параллельной команды:

- Производится распараллеливание программы обычными методами, затем параллельные команды с помощью оптимизирующего планировщика упаковываются в виде СДК фиксированного формата. В таком СДК каждое

поле формата закреплено за определенным конвейером и хранится в одной ячейке параллельной памяти. При чтении СДК из памяти выбираются все поля СДК независимо от их заполнения. Конвейеры запускаются синхронно в каждом такте. Это является возможным, поскольку в RISC-конвейерах основные операции выполняются за 1 такт. Основной недостаток схемы с фиксированным форматом СДК - неполное использование памяти при отсутствии команд в некоторых позициях СДК.

- Более полное использование ячейки памяти фиксированного формата применяется в МП i860 (рисунок 4.2).

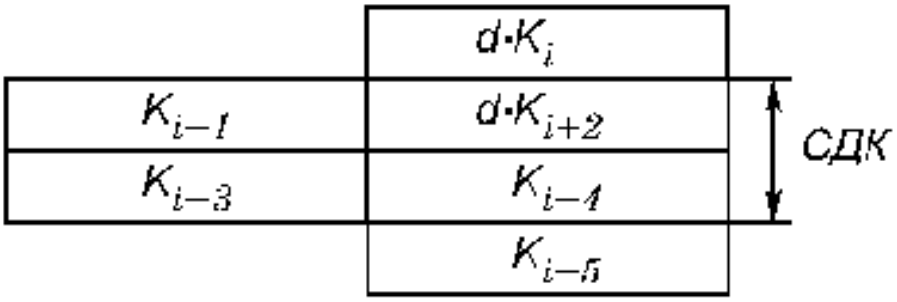


Рис. 4.2.–Представление СДК в памяти МП i860

В МП i860 только два конвейера, поэтому СДК имеет длину 2. Если в некоторой команде, например, K_i указан признак *d* (*dual*-двойной), это означает, что следующие две команды по списку, а именно, и образуют параллельную пару. Физически эта пара может быть расположена либо в одной ячейке параллельной памяти, либо в смежных ячейках последовательной памяти.

- Описанные выше способы годятся для вновь разрабатываемых систем команд, поскольку на этапе разработки можно заложить средства, обеспечивающие объединение команд в СДК фиксированного формата. Однако эти способы нельзя применить к системам команд CISC-процессоров, которые нельзя "подкорректировать", поскольку для них уже существует большой объем написанного математического обеспечения.

4.2. Особенности архитектуры микропроцессора P6 INTEL

Промышленное производство МП P6 и компьютеров на его основе началось в 1995 году. В данной лекции рассмотрены некоторые особенности его архитектуры.

Структурная схема МП P6 представлена на рисунке 4.3

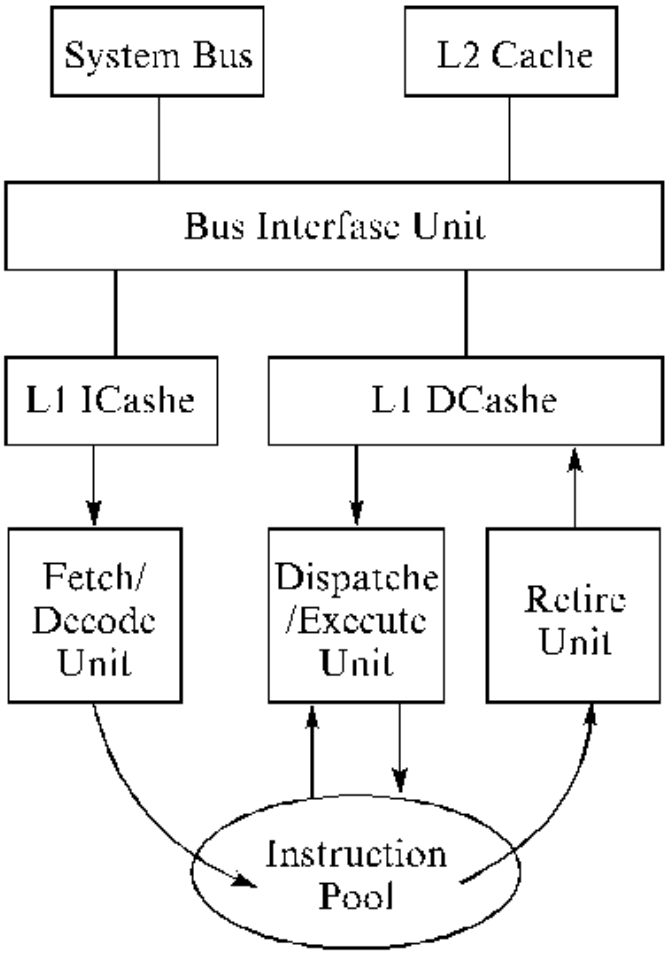


Рис. 4.3.– Структурная схема МП P6

КЭШ команд (L1 ICache) и КЭШ данных (L1 DCache) расположены внутри кристалла P6, а КЭШ второго уровня (внешний) объемом 256 Кбайт выполнен на отдельном кристалле. Оба кристалла (MP P6 и L2 Cache) конструктивно расположены на единой подложке и взаимодействуют через независимую систему шин.

Основу МП P6 составляют три устройства:

- устройство выборки и декодирования команд - Fetch/Decode Unit (FD);
- устройство диспетчирования и выполнения команд - Dispatche/Execute Unit (DE);
- устройство удаления выполненных команд - Retire Unit (R).

Все эти устройства функционируют независимо и обмениваются результатами своей работы только через буфер команд - Instruction Pool (IP). Рассмотрим поочередно работу этих устройств.

Устройство выборки и декодирования команд FD выполняет следующие функции:

•Команды вводятся в буфер команд IP устройством FD, а удаляются из него - устройством R. Вместе с тем для полноценной работы устройства DE необходимо, чтобы в буфере команд IP находилось 20...30 команд программы, среди которых по статистике в среднем находится 4...5 команд условных переходов. Таким образом, первой функцией устройства FD является построение наиболее вероятной трассы выполнения программы путем динамического прогнозирования последовательных ветвлений. Предсказание каждого осуществляется с помощью большого буфера истории переходов BTB (Branch Target Buffer), содержащего 512 входов. Правильность предсказания обеспечивается более, чем в 90% случаев. В начальный момент, когда BTB еще не заполнен, используется аппаратно реализованный алгоритм статического предсказания.

•На последующих ступенях конвейера FD производится замена исходных команд системы i80x86 на универсальные опера-ции ***uops*** (universal operations). Каждая ***uop*** имеет два логических адреса для операндов и один - для результата. Большинство команд i80x86 преобразуются в одиночную ***uop***, некоторые - в две, три и более ***uop***. Сложные команды реализуются в виде микрокода из ***uop***.

•Выстроенные в очередь ***uops*** посылаются в блок переименования регистров, Register Alias Table Unit (RAT). Дело в том, что небольшое число регистров общего назначения в системе команд i80x86 (всего 8) ограничивает параллелизм, внося ложные зависимости по данным.

Чтобы устранить ложные зависимости, в P6 используется 40 физических регистров, которые на время обработки заменяют имена логических регистров, указанных в программе. Замена производится автоматически при каждом обращении на запись в данный регистр. Соответствие между номерами физических и логических регистров хранится в RAT. При каждом обращении к логическому регистру на чтение, производится обращение к RAT.

На выходе из устройства FD каждая ***uop*** снабжается статусной информацией (номер команды, номер ***uop***, готовность операндов и др.) и передается в буфер команд IP, который выполнен в виде контекстно адресуемой памяти. Пропускная способность устройства FD - 3 команды за один такт. Все команды обрабатываются в порядке их расположения в программе (in order). Для работы устройства FD используется 6 первых ступеней конвейера. Структура устройства диспетчирования и выполнения DE представлена на рисунке 4.4.

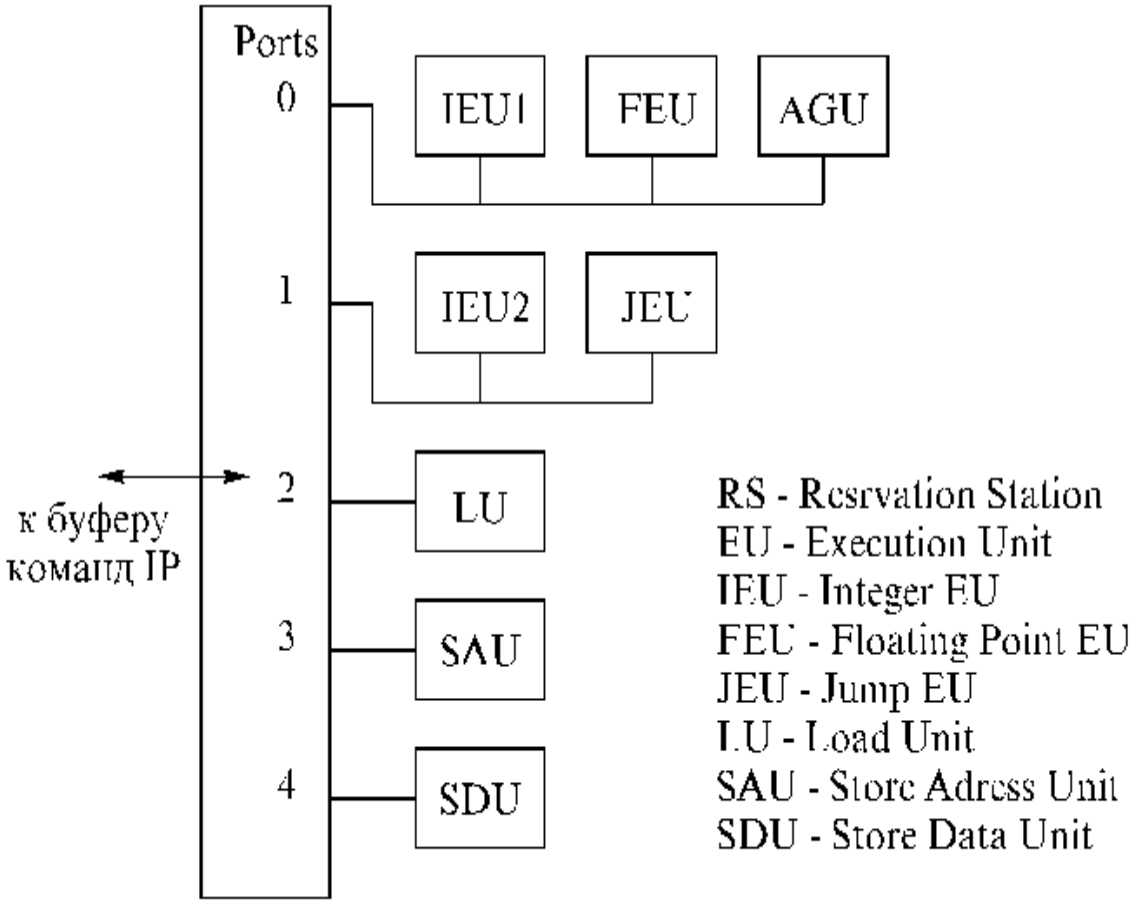


Рис.4.4.– Структура устройства DE диспетчирования и выполнения команд

Блок диспетчирования RS в каждом такте, выбирает из IP готовые для исполнения ***uops*** и запускает их на исполнение параллельно по каждому из 5 портов. Естественно, в каждом такте по одному порту можно запустить только одну ***uop***. Функции устройства DE следующие:

1. Выбор готовых для исполнения в данном такте ***uops***. Алгоритм выбора основан на том, что каждая ***uop*** содержит биты состояния, облегчающие готовность операндов. Если оба операнда готовы, то ***uop*** назначается на соответствующий порт. Назначение является простой операцией, если в данном такте готова только одна ***uop***. Если готовых операций несколько, то необходимо оптимальное планирование. Идеальным было бы выбирать такие ***uops***, которые уменьшали длину информационного графа отрезка программы. Поскольку такое планирование требует слишком много времени, то в P6 используется значительно более простой алгоритм типа "первый пришел - первый ушел" (FIFO).

На рисунке 4.5 представлен пример запуска и выполнения ***uops*** исполнительными блоками устройства DE. На рисунке затемненные команды можно выполнять параллельно, а светлые - только последовательно.

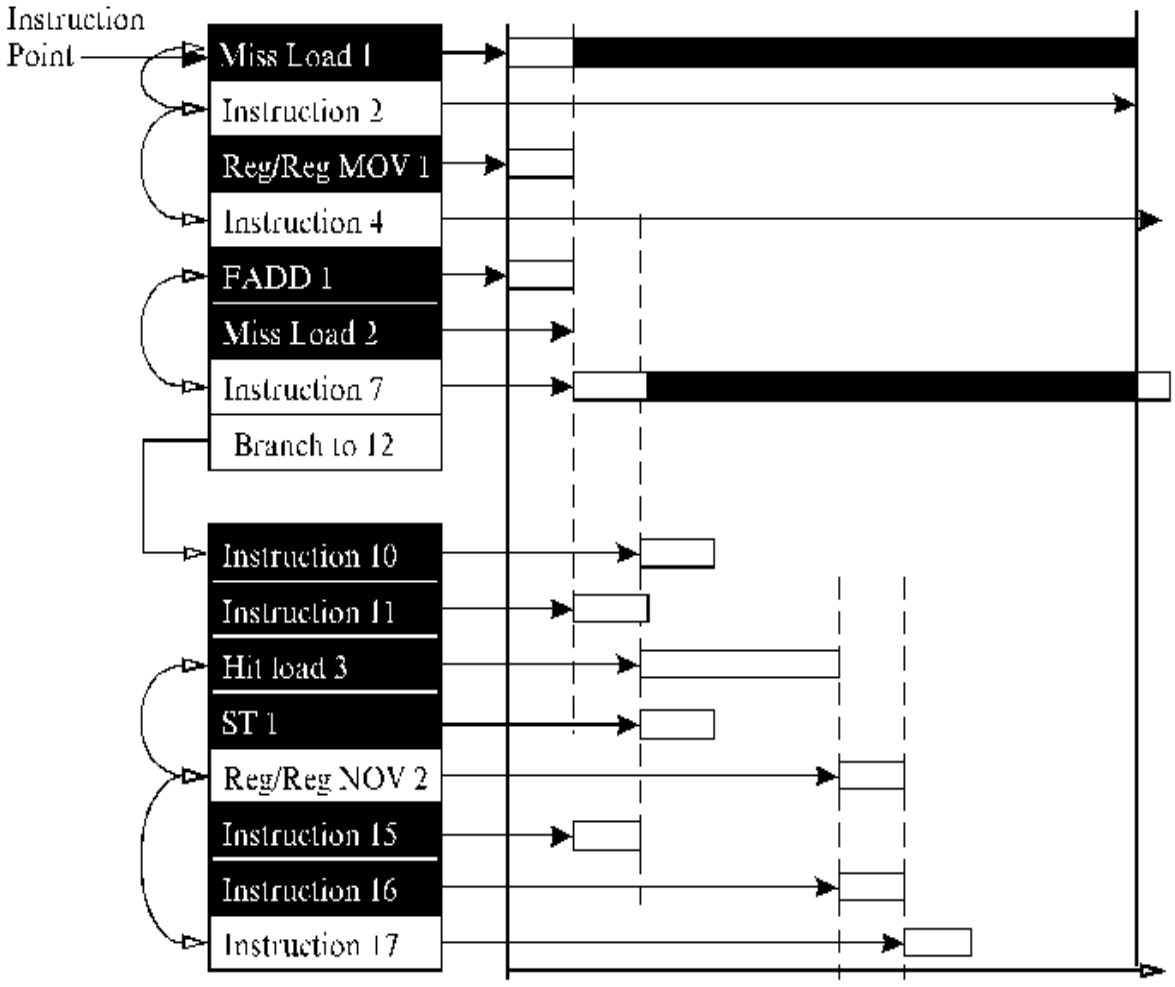


Рис. 4.5.– Диаграмма выполнения отрезка программы в устройстве DE

•В первом такте запускается три ***uops***. Первая команда Load должна загружать регистр из L1 DCashe, но вследствие промаха ее выполнение затягивается на много тактов. Две других ***uops*** выполняются за один такт. Во втором такте запускаются три следующие ***uops***, первая из них также соответствует промаху КЭШ. Из диаграммы видно, что выполняются четыре команды и стартует пятая (вторая Load с промахом КЭШ) перед окончанием второго такта, и все эти команды выполняются внутри задержки , вызванной первым промахом памяти. В этом случае в обычном суперскалярном МП после промаха выполнение последующих команд приостанавливается до появления данных из памяти.

Диаграмма на рисунке 4.5 показывает, что:

- команды в DE выполняются в порядке их готовности, а не в порядке их размещения в программе (out of order), и это соответствует принципу функционирования процессоров типа Data Flow;
- выполнение команд является условным, т. е. результаты записываются во временные (физические) РОН, а не в логические РОН или память;
- каждый исполнительный блок (целочисленные и плавающие АЛУ, блоки обращения к памяти и др.) содержит несколько ступеней конвейера;
- документация по P6 содержит таблицы, в которых указано: на какое число и каких ***uops*** разбивается каждая команда 80x86; какой порт требуется для исполнения каждой ***uop*** и сколько тактов занимает это исполнение. Эта информация позволяет подсчитать время выполнения программы;
- в пиковом режиме устройство DE может запускать 5 ***uops***, но в длительном режиме - в среднем 3.

2. После выполнения ***uop*** в устройстве DE она возвращается в буфер команд IP, причем в ней один из битов состояния устанавливается в состояние "выполнена". Кроме того, выявляются все команды, которые используют полученный результат в качестве операнда, и в этих командах устанавливается бит состояния "операнд готов". Такое изменение состояния переводит некоторые команды в состояние готовности, и они могут запускаться на параллельное исполнение в следующем такте.

3. Когда в DE выполняется команда перехода, то результат ее выполнения сравнивается с предсказанным. Если они совпадают, то команды, которые находятся в программе после данной команды перехода и уже выполнены условно считаются выполненными правильно и их результаты могут быть переданы в память или логический регистр. Если же результат выполнения команды не совпал с предсказанным, то блок JEU изменит состояние всех команд, находящихся в программе после данной команды перехода таким образом, чтобы они были удалены из IP, а устройство FD обеспечит выборку новой последовательности команд по новому адресу перехода.

Устройство удаления выполненных команд R работает следующим образом:

- просматривается буфер команд IP. Кандидатами на удаление являются все ***uops*** с признаком "выполнено";
- среди кандидатов на удаление выбираются те ***uops***, адреса которых составляют непрерывную последовательность по отношению к последней, ранее удаленной ***uop***, т. е. устанавливается исходная последовательность

uops, принятая для устройства FD (in order). За один такт может удаляться до трех смежных ***uops***;

- удаляемые одиночные ***uop*** или группы ***uops*** заменяются на исходные команды 80x86, и результаты записываются по соответствующим логическим адресам, т. е. окончательно. При этом не возникает вопроса о правильности результатов удаленных команд, так как выполненные команды переходов удаляются ранее следующих за ними обычных команд.

При удалении команд вносятся соответствующие изменения в буфер команд IP и другие таблицы. Например, если использование некоторого физического регистра прекращается при удалении команды, то он вычеркивается из таблицы RAT.

В заключение перечислим основные алгоритмы, аппаратно реализованные в P6.

1. Устройство выборки и декодирования FD:
   - алгоритм предсказания направления переходов в нескольких последовательных командах условных переходов;
   - механизм замены команд МП i80x86 соответствующим набором ***uops***;
   - алгоритм переименования логических РОН на физические.
2. Устройство диспетчирования и выполнения ED:
   - алгоритм выявления и распределения "готовых" команд по портам исполнительных устройств;
   - алгоритм выполнения ***uops*** в исполнительных устройствах;
   - алгоритм изменения статуса команд в буфере команд IP после выполнения очередной порции команд;
   - алгоритм отмены результатов выполненных команд и набора новых команд после выявления неверно предсказанного перехода.
3. Устройство удаления команд R:
1. механизм выявления команд для удаления из IP;
2. механизм преобразования ***uops*** в команды МП i80x86 и запись результатов по безусловным адресам;
3. механизм внесения изменений в буфер команд и другие таблицы после удаления одной или нескольких команд.

Все эти алгоритмы, называемые в совокупности разработчиками Dinamic Execution, обеспечивают неблокируемость конвейера P6. Кроме того, конвейер P6 имеет средства для образования многопроцессорных систем.

4.3. Организация операционных устройств (ОУ)

Обработка информации в ЭВМ осуществляется в ЦП, в частности, в АЛУ (основная обработка), а также в контроллерах ПУ (вспомогательная, предварительная обработка). Все эти устройства – АЛУ, КПУ – по принципам организации, построения относятся к классу ОУ и предназначены для выполнения операций из списка операций F по инициативе ЦП (рисунок 4.6). Принципы построения всех этих устройств едины, поэтому, сначала рассмотрим их (как теорию, абстракцию), а затем перейдем к рассмотрению конкретных устройств: АЛУ и т.д.

4.3.1. Принцип микропрограммного управления (функциональная организация операционных устройств)

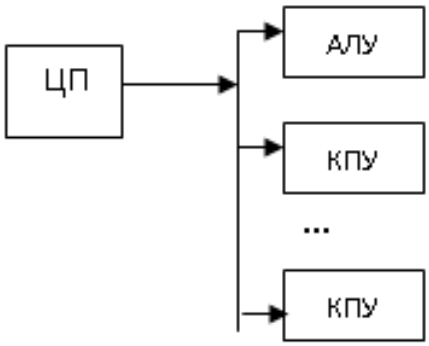


Рис.4.6.– Обработка информации в ЭВМ

Организация ОУ базируется на принципе микропрограммного управления, основные положения которого можно сформулировать в виде следующих 4 тезисов.

Любая операция $f_g \in F$ рассматривается как сложное действие и разделяется на совокупность элементарных действий, называемых микрооперациями (МО). Выполнение каждой МО осуществляется специальной комбинационной схемой (КС) за один такт машинного времени.

Порядок выполнения МО задается алгоритмом операции $f_g \in F$ и зависит от значений логических условий (ЛУ). ЛУ принимают значения истина или ложь в зависимости от значений операндов. ЛУ используются в качестве условий альтернативных переходов в алгоритмах операций.

Алгоритм, представленный, записанный в терминах МО и ЛУ, называется микропрограммой (МП). МП задает порядок выполнения МО и проверки ЛУ во времени.

Совокупность микропрограмм $МП_1$, …, $МП_g$ задает функцию ОУ.

Итак, принцип микропрограммного управления является основой организации (построения) ОУ. Нетрудно увидеть, что эти тезисы достаточно схожи с фон Неймановскими тезисами программного управления.

И там, и здесь в основу управления положен алгоритм. Только у Неймана он представляется в виде программы и поступает в процессор извне (из ОП извлекается процессором). Здесь алгоритм в виде МП уже находится внутри ОУ.

При выполнении программы ЦП генерирует определенную последовательность операций, реализуемых ОУ. При выполнении операции f_g ОУ генерирует последовательность МО, реализуемых комбинационными схемами КС.

Отличия между этими принципами:

- операция - сложное действие, для реализации которого необходимо ОУ. МО - элементарное действие, для реализации которого достаточно КС.
- Операция выполняется за n тактов: $t_{опер}=nT$. МО выполняется за 1 такт (алгоритм - в КС).

4.3.2. Средства описания функций операционных устройств

Функция ОУ задается (определяется) совокупностью микропрограмм $МП_1$, …, $МП_G$, описывающих алгоритмы операций f_1, …, f_g. Для описания МП в вычислительной технике используются специальные средства описания – язык функционального микропрограммирования.

Функциональная микропрограмма (ФМП) – это микропрограмма, описывающая алгоритм операции без привязки его к конкретной структуре ОУ. Отсюда название языка.

Для описания ФМП в языке используются различные средства, обеспечивающие описание слов, МО, ЛУ, а также средства, описывающие порядок их выполнения во времени.

Описание слов и массивов. Слово описывается своим именем и длиной: $C(n_1{:}n_2)$. Здесь С - имя слова, n_1, n_2 - номера старшего и младшего разрядов слова соответственно. Примеры (смотри МП умножения): A(0:15) и др.

Часть слова называется **полем** и описывается аналогично словам: A(0:7), A(0:15), B(15), A(0) и т.п.

Слово обычно описывается один раз. После этого можно использовать его сокращенное описание, т.е. только имя A, B, C …

Массивы слов (например, запоминающее устройство) описывается в виде: $M[m_1{:}m_2](n_1{:}n_2)$. Здесь М - имя массива, m_1,m_2 - номера первого и последнего элемента (ячейки) массива соответственно, n_1, n_2 - определены выше. Пример описания локальной памяти как массива регистров: ЛП[0:15](0:31) - 16 тридцатидвухразрядных регистров.

Описание МО. Для описания МО используется оператор присваивания «:=» (или «←»). Слева от оператора указывается слово, поле, составное слово или элемент массива. Справа - двоичное выражение, которое описывает правило получения результата МО. Запись двоичного выражения

осуществляется при помощи символов, обозначающих различные операции над операндами (словами), представленными в двоичной форме. Например, + - сложение кодов, $\vee$ - логическая операция или и т.п. Примеры - в МП умножения.

Описание ЛУ. Для описания ЛУ используется различного рода отношения: ”<” - меньше, ”>” - больше, ”=0” - равно нулю, “≠0” - не равно нулю и т.п.

Примеры: A<0, B≥0, C=0 и т.п.

Порядок выполнения МО. Порядок выполнения МО и проверки ЛУ задается в графической форме - в виде так называемой **граф-схемы алгоритма** (ГСА) (смотри пример МП умножения). ГСА строится с использованием вершин четырех типов: начальной, конечной, операторной и условной и дуг, связывающих эти вершины (рисунок 4.7.).

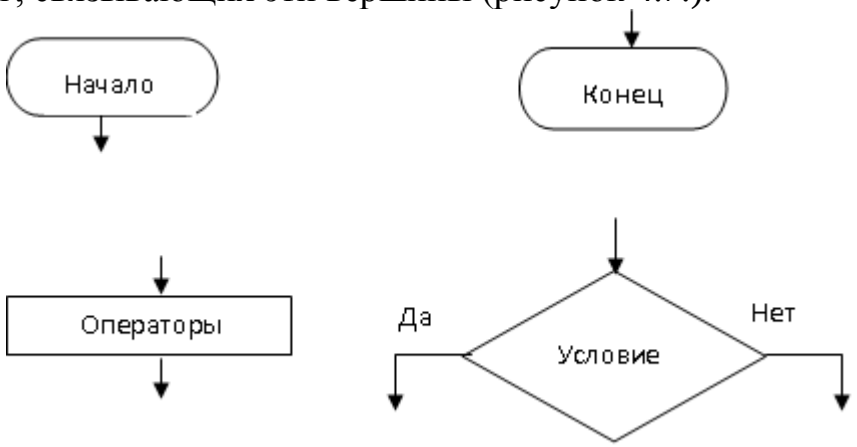


Рис.4.7.– Вершины ГСА

Начальная вершина имеет одну выходящую дугу. Конечная вершина имеет одну входящую дугу. Операторная вершина имеет одну входящую и одну выходящую дугу. В ней записывается один или несколько операторов присваивания, описывающих МО. Условная вершина имеет одну входящую и две исходящих, отмеченных символами «да» (1) и «нет» (0). Выход по дуге «да» осуществляется в случае, если ЛУ принимает истинное значение (1), и по дуге «нет» - если ложное значение (0).

4.4. Структурная организация операционных устройств

ОУ состоит из двух 2 частей – **операционного автомата** (ОА) и **управляющего автомата** (УА) (рисунок 4.8.).

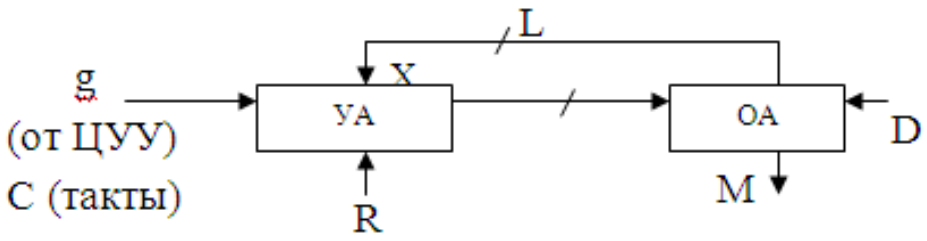


Рис.4.8.– Структура ОУ

Назначение ОА – выполнение микроопераций из списка $Y=\{y_1,\ldots,y_M\}$ под воздействием управляющих сигналов $y_m \in Y$ и формирование значений логических условий (осведомительных сигналов) $X=\{x_1, \ldots, x_L\}$.

С каждым сигналом $y_m \in Y$ в ОА отождествляется определенная МО. Например, МО сложения в МП умножения y_3: C:=C+A. Поступление сигнала y_3 в ОА приводит к выполнению этой МО и записи её результата в С.

С каждым логическим условиям $x_l \in X$ в ОА отождествляется значение осведомительного сигнала, который принимает значение истина (единица) или ложь (ноль). Например, в МП умножения логическое условие (ЛУ) x_2: СЦ=0 - это булева функция, которая принимает значение “истина” ($x_2=1$), если СЦ=0, и ложь ($x_2=0$), если СЦ≠0.

Управляющий автомат (УА) предназначен для управления работой ОА. Он задает порядок выполнения МО в ОА путем выработки управляющих сигналов $y_m \in Y$ в той последовательности, которая задается микропрограммой операции $f_g \in F$ и значениями осведомительных сигналов $X=\{x_1,\ldots x_L\}$, поступающих из ОА. Например, если в ОУ (на вход «g» УА) подан код операции умножения, то управляющие сигналы вырабатываются в соответствии с алгоритмом (микропрограммой) умножения.

Таким образом, ОА является исполнительной (пассивной) частью ОУ, а УА - управляющей (активной) частью ОУ.

Идея декомпозиции ОУ на ОА и УА принадлежит академику В. М. Глушкову. Конструктивность идеи заключается в следующем. Разделение ОУ (АЛУ прежде всего) на две части - пассивную исполнительную (ОА) и активную управляющую (УА) - это не волевое разделение, а объективное.

В.М. Глушков увидел, что в АЛУ часть узлов (сумматор и др. КС) являются исполнителями, а часть узлов (распределитель сигналов, например) являются «командирами». После этого разделения выяснилось, что у них разные принципы построения, организации, причем принципы построения ОА сложнее, чем УА. Кроме того, поскольку принципы организации разные, то и их проектирование тоже раздельное.

Таким образом, идея Глушкова позволила ввести очередной уровень иерархии для целой группы устройств (АЛУ, КПУ) и обобщить принципы их построения в виде абстрактного понятия - ОУ.

Зная внутреннюю структуру ОУ, можно переходить на следующий уровень иерархии - к рассмотрению внутренней структуры ОА и УА. С этой целью сначала необходимо рассмотреть функции ОА и УА.

4.4.1. Синтез канонической структуры операционного автомата. Классификация операционных автоматов.

Функция ОА считается заданной, если заданы (определены) три множества S,Y,X:

- $S=D \cup R \cup I$ - множество слов , где D - множество входных слов ОА (в МП умножения - это два слова A и B); R - множество выходных слов-результатов (в нашем примере это слово C); I - множество внутренних слов (в примере с умножением это слово СЦ).
- $Y=\{y_1, ..., y_m\}$ - множество (список) микроопераций (в примере с умножением - это $y_1, ..., y_7$).
- $X=\{x_1, ..., x_L\}$ - множество (список) ЛУ (в примере это x_1, x_2, x_3).

Объединение производится по всем операциям ОУ:

$$\mathbf{S}=\bigcup_{g}\mathbf{Sg}, \quad \mathbf{Y}=\bigcup_{g}\mathbf{Yg}, \quad \mathbf{X}=\bigcup_{g}\mathbf{Xg}.$$

В общем случае МО описывается выражением y_m: $S_i:=\varphi_m(S_j, S_n)$. Здесь: φ_m - некоторая вычислимая функция (например, сумма), S_i, S_n - её аргументы, S_i - значение функции φ_m, вычисленное при заданных значениях аргументов $S_j = S_j^*$, $S_n = S_n^*$ (например: $S_j^* = 5$, $S_j^* = 10$, $S_j = 5+10=15$).

В общем случае ЛУ описывается выражением: $x_l:=\varphi_l (S_j)$, где φ_L - булева функция, S_j - аргумент функции φ_l, x_l - её значение при $S_j=Sj^*$.

Пример: $x_2 = 1$, если СЦ = 0, или $x_2 = 1$,если СЦ $\neq 0$.

Набор ЛУ $X=\{x_1, \ldots, x_L\}$ отображает состояние ОА.

Следует отметить, что время не является аргументом функции ОА. Это означает, что функции ОА устанавливают только список действий Y и формируемых осведомительных сигналов X. Порядок выполнения МО и формируемых ЛУ в функциях ОА не указывается, отсутствует. Это означает, что функции ОА характеризуют только те средства, которые могут быть использованы для обработки информации, но не сам вычислительный процесс. Порядок выполнения МО - вычислительный процесс (динамику) задают микропрограммы операций множества F: $МП_1, \ldots, МП_G$, реализуемые УА (совместно с ОА, разумеется). Следовательно, совокупность микропрограмм $МП_1, \ldots, МП_G$ задает функцию УА. Функция УА задана, если заданы (описаны, выбраны, разработаны) МП всех операций $F=\{f_1, \ldots, f_G\}$.

Организацию УА рассмотрим позже. Алгоритмы выполнения операций в ОУ обычно задаются в форме граф-схем алгоритмов (ГСА). Именно ГСА (микропрограмма) задает вычислительный процесс во временном аспекте, как последовательность действий (шагов), ведущая к результату.

Определив функцию ОА, можно переходить к его структуре. Сначала продолжим пример с умножением. Для реализации умножения ОА должен выполнять следующие функции:

1) $S_{умн}$ = A(0:15), B(0:15), C(0:15), СЦ(1:4)

2) $Y_{умн}$ - y_1:C:=0; y_2: СЦ:=15; y_3: C:=C+A; y_4: C.B:=SHR1(0.C.B); y_5: СЦ:=СЦ-1; y_6: C:=C+1; y_7:C(0):=A(0)⊕B(15);

3) $X_{умн}$ - x_1:B(15); x_2:СЦ=0; x_3:B(0).

Структура ОА для операции умножения представлена на рисунке 5.5.

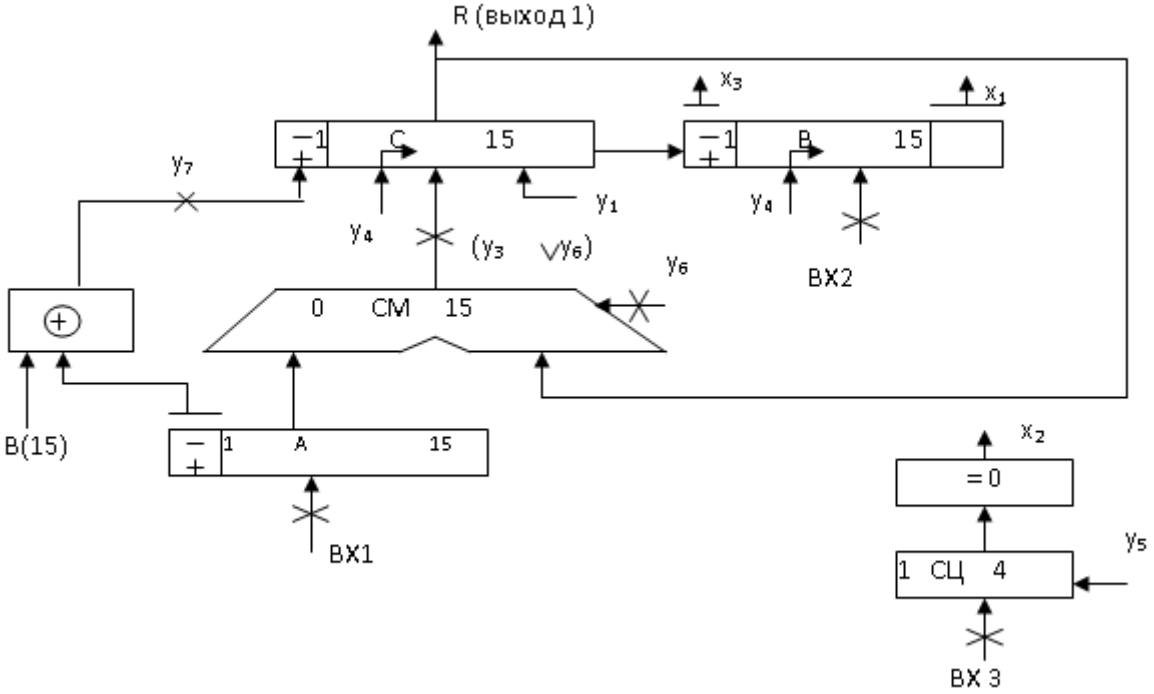


Рис.4.9.– Структура ОА для операции умножения

В общем случае структура ОА имеет вид, представленный на рисунке 4.10. ОА состоит из трех составных частей:

S - память ОА (для хранения слов $S=S_1, \ldots, S_N$),

Ф - множество (набор) комбинационных схем для реализации МО,

ψ - набор КС для формирования осведомительных сигналов.

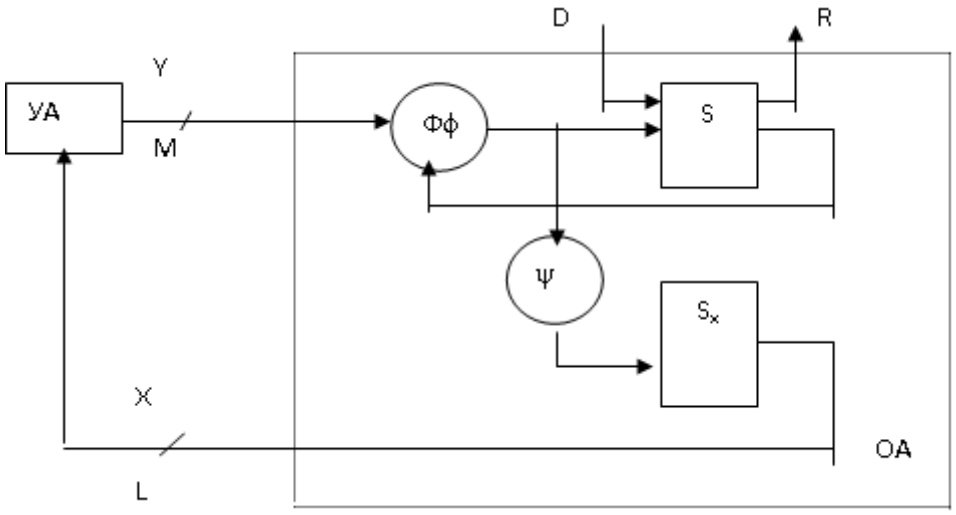


Рис.4.10.– Общий случай структуры ОА

4.4.2. Организация операционного автомата

При организации ОА используются различные способы их построения. Каждый способ построения ОА обеспечивает переход от функции ОА к его структуре, удовлетворяющей следующим требованиям (спецификациям). Основные требования предъявляются к основным характеристикам ОА – быстродействию (производительности) и затратам оборудования на его реализацию. Под производительностью ОА понимается количество МО, выполняемых ОА за один такт.

По способам построения (организации) ОА разделяются на пять классов:

1. ОА с канонической структурой.
2. ОА с максимальной производительностью (I - автоматы).
3. ОА с минимальными затратами оборудования (M - автоматы).
4. ОА с промежуточными характеристиками (IM - автоматы).
5. ОА с памятью (S - автоматы).

Организация ОА с канонической структурой. Такого типа ОА строятся по следующим каноническим правилам:

Словам множества S (S – одна из функций ОА) ставятся в соответствие одноименные регистры : $S_1 \ldots S_n$.

Кроме того, входным словам D множества S ставятся в соответствие входные полюса (входы) $d_1 \ldots d_H$. Каждый вход d_h соединяется с соответствующим регистром шиной.

Входным словам R множества S ставятся в соответствие выходные полюса (выходы ОА) $r_1 ... r_q$. Регистры, соответствующие выходным полюсам, соединяются с выходами с помощью шин.

Каждой МО y_m принадлежащей Y (Y – другая функция ОА) ставятся в соответствие КС ψ_m, входы которой подключаются к регистрам S_A, S_B, а её

выход соединяется с регистром S_C управляемой шиной. Управление шиной отмечается символом y_m (обеспечивается символом y_m).

Каждому осведомительному сигналу x_l, принадлежащему X, ставится в соответствие КС ψ_l, входы которой подключаются к соответствующим регистрам (полям). Выход КС ψ_l отмечается символом x_l.

Пример ОА, построенного по каноническим правилам (для операции умножения) приведен на рисунке 4.11.

ОА с канонической структурой имеет максимальную производительность, так как не вносит ограничений на функциональную совместимость МО и избыточен по затратам оборудования, так как комбинационные схемы могут содержать КС, эквивалентные с точки зрения реализуемых ими функций. Например, два сумматора, входы которых подключены к одному регистру.

Избавиться от избыточности, очевидно, можно, объединив КС с одинаковыми функциями, т.е. оставив по одной схеме с одинаковыми функциями. В результате получим другой тип автомата – автомат типа I (I - автомат). Устранение избыточных КС из канонической структуры ОА, естественно, не вносит ограничений на функциональную совместимость МО. Поэтому I-автомат сохраняет максимальную производительность.

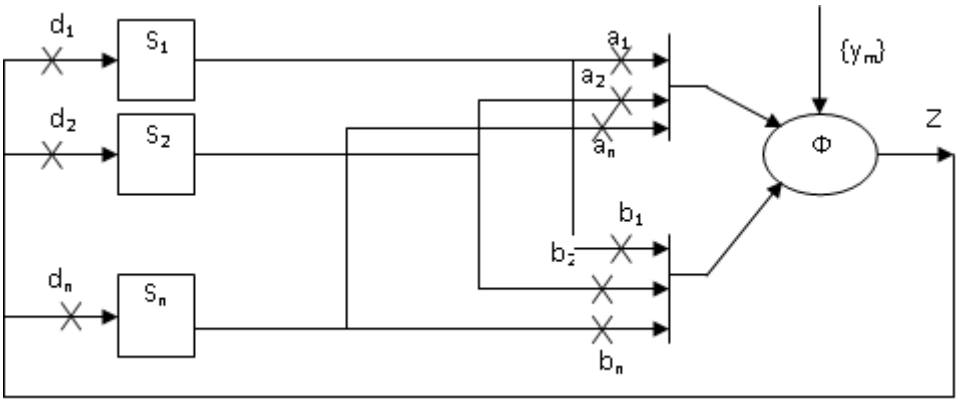


Рис.4.11.– Канонический операционный автомат

ОА типа M (M-автомат) строится с целью минимизации затрат оборудования на их реализацию. Минимизация обеспечивается за счет объединения (“склеивания”) всех КС, выполняющих одинаковые функции на множестве всех КС, реализующих функции Y ОА. Структура ОА типа M представлена на рисунке 4.12.

Особенности структуры M – автомата:

- два мультиплексора (A и B) в структуре,
- одна универсальная КС Ф,
- один демультиплексор,
- в каждом такте выполняется только одна МО (вида $S_k:=\psi_m(s_i,s_j)$), возбуждаемая сигналом y_m из Y. Следует отметить, что одновременно с y_m

необходимо подавать три сигнала: a_j, b_j, d_k, обеспечивающие подачу операндов на входы А и В КС Ф (сигналы a_j, b_j) и прием результатов МО в регистр S_k (сигнал d_k).

Итак, производительность М–автомата минимальна и равна одной МО за один такт, затраты оборудования также минимальны.

Автоматы типов I и M , как нетрудно заметить, обладают противоположными характеристиками. Поэтому ясно, что между ними есть варианты структур ОА, обладающих промежуточными свойствами по производительности и затратам оборудования. Они и образуют класс IM–автоматов. Структурная организация IM–автоматов такова, что она вносит ограничения на совместимость некоторых МО, уменьшая тем самым производительность ОА (относительно I-автомата). Затраты оборудования в IM–автоматах меньше максимальных значений за счет склеивания некоторых КС, реализующих одинаковые МО.

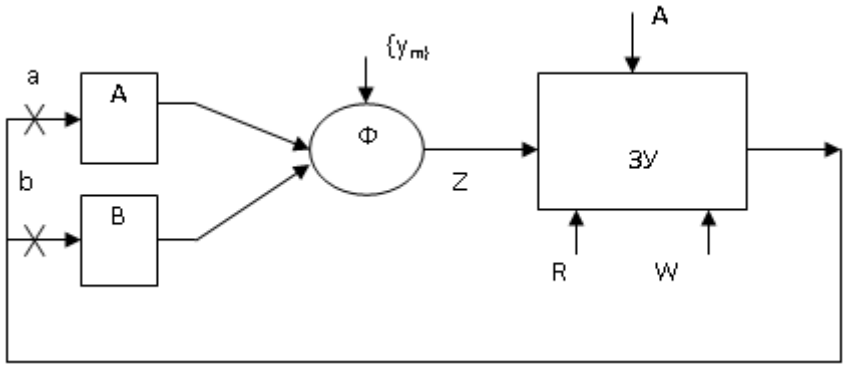


Рис.4.12.– Структура ОА М-типа

В некоторых случаях множество S содержит большое количество слов (сотни). Для уменьшения стоимости таких устройств регистры ОА можно заменять ячейками ЗУ с адресной организацией, если память S организовать как ЗУ. ОА с ЗУ в качестве памяти называется S – автоматом.

Нетрудно заметить, что введение ЗУ существенно упрощает и удешевляет структуру ОА, т.к. в ней нет мультиплексоров А и В, упростился демультиплексор, количество регистров сократилось до двух (А и В). Однако продолжительность такта работы больше, т.к. выборка операндов из ЗУ занимает больше времени, чем из регистров, следовательно быстродействие S–автоматов ниже всех остальных. Для уменьшения продолжительности такта вместо ЗУ с одним адресным входом А и одним информационным выходом S можно использовать ЗУ с двумя адресными входами (А и В) и двумя информационными выходами, что обеспечивает одновременную выборку двух операндов.

4.5. Структура микропроцессорных устройств и систем

Любая МПС состоит из МП, системы памяти, системы ввода-вывода информации и системы сопряжения с объектом управления или контроля.

Микропроцессор играет роль центрального устройства управления и устройства арифметико-логических преобразований данных.

Память физически реализуется в виде системы, которая состоит из нескольких уровней.

Постоянные запоминающие устройства (ПЗУ) предназначены для долгосрочного сохранения предварительно записанных данных и используются только в режиме считывания. Они энергонезависимы.

Оперативные запоминающие устройства (ОЗП) работают в режимах оперативной записи и считывание данных с быстродействием, которое приближается к быстродействию процессора. Они энергозависимы.

Внешние запоминающие устройства (ВЗУ) выполняют функции сохранения больших объемов информации, содержат накопители на гибких и жестких магнитных дисках, компакт-дисках (лазерных) и др.

Устройства ввода данных (УВ) предназначены для передачи данных извне в регистры МП или в память. Среди них – клавиатура, разнообразные пульты управления, магнитные и лазерные диски и др.

Устройства вывода данных (УВыв) предназначены для восприятия данных, передаваемых из регистров МП или ячеек памяти. Это дисплеи, печатающие устройства, ВЗУ, пульты управления, графопостроители (плоттеры) и др.

Для сопряжения объекта управления или контроля с МПУ или МПС в состав оснащения должны входить датчики и исполняющие механизмы. Для их подключения к МПУ ли МПС используют блоки сопряжения, которые выполняют функции согласования интерфейсов. Иногда эти блоки называют устройствами связи с объектом (УСО).

4.5.1. Интерфейсы микропроцессорных устройств и систем

Архитектурные возможности МПС в значительной мере зависят от типа интерфейса.

Унифицированный интерфейс *— это совокупность правил, которые устанавливают единые принципы взаимодействия устройств МПС.*

В состав интерфейса входят аппаратные средства соединения устройств (соединители, связи), спецификация номенклатуры и характеристик связей, программные средства, описания характера сигналов интерфейса и их временные диаграммы, а также описание электрофизических параметров сигналов.

Основная задача интерфейса – на основе унификации обеспечить совместимость аппаратных, программных и конструктивных средств,

которые предопределяют заданное качество автоматического взаимодействия разных функциональных элементов в едином процессе обработки информации в МПС на этапах сбора, преобразования, сохранения и выдачи результатов и управляющих действий.

Архитектура МПС определяется преимущественно тремя интерфейсными уровнями: системным, межмашинным и малым интерфейсом (интерфейсом периферийных устройств).

Системный интерфейс обеспечивает объединения основных модулей (блоков) МПС в единую систему на равные обмена информацией с процессором и ОЗП.

Системные интерфейсы разделяют на сосредоточенные (интерфейсы ПЭВМ), локально-сосредоточенные (Q-bus) и локальные (Unibus).

Межмашинный интерфейс обеспечивает построение мультипроцессорных систем и локальных и распределенных систем и сетей.

Малые интерфейсы учитывают отличие физических принципов работы групп периферийных устройств и ПЗУ. Контролеры малых интерфейсов обеспечивают выход на системный интерфейс. При этом контролеры периферийных устройств и ПЗУ выходят на соответствующий малый интерфейс.

4.5.2. Управление работой микропроцессорных устройств (систем)

Временное согласование информационных сигналов в МПУ осуществляется с помощью специальных сигналов, поступающих от устройства управления МП. МПУ или МПС функционирует синхронно с появлением тактовых сигналов. Простейшее действие, которое выполняется в МПУ (МПС), называется *состоянием*. Он охватывает один период сигнала тактирования — *тактовый интервал или такт.*

Определенное количество тактовых интервалов составляет *машинный цикл.* Для одного обращения к памяти или к устройству ввода/вывода необходим один машинный цикл. За один цикл осуществляется выборка команды или данных, а также кода адреса (возможно, байта команды или данных и байта кода адреса).

Машинный цикл — часть команды (иногда полностью команда). С началом каждого машинного цикла на выводе синхронизации МП возникает сигнал синхронизации. Он передается в запоминающее устройство (ЗУ) и(или) в устройство ввода/вывода (УВВ) и «извещает» о начале нового машинного цикла, в результате чего достигается согласование во времени действия этих устройств с работой МП.

Цикл команды — интервал времени, необходимое для выборки команды из памяти и его выполнения. Он состоит из одного или нескольких машинных циклов, их количество, как правило, равняется

количеству обращений МП к памяти или к одному из УВВ. Структура команды показана на рисунке 4.13.

Рис.4.13.– Структура команды

Управляющее устройство выполняет функции управления и синхронизации, то есть контролирует изменение состояний МП в необходимой последовательности, согласовывая их с сигналами тактового генератора. Он состоит из управляющего конечного автомата, предназначенного для управления процессами внутри МП, и схемы, которая, получая сигналы извне, вырабатывает сигналы, которые управляют системой.

Код команды дешифруется, превращаясь в двоичные сигналы, которые воздействуют на модули и блоки МП, принимающие участие в выполнении этой команды.

Цикл команды делится на две фазы: фаза выборки и фаза выполнения.

Фаза выборки – автомат задает начало очередного цикла, по которому число, находящееся в счетчике команд, передается в буферный регистр адреса. Оттуда через шину адреса код адреса команды направляется в память, где дешифруется. За сигналом «считывания» из ячейки памяти слово команды считывается и передается по шине данных в буферный регистр данных, из которого пересылается в регистр команд, а потом дешифруется.

Фаза выполнения – устройство управления генерирует последовательность сигналов, необходимую для выполнения команды. За это время данные счетчика увеличиваются на единицу. Этим формируется адрес следующей выполняемой команды.

Считывание или запись слова происходит на протяжении некоторого интервала времени, который называют временем доступа. Интервал времени, которое расходуется на обращение к памяти и получение от нее сигнала готовности, называется циклом ожидания готовности. Он составляет часть машинного цикла.

Обмен информацией между МП, ЗУ и УВВ реализуется преимущественно в трех режимах: программно управляемого обмена, обмена в режиме прерываний, обмена в режиме прямого доступа.

Программно управляемый обмен. В этом режиме МП определяет, готово ли ЗУ или периферийное устройство (ПУ) к выполнению операции ввода-вывода, к началу программной передачи данных. УВВ должны иметь аппаратные средства для выработки сигналов о внутреннем состоянии. МП считывает эту информацию и на основании анализа результата делает вывод о готовности устройства к обмену информацией. В дальнейшем в соответствии с протоколом интерфейса происходит обмен данными.

Режим прерывания. Используется при необходимости немедленной передачи данных между УВВ и МП (реакция на неожиданное возникновение внешних условий). При этом МП должен прервать работу основной программы и начать выполнять программу обслуживания внешнего устройства. Такой режим называют прерыванием. Прерывание МП возможны только тогда, когда МП разрешено реагировать на запросы прерывания.

После приема сигнала прерывания МП завершает текущую операцию, передает на сохранение в память всю информацию внутренних регистров данных и управления и переходит к подпрограмме обслуживания прерывания. После окончания обмена информацией по прерыванию восстанавливается состояние МП, которое существовало к началу прерывания.

Различают три вида прерываний: простое, векторное и приоритетное.

Простое прерывание извещает о том, что какое-то устройство ввода/вывода требует обслуживания МП.

Векторное прерывание делает возможным распознать тип (уровень) прерывания, требуемое периферийному устройству. В векторе указывается конкретный адрес устройства.

Приоритетное прерывание состоит в том, что, кроме распознавания прерывания, определяется приоритет в обслуживании прерывающих устройств.

Режим прямого доступа к памяти. Иногда возникает необходимость осуществить обмен информацией вн МП. Это связан с уменьшением затрат времени для обмена массивами данных. В таком случае в состав аппаратной части МПП ли МПС входит контролер прямого доступа к памяти, которая руководит передачей данных, освобождая от этих функций МП.

Устройства прямого доступа к памяти подключаются параллельно процессору. Разделение этих каналов осуществляется с использованием тристабильной логики управления состоянием шин МПС. МП во время прямого доступа к памяти переводит свои исходные схемы в высокоимпедансное состояние и изолируется от системы, что аналогично разрыву информационного канала. Состояние внутренних регистров сохраняется таким, как на момент запроса канала прямого доступа.

Существует несколько способов реализации прямого доступа к памяти. Все они обеспечивают высшую скорость обмена данными

сравнительно с режимом программно управляемого обмена. Чаще всего режим прямого доступа к памяти реализуется с остановкой МП и увеличением (удлинением во времени) цикла МП.

Метод остановки основан на том, что в этом состоянии МП отключается от системных шин на время передачи данных. Перед переходом в состояние остановки МП завершает выполнения текущей команды и задерживается в этом состоянии на несколько тактов, до момента, когда шины освободятся. По такой схеме прямого доступа к памяти МП, будучи отключенным от шин, не реагирует на прерывания, что в некоторых случаях может быть неприемлемым для МПС.

Метод захвата состоит в последовательном обмене данными. Быстродействующие УВВ обмениваются только одним словом; их запрос на обслуживание удовлетворяется путем задержки выполнения текущей команды на один машинный цикл, когда МП находится в состоянии перехода от одного машинного цикла к другому. В этом режиме прямого доступа к памяти, МП приостанавливается только на один машинный цикл для передачи каждого слова данных, после чего управления возвращается МП.

4.6. Микропроцессоры

Термин ‘микропроцессор’ (МП) появился в связи с созданием интегральных схем (ИС), реализующих основные функции процессора (фон Неймановского): автоматическое выполнение программы путём поочерёдной их выборки из памяти (ОЗУ) по адресу, который формируется в счётчике команд и выставляется на ША МП. Выбранная из ОЗУ команда принимается процессором и загружается в регистр команд – для хранения и исполнения. В соответствии с адресной частью команды извлекаются операнды из ОЗУ или внутренних регистров процессора, над ними выполняется операция (в АЛУ), а затем – засылка результата.

Степень интеграции была недостаточной, чтобы на одном кристалле (в одном корпусе) помещались все блоки процессора. Поэтому ИС выпускались в виде набора, комплекта микросхем, из которых можно было собрать процессор целиком. Таким образом, микропроцессор – это одна или несколько БИС, обеспечивающих выполнение функций процессора.

В комплект ИС, кроме схем, реализующих процессорные функции, входят и другие БИС, необходимые для построения ЭВМ и систем на основе ЭВМ: БИС ОЗУ, БИС ПЗУ, БИС управления, интерфейсные БИС и др.

ИС в микропроцессорном комплекте (МПК БИС) объединяются по принципу совместимости: конструктивной, функциональной, электрической.

ИС, из которых строится процессор, образуют т. н. **базовый комплект ИС**. Если он состоит из одной БИС, то комплект называют однокристальным (примеры: i4004, i8008, i8080 (К580), i8086 (К1810), i80286, ...).

4.6.1. Многокристальный МПК БИС

Содержит несколько типов ИС, на основе которых строится (собирается) процессор (примеры: К584, К1804 и др.).

В первом случае процессор обычно имеет фиксированную разрядность АЛУ, шины адреса, данных и др., а также фиксированную систему команд: МПК БИС с фиксированной системой команд.

Во втором случае обеспечивают возможность строить АЛУ заданной разрядности и процессор с заданной системой команд. Систему команд формирует разработчик процессора, а не разработчик ИС. В базовый комплект в этом случае включаются две основные БИС: 1) МПЭ – АЛУ ограниченной разрядности вместе с РОН (пример: К1804ВС1 – четырёхразрядная МП секция), 2) блок микропрограммного управления также фиксированной разрядности (примеры: К1804ВУ1, ВУ3, ВУ4).

Приставка ’микро’, прежде всего, указывает на малые габариты процессора, на тот факт, что он выполнен в виде БИС (одной или нескольких), т.е. термин ‘микропроцессор’ означает название БИС, реализующей функции ЦП ЭВМ.

К настоящему времени степень интеграции ИС достигла уровня, при котором на одном кристалле МП удаётся разместить не только многоразрядные блоки самого процессора (АЛУ, ЦУУ, РОН), но и дополнительные блоки скрытой буферной памяти – т.н. кэш-памяти значительного объёма (десятки, сотни килобайт).

Далее, если на одном кристалле кроме процессора разместить другие устройства ЭВМ: ОЗУ, интерфейсные блоки для связи с внешним миром (с периферийными устройствами), то в этом случае мы имеем дело с т.н. **однокристальной микро ЭВМ (микроконтроллером).**

4.6.2. Принцип микропрограммного управления.

В общем случае между программными и аппаратными средствами четкие границы отсутствуют. В большинстве современных ЭВМ непосредственная связь между аппаратурой и программными средствами осуществляется через микропрограммный уровень. Любая машинная команда исполняется аппаратурой не непосредственно, а путем их интерпретации в соответственную последовательность более простых действий. А значит, всегда существует задача программирования машинных команд из более простых действий – микропрограммирование. Впервые этот термин был введен в 1953 году специалистом по ВТ Уилксом. Но это было применимо только к аппаратным средствам. Примерно в середине 60-х годов, усилиями разработчиков IBM, идеи Уилкса превратились в принцип организации ВМ. Микропрограммирование обеспечило переход к модульному построению ЭВМ. Развивая идеи микропрограммирования,

Глушков показал, что в любом устройстве обработки информации функционально можно выделить операционный автомат и управляющий автомат. На этом уровне структура любой информации:

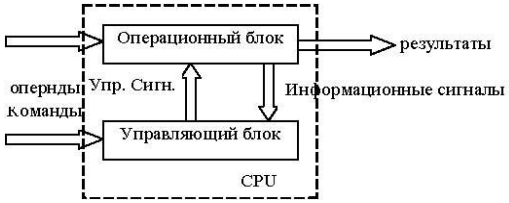


Рис. 4.14.– Структура любой информации

Управляющий блок выдает последовательность сигналов, которые обеспечивают выполнение данной команды. Информационные сигналы зависят не только от исходных значений обрабатываемых данных, но и от результатов, получаемых в процессе обработки.

Порядок функционирования устройства базируется на следующих положениях:

- Любая машинная команда рассматривается, как некоторое сложное действие, которое состоит из последовательности элементарных действий над словами информации – микроопераций.
- Порядок следования микроопераций зависит не только от значений преобразуемых слов, но также от их информационных сигналов, вырабатываемых операционным автоматом. Примерами таких сигналов могут быть признаки результата операции, значения отдельных битов данных и т.п.
- Процесс выполнения машиной команды описывается в виде некоторого алгоритма в терминах микроопераций и логических условий. Описание информационных сигналов – микропрограмма.
- Микропрограмма служит не только для обработки данных, но и обеспечивает управление работой всего устройства в целом – принцип микропрограммного управления. Операционный блок обеспечивает выполнение определенного набора микроопераций и вычисление необходимых логических условий. Управляющий автомат, согласно заданной машинной команде, генерирует необходимую последовательность сигналов, инициирующих соответствующие микрооперации, согласно микропрограмме и значениями логических условий, формируемых операционным в ходе обработки по микропрограмме. Таким образом, микропрограммы выступают, с одной стороны, в роли закона по которому выполняется обработка, с другой стороны, закон, по которому работает управляющий блок.

4.6.3. Описание функциональных микропрограмм.

Существуют языки микропрограммирования, которые обеспечивают описание законов функционирования микропроцессора. Все функциональные блоки рассматриваются на уровне регистров. Языки микропрограммирования являются сильно ориентируемыми на конкретную структуру обработки информации. Основным оператором языка является микрокоманда, которая состоит из нескольких функционально совместимых микроопераций, т.е. порядок выполнения операций не влияет на конечный результат. Совместимость микроопераций зависит не только от действий, но и от структуры операционного автомата. С точки зрения языка для описания объектов используются конструкция идентификатор. Эти идентификаторы используются с конкретным числом битов того или иного данного. Отдельные поля регистров могут иметь собственное обозначение (идентификатор). Любая микрооперация, как некоторый акт преобразования данных, записывается в виде оператора :=. Сами по себе операции по преобразованию могут быть арифметическими, логическими, функциональными. Многофункциональное действие может быть использовано только в случае, если в операционном автомате есть соответствующий аппаратурный элемент. Основной единицей информации является слово. Микропрограммы пишутся с точки зрения слов. Любая микрокоманда выполняется за 1 такт машинного (автоматного) времени. Микрокоманда записывается всегда в 1 строку. Сама микрокоманда записывается в виде отдельных микрооператоров. Для того, чтобы организовать ветвления, используются условные микрооператоры – if, как полный так и укороченный вариант, вплоть до операторов условного присваивания := (условие). Для наглядности представления микропрограмм используется графическое представление – схема алгоритма микропрограммы. В основе вычерчивания схем лежат общие требования, которые определяются стандартами группы ГОСТ 19… -группа единой системы программной документации. При графическом представлении учитывается, что в микропрограмме количество различных типов вершин в схеме алгоритмов резко ограниченно.

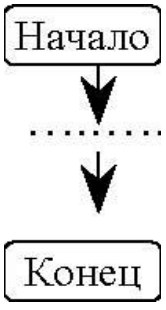


Рис.4.15.– Графическое представление МП

Способы кодирования МО:
1) горизонтальное микропрограммирование
2) вертикальное микропрограммирование
3) смешанное микропрограммирование Ym = {y0,…ym} – номер МО

4.6.3.1. Горизонтальное микропрограммирование

Любому сигналу ставится 1 бит в операционной части МК (для примера 1 – выполняется 0 – нет)

V1 V2 … Vm-1Vm

В любой МК может выполняться любая МО

Достоинства:

1) Если нужен ввод новой МК, то никаких схемотехнических изменений нет.

2) Добавление новой МО к существенному увеличению схемы УУ не приводит

3) Нет схемы дешифрации Недостатки: 1) Сложность (длинна операционной части МК определяется количеством МО, следовательно, емкость памяти МК увеличивается) 2) Сложность доступа 3) Из операционной части полезную информацию несет 10-15%

Горизонтальное микропрограммирование применяется в простых устройствах обработки с малым количеством МО.

4.6.3.2. Вертикальное микропрограммирование

Любой сигнал управления задается значением всего кода операционной части. Достоинства: 1) $n_{оч}$ =]log2(M+1)[(длина операционной части) 2) появление новых МК к принципиальным трудностям не приводит 3) емкость памяти МК достаточно небольшая. Недостатки: 1) нужна дешифрация 2) две одинаковых МО в одной МК не могут быть, следовательно, количество МК возрастает.

4.6.3.3. Смешанное микропрограммирование

Весь набор микроопераций разбивается на некоторое количество подмножеств, причем необязательно, чтобы подмножества были непересекающимися, тогда операционная часть МК будет состоять из нескольких подмножеств. Длина любого поля $n_{iоч}$ = [log2(Mi+1)]. Длина операционной части nоч = $\sum n_{iоч}$. Внутри подмножества любая МО кодируется вертикальным способом, может реализовываться способ горизонтально – вертикального программирования.

Вопросы для самопроверки.

1. За счет чего можно повысить быстродействие процессоров?
2. Охарактеризуйте процессор с разнесенной архитектурой.
3. Что такое открытая система?
4. Охарактеризуйте суперскалярный процессор.
5. Дайте характеристику микропроцессора P6 INTEL.
6. Приведите принцип микропрограммного управления.
7. Приведите структуру операционных устройств.
8. Какова классификация операционных автоматов?
9. Приведите принципы организации операционного автомата.
10. Приведите интерфейсы микропроцессорных устройств и систем.

Модуль 5.

УСТРОЙСТВА УПРАВЛЕНИЯ ЭВМ

Цель модуля: изучение студентами устройств управления, управляющих автоматов их структуры и функции.

В результате изучения модуля студенты должны знать:

- организацию управляющих автоматов с жесткой и программируемой логикой;
- характеристики устройств управления.

Содержание модуля:

5.1 Организация управляющего автомата

5.1.1 Организация управляющего автомата с программируемой логикой управления.

5.1.2 Укрупненная структура управляющего автомата с программируемой логикой.

5.1.3 Управляющие автоматы с жесткой логикой управления.

5.2 Сравнение характеристик управляющих автоматов с программируемой и жесткой логикой.

При построении УА используются два основных способа: 1) схемная (‘жёсткая’) логика управления и программируемая (‘мягкая’) логика управления.

5.1. Организация управляющего автомата

5.1.1. Организация управляющего автомата с программируемой логикой управления

ОУ работает во времени тактами. В каждом такте ОА может выполнить одну или несколько совместимых МО. Для этого УА должен вырабатывать в каждом такте один или несколько управляющих сигналов (импульсов). Например:

такт i – y_1,

такт i+1 – y_2, y_5, y_6,

такт i+2 – y_2, y_7, y_8.

Очевидный способ формирования управляющих сигналов состоит в следующем: набору сигналов $y_1, y_2, ..., y_M$ (одна из функций ОА) ставится в соответствие слово $Y= y_1, y_2, ..., y_M$, в котором каждой МО, которая должна выполнятся в данном такте, ставится в соответствие единица, если нет – то ноль, т.е. производится **унитарное кодирование МО**. Пример: M=8, такт i+1, $y_2 = y_5 = y_6 =1$, остальные нули. Это слово используется для управления

(элементами И): 1 – элемент И открыт для прохождения импульса с ГТИ, 0 – закрыт (рисунок 5.1).

Чтобы в следующем такте можно было вырабатывать другие сигналы, достаточно сменить слово Y на входе этой схемы.

Двоичный код Y, на основе которого вырабатываются управляющие сигналы, естественно называют управляющим словом или микрокомандой (МК). Алгоритм операции fgєF, описанный в терминах МК, называют микропрограммой. Для хранения микропрограммы естественно использовать ПЗУ. Из ПЗУ МК извлекаются поочерёдно и используются для выработки управляющих сигналов ym.

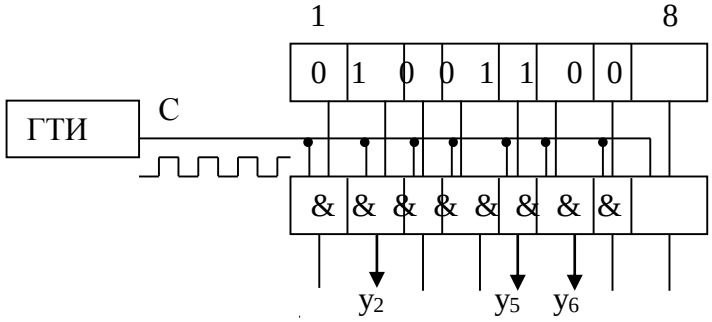


Рис.5.1. – Схема формирования управляющих сигналов

Идея формирования управляющих сигналов на основе МК, извлекаемых из ПЗУ, принадлежит англичанину Уилксу. УА, построенный в соответствии с идеей Уилкса, и называется УА с программируемой логикой (ПЛ) управления. Со времени своего рождения (начало 50-х годов) идея Уилкса претерпела значительные изменения, усовершенствования (эволюцию).

Усовершенствования вызваны следующими обстоятельствами: при большом числе управляющих сигналов (М – десятки, сотни) использование идеи Уилкса, т.е. унитарного кодирования МО, ведёт к большой длине МК и, следовательно, к большой ёмкости ПЗУ для хранения МП.

В связи с этим встаёт естественная задача сокращения длины МК. Самый простой способ сокращения длины МК – использование позиционного кода вместо унитарного: т.н. позиционное кодирование МО. Пример: М=8, y_5=1, остальные сигналы равны нулю (рисунок 5.2). Пусть y_5 кодируется кодом 5=101.

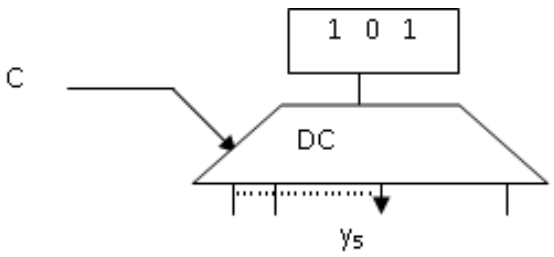


Рис.5.2.– Способ сокращения длины МК

Длина позиционного кода m= $\log_2 M$ ($M=2^m$). Недостаток позиционного кодирования очевиден: в каждом такте можно вырабатывать только один сигнал. Это нежелательно, т.к. вносит ограничения на совместимость МО и снижает производительность ОА.

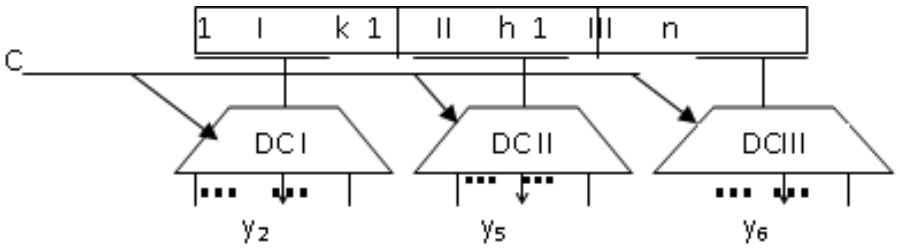


Рис.5.3.– Смешанное кодирование

Поэтому на практике обычно применяется третий способ кодирования МО – т.н. **смешанное кодирование**. Суть: всё множество МО Y= y_1, y_2, ..., y_M разделяется на группы несовместимых МО (в каждую группу включаются несовместимые МО). Количество групп МО определяется максимальным количеством совместимых МО, выполняемых в одном такте. Каждой группе МО ставится в соответствие поле МК.

В этом поле указывается позиционный код одной МО из этой группы (рисунок 5.3).

Пример: три группы МО - первая состоит из семи МО, вторая – из 9, третья – из 18. Разрядность полей: k=3 ($\log_2 7$), h=4 ($\log_2 9$), n=5 ($\log_2 18$). Всего: 3+4+5=12 разрядов вместо M= 36 (7+9+18) при унитарном кодировании.

Итак, при унитарном кодировании длина МК равна M (максимальное значение), при позиционном кодировании m=$\log_2 M$(минимальное значение), при смешанном m<k+h+n<M.

При определённых условиях можно ещё уменьшить длину МК, если использовать четвёртый способ кодирования – т.н. **косвенное кодирование МО**.

Суть: кодированию подвергаются не отдельные МО, а наборы совместимых МО. Пример: набор МО $Y_i=y_2y_5y_6$ кодируется кодом длиной l=$\log_2 L$, где L – количество наборов совместимых МО (рисунок 5.4). Условие применения косвенного кодирования: $\log_2 L < k+h+n$.

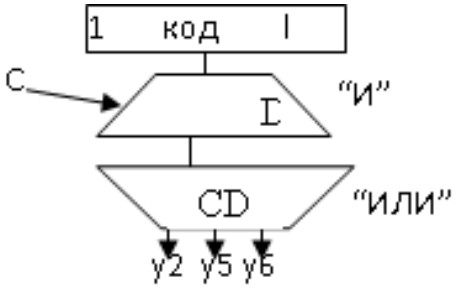


Рис.5.4.– Косвенное кодирование

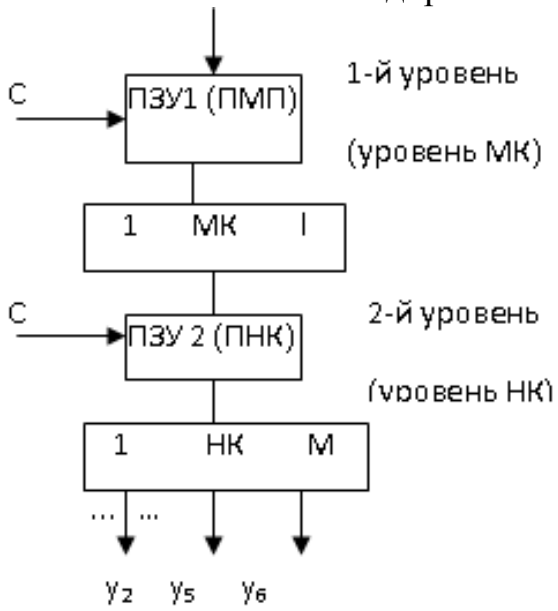


Рис.5.5.– УА с косвенным микрокодированием

Следует отметить, что вместо связки из дешифратора DC и шифратора CD можно использовать т. н. программируемую логическую матрицу (ПЛМ) или ПЗУ. В случае использования ПЗУ в ячейках ПЗУ, количество которых L (совпадает с количеством групп МО), размещаются т.н. нанокоманды (НК) – унитарно закодированные МО y_1, y_2, ..., y_M (длина ячеек ПЗУ – M разрядов). В этом случае структура УА приобретает вид (рисунок 5.5).

В ней используется ПЗУ на двух уровнях кодирования: на уровне МК (позиционное) и на уровне НК (унитарное). В ПЗУ1 хранятся микрокоманды операций f_1, …, f_g ОУ, в ПЗУ2 – нанокоманды совместимых наборов МО. По этой причине косвенное кодирование иначе еще называют двухуровневым кодированием (в отличие от одноуровневого смешанного кодирования).

Итак, назначение УА – выработка управляющих сигналов в зависимости от значений ЛУ x_1, …, x_L, поступающих из ОА. Поскольку сигналы y_1, y_2, ..., y_M фактически хранятся в ПЗУ в виде управляющих слов – МК, то порядок их выборки (чтения) из ПЗУ нужно поставить в зависимость от осведомительных сигналов: если, например, x_i=1, то выбирается одна МК, если x_i=0, то другая.

Для этого в МК достаточно указать i – номер осведомительного сигнала в списке x_1, …, x_L в специальном поле – в поле X МК, а в третьем поле A необходимо указать адрес перехода. Тогда формат команды приобретает вид:

1 Y m	1 X l	1 A k

Адрес очередной команды, извлекаемой из ПЗУ, в простейшем случае формируется следующим образом:

$$A_{MK} = \begin{cases} A.0, \text{ если } X = 0 \\ A.0, \text{ если } X \neq 0,\ x_X = 0 \\ A.1, \text{ если } X \neq 0,\ x_X = 1 \end{cases} \qquad (5.1)$$

Здесь A – k-разрядное поле (адрес) из микрокоманды, к которому добавляется еще один бит – k+1-й: $E^{ПЗУ}=2^{k+1}$- емкость ПЗУ.

В поле X заносится ноль, если организуется переход от одной МК к другой на линейных участках микрокоманды: это т. н. принудительный (безусловный) переход. В случае условного перехода в поле X МК указывается номер l=X осведомительного сигнала x_l из набора x_1 … x_L, значение которого используется в качестве условия альтернативного перехода в алгоритме. Если условие выполняется ($x_l = x_x = 1$), то адрес – A1 – нечетный, если не выполняется ($x_l = x_x = 0$), то адрес –A.0 – четный.

Рассмотренный (первый) способ формирования адреса МК называется принудительной адресацией МК. Простейшая структура УА с ПЛ и с принудительной адресацией МК изображена рисунке 5.6. Порядок функционирования этой схемы иллюстрируется временной диаграммой, представленной на рисунке 5.7.

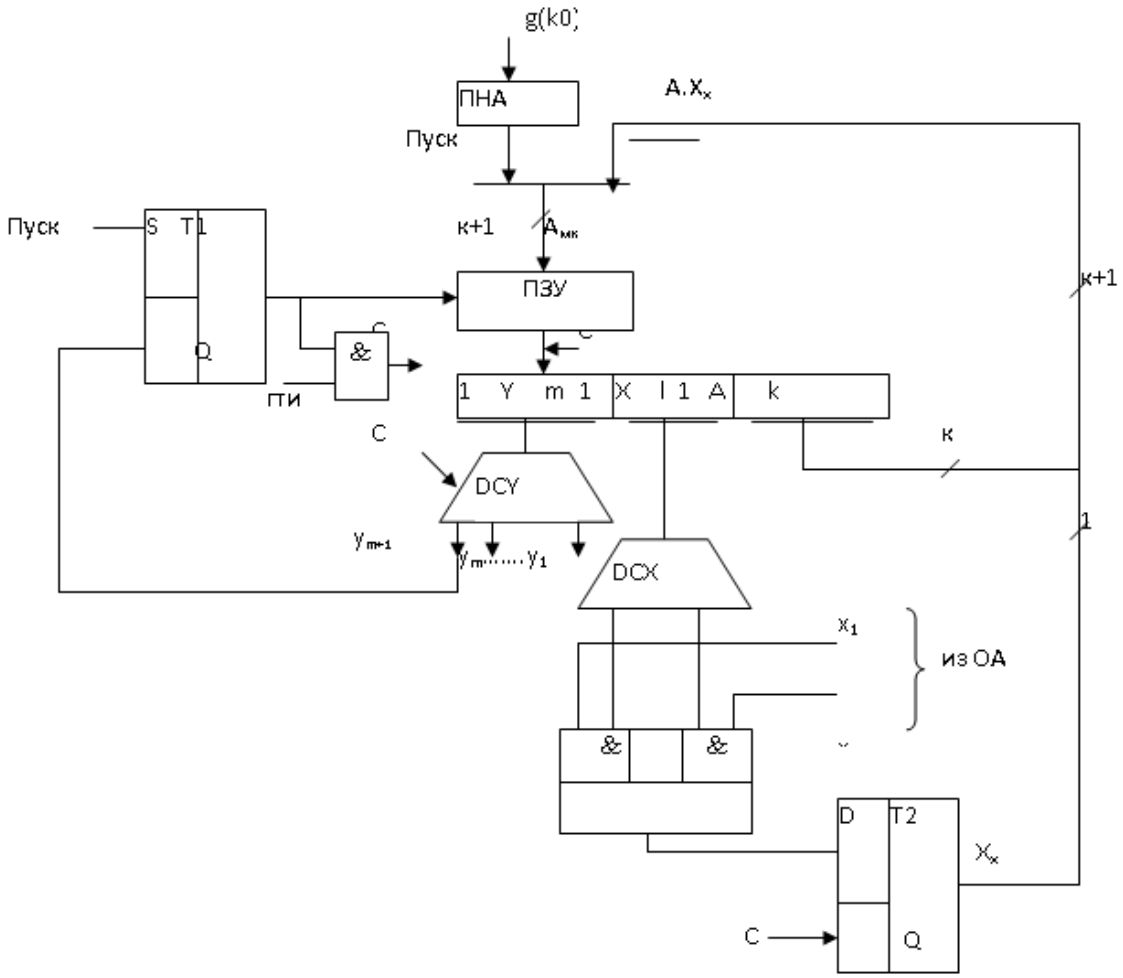


Рис.5.6.– Структура УА с принудительной адресацией

ЦУУ процессора, расшифровав команду, посылает код операции g в соответствующее ОУ (например, в АЛУ) и сигнал “ПУСК”.

По сигналу ПУСК УА ОУ (например, АЛУ) выбирает первую микрокоманду из ПЗУ по адресу, который формируется преобразователем начального адреса ПНА на основе кода операции g.

По фронту сигнала С МК заносится в регистр МК (РМК) для исполнения. На основе поля Y вырабатываются управляющие сигналы y_m для ОА. На основе полей X, A формируется адрес следующей МК в соответствии с формулой (5.1).

По завершении этапа выполнения МО в ОА и формирования осведомительных сигналов один из них x_l заносится (по спаду сигнала С) в триггер Т2 и используется при формировании адреса в следующем такте.

Недостатки принудительной адресации: неэффективное использование ПЗУ из-за четной адресации ячеек - линейная цепочка МК в ПЗУ занимает только четные ячейки (нечетные пустуют). Чтобы избавиться от этого недостатка, можно использовать другой формат МК:

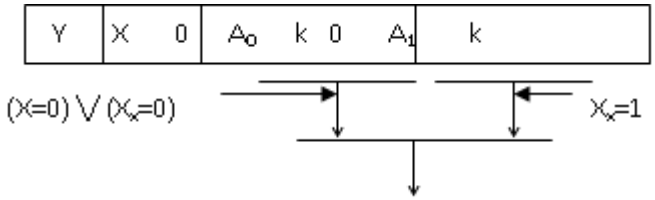


Рис.5.7.– Оптимизированный формат микрокоманды

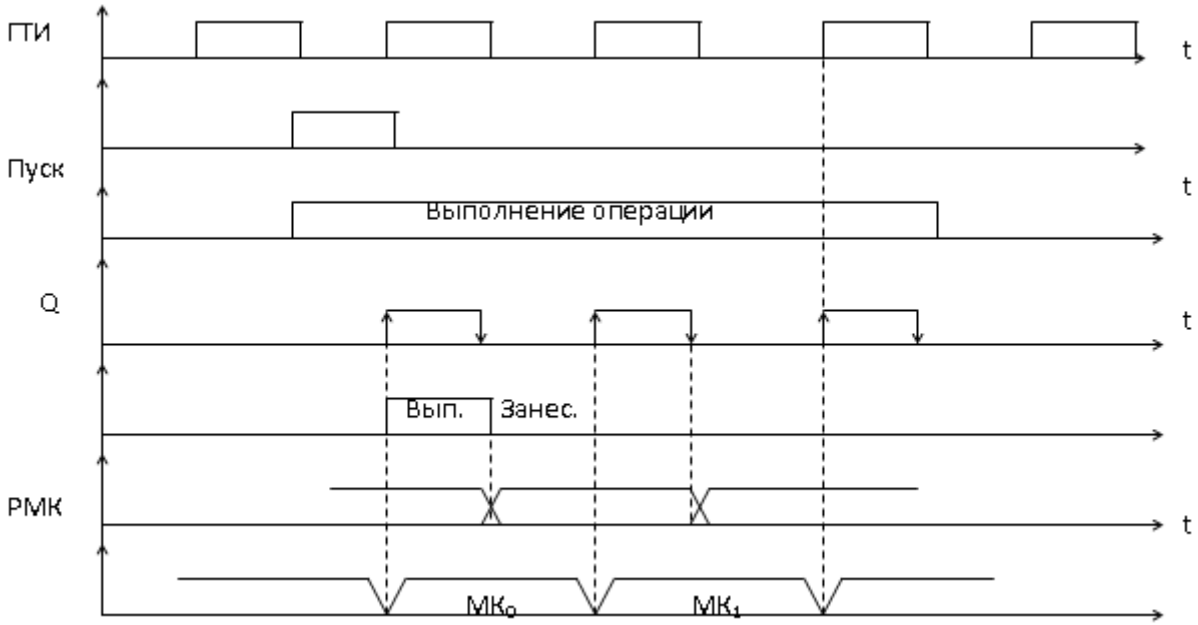


Рис.5.8.– Временная диаграмма

В этом случае в ПЗУ используются все ячейки: и четные и нечетные.

$$A_{МК} = \begin{cases} A0, & если \quad X = 0 \\ A0, & если \quad X \neq 0, \quad x_x = 0 \\ A1, & если \quad X \neq 0, \quad x_x = 1 \end{cases} \qquad (5.2)$$

Однако длина МК увеличилась за счет дополнительного поля адреса A_1, кроме того, в случае принудительного (не условного) перехода поле A_1 не используется совсем. Это означает, что длину МК можно уменьшить. Как? Путем использования естественной (как у Неймана в процессоре) адресации МК на линейных участках. Для этого в состав УА необходимо ввести счетчик МК (СМК) и использовать его для адресации МК в соответствии с выражением:

$$СМК = \begin{cases} A, & если \quad X = 0 \ (БП) \\ СМК + 1, & если \quad X \neq 0, \quad x_x = 0 \\ A, & если \quad X \neq 0, \quad x_x = 1 \end{cases} \qquad (5.3)$$

где А – адресная часть МК.

УА, в котором формирование адреса следующей МК осуществляется на основе СМК, называют УА с естественной адресацией МК.

Следует отметить, что при естественной адресации нет необходимости вводить адресное поле А в каждую МК.

Адресное поле А необходимо только в МК перехода – условного или безусловного.

В результате множество МК делится на два типа: управления и операционные.

Операционные МК состоят из одного поля Y, управляющие – из полей X, A. Для того, чтобы отличить МК одного типа от МК другого типа, в формат МК вводят бит типа МК – бит P: P=0 – операционная МК, P=1 – управляющая:

1	1 X l	1 A k
0	1 Y	m

МК управления

операционная МК

Длина ячейки ПЗУ определяется из условия: max ((m+1),(k+l+1)).

Достоинство УА с естественной адресацией МК: сокращение длины МК и, следовательно, емкости ПЗУ (затрат оборудования).

Недостаток: использование МК двух типов увеличивает время выполнения всей микрокоманды, т. к. в этом случае невозможно совмещение во времени выполнения МО и переходов в микропрограмме: либо переход, либо обработка. Избавиться от этого недостатка можно, если использовать МК с форматом из трех полей: Y, X, A.

5.1.2. Укрупненная структура управляющего автомата с программируемой логикой

В общем случае (без деталей) УА с ПЛ строится (состоит) из трех составляющих: схемы формирования адреса следующей МК, ПЗУ и блока формирования управляющих сигналов для ОА (рисунок 5.9)

Схема адресации выполняется в виде БИС управления (например К1804 ВУ1, ВУ3), ПЗУ – на основе БИС ПЗУ.

ОА также выполняется в виде отдельных БИС – БИС МПЭ (микропроцессорных элементов). Все эти БИС и входят в состав МПК БИС. Что касается третьего блока, то здесь возможны разные варианты реализации - в зависимости от способа организации работы ОУ во времени. В

простейшем случае используется последовательная организация работы ОА и УА (временная диаграмма – рисунок 5.10). Как видно из временной диаграммы, адрес МК не должен меняться в течение такта работы ОУ. Следовательно, он должен фиксироваться в буферном регистре адреса РА по фронту сигнала синхронизации С.

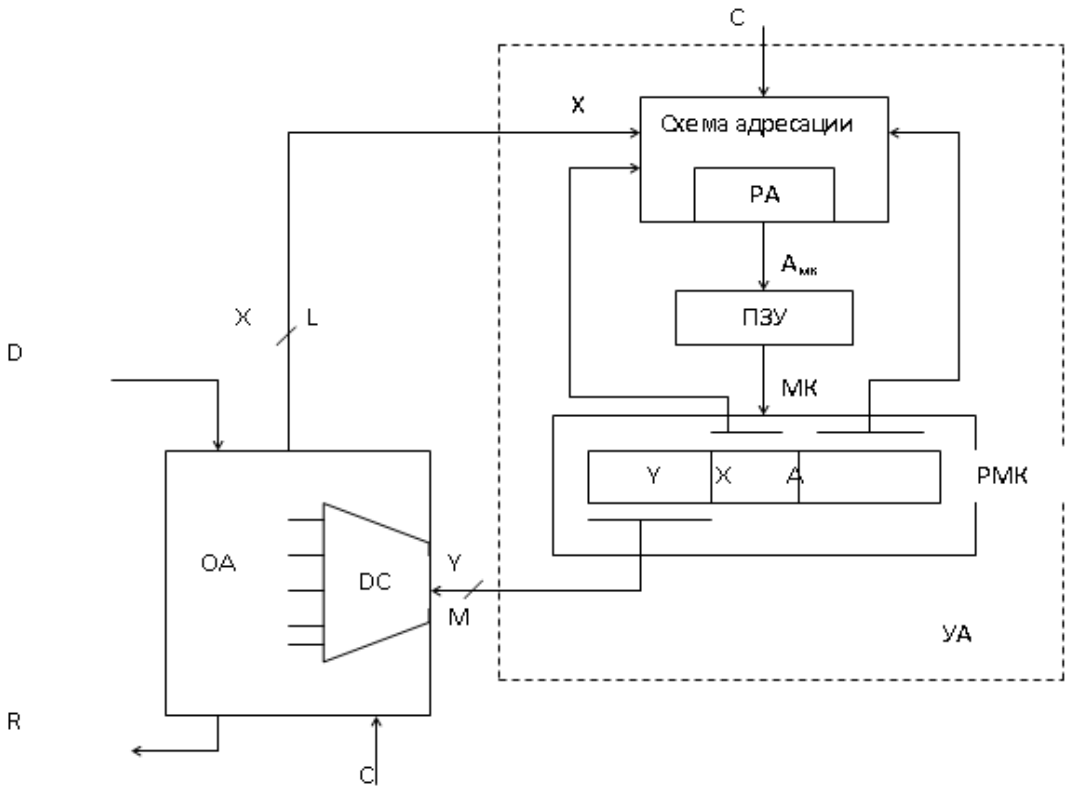


Рис.5.9.– Укрупненная структура УА

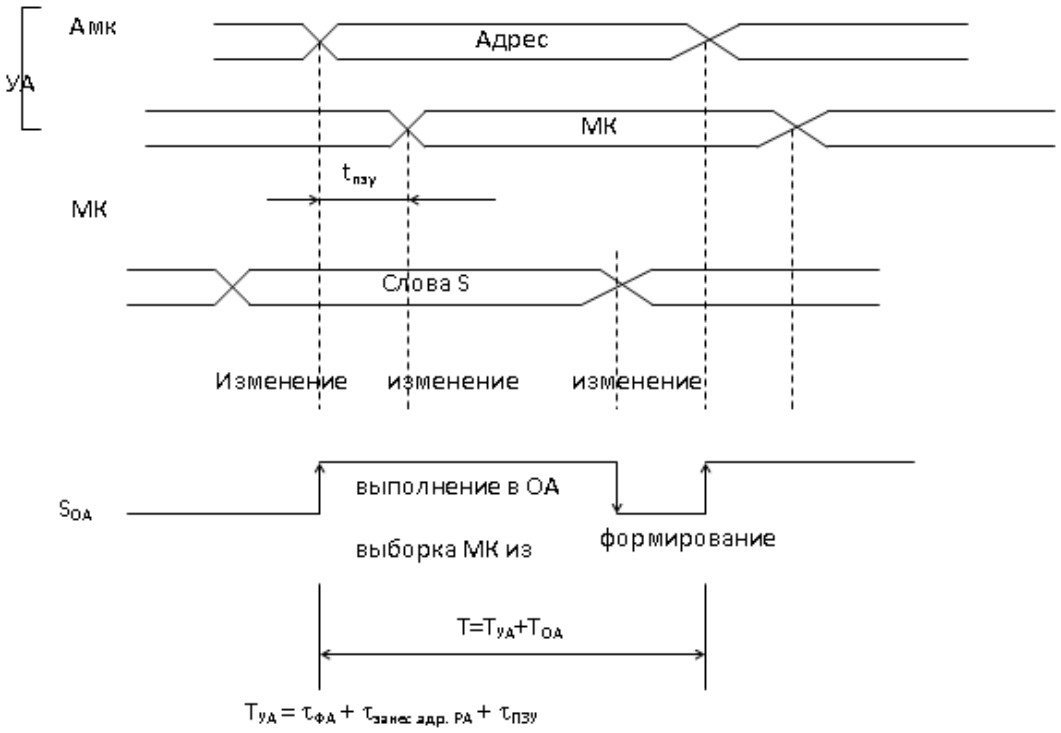


Рис.5.10.–Временная диаграмма

Буферный РА в этом случае входит в состав блока (схемы) адресации. Продолжительность такта при последовательной организации определяется:

$$T = T_{УА} + T_{ОА}. \qquad (5.4)$$

Для уменьшения Т можно использовать конвейерную организацию работы ОУ:

$$T_к = \max (T_{УА}, T_{ОА}). \qquad (5.5)$$

В этом случае работа ОА и УА совмещаются во времени (временная диаграмма – рисунок 5.10).

В течение такта $T_к$ микрокоманда не должна изменяться, поэтому в состав УА в этом случае необходимо ввести буферный (конвейерный) регистр МК (РМК), в который МК заносится по фронту сигнала синхронизации С. В этом случае буферный регистр РА не потребуется.

Занесение результатов МО в память S ОА в этом случае (при конвейерной организации) производится по спаду сигнала С (т. е. по $\overline{C}$).

Продолжительность работы УА определяется выражением:

$T_{УА} = \tau_{ФА} + \tau_{ПЗУ} + \tau_{РМК}$ (формирование адреса выборка из ПЗУ, занесение в РМК);

и ОА: $T_{ОА} = \tau_{МО} + \tau_{рез}$. (выполнение МО, занесение результата в S ОА).

Максимальное из них определяет продолжительность такта $T_к$ (5.5).

Следует отметить, что дешифрация поля Y МК обычно осуществляется не в блоке формирования сигналов y_m УА, а в ОА, т. е. в БИС МПЭ. Такая организация эффективна, т. к. количество входов у дешифратора m, а выходов $M = 2^m$ (y_1, … y_m) и, следовательно, количество проводников, связывающих ОА с УА уменьшается с 2^m до m (и количество выходов на корпусе БИС МПЭ). В результате в блоке формирования сигналов y_m остается один буферный РМК.

Недостатки конвейерной организации: потери времени на старт (рестарт) конвейера при выполнении условных и безусловных переходов в микропрограмме. В случае альтернативного перехода из ПЗУ должна извлекаться либо одна МК (по СМК), либо другая (по А перехода), т. е. в соответствии с выражением (5.3). Но на этапе выборки из ПЗУ извлекается только одна МК. Если не та МК необходимо начать сначала, с выборки нужной МК:

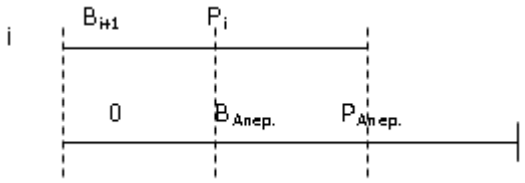


Рис.5.11– Выборка нужной МК

Здесь В – выборка МК (в РМК), Р – реализация МК (в ОА), 0 – пустой для ОА такт (рестарт конвейера).

Если разветвлений в микрокоманде не очень много, то потери времени небольшие, если много – то потери ощутимые, что снижает быстродействие.

Выход: выбирать в i-ом такте сразу две МК, а использовать только одну из них, в зависимости от условия перехода. Для этого ПЗУ в УА должно иметь два адресных входа и два информационных выхода (двухпортовые ПЗУ): на один адресный вход подается адрес из СМК, на другой – адрес перехода А (из РМК). Другой вариант – использовать два ПЗУ (с одинаковой информацией): одно используется для выбора одной МК, другое для выбора альтернативной МК (если нет двухпортового ПЗУ).

5.1.3. Управляющие автоматы с жесткой логикой управления

В основе построения УА с жесткой логикой (ЖЛ) управления лежит теория конечных (цифровых) автоматов. Наибольшую известность имеют конечные автоматы Мили и Мура.

Закон функционирования автомата Мили задается функциями:

$A(t+1) = \delta\,(A(t), X(t))$ – функция переходов,

$Y(t+1) = \lambda\,(A(t), X(t))$ – функция выходов,

Здесь $t = 0, 1, 2, \ldots$ -автоматное время,

$A = \{a_1, a_2, \ldots, a_N\}$ – множество состояний автомата, N – конечно,

$A(0) = a_1$ – начальное состояние автомата,

$X = \{x_1, \ldots, x_L\}$ – входной алфавит автомата,

$Y = \{y_1, \ldots, y_M\}$ – выходной алфавит автомата.

Чтобы построить УА с ЖЛ, необходимо задать функцию переходов δ и функцию выходов λ автомата. Определение функций λ, δ легко производится по ГСА операций f_g ОУ.

Техническая реализация функций автомата Мили приводит к структуре УА с ЖЛ (рисунок 5.12).

Выходной алфавит Y интерпретируется как совокупность выходов УА, с помощью которых управляющие сигналы $y_1 \ldots y_M$ подаются в ОА.

Входной автомат X интерпретируется как совокупность осведомительных сигналов x_1 … x_L, поступающих на вход комбинационной схемы КС в качестве аргументов булевских функций y_1, …, y_M, аргументами которых является еще и информация о состоянии автомата в момент времени t: A(t).

Для отображения состояния автомата используется память П, реализуемая на триггерах, количество которых $n = \log_2 N$ зависит от множества состояний. Переход из одного состояния a_i в другое a_j осуществляется под воздействием сигналов Z, которые формируются на выходах КС, реализующих булевы функции в соответствии с функцией переходов δ автомата.

Сигналы синхронизации С имитируют автоматное время t.

Следует отметить, что переход от функции ОА, которая задается набором $ГСА_1$ … $ГСА_G$ к структуре УА с ЖЛ, т. е. конкретной конфигурации комбинационных схем, реализующих функции Y и Z, формализован и, следовательно, может быть автоматизирован. Это позволяет получать схемы УА автоматически: на входе системы набор ГСА, а на выходе – документация, схема УА (фактически это САПР УА с ЖЛ).

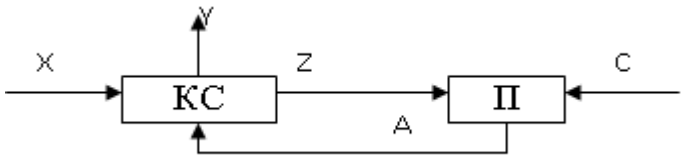


Рис.5.12.– УА с ЖЛ

Характеристики УА с ЖЛ – быстродействие и затраты оборудования. Быстродействие УА с ЖЛ характеризуется временем формирования управляющих сигналов y_m:

$t_y = \tau_{ТР} + 2\tau_{АЭ} = 5\tau_{АЭ}$,

где $\tau_{ТР}$ - время переключения триггеров памяти, $\tau_{ЛЭ}$ - задержка одного логического элемента в КС УА.

5.2. Сравнение характеристик управляющих автоматов с программируемой и жесткой логикой

$$\mathbf{T_Y^{ЖЛ}} = \boldsymbol{\tau_{тр}} + 2\boldsymbol{\tau} = 5\boldsymbol{\tau}$$

$$\mathbf{T_Y^{ПЛ}} = \boldsymbol{\tau_{ФА}} + \boldsymbol{\tau_{ПЗУ}} + \boldsymbol{\tau_{тр}} + \boldsymbol{\tau_{ДС}}$$

Сравнение выполним по быстродействию и затратам оборудования.

По быстродействию:

Разность $\Delta\tau = T_{ЖЛ} - T_{ПЛ} > 5\tau_{ПЗУ}$, т. е. время формирования управляющих сигналов в УА с ЖЛ меньше, по крайней мере, на величину $\tau_{ПЗУ}$ - время

выборки из ПЗУ, что существенно меньше. Следовательно, УА с ЖЛ обладают большим быстродействием, чем УА с ПЛ.

По затратам оборудования С. Количество оборудования С в УА с ЖЛ пропорционально сложности микропрограмм S (рисунок 5.13). В УА с ПЛ оно зависит от сложности микропрограмм, определяемой количеством микрокоманд. Длина микрокоманды логарифмически зависит от количества МО M, числа ЛУ L, емкости ПЗУ. На рисунке 5.13 S* - характерная точка, точка пересечения линий, которая означает, что затраты оборудования в УА с ПЛ до точки S* больше, а после нее меньше, причем разница в затратах тем больше, чем выше сложность УА.

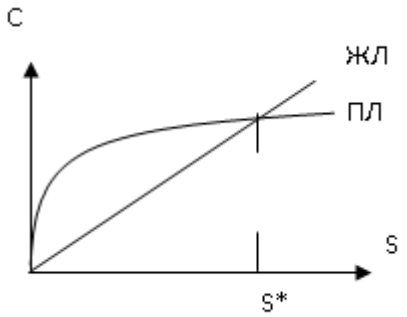


Рис.5.13.– Характеристика по затратам оборудования

Следовательно, с точки зрения затрат оборудования УА с ПЛ экономнее, чем УА с ЖЛ. Отсюда область их применения: если целью проектирования является высокое быстродействие, то следует использовать УА с ЖЛ, если критерий проектирования – минимальные затраты оборудования, то – УА с ПЛ.

Вопросы для самопроверки.

1. Назовите основные отличия управляющих автоматов с жесткой и программируемой логикой управления.
2. Что такое унитарное кодирование микроопераций?
3. Объясните смешанное кодирование микроопераций.
4. Приведите схему УА с косвенным кодированием.
5. Что такое принудительная адресация?
6. Приведите укрупненную схему УА с программируемой логикой управления?
7. Приведите законы функционирования автоматов Мили и Мура.

Модуль 6.

УСТРОЙСТВА ВВОДА-ВЫВОДА

Цель модуля: изучение студентами устройств ввода-вывода (периферийных устройств), организацию их интерфейсов; прямого доступа к памяти.

В результате изучения модуля студенты должны знать:

- организацию интерфейсов устройств ввода-вывода;
- основные принципы построения устройств ввода-вывода;
- порты ввода-вывода.

Содержание модуля:

6.1 Организация систем ввода-вывода.

6.1.1 Элементы организации интерфейсов.

6.1.2 Программно-управляемая передача данных и прямой доступ к памяти.

6.2 Основные принципы построения и структуры систем ввода-вывода.

6.3 Описание и структура многофункциональных линий порта ввода-вывода.

6.4 Принципы построения параллельного порта.

ЭВМ содержит помимо процессора (процессоров) и основной памяти, образующих ее ядро, многочисленные и разнообразные по выполняемым функциям и принципам действия внешние (периферийные) устройства (ПУ), предназначенные для хранения больших объемов информации (внешние запоминающие устройства) и для ввода в ЭВМ и для вывода из нее информации, в том числе для ее регистрации и отображения (устройства ввода-вывода).

Передача информации с внешнего устройства в ядро ЭВМ (память и процессор) называется операцией ввода, а передача из ядра ЭВМ в периферийное устройство – операцией вывода.

Производительность ЭВМ в значительной степени зависит от состава ВУ, их технических данных и способа организации их совместной работы с ядром.

Связь устройств ЭВМ друг с другом осуществляется с помощью интерфейсов, от характеристик которых во многом зависят производительность и надежность вычислительной машины.

6.1. Организация систем ввода/вывода

6.1.1. Элементы организации интерфейсов

Интерфейс представляет собой совокупность линий и шин, сигналов, электронных схем и алгоритмов (протоколов), предназначенную для осуществления обмена информацией между устройствами ЭВМ.

Физически интерфейс представляет собой совокупность линий и схем формирования сигналов. Все линии интерфейса разбиты на группы линий – шины.

Шины интерфейса в зависимости от их назначения можно разделить на:

- информационные, предназначенные для передачи команд, данных и адресов;
- идентификации типа информации, передаваемой по информационным шинам;
- управляющие, предназначенные для синхронизации, инициирования и завершения передачи информации.

В случае если для передачи адресов, данных и команд используются физически разные шины, необходимость в шинах идентификации отпадает.

Различные структуры шин интерфейса подразделяются на: индивидуальные, коллективные и комбинированные.

Наиболее надежной является структура с индивидуальными шинами, поскольку выход из строя группы шин не влияет на работу других устройств. При использовании индивидуальных шин упрощается адресация и идентификация, но увеличивается кол-во оборудования.

Структура с коллективными шинами имеет меньшую надежность. Но при необходимости организации связи с большим числом устройств такая структура позволяет уменьшить объем оборудования.

Интерфейсы характеризуются следующими параметрами:

- пропускной способностью интерфейса – количеством информации, которое может быть передано через интерфейс в единицу времени;
- максимальной частотой передачи информационных сигналов через интерфейс;
- максимально допустимым расстоянием между соединяемыми устройствами;

- динамическими параметрами интерфейса – временем передачи отдельного слова или блока данных с учетом продолжительности процедур подготовки и завершения передачи;
- общим числом проводов (линий) в интерфейсе;
- информационной шириной интерфейса – числом бит или байтов данных, передаваемых параллельно через интерфейс.

Количество информационных цепей интерфейса определяется типом элементов информации, которыми обмениваются устройства. Чаще всего по интерфейсу передаются байты, 16-ти или 32-х разрядные слова.

6.1.2. Программно-управляемая передача данных и прямой доступ к памяти

В системах ввода – вывода ЭВМ используется два основных способа организации передачи информации между памятью и ВУ: программно–управляемая передача и прямой доступ к памяти (ПДП).

Программно-управляемая передача данных осуществляется при непосредственном участии и под управлением процессора, который при этом выполняет специальную подпрограмму процедуры ввода-вывода. Данные между памятью и ВУ пересылаются через процессор.

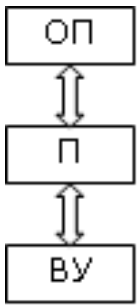


Рис.6.1.– Программно-управляемая передача данных

Операция ввода-вывода инициируется текущей командой программы или запросом прерывания от ВУ. При программно-управляемой передаче данных процессор на все время этой передачи отвлекается от выполнения основной программы решения задачи.

Вместе с тем при пересылке блока данных процессору приходится для каждой единицы передаваемых данных (байт, слово) выполнять довольно много команд, чтобы обеспечить: буферизацию данных, преобразование форматов, подсчет количества переданных данных, формирование адресов памяти и т.д. В результате скорость передачи данных при пересылке блока

данных даже через высокопроизводительный процессор может оказаться неприемлемой для работы ЭВМ в реальном времени.

Т.е. программно-управляемый обмен данными приемлем только для передачи небольших объемов информации и в основном используется в микро-ЭВМ.

Для быстрого ввода-вывода информации и разгрузки процессора от управления операциями ввода-вывода используют прямой доступ к памяти (ПДП).

Прямым доступом к памяти называется способ обмена данными, обеспечивающий автономно от процессора установление связи и передачу данных между ОП и ВУ.

Прямой доступ к памяти организуется по двум основным схемам(рисунок 6.2):

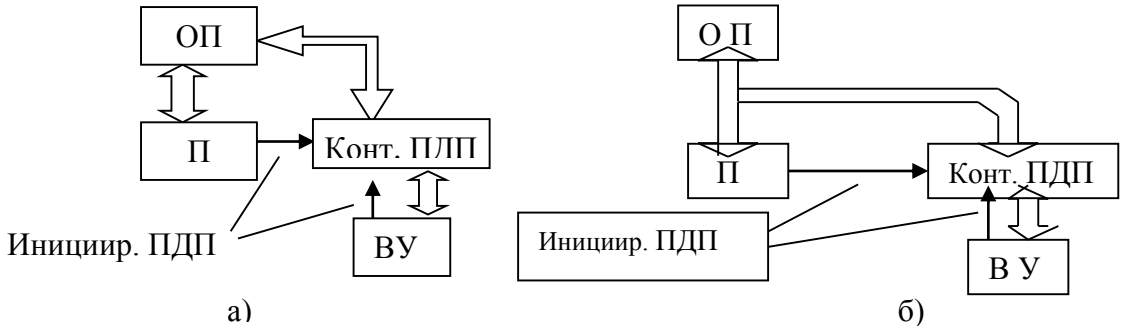


Рис.6.2.– Организация ПДП

а) прямой доступ к памяти при наличии отдельной шины в памяти для ПДП;

б) прямой доступ к памяти при использовании процессора и ПДП одной шины для связи с памятью.

ПДП освобождает П или МП от управления операциями ввода-вывода. П и ВУ работают независимо, поочередно обращаясь к памяти для чтения-записи слов информации, причем приоритет на доступ к памяти предоставляется ВУ. ПДП позволяет осуществлять параллельно во времени выполнение процессором текущей программы и обмен данными между ВУ и ОП. Т.о., ПДП, разгружая процессор от обслуживания операций ввода-вывода, способствует возрастанию общей производительности ЭВМ.

Прямым доступом к памяти управляет контроллер ПДП, который выполняет следующие функции:

• управление инициируемой процессором или ВУ передачей данных между ОП и ВУ;

• задание размера блока данных, который подлежит передаче, и области памяти, используемой при передаче;

• формирование адресов ячеек ОП, участвующих в передаче;

• подсчет числа единиц данных (байтов, слов), передаваемых от ВУ в ОП или обратно, и определение момента завершения заданной операции ввода-вывода.

Эти функции могут быть реализованы контроллером ПДП по следующей структурной схеме:

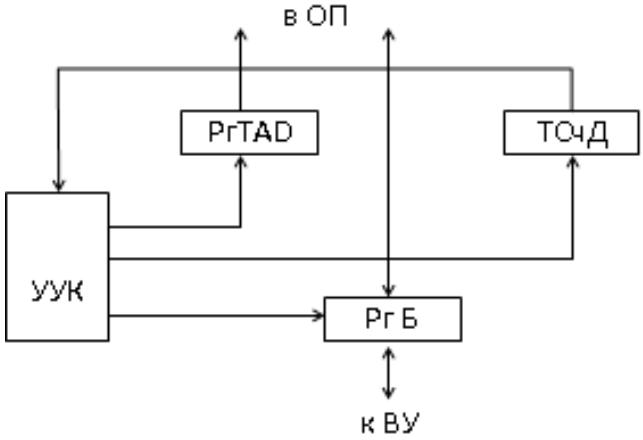


Рис.6.3.– Структурная схема контроллера ПДП

В состав схемы контроллера ПДП входят несколько буферных регистров (РгБ) (в зависимости от числа подключенных ВУ), регистр-счетчик текущего адреса данных (РгTAD), текущий счетчик данных (ТСчД) и устройство управления контроллера (УУК).

При инициировании операции ввода-вывода в ТСчД заносится размер подлежащего передаче блока (число байт или число слов), а в РгТАД - начальный адрес области памяти, используемой при передаче. После передачи каждого байта содержимое РгТАД увеличивается на 1, при этом формируется адрес очередной ячейки ОП, участвующей в передаче. Одновременно уменьшается на 1 содержимое ТСЧД. Обнуление ТСЧД указывает на завершение передачи.

Контроллер ПДП обычно имеет более высокий приоритет в занятии цикла памяти по сравнению с процессором. Поэтому управление памятью переходит к контроллеру ПДП, как только завершится цикл работы памяти, выполняемый для текущей команды процессора.

Прямой доступ к памяти обеспечивает высокую скорость обмена данными за счет того, что управление обменом производится не программным путем, а аппаратными средствами.

Как для программного обмена данными, так и для управления работой ВУ с ПДП используется один и тот же набор команд ввода-вывода. Такая унификация управления режимами передачи данных обеспечивается в основном за счет особой адресации ВУ. При этом каждому источнику и приемнику информации присваивается соответственный адрес. ВУ, работающее в режиме программного обмена данными, получает один адрес, обращаясь по которому можно записать или прочитать слово информации.

6.2. Основные принципы построения и структуры систем ввода-вывода.

В зависимости от способа реализации интерфейса различают два характерных принципа построения и соответствующие им структуры систем ввода-вывода: ЭВМ с одним общим интерфейсом и ЭВМ с множеством интерфейсов и процессорами (каналами) ввода - вывода.

Структура первого типа имеет следующий вид.

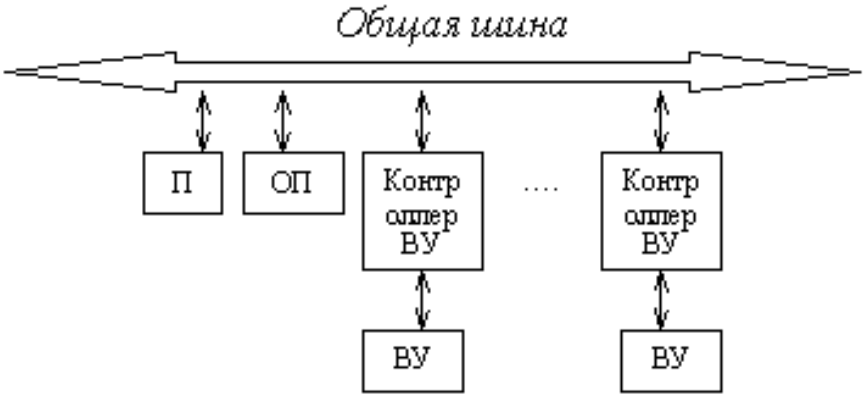


Рис.6.4.– ЭВМ с одним общим интерфейсом

Внешние устройства подключаются к единому интерфейсу через контроллер – устройство управления ВУ.

Контроллер ВУ относится к классу операционных устройств и выполняет следующие функции:

- согласуют форматы данных, используемые в ВУ, с форматом, принятым для передачи по единому интерфейсу;
- обеспечивают синхронизацию работы ВУ с другими устройствами;
- обеспечивают буферизацию информации.

Каждый тип ВУ требует применение специфического контроллера.

Применение единого интерфейса порождает следующие правила обмена информацией:

- информация передается словами, а информационная ширина интерфейса равна длине слова ОП;
- в каждый момент времени обеспечивается информацией только одна пара у-в;
- прямой обмен информацией между двумя ВУ невозможен, источником или приемником информации всегда является П или ОП.

Если необходимость обмена возникает одновременно в нескольких устройствах, то конфликт между ними разрешается с помощью специального устройства – контроллера шин, который часто включается в состав процессора.

Причина обмена информацией между ВУ только через П или ОП – различие быстродействия ВУ, которое приводит к необходимости буферизации. Для буферизации в контроллеры вводят внутреннюю память небольшого объема.

Единый интерфейс – высокоэффективный способ организации обмена информацией в ЭВМ, комплектуемых небольшим количеством ВУ. В едином интерфейсе обеспечивается большое многообразие режимов обмена.

При общем интерфейсе аппаратура управления вводом-выводом рассредоточена по отдельным модулям, и ее объем существенно зависит от числа ВУ в составе ЭВМ.

Поэтому по схеме с единым интерфейсом строятся мини- и микро- ЭВМ с небольшим числом ВУ. При этом структуры мини ЭВМ строятся по схеме с общей шиной, с микро ЭВМ с мультишиной (модификацией единого интерфейса). Мультишина обладает большими логическими возможностями.

Для построения высокопроизводительных ЭВМ общего назначения, работащих с многобайтными словами, с большим набором ВУ, используется более сложная иерархическая структура системы ввода-вывода с процессорами (каналами) ввода - вывода.

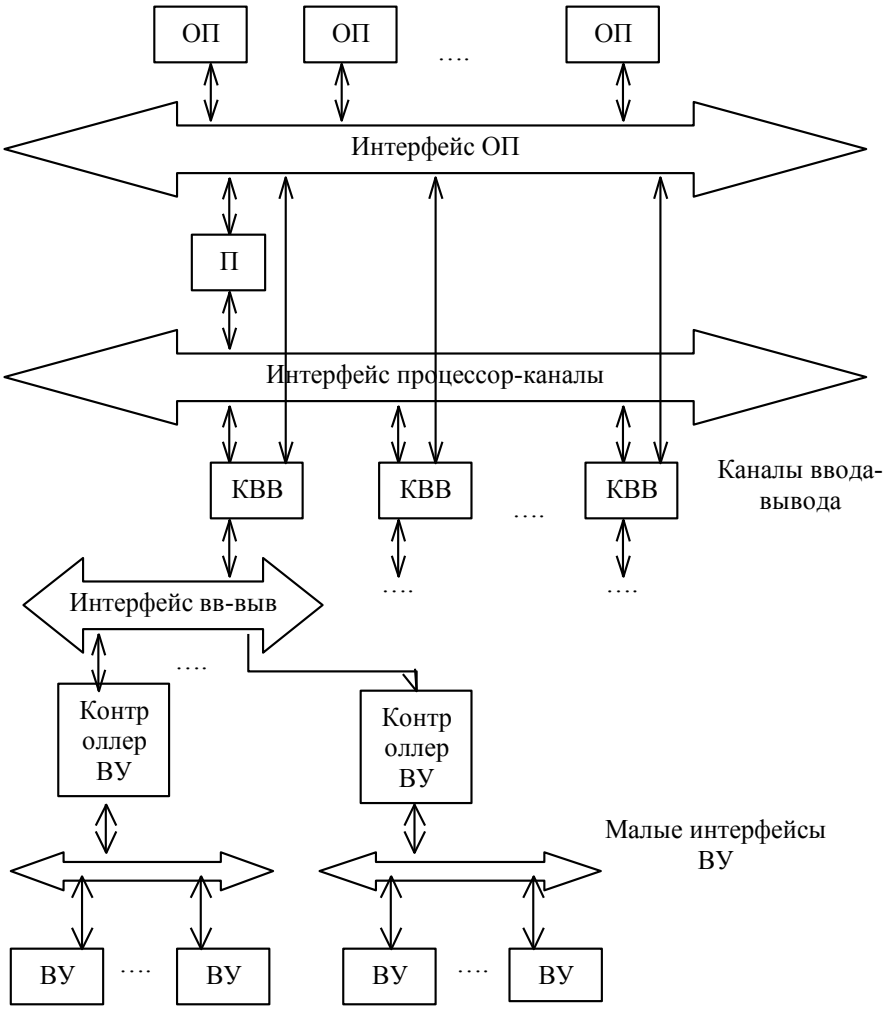


Рис. 6.5.– Структура ЭВМ на основе канала ввода/вывода

Эта структура часто называется структура ЭВМ на основе канала ввода-вывода. Здесь отсутствует однородность в структуре потоков и форматах представления данных, что приводит к необходимости иметь в ЭВМ несколько специализированных интерфейсов.

В данной структуре используются интерфейсы четырех типов:

- оперативной памяти. Через интерфейс основной памяти производится обмен информацией между ОП, с одной стороны, и процессором (процессорами) и каналами ввода-вывода – с другой;
- процессор - каналы. Интерфейс "процессор - каналы" предназначается для передачи управляющей информации между процессорами и каналами ввода-вывода;
- ввода-вывода. Через интерфейс ввода-вывода производится обмен информацией между каналами и контроллерами ВУ;
- малые интерфейсы внешних устройств. Через малые интерфейсы осуществляется передача информации между контроллерами ВУ и ВУ.

Наиболее быстродействующими являются интерфейс ОП и интерфейс “процессор-канал”.

При проектировании ЭВМ интерфейсы стремятся унифицировать, в первую очередь интерфейсы, обеспечивающие сопряжение с периферийными устройствами (интерфейсы ввода-вывода). Интерфейсы периферийных устройств не могут быть унифицированы, т.к. сами эти устройства весьма разнообразны по принципу действия, по выполняемым операциям и по используемым форматам данных и сигналам.

Каналы ввода-вывода разгружают процессор от операций ввода-вывода. Они осуществляют прямой доступ к памяти.

Функции контроллеров ВУ в основном остаются такие же, что и в структуре с общим интерфейсом, но общее для всех контроллеров оборудование вынесено в канал, а в контроллере оставлены только схемы специфичные для конкретного типа ВУ.

При большом числе ВУ использование КВВ экономит оборудование за счет централизации в канале весьма сложных функций по обслуживанию ВУ.

6.3. Описание и структура многофункциональных линий порта ввода-вывода

6.3.1. GP0/AN0/CIN+

Каждый вывод GPIO может выполнять несколько различных функций. Структурная схема порта изображена на рис. 6.6.

Вывод GP0 конфигурируется для одной из следующих функций:

- порт ввода/вывода общего назначения;
- аналоговый вход для АЦП;
- аналоговый вход компаратора.

6.3.2. GP1/AN1/CIN-/VREF

Структурная схема этого порта изображена на рис. 6.6. Вывод GP1 конфигурируется для одной из следующих функций:

- порт ввода/вывода общего назначения;
- аналоговый вход для АЦП;
- аналоговый вход компаратора;
- вход опорного напряжения для АЦП.

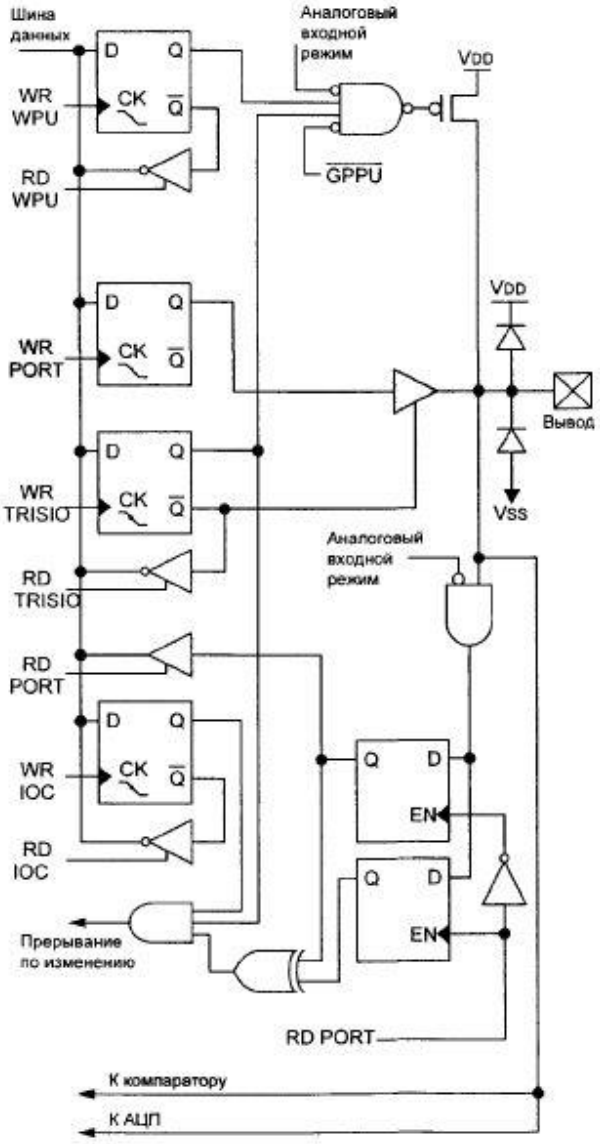


Рис.6.6.– Структурная схема порта ввода/вывода

6.3.3. GP2/AN2/T0CKI/INT/COUT

Структурная схема этого порта изображена на рис. 6.7. Вывод GP2 конфигурируется для одной из следующих функций:

- порт ввода/вывода общего назначения;
- аналоговый вход для АЦП;
- вход импульсов для TMR0;
- внешнее прерывание по фронту импульса;
- цифровой выход компаратора.

6.3.4. GP3/MCJLR/VPp

Структурная схема этого порта изображена на рис. 6.8. Вывод GP3 конфигурируется для одной из следующих функций:

- порт ввода/вывода общего назначения;
- вывод сигнала общего сброса.

6.3.5. GP4/AN3/HG/OSC2/CLKOUT

Структурная схема этого порта изображена на рис. 6.9. Вывод GP4 конфигурируется для одной из следующих функций:

- порт ввода/вывода общего назначения;
- аналоговый вход для АЦП;
- вход TMR1 Gate;
- подключение кварца или керамического резонатора;
- выход тактовой частоты.

6.3.6. GP5fT/CKI/OSC1/CLKIN

Структурная схема этого порта изображена на рис. 2.13. Вывод GP5 конфигурируется для одной из следующих функций:

- порт ввода/вывода общего назначения;
- вход импульсов для TMR1;
- подключение кварца или керамического резонатора;
- вход внешней тактовой частоты.

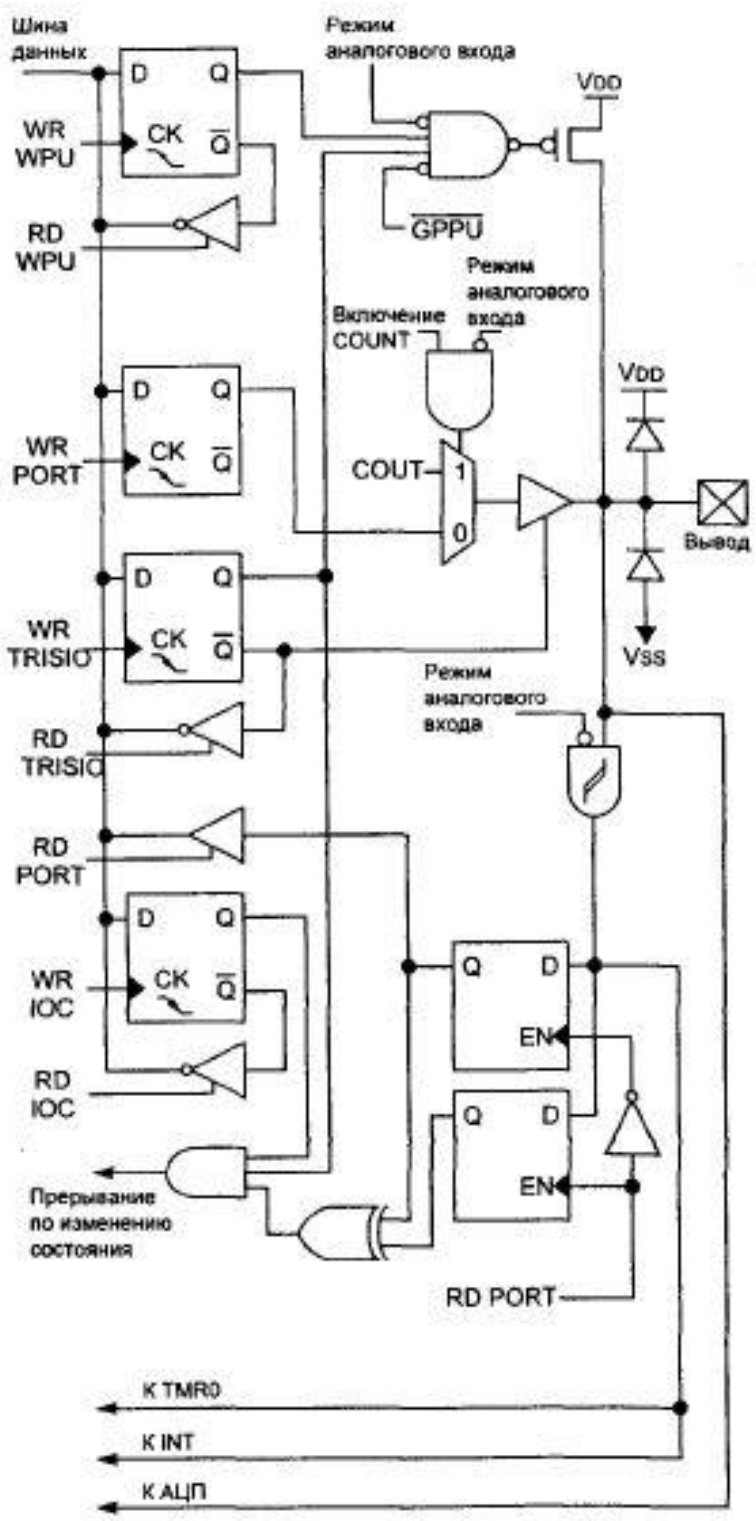


Рис.6.7.– Структурная схема порта ввода/вывода GP2

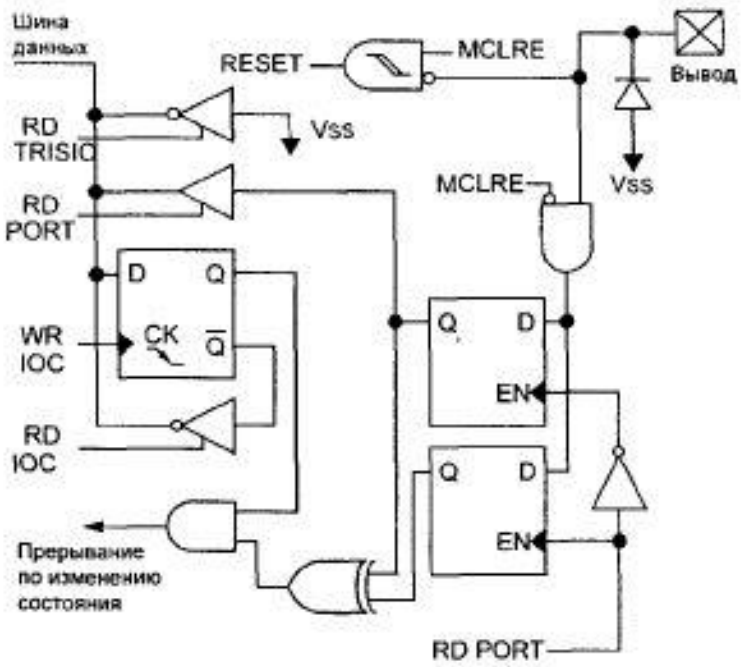
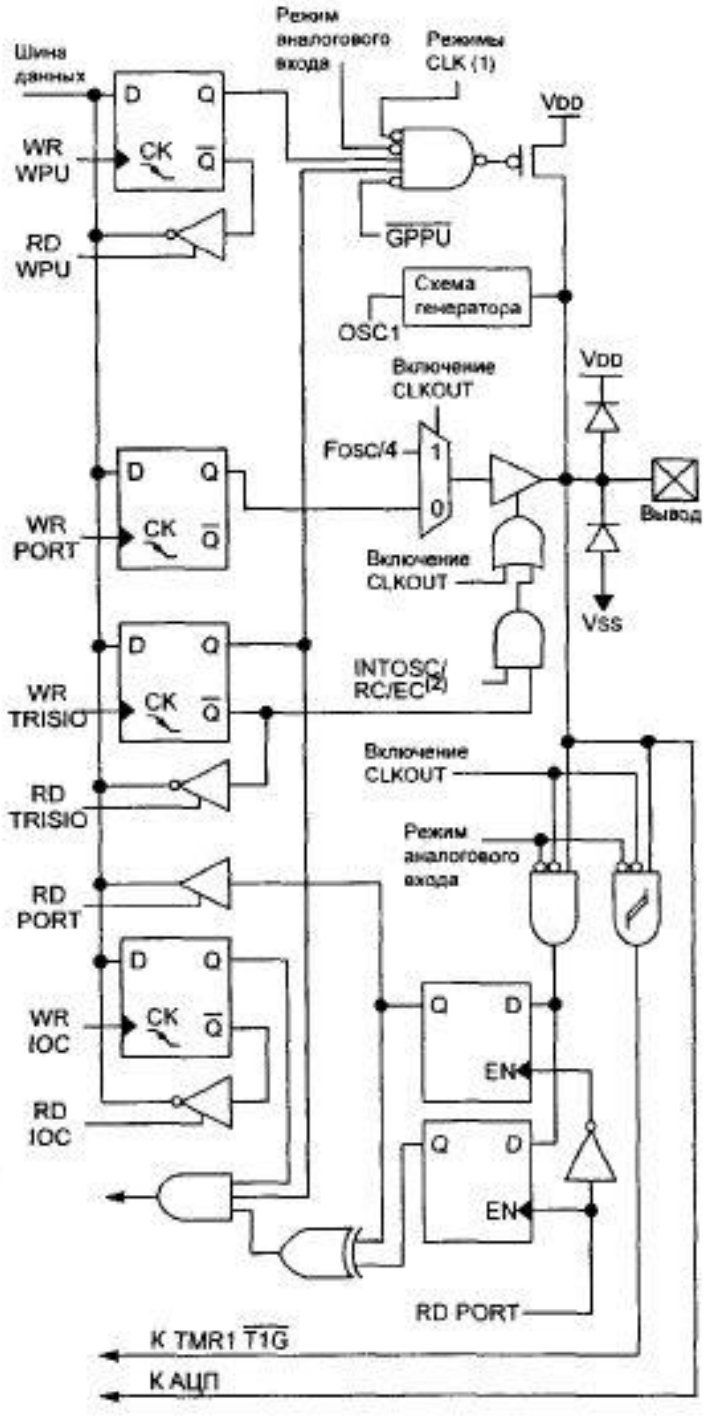


Рис.6.8.– Структурная схема порта ввода/вывода GP3

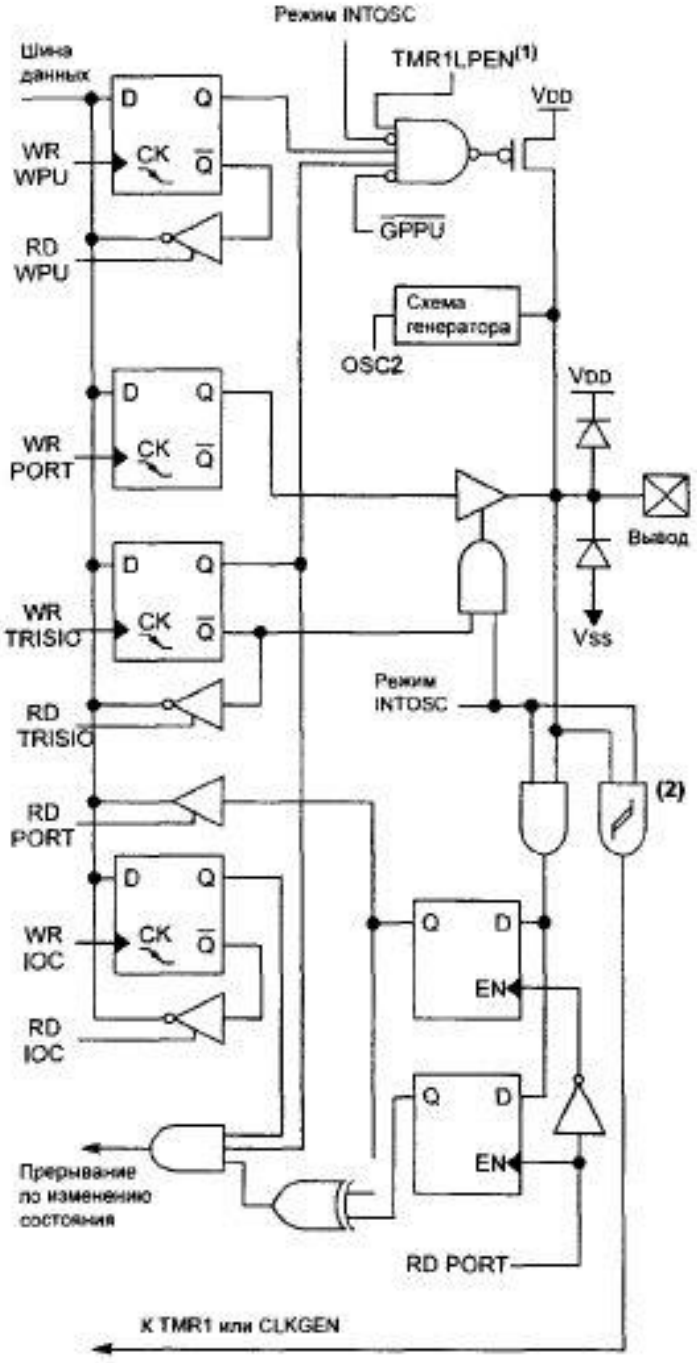
Таблица 6.1. Перечень регистров, ассоциированных с портами ввода-вывода

Адрес	Название	bit7	bit6	bit5	bit4
05h	GPIO	-	-	GP5	GP4
0Bh/8Bh	INTCON	GIE	PEIE	TOIE	INTE
19h	CMCON	-	COUT	-	CINV
81h	OPTION_REG	GPPU	INTEDG	TOCS	TOSE
85h	TRISIO	-		TRISI05	TRISI04
95h	WPU	-	-	WPU5	WPU4
96h	IOC	-	-	IOC5	IOC4
9Fh	ANSEL	-	ADCS2	ADCSI	ADCSO



(1) Режимы CLK: XT, HS, LP, LPTMR1 и "CLKOUT включен"
(2) Совместно с опцией CLKOUT

Рис.6.9.– Структурная схема порта ввода-вывода GP4



(1) Генератор типа LP для Timer1 включен.
(2) Если используется Timer1 с генератором типа LP, то триггер Шмитта обходится.

Рис.6.10.– Структурная схема порта ввода-вывода GP5

Продолжение таблицы 6.1.

bit3	bit2	bitl	bitO	Значение при сбросе по питанию	Значение при других вариантах сброса
GP3	GP2	GP1	GPO	--XXXXXX	-UU UUUU
GPIE	TOIF	INTF	GPIF	0000 0000	0000 0000
CIS	CM2	CM1	CMO	-0-0 0000	-0-0 0000
PSA	PS2	PS1	PSO	1111 1111	1111 1111
TRIS103	TRISI02	TRISI01	TRISIOO	--11 1111	-111111
-	WPU2	WPU1	WPUO	--11 1111	-111111
IOC3	IOC2	IOC1	IOC0	-00 0000	-00 0000
ANS3	ANS2	ANSI	ANSO	-000 1111	-000 1111

Обозначения: x = неопределенное, и = не изменяется, - = не применяется, читается как «О». Выделенные заливкой биты не влияют на работу портов.

6.4. Принципы построения параллельного порта.

Параллельные порты предназначены для обмена информацией микропроцессора с внешними устройствами, при этом в качестве внешнего устройства может использоваться другой компьютер. Параллельные порты позволяют согласовывать низкую скорость работы внешнего устройства и высокую скорость работы системной шины микропроцессора. С точки зрения внешнего устройства порт представляет собой обычный источник или приемник информации со стандартными цифровыми логическими уровнями (обычно ТТЛ), а с точки зрения микропроцессора - это ячейка памяти, в которую можно записывать данные или в которой сама собой появляется информация.

В качестве внешнего устройства может служить любой объект управления или источник информации (различные кнопки, датчики, микросхемы приемников, синтезаторов частот, дополнительной памяти, исполнительные механизмы, двигатели, реле и т.д.).

В зависимости от направления передачи данных параллельные порты называются портами ввода, вывода или портами ввода вывода.

Структурная схема порта ввода приведена на рис 6.11.

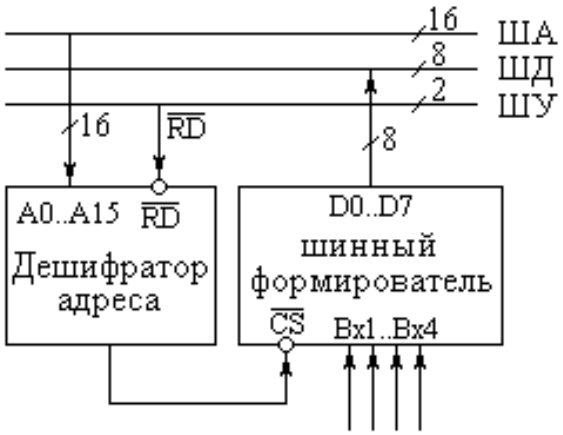


Рис.6.11.– Структурная схема порта ввода

В качестве порта ввода может быть использована схема с открытым коллектором или с третьим (Z) состоянием. В настоящее время обычно используются схемы с третьим состоянием. Параллельное соединение таких схем называется шинным формирователем. Из порта ввода возможно только чтение информации.

Выход шинного формирователя подключается к внутренней шине. Значение сигнала с внешнего вывода порта считывается по сигналу "RD".

Для отображения этого шинного формирователя только в одну ячейку памяти адресного пространства микропроцессорного устройства в составе порта ввода-вывода всегда присутствует дешифратор адреса.

Так как с точки зрения программиста эта ячейка памяти ничем не отличается от регистра данных порта вывода, то по аналогии она называется регистром данных порта ввода.

Структурная схема порта вывода приведена на рис 6.12.

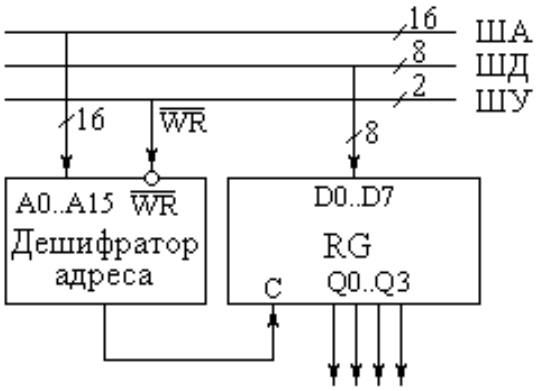


Рис.6.12.– Структурная схема порта вывода

В качестве порта ввода может быть использована схема с открытым коллектором или с третьим (Z) состоянием. В настоящее время обычно используются схемы с третьим состоянием. Параллельное соединение таких схем называется шинным формирователем. Из порта ввода возможно только чтение информации.

Выход шинного формирователя подключается к внутренней шине. Значение сигнала с внешнего вывода порта считывается по сигналу "RD".

Для отображения этого шинного формирователя только в одну ячейку памяти адресного пространства микропроцессорного устройства в составе порта ввода-вывода всегда присутствует дешифратор адреса.

Так как из порта ввода возможно только чтение, а в порт вывода возможна только запись, то для них обычно отводится один и тот же адрес в адресном пространстве памяти микропроцессора.

Порты выпускаются в качестве универсальных микросхем, но на заводе, где производятся эти микросхемы неизвестно, сколько на самом деле потребуется линий ввода информации, и сколько потребуется линий вывода информации. Количество же ножек у микросхемы ограничено. Поэтому в одной универсальной микросхеме размещаются и порт ввода, и порт вывода информации, а для подключения этих портов к внешним ножкам микросхемы используется коммутатор. Для управления этим коммутатором используется еще один (внутренний) параллельный порт вывода, регистр данных которого называется регистром управления параллельного порта ввода-вывода, а сам порт называется портом ввода-вывода. Адрес для регистра управления обычно назначается рядом с адресом регистра данных порта ввода-вывода.

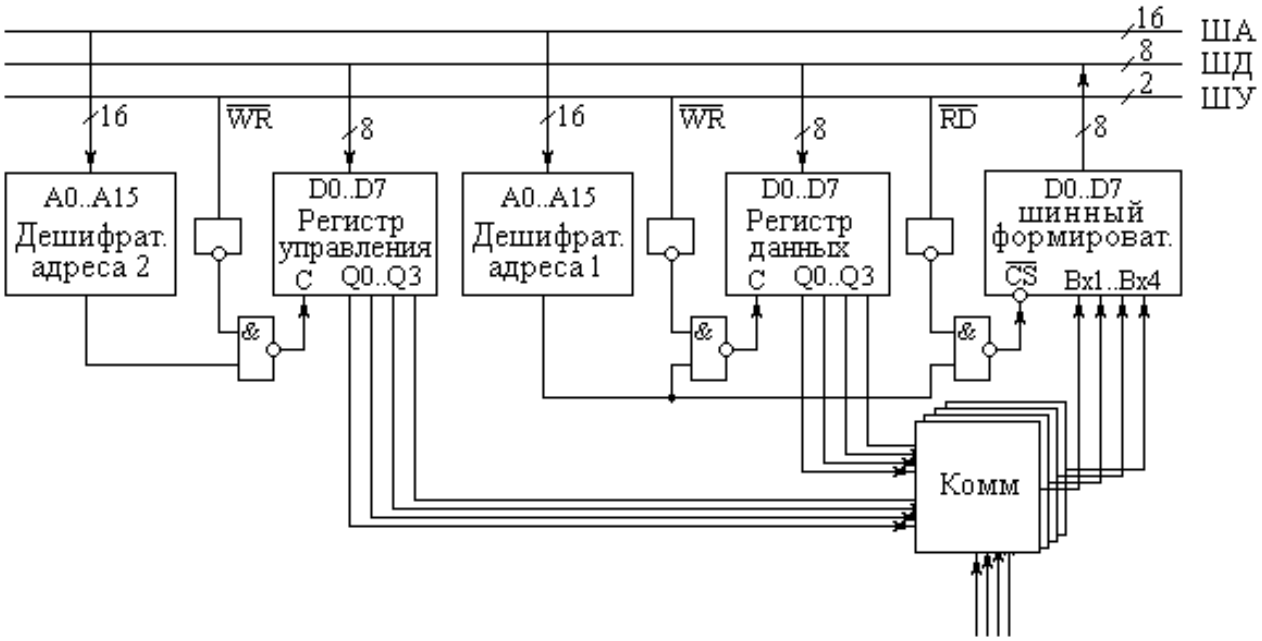


Рис. 6.13.– Структурная схема параллельного порта ввода-вывода.

В некоторых микропроцессорах для портов ввода вывода выделяется отдельное адресное пространство. В этом случае для записи в порт и для чтения из порта используются отдельные сигналы чтения и записи. Чаще всего они называются IOWR# и IORD#.

Параллельные порты, предназначенные для обмена данными между компьютерами, или компьютером и принтером, устроены несколько иначе. Основным отличием обмена данных между компьютерами или контроллерами от обмена данными между компьютером и простым внешним устройством является большой объем передаваемых данных. В этом случае недостаточно выдачи на выход порта одного или даже нескольких байт информации, поэтому приходится передавать данные последовательно байт за байтом через один и тот же параллельный порт. Байты необходимо каким-либо образом отличать друг от друга, поэтому вводится специальный сигнал синхронизации CLK, который позволяет отличать один байт от другого. Для формирования такого сигнала можно воспользоваться вторым параллельным портом, и получить его программным способом, но обычно этот сигнал формируется аппаратно из сигнала WR# при записи очередного байта в параллельный порт вывода. Временная диаграмма обмена данными через параллельный порт приведена на рисунке 6.14.

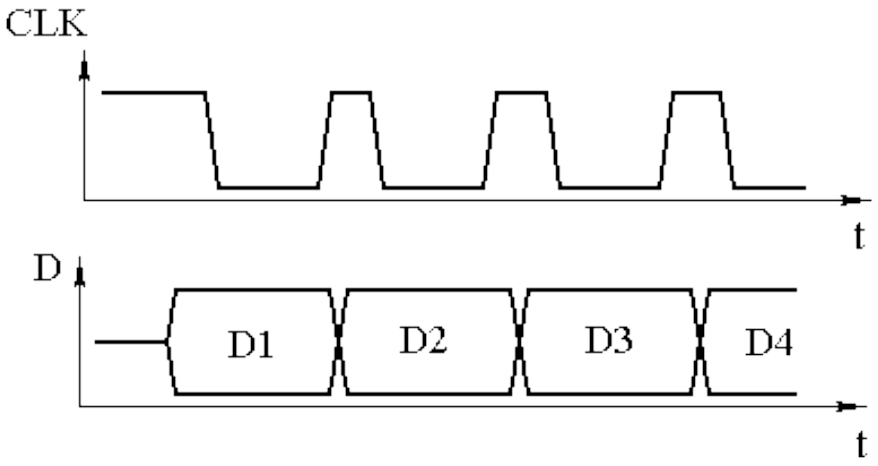


Рис. 6.14.– Временная диаграмма работы параллельного порта.

Вопросы для самопроверки.

1. Как можно разделить шины интерфейсов?
2. Какими параметрами можно охарактеризовать интерфейс?
3. Охарактеризуйте программно-управляемую передачу данных.
4. Что такое прямой доступ к памяти?
5. Приведите структурную схему контроллера ПДП.
6. Приведите два характерных принципа построения систем ввода-вывода.
7. Какие функции выполняет контроллер внешнего устройства?
8. Приведите структуру параллельного порта.
9. Назовите основные отличия ПУП от ПДП.

Модуль 7.

ЭЛЕМЕНТЫ АРХИТЕКТУРЫ КОМПЬЮТЕРНЫХ СИСТЕМ

Цель модуля: изучение студентами различных элементов архитектуры компьютерных систем и принципов организации их работы.

В результате изучения модуля студенты должны:

- знать форматы команд и машинные операции;
- знать способы адресации информации в памяти ЭВМ;

Содержание модуля:

7.1 Форматы команд и машинные операции.

7.2 Способы адресации информации в памяти электронных вычислительных машин.

7.2.1 Прямая адресация.

7.2.2 Страничная адресация.

7.2.3 Базовая адресация.

7.2.4 Косвенная адресация.

7.2.5 Индексная адресация.

7.2.6 Индексно-относительная адресация.

7.2.7 Непосредственная адресация.

7.2.8 Неявная адресация.

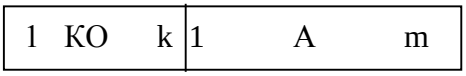
Множество (набор) команд, которые используются в ЭВМ, принято называть **системой машинных команд**: $К=\{К_1, К_2, \ldots, К_N\}$. Система команд К является основой языка ассемблер.

Особенности системы машинных команд:

- устойчивость (стабильность) во времени. Нужна, чтобы обеспечить преемственность ПО. Изменение системы команд неизбежно ведет к необходимости разработки нового ПО, а это процесс трудоемкий;
- особенность – ЭВМ различных типов имеют различные системы команд, имеют различные машинные языки и, следовательно, ассемблеры;
- особенность – количество машинных команд в наборе может быть небольшим (RISC-машины, с сокращенным набором машинных команд – в пределах сотни) и большим (CISC-машины, могут содержать несколько сотен команд).

7.1. Форматы команд и машинные операции

Под форматом команды понимается управляющее слово, разделенное на



поля фиксированного назначения и фиксированной длины (разрядности).

Команда, состоящая из двух полей, имеет вид:

Здесь: k=const, m=const, например, k=8, m=16. Поле КО в виде двоичного кода задает тип операции (например, сложение) и тип данных, например, сложение целых чисел (КО – сложение целых чисел), или сложение с плавающей запятой (КО – сложение с плавающей запятой), или сложение двоично-десятичных чисел (КО – сложение чисел BCD). Все они кодируются тремя разными кодами. В адресной части команды – в полях А - указываются адреса операндов.

Количество адресных полей, и их длина зависят от различных факторов. Например, для двуместных операций типа сложение (вычитание, умножение, и т.п.) адресная часть команды может содержать три адресных поля:

Схема выполнения трехадресной команды: [A3]:=[A1]*[A2]. Содержимое ячейки с адресом A1 - [A1] (первый операнд) и содержимое

КО	A1	A2	A3

ячейки с адресом A2 - [A2] (второй операнд), извлеченные из памяти, вступают в операцию *, заданную полем КО, и результат операции засылается (записывается) в ячейку памяти с адресом A3.

В общем случае в зависимости от количества адресных полей принято различать команды: 0-адресные (безадресные), одноадресные, двухадресные, трехадресные.

Кроме того, в ВТ применяют команды различных форматов в зависимости от уровней памяти, т.е. в зависимости от того, адрес какого типа указан в адресной части команды - адрес типа R, в котором указывается номер РОНа, или адрес типа А – номер ячейки ОП. Отсюда различные типы форматов: RR – команды типа регистр-регистр, RM – команды типа регистр-память, MM - команды типа память-память.

Другие особенности форматов команд проявятся потом, когда будем рассматривать различные способы адресации информации в памяти ЭВМ.

С системой команд неразрывно связано другое понятие – **система машинных операций** $F=\{f_1, f_2, \dots, f_G\}$.

Типичная система машинных операций содержит арифметические операции типа сложение, вычитание, умножение, деление и др., логические операции конъюнкция, дизъюнкция и др., операции сдвига кодов, чисел и т.п.

Машинная операция – это действие, которое инициируется командой и реализуется (интерпретируется) аппаратурой ЭВМ. В ЭВМ на аппаратном уровне могут выполняться операции над различными типами данных: над числами (целыми, с плавающей запятой, двоично-десятичными), над символами, над логическими значениями (логические операции). Отсюда многообразие операций F. В общем случае все машинные операции F

принято разделять на 5 классов (типов): арифметико-логические (АЛО); посылочные операции – обеспечивают пересылку данных; операции управления; операции ввода/вывода; системные (привилегированные) операции.

АЛО выполняются в специальном устройстве - АЛУ. Посылочные операции, операции управления – условные (УП) и безусловные переходы (БП) - реализуются центральным устройством управления (ЦУУ) центрального процессора. Операции ввода/вывода реализуются различного рода ПУ. Системные операции предназначены для управления вычислительным процессом и распределением ресурсов ЭВМ и могут использоваться только программами ОС. Привилегированные операции пользователь использовать не может.

7.2. Способы адресации информации в памяти электронных вычислительных машин

Адресация информации – это способ использования адресной части команды для адресации информации в памяти ЭВМ. Память ЭВМ имеет трехуровневую иерархическую структуру, поэтому, сначала рассмотрим адресацию информации в памяти различных уровней, т.е. как организуется адресное пространство ЭВМ, а затем - как используется адресная часть команды для указания местоположения информации в этом адресном пространстве. Типичная схема адресного пространства представлена на рисунке 7.1.

Следует отметить, что в адресном пространстве процессора информация адресуется обычно с точностью до байта. В адресном пространстве ввода-вывода адресуется не информация, а периферийные устройства ПУ1, ПУ2, … ПУN, обращение к которым осуществляется через регистры, входящие в состав контроллеров ПУ. Каждому ПУ выделяется не мене трех регистров различного назначения – регистры данных, состояния и управления. Эти регистры нумеруются и образуют адресное пространство ввода-вывода. Информация же располагается на носителях информации (съемных или нет) ПУ. На них информация организуется в виде массивов, которым присваиваются различные имена, - так называемых файлов. Имя файла используется при обращении к массиву на носителе информации. Местоположение файлов на носителе информации хранится в каталоге. Каталог – это таблица, в которой указывается соответствие имен файлов и их физических адресов. Сам каталог располагается также на носителе информации, т. е. каталог – это тоже файл, только специфический, адресный.

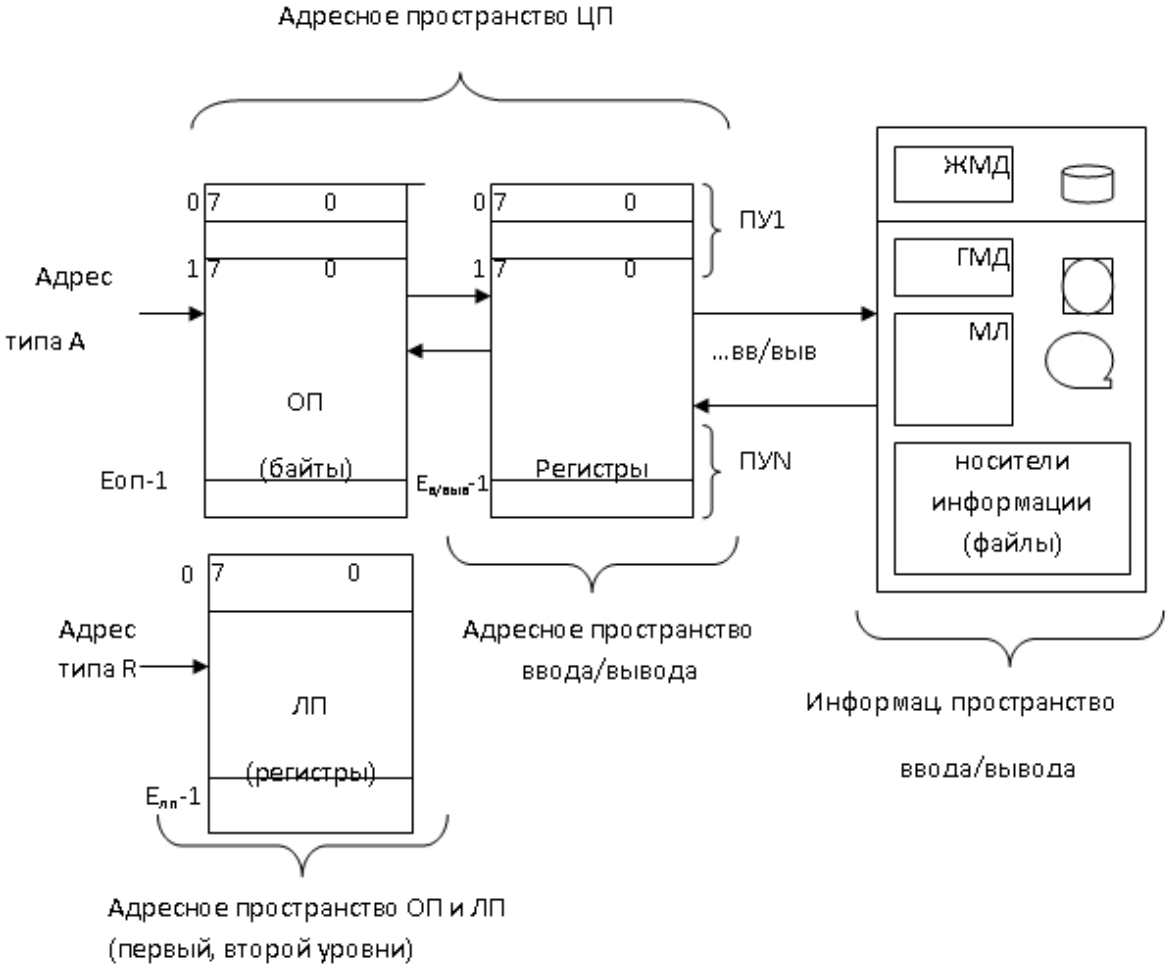


Рис.7.1.– Схема адресного пространства

Таким образом, обращение к информации осуществляется по имени файла на логическом уровне и по номеру ПУ и номеру блока информации на носителе информации:

№ ПУ	№ блока

Элементы информации, из которых состоит файл, становятся доступны ЦП только после ввода файла в ОП. Отсюда понятие ввода-вывода информации. Процесс ввода-вывода осуществляется обычно через РД периферийного устройства, побайтно.

Следует отметить, что ввод-вывод файлов, т.е. обмен информацией между ОП и ПУ, осуществляется под управлением алгоритма ввода-вывода. Это достаточно сложный алгоритм. В ВТ он обычно реализуется либо программными средствами (чаще всего), либо аппаратными.

В этом параграфе мы рассмотрим, как используется адресная часть команд для адресации информации в адресном пространстве процессора, т.е. как адресуются регистры ЛП (РОНы), как адресуются ячейки ОП, как адресуются ПУ, т.е. регистры ПУ.

Регистры памяти первого уровня адресуются напрямую (прямая регистровая адресация), путём указания номера регистра из набора: 0,1, ... ,N-1. Номер регистра указывается в адресной части команды и называется адресом типа R. При программировании на ассемблере используются буквенные имена (обозначения) для РОНов. При трансляции в машинные коды имя регистра преобразуется в номер регистра, который указывается в поле R команды (длина поля k зависит от количества РОНов, k=3,4,5, ...).

Данные, которые **расположены в ячейках ОП**, можно адресовать различными способами.

7.2.1. Прямая адресация

Прямая адресация – в адресной части команды указывают ячейки ОП (адрес типа А). Если это байт – то номер байта, если это слово – то номер слова.

Основные недостатки прямой адресации:

- длина адреса A m – переменная величина, т. к. зависит от емкости ОП: $m=\log_2 E_{ОП}$ ($E_{ОП}=2^m$),
- m принимает большие значения, т.к. емкость ОП современных ЭВМ измеряется мегабайтами.

Поэтому, прямая адресация обычно используется только для фиксированной части ОП (например, n=16; 2^{16}=64KB), т.е. с помощью прямого адреса фиксированной длины n можно адресовать часть ОП - т.н. страницу ОП емкостью $2^n=E_{СТР}$, n=const, n<m, $m=\log_2 E_{ОП}$.

Чтобы узнать, в какой странице (сегменте) находятся адресуемые прямым адресом данные, процессору нужно указать номер страницы.

7.2.2. Страничная адресация.

В результате получаем второй способ адресации – классическую **страничную адресацию.** Суть (рисунок 7.2): ОП разделяется на страницы фиксированной емкости $E_{СТР}=2^n$.

Адрес А ячейки памяти формируется из двух полей: A=P.D, P – номер страницы (старшие разряды адреса), D – номер ячейки в странице (младшие разряды адреса).

Номер страницы P заносится и хранится в специальном регистре номера страницы РС ЦП, адрес ячейки D указывается в адресном поле команды. Формирование физического (исполнительного) адреса А, по которому

производится обращение к ОП, осуществляется по схеме, представленной на рисунке 7.2.

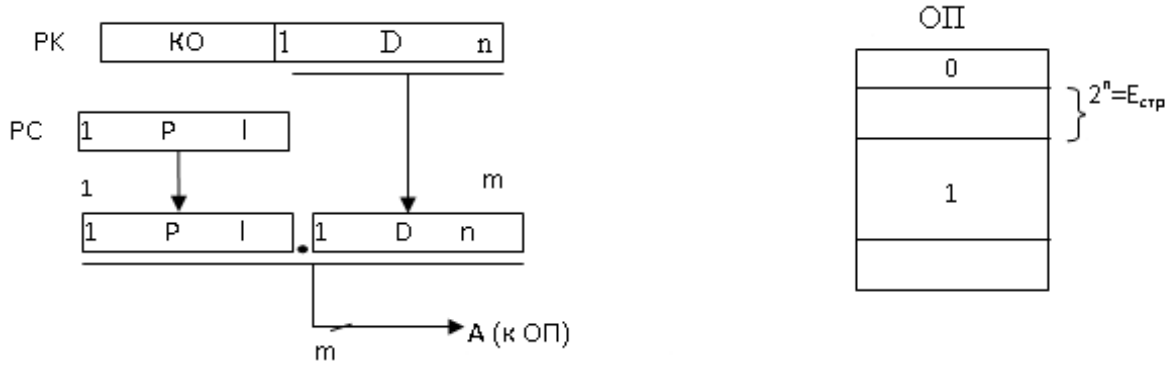


Рис.7.2.– Схема формирования адреса при страничной адресации

Недостатки:

- сложности с организацией переходов в программе за пределы страницы. При таком переходе необходимо модифицировать содержимое регистра номера страницы в ЦП – P:=P+ΔP (P-ΔP), т. е. требуется выполнение дополнительных операций с регистром P, что снижает быстродействие процессора;
- увеличивается длина программы – за счет модификации регистра P, что также ведет к увеличению времени выполнения программы и снижению производительности процессора. Следует отметить, что регистр P обычно загружается и модифицируется ОС.

7.2.3. Базовая адресация

В ЭВМ типа IBM PC страничная адресация в классическом (чистом) виде не используется. В PC используется более общая адресация, т.н. **относительная (базовая) адресация.**

Суть. Адрес A ячейки памяти формируется путем суммирования двух полей:

$$A=B_M+D, \qquad (2.8)$$

где B_M - базовый адрес массива данных,
M - страницы ёмкостью 2^n,
D - смещение относительно начала массива M (рисунок 7.3).

Отсюда название способа: относительная, базовая адресация. Базовый адрес B_M указывается в специальном регистре ЦП – базовом регистре, отсюда – второе название способа – базовая адресация.

Смещение D указывается в адресной части команды. Формирование исполнительного адреса A осуществляется по схеме, представленной на рисунке 7.3.

В поле B команды указывается номер регистра в локальной памяти ЛП, который используется в качестве базового регистра ЦП (для хранения базового адреса массива).

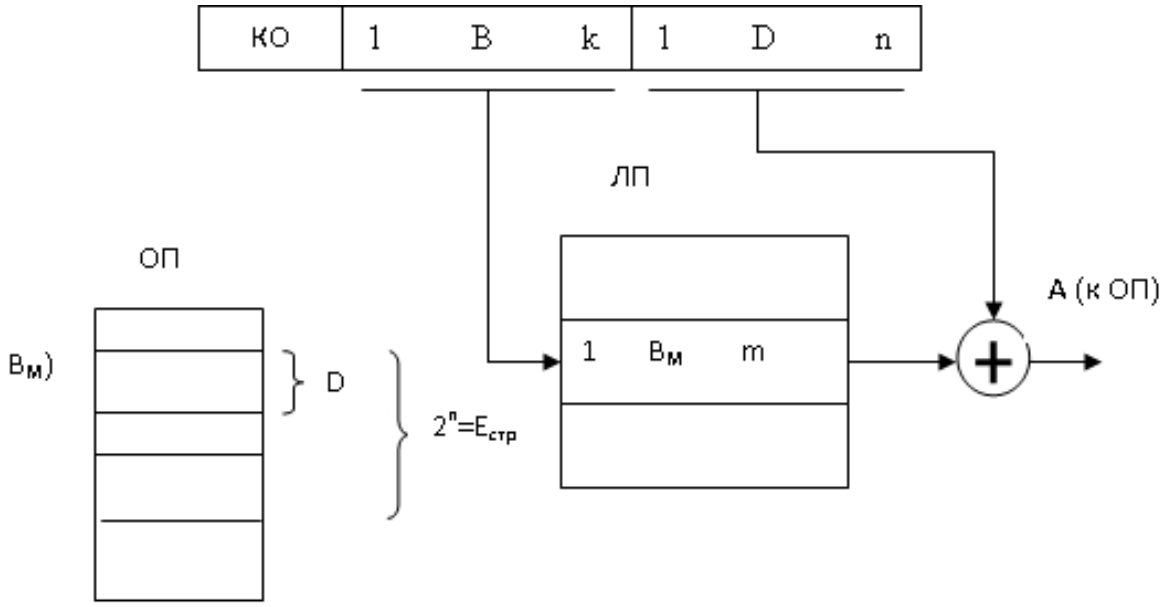


Рис.7.3.– Формирование исполнительного адреса

Вычисление относительного адреса A производится по формуле (2.8): базовый адрес B_M извлекается из ЛП по номеру B и складывается со смещением D из адресной части команды – A=[B]+D. Здесь [B] – содержимое регистра ЛП с номером B. Сложение производится по схеме(рисунок 7.4):

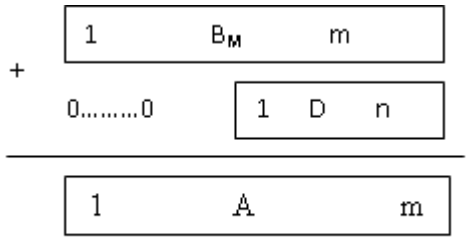


Рис.7.4.– Схема сложения

Следует отметить, что основное назначение относительной адресации – обеспечить перемещаемость программ и данных при динамическом распределении ОП в мультипрограммном режиме работы компьютера. Для

изменения расположения программы и данных в ОП достаточно новый базовый адрес занести в базовый регистр ЦП. Этим обычно занимается ОС.

В РС роль базового регистра ЦП могут выполнять регистры BX, BP, а также сегментные регистры CS, DS, ES, SS.

Следует отметить, что страничная адресация является частным случаем относительной (базовой) адресации: младшие n разрядов базового адреса равны нулю, а старшие содержат номер страницы, т.е. в случае страничной адресации на базовый адрес накладывается ограничение – он должен быть кратным 2^n, емкости страницы.

Таким образом, относительная адресация обеспечивает: перемещаемость программ и данных, расширение адресного пространства ОП до 2^m, уменьшение длины адресной части команды.

7.2.4. Косвенная адресация

Четвертый способ – **косвенная адресация** – в адресной части команды указывается не прямой, а косвенный адрес операнда $A_к$, т.е. адрес адреса операнда: $A_к \rightarrow A \rightarrow O$.

По адресу $A_к$ производится первое обращение к памяти и извлекается адрес операнда А, а затем уже по адресу А производится второе обращение и извлекается операнд О.

Недостаток косвенной адресации – дополнительное обращение к памяти и, как следствие, уменьшение быстродействия ЦП.

Этот недостаток можно сгладить, если адрес А хранить не в ОП, а в ЛП, т.е. использовать косвенный адрес типа $R_к$, который и указывается в адресной части команды.

Схема формирования адреса в этом случае имеет вид, представленный на рисунке 7.5.

Поскольку время обращения к регистрам ЛП обычно в 5…10 раз меньше, чем к ячейкам ОП, то время выборки операнда увеличивается незначительно – на 10…20%.

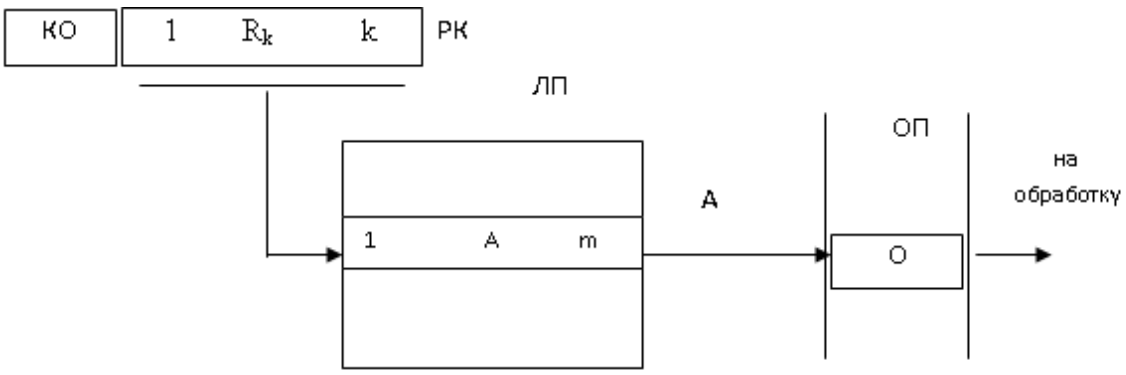


Рис.7.5.– Схема формирования адреса при косвенной адресации

Достоинства:

- возможность обработки адреса операнда – $A=A \pm \Delta A$,
- уменьшение длины адресной части команды при косвенной регистровой адресации, т. к. адрес типа R короче адреса типа A.

При косвенной адресации в качестве регистров в IBM PC могут использоваться регистры BX, DX, SI, DI и др.

7.2.5. Индексная адресация

Пятый способ адресации – **индексная адресация –** используется для адресации элементов массивов (переменных с индексами): $x_i \in X$,

$X=\{x_0, x_1, ..., x_I\}$, i – порядковый номер элемента в массиве X. Суть: адрес элемента x_i складывается из двух составляющих:

$$A=<x_i>=B_X+i, \qquad (7.2)$$

где B_X – базовый (начальный) адрес массива X, указывается в адресной части команды,

i – индекс, указывается в специальном **индексном регистре** ЦП,

$<x_i>$ - адрес элемента x_i.

В качестве индексного регистра можно использовать один из РОН. В этом случае вычисление адреса A осуществляется по формуле (7.3):

$$A=[X]+B_X, \qquad (7.3)$$

Где: [X] – индекс, извлеченный из индексного регистра с номером X. Схема формирования адреса A представлена на рисунке 7.6.

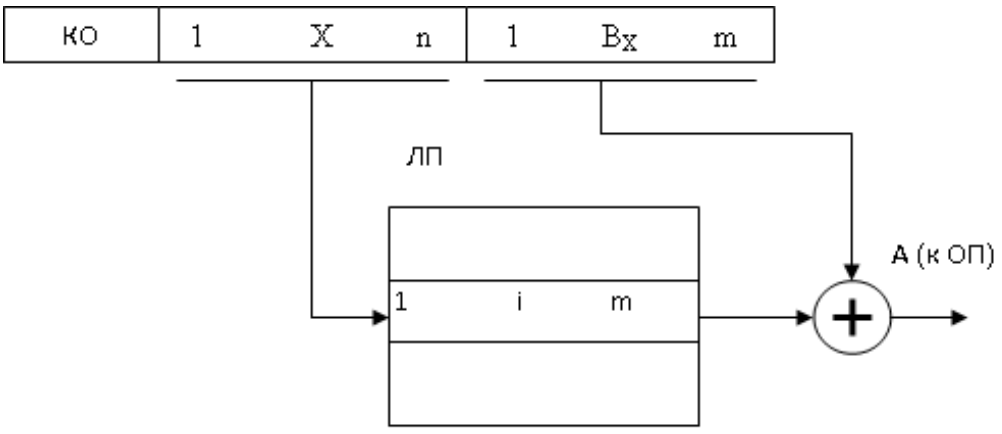


Рис.7.6.– Схема формирования адреса A

В РС в качестве индексных регистров можно использовать регистры SI, DI. Модификация индекса i осуществляется по схеме $i=i \pm \Delta i$, т.е. путём выполнения операций с индексными регистрами BI, DI.

7.2.6. **Индексно-относительная адресация**

Шестой способ – **индексно-относительная адресация** – совместное использование индексной и относительной адресации.

Адрес А формируется как сумма трех величин:

$$A=[X]+[B]+D, \qquad (7.4)$$

где: - [X] = i - индекс, $B_X=[B]+D=B_M+D$ – базовый адрес массива X.

Схема формирования адреса А представлена на рисунке 7.7.

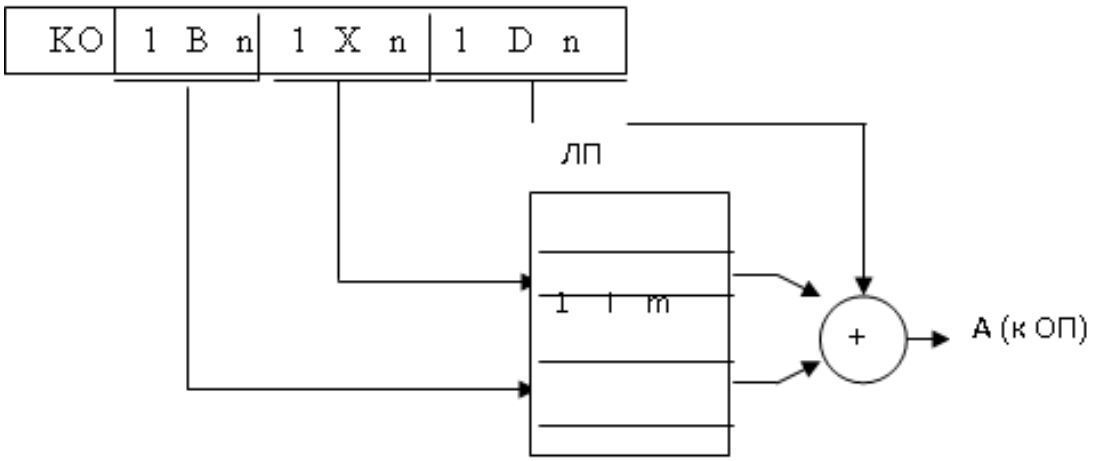


Рис.7.7.– Схема формирования адреса А

Сложение в формуле (7.4) производится по схеме (рисунок 7.8):

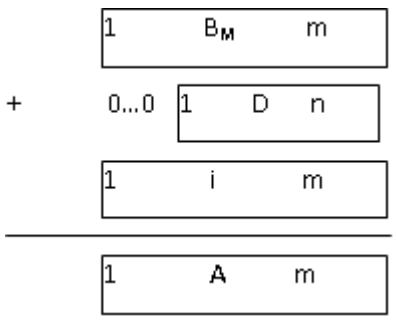
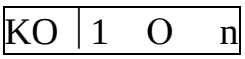


Рис.7.8.– Схема сложения

7.2.7. Непосредственная адресация

Седьмой способ – **непосредственная адресация –** адресная часть команды используется непосредственно для размещения операнда О:

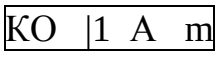


Операнд О – это обычно константа.

7.2.8. **Неявная адресация**

Восьмой – **неявная адресация** – по умолчанию известно положение одного из операндов.

Используется с целью экономии адресного поля в формате команды:



.

Схема выполнения операции: АС:=АС∗[А].

Первый операнд извлекается из регистра-аккумулятора АС по умолчанию, второй операнд – из памяти по адресу А, указанному в адресной части команды, над ними выполняется операция Σ, заданная КО, и результат заносится в регистр АС.

В РС неявная адресация используется в командах ввода-вывода IN/OUT, в которых в адресной части команды указывают адрес порта ввода-вывода, а источником (приёмником) является регистр-аккумулятор ЦП: АС↔порт. Кроме того, в РС неявная адресация используется для указания местоположения операндов в сложных операциях с плавающей запятой (неявная адресация регистров).

Итак, рассмотренные 8 способов адресации используются для адресации информации в адресном пространстве ЦП, которое включает пространство ЛП (РОН), пространство ОП и пространство ввода-вывода.

Размер адресного пространства и длина адреса, по которому производится обращение к элементам адресного пространства, связаны известным соотношением $E=2^m$, m - длина адреса.

Отсюда следует, что для обращения, например, к байту ОП на адресный вход ОП необходимо подавать m - разрядный адрес А от источника запроса, например, от ЦП. Для этих целей обычно используют ША шириной m разрядов.

Например, в РС АТ m=24, в 32-разрядных РС - m=32 разряда. Информация передаётся (принимается) по ШД. В РС, например, ШД 16-разрядная, в 32-разрядных компьютерах разрядность ШД n=32.

Направление обмена задаётся по шине управления ШУ путём подачи сигнала управления ЧТ (ЗП).

Вопросы для самопроверки.

1. Что такое система машинных команд?
2. Каковы особенности системы машинных команд?
3. Что такое машинная операция?
4. Что такое адресация информации?
5. Приведите назначение регистров памяти.
6. Опишите прямую адресацию.
7. Охарактеризуйте индексную и косвенную адресацию.
8. Приведите достоинства и недостатки страничной адресации.
9. Приведите схему формирования исполнительного адреса при базовой адресации.
10. Опишите неявную адресацию.

Модуль 8.

СПЕЦИАЛЬНЫЕ ВОПРОСЫ ОРГАНИЗАЦИИ ПАМЯТИ ЭВМ.

Цель модуля: изучение студентами вопросов организации памяти ЭВМ различного уровня.

В результате изучения модуля студенты должны знать:

- виртуальную организацию памяти и адресного пространства;
- концепцию виртуальной памяти;
- страничную организацию и сегментацию памяти;
- организацию защиты памяти в ЭВМ.

Содержание модуля:

8.1 Организация адресного пространства внешней памяти. Виртуальная организация памяти.

8.2 Виртуальная память и организация защиты памяти.

8.2.1 Концепция виртуальной памяти.

8.2.2 Страничная организация памяти.

8.2.3 Сегментация памяти.

8.2.4 Организация защиты памяти в ЭВМ.

8.2.5 Средства защиты памяти в персональной ЭВМ.

8.3 Специализированные ЭВМ*.

Память третьего уровня – внешняя память – строится на основе различного рода ВЗУ: НМД, НМЛ, НОД (CD–ROM и др.). При этом накопители один от другого отличаются номерами (адресами):

ВЗУ0, ВЗУ1, ... , ВЗУ(N-1).

Информационное пространство каждого $ВЗУ_i$ адресуется обычно с точностью до блока информации фиксированной ёмкости (в PC, например, типичный размер блока (сектора) 2^9=512B).

Номер блока обычно указывают напрямую, т.е. используется прямая адресация.

Информационная ёмкость накопителя оценивается количеством блоков, умноженным на ёмкость блока. При обращении к накопителю адресуются блоки (секторы).

Доступ к байтам становится возможным только после ввода блока в ОП.

8.1. Организация адресного пространства внешней памяти. Виртуальная организация памяти

Таким образом, формат адреса при обращении к ВП:

1	№ ВЗУ	n 1	№ блока	k

Здесь: 2^n – количество регистров в контроллере ВЗУ, 2^k – количество блоков на носителе информации ВЗУ.

Ёмкость ВП практически не ограничена за счёт съёмных носителей информации. Их идентификацией (т.е. адресацией) руководит человек. На техническом уровне для ВЗУ один съёмный носитель от другого неотличим.

ВЗУ как ПУ подключаются к ЭВМ не напрямую, а через специальные устройства управления – т.н. контроллеры ПУ (рисунок 8.1).

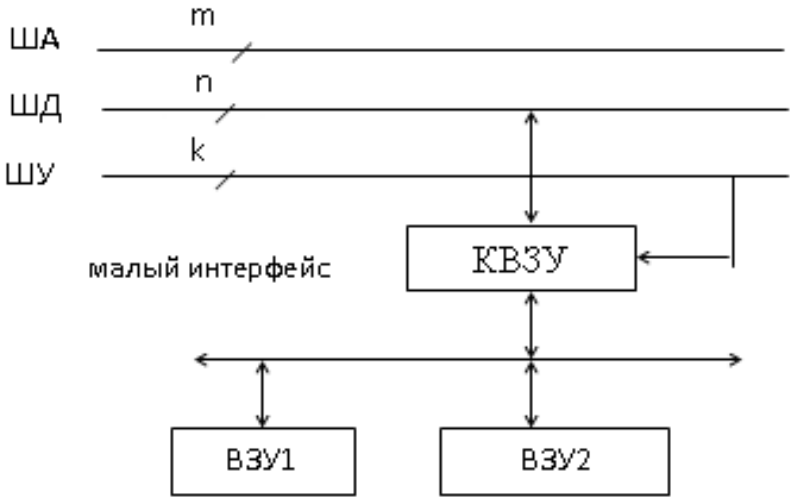


Рис.8.1.– Подключение ВЗУ

По шине адреса номер ВЗУ передаётся в контроллер ВЗУ (т.е. выбирается один из контроллеров), а по ШД – номер ВЗУ и номер блока (сектора) на носителе информации, по ШУ – направление обмен – ввод или вывод.

После этого производится поиск адресуемого блока на носителе и запускается алгоритм обмена информацией между ОП и ВЗУ.

Этот алгоритм реализуется КПУ, механизмом ВЗУ и последовательностью команд, которые поступают от ЦП. Обычно эти программы порождаются программами ввода-вывода, которые являются программами ОС, а не программами пользователя. Эти специальные программы ввода-вывода инициализируются из программ пользователя, когда в них дело доходит до операторов ‘ввод’ или ‘вывод’.

В этот момент их выполнение прерывается и управление передается ОС, а именно, специальным программам ввода-вывода ОС – т.н. супервизору (монитору) ввода-вывода. Он собирает необходимую информацию об обмене из соответствующих таблиц ОС и каталогов и вызывает программу-драйвер. Под управлением драйвера и осуществляется обмен между ОП и ВП.

Следует отметить, что программирование (организация) обмена информацией между ОП и ВП требует от исполнителя высокой квалификации.

В противном случае обмен организуется неоптимальным образом, что приводит к значительным потерям времени на ‘свопинг’.

Свопинг – это взаимный обмен информацией между ОП и ВП: прежде чем в ОП ввести новую информацию, необходимо освободить часть ОП путём переписи старой информации во ВП. В этой ситуации актуальной является задача автоматизации обмена (свопинга). Такого рода автоматизация обычно осуществляется на базе т.н. **виртуальной организации памяти ЭВМ.**

Идея (концепция) виртуальной (кажущейся) памяти заключается в следующем: ОП и ВП в функциональном (логическом) отношении (т.е. с точки зрения пользователя) рассматривается как одноуровневая память с прямым (произвольным) доступом (как бы одноуровневая память): ОП и ВП – единое адресное пространство.

Для адресации такой памяти используются адреса 0,1, ... ,$E_{оп}$-1,$E_{оп}$,$E_{оп}$+1, ... ,E-1, где $E=E_{оп}+E_{вп}$.

Например, в ПЭВМ типа IBM PC шина адреса ША содержит 32 разряда, т.е. $E_{оп}$=4ГВ (2^{32}В). Размер же виртуального адресного пространства – 2^{46}В=64ТВ (32+14=46).

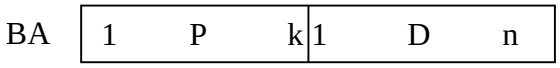
Для адресации виртуальной памяти используются длинные (m=46, например) **виртуальные адреса**. При этом разрешено считать, что информация, которая хранится во ВП, также доступна ЦП, как и информация, которая хранится в ОП.

Для того, чтобы обеспечить это технически в ЭВМ встраиваются специальные программные и технические средства, которые и обеспечивают **автоматический свопинг**.

Эти средства должны обеспечить выполнение двух действий:

- преобразование виртуальных адресов в физические, т. е. либо в адреса ячеек ОП, либо, если это ВП, в адреса ВЗУ и номера блоков на носителе информации, по которым и производится обращение к памяти;
- если адресуемая информация в момент обращения находится не в ОП, а во ВП, то передачу её (вместе с блоком информации) из ВП в ОП на заранее освобожденное место (свопинг).

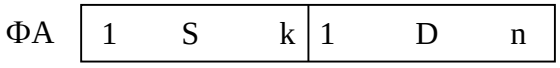
Технически наиболее просто указанные действия реализуются при страничной организации памяти. В этом случае для адресации ячеек виртуальной памяти используются длинные виртуальные адреса ВА следующего формата:



где m=k+n - длина ВА,

P – номер виртуальной страницы,

D – номер ячейки в странице. Физический адрес имеет аналогичную структуру:



,

только старшие разряды - поле S - указывают адрес (номер) физической страницы.

Понятие ‘**виртуальная страница’** связывается с той информацией, которая хранится в этой странице, а не с местом её хранения. А вот понятие ‘**физическая страница’** связано с местом хранения виртуальной страницы с номером P.

Таким образом, виртуальные страницы подвижны, а физические – неподвижны. Это означает, что одна и та же виртуальная страница в разные моменты времени может находиться в различных физических страницах памяти: то в ОП, то в ВП, то опять в ОП. Т.е. информация в процессе функционирования ЭВМ перемещается внутри физической памяти.

За процессом перемещения виртуальных страниц **необходимо следить.** С этой целью текущее состояние виртуальной памяти отображается в виде **таблицы страниц**. В таблице страниц каждой виртуальной странице с номером P ставится в соответствие одна строка.

В этой строке и указывается, в каком конкретно месте физической памяти расположены виртуальные страницы в данный момент времени, т.е. в этой строке указывается номер физической страницы S.

Кроме того, здесь же указывается и признак доступности страницы ЦП d_p, если d_p=0 - страница P находится во ВП ($P \notin$ОП),если d_p=1 – страница P находится в ОП ($P \in$ОП), доступна ЦП. В случае d_P=1 преобразование ВА в физический ФА производится следующим образом (рисунок 8.2).

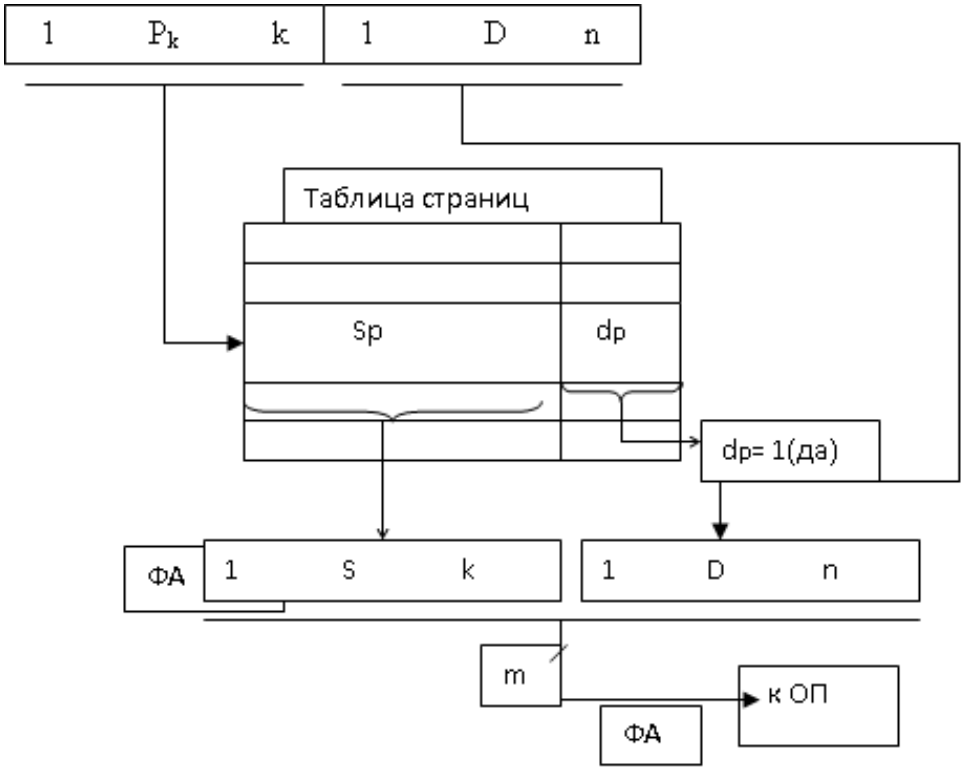


Рис.8.2.– Преобразование ВА в ФА

Физический адрес формируется конкатенацией (склеиванием) поля S из таблицы страниц и поля D из ВА: ФА=S.D. В IBM PC физический адрес формируется суммированием этих полей: ФА=S+D, где S интерпретируется как базовый адрес, а поле D – как смещение.

В случае если d_p =0 – т.е. страница недоступна ЦП, происходит прерывание текущей команды и управление передаётся средствам перемещения страниц, т.е. средствам, обеспечивающим свопинг.

Эти средства обеспечивают перемещение виртуальной страницы P из ВП в ОП, предварительно удалив страницу из ОП в свободное место (в свободную физическую страницу) ВП.

Порядок использования таблицы страниц можно проиллюстрировать рисунком 8.3.

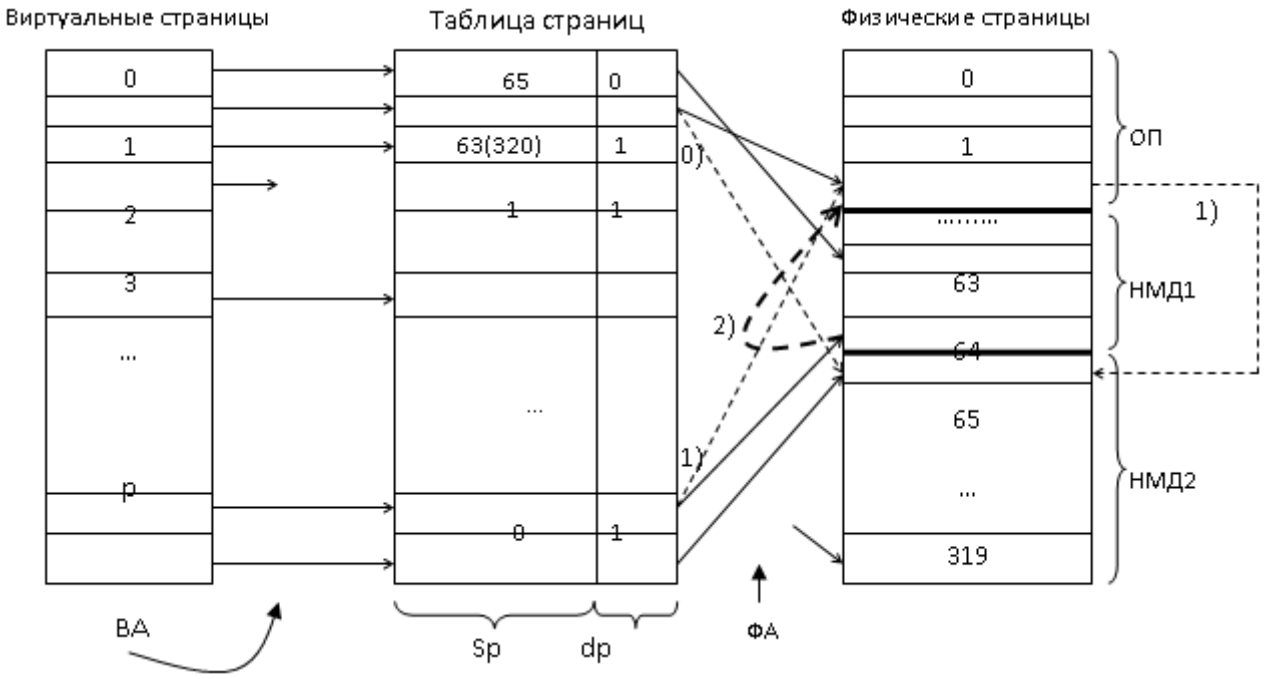


Рис.8.3.–Порядок использования таблицы страниц

Хранить таблицы страниц проще всего в ОП. Но при этом при каждом обращении по ВА будет производиться два физических обращения к ОП: сначала к таблице страниц, а затем к ячейке ОП по ФА. Ясно, что такая организация памяти снижает быстродействие памяти вдвое. Необходимо хранить таблицы страниц не в ОП, а в СОП (в КЭШ, например). Это дороже, но быстрее.

8.2. Виртуальная память и организация защиты памяти

8.2.1. Концепция виртуальной памяти

Общепринятая в настоящее время концепция виртуальной памяти появилась достаточно давно. Она позволила решить целый ряд актуальных вопросов организации вычислений. Прежде всего, к числу таких вопросов относится обеспечение надежного функционирования мультипрограммных систем.

В любой момент времени компьютер выполняет множество процессов или задач, каждая из которых располагает своим адресным пространством. Было бы слишком накладно отдавать всю физическую память какой-то одной задаче тем более, что многие задачи реально используют только небольшую часть своего адресного пространства. Поэтому необходим механизм разделения небольшой физической памяти между различными задачами. Виртуальная память является одним из способов реализации такой

возможности. Она делит физическую память на блоки и распределяет их между различными задачами. При этом она предусматривает также некоторую схему защиты, которая ограничивает задачу теми блоками, которые ей принадлежат. Большинство типов виртуальной памяти сокращают также время начального запуска программы на процессоре, поскольку не весь программный код и данные требуются ей в физической памяти, чтобы начать выполнение.

Другой вопрос, тесно связанный с реализацией концепции виртуальной памяти, касается организации вычислений на компьютере задач очень большого объема. Если программа становилась слишком большой для физической памяти, часть ее необходимо было хранить во внешней памяти (на диске) и задача приспособить ее для решения на компьютере ложилась на программиста. Программисты делили программы на части и затем определяли те из них, которые можно было бы выполнять независимо, организуя оверлейные структуры, которые загружались в основную память и выгружались из нее под управлением программы пользователя. Программист должен был следить за тем, чтобы программа не обращалась вне отведенного ей пространства физической памяти. Виртуальная память освободила программистов от этого бремени. Она автоматически управляет двумя уровнями иерархии памяти: основной памятью и внешней (дисковой) памятью.

Кроме того, виртуальная память упрощает также загрузку программ, обеспечивая механизм автоматического перемещения программ, позволяющий выполнять одну и ту же программу в произвольном месте физической памяти.

Системы виртуальной памяти можно разделить на два класса: системы с фиксированным размером блоков, называемых страницами, и системы с переменным размером блоков, называемых сегментами. Ниже рассмотрены оба типа организации виртуальной памяти.

8.2.2. Страничная организация памяти

В системах со страничной организацией основная и внешняя память (главным образом дисковое пространство) делятся на блоки или страницы фиксированной длины. Каждому пользователю предоставляется некоторая часть адресного пространства, которая может превышать основную память компьютера и которая ограничена только возможностями адресации, заложенными в системе команд. Эта часть адресного пространства называется виртуальной памятью пользователя. Каждое слово в виртуальной памяти пользователя определяется виртуальным адресом, состоящим из двух

частей: старшие разряды адреса рассматриваются как номер страницы, а младшие - как номер слова (или байта) внутри страницы.

Управление различными уровнями памяти осуществляется программами ядра операционной системы, которые следят за распределением страниц и оптимизируют обмены между этими уровнями. При страничной организации памяти смежные виртуальные страницы не обязательно должны размещаться на смежных страницах основной физической памяти. Для указания соответствия между виртуальными страницами и страницами основной памяти операционная система должна сформировать таблицу страниц для каждой программы и разместить ее в основной памяти машины. При этом каждой странице программы, независимо от того находится ли она в основной памяти или нет, ставится в соответствие некоторый элемент таблицы страниц. Каждый элемент таблицы страниц содержит номер физической страницы основной памяти и специальный индикатор. Единичное состояние этого индикатора свидетельствует о наличии этой страницы в основной памяти. Нулевое состояние индикатора означает отсутствие страницы в оперативной памяти.

Для увеличения эффективности такого типа схем в процессорах используется специальная полностью ассоциативная кэш-память, которая также называется буфером преобразования адресов (TLB tranclation-lookaside buffer). Хотя наличие TLB не меняет принципа построения схемы страничной организации, с точки зрения защиты памяти, необходимо предусмотреть возможность очистки его при переключении с одной программы на другую.

Поиск в таблицах страниц, расположенных в основной памяти, и загрузка TLB может осуществляться либо программным способом, либо специальными аппаратными средствами. В последнем случае для того, чтобы предотвратить возможность обращения пользовательской программы к таблицам страниц, с которыми она не связана, предусмотрены специальные меры. С этой целью в процессоре предусматривается дополнительный регистр защиты, содержащий описатель (дескриптор) таблицы страниц или базово-граничную пару. База определяет адрес начала таблицы страниц в основной памяти, а граница - длину таблицы страниц соответствующей программы. Загрузка этого регистра защиты разрешена только в привилегированном режиме. Для каждой программы операционная система хранит дескриптор таблицы страниц и устанавливает его в регистр защиты процессора перед запуском соответствующей программы.

Отметим некоторые особенности, присущие простым схемам со страничной организацией памяти. Наиболее важной из них является то, что все программы, которые должны непосредственно связываться друг с другом без вмешательства операционной системы, должны использовать общее

пространство виртуальных адресов. Это относится и к самой операционной системе, которая, вообще говоря, должна работать в режиме динамического распределения памяти. Поэтому в некоторых системах пространство виртуальных адресов пользователя укорачивается на размер общих процедур, к которым программы пользователей желают иметь доступ. Общим процедурам должен быть отведен определенный объем пространства виртуальных адресов всех пользователей, чтобы они имели постоянное место в таблицах страниц всех пользователей. В этом случае для обеспечения целостности, секретности и взаимной изоляции выполняющихся программ должны быть предусмотрены различные режимы доступа к страницам, которые реализуются с помощью специальных индикаторов доступа в элементах таблиц страниц.

Следствием такого использования является значительный рост таблиц страниц каждого пользователя. Одно из решений проблемы сокращения длины таблиц основано на введении многоуровневой организации таблиц. Частным случаем многоуровневой организации таблиц является сегментация при страничной организации памяти. Необходимость увеличения адресного пространства пользователя объясняется желанием избежать необходимости перемещения частей программ и данных в пределах адресного пространства, которые обычно приводят к проблемам переименования и серьезным затруднениям в разделении общей информации между многими задачами.

8.2.3. Сегментация памяти

Другой подход к организации памяти опирается на тот факт, что программы обычно разделяются на отдельные области-сегменты. Каждый сегмент представляет собой отдельную логическую единицу информации, содержащую совокупность данных или программ и расположенную в адресном пространстве пользователя. Сегменты создаются пользователями, которые могут обращаться к ним по символическому имени. В каждом сегменте устанавливается своя собственная нумерация слов, начиная с нуля.

Обычно в подобных системах обмен информацией между пользователями строится на базе сегментов. Поэтому сегменты являются отдельными логическими единицами информации, которые необходимо защищать, и именно на этом уровне вводятся различные режимы доступа к сегментам. Можно выделить два основных типа сегментов: программные сегменты и сегменты данных (сегменты стека являются частным случаем сегментов данных). Поскольку общие программы должны обладать свойством повторной входимости, то из программных сегментов допускается только выборка команд и чтение констант. Запись в программные сегменты может рассматриваться как незаконная и запрещаться системой. Выборка команд из сегментов данных также может считаться незаконной и любой

сегмент данных может быть защищен от обращений по записи или по чтению.

Для реализации сегментации было предложено несколько схем, которые отличаются деталями реализации, но основаны на одних и тех же принципах.

В системах с сегментацией памяти каждое слово в адресном пространстве пользователя определяется виртуальным адресом, состоящим из двух частей: старшие разряды адреса рассматриваются как номер сегмента, а младшие - как номер слова внутри сегмента. Наряду с сегментацией может также использоваться страничная организация памяти. В этом случае виртуальный адрес слова состоит из трех частей: старшие разряды адреса определяют номер сегмента, средние - номер страницы внутри сегмента, а младшие - номер слова внутри страницы.

Как и в случае страничной организации, необходимо обеспечить преобразование виртуального адреса в реальный физический адрес основной памяти. С этой целью для каждого пользователя операционная система должна сформировать таблицу сегментов. Каждый элемент таблицы сегментов содержит описатель (дескриптор) сегмента (поля базы, границы и индикаторов режима доступа). При отсутствии страничной организации поле базы определяет адрес начала сегмента в основной памяти, а граница - длину сегмента. При наличии страничной организации поле базы определяет адрес начала таблицы страниц данного сегмента, а граница - число страниц в сегменте. Поле индикаторов режима доступа представляет собой некоторую комбинацию признаков блокировки чтения, записи и выполнения.

Таблицы сегментов различных пользователей операционная система хранит в основной памяти. Для определения расположения таблицы сегментов выполняющейся программы используется специальный регистр защиты, который загружается операционной системой перед началом ее выполнения. Этот регистр содержит дескриптор таблицы сегментов (базу и границу), причем база содержит адрес начала таблицы сегментов выполняющейся программы, а граница - длину этой таблицы сегментов. Разряды номера сегмента виртуального адреса используются в качестве индекса для поиска в таблице сегментов. Таким образом, наличие базово-граничных пар в дескрипторе таблицы сегментов и элементах таблицы сегментов предотвращает возможность обращения программы пользователя к таблицам сегментов и страниц, с которыми она не связана. Наличие в элементах таблицы сегментов индикаторов режима доступа позволяет осуществить необходимый режим доступа к сегменту со стороны данной программы. Для повышения эффективности схемы используется ассоциативная кэш-память.

Отметим, что в описанной схеме сегментации таблица сегментов с индикаторами доступа предоставляет всем программам, являющимся частями некоторой задачи, одинаковые возможности доступа, т. е. она определяет единственную область (домен) защиты. Однако для создания защищенных подсистем в рамках одной задачи для того, чтобы изменять возможности доступа, когда точка выполнения переходит через различные программы, управляющие ее решением, необходимо связать с каждой задачей множество доменов защиты. Реализация защищенных подсистем требует разработки некоторых специальных аппаратных средств. Рассмотрение таких систем, которые включают в себя кольцевые схемы защиты, а также различного рода мандатные схемы защиты, выходит за рамки данного обзора.

8.2.4. Организация защиты памяти в ЭВМ

При мультипрограммном режиме работы ЭВМ в ее памяти одновременно могут находиться несколько независимых программ. Поэтому необходимы специальные меры по предотвращению или ограничению обращений одной программы к областям памяти, используемым другими программами. Программы могут также содержать ошибки, которые, если этому не воспрепятствовать, приводят к искажению информации, принадлежащей другим программам. Последствия таких ошибок особенно опасны, если разрушению подвергнутся программы операционной системы. Другими словами, надо исключить воздействие программы пользователя на работу программ других пользователей и программ операционной системы. Следует защищать и сами программы от находящихся в них возможных ошибок.

Таким образом, средства защиты памяти должны предотвращать

- неразрешенное взаимодействие пользователей друг с другом,
- несанкционированный доступ пользователей к данным,
- повреждение программ и данных из-за ошибок в программах,
- намеренные попытки разрушить целостность системы,
- использование информации в памяти не в соответствии с ее функциональным назначением.

Чтобы воспрепятствовать разрушению одних программ другими, достаточно защитить область памяти данной программы от попыток записи в нее со стороны других программ, а в некоторых случаях и своей программы (защита от записи), при этом допускается обращение других программ к этой области памяти для считывания данных.

В других случаях, например при ограничениях на доступ к информации, хранящейся в системе, необходимо запрещать другим программам любое

обращение к некоторой области памяти, как на запись, так и на считывание. Такая защита от записи и считывания помогает в отладке программы, при этом осуществляется контроль каждого случая обращения за область памяти своей программы.

Для облегчения отладки программ желательно выявлять и такие характерные ошибки в программах, как попытки использования данных вместо команд или команд вместо данных в собственной программе, хотя эти ошибки могут и не разрушать информацию (несоответствие функционального использования информации).

Если нарушается защита памяти, исполнение программы приостанавливается и вырабатывается запрос прерывания по нарушению защиты памяти.

Защита от вторжения программ в чужие области памяти может быть организована различными методами. Но при любом подходе реализация защиты не должна заметно снижать производительность компьютера и требовать слишком больших аппаратурных затрат.

Методы защиты базируются на некоторых классических подходах, которые получили свое развитие в архитектуре современных ЭВМ. К таким методам можно отнести защиту отдельных ячеек, метод граничных регистров, метод ключей защиты.

Защита отдельных ячеек памяти организуется в ЭВМ, предназначенных для работы в системах управления, где необходимо обеспечить возможность отладки новых программ без нарушения функционирования находящихся в памяти рабочих программ, управляющих технологическим процессом. Это может быть достигнуто выделением в каждой ячейке памяти специального "разряда защиты". Установка этого разряда в "1" запрещает производить запись в данную ячейку, что обеспечивает сохранение рабочих программ. Недостаток такого подхода - большая избыточность в кодировании информации из-за излишне мелкого уровня защищаемого объекта (ячейка).

В системах с мультипрограммной обработкой целесообразно организовывать защиту на уровне блоков памяти, выделяемых программам, а не отдельных ячеек.

Метод граничных регистров заключается во введении двух граничных регистров, указывающих верхнюю и нижнюю границы области памяти, куда программа имеет право доступа.

При каждом обращении к памяти проверяется, находится ли используемый адрес в установленных границах. При выходе за границы обращение к памяти не производится, а формируется запрос прерывания, передающий управление операционной системе. Содержание граничных регистров устанавливается операционной системой при загрузке программы в память.

Модификация этого метода заключается в том, что один регистр используется для указания адреса начала защищаемой области, а другой содержит длину этой области.

Метод граничных регистров, обладая несомненной простотой реализации, имеет и определенные недостатки. Основным из них является то, что этот метод поддерживает работу лишь с непрерывными областями памяти.

Метод ключей защиты, в отличие от предыдущего, позволяет реализовать доступ программы к областям памяти, организованным в виде отдельных модулей, не представляющих собой единый массив.

Память в логическом отношении делится на одинаковые блоки, например, страницы. Каждому блоку памяти ставится в соответствие код, называемый **ключом защиты памяти**, а каждой программе, принимающей участие в мультипрограммной обработке, присваивается код **ключа программы**. Доступ программы к данному блоку памяти для чтения и записи разрешен, если ключи совпадают (то есть данный блок памяти относится к данной программе) или один из них имеет код 0 (код 0 присваивается программам операционной системы и блокам памяти, к которым имеют доступ все программы: общие данные, совместно используемые подпрограммы и т. п.). Коды ключей защиты блоков памяти и ключей программ устанавливаются операционной системой.

В ключе защиты памяти предусматривается дополнительный разряд режима защиты. Защита действует только при попытке записи в блок, если в этом разряде стоит 0, и при любом обращении к блоку, если стоит 1. Коды ключей защиты памяти хранятся в специальной памяти ключей защиты, более быстродействующей, чем оперативная память.

При обращении к памяти группа старших разрядов адреса ОЗУ, соответствующая номеру блока, к которому производится обращение, используется как адрес для выборки из памяти ключей защиты кода ключа защиты, присвоенного операционной системой данному блоку. Схема анализа сравнивает ключ защиты блока памяти и ключ программы, находящийся в регистре слова состояния программы (ССП), и вырабатывает сигнал "Обращение разрешено" или сигнал "Прерывание по защите памяти". При этом учитываются значения режима обращения к ОЗУ (запись или считывание), указываемого триггером режима обращения ТгРО, и режима защиты, установленного в разряде режима обращения (РРО) ключа защиты памяти.

8.2.5. Средства защиты памяти в персональной ЭВМ

Защита памяти в персональной ЭВМ делится на защиту при управлении памятью и защиту по привилегиям.

Средства **защиты при управлении памятью** осуществляют проверку

- превышения эффективным адресом длины сегмента,
- прав доступа к сегменту на запись или только на чтение,
- функционального назначения сегмента.

Первый механизм базируется на методе граничных регистров. При этом начальные адреса того или иного сегмента программы устанавливаются операционной системой. Для каждого сегмента фиксируется его длина. При попытке обращения по относительному адресу, превышающему длину сегмента, вырабатывается сигнал нарушения защиты памяти.

При проверке функционального назначения сегмента определяются операции, которые можно проводить над находящимися в нем данными. Так, сегмент, представляющий собой стек программы, должен допускать обращение, как на запись, так и на чтение информации. К сегменту, содержащему программу, можно обращаться только на исполнение. Любое обращение на запись или чтение данных из этого сегмента будет воспринято как ошибочное. Здесь наблюдается некоторый отход от принципов Неймана в построении ЭВМ, в которых утверждается, что любая информация, находящаяся в ЗУ, функционально не разделяется на программу и данные, а ее идентификация проводится лишь на стадии применения данной информации. Очевидно, что такое развитие вызвано во многом появлением мультипрограммирования и необходимостью более внимательного рассмотрения вопросов защиты информации.

Защита по привилегиям фиксирует более тонкие ошибки, связанные, в основном, с разграничением прав доступа к той или иной информации.

В какой-то степени защиту по привилегиям можно сравнить с классическим методом ключей защиты памяти. Различным объектам (программам, сегментам памяти, запросам на обращение к памяти и к внешним устройствам), которые должны быть распознаны процессором, присваивается идентификатор, называемый уровнем привилегий. Процессор постоянно контролирует, имеет ли текущая программа достаточные привилегии, чтобы:

- выполнять некоторые команды,
- выполнять команды ввода-вывода на том или ином внешнем устройстве,
- обращаться к данным других программ,
- вызывать другие программы.

На аппаратном уровне в процессоре различаются 4 уровня привилегий. Наиболее привилегированными являются программы на уровне 0.

Число программ, которые могут выполняться на более высоком уровне привилегий, уменьшается от уровня 3 к уровню 0. Программы уровня 0

действуют как ядро операционной системы. Поэтому уровни привилегий обычно изображаются в виде четырех колец защиты.

Как правило, **распределение программ по кольцам защиты** имеет следующий вид:

уровень 0 – ядро ОС, обеспечивающее инициализацию работы, управление доступом к памяти, защиту и ряд других жизненно важных функций, нарушение которых полностью выводит из строя процессор;

уровень 1 – основная часть программ ОС (утилиты);

уровень 2 – служебные программы ОС (драйверы, СУБД, специализированные подсистемы программирования и др.);

уровень 3 – прикладные программы пользователя.

Конкретная операционная система не обязательно должна поддерживать все четыре уровня привилегий. Так, ОС **UNIX** работает с двумя кольцами защиты: супервизор (уровень 0) и пользователь (уровни 1,2,3). Операционная система **OS/2** поддерживает три уровня: код ОС работает в кольце 0, специальные процедуры для обращения к устройствам ввода-вывода действуют в кольце 1, а прикладные программы выполняются в кольце 3.

Простую незащищенную систему можно целиком реализовать в одном кольце 0 (в других кольцах это сделать невозможно, так как некоторые команды доступны только на этом уровне).

Уровень привилегий сегмента определяется полем DPL уровня привилегий его дескриптора. Уровень привилегий запроса к сегменту определяется уровнем привилегий RPL, закодированном в селекторе. Обращение к сегменту разрешается только тогда, когда уровень привилегий сегмента не выше уровня запроса. Обращение к программам, находящимся на более высоком уровне привилегий (например, к утилитам операционной системы из программ пользователя), возможно с помощью специальных аппаратных механизмов - шлюзов вызова.

При страничном преобразовании адреса применяется простой двухуровневый механизм защиты: пользователь (уровень 3) / супервизор (уровни 0,1,2), указываемый в поле U/S соответствующей таблицы страниц.

При сегментно-страничном преобразовании адреса сначала проверяются привилегии при доступе к сегменту, а затем - при доступе к странице. Это дает возможность установить более высокую степень защиты отдельных страниц сегмента.

8.3. Специализированные ЭВМ. Общие понятия

Специализированная ЭВМ [specialized computer] – ЭВМ, предназначенная для решения узкого класса определенных задач.

Характеристики и архитектура машин этого класса определяются спецификой задач, на которые они ориентированы, что делает их более эффективными в соответствующем применении по отношению к

универсальным ЭВМ. К разряду специализированных могут быть отнесены, в частности, - “управляющие”, “бортовые“, “бытовые“ и “выделенные“ ЭВМ.

1. Управляющая ЭВМ [control computer] – ЭВМ, предназначенная для автоматического управления объектом (устройством, системой, процессом) в реальном масштабе времени. Сопряжение ЭВМ с объектом управления производится с помощью аналого-цифровых и цифро-аналоговых преобразователей.
2. Бортовая ЭВМ [onboard computer] – специализированная управляющая ЭВМ, устанавливаемая на борту транспортного средства (самолета, спутника, корабля, автомобиля и т.п.) и предназначенная для оптимального управления функционированием других бортовых устройств, в частности, связанных с управлением перемещением своего носителя в пространстве.
3. Выделенная ЭВМ [dedicated computer] – разновидность (как правило) однокристальной специализированной ЭВМ, встроенной в какое-либо устройство с целью управления им или передачи ему данных. Используется в бытовой технике и других видах устройств – нагревательных приборах, часах, автомобилях, магнитофонах и т.д.
4. Бытовая (домашняя) ЭВМ [home computer] – То же, что и домашняя ПЭВМ или домашний ПК.

Вопросы для самопроверки.

1. Что такое память третьего уровня?
2. Как осуществляется подключение ВЗУ?
3. Поясните термины **виртуальный адрес** и **автоматический свопинг**.
4. Назовите различия между понятиями **виртуальная** и **физическая** страница.
5. Как происходит преобразование виртуального адреса в физический?
6. Что такое сегментация памяти?
7. Что должны предотвращать средства защиты памяти?
8. Как организуется защита отдельных ячеек памяти?
9. В чем заключается метод граничных регистров?
10. Приведите распределение программ по кольцам защиты.
11. Какие ЭВМ могут быть отнесены к разряду специализированных?

Модуль 9.

ПРАКТИЧЕСКИЕ ЗАНЯТИЯ

Цель модуля: изучение студентами структуры и функций микро-ЭВМ для помощи в курсовом проектировании.

В результате изучения модуля студенты должны:

- освоить технологию проектирования различных блоков и устройств микро-ЭВМ;
- уметь грамотно технически реализовать поставленные в курсовом проектировании задачи в среде автоматизированного проектирования фирмы Altera: Max Plus II или Quartus II.

Содержание модуля:

9.1 Практическое занятие №1. Разработка функционального состава микро-ЭВМ заданной конфигурации.

9.2 Практическое занятие №2. Разработка структуры ПЗУ заданной конфигурации.

9.3 Практическое занятие №3. Разработка структуры ОЗУ заданной конфигурации.

9.4 Практическое занятие №4. Разработка узла местного управления фазами выполнения команд и структуры микропрограммы устройства управления.

9.5 Практическое занятие №5. Разработка устройства управления.

9.6 Практическое занятие №6. Построение арифметико-логического устройства.

9.7 Практическое занятие №7. Разработка контроллера прямого доступа к памяти.

9.8 Практическое занятие №8. Разработка системы прерываний. Реализация функционального состава микро-ЭВМ.

9.1. Практическое занятие №1. Разработка функционального состава микро-ЭВМ заданной конфигурации

Теория и методы решения для данного практического занятия представлены в модулях 1 и 2.

Задание:

- в соответствии с выданным преподавателем заданием разработать функциональную схему микро-ЭВМ в зависимости от типа архитектуры;
- указать на схеме взаимосвязь узлов и блоков микро-ЭВМ;
- указать в форме таблицы перечень основных сигналов, которые будут использоваться для задач дальнейшего проектирования для каждого из устройств в соответствии с заданием;
- представить преподавателю отчет, содержащий полную функциональную схему разрабатываемой ЭВМ; функциональное описание всех узлов и блоков, входящих в её состав, и таблицу используемых при реализации основных сигналов.

9.2. Практическое занятие №2. Разработка структуры ПЗУ заданной конфигурации

Теория и методы решения для данного практического занятия представлены в пп. 3.8.

Задание:

- в соответствии с выданным преподавателем заданием разработать функциональную схему ПЗУ в зависимости от его типа;
- указать в форме таблицы перечень сигналов, которые будут использоваться для реализации дальнейшего проектирования в соответствии с заданием;
- разработать устройство программно при помощи одного из следующих языков программирования: VHDL, AHDL, Verilog;
- представить преподавателю отчет, содержащий функциональную схему ПЗУ, перечень сигналов, листинг программы для реализации устройства, а также временную диаграмму его работы.

9.3. Практическое занятие №3. Разработка структуры ОЗУ заданной конфигурации

Теория и методы решения для данного практического задания представлены в пп. 3.1-3.7.

Задание:

- в соответствии с выданным преподавателем заданием разработать функциональную схему ОЗУ в зависимости от его типа;
- указать в форме таблицы перечень сигналов, которые будут использоваться для реализации дальнейшего проектирования в соответствии с заданием;
- разработать устройство программно при помощи одного из следующих языков программирования: VHDL, AHDL, Verilog;
- представить преподавателю отчет, содержащий функциональную схему ОЗУ, перечень сигналов, листинг программы для реализации устройства, а также временную диаграмму его работы.

9.4. Практическое занятие №4. Разработка узла местного управления фазами выполнения команд и структуры микропрограммы устройства управления

Теория и методы решения для данного практического занятия представлены в пп. 4.4, 5.1.

Задание:

- в соответствии с выданным преподавателем заданием разработать функциональную схему узла местного управления фазами выполнения команд в зависимости от типа устройства управления;
- указать в форме таблицы перечень сигналов, которые будут использоваться для реализации дальнейшего проектирования в соответствии с заданием;
- разработать устройство программно при помощи одного из следующих языков программирования: VHDL, AHDL, Verilog;
- представить преподавателю отчет, содержащий функциональную схему узла, перечень сигналов, микропрограмму для реализации устройства.

9.5. Практическое занятие №5. Разработка устройства управления

Теория и методы решения для данного практического занятия представлены в пп. 4.4, 5.1.

Задание:

- в соответствии с выданным преподавателем заданием разработать функциональную схему устройства управления;
- указать в форме таблицы перечень сигналов, которые будут использоваться для реализации дальнейшего проектирования в соответствии с заданием;
- разработать устройство программно при помощи одного из следующих языков программирования: VHDL, AHDL, Verilog;
- представить преподавателю отчет, содержащий функциональную схему устройства управления, перечень сигналов, листинг программы для реализации устройства, а также временную диаграмму его работы.

9.6. Практическое занятие №6. Построение арифметико-логического устройства

Теория и методы решения для данного практического занятия представлены в пп. 6.1, 6.2, 7.1, 7.2.

Задание:

- в соответствии с выданным преподавателем заданием разработать функциональную схему арифметико-логического устройства;
- указать в форме таблицы перечень сигналов, которые будут использоваться для реализации дальнейшего проектирования в соответствии с заданием;
- разработать устройство программно при помощи одного из следующих языков программирования: VHDL, AHDL, Verilog;
- представить преподавателю отчет, содержащий функциональную схему арифметико-логического устройства, перечень сигналов, листинг программы для реализации устройства, а также временную диаграмму его работы.

9.7. Практическое занятие №7. Разработка контроллера прямого доступа к памяти.

Теория и методы решения для данного практического занятия представлены в модуле 6.

Задание:

- в соответствии с выданным преподавателем заданием разработать функциональную схему контроллера прямого доступа к памяти;
- указать в форме таблицы перечень сигналов, которые будут использоваться для реализации дальнейшего проектирования в соответствии с заданием;
- разработать устройство программно при помощи одного из следующих языков программирования: VHDL, AHDL, Verilog;
- представить преподавателю отчет, содержащий функциональную схему контроллера прямого доступа к памяти, перечень сигналов, листинг программы для реализации устройства, а также временную диаграмму его работы.

9.8. Практическое занятие №8. Разработка системы прерываний. Реализация функционального состава микро-ЭВМ.

Теория и методы решения для данного практического занятия представлены в модуле 6.

Задание:

- в соответствии с выданным преподавателем заданием разработать функциональную схему системы прерываний;
- указать в форме таблицы перечень сигналов, которые будут использоваться для реализации дальнейшего проектирования в соответствии с заданием;
- разработать устройство программно при помощи одного из следующих языков программирования: VHDL, AHDL, Verilog;
- объединить результаты предыдущих работ, составив функциональную схему, разработанную на практическом занятии №1. Реализовать данную схему в среде автоматизированного проектирования.
- представить преподавателю **итоговый** отчет, содержащий реализованную итоговую функциональную схему, листинг программы для

реализации системы прерываний, временную диаграмму его работы, а также временную диаграмму работы всей схемы.

Модуль 10.

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Цель модуля: изучение студентами структуры основных устройств ЭВМ и их реализация в программном обеспечении фирмы Altera.

В результате изучения модуля студенты должны:

- освоить технологию проектирования различных блоков и устройств;
- уметь грамотно технически реализовать поставленные в лабораторном практикуме задачи в среде автоматизированного проектирования фирмы Altera: Max Plus II или Quartus II.

Содержание модуля:

Достижения в области микроэлектронных технологий привели к тому, что основу многих современных радиоэлектронных и вычислительных устройств составляют специализированные большие и сверхбольшие интегральные схемы (БИС и СБИС), позволяющие значительно улучшить технико-экономические характеристики аппаратуры конкретного назначения.

На практике используют пять способов реализации специализированных СБИС:

- полностью заказные – предполагающие полный цикл проектирования, включающие разработку всех литографических шаблонов на уровне отдельных областей транзисторных структур;
- заказные на основе библиотечных элементов – предполагающие использование заранее разработанных топологических библиотек элементов, узлов и блоков различной сложности и включающие разработку всех литографических шаблонов, но на уровне элементов, узлов и блоков;
- полузаказные – на основе базовых матричных кристаллов (БМК) – предполагающие использование заранее изготовленных «полуфабрикатов» – БМК-кристаллов с матрицами, так называемых базовых ячеек, каждая из

которых содержит набор нескоммутированных элементов (транзисторов, диодов, резисторов и др.), позволяющих посредством разработки только заказных шаблонов металлизации соединить базовые элементы в соответствии с проектируемой схемой для выполнения заданного набора функций;

- на основе постоянных запоминающих устройств (ПЗУ) – предполагающие программирование заранее изготовленных микросхем ПЗУ, содержащих полный дешифратор и программируемый шифратор (поле ячеек памяти), для выполнения определенных функций;
- на основе программируемых логических матриц (ПЛМ) – также предполагающие программирование заранее изготовленных микросхем, содержащих ПЛМ, отличающихся от ПЗУ наличием программируемых дешифраторов, что обеспечивает дополнительную гибкость.

Каждый из перечисленных способов имеет достоинства и недостатки, определяющие область его практического использования. Так, полностью заказные СБИС отличаются наиболее высокими показателями эффективности использования площади кристалла, быстродействия, надежности, но при этом предполагают длительный цикл проектирования (несколько месяцев работы проектного коллектива) и имеют наиболее высокую стоимость. Напротив, СБИС на основе ПЛМ позволяют сократить цикл проектирования до нескольких часов и свести к минимуму затраты на проектирование за счет определенной избыточности в числе логических матриц, длине связей и т.д., что несколько снижает качественные показатели.

Следует отметить, что переход к субмикронным топологическим размерам элементов СБИС (до 0,18 мкм) привел к резкому улучшению характеристик СБИС, в том числе и на основе ПЛМ. При этом степень интеграции, быстродействие, помехоустойчивость и надежность программируемых интегральных схем достигли столь высокого уровня, что в настоящее время во всем мире наблюдается тенденция все более широкого их использования для реализации специализированных СБИС.

Современные микросхемы на основе ПЛМ представляют собой сложные комплексы, включающие блоки ПЛМ, специальные встроенные блоки, мультиплексируемые линии межсоединений, блоки ОЗУ и ПЗУ, дешифраторы, устройства ввода-вывода, преобразователи питающих напряжений и др. Поэтому они получили более емкое название – программируемые логические интегральные схемы (ПЛИС).

На рисунке 10.1 показан внешний вид, а на рисунке 10.2 схематически приведена топологическая организация одной из последних разработок фирмы Altera – ПЛИС серии FLEX 10K.

Рис. 10.1.–Внешний вид ПЛИС FLEX 10K

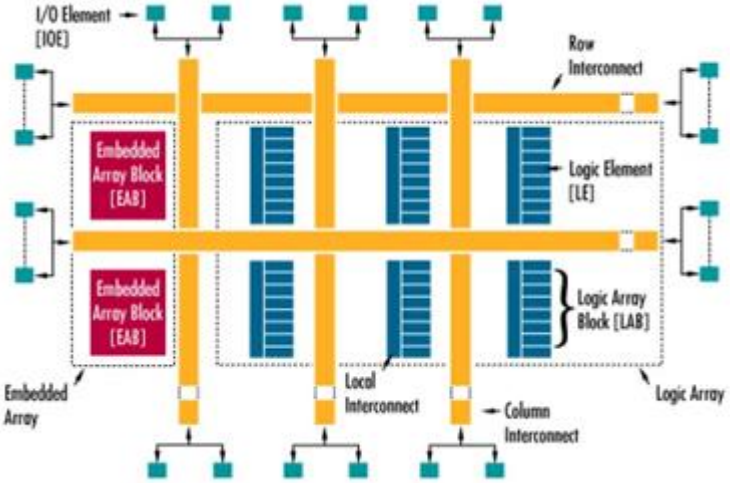


Рис. 10.2.–Топологическая организация ПЛИС серии FLEX 10K:
IOE – элемент ввода-вывода;
EAB -встроенный блок;
LE -логический элемент;
LAB -блок ПЛМ;
Row, Column, Local Interconnect – горизонтальные,
вертикальные и локальные линии связи

В табл. 10.1 приведены основные рабочие характеристики некоторых ПЛИС фирмы Altera (серий FLEX 10K и APEX 20K).

Таблица 10.1 – Характеристики ПЛИС фирмы Altera

Параметр	FLEX 10K	FLEX 10KA	FLEX 10KE	APEX 20K	APEX 20KE
Проектная норма, мкм	0,5	0,35	0,25	0,22	0,18
Напряжение питания, В	5	3,3	2,5	2,5	1,8
Макс. Число транзисторов	250 000	250 000	250 000	1 052 000	2 670 000
Макс. число логических элементов	12 160	12 160	12 160	16 640	42 240
Макс. число внешних выводов	-	-	-	502	756

В табл. 10.2 приведены данные по максимальному быстродействию ПЛИС конкурирующих фирм – мировых лидеров в данной области.

Таблица 10.2 – Максимальное быстродействие ПЛИС, производимых фирмами Altera, Xilinx, Lucent

Параметр	Altera FLEX 10K	Xilinx XC4000E-2	Lucent OR2C15A-4S208
Максимальная рабочая частота, МГц	125	52	89

Разработка специализированной СБИС на основе ПЛИС невозможна без систем автоматизированного проектирования (САПР), опирающихся не только на персональные компьютеры, но и на более мощную вычислительную базу – рабочие станции.

Предлагаемый курс лабораторных работ посвящен изучению подсистемы автоматизированного проектирования специализированных СБИС на основе ПЛИС MAX+plus II, являющейся составной частью одной из самых современных САПР для рабочих станций – Mentor Graphics.

10.1. Общие сведения о системе САПР Max Plus II.

Процедуру разработки нового проекта от концепции до завершения можно упрощенно представить следующим образом:

- создание нового файла или иерархической структуры нескольких файлов проекта с помощью любого сочетания редакторов в системе MAX+plus II, то есть графического, текстового и сигнального редакторов;
- задание имени файла œ проекта верхнего уровня в качестве имени проекта;
- назначение семейства ПЛИС для проекта;
- открытие окна компилятора Compiler и выбор кнопки Start для начала компиляции проекта. По желанию пользователя можно подключить модуль извлечения временных параметров проекта Timing SNF Extractor для создания файла, используемого при временном моделировании;
- в случае успешной компиляции возможен временной анализ, для чего следует выполнить следующее: для проведения анализа задержек открыть окно Timing Analyzer, выбрать режим анализа и нажать кнопку Start; для проведения моделирования нужно сначала создать текстовый вектор в файле (.scf), пользуясь сигнальным редактором, или в файле вектора (.vec), пользуясь текстовым редактором. Затем открыть окно отладчика Simulator и нажать кнопку Start;
- открытие окна программатора Programmer с последующим выбором одного из двух способов: использование программатора Master Programming

Unit (MPU) или подключение загрузочных устройств BitBlaster, ByteBlaster или FLEX Download Cable к устройству, программируемому в системе;

•выбор кнопки Program для программирования устройств с памятью типа EPROM или EEPROM (электрически перепрограммируемых ПЗУ) либо выбор кнопки Configure для конфигурации устройства с памятью типа SRAM (статического ОЗУ).

Система MAX+plus II содержит 11 приложений и главную управляющую программу. Различные приложения, обеспечивающие создание проекта, могут быть активизированы мгновенно, что позволяет пользователю переключаться между ними щелчком мыши или с помощью команд меню. В это же время может работать одно из фоновых приложений, например компилятор, программа моделирования, анализатор синхронизации и программатор. Одни и те же команды разных приложений работают одинаково, что облегчает задачу разработки логического дизайна.

Перед тем как начать работать в системе MAX+plus II, следует понять разницу между файлами проекта, вспомогательными файлами и проектами.

Файл проекта – это графический, текстовый или сигнальный файл, созданный с помощью графического или сигнального редакторов системы MAX+plus II или в любом другом, использующем промышленные стандарты, схемном или текстовом редакторе либо при помощи программы netlist writer, имеющейся в пакетах, поддерживающих стандартные форматы описания аппаратуры EDIF (Electronic Design Interchange Format), VHDL (Very High Speed Integrated Circuit Hardware Description Language) и Verilog HDL. Этот файл содержит функциональное описание проекта MAX+plus II и обрабатывается компилятором.

Вспомогательные файлы – это файлы, связанные с проектом MAX+plus II, но не являющиеся частью его иерархического дерева. Большинство таких файлов не содержит функционального описания проекта. Некоторые из них создаются автоматически приложением системы MAX+plus II, другие – пользователем. Примерами вспомогательных файлов являются файлы назначений и конфигурации (.acf), символьные файлы (.sym), файлы отчета (.rpt) и файлы тестовых векторов (.vec).

Проект состоит из всех файлов иерархической структуры проекта, в том числе вспомогательных и выходных файлов. Именем проекта является имя файла проекта верхнего уровня без расширения. Система MAX+plus II выполняет компиляцию, тестирование, анализ синхронизации и программирование сразу целого проекта, хотя пользователь может в это время редактировать файлы этого проекта в рамках другого проекта. Для каждого проекта желательно создавать отдельный подкаталог в рабочем каталоге системы MAX+plus II.

В системе MAX+plus II легко доступны все инструменты для создания

проекта. Разработка проекта ускоряется за счёт имеющихся стандартных функций, в том числе примитивов, мегафункций, библиотеки параметризованных модулей (LPM) и макрофункций устаревшего типа микросхем 74-й серии.

В системе MAX+plus II есть три редактора для разработки проекта: графический, текстовый и сигнальный, а также два вспомогательных редактора: редактор базового плана кристалла и символьный редактор.

В иерархической структуре проекта на любом уровне допускается смешанное использование файлов, созданных в различных редакторах системы MAX+plus II.

Редакторы системы Max Plus II.

Система MAX+plus II включает в себя четыре редактора создания проекта:

- графический редактор (Graphic Editor);
- символьный редактор (Symbol Editor);
- текстовый редактор (Text Editor);
- сигнальный редактор (Waveform Editor).

Данные редакторы имеют следующие общие функции:

- создание, сохранение и открытие файла;
- вывод данных на печать;
- поиск узлов;
- поиск и замена текста;
- отмена последнего шага редактирования, его возвращения;
- удаление, копирование, вставка выделенных фрагментов;
- обмен фрагментами между приложениями системы MAX+plus II или приложениями Windows;
- всплывающие окна меню.

Графический редактор позволяет создавать проекты на основе библиотек УГО (рисунок 10.3). Созданные в данном редакторе проекты могут содержать любую комбинацию УГО, мегафункции и макрофункции.

Графический редактор поддерживает следующие форматы файлов: .gdf и .sch.

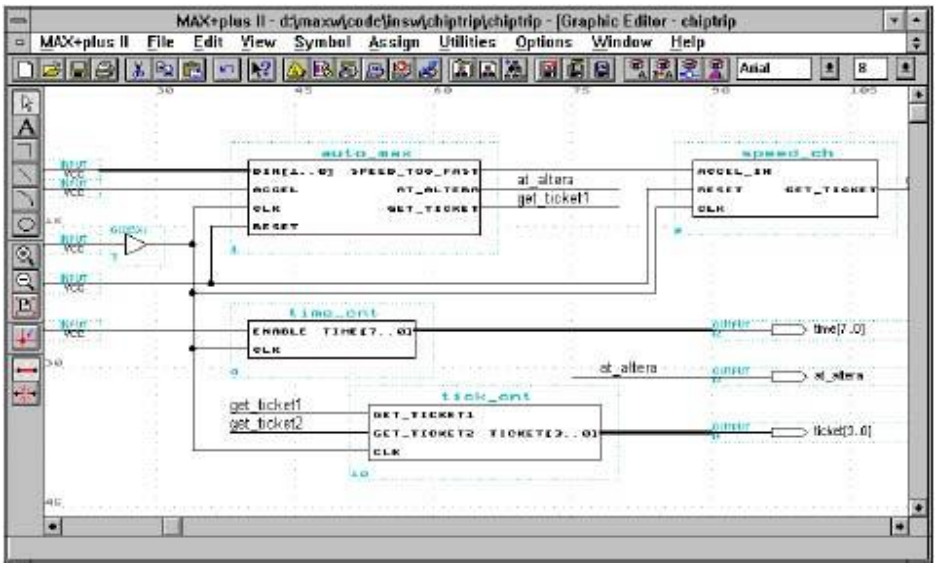


Рис.10.3.– Рабочее окно графического редактора

Графический редактор позволяет увеличивать или уменьшать масштаб изображения проекта на экране, выбирать размер шрифта, задавать стили линий, получать зеркальное отображение, поворачивать выделенные фрагменты на 90, 180 и 270 градусов, задавать размер и ориентацию текущего листа схемы.

Символьный редактор позволяет создавать и редактировать УГО (символ). Символьный файл имеет то же имя, что и проект, с расширением .sym.

Вызов редактора осуществляется по команде Create Default Symbol в меню File (рисунок 10.4).

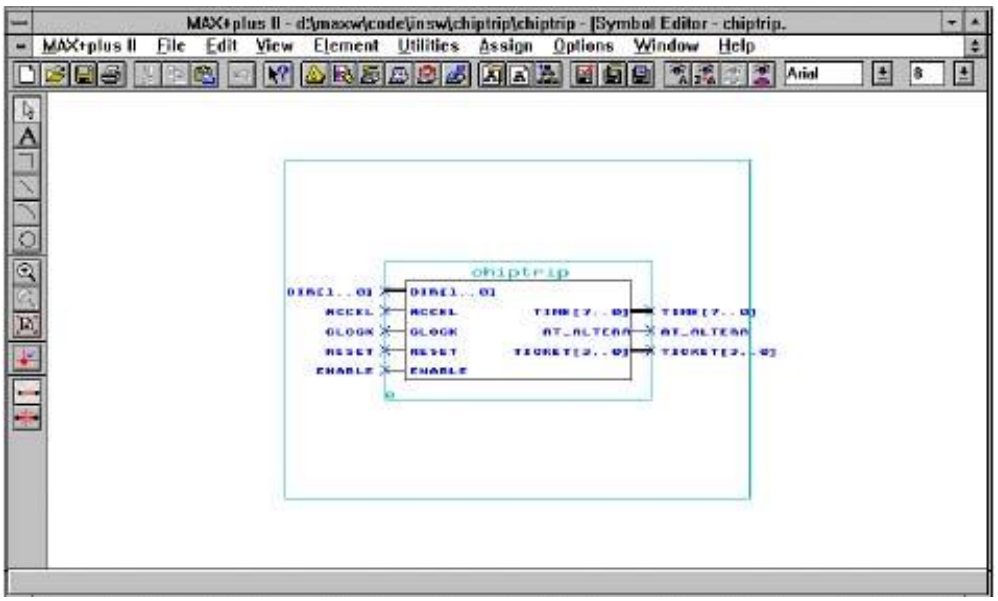


Рис.10.4.– Рабочее окно символьного редактора

Символьный редактор позволяет переопределять символ,

представляющий файл проекта, создавать и редактировать выводы и их имена, используя входные, выходные и двунаправленные выводы, а также задавать значения параметров и устанавливать их по умолчанию, вводить комментарии.

Текстовый редактор позволяет создавать текстовые файлы проекта на языках описания аппаратуры: AHDL (Altera Hardware Description Language) – .tdf, VHDL – .vhd, Verilog HDL – .v. При этом ввод исходных данных о проекте осуществляется не в виде схемы, выполненной в графическом редакторе (см. рис. 10.3), а в виде текстового описания проекта, что позволяет приблизить разработку проекта к процессу программирования, повысить наглядность представления информации, сократить вероятность ошибок и сроки проектирования.

В текстовом редакторе также можно работать со следующими форматами файлов: .acf, .aco, .adf, .cmd, .edc, .edf, .fit, .hst, .lmf, .log, .mif,.mio, .mtf, .plf, .rpt, .sdo, .smf, .tao, .tdf, .tdo, .tdx, .ttf, .vec, .vho, .vmo, .vo, .xnf и с произвольным файлом формата ASCII.

Данный текстовый редактор имеет следующие встроенные функции:

- ввод файла проекта;
- их компиляции и отладки с выдачей сообщения об ошибках и их локализацией в исходном тексте или в тексте вспомогательных файлов.

Кроме того, данный редактор содержит шаблоны языковых конструкций для AHDL, VHDL и Verilog HDL. В текстовом редакторе можно редактировать файлы конфигурации, а также делать установки для компилятора, программы моделирования и временного анализатора (рисунок 10.5).

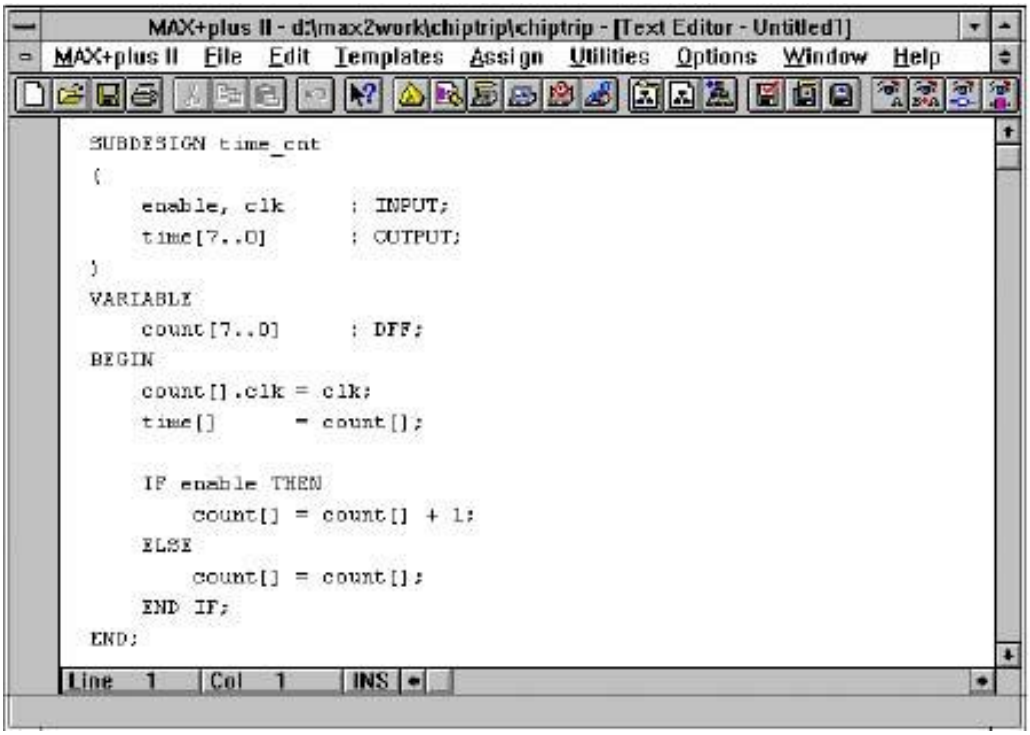


Рис.10.5.– Рабочее окно текстового редактора

Сигнальный редактор позволяет создавать описание проекта, вводить тестовые векторы и просматривать результаты тестирования (рисунок 10.6). Данный редактор поддерживает формат файлов сигнальных проектов .wdf, содержащих временные диаграммы, а также формат файлов тестирования .scf, содержащих входные векторы для функциональной отладки.

Сигнальный редактор является альтернативой графического или текстового редакторов. С его помощью можно графическим способом задавать комбинации входных и выходных логических уровней.

Созданный файл формата .wdf содержит логические входы, выходы комбинаторной логики, счетчиков и т.д.

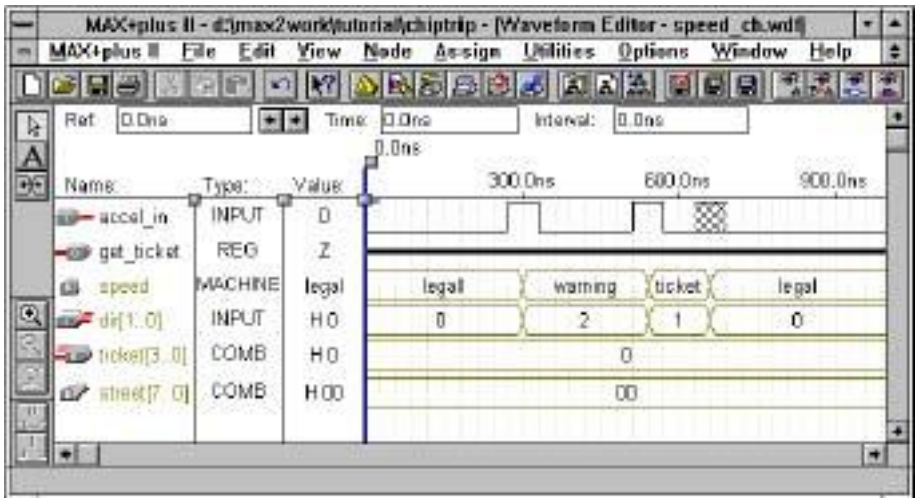


Рис.10.6.–Рабочее окно сигнального редактора

Сигнальный редактор системы MAX+plus II позволяет выполнять следующие встроенные функции:

- редактировать временные диаграммы целиком или частично;
- создавать и редактировать временные диаграммы отдельных узлов.
- позволяет создавать группы, содержащие от 2 до 256 узлов, и объединять их с уже существующими;
- отображать значение группы в двоичной, десятичной, шестнадцатеричной или восьмеричной системе счисления с преобразованием или без в код Грея;
- копировать, вставлять, удалять и перемещать выбранную часть или весь сигнал как в пределах одной группы, так и между группами.
- можно инвертировать, вставлять, переписывать, повторять, расширять или сжимать интервал сигнала с любым логическим уровнем, тактовым сигналом, последовательностью счета или именем состояния;
- задавать сетку для выравнивания переходов между логическими уровнями;
- вводить комментарии в любом месте файла проекта;
- изменять масштаб отображения проекта.
- для облегчения тестирования можно делать наложение любых выводов как одного файла, так и нескольких файлов для сравнения сигналов.

Для отображения иерархической структуры файлов проекта в виде дерева с ветвями, представляющими собой подпроекты, система MAX+plus II содержит дисплей иерархии (Hierarchy Display). В иерархии текущего проекта отображается имя и иконка файла для каждого подпроекта. Кроме того, дисплей иерархии показывает вспомогательные файлы, связанные с текущей иерархией (рисунок 10.7).

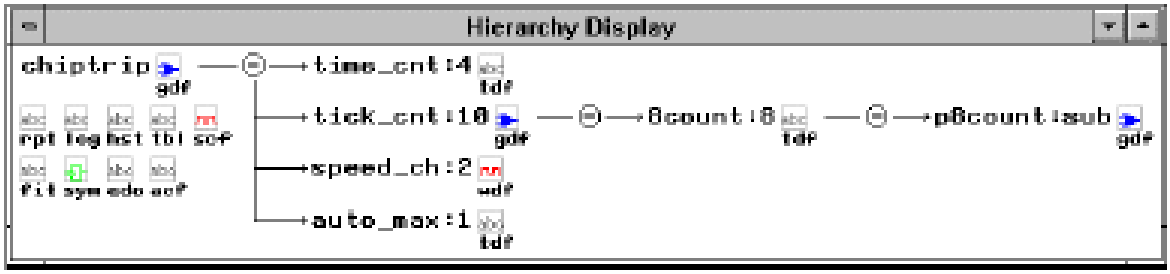


Рис.10.7.– Дисплей иерархии

Для назначения ресурсов физических устройств и просмотра результатов разводки, выполненных компилятором, система MAX+plus II содержит ре-дактор базового плана кристалла (Floorplan Editor) (рисунок 10.8).

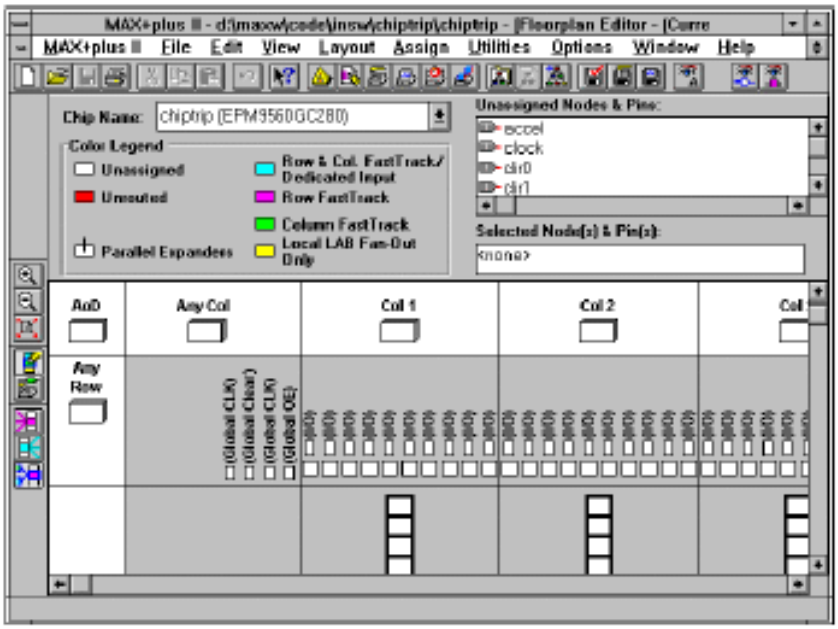


Рис.10.8.– Рабочее окно редактора базового плана кристалла

Редактор базового плана кристалла может представлять два типа изображения:

- Device View, показывающий все контакты устройства и их функции;
- LAB View, показывающий внутреннюю часть устройства, в том числе, все логические структурные блоки и отдельные логические элементы.

Для компиляции созданных проектов система MAX+plus II содержит компилятор (Compiler) (рисунок 10.9).

При компиляции проекта извлекается информация об иерархических связях между файлами проекта и производится проверка на ошибки ввода описания проекта. Если проект слишком большой для реализации в одном устройстве, компилятор автоматически разбивает его на части для реализации в нескольких устройствах того же самого семейства, при этом число соединений между устройствами минимизируется. Способ реализации проекта отражается в файле отчета компилятора .rpf.

Компилятор может автоматически обрабатывать следующие файлы проекта: графические файлы проекта (.gdf); текстовые файлы проекта на языке AHDL (.tdf); сигнальные файлы проекта (.wdf); файлы проекта на языке VHDL (.vhd); файлы проекта на языке Verilog (.v); схемные файлы OrCAD (.sch); входные файлы EDIF (.edf); файлы формата Xilinx Netlist (.xnf); файлы проекта Altera (.adf); файлы цифрового автомата (.smf).

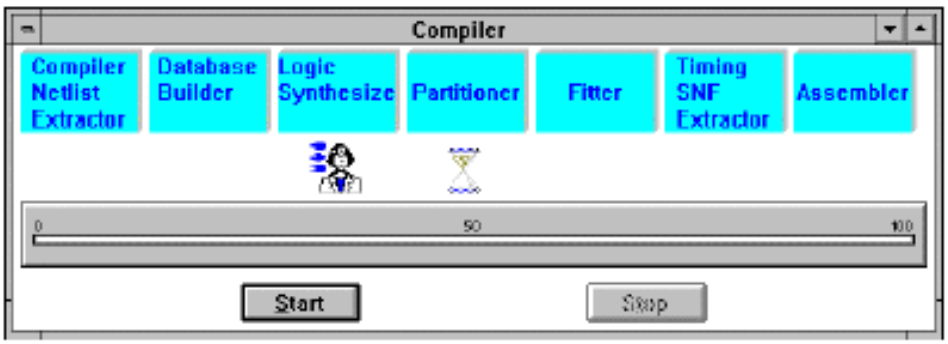


Рис.10.9.– Рабочее окно компилятора

Компилятор системы MAX+plus II позволяет создавать файлы, используемые программатором одного или нескольких устройств для программирования ПЛИС.

Кроме того, разработчик проекта может сам задавать стиль логического синтеза проекта по умолчанию и другие параметры логического синтеза в рамках всего проекта, что позволяет производить логический синтез в соответствии с потребностями разработчика. Кроме того, разработчик может ввести требования по синхронизации в рамках всего проекта, точно задать разбиение большого проекта на части для реализации в нескольких устройствах и выбрать варианты параметров применяемых устройств.

Загрузка готового проекта в ПЛИС в системе MAX+plus II выполняется с помощью программатора (Programmer), который позволяет программировать, конфигурировать, проводить верификацию и тестировать ПЛИС фирмы Altera (рисунок 10.10).

Для тестирования логических операций и внутренней синхронизации проекта в системе MAX+plus II содержится программа моделирования (Simulator) (рисунок10.11).

Она позволяет разработчику моделировать проект прежде, чем он будет реализован в устройстве, что существенно сокращает время разработки проекта. Кроме того, разработчик может производить моделирование проекта независимо от количества используемых устройств, требуемых для его реализации.

Для анализа исполнения проекта в системе MAX+plus II используется временной анализатор (Timing Analyzer) (рисунок 10.12). Данный анализатор позволяет разработчику анализировать работу проектируемой логической цепи после того, как она была синтезирована и оптимизирована компилятором. Разработчик может исследовать все пути прохождения сигналов в проектируемой логической цепи, определить критические задержки в цепях.

Рис.10.10.– Рабочее окно программатора

Для выдачи на экран сообщений об ошибках, предупреждающих и информационных сообщений, в системе MAX+plus II используется генератор сообщений (Message Processor). Генератор сообщений взаимодействует со всеми приложениями системы MAX+plus II, что позволяет определять и корректировать ошибки, получать предупреждающие сообщения и подсказки в любой момент времени (рисунок 10.13).

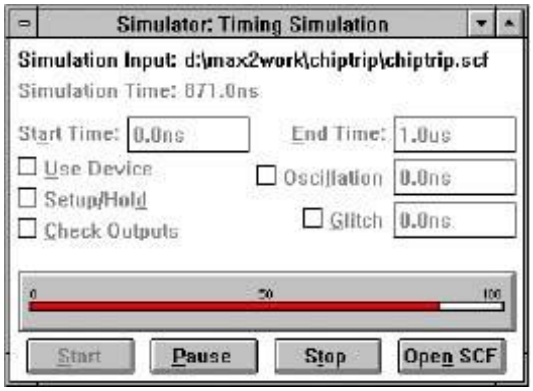


Рис.10.11.– Рабочее окно программы моделирования

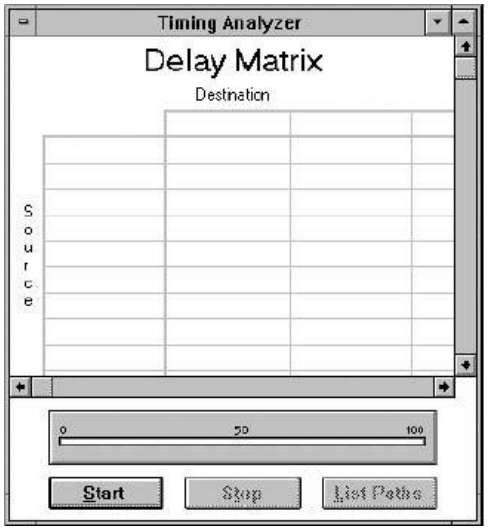


Рис.10.12.–Рабочее окно временного анализатора

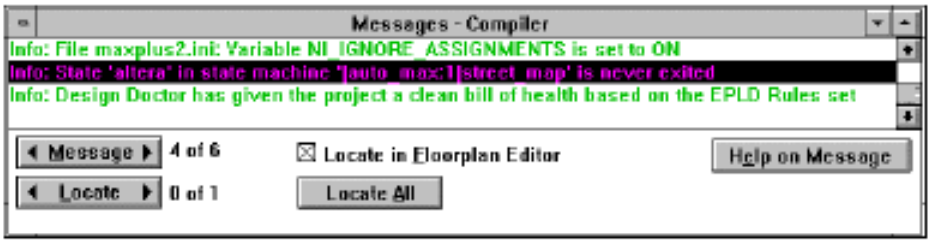


Рис.10.13.– Рабочее окно генератора сообщений

Компиляция проекта в системе MAX+plus II

Система MAX+plus II позволяет вводить, редактировать и удалять типы назначений ресурсов, устройств и параметров управления компиляцией проекта с помощью команд из меню Assign (рисунок 10.14).

Рис.10.14.– Меню Assign

Разработчик может выполнять назначения для своего проекта

независимо от того, открыт ли какой-нибудь файл проекта или окно приложений. Доступны следующие типы назначений:

- Clique… – задает логические функции, которые должны оставаться вместе в одном и том же блоке логической структуры LAB, блоке ячеек EAB, в одном ряду или устройстве;
- Pin/Location/Chip… – назначает конкретному контакту или нескольким контактам кристалла вход или выход одной логической функции (назначает единственную логическую функцию конкретной ячейке кристалла) задает логические функции, которые должны быть реализованы в одном и том же устройстве в случае разделения проекта на несколько устройств.
- Probe… – присваивает уникальное имя входу или выходу логической функции;
- Connected Pins… – задает внешнее соединение двух или более контактов на схеме разработчика;
- Local Routing… – присваивает коэффициент разветвления по выходу логическому элементу;
- Device… – назначает тип ПЛИС, на которой реализуется проект;
- Logic Options… – управляет синтезом отдельных логических функций во время компиляции;
- Timing Requirements… – управляет синтезом и подгонкой отдельных логических функций для получения требуемых характеристик для времени задержки вход-неподрегистренный выход (tPD), синхросигнал-выход (tCO), синхросигнал-время установки (tSU) и для частоты синхросигнала (fMAX).

Можно определить глобальные настройки устройства для компилятора, с целью использования их для всех устройств при обработке проекта:

- Global Project Parameters – позволяет задавать имена и глобальные установки, которые могут быть использованы компилятором для параметров всех функций в проекте;
- Global Project Timing Requirements – позволяет ввести глобальные требования по синхронизации для проекта, задавая общие характеристики для времени задержки вход-неподрегистренный выход (tPD), синхросигнал-выход (tCO), синхросигнал-время установки (tSU) и для частоты синхросигнала (fMAX);
- Global Project Logic Synthesis – позволяет сделать глобальные установки для компилятора в части логического синтеза проекта;
- Global Project Device Options – позволяет определять настройки выбранного устройства для всех устройств, использованных в текущем проекте.

Пример проектирования специализированной ИС.

В качестве примера рассмотрим маршрут проектирования с помощью программы MAX+plus II ИС четырехразрядного сдвигового регистра (рисунок 10.15) на двухтактных D-триггерах (рисунок 10.16).

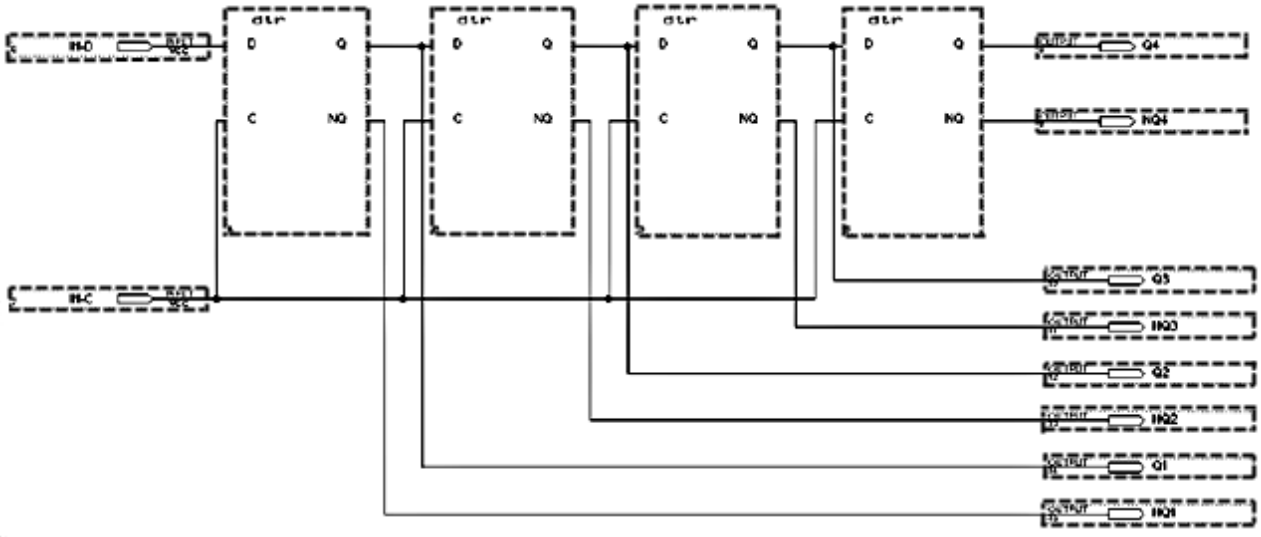


Рис.10.15.– Схема четырехразрядного сдвигового регистра

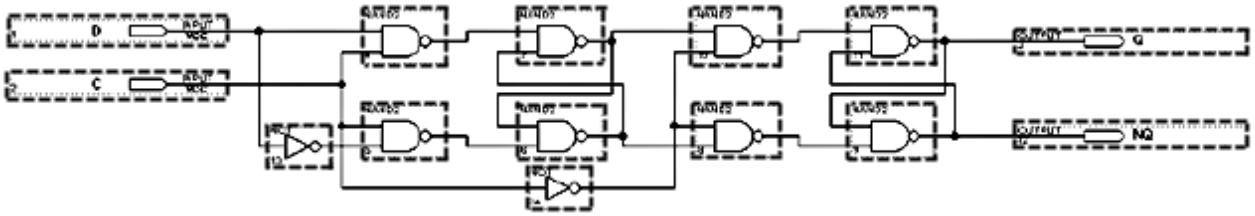


Рис.10.16.–Схема двухтактного D-триггера

Основные этапы проектирования:

- запуск программы MAX+plus II;
- ввод имени проекта;
- ввод электрической схемы проектируемого устройства;
- ввод входных сигналов для функционально-логического моделирования;
- выбор типа ПЛИС для реализации проекта;
- проверка корректности ввода электрической схемы;
- компиляция схемы проекта;
- функционально-логическое моделирование проектируемого устройства;
- расчет задержек прохождения сигналов со входов на выходы проектируемого устройства;
- просмотр базового плана кристалла ПЛИС с размещенными элементами схемы;
- сохранение файлов проекта;
- формирование файлов для программирования ПЛИС.

Для запуска программы MAX+plus II необходимо:

•ввести логическое имя пользователя и пароль, указанные администратором сети;

•запустить файл maxstart.exe в каталоге, указанном преподавателем.

Для ввода имени проекта следует щелкнуть левой кнопкой манипулятора «мышь» на пункте меню File\Project\Name… или нажать комбинацию клавиш Ctrl+J (рисунок 10.17).

В появившемся окне необходимо указать диск (Drives:) и каталог (Directories:) для размещения файлов проекта, а также ввести имя проекта (Project Name:) (например, sd_reg без расширения), после чего щелкнуть – “мышью” на кнопке ОК (рисунок 10.18) или нажать ENTER на клавиатуре.

Ввод электрической схемы проектируемого устройства необходимо начинать с первого уровня иерархии. Нижний (нулевой) уровень представлен логическими вентилями. В рассматриваемом примере требуется осуществить ввод схемы D-триггера (1-й уровень) на основе элементов 2И-НЕ, НЕ (0-й уровень) (см. рис. 10.16).

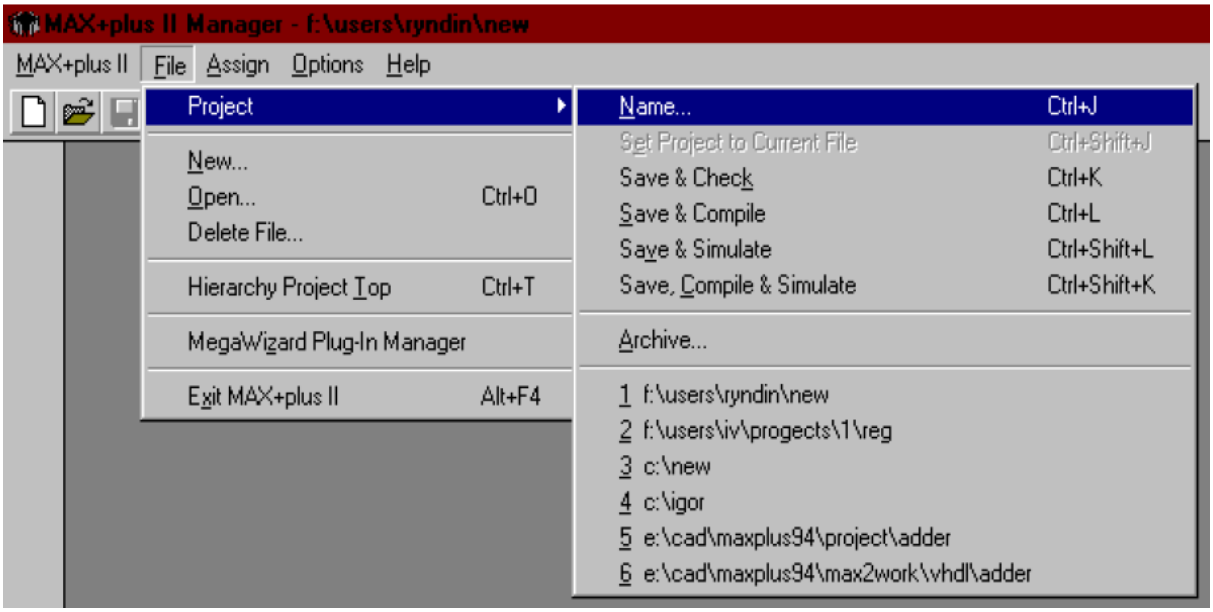


Рис.10.17.– Ввод имени проекта

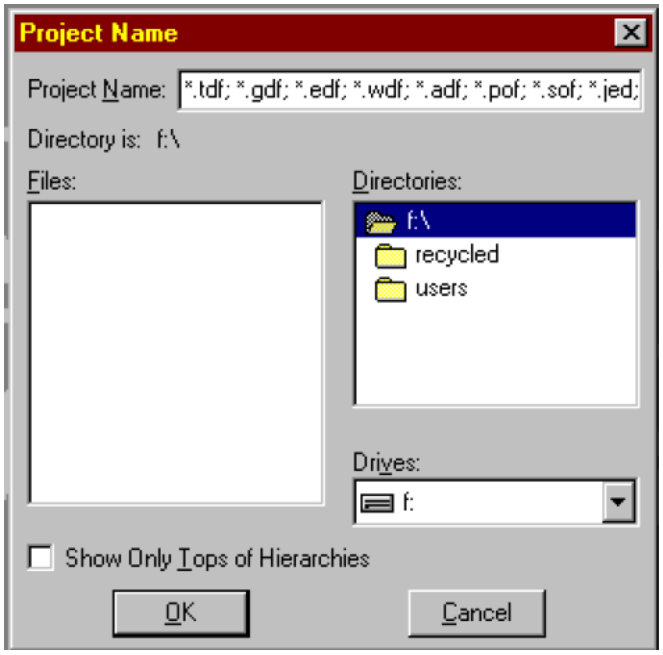


Рис.10.18.– Окно для ввода имени проекта

Для ввода схемы D-триггера необходимо:

1) войти в графический редактор, щелкнув левой кнопкой «мыши» на пункте меню MAX+plus II\Graphic Editor;

2) в появившемся пустом окне щелкнуть правой кнопкой «мыши» на любом свободном месте и в появившемся динамическом меню выбрать пункт Enter Symbol;

3) в появившемся диалоговом окне (рисунок 10.19) выбрать двойным щелчком левой кнопки «мыши» библиотеку maxplus2\max2lib\prim в списке под заголовком Symbol Libraries и библиотечный элемент nand2 (2И-НЕ) в списке Symbol Files. При этом в окне графического редактора появится условное изображение элемента 2И-НЕ;

4) навести курсор «мыши» на изображение элемента, нажать левую кнопку манипулятора и, удерживая ее нажатой, переместить изображение элемента в требуемую точку в соответствии со схемой (см. рисунок 10.16), после чего отпустить кнопку;

5) навести курсор «мыши» на изображение элемента, нажать Ctrl на клавиатуре и левую кнопку «мыши» и, удерживая их, скопировать изображение элемента в требуемую точку в соответствии со схемой (см. рисунок 10.16), после чего отпустить кнопки;

6) повторить пункт 5 для всех элементов 2И-НЕ D-триггера (см. рисунок 10.16);

7) повторить пункты 2 – 5 для элементов not (НЕ), input (Вход), output (Выход), разместив их в соответствии со схемой (см. рисунок 10.16);

8) дважды щелкнуть левой кнопкой «мыши» на заголовке PIN_NAME элемента input и ввести с клавиатуры имя входа (например, D);

9) повторить пункт 8 для всех входов (input) и выходов (output) схемы;

10) провести межэлементные соединения. Навести курсор «мыши» на изображение соответствующего контакта элемента, нажать левую кнопку и, удерживая ее, провести соединение. В месте изгиба соединительной линии следует отпускать левую кнопку «мыши», не сдвигая курсор, вновь нажимать ее, после чего вести линию далее в ином направлении. Проводить ветвящиеся линии рекомендуется от соединяемого внешнего вывода к узлу. При этом изображения узлов проставляются автоматически. При необходимости переместить отдельные участки линий или отдельные уже соединенные в схему элементы, следует навести курсор «мыши» на перемещаемый элемент изображения, нажать левую кнопку и, удерживая ее, осуществить перемещение. При этом все линии связи перемещаются соответствующим образом автоматически;

11) сохранить файл схемы D-триггера, щелкнув левой кнопкой «мыши» на пункте меню File\Save и введя с клавиатуры имя файла (например, dtr.gdf). Файл сохранится в каталоге проекта по умолчанию. Для ввода электрической схемы сдвигового регистра (второй уровень иерархии) на основе D-триггеров необходимо предварительно создать условное графическое изображение (Symbol) D-триггера.

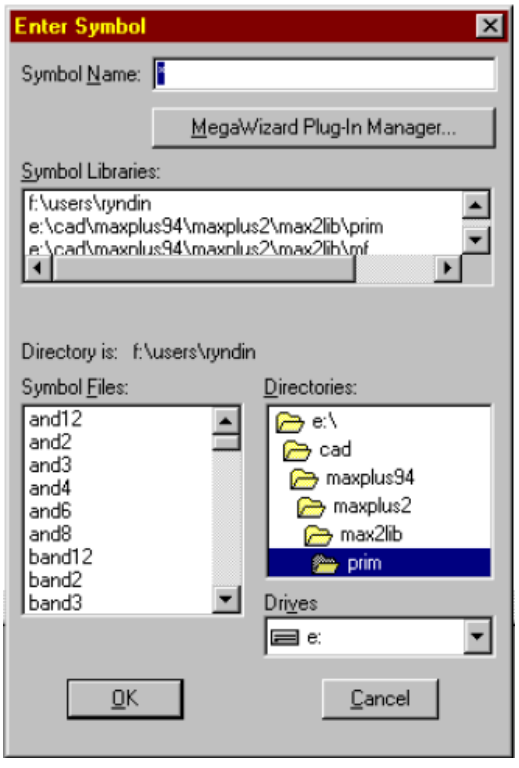


Рис.10.19.– Выбор библиотечного элемента

Для создания условного графического изображения необходимо:

1) войти в символьный редактор, щелкнув левой кнопкой «мыши» на пункте меню MAX+plus II\Symbol Editor;

2) в появившемся окне символьного редактора (рисунок 10.20) навести курсор «мыши» на левую границу графического изображения и дважды щелкнуть левой кнопкой;

3) в появившемся диалоговом окне (рисунок 10.21) ввести с клавиатуры имя входа (например, D) в поле Full Pinstub Name, щелкнуть левой кнопкой «мыши» на заголовке Input Pin в поле I/O Type и на заголовке Used в поле Default Status, установить флажок Show Visible Pinstub Name in Graphic Editor, после чего щелкнуть левой кнопкой «мыши» на ОК или нажать клавишу ENTER на клавиатуре;

4) навести курсор «мыши» на появившееся изображение входа в виде символа × (рисунок 10.22), нажать левую кнопку и, удерживая ее, переместить изображение входа в требуемую точку;

5) повторить предыдущие пункты для всех входов и выходов D-триггера, учитывая, что изображения выходов удобнее вводить, щелкая «мышью» в пункте 2 на правой границе условного графического изображения элемента. Имена входов и выходов должны совпадать с соответствующими именами на схеме, введенной в графическом редакторе (файл dtr.gdf);

6) при необходимости удалить изображение входа или выхода необходимо навести курсор «мыши» на удаляемое изображение, щелкнуть правой кнопкой и в появившемся динамическом меню щелкнуть левой кнопкой на пункте Delete;

7) сохранить файл условного графического изображения D-триггера, щелкнув левой кнопкой «мыши» на пункте меню File\Save и введя с клавиатуры имя файла (например, dtr.sym). Имена соответствующих файлов с расширениями .sym и .gdf должны совпадать. Файл сохранится в каталоге проекта по умолчанию.

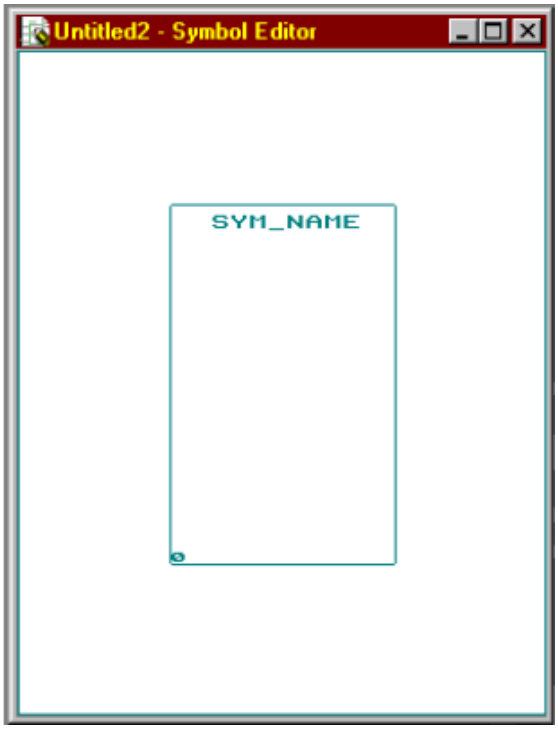


Рис.10.20.– Окно символьного редактора

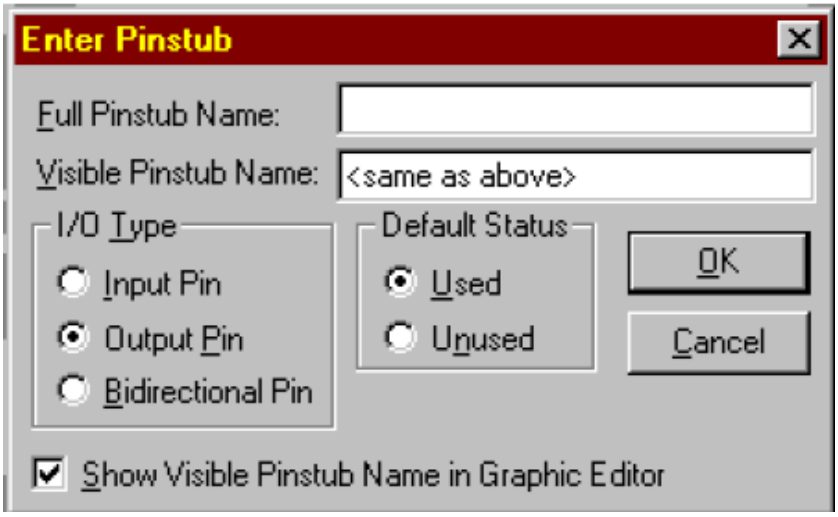


Рис.10.21.– Диалоговое окно символьного редактора

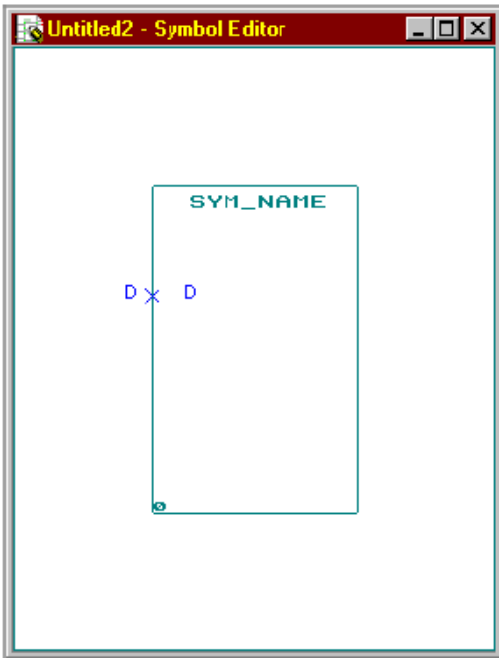


Рис.10.22.– Изображение входа в символьном просмотре

Ввод электрической схемы сдвигового регистра на основе D-триггеров (см. рисунок 10.15) осуществляется в графическом редакторе аналогично вводу схемы D-триггера. Отличия состоят в следующем:

1) в пункте 3 необходимо выбрать библиотечный элемент dtr в каталоге проекта;
2) в пунктах 8 и 9 имена входов и выходов регистра должны отличаться от имен входов и выходов на схеме триггера (в файле dtr.gdf);
3) в пункте 11 сохранить схему регистра в файле с именем проекта (в приведенном примере – sd_reg.gdf).

Для ввода входных сигналов необходимо:

1) войти в редактор сигналов, щелкнув левой кнопкой «мыши» на пункте меню MAX+plus II\Waveform Editor;

2) в появившемся окне редактора (рисунок 10.23) дважды щелкнуть левой кнопкой «мыши» на свободной строке (ниже заголовка Name);

3) в появившемся диалоговом окне щелкнуть левой кнопкой «мыши» на кнопке List;

4) щелкнуть левой кнопкой «мыши» на имени входа из списка, появившегося под заголовком Nodes and Gropes from SNF, после чего щелкнуть левой кнопкой «мыши» на ОК или нажать клавишу ENTER на клавиатуре;

5) повторить пункты 2 – 4 для всех входов и выходов схемы, которые предполагается использовать при моделировании;

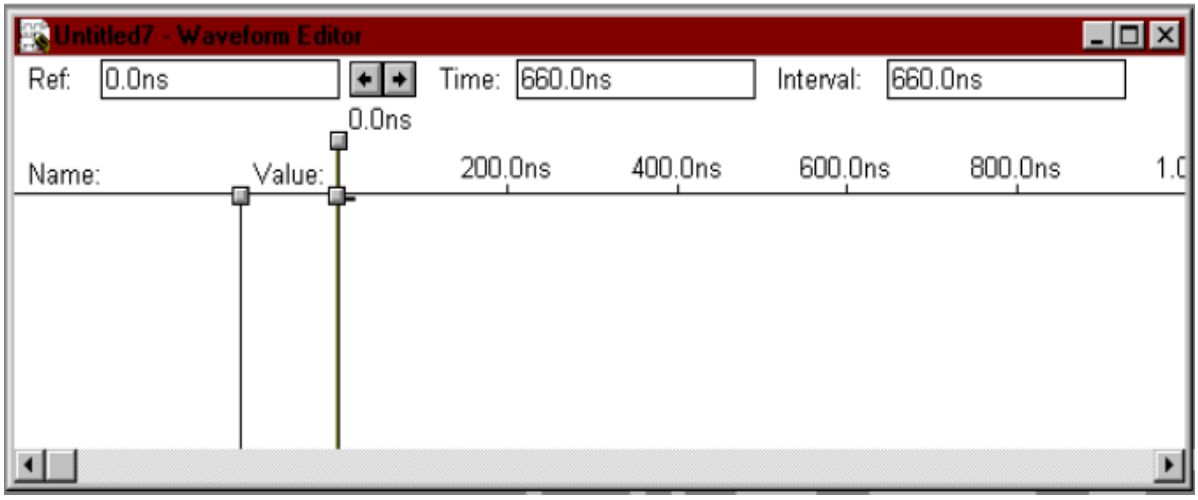


Рис.10.23.–Окно редактора сигналов

6) определить все входные сигналы, щелкая левой кнопкой «мыши» на строке с именем входного сигнала, а затем на одной из следующих кнопок:


- уменьшить масштаб изображения;


- увеличить масштаб изображения;


- установить низкий уровень логического нуля;


- установить высокий уровень логической единицы;


- инвертировать логический уровень;


- установить неопределенный логический уровень;


- установить высокоимпедансное Z-состояние;


- установить периодическое изменение логического уровня.

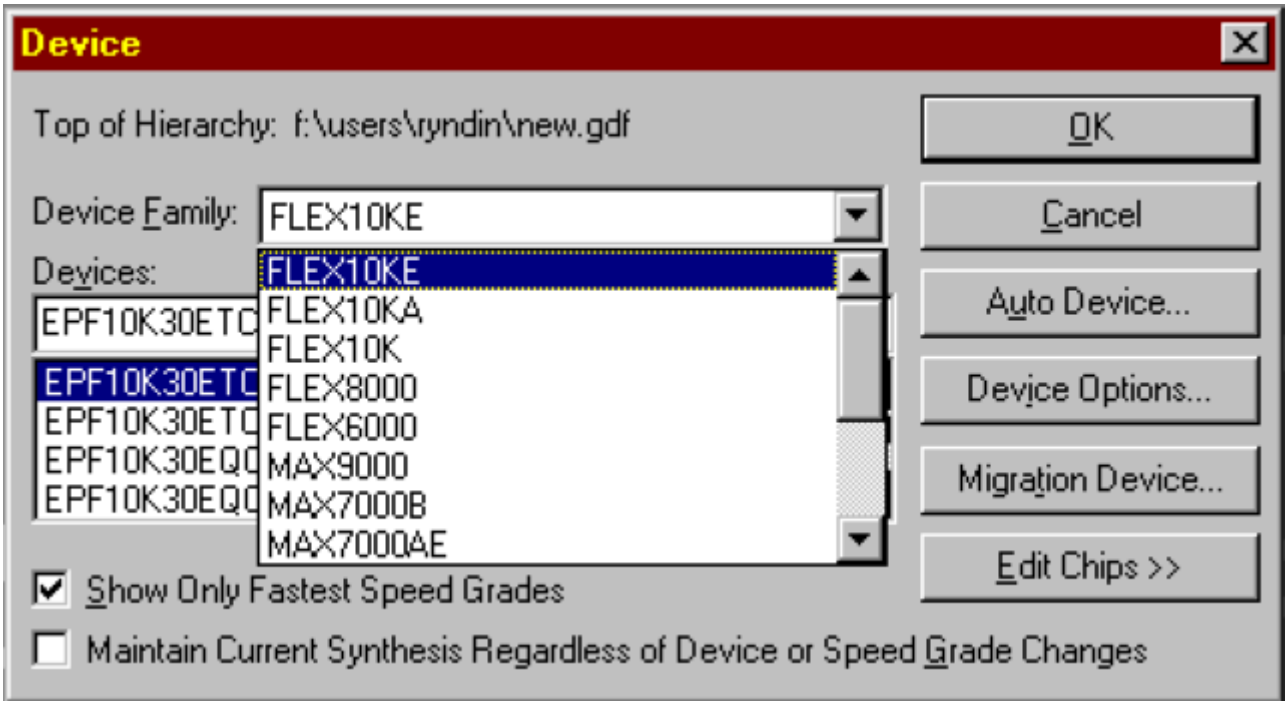


Рис.10.24.– Диалоговое окно выбора типа ПЛИС

Для проверки корректности ввода электрической схемы проекта, компиляции и функционально-логического моделирования необходимо щелкнуть левой кнопкой «мыши» на пункте меню File\Project\Save,Compile & Simulate или нажать комбинацию клавиш Ctrl+Shift+K.

При этом последовательно появятся следующие окна:

- окно компилятора с перечисленными в верхней части этапами компиляции (рисунок 10.25);
- окно сообщений компилятора об ошибках (Error) и замечаниях (Warning) (рисунок 10.26);
- окно подсистемы функционально-логического моделирования

(рисунок 10.27);

•окно сообщений подсистемы функционально-логического моделирования (рисунок 10.28).

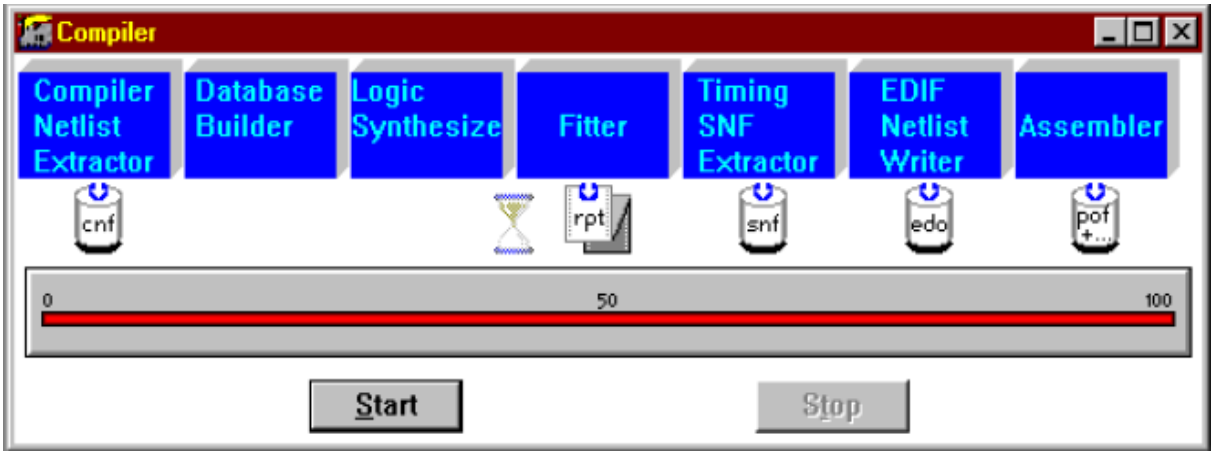


Рис.10.25.– Окно компилятора

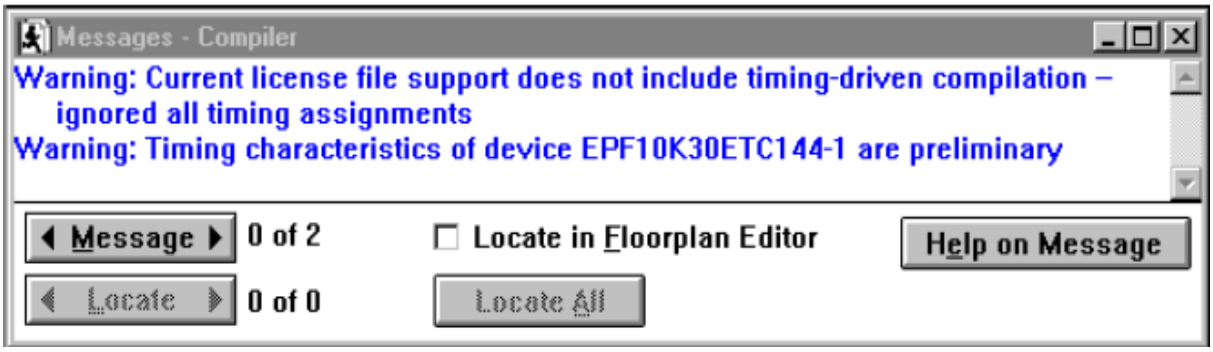


Рис.10.26.– Окно сообщений компилятора

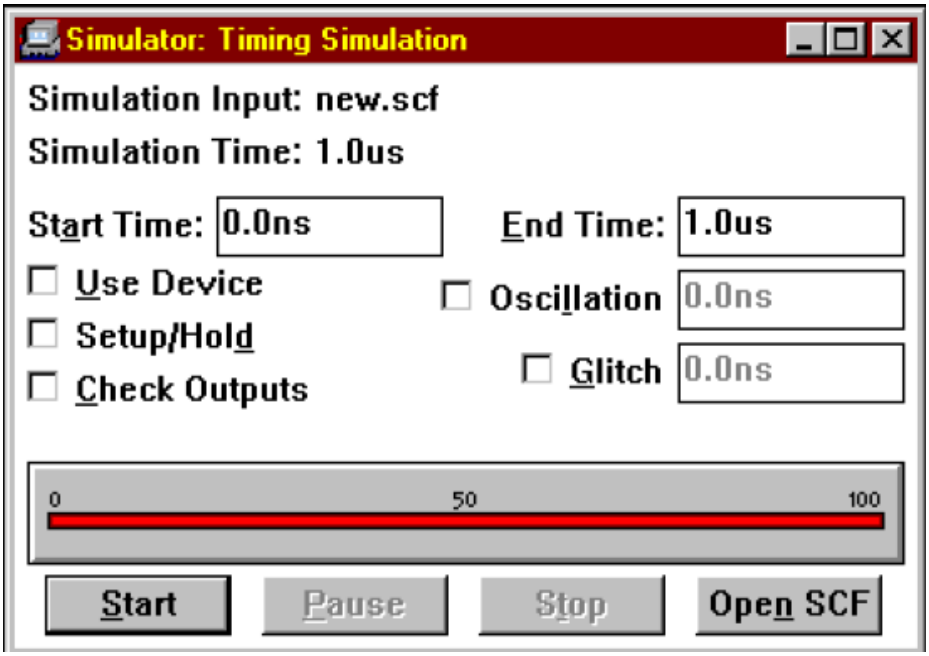


Рис.10.27.– Окно подсистемы функционально-логического моделирования

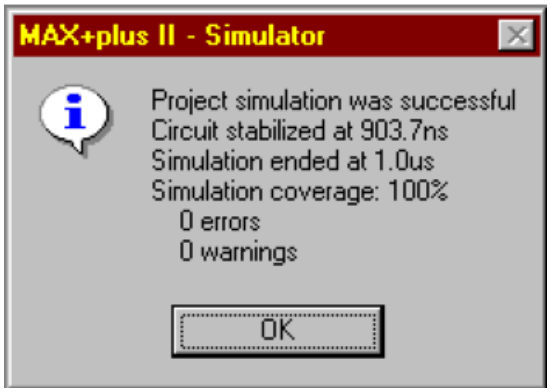


Рис.10.28.– Окно сообщений подсистемы функционально-логического моделирования

Для просмотра результатов моделирования необходимо щелкнуть левой кнопкой «мыши» на ОК в окне сообщений (см. рисунок 10.28) или нажать клавишу ENTER на клавиатуре, после чего щелкнуть левой кнопкой «мыши» на кнопке Open SCF в окне подсистемы моделирования (см. рисунок 10.27). При этом появится окно редактора сигналов с результатами моделирования (рисунок 10.29).

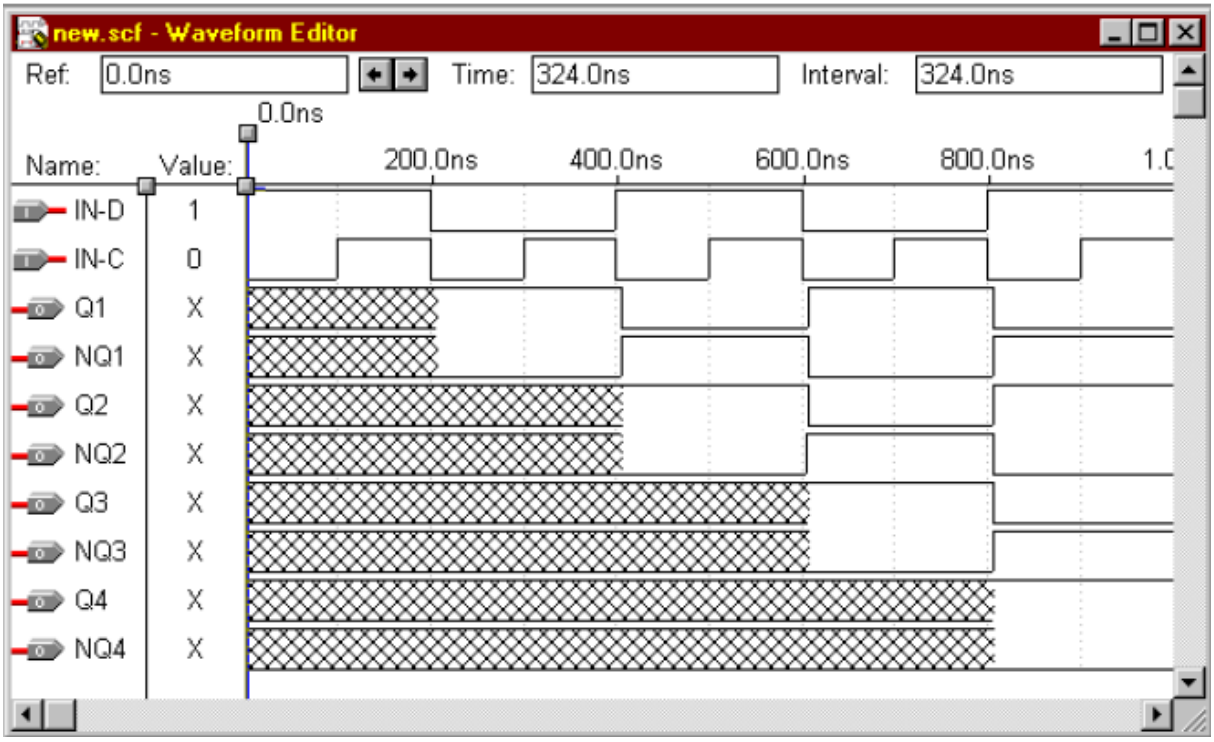


Рис.10.29.– Результаты функционально-логического моделирования

Для расчета задержек прохождения сигналов со входов на выходы проектируемого устройства следует щелкнуть левой кнопкой «мыши» на пункте меню MAX+plus II\Timing Analyzer. При этом появится таблица задержек сигналов (рисунок 10.30), а результаты автоматически сохранятся в текстовом файле с именем проекта и расширением .tao (в рассматриваемом

примере sd_reg.tao). Файл с таблицей задержек можно просмотреть любым текстовым редактором, в том числе и встроенным в MAX+plus II. Он вызывается щелчком левой кнопки «мыши» на пункте меню MAX+plus II\Text Editor.

Для просмотра базового плана кристалла ПЛИС с размещенными элементами схемы проекта следует щелкнуть левой кнопкой «мыши» на пункте меню MAX+plus II\Floorplan Editor. При этом появится окно базового плана ПЛИС (рисунок 10.31).

Для просмотра цоколевки внешних выводов ПЛИС необходимо щелкнуть левой кнопкой «мыши» на пункте меню Layout\Device View (рисунок 10.32).

Подготовка файлов для программирования ПЛИС в соответствии с разработанным проектом осуществляется автоматически после активизации левой кнопкой «мыши» пункта меню MAX+plus II\Programmer.

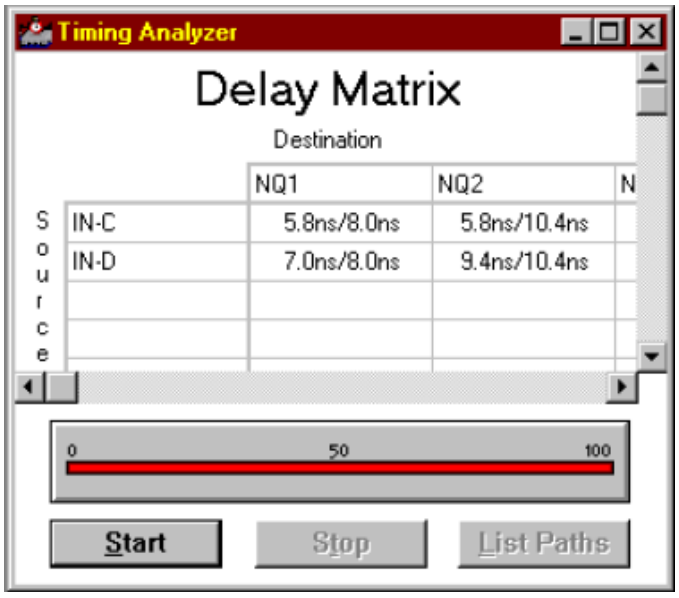


Рис.10.30.– Таблица задержек сигнала

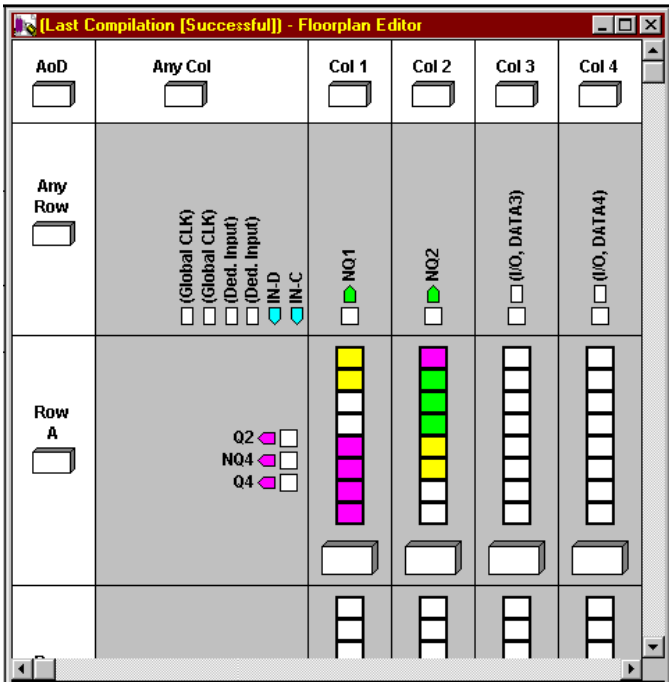


Рис.10.31.– Базовый план ПЛИС

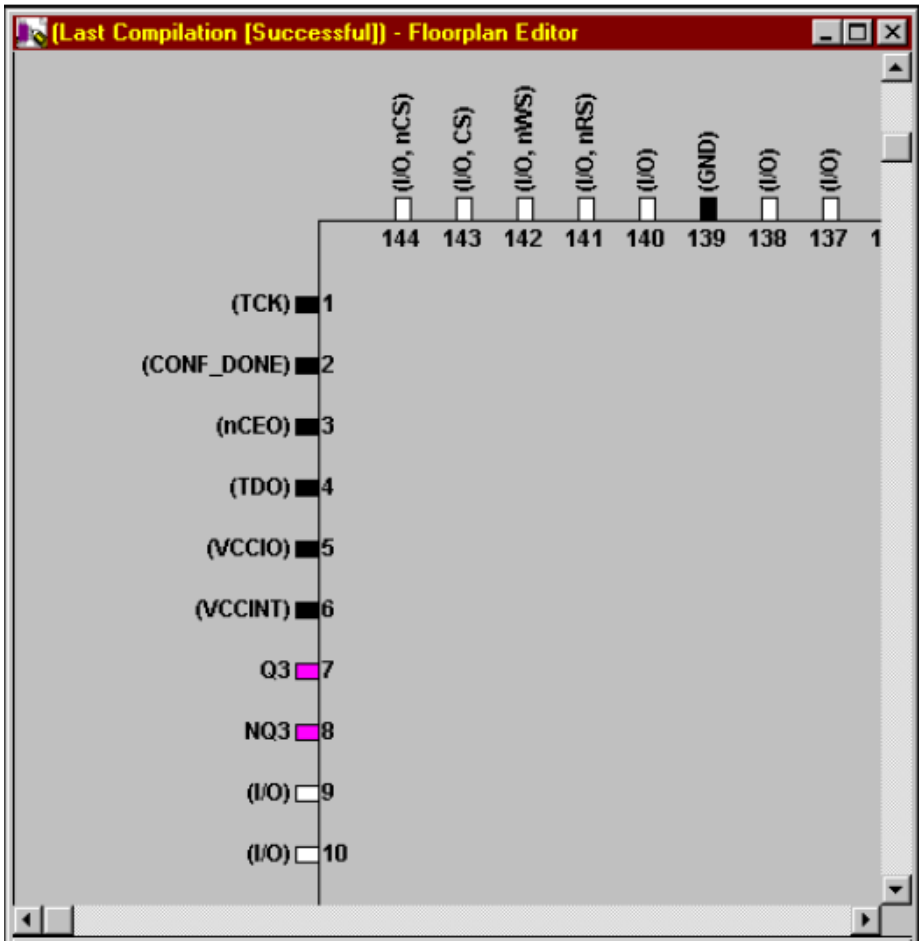


Рис.10.32.– Цоколевка ПЛИС

10.2. Лабораторная работа №1. Моделирование простейших устройств в пакете программ Max Plus II

Цель лабораторной работы: получить представление о системе автоматизированного проектирования Max Plus II.

Порядок выполнения лабораторной работы:

- ознакомиться со сведениями о системе Max Plus II, изложенными в пп.10.1;
- получить у преподавателя вариант задания к лабораторной работе;
- выполнить последовательно все этапы проектирования, изложенные в пп.10.1;
- результаты проектирования показать преподавателю;
- приступить к проектированию заданного преподавателем устройства;

Отчет по лабораторной работе сдается преподавателю в распечатанном с одной стороны листа на формате А4 виде.

Содержание отчета по лабораторной работе №1:

1. Титульный лист установленного образца (Приложение №1).
2. Содержание.
3. Схема электрическая принципиальная заданного устройства (согласно варианту) с объяснением на вентильно-регистровом уровне.
4. Временная диаграмма работы устройства с описанием на уровне входных и выходных сигналов.
5. Выводы с указанием на результаты проектирования.
6. Файлы проекта в Max Plus II на носителе информации.

Варианты заданий к лабораторной работе №1:

Вариант №1.

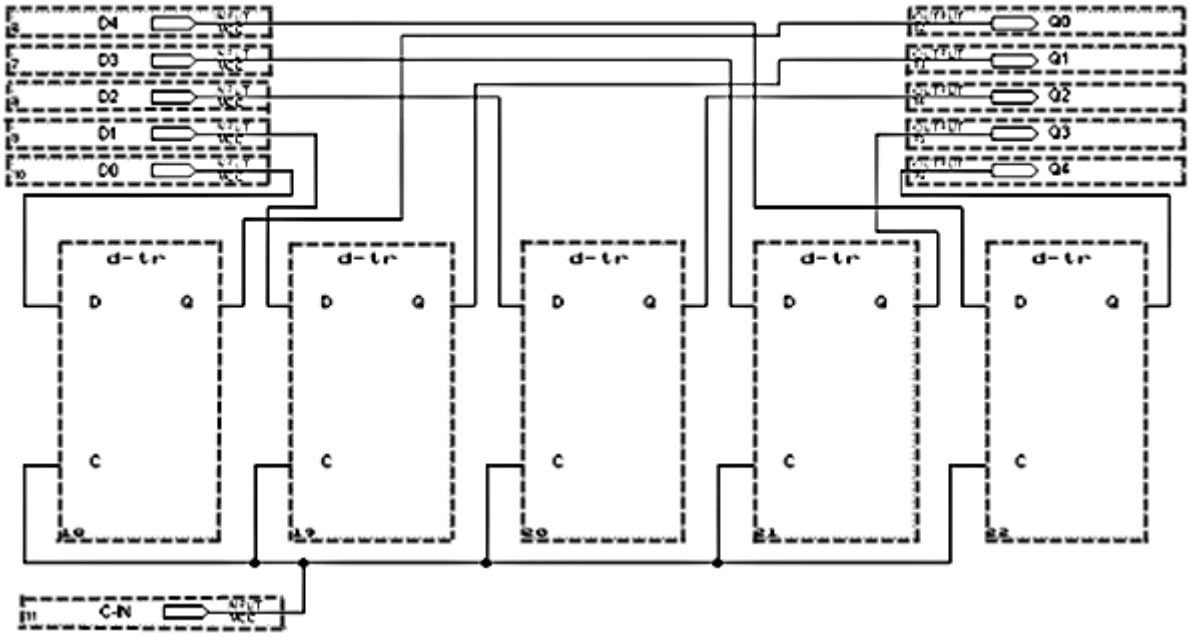


Рис.10.33.–Регистр

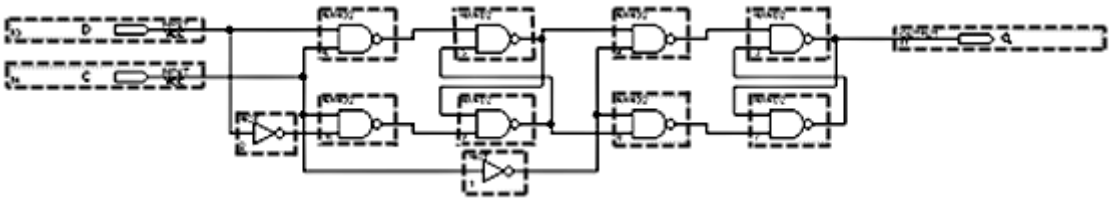


Рис.10.34.–D-триггер

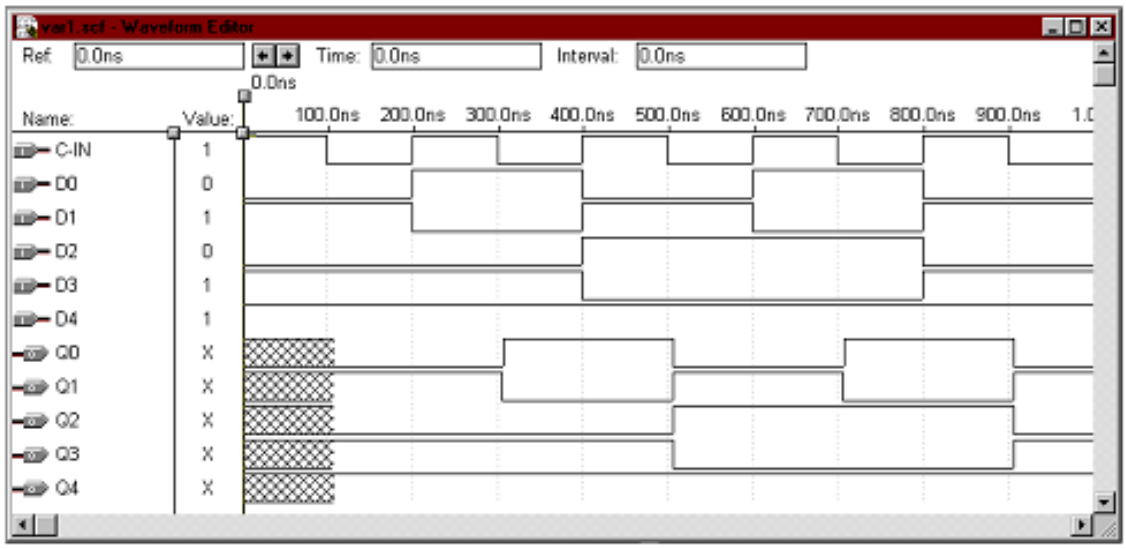


Рис.10.35.–Результаты моделирования регистра

Вариант №2.

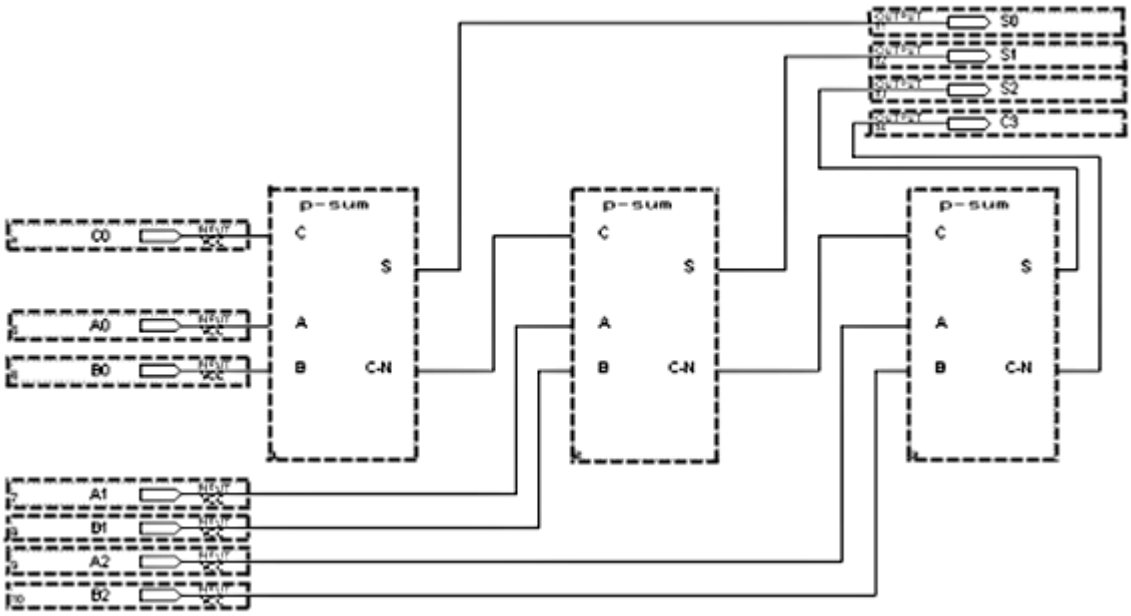


Рис.10.36.– Трехразрядный сумматор на элементах исключающее ИЛИ, И-НЕ

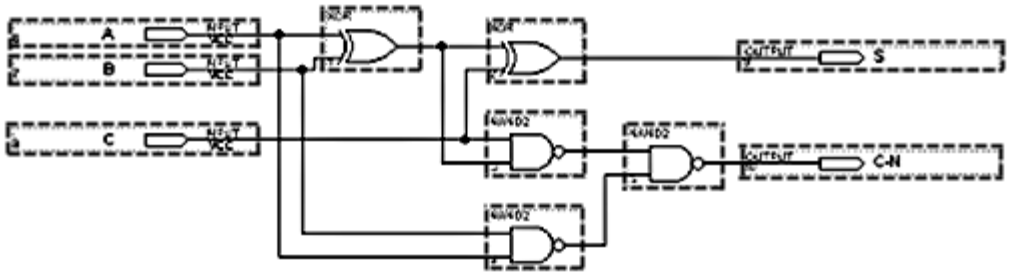


Рис.10.37.– Сумматор на элементах исключающее ИЛИ, И-НЕ

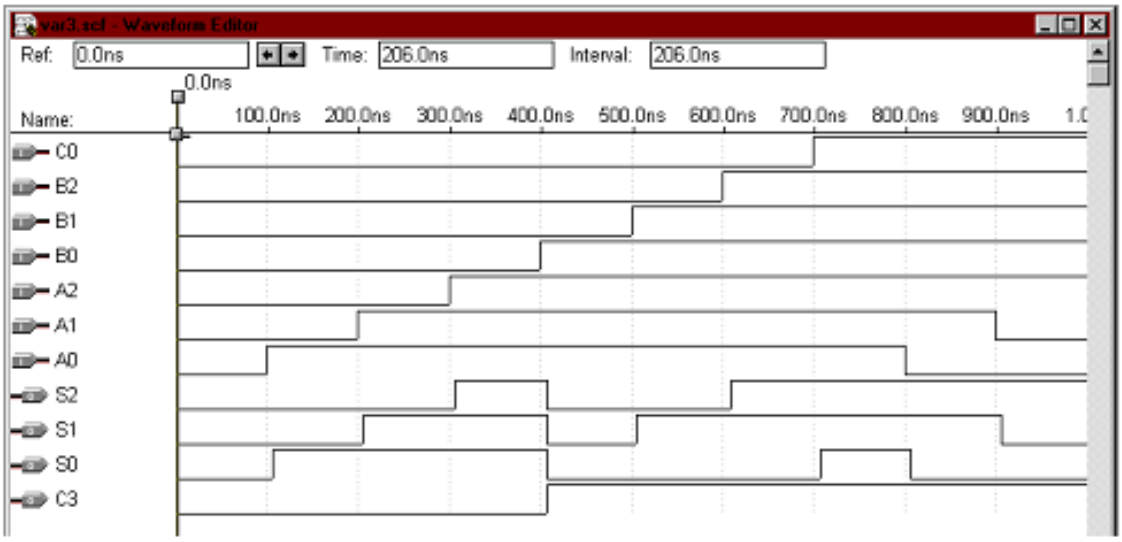


Рис.10.38.– Результаты моделирования трехразрядного сумматора

Вариант №3.

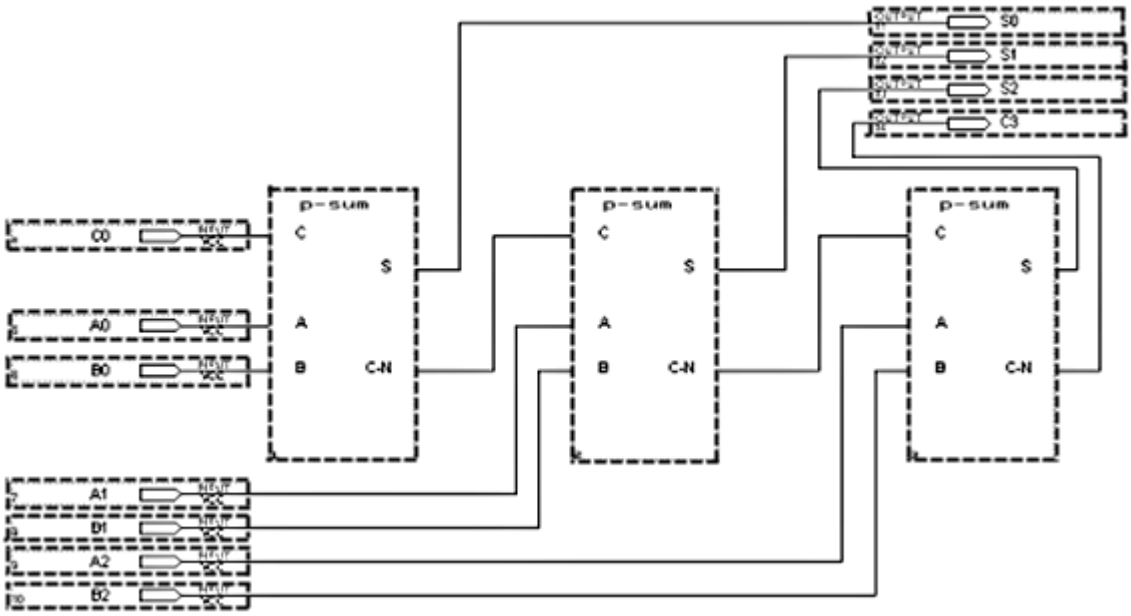


Рис.10.39.– Трехразрядный сумматор на элементах И-НЕ

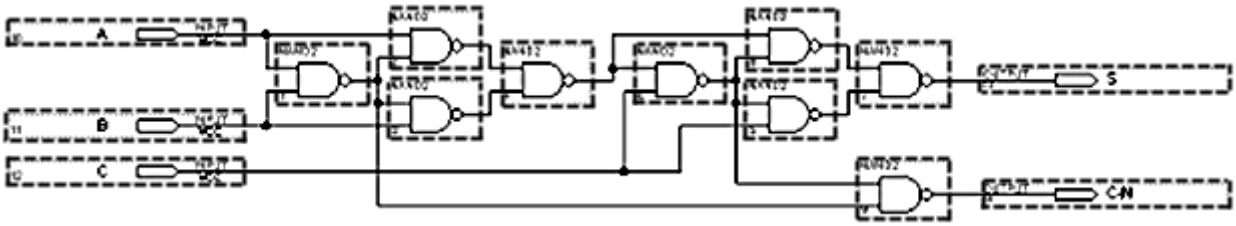


Рис. 10.40.– Сумматор на элементах И-НЕ

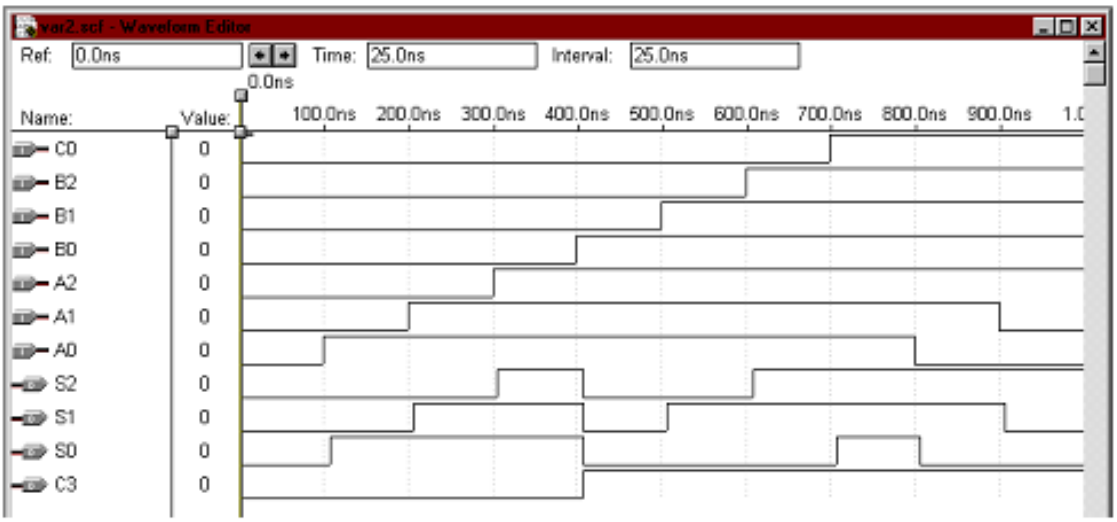


Рис.10.41.– Результаты моделирования сумматора

Вариант №4.

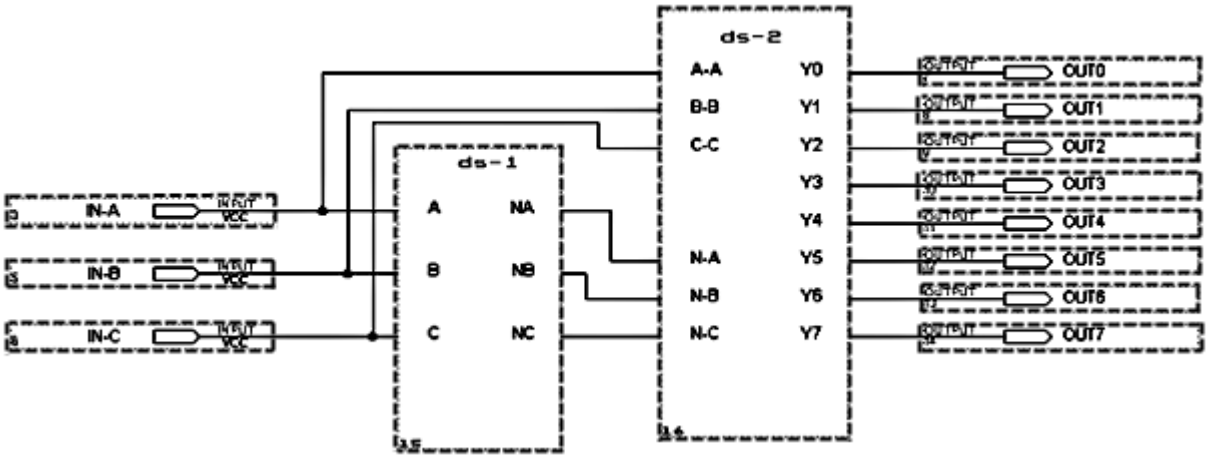


Рис.10.42.– Дешифратор на элементах И-НЕ

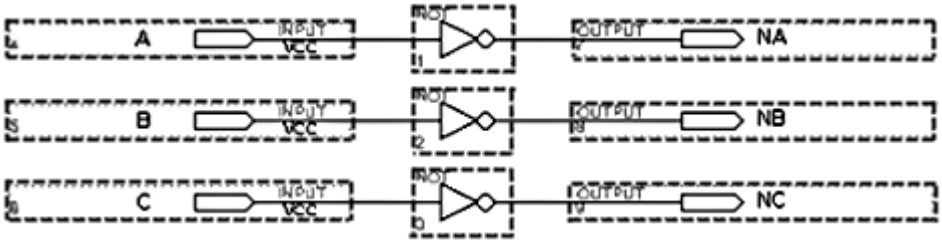


Рис.10.43.– Блок DS1

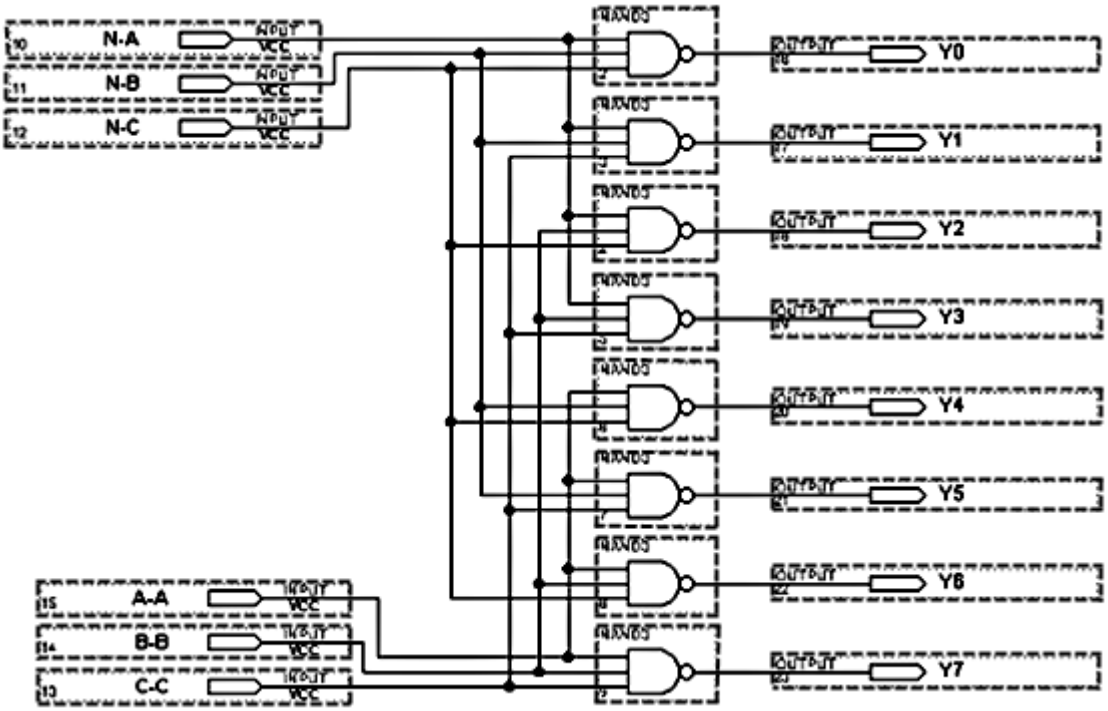


Рис. 10.44.– Блок DS2

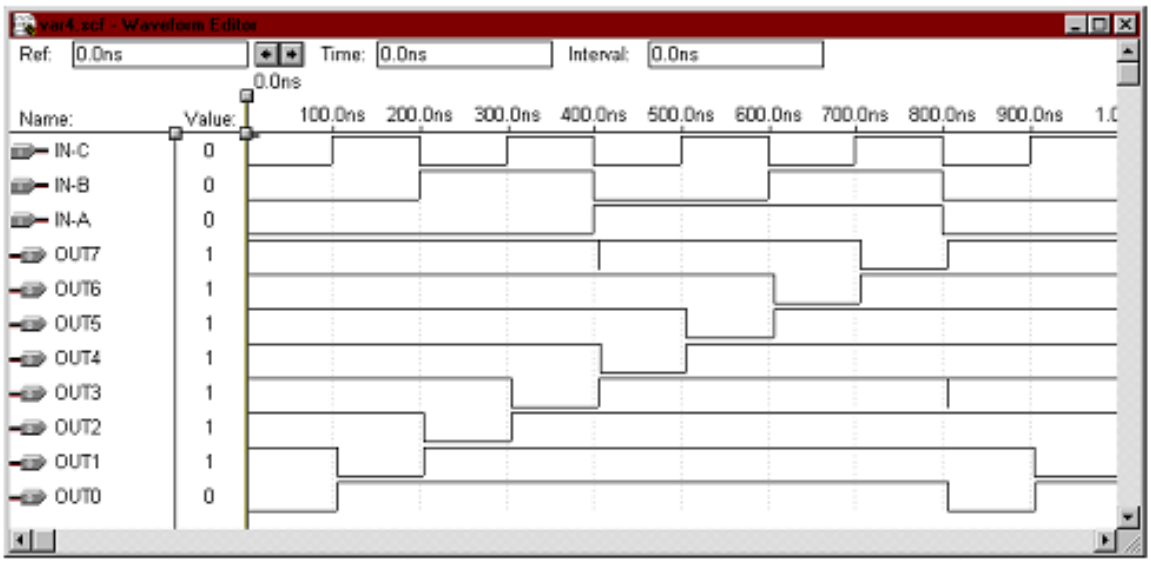


Рис.10.45.– Результаты моделирования дешифратора

Вариант №5.

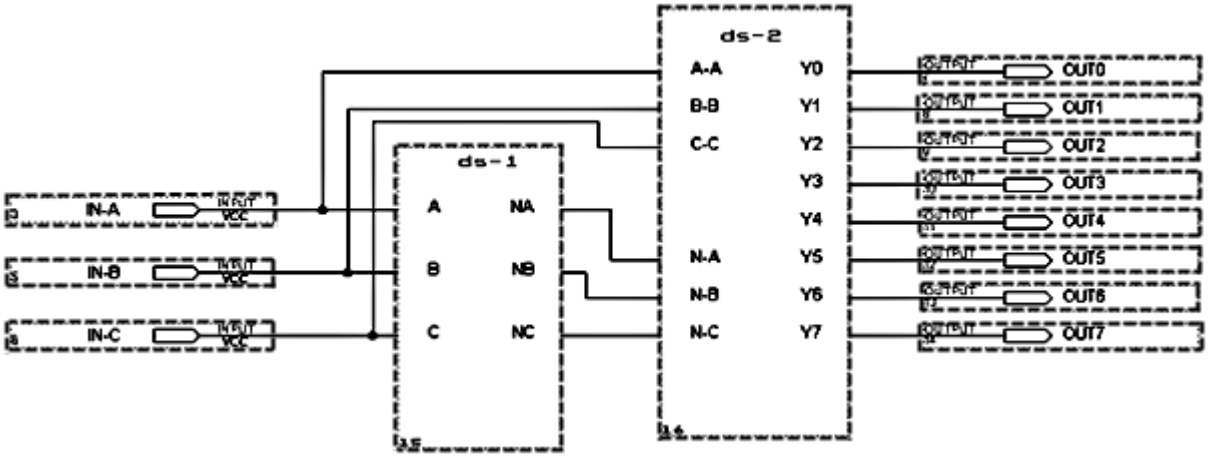


Рис.10.46.– Дешифратор на элементах ИЛИ-НЕ

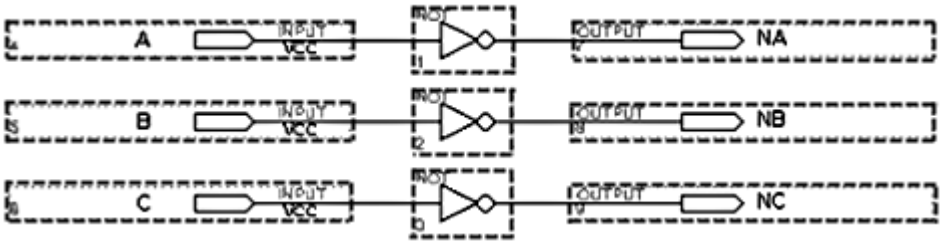


Рис.10.47.–Блок DS-1

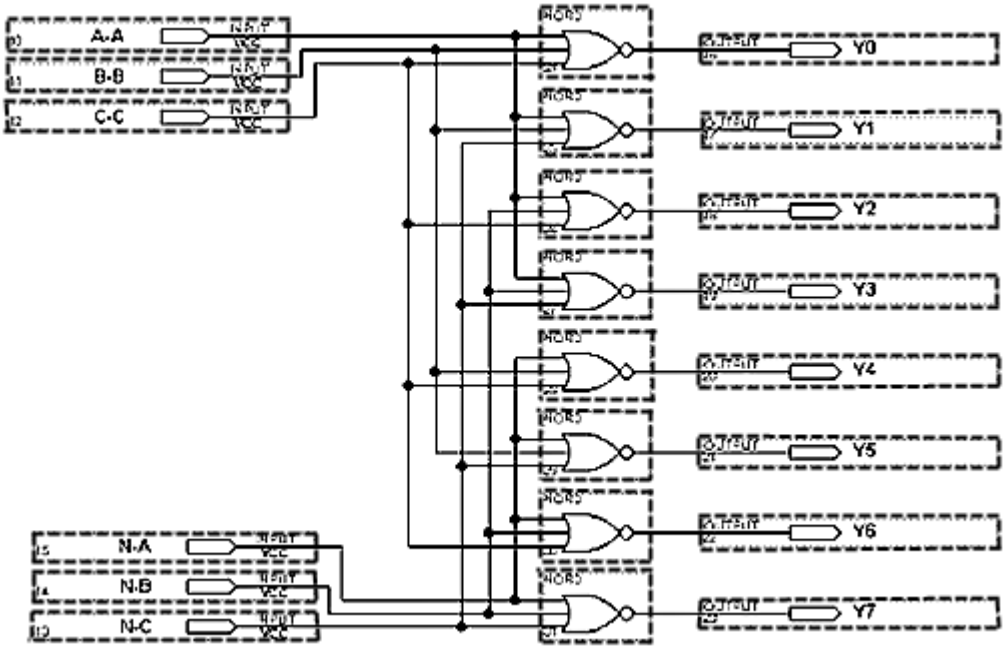


Рис.10.48.– Блок DS-2

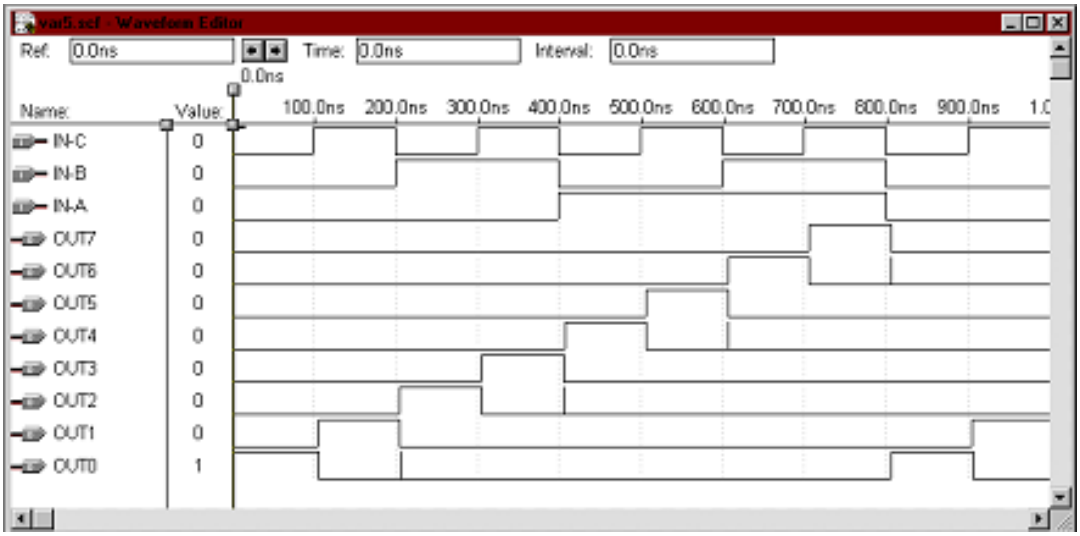


Рис.10.49.– Результаты моделирования дешифратора

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Способы реализации специализированных СБИС. Достоинства и недостатки.
2. Структурная организация ПЛИС.
3. Основные характеристики современных ПЛИС.
4. Маршрут проектирования специализированных СБИС на основе ПЛИС с использованием системы MAX+plus II.
5. Назначение и особенности графического редактора (Graphic Editor).
6. Назначение и особенности символьного редактора (Symbol Editor).
7. Назначение и особенности текстового редактора (Text Editor).
8. Назначение и особенности сигнального редактора (Waveform Editor).

9. Назначение и особенности редактора базового плана (Floorplan Editor).
10. Подготовка и порядок проведения компиляции проекта.
11. Моделирование проекта в системе MAX+plus II.

10.3. Лабораторная работа №2. Реализация графа состояний

Цель работы: реализация схемы автомата, её проектирование и компиляция

Порядок выполнения лабораторной работы:

- ознакомиться со сведениями о системе Max Plus II, изложенными в пп.10.1;
- получить у преподавателя вариант задания к лабораторной работе;
- выполнить последовательно все этапы проектирования;
- результаты проектирования показать преподавателю;
- приступить к проектированию заданного преподавателем варианта;

Задание: ввести схему автомата, выполнить компиляцию и моделирование.

Исходные данные: граф состояний на рисунке 10.50, значения состояний взять из таблицы 10.3 для своего варианта.

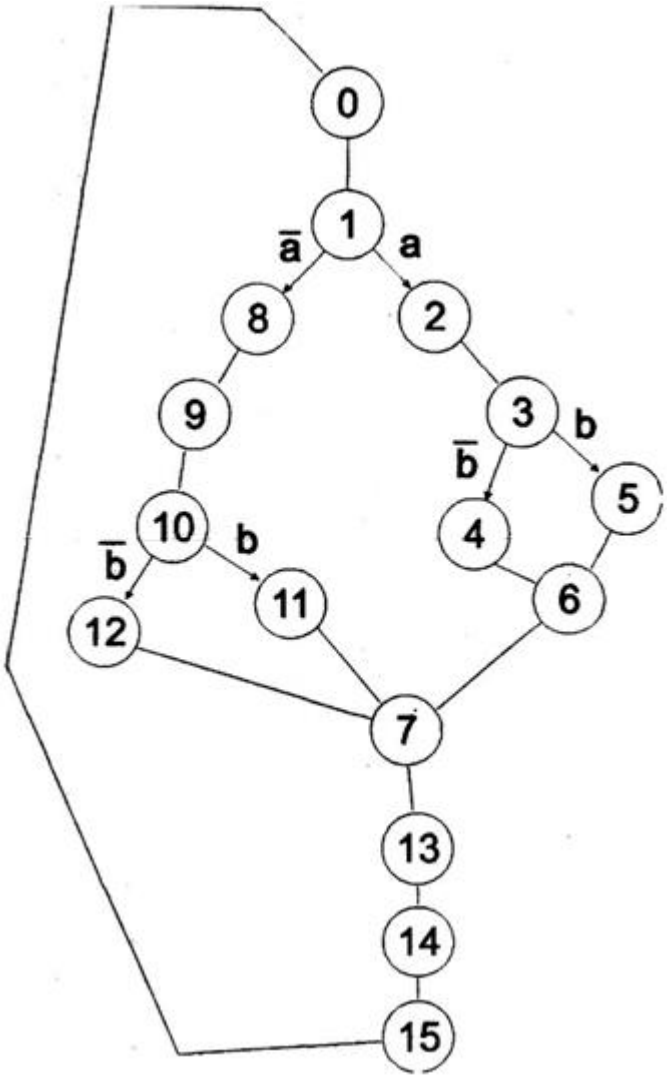


Рис.10.50.– Граф состояний для выбора варианта

Пример выполнения варианта №1.

Составим таблицу 10.4 перекодировки состояний автомата и их двоичный код:

Таблица 10.4.–Перекодировка состояний автомата в двоичный код

№ состояния	№ состояния из табл. 1	Двоичный код q3,q2,ql,q0
0	0	0000
1	3	0011
2	11	1011
3	12	1100
4	1	0001
5	9	1001
6	2	0010
7	14	1110
8	8	1000
9	7	0111
10	6	0110
11	4	0100
12	13	1101
13	10	1010
14	15	1111
15	5	0101

Подставляем новые значения в граф состояний:

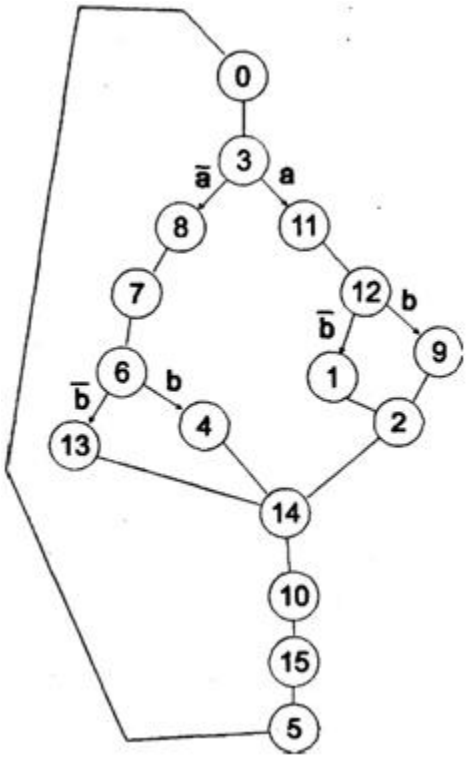


Рис.10.51.– Полученный граф состояний

Составим таблицу истинности автомата (таблица 10.5):

Таблица 10.5.–Таблица истинности автомата

старое состояние		условие	новое состояние	
№	код		№	КОД
0	0000	-	3	0011
3	0011	A=0	8	1000
3	0011	A=1	11	1011
8	1000	-	7	0111
7	0111	-	.6	0110
6	0110	B=0	13	1101
6	0110	B=1	4	0100
13	1101	-	14	1110
14	1110	-	10	1010
10	1010	-	15	1111
15	1111	-	5	0101
5	0101	-	0	0000
11	1011	-	12	1100
12	1100	B=0	1	0001
12	1100	B=1	9	1001
1	0001	-	2	0010
2	0010	-	14	1110
4	0100	-	14	1110
9	1001	-	2	0010

После получения таблицы истинности автомата создается функциональная схема в САПР Max+PlusII без минимизации.

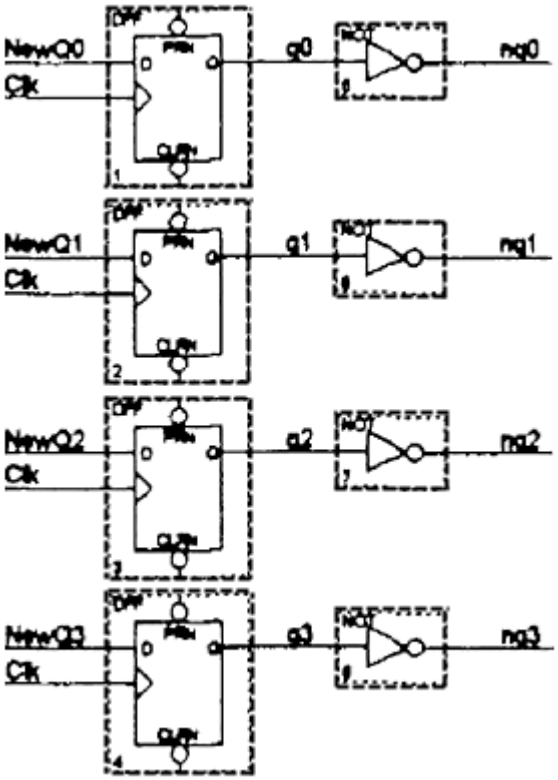


Рис.10.52.– Схема инверсии

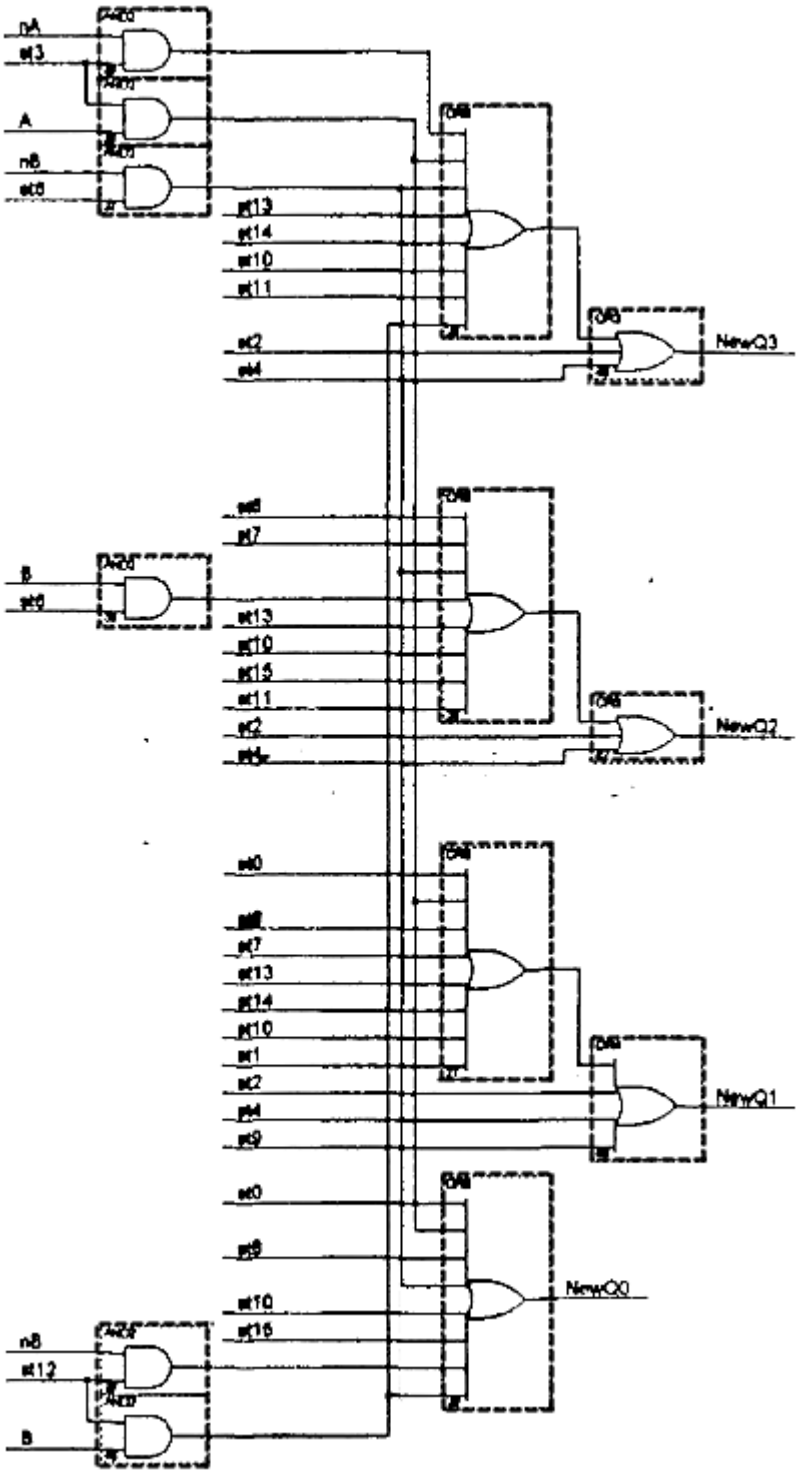


Рис.10.53.– Схема, реализованная согласно таблицы истинности 10.5

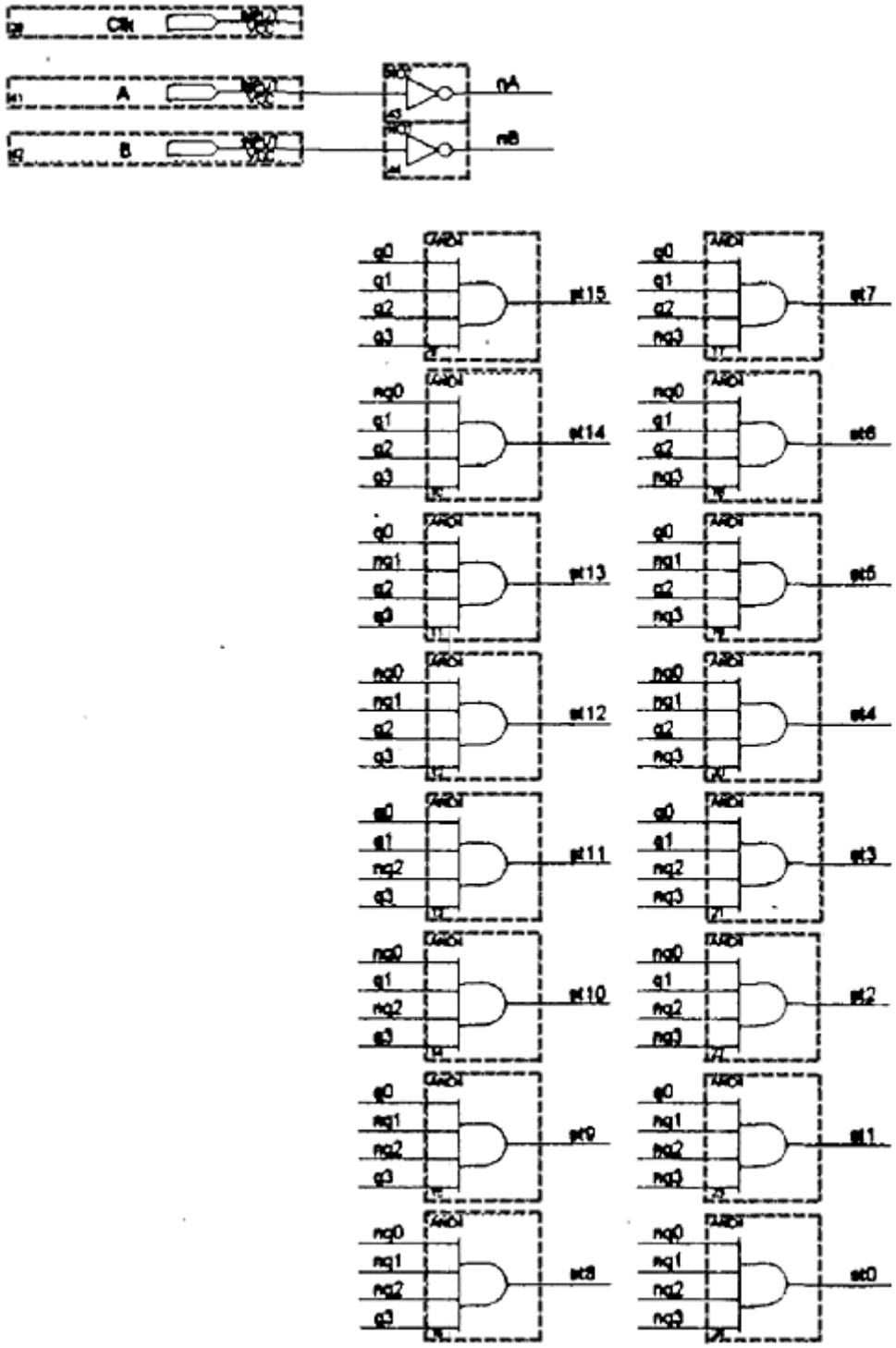


Рис.10.54.– Схема, согласно таблице истинности без минимизации

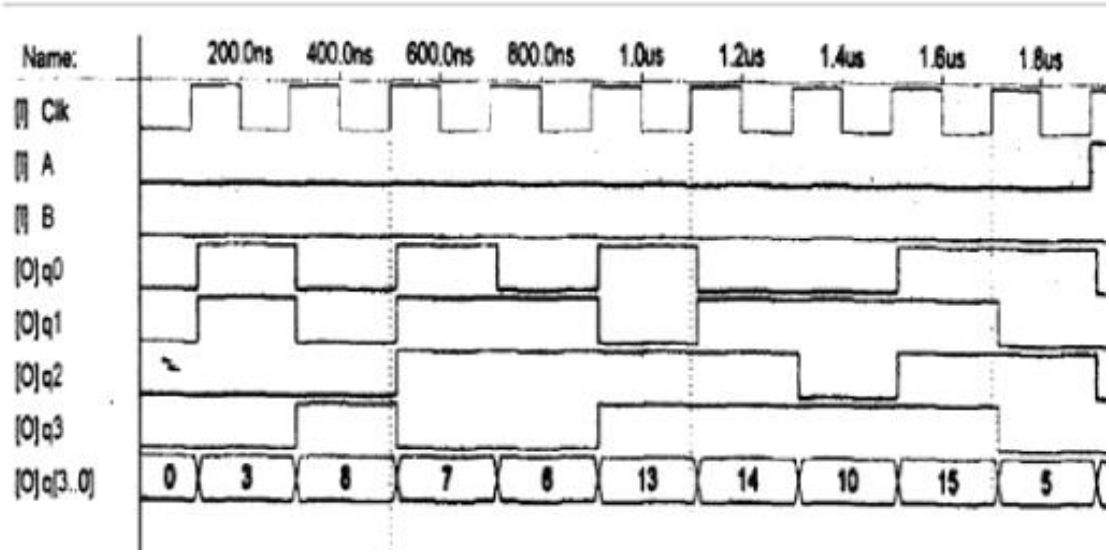


Рис.10.55.– Временная диаграмма работы устройства.

Отчет по лабораторной работе сдается преподавателю в распечатанном с одной стороны листа на формате А4 виде.

Содержание отчета по лабораторной работе №2:

- Титульный лист установленного образца (Приложение №1).
- Содержание.
- Схема электрическая принципиальная заданного устройства (согласно варианту) с объяснением на вентильно-регистровом уровне.
- Временная диаграмма работы устройства с описанием на уровне входных и выходных сигналов.
- Выводы с указанием на результаты проектирования.
- Файлы проекта в Max Plus II на носителе информации.

Таблица 10.6.–Варианты заданий к лабораторной работе №2.

№ вар.	Состояния графа															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	3	11	12	1	9	2	14	8	7	6	4	13	10	15	5
2	5	0	3	11	12	1	9	2	14	8	7	6	4	13	10	15
3	15	5	0	3	11	12	1	9	2	14	8	7	6	4	13	10
4	10	15	5	0	3	11	12	1	9	2	14	8	7	6	4	13
5	13	10	15	5	0	3	11	12	1	9	2	14	8	7	6	4
6	4	13	10	15	5	0	3	11	12	1	9	2	14	8	7	6
7	6	4	13	10	15	5	0	3	11	12	1	9	2	14	8	7
8	7	6	4	13	10	15	5	0	3	11	12	1	9	2	14	8
9	8	7	6	4	13	10	15	5	0	3	11	12	1	9	2	14
10	14	8	7	6	4	13	10	15	5	0	3	11	12	1	9	2
11	2	14	8	7	6	4	13	10	15	5	0	3	11	12	1	9
12	9	2	14	8	7	6	4	13	10	15	5	0	3	11	12	1
13	1	9	2	14	8	7	6	4	13	10	15	5	0	3	11	12
14	12	1	9	2	14	8	7	6	4	13	10	15	5	0	3	11
15	11	12	1	9	2	14	8	7	6	4	13	10	15	5	0	3
16	3	11	12	1	9	2	14	8	7	6	4	13	10	15	5	0
17	0	5	15	10	13	4	6	7	.8	14	2	9	1	12	11	3
18	3	0	5	15	10	13	4	6	7	8	14	2	9	1	12	11
19	11	3	0	5	15	10	13	4	6	7	8	14	2	9	1	12
20	12	11	3	0	5	15	10	13	4	6	7	8	14	2	9	1
21	1	12	11	3	0	5	15	10	13	4	6	7	8	14	2	9
22	9	1	12	11	3	0	5	15	10	13	4	6	7	8	14	2
23	2	9	1	12	11	3	0	5	15	10	13	4	6	7	8	14
24	14	2	9	1	12	11	3	0	5	15	10	13	4	6	7	8
25	8	14	2	9	1	12	11	3	0	5	15	10	13	4	6	7
26	7	8	14	2	9	1	12	11	3	0	5	15	10	13	4	6
27	6	7	8	14	2	9	1	12	11	3	0	5	15	10	13	4
28	4	6	7	8	14	2	9	1	12	11	3	0	5	15	10	13
29	13	4	6	7	8	14	2	9	1	12	11	3	0	5	15	10
30	10	13	4	6	7	8	14	2	9	1	12	11	3	0	5	15
31	15	10	13	4	6	7	8	14	2	9	1	12	11	3	0	5
32	5	15	10	13	4	6	7	8	14	2	9	1	12	11	3	0

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как составляется таблица истинности для данного автомата?
2. Поясните порядок составления схемы на основании таблицы истинности для вашего случая.
3. Поясните временную диаграмму работы данной схемы.
4. Почему используется для составления проекта таблица истинности без минимизации?

10.4. Лабораторная работа №3. Разработка устройства на основании имеющейся логической функции.

Цель работы: реализация схемы, её проектирование и компиляция на основании заданной логической функции.

Порядок выполнения лабораторной работы:

- ознакомиться со сведениями о графическом редакторе системы Max Plus II, изложенными в пп.10.1;
- получить у преподавателя вариант задания к лабораторной работе;
- рассчитать таблицу истинности для индивидуального задания;
- приступить к проектированию заданного преподавателем варианта;
- реализовать индивидуальное задание схематически в среде Max Plus II;
- протестировать работу полученной схемы;
- выполнить размещение полученного решения на кристалле.

Задание: составить схему и таблицу истинности на основании данной логической функции, выполнить компиляцию и моделирование.

Исходные данные: значения логических функций взять ниже для своего варианта.

Отчет по лабораторной работе должен содержать:

- Титульный лист установленного образца (Приложение №1).
- Содержание.
- Схема электрическая принципиальная заданного устройства (согласно варианту) с объяснением на вентильно-регистровом уровне.
- Временная диаграмма работы устройства с описанием на уровне входных и выходных сигналов.
- Выводы с указанием на результаты проектирования.
- Файлы проекта в Max Plus II на носителе информации.

Варианты заданий:

Вариант1

$$Y1 = X4 \wedge \overline{\left(X5 \wedge \overline{(X4)} \wedge X4 \wedge \overline{(X1)}\right)}$$

$$Y2 = X3 \wedge X6 \vee \overline{\left(X2 \wedge \overline{(X4)} \vee X8\right)} \wedge X7 \wedge X8$$

$$Y3 = X4 \wedge \overline{\left(X8 \wedge \overline{\left(X6 \vee \overline{(X3)} \vee X8\right)}\right)}$$

$$Y4 = X4 \wedge \overline{\left(X7 \vee X7 \wedge \overline{(X4)} \wedge \overline{(X5)}\right)}$$

$$Y5 = X4 \vee \overline{\left(X1 \wedge \overline{\left(X2 \vee \overline{(X8)}\right)}\right)}$$

Вариант2

$$Y1 = X1 \wedge \overline{\left(X8 \oplus X8 \vee X6 \wedge \overline{(X3)}\right)} \wedge X5$$

$$Y2 = X4 \vee \overline{\left(X3 \wedge \overline{(X3)}\right)} \wedge X5 \vee \overline{(X4)}$$

$$Y3 = X5 \wedge \overline{(X5 \oplus X3 \oplus X1 \wedge X8 \wedge X7)}$$

$$Y4 = X5 \wedge \overline{\left(X1 \wedge \overline{(X5 \oplus X5 \vee X4)} \vee X1\right)}$$

$$Y5 = X7 \wedge \overline{\left(X5 \wedge \overline{\left(X2 \vee \overline{(X1)} \vee X2\right)}\right)}$$

Вариант3

$$Y1 = X4 \wedge \overline{\left(X8 \oplus X5 \oplus X1 \vee \overline{(X5)} \vee X2\right)}$$

$$Y2 = X4 \oplus X6 \oplus X7 \vee \overline{(X8)} \vee \overline{(X7)}$$

$$Y3 = X8 \oplus X2 \vee \overline{(X4)} \vee \overline{(X7)} \wedge X2$$

$$Y4 = X1 \wedge \overline{(X3 \vee X4 \wedge X6 \oplus X1 \vee X3 \wedge X5 \oplus X5)}$$

$$Y5 = X8 \wedge X5 \wedge \overline{(X1)} \wedge \overline{(X2)} \vee \overline{(X3)}$$

Вариант4

$$Y1 = X5 \wedge X3 \vee \overline{(X4 \oplus X4 \wedge X4 \wedge X6 \vee X7)} \oplus X5$$

$$Y2 = X3 \wedge \overline{(X1 \oplus X1 \vee X3 \vee X8 \wedge X2 \vee X1)}$$

$$Y3 = X4 \wedge \overline{\left(X6 \vee \overline{(X6 \wedge X4)} \vee \overline{(X6)}\right)}$$

$$Y4 = X6 \wedge \overline{\left(X4 \vee \overline{(X2)} \oplus X2 \wedge X1\right)}$$

$$Y5 = X5 \wedge \overline{\left(X4 \vee \overline{(X3)} \wedge \overline{(X1)}\right)}$$

Вариант5

$$Y1 = X2 \oplus X8 \vee \overline{\left(X4 \vee X3 \wedge X1 \wedge X8 \vee \overline{(X2)}\right)}$$

$$Y2 = X4 \vee \overline{(X5)} \wedge X8 \wedge \overline{(X6)} \vee \overline{(X1)}$$

$$Y3 = X2 \oplus X2 \vee X3 \vee X7 \vee X3 \vee \overline{(X1)} \vee X2$$

$$Y4 = X6 \oplus X1 \oplus X1 \wedge \overline{(X2)} \wedge X8 \vee X7$$

$$Y5 = X3 \vee X7 \wedge \overline{(X2)} \vee \overline{(X6)} \wedge \overline{(X8)}$$

Вариант6

$$Y1 = X3 \oplus X4 \oplus X3 \wedge X3 \wedge \overline{(X6)}$$

$$Y2 = X7 \vee \overline{(X4)} \oplus X6 \wedge \overline{(X6)} \wedge X8$$

$$Y3 = X6 \vee X5 \wedge \overline{\left(X1 \vee \overline{(X1)} \oplus X8\right)}$$

$$Y4 = X7 \wedge \overline{\left(X6 \wedge \overline{(X7)} \wedge X7\right)} \vee \overline{(X8)} \wedge X3$$

$$Y5 = X4 \vee \overline{(X7)} \wedge \overline{(X8)} \wedge X3 \vee X6 \vee X4$$

Вариант7

$$Y1 = X6 \oplus X7 \vee \overline{\left(X5 \oplus X5 \wedge \overline{(X2)}\right)} \vee X7$$

$$Y2 = X8 \wedge \overline{(X8)} \wedge X1 \vee \overline{(X8)} \oplus X6$$

$$Y3 = X1 \vee \overline{\left(X8 \vee \overline{\left(X8 \vee X5 \wedge \overline{(X3)} \vee X8\right)} \vee X3\right)}$$

$$Y4 = X1 \vee \overline{(X6)} \vee X8 \oplus X7 \wedge X3 \wedge \overline{(X1)}$$

$$Y5 = X1 \oplus X4 \wedge \overline{(X6)} \vee X3 \oplus X5$$

Вариант8

$$Y1 = X7 \vee X6 \vee X1 \oplus X2 \vee \overline{(X4)} \vee \overline{(X2)}$$

$$Y2 = X5 \wedge \overline{\left(X4 \wedge \overline{(X5)} \wedge \overline{(X3)}\right)} \oplus X2$$

$$Y3 = X7 \vee X1 \wedge X3 \vee \overline{(X5)} \wedge \overline{(X2)} \oplus X5$$

$$Y4 = X3 \vee X6 \wedge X7 \vee X2 \wedge \overline{(X6)} \wedge X6 \oplus X8$$

$$Y5 = X8 \vee \overline{\left(X5 \vee X2 \oplus X5 \wedge X5 \wedge \overline{(X7)}\right)}$$

Вариант9

$$Y1 = X2 \wedge \overline{\left(X1 \wedge X4 \oplus X8 \wedge \overline{(X8)}\right)}$$

$$Y2 = X5 \oplus X5 \oplus X1 \vee \overline{(X6)} \wedge \overline{(X2)} \vee X2$$

$$Y3 = X5 \wedge \overline{\left(X7 \vee \overline{(X6)}\right)} \wedge X4 \oplus X2$$

$$Y4 = X7 \oplus X7 \wedge \overline{(X5)} \oplus X5 \wedge \overline{(X5)}$$

$$Y5 = X3 \oplus X2 \oplus X3 \vee X3 \wedge X4 \vee \overline{(X4)} \wedge X4 \wedge X3$$

Вариант10

$$Y1 = X6 \vee \overline{\left(X3 \wedge \overline{\left(X6 \vee \overline{(X7)} \right)} \right)} \wedge X8$$

$$Y2 = X8 \oplus X6 \oplus X5 \vee X3 \oplus X4 \wedge X8$$

$$Y3 = X7 \vee X8 \wedge \overline{\left(X4 \vee X7 \wedge \overline{(X6)} \vee \overline{(X7)} \right)}$$

$$Y4 = X3 \vee X4 \oplus X8 \vee X3 \wedge X6 \vee X2 \wedge X3 \oplus X1 \vee X2$$

$$Y5 = X7 \vee X3 \wedge \overline{(X7)} \wedge X2 \vee X5 \vee \overline{(X1)}$$

Вариант11

$$Y1 = X3 \vee X2 \wedge \overline{\left(X3 \wedge \overline{(X5)} \vee \overline{(X5)} \right)} \wedge X1$$

$$Y2 = X1 \wedge X4 \vee X3 \oplus X1 \oplus X4 \vee X7 \oplus X1$$

$$Y3 = X2 \vee \overline{\left(X2 \wedge X5 \wedge \overline{(X2 \wedge X4 \vee X6)} \vee \overline{(X7)} \right)}$$

$$Y4 = X8 \vee X4 \wedge X8 \wedge \overline{(X3)} \vee \overline{(X8)} \wedge X6 \wedge X1 \wedge X5$$

$$Y5 = X8 \wedge X7 \wedge X4 \vee \overline{(X1)} \wedge \overline{(X6)} \wedge X7$$

Вариант12

$$Y1 = X7 \wedge X1 \vee \overline{(X1)} \wedge X8 \vee X4 \oplus X2 \vee X6$$

$$Y2 = X5 \wedge \overline{\left(X8 \wedge \overline{(X6)} \oplus X5 \right)} \oplus X4$$

$$Y3 = X5 \wedge \overline{\left(X8 \vee \overline{\left(X3 \vee X5 \wedge X5 \wedge \overline{(X5)} \right)} \right)}$$

$$Y4 = X1 \vee X7 \oplus X8 \wedge X3 \vee X3 \oplus X2 \wedge X3$$

$$Y5 = X4 \wedge X2 \oplus X2 \vee \overline{(X4)} \oplus X7$$

Вариант13

$$Y1 = X5 \wedge X4 \wedge \overline{\left(X5 \vee \overline{(X3)} \right)} \oplus X6$$

$$Y2 = X7 \vee X8 \wedge \overline{\left(X7 \wedge \overline{\left(X4 \wedge \overline{(X4)} \right)} \right)}$$

$$Y3 = X3 \wedge X1 \oplus X7 \vee \overline{(X1)} \wedge \overline{(X8)}$$

$$Y4 = X4 \wedge \overline{\left(X2 \wedge \overline{(X6 \vee X3)} \vee X8 \right)} \vee X2$$

$$Y5 = X6 \vee X7 \wedge X8 \oplus X2 \oplus X6 \wedge \overline{(X8)} \vee X5 \vee X5$$

Вариант14

$$Y1 = X7 \vee \overline{\left(X4 \vee X4 \vee \overline{(X3)} \wedge \overline{(X8)} \right)}$$

$$Y2 = X6 \vee \overline{(X5 \vee X5 \oplus X6 \vee X7)} \wedge \overline{(X2)}$$

$$Y3 = X1 \wedge X2 \vee X1 \oplus X7 \vee X3 \vee X4 \vee \overline{(X8)} \vee X5$$

$$Y4 = X6 \wedge X1 \wedge \overline{\left(X5 \oplus X2 \wedge \overline{(X5)} \vee X5 \right)}$$

$$Y5 = X7 \vee \overline{\left(X4 \vee X1 \oplus X3 \wedge \overline{(X7 \oplus X8)} \right)}$$

Вариант15

$$Y1 = X3 \wedge \overline{\left(X1 \oplus X7 \wedge X2 \wedge \overline{(X8)}\right)}$$

$$Y2 = X1 \vee X7 \wedge X7 \wedge \overline{\left(X7 \vee X3 \vee \overline{(X5)}\right)} \wedge X1$$

$$Y3 = X2 \vee \overline{\left(X7 \vee X5 \wedge \overline{(X2)} \wedge \overline{(X8)}\right)}$$

$$Y4 = X5 \vee \overline{\left(X6 \wedge X6 \oplus X2 \wedge \overline{(X7)} \oplus X5\right)}$$

$$Y5 = X8 \oplus X6 \wedge X7 \wedge \overline{(X2)} \oplus X4$$

Вариант16

$$Y1 = X2 \wedge X4 \oplus X6 \oplus X5 \oplus X8 \oplus X6 \vee X2$$

$$Y2 = X8 \vee \overline{\left(X5 \vee \overline{(X4 \oplus X3)} \vee X1\right)}$$

$$Y3 = X3 \oplus X4 \wedge \overline{\left(X3 \oplus X1 \vee \overline{(X5)}\right)} \wedge X6$$

$$Y4 = X7 \vee X6 \wedge \overline{\left(X8 \oplus X3 \vee X8 \wedge \overline{(X7)}\right)}$$

$$Y5 = X7 \wedge X1 \vee \overline{\left(X4 \oplus X4 \vee \overline{(X4)}\right)}$$

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Поясните принцип реализации заданной логической функции в общем виде.
2. Объясните принцип работы схемы на вентильно-регистровом уровне.
3. Поясните ВД работы схемы. На примере покажите правильность работы схемы.

10.5. Лабораторная работа №4. Разработка VHDL-кода устройства на основании имеющегося описания логической функции.

Цель работы: реализация VHDL-описания, её проектирование и компиляция на основании заданной логической функции.

Порядок выполнения лабораторной работы:

- ознакомиться со сведениями о текстовом редакторе системы Max Plus II, изложенными в пп.10.1;
- получить у преподавателя вариант задания к лабораторной работе;
- рассчитать таблицу истинности для индивидуального задания;
- приступить к проектированию заданного преподавателем варианта;
- реализовать индивидуальное задание в текстовом редакторе на языке VHDL в среде Max Plus II;
- протестировать работу полученного описания;
- сравнить полученную ВД с ВД, полученной в предыдущей работе;
- выполнить размещение полученного решения на кристалле.

Задание: составить VHDL-описание и таблицу истинности на основании данной логической функции, выполнить компиляцию и моделирование.

Исходные данные: значения логических функций взять ниже для своего варианта.

Отчет по лабораторной работе должен содержать:

- Титульный лист установленного образца (Приложение №1).
- Содержание.
- Программное описание на языке VHDL заданного устройства (согласно варианту) с построчным объяснением.
- Временная диаграмма работы устройства с описанием на уровне входных и выходных сигналов.
- Выводы с указанием на результаты проектирования.
- Файлы проекта в Max Plus II на носителе информации.

Варианты заданий и контрольные вопросы взять из лабораторной работы №3(пп.10.4).

10.6. Лабораторная работа №5. Моделирование передаточных функций

Цель работы: ознакомиться с методами полиномиальной аппроксимации функциональных зависимостей и их реализацией на языке VHDL в среде Max Plus II.

Краткие сведения из теории:

Общие сведения

На практике при моделировании различных процессов – в частности, экономических, физических, технических, социальных – широко используются те или иные способы вычисления приближенных значений функций по известным их значениям в некоторых фиксированных точках.

Такого рода задачи приближения функций часто возникают:

- при построении приближенных формул для вычисления значений характерных величин исследуемого процесса по табличным данным, полученным в результате эксперимента;
- при численном интегрировании, дифференцировании, решении дифференциальных уравнений и т. д.;
- при необходимости вычисления значений функций в промежуточных точках рассматриваемого интервала;
- при определении значений характерных величин процесса за пределами рассматриваемого интервала, в частности при прогнозировании.

Если для моделирования некоторого процесса, заданного таблицей, построить функцию, приближенно описывающую данный процесс на основе метода наименьших квадратов, она будет называться аппроксимирующей функцией (регрессией), а сама задача построения аппроксимирующих функций - задачей аппроксимации.

В Excel для построения регрессий имеется возможность добавления выбранных регрессий (линий тренда - trendlines) в диаграмму, построенную на основе таблицы данных для исследуемой характеристики процесса (доступно лишь при наличии построенной диаграммы);

Полиномиальная линия тренда полезна для описания характеристик, имеющих несколько ярко выраженных экстремумов (максимумов и минимумов). Выбор степени полинома определяется количеством экстремумов исследуемой характеристики. Так, полином второй степени может хорошо описать процесс, имеющий только один максимум или минимум; полином третьей степени - не более двух экстремумов; полином четвертой степени - не более трех экстремумов и т. д.

В этом случае линия тренда строится в соответствии с уравнением: $y = c_0 + c_1x + c_2x^2 + c_3x^3 + c_4x^4 + c_5x^5 + c_6x^6$ где коэффициенты c_0, c_1, c_2,... c_6 - константы, значения которых определяются в ходе построения.

Порядок реализации полиномиальной аппроксимации функциональной зависимости:

- По графику определить значения функции в точках 1, 2, 3, … и т.д.
- Занести значения в таблицу в Microsoft Excel.
- Построить график по таблице
- Выбрав график войти в меню "Диаграмма" - "Добавить линию тренда".
- Выбрать тип аппроксимации "Полиномиальная".
- Выбрать требуемую степень полинома.

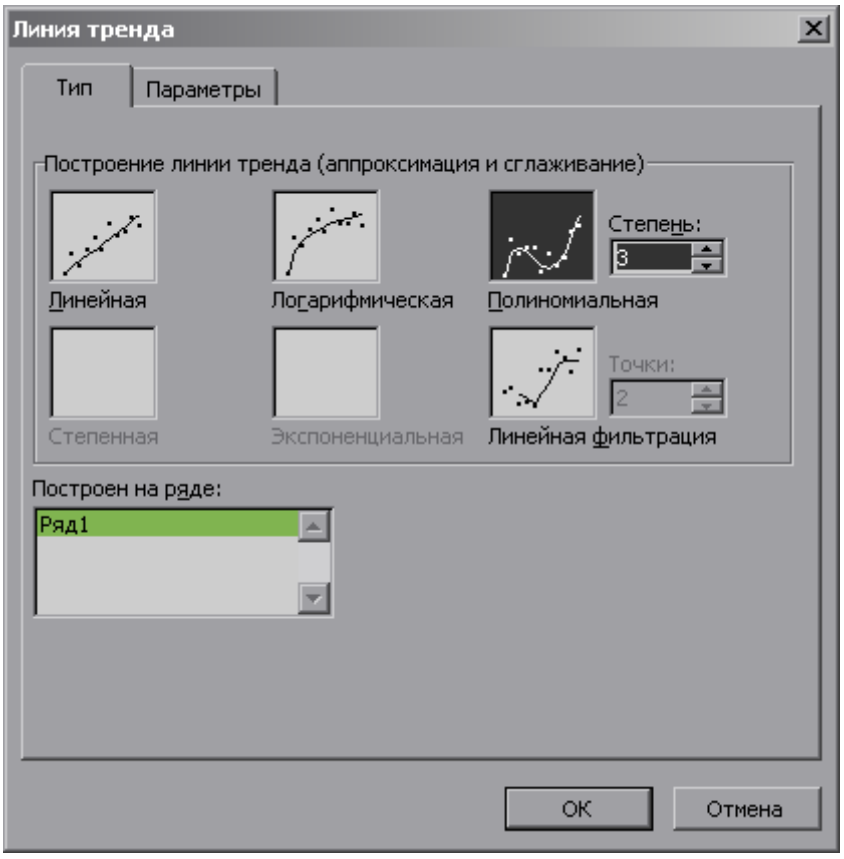


Рис.10.56.– Меню "Диаграмма" - "Добавить линию тренда"

На вкладке "Параметры" установить флажок "Показывать уравнение на диаграмме".

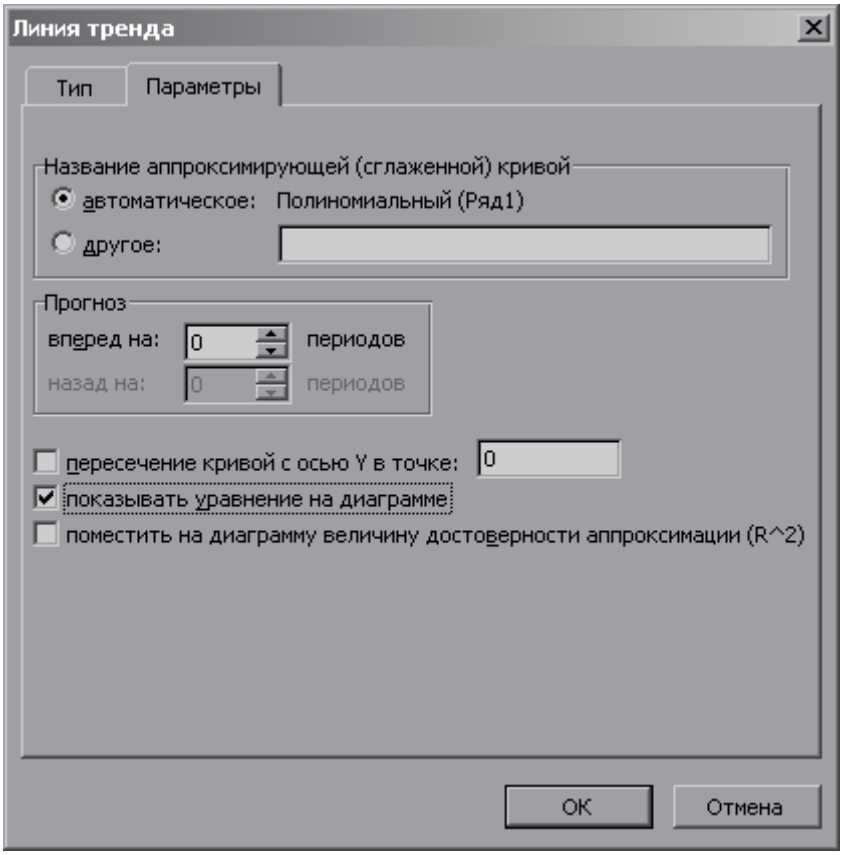


Рис.10.57.– Вкладка "Параметры"

На полученной диаграмме будет выведено уравнение и график аппроксимированной функции.

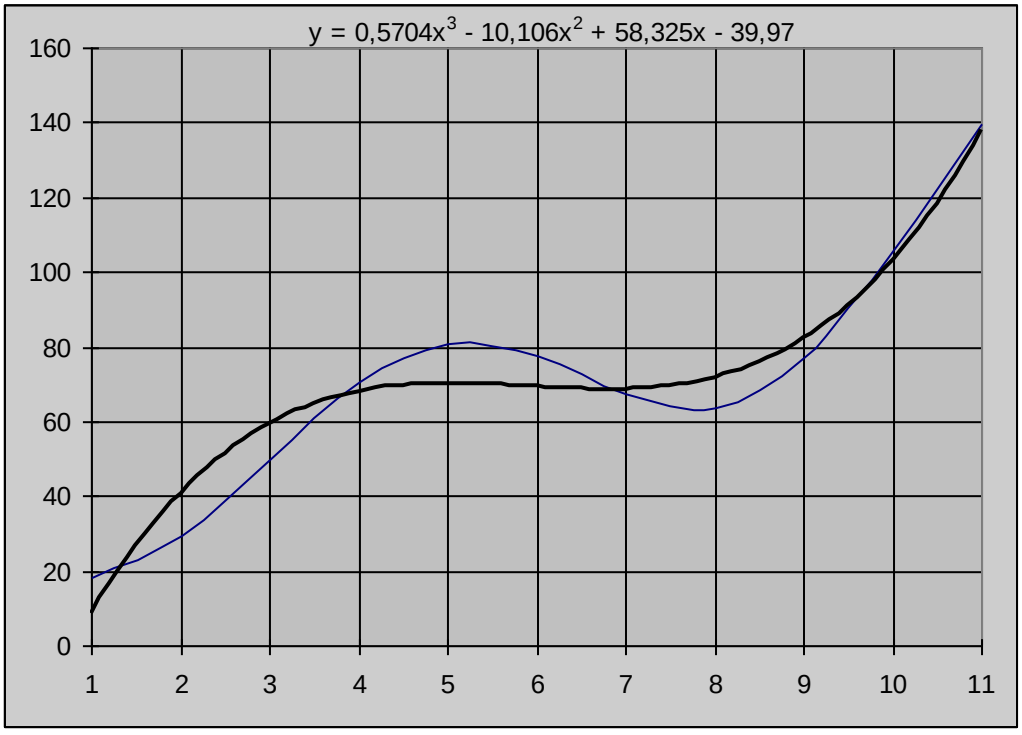


Рис.10.58.– Диаграмма аппроксимированной функции

Создаем архитектуру и процесс в ней. Входные и выходные параметры должны иметь такую размерность, что бы вмещать все входные и выходные значения. В списке чувствительности процесса должен быть задан входной порт. Реализовать вычисление полученного полинома с учетом преобразований типов.

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity pol is
     port(
            -- Входное число разрядностью 8 бит
            in1 : in std_logic_vector (7 downto 0);
            -- Результирующее число разрядностью 16 бит
            out1 : out std_logic_vector (15 downto 0)
         );
end pol;

architecture pol of pol is
begin
    process(
    in1 )
    variable x : INTEGER;
    variable y : INTEGER;
    variable sum : real;
    variable cur : real;
    begin
        -- Исходное целое число переводим в вещественный формат
        x := conv_integer( unsigned( in1 ) );
        cur := real(x);
        -- Аппроксимирующая функция
        -- y = 0,5704x3 - 10,106x2 + 58,325x - 39,97
        sum := ( cur**3 ) * (0.5704) + ( cur**2 ) * (-10.106) + ( cur ) * (58.325) - 39.97;
        -- Результат необходимо преобразовать в целое число,
        -- а затем в битовый вектор
        y := integer( sum );
        out1 <= std_logic_vector( conv_unsigned (y, 16 ) );
    end process;
end pol;
```

Рис.10.59.– Описание функции на языке VHDL

Создать график, в котором для входного сигнала установлен тип "Binary counter".

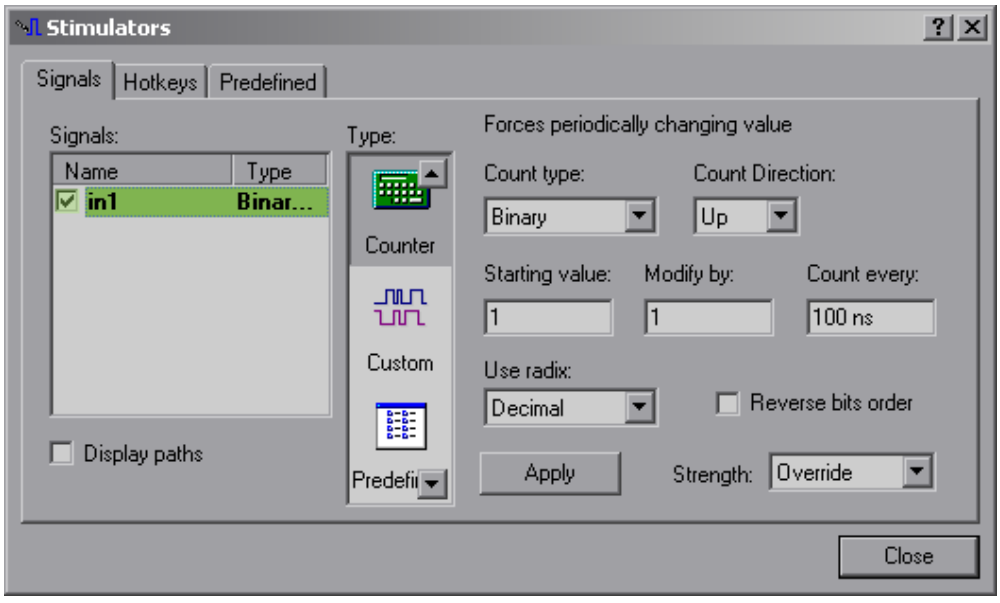


Рис.10.60.– Построение графика функции

Произвести "симуляцию" для заданного диапазона входных значений. Полученные значения свести в таблицу и построить по ним график.

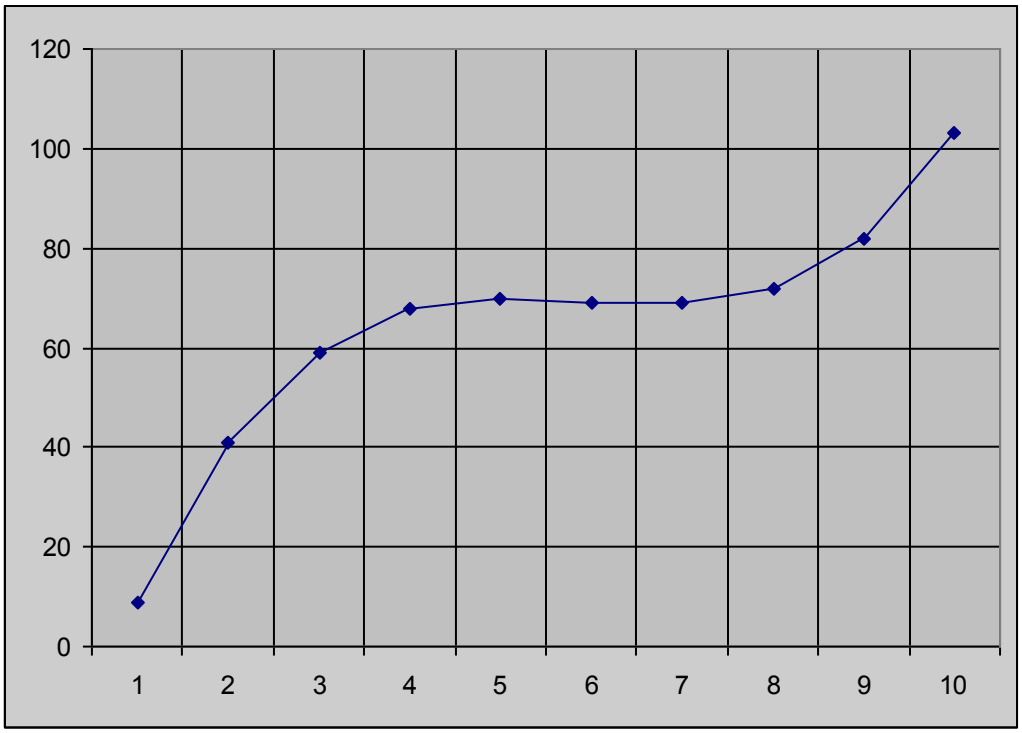


Рис.10.61.– Полученный график с помощью VHDL-описания

Создание синтезируемой реализации аппроксимирующей функции.

При программировании синтезируемых вычислений следует учесть несколько особенностей:

- Для синтезируемой операции деления второй операнд должен быть степенью двойки. В этом случае операция деления эквивалентна операции сдвига вправо.
- Для операции возведения в степень, возводимое число должно быть равно степени двойки. В этом случае операция эквивалентна сдвигу влево.
- При синтезе нельзя использовать тип **real**.
- Возведение в степень необходимо заменить несколькими умножениями.
- Все дробные коэффициенты должны быть приведены к целым числам. Этого можно достигнуть, представив каждый коэффициент как результат деления двух целых чисел. Например, выражение 0.5*x представляется как x*5/10. Необходимо учесть, что целесообразнее сначала выполнять умножение, а затем деление, что бы не произошло значительной потери точности.
- Все операции деления должны быть приведены к делению на степень двойки. Это необходимо учесть при формировании коэффициентов. Например, x*0.5 представляется как x*(0.5*16)/16 = x*8/16.

В связи с невозможностью использования чисел с плавающей точкой и для повышения точности входные значения и результат функции должны быть промасштабированы. Например, входные значения увеличены в 128 раз и результат должен быть увеличен в 128 раз. Таким образом, на вход функции должны подаваться значения 128, 256, 384 и т.д., а полученный результат необходимо будет разделить на 128 перед построением графика. При масштабировании чисел необходимо учитывать степени параметра функции.

Для примера из пункта 2, промасштабируем все числа в 16 раз. Получим:
y := (x*x*x) * 9 / (16*16*16) + (x*x) * (-162) / (16*16) + x * 933/ 16 - 639;

Как было указано выше, для повышения точности деление необходимо выполнять как можно позже. Поэтому полученное выражение можно преобразовать так:

```
y := ( x*x*x ) * 9 / 16;
y := (y + ( x*x ) * (-162)) / 16;
y := (y + x * 933)/(16);
y := y - 639;
```

При выборе коэффициента масштабирования необходимо предусмотреть, что бы максимальное вычисляемое значение не превышало максимального значения типа INTEGER. Например, если выбрать масштабирующий коэффициент 256, а не 16, то максимально возможное число (при интервале от 1 до 10) будет $(10*256)^3=16777216000$, что заведомо больше 2^{32} и приведет к переполнению.

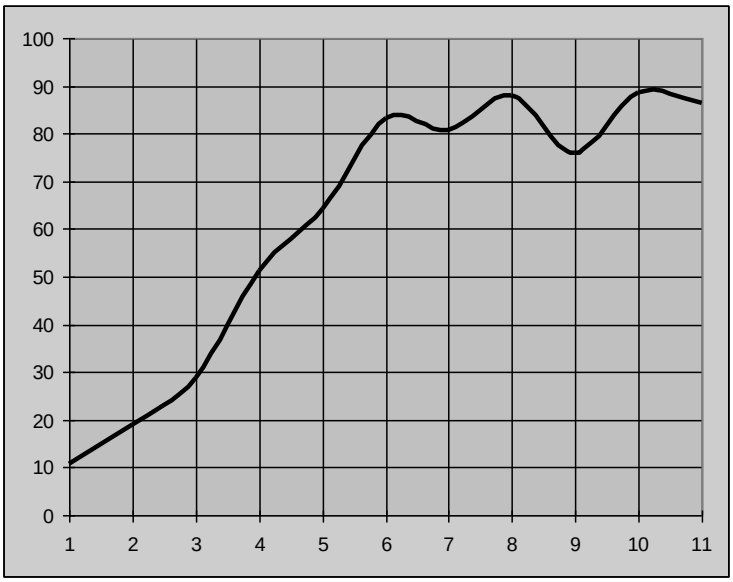
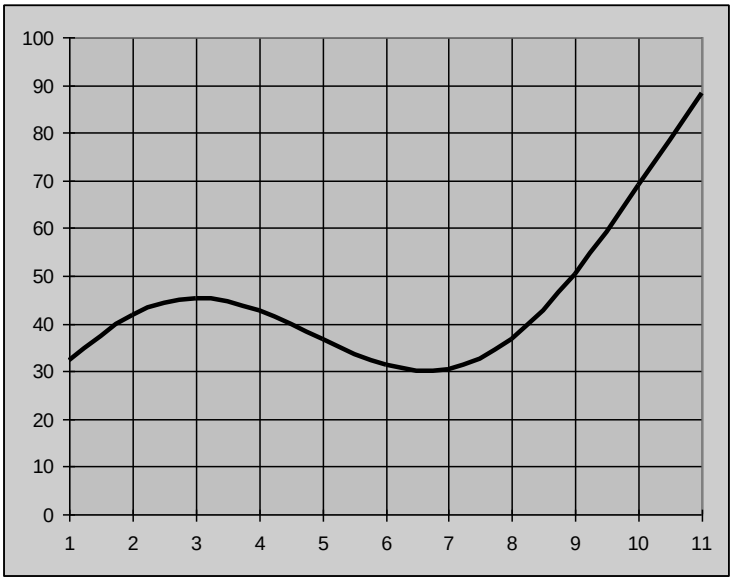
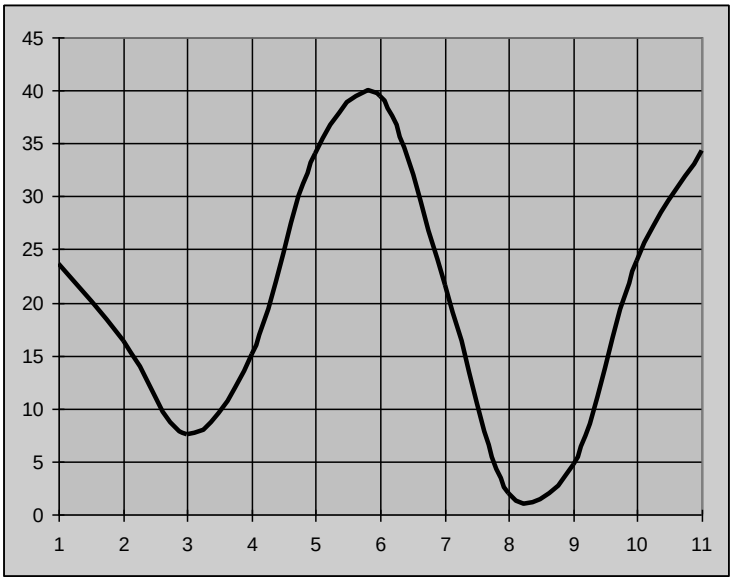
Для полученной зависимости провести «Симуляцию» и построить график.

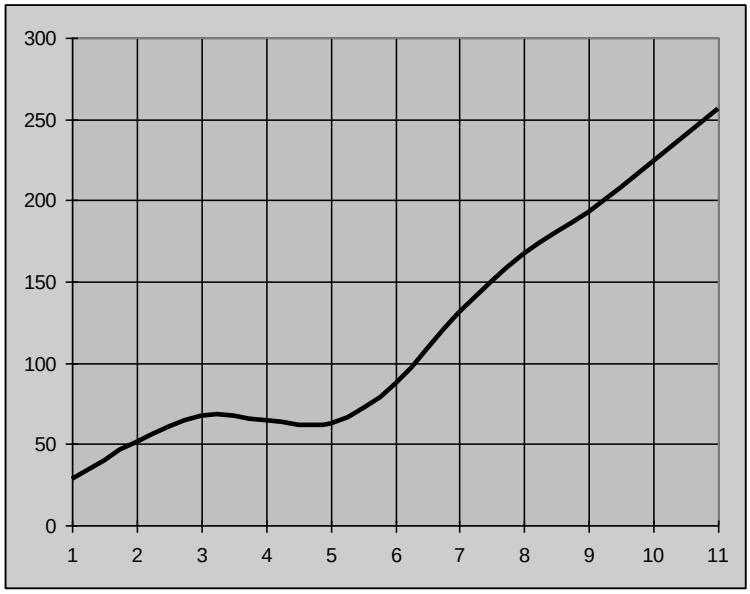
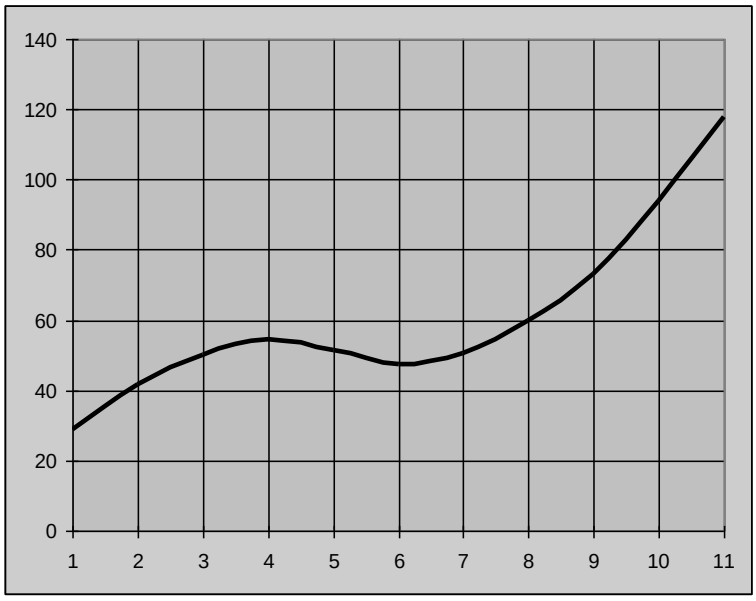
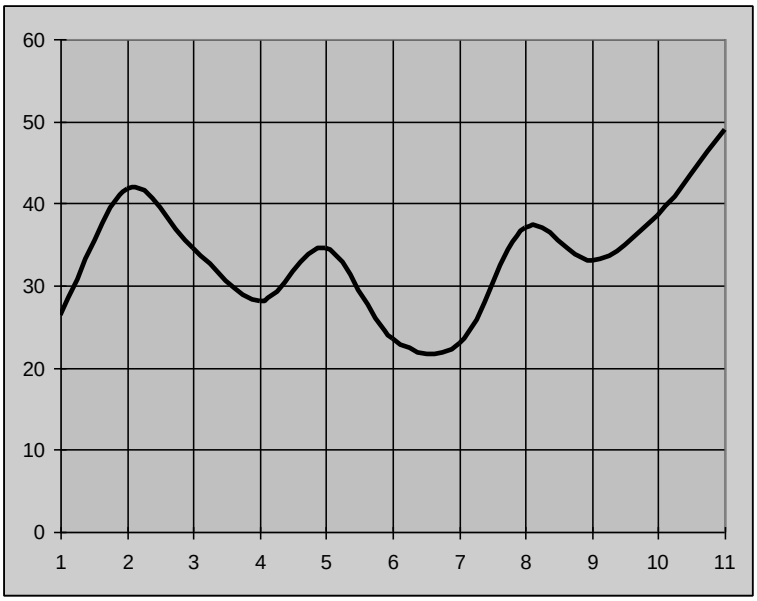
Разместить код на кристалле. При размещении необходимо учитывать, что будет занята значительная площадь кристалла, а на некоторых малых кристаллах размещение может быть неудачным.

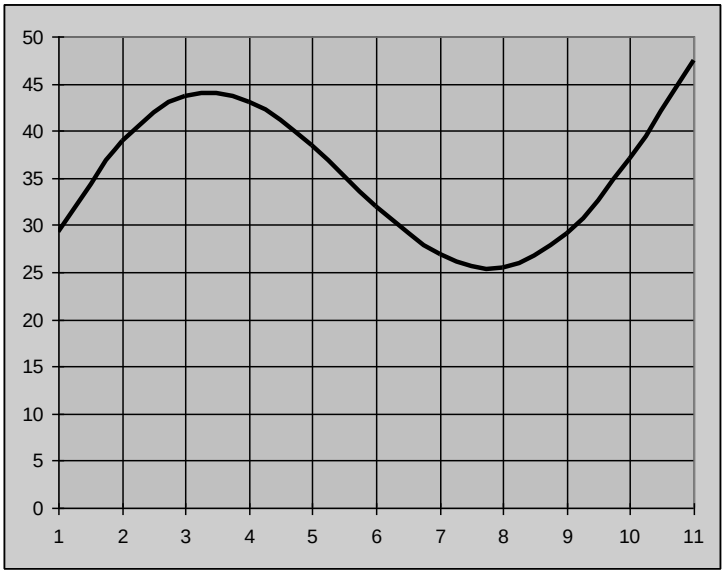
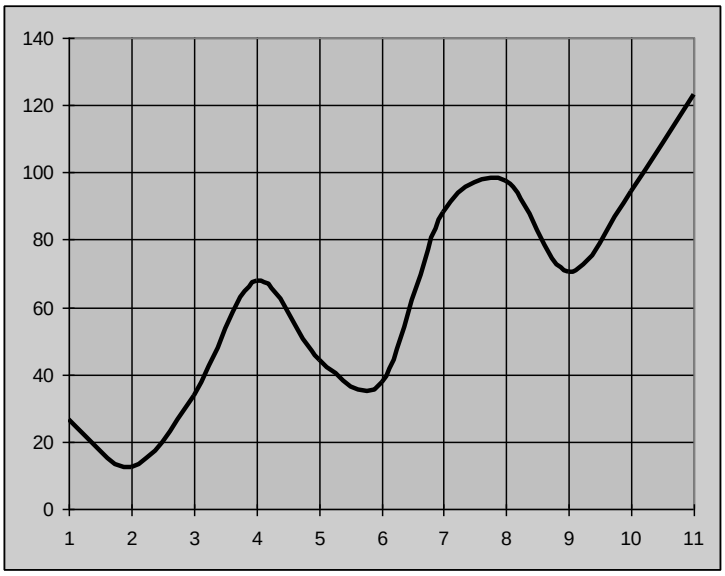
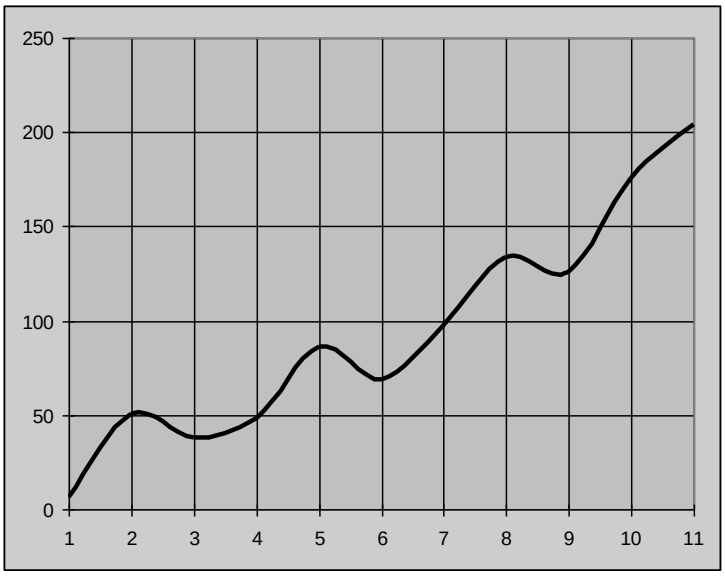
Отчет по лабораторной работе должен содержать:

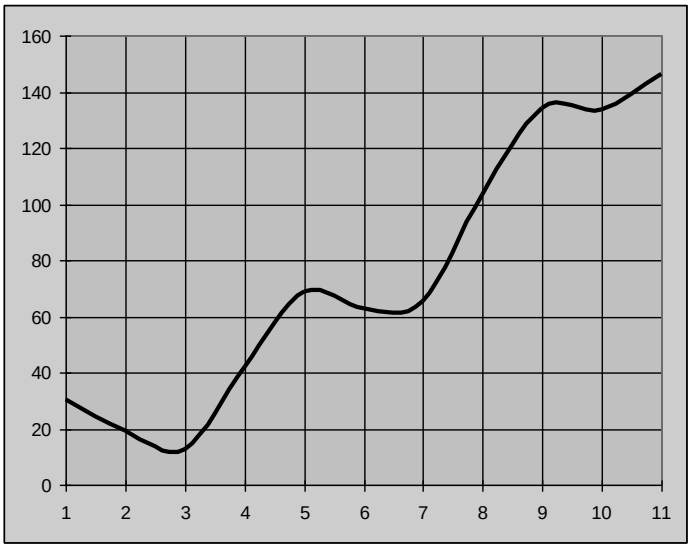
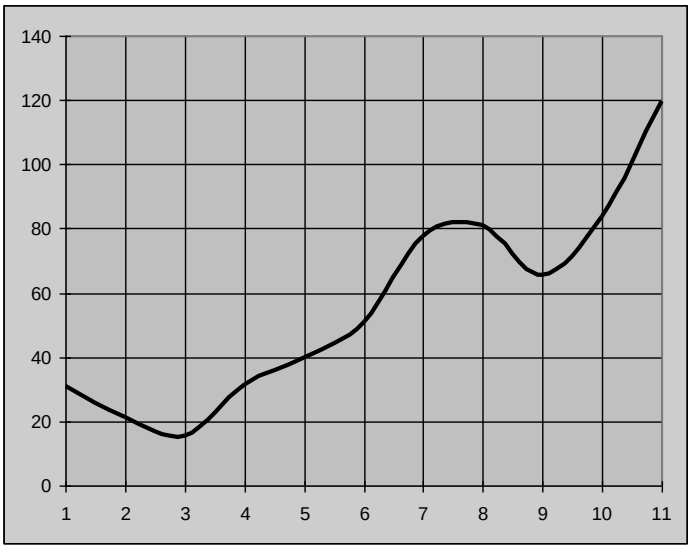
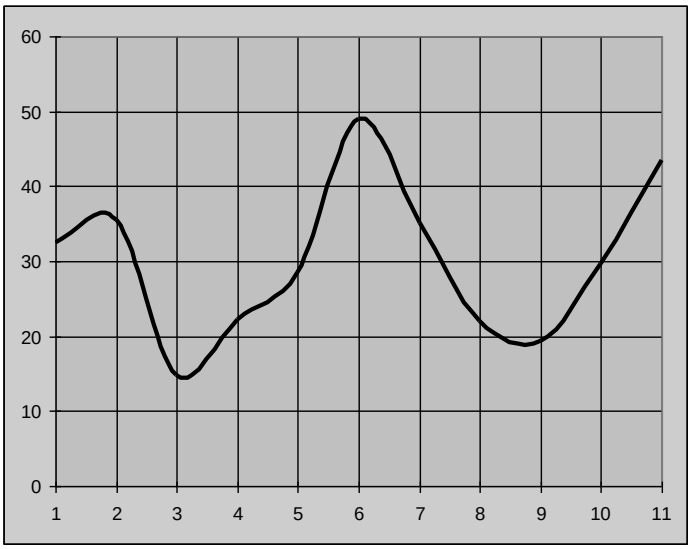
- Титульный лист установленного образца (Приложение №1).
- Содержание.
- Программное описание на языке VHDL заданного устройства (согласно варианту) с построчным объяснением.
- Временная диаграмма работы устройства с описанием на уровне входных и выходных сигналов.
- Диаграммы, полученные с помощью MS Excel.
- Выводы с указанием на результаты проектирования.
- Файлы проекта в Max Plus II на носителе информации.

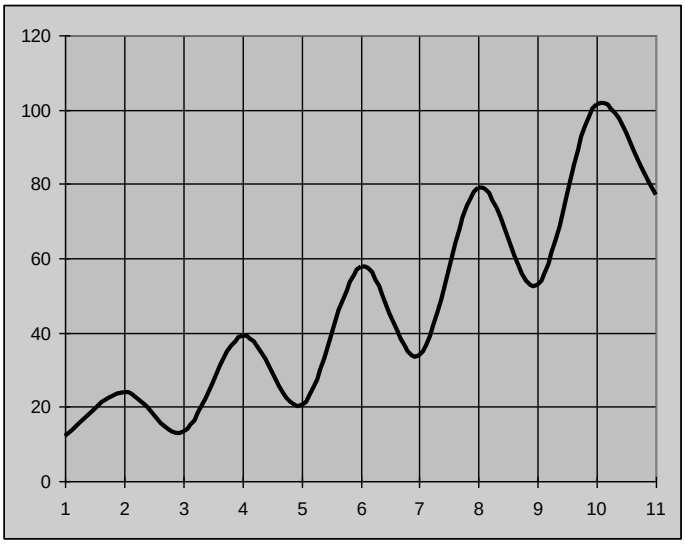
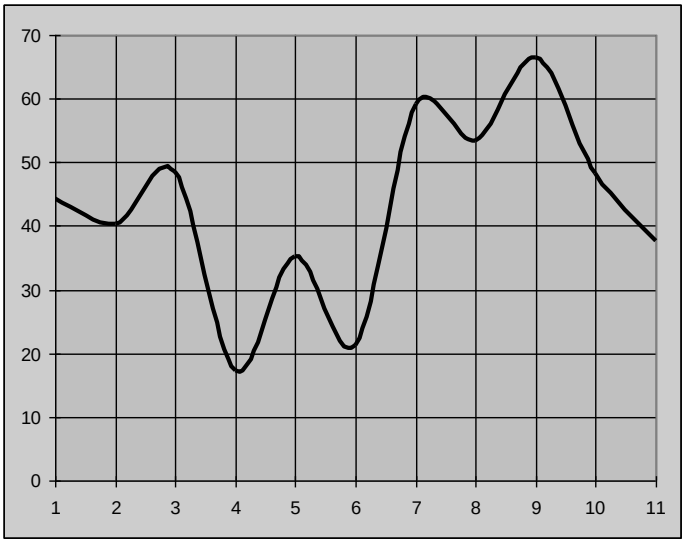
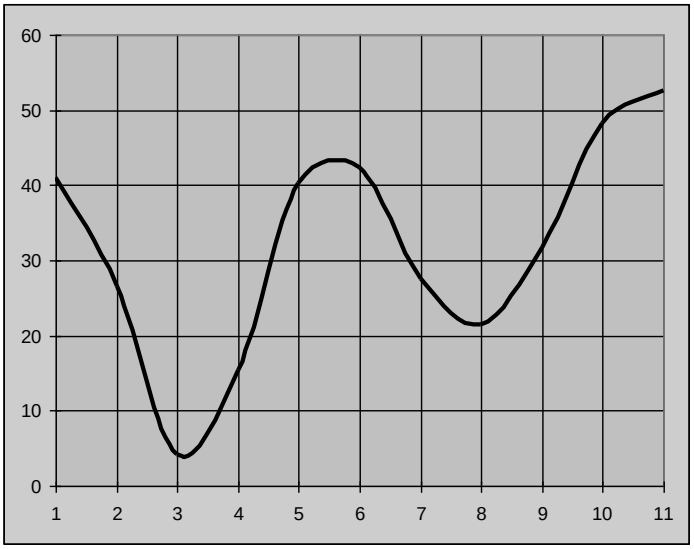
Варианты задания:

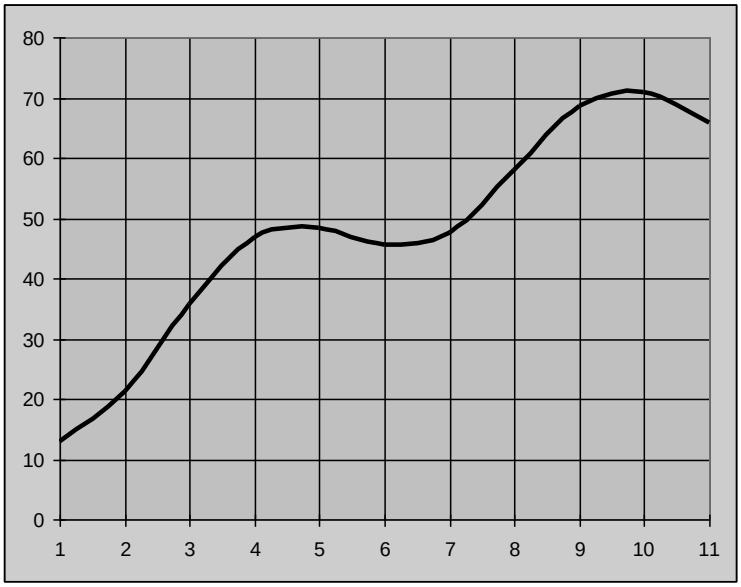
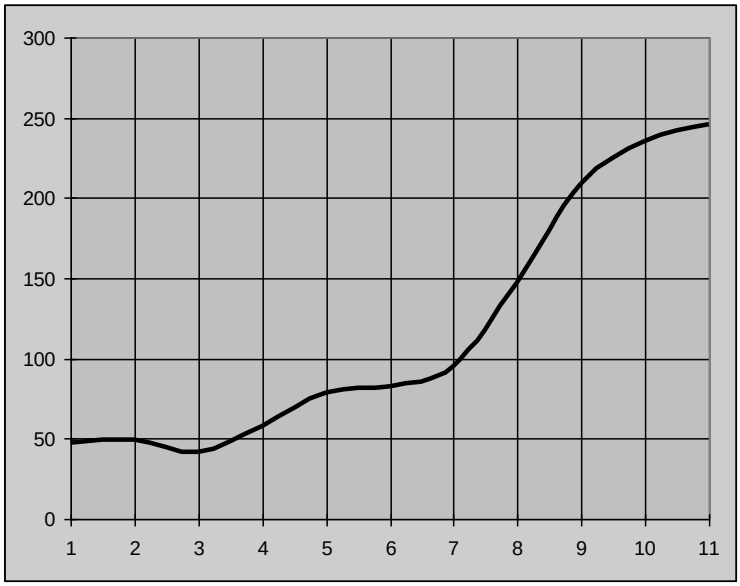
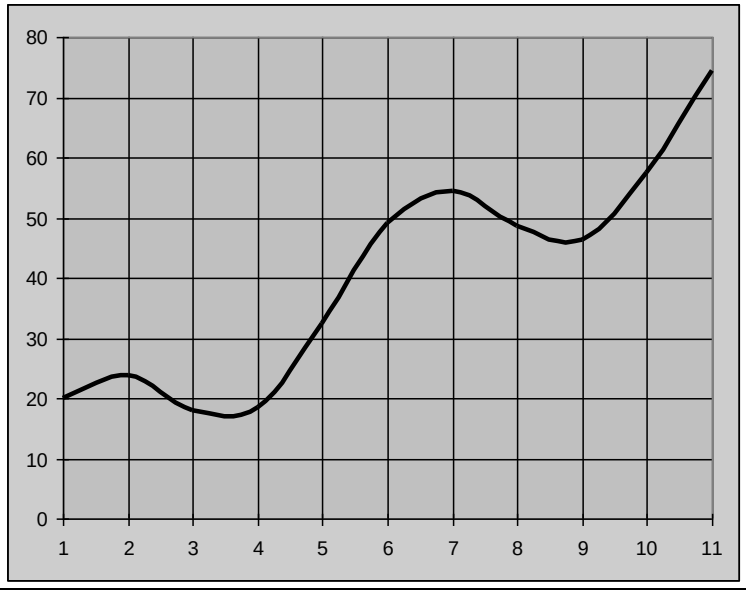
N	Степень	График
1	3	
2	4	
3	5	

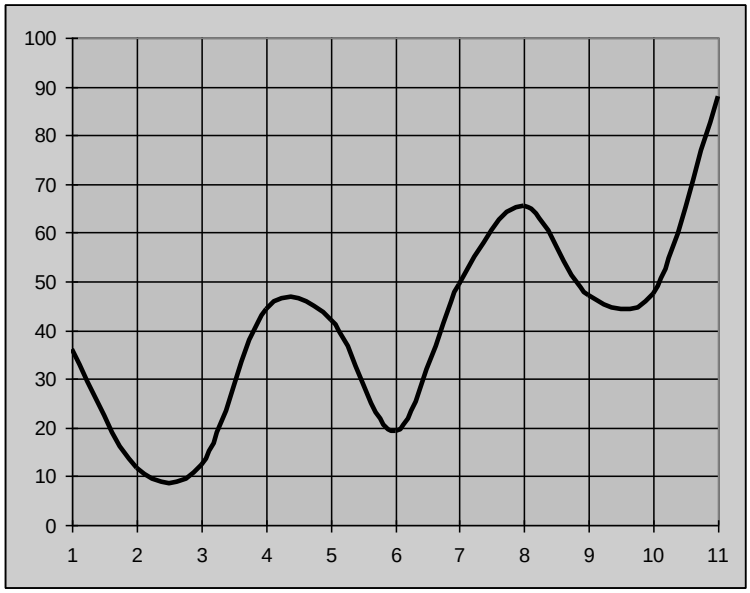
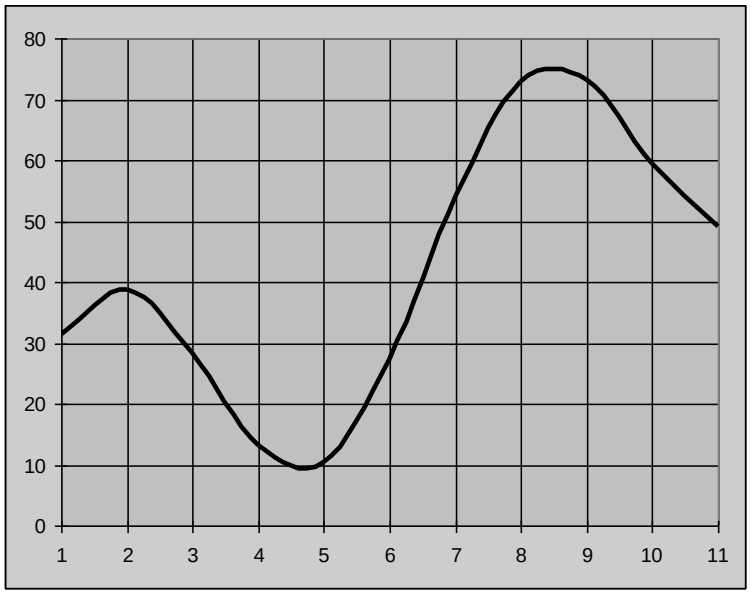
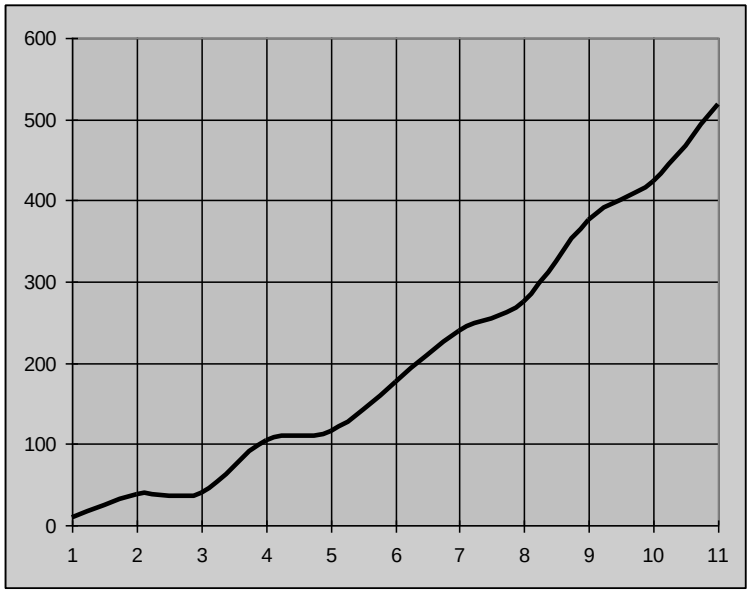
4	6	 300 250 200 150 100 50 0 1 2 3 4 5 6 7 8 9 10 11
5	3	 140 120 100 80 60 40 20 0 1 2 3 4 5 6 7 8 9 10 11
6	4	 60 50 40 30 20 10 0 1 2 3 4 5 6 7 8 9 10 11

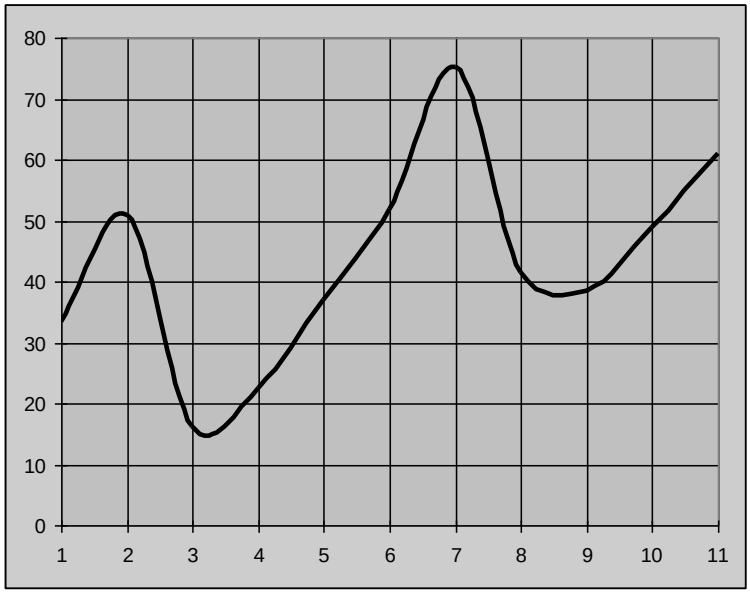
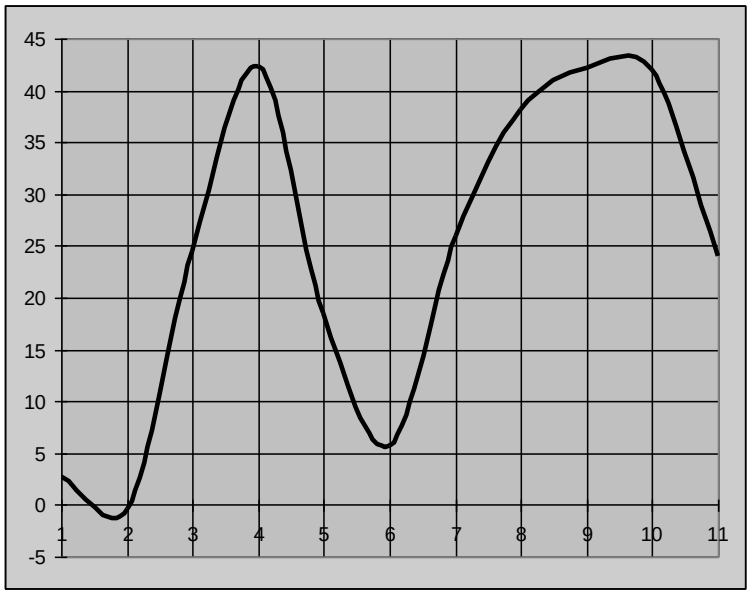
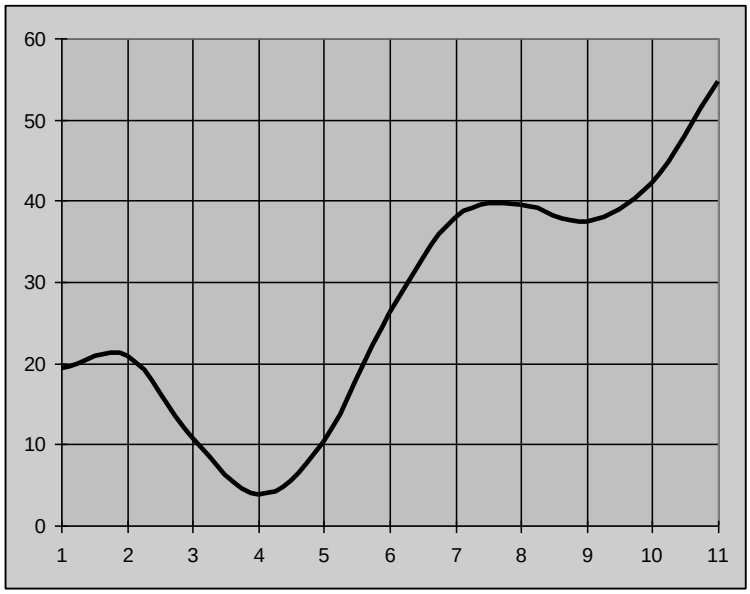
7	5	 50 45 40 35 30 25 20 15 10 5 0 1 2 3 4 5 6 7 8 9 10 11
8	6	 140 120 100 80 60 40 20 0 1 2 3 4 5 6 7 8 9 10 11
9	3	 250 200 150 100 50 0 1 2 3 4 5 6 7 8 9 10 11

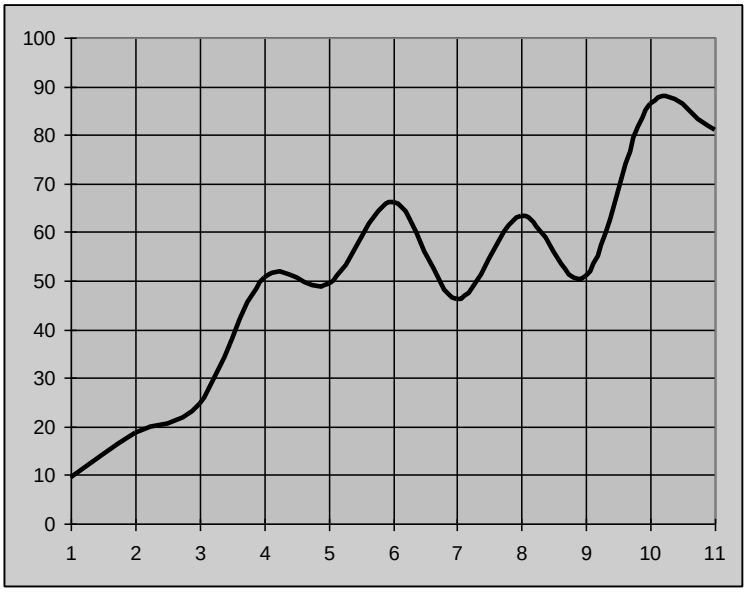
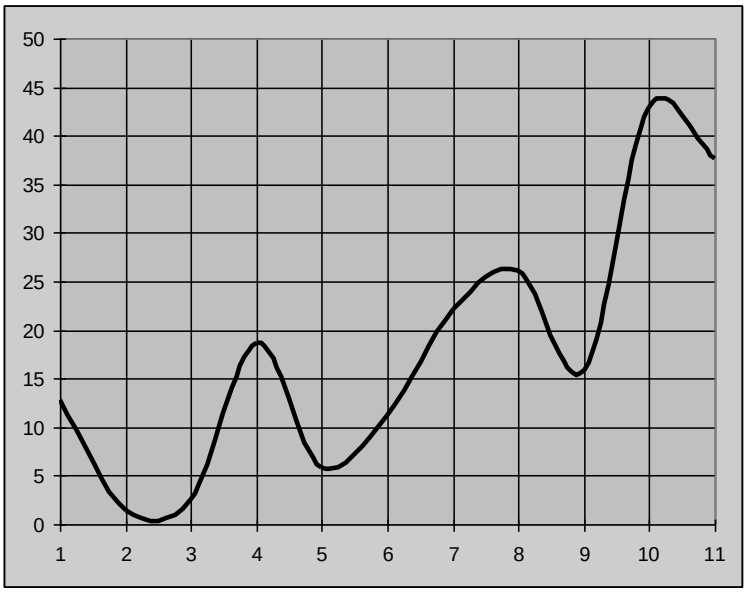
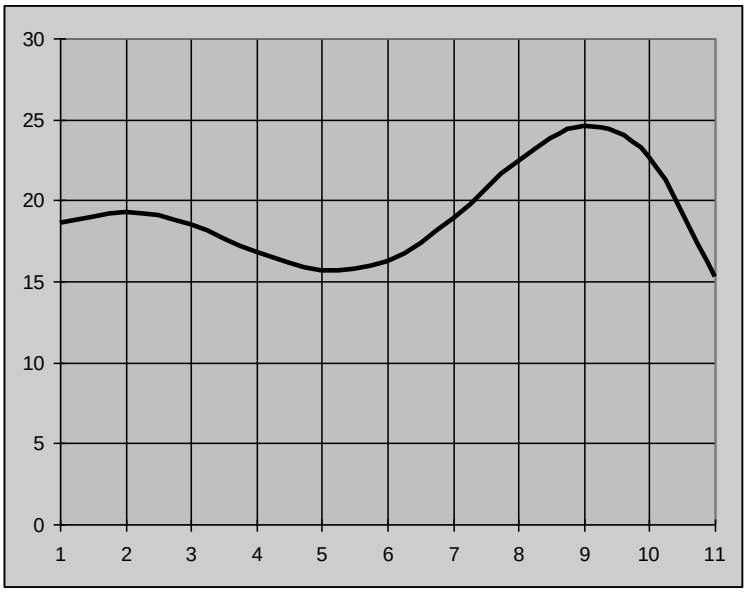
10	4	 160 140 120 100 80 60 40 20 0 1 2 3 4 5 6 7 8 9 10 11
11	5	 140 120 100 80 60 40 20 0 1 2 3 4 5 6 7 8 9 10 11
12	6	 60 50 40 30 20 10 0 1 2 3 4 5 6 7 8 9 10 11

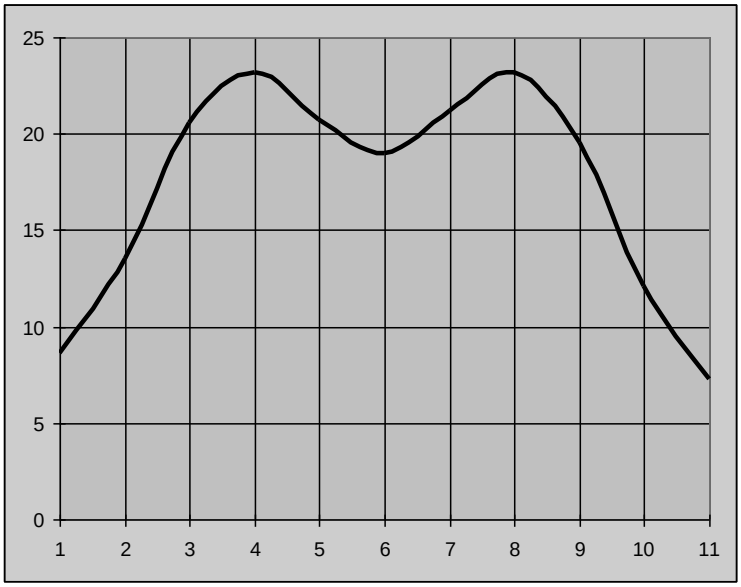
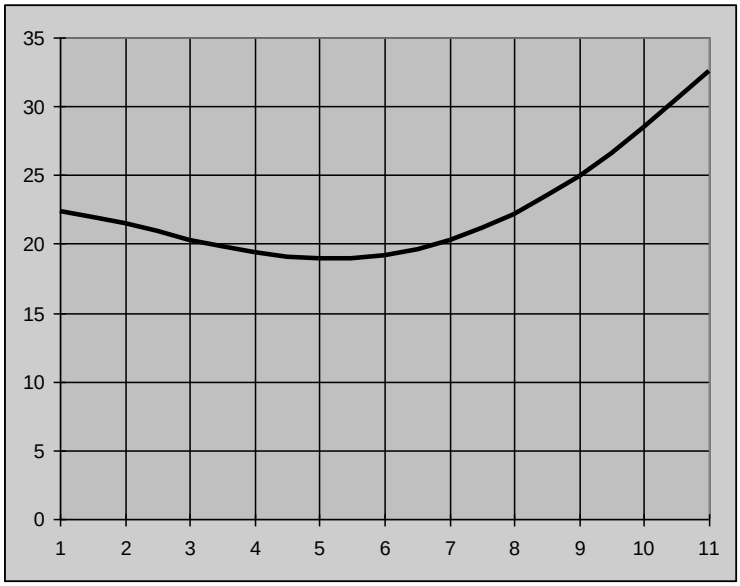
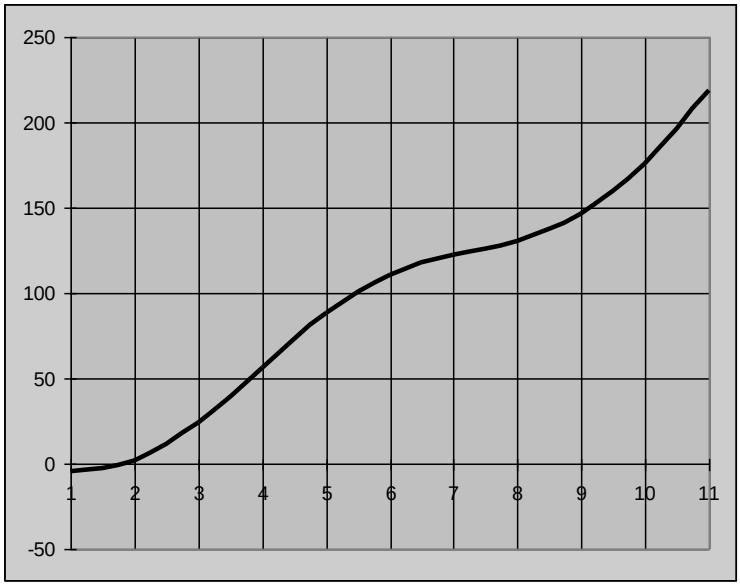
13	3	 120 100 80 60 40 20 0 1 2 3 4 5 6 7 8 9 10 11
14	4	 70 60 50 40 30 20 10 0 1 2 3 4 5 6 7 8 9 10 11
15	5	 60 50 40 30 20 10 0 1 2 3 4 5 6 7 8 9 10 11

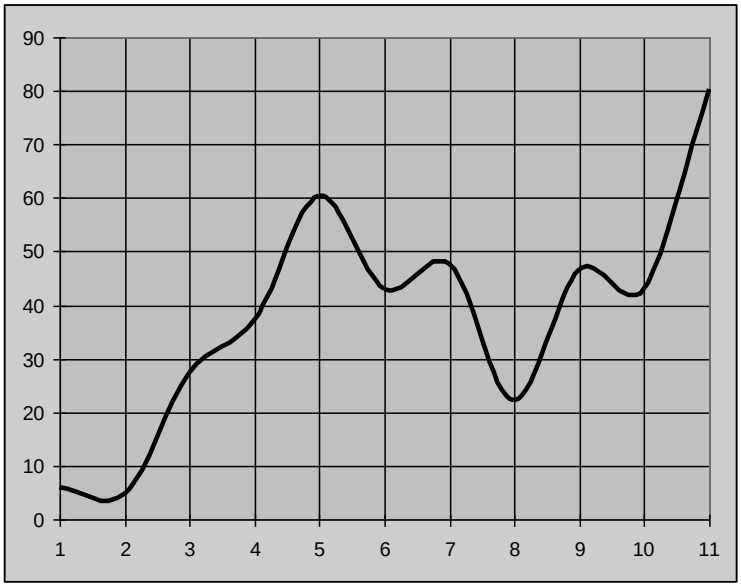
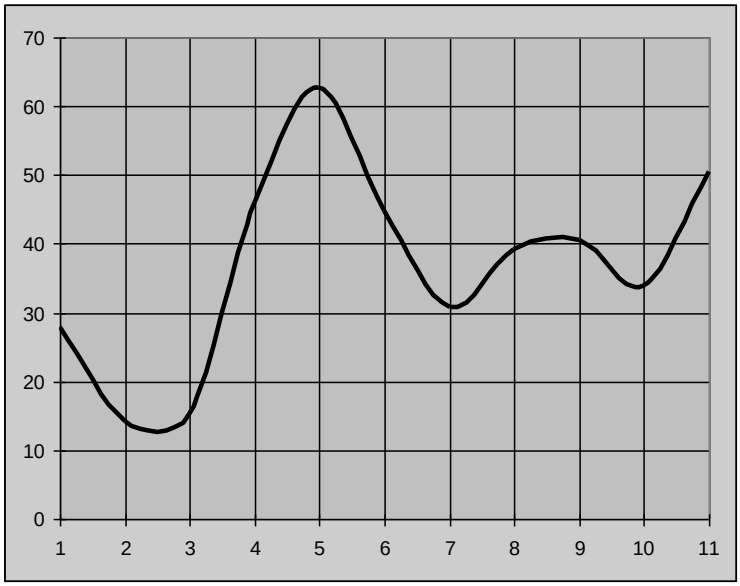
16	6	 80 70 60 50 40 30 20 10 0 1 2 3 4 5 6 7 8 9 10 11
17	3	 300 250 200 150 100 50 0 1 2 3 4 5 6 7 8 9 10 11
18	4	 80 70 60 50 40 30 20 10 0 1 2 3 4 5 6 7 8 9 10 11

19	5	 100 90 80 70 60 50 40 30 20 10 0 1 2 3 4 5 6 7 8 9 10 11
20	6	 80 70 60 50 40 30 20 10 0 1 2 3 4 5 6 7 8 9 10 11
21	3	 600 500 400 300 200 100 0 1 2 3 4 5 6 7 8 9 10 11

22	4	 80 70 60 50 40 30 20 10 0 1 2 3 4 5 6 7 8 9 10 11
23	5	 45 40 35 30 25 20 15 10 5 0 -5 1 2 3 4 5 6 7 8 9 10 11
24	6	 60 50 40 30 20 10 0 1 2 3 4 5 6 7 8 9 10 11

25	3	 100 90 80 70 60 50 40 30 20 10 0 1 2 3 4 5 6 7 8 9 10 11
26	4	 50 45 40 35 30 25 20 15 10 5 0 1 2 3 4 5 6 7 8 9 10 11
27	5	 30 25 20 15 10 5 0 1 2 3 4 5 6 7 8 9 10 11

28	6	 25 20 15 10 5 0 1 2 3 4 5 6 7 8 9 10 11
29	3	 35 30 25 20 15 10 5 0 1 2 3 4 5 6 7 8 9 10 11
30	4	 250 200 150 100 50 0 -50 1 2 3 4 5 6 7 8 9 10 11

31	5	
32	6	

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что представляет собой полиномиальная функция?
2. Объясните принципы проведения аппроксимации?
3. Объясните полученное описание VHDL?
4. Объясните разницу в полученных временных диаграммах?

10.7. Лабораторная работа №6. Проектирование специализированных устройств микро-ЭВМ (ч.1)

Цель работы: изучить базовые принципы построение основных устройств, используемых для построения микро ЭВМ.

Порядок выполнения лабораторной работы:

- ознакомиться со сведениями о графическом редакторе системы Max Plus II, изложенными в пп.10.1;
- получить у преподавателя вариант задания к лабораторной работе;
- рассчитать таблицу истинности для индивидуального задания;
- приступить к проектированию заданного преподавателем варианта;
- реализовать индивидуальное задание в графическом редакторе на языке VHDL в среде Max Plus II;
- протестировать работу полученного описания;
- выполнить размещение полученного решения на кристалле.

Задание: составить схему на основании заданного варианта, выполнить компиляцию и моделирование.

Отчет по лабораторной работе должен содержать:

- Титульный лист установленного образца (Приложение №1).
- Содержание.
- Схема электрическая принципиальная заданного устройства (согласно варианту) с объяснением на вентильно-регистровом уровне.
- Временная диаграмма работы устройства с описанием на уровне входных и выходных сигналов.
- Выводы с указанием на результаты проектирования.
- Файлы проекта в Max Plus II на носителе информации.

Варианты заданий:

Таблица 10.7.– Варианты заданий к лабораторной работе №6

№	*Задание*
1	19-входовый мультиплексор со входом разрешения
2	17-входовый демультиплексор со входом разрешения
3	18-разрядный синхронный суммирующий счетчик с асинхронным сбросом
4	14-разрядный синхронный вычитающий счетчик со входом асинхронной установки
5	12-разрядный синхронный реверсивный счетчик с асинхронным сбросом
6	9-разрядный параллельный сумматор с входом и выходом переноса
7	12-разрядный цифровой компаратор с входами и выходами,

	позволяющими обеспечить наращивание разрядности
8	18-разрядный регистр сдвига с параллельной загрузкой и последовательным выходом
9	18-разрядный регистр сдвига с последовательным входом и параллельным выходом
10	6-разрядную схему контроля четности с параллельной загрузкой и выходом четности
11	20-входовый мультиплексор со входом разрешения
12	16-входовый демультиплексор со входом разрешения
13	17-разрядный синхронный суммирующий счетчик с асинхронным сбросом
14	15-разрядный синхронный вычитающий счетчик со входом асинхронной установки
15	11-разрядный синхронный реверсивный счетчик с асинхронным сбросом
16	10-разрядный параллельный сумматор с входом и выходом переноса
17	11-разрядный цифровой компаратор с входами и выходами, позволяющими обеспечить наращивание разрядности
18	19-разрядный регистр сдвига с параллельной загрузкой и последовательным выходом
19	19-разрядный регистр сдвига с последовательным входом и параллельным выходом
20	7-разрядную схему контроля четности с параллельной загрузкой и выходом четности

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое мультиплексор?
2. Поясните принцип работы демультиплексора на примере более простого
3. Укажите возможные варианты построения регистров.
4. Дайте классификацию счетчиков.
5. Поясните принцип работы сумматора.
6. Какая разница между синхронным и асинхронным сбросом?

10.8. Лабораторная работа №7. Проектирование специализированных устройств микро-ЭВМ (ч.2)

Цель работы: изучить базовые принципы построение основных устройств, используемых для построения микро ЭВМ.

Порядок выполнения лабораторной работы:

- ознакомиться со сведениями о текстовом редакторе системы Max Plus II, изложенными в пп.10.1;
- получить у преподавателя вариант задания к лабораторной работе;
- рассчитать таблицу истинности для индивидуального задания;
- приступить к проектированию заданного преподавателем варианта;
- реализовать индивидуальное задание в текстовом редакторе на языке VHDL в среде Max Plus II;
- протестировать работу полученного описания;
- выполнить размещение полученного решения на кристалле.

Задание: составить схему на основании заданного варианта, выполнить компиляцию и моделирование.

Отчет по лабораторной работе должен содержать:

- Титульный лист установленного образца (Приложение №1).
- Содержание.
- Листинг программы (согласно варианту) с построчным объяснением.
- Временная диаграмма работы устройства с описанием на уровне входных и выходных сигналов.
- Выводы с указанием на результаты проектирования.
- Файлы проекта в Max Plus II на носителе информации.

Варианты заданий:

Таблица 10.8.– Варианты заданий к лабораторной работе №7

№	*Задание*
1	19-входовый мультиплексор со входом разрешения
2	17-входовый демультиплексор со входом разрешения
3	18-разрядный синхронный суммирующий счетчик с асинхронным сбросом
4	14-разрядный синхронный вычитающий счетчик со входом асинхронной установки
5	12-разрядный синхронный реверсивный счетчик с асинхронным сбросом
6	9-разрядный параллельный сумматор с входом и выходом переноса
7	12-разрядный цифровой компаратор с входами и выходами,

	позволяющими обеспечить наращивание разрядности
8	18-разрядный регистр сдвига с параллельной загрузкой и последовательным выходом
9	18-разрядный регистр сдвига с последовательным входом и параллельным выходом
10	6-разрядную схему контроля четности с параллельной загрузкой и выходом четности
11	20-входовый мультиплексор со входом разрешения
12	16-входовый демультиплексор со входом разрешения
13	17-разрядный синхронный суммирующий счетчик с асинхронным сбросом
14	15-разрядный синхронный вычитающий счетчик со входом асинхронной установки
15	11-разрядный синхронный реверсивный счетчик с асинхронным сбросом
16	10-разрядный параллельный сумматор с входом и выходом переноса
17	11-разрядный цифровой компаратор с входами и выходами, позволяющими обеспечить наращивание разрядности
18	19-разрядный регистр сдвига с параллельной загрузкой и последовательным выходом
19	19-разрядный регистр сдвига с последовательным входом и параллельным выходом
20	7-разрядную схему контроля четности с параллельной загрузкой и выходом четности

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое мультиплексор?
2. Поясните принцип работы демультиплексора на примере более простого вида.
3. Укажите возможные варианты построения регистров.
4. Дайте классификацию счетчиков.
5. Поясните принцип работы сумматора.
6. Какая разница между синхронным и асинхронным сбросом?

10.9. Лабораторная работа №8. Проектирование специализированных устройств микро-ЭВМ (ч.3)

Цель работы: изучить базовые принципы построение основных устройств, используемых для построения микро ЭВМ.

Порядок выполнения лабораторной работы:

- ознакомиться со сведениями о графическом редакторе системы Max Plus II, изложенными в пп.10.1;
- получить у преподавателя вариант задания к лабораторной работе;
- рассчитать таблицу истинности для индивидуального задания;
- приступить к проектированию заданного преподавателем варианта;
- реализовать индивидуальное задание в графическом редакторе в среде Max Plus II;
- протестировать работу полученного описания;
- выполнить размещение полученного решения на кристалле.

Задание: составить схему на основании заданного варианта, выполнить компиляцию и моделирование.

Отчет по лабораторной работе должен содержать:

- Титульный лист установленного образца (Приложение №1).
- Содержание.
- Схема электрическая принципиальная заданного устройства (согласно варианту) с объяснением на вентильно-регистровом уровне.
- Временная диаграмма работы устройства с описанием на уровне входных и выходных сигналов.
- Выводы с указанием на результаты проектирования.
- Файлы проекта в Max Plus II на носителе информации.

Варианты заданий:

Таблица 10.9.– Варианты заданий к лабораторной работе №8

№	*Задание*
1	4-разрядный счетчик Джонсона с выходным дешифратором, формирующим двоичный код
2	14-разрядный регистр хранения с параллельной загрузкой и Z-состоянием выходной шины
3	Цифровая линия задержки 8-разрядного слова на 12 тактов с параллельными входом и выходом
4	АЛУ, обеспечивающее сложение или вычитание 4-разрядных операндов в зависимости от состояния управляющего входа.
5	Формирователь пачек импульсов, длина которых задается 4-х разрядным

	словом (тактирование от внешнего синхросигнала)
6	Двоично-десятичный синхронный счетчик емкостью 100
7	Делитель частоты на 37
8	Устройство, выделяющее каждый пятый бит из последовательного цифрового сигнала
9	10-разрядный реверсивный регистр сдвига с параллельной загрузкой
10	6-разрядную схему контроля четности с параллельной загрузкой и выходом нечетности
11	6-разрядный счетчик Джонсона с выходным дешифратором, формирующим двоичный код
12	15-разрядный регистр хранения с параллельной загрузкой и Z-состоянием выходной шины
13	Цифровая линия задержки 8-разрядного слова на 8 тактов с параллельными входом и выходом
14	АЛУ, обеспечивающее сложение или вычитание 3-разрядных операндов в зависимости от состояния управляющего входа.
15	Формирователь пачек импульсов, длина которых задается 5-х разрядным словом (тактирование от внешнего синхросигнала)
16	Двоично-десятичный синхронный счетчик емкостью 128
17	Делитель частоты на 31
18	Устройство, выделяющее каждый шестой бит из последовательного цифрового сигнала
19	11-разрядный реверсивный регистр сдвига с параллельной загрузкой
20	7-разрядную схему контроля четности с параллельной загрузкой и выходом нечетности

КОНТРОЛЬНЫЕ ВОПРОСЫ

7. Что такое мультиплексор?
8. Поясните принцип работы демультиплексора на примере более простого
9. Укажите возможные варианты построения регистров.
10. Дайте классификацию счетчиков.
11. Поясните принцип работы сумматора.
12. Какая разница между синхронным и асинхронным сбросом?

10.10. Лабораторная работа №9. Проектирование специализированных устройств микро-ЭВМ (ч.4)

Цель работы: изучить базовые принципы построение основных устройств, используемых для построения микро ЭВМ.

Порядок выполнения лабораторной работы:

- ознакомиться со сведениями о текстовом редакторе системы Max Plus II, изложенными в пп.10.1;
- получить у преподавателя вариант задания к лабораторной работе;
- рассчитать таблицу истинности для индивидуального задания;
- приступить к проектированию заданного преподавателем варианта;
- реализовать индивидуальное задание в текстовом редакторе на языке VHDL в среде Max Plus II;
- протестировать работу полученного описания;
- выполнить размещение полученного решения на кристалле.

Задание: составить схему на основании заданного варианта, выполнить компиляцию и моделирование.

Отчет по лабораторной работе должен содержать:
- Титульный лист установленного образца (Приложение №1).
- Содержание.
- Листинг программы (согласно варианту) с построчным объяснением.
- Временная диаграмма работы устройства с описанием на уровне входных и выходных сигналов.
- Выводы с указанием на результаты проектирования.
- Файлы проекта в Max Plus II на носителе информации.

Варианты заданий:

Таблица 10.10.– Варианты заданий к лабораторной работе №9

№	*Задание*
1	4-разрядный счетчик Джонсона с выходным дешифратором, формирующим двоичный код
2	14-разрядный регистр хранения с параллельной загрузкой и Z-состоянием выходной шины
3	Цифровая линия задержки 8-разрядного слова на 12 тактов с параллельными входом и выходом
4	АЛУ, обеспечивающее сложение или вычитание 4-разрядных операндов в зависимости от состояния управляющего входа.
5	Формирователь пачек импульсов, длина которых задается 4-х разрядным

	словом (тактирование от внешнего синхросигнала)
6	Двоично-десятичный синхронный счетчик емкостью 100
7	Делитель частоты на 37
8	Устройство, выделяющее каждый пятый бит из последовательного цифрового сигнала
9	10-разрядный реверсивный регистр сдвига с параллельной загрузкой
10	6-разрядную схему контроля четности с параллельной загрузкой и выходом нечетности
11	6-разрядный счетчик Джонсона с выходным дешифратором, формирующим двоичный код
12	15-разрядный регистр хранения с параллельной загрузкой и Z-состоянием выходной шины
13	Цифровая линия задержки 8-разрядного слова на 8 тактов с параллельными входом и выходом
14	АЛУ, обеспечивающее сложение или вычитание 3-разрядных операндов в зависимости от состояния управляющего входа.
15	Формирователь пачек импульсов, длина которых задается 5-х разрядным словом (тактирование от внешнего синхросигнала)
16	Двоично-десятичный синхронный счетчик емкостью 128
17	Делитель частоты на 31
18	Устройство, выделяющее каждый шестой бит из последовательного цифрового сигнала
19	11-разрядный реверсивный регистр сдвига с параллельной загрузкой
20	7-разрядную схему контроля четности с параллельной загрузкой и выходом нечетности

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое мультиплексор?
2. Поясните принцип работы демультиплексора на примере более простого вида.
3. Укажите возможные варианты построения регистров.
4. Дайте классификацию счетчиков.
5. Поясните принцип работы сумматора.
6. Какая разница между синхронным и асинхронным сбросом?

10.11. Лабораторная работа №10. Проектирование специализированных устройств микро-ЭВМ (ч.5)

Цель работы: изучить базовые принципы построение основных устройств, используемых для построения микро ЭВМ.

Порядок выполнения лабораторной работы:

- ознакомиться со сведениями о графическом редакторе системы Max Plus II, изложенными в пп.10.1;
- получить у преподавателя вариант задания к лабораторной работе;
- рассчитать таблицу истинности для индивидуального задания;
- приступить к проектированию заданного преподавателем варианта;
- реализовать индивидуальное задание в графическом редакторе в среде Max Plus II;
- протестировать работу полученного описания;
- выполнить размещение полученного решения на кристалле.

Задание: составить схему на основании заданного варианта, выполнить компиляцию и моделирование.

Отчет по лабораторной работе должен содержать:

- Титульный лист установленного образца (Приложение №1).
- Содержание.
- Схема электрическая принципиальная заданного устройства (согласно варианту) с объяснением на вентильно-регистровом уровне.
- Временная диаграмма работы устройства с описанием на уровне входных и выходных сигналов.
- Выводы с указанием на результаты проектирования.
- Файлы проекта в Max Plus II на носителе информации.

Варианты заданий:

Таблица 10.11.– Варианты заданий к лабораторной работе №10

№	*Задание*
1	12-разрядный регистр хранения с общей шиной для чтения и записи
2	Одновибратор, длительность импульса, которого задается параллельным 4-разрядным кодом (тактовый сигнал - внешний)
3	13-разрядный синхронный счетчик с Z-состоянием на выходе и асинхронным сбросом
4	18-разрядный измеритель временных интервалов с запуском по фронту, остановкой счета по срезу и асинхронным сбросом
5	18-разрядный "Бегущий огонь" с внешним тактированием и 8 режимами, задаваемыми внешним параллельным сигналом

6	16-разрядный счетчик импульсов, поступающих по 3-м независимым линиям. Необходим асинхронный сброс
7	Формирователь секундных, минутных и часовых импульсов из входного тактового сигнала частотой 32768 Гц
8	АЛУ, обеспечивающее сравнение или вычитание 5-разрядных операндов в зависимости от состояния управляющего входа.
9	12-разрядный двоичный счетчик с параллельной загрузкой и чтением по одной шине
10	Мультиплексор структуры 12x4 с Z-состоянием на выходе
11	11-разрядный регистр хранения с общей шиной для чтения и записи
12	Одновибратор, длительность импульса которого задается параллельным 5-разрядным кодом (тактовый сигнал - внешний)
13	10-разрядный синхронный счетчик с Z-состоянием на выходе и асинхронным сбросом
14	19-разрядный измеритель временных интервалов с запуском по фронту, остановкой счета по срезу и асинхронным сбросом
15	20-разрядный "Бегущий огонь" с внешним тактированием и 8 режимами, задаваемыми внешним параллельным сигналом
16	17-разрядный счетчик импульсов, поступающих по 3-м независимым линиям. Необходим асинхронный сброс
17	Формирователь секундных, минутных и часовых импульсов из входного тактового сигнала частотой 16339 Гц
18	АЛУ, обеспечивающее сравнение или вычитание 8-разрядных операндов в зависимости от состояния управляющего входа.
19	18-разрядный двоичный счетчик с параллельной загрузкой и чтением по одной шине
20	Мультиплексор структуры 12x8 с Z-состоянием на выходе

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как работает “бегущий огонь”?
2. Поясните принцип работы мультиплексор структуры RxX на примере более простого.
3. Укажите принципы построения регистров.
4. Дайте классификацию сумматоров.
5. Поясните принцип работы сумматора.
6. Какая разница между синхронным и асинхронным сбросом?

10.12. Лабораторная работа №11. Проектирование специализированных устройств микро-ЭВМ (ч.6)

Цель работы: изучить базовые принципы построение основных устройств, используемых для построения микро ЭВМ.

Порядок выполнения лабораторной работы:

- ознакомиться со сведениями о текстовом редакторе системы Max Plus II, изложенными в пп.10.1;
- получить у преподавателя вариант задания к лабораторной работе;
- рассчитать таблицу истинности для индивидуального задания;
- приступить к проектированию заданного преподавателем варианта;
- реализовать индивидуальное задание в текстовом редакторе на языке VHDL в среде Max Plus II;
- протестировать работу полученного описания;
- выполнить размещение полученного решения на кристалле.

Задание: составить схему на основании заданного варианта, выполнить компиляцию и моделирование.

Отчет по лабораторной работе должен содержать:

- Титульный лист установленного образца (Приложение №1).
- Содержание.
- Листинг программы (согласно варианту) с построчным объяснением.
- Временная диаграмма работы устройства с описанием на уровне входных и выходных сигналов.
- Выводы с указанием на результаты проектирования.
- Файлы проекта в Max Plus II на носителе информации.

Варианты заданий:

Таблица 10.12.– Варианты заданий к лабораторной работе №11

№	*Задание*
1	12-разрядный регистр хранения с общей шиной для чтения и записи
2	Одновибратор, длительность импульса, которого задается параллельным 4-разрядным кодом (тактовый сигнал - внешний)
3	13-разрядный синхронный счетчик с Z-состоянием на выходе и асинхронным сбросом
4	18-разрядный измеритель временных интервалов с запуском по фронту, остановкой счета по срезу и асинхронным сбросом
5	18-разрядный "Бегущий огонь" с внешним тактированием и 8 режимами, задаваемыми внешним параллельным сигналом

6	16-разрядный счетчик импульсов, поступающих по 3-м независимым линиям. Необходим асинхронный сброс
7	Формирователь секундных, минутных и часовых импульсов из входного тактового сигнала частотой 32768 Гц
8	АЛУ, обеспечивающее сравнение или вычитание 5-разрядных операндов в зависимости от состояния управляющего входа.
9	12-разрядный двоичный счетчик с параллельной загрузкой и чтением по одной шине
10	Мультиплексор структуры 12x4 с Z-состоянием на выходе
11	11-разрядный регистр хранения с общей шиной для чтения и записи
12	Одновибратор, длительность импульса которого задается параллельным 5-разрядным кодом (тактовый сигнал - внешний)
13	10-разрядный синхронный счетчик с Z-состоянием на выходе и асинхронным сбросом
14	19-разрядный измеритель временных интервалов с запуском по фронту, остановкой счета по срезу и асинхронным сбросом
15	20-разрядный "Бегущий огонь" с внешним тактированием и 8 режимами, задаваемыми внешним параллельным сигналом
16	17-разрядный счетчик импульсов, поступающих по 3-м независимым линиям. Необходим асинхронный сброс
17	Формирователь секундных, минутных и часовых импульсов из входного тактового сигнала частотой 16339 Гц
18	АЛУ, обеспечивающее сравнение или вычитание 8-разрядных операндов в зависимости от состояния управляющего входа.
19	18-разрядный двоичный счетчик с параллельной загрузкой и чтением по одной шине
20	Мультиплексор структуры 12x8 с Z-состоянием на выходе

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Как работает “бегущий огонь”?
2. Поясните принцип работы мультиплексор структуры RxX на примере более простого.
3. Укажите принципы построения регистров.
4. Дайте классификацию сумматоров.
5. Поясните принцип работы сумматора.
6. Какая разница между синхронным и асинхронным сбросом?

10.13. Лабораторная работа №12. Проектирование АЛУ.

Цель работы: ознакомиться с функциями реализуемыми простейшим АЛУ. Получить практические навыки в реализации и тестировании АЛУ на языке VHDL в среде VHDL.

Краткие сведения из теории:

1. Общие сведения о проектируемом АЛУ
   1.1. Простейшее АЛУ содержит:
      1.1.1. Два входа для операндов;
      1.1.2. Вход для указания выполняемой операции;
      1.1.3. Выход для результата;
      1.1.4. Выходы для флагов;
      1.1.5. Блоки выполняющие операции;
      1.1.6. Блоки устанавливающие значения флагов.
   1.2. Простейшее АЛУ реализует:
      1.2.1. Суммирование;
      1.2.2. Вычитание;
      1.2.3. Умножение;
      1.2.4. Побитовое И;
      1.2.5. Побитовое ИЛИ;
      1.2.6. Побитовое исключающее ИЛИ;
      1.2.7. Побитовое НЕ;
      1.2.8. Циклический сдвиг влево, через перенос;
      1.2.9. Циклический сдвиг вправо, через перенос;
      1.2.10. Арифметический сдвиг влево, через перенос;
      1.2.11. Арифметический сдвиг вправо, через перенос.
   1.3. Простейшее АЛУ устанавливает значения флагов:
      1.3.1. Z – флаг нулевого результата;
      1.3.2. CY – флаг переноса/заема;
      1.3.3. OV – флаг переполнения;
      1.3.4. P – флаг четности.
2. Добавить в новый проект файл с исходным VHDL-кодом из приложения 1 и файл с тестовым модулем из приложения 2
3. Реализовать в VHDL-коде АЛУ необходимые операции, с учетом влияния результата на значения флагов:

Операция / Затрагиваемые флаги	+	-	*	And	Or	Xor	Not	Rol	Ror	Shl	Shr
Z	□	□	□	□	□	□	□				
CY	□	□						□	□	□	□
OV			□								
P	□	□	□	□	□	□	□	□	□	□	□

4. Исключить из тестового модуля тесты для операций, которые АЛУ не должно реализовать.
5. Полностью выполнить тестовый модуль и получить сообщение о правильной работе АЛУ.

Порядок выполнения работы:

1. Ознакомиться с требованиями методических указаний по выполнению лабораторной работы.
2. Изучить требования, предъявляемые к простейшим АЛУ.
3. Изучить возможности языка VHDL применяемые для реализации АЛУ.
4. Реализовать процедуры для установки флагов Z, CY, OV по результату выполнения операции.
5. Реализовать выполнение операций заданных в индивидуальном задании.
6. Модифицировать тестовый модуль и выполнить тестирование полученного кода.
7. Оформить отчет.

Требования к содержанию и оформлению отчета:

1. Титульный лист (согласно Приложения 1).
2. Индивидуальное задание на лабораторную работу.
3. Содержание.
4. Исходный VHDL-текст реализованного АЛУ.
5. Исходный код реализованного тестового модуля.
6. Копия текста из консоли VHDL с результатом тестирования.
7. Вывод по работе, с учетом индивидуального задания.

Исходный код простейшего АЛУ с несколькими реализованными функциями:

```
-- реализация простейшего АЛУ

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

-- порты АЛУ
-- reset - сброс
-- op_code - код операции
-- in1 - первый операнд
-- in2 - второй операнд
```

```
-- out1 - результат
-- flag_z - флаг нулевого результата
-- flag_cy - флаг переноса
-- flag_ov - флаг переполнения
entity alu is
   port(
      reset   : in std_logic;
      op_code : in  std_logic_vector (3 downto 0);
      in1     : in  std_logic_vector (7 downto 0);
      in2     : in  std_logic_vector (7 downto 0);
      out1    : out std_logic_vector (7 downto 0);
      flag_z  : out std_logic;
      flag_cy : out std_logic;
      flag_ov : out std_logic);
end alu;

architecture behaviour of alu is

--Коды операций, которые выполняет АЛУ
constant  alu_op_add : std_logic_vector (3 downto 0)  := "0001";
constant  alu_op_sub : std_logic_vector (3 downto 0)  := "0010";
constant  alu_op_mul : std_logic_vector (3 downto 0)  := "0011";
constant  alu_op_and : std_logic_vector (3 downto 0)  := "0100";
constant  alu_op_or  : std_logic_vector (3 downto 0)  := "0101";
constant  alu_op_xor : std_logic_vector (3 downto 0)  := "0110";
constant  alu_op_ror : std_logic_vector (3 downto 0)  := "0111";
constant  alu_op_rol : std_logic_vector (3 downto 0)  := "1000";
constant  alu_op_not : std_logic_vector (3 downto 0)  := "1001";

-- Процедура вычисляющая по значению результата значение флага OV
procedure calc_overflow_flag (
   result         : in  std_logic_vector (15 downto 0);
   overflow_flag   : out std_logic ) is

   begin

            -- Проверяются старшие 8 бит результата
      if( result( 15 downto 8 ) /= "00000000" ) then
                  -- Если они не нулевые, то установить флаг переполнения
         overflow_flag := '1';
      else
                  -- Если они нулевые, то сбросить флаг переполнения
         overflow_flag := '0';
      end if;
```

```
    end calc_overflow_flag;


begin
    process(
    reset,
    op_code,
    in1,
    in2 )

        -- Временные переменные для хранения значений внутри процесса
        variable tmp_in1, tmp_in2 : std_logic_vector (15 downto 0);
        variable res : std_logic_vector (15 downto 0);
        variable res_z, res_cy, res_ov : std_logic;
    begin

    if reset = '0' then

        -- Обнуляем внутренние переменные и записываем в них
        -- входные значения
        -- Внутренние переменные имеют вдвое большую разрядность, чем
        -- входные, для того, что бы иметь возможность устанавливать
        -- флаги
        tmp_in1 := "0000000000000000";
        tmp_in2 := "0000000000000000";
        tmp_in1( 7 downto 0 ) := in1;
        tmp_in2( 7 downto 0 ) := in2;

        -- Определяем какой код операции установлен
        case op_code is
            when alu_op_add =>
            -- Установлен код операции сложения
            -- Выполняем сложение
            res := tmp_in1 + tmp_in2;

            when alu_op_mul =>
            -- Умножение
                        -- Максимальное значение результата умножения
                        -- имеет разрядность в 2 раза больше чем операнды,
                        -- поэтому в качестве множителей берутся не 16-разрядные
                        -- tmp_in1 и tmp_in1, а 8 разрядные in1 и in2
            res := in1 * in2;
            -- Устанавливаем флаг OV
```

```
            calc_overflow_flag( res, res_ov );

            when alu_op_rol =>
            -- Циклический сдвиг влево
            res( 0 ) := tmp_in1( 7 );
            res( 7 downto 1 ) := tmp_in1( 6 downto 0 );

            when others =>
            -- Неизвестный код операции
            res := "ZZZZZZZZZZZZZZZZ";

        end case;

    end if;

    -- Выдаем в порты 8 младших бит результата и
    -- значения флагов
    out1 <= res( 7 downto 0 );
    flag_z <= res_z;
    flag_cy <= res_cy;
    flag_ov <= res_ov;

end process;

end behaviour;
```

<u>Тестовый модуль для АЛУ:</u>

```
library ieee;
use ieee.std_logic_arith.all;
use ieee.std_logic_1164.all;

    -- Add your library and packages declaration here ...

entity alu_tb is
end alu_tb;

architecture TB_ARCHITECTURE of alu_tb is
    -- Component declaration of the tested unit
--Коды операций, которые выполняет АЛУ
constant  alu_op_add : std_logic_vector (3 downto 0)  := "0001";
constant  alu_op_sub : std_logic_vector (3 downto 0)  := "0010";
constant  alu_op_mul : std_logic_vector (3 downto 0)  := "0011";
constant  alu_op_and : std_logic_vector (3 downto 0)  := "0100";
```

```
constant  alu_op_or  : std_logic_vector (3 downto 0)  := "0101";
constant  alu_op_xor : std_logic_vector (3 downto 0)  := "0110";
constant  alu_op_ror : std_logic_vector (3 downto 0)  := "0111";
constant  alu_op_rol : std_logic_vector (3 downto 0)  := "1000";
constant  alu_op_not : std_logic_vector (3 downto 0)  := "1001";

   component alu
   port(
      reset : in std_logic;
      op_code : in std_logic_vector(3 downto 0);
      in1 : in std_logic_vector(7 downto 0);
      in2 : in std_logic_vector(7 downto 0);
      out1 : out std_logic_vector(7 downto 0);
      flag_z : out std_logic;
      flag_cy : out std_logic;
      flag_ov : out std_logic );
   end component;

   -- Stimulus signals - signals mapped to the input and inout ports of tested entity
   signal reset : std_logic;
   signal op_code : std_logic_vector(3 downto 0);
   signal in1 : std_logic_vector(7 downto 0);
   signal in2 : std_logic_vector(7 downto 0);
   -- Observed signals - signals mapped to the output ports of tested entity
   signal out1 : std_logic_vector(7 downto 0);
   signal flag_z : std_logic;
   signal flag_cy : std_logic;
   signal flag_ov : std_logic;

   -- Add your code here ...

begin

   -- Unit Under Test port map
   UUT : alu
      port map (
         reset => reset,
         op_code => op_code,
         in1 => in1,
         in2 => in2,
         out1 => out1,
         flag_z => flag_z,
         flag_cy => flag_cy,
         flag_ov => flag_ov
```

```
    );

  -- Add your stimulus here ...
-- Процесс, генерирующий тестовые последовательности
test_data_generator:
process

variable result : std_logic_vector (7 downto 0);
variable right_result : std_logic_vector (7 downto 0);

begin

  -- Подаем Reset на АЛУ -- пока не выполняются никакие действия
  reset <= '1';

  -- Тестирование сложения без переноса
  -- Задаем код операции
  op_code <= alu_op_add;
  -- Задаем первый операнд
  in1 <= "00001100";
  -- Задаем второй операнд
  in2 <= "00010110";
  -- Сбрасываем Reset, тем самым запуская работу АЛУ
  reset <= '0';
  -- Пауза
  wait for 50ns;

  -- Записываем в одну переменную полученный результат,
  -- а в другую правильный результат
  result := out1;
  right_result := "00100010";

  -- Содержимое блока assert будет выполнено только тогда
  -- когда нарушается указанное условие. Т.е. если полученный
  -- результат равен правильному и флаг CY так же установлен
  -- правильно, блок assert выполнен не будет
  assert ( result = right_result ) and ( flag_cy = '0' )
  -- Выводим сообщение о том что АЛУ функционирует неправильно
  report "Сложение без переноса выполняется неправильно"
  -- Установлен самый жесткий тип проверки assert -- failure
  -- В случае если условие неистинно, выполнение моделирования
  -- будет прервано
  severity failure;
```

```
-- Снова останавливаем работу АЛУ
reset <= '1';

-- Тестирование сложения с переносом
op_code <= alu_op_add;
in1 <= "10001100";
in2 <= "10010110";
reset <= '0';
wait for 50ns;

result := out1;
right_result := "00100010";
assert ( result = right_result ) and ( flag_cy = '1' )
report "Сложение с переносом выполняется неправильно"
severity failure;

reset <= '1';

-- Тестирование вычитания без установки флага Z
op_code <= alu_op_sub;
in1 <= "00010100";
in2 <= "00001100";
reset <= '0';
wait for 50ns;

result := out1;
right_result := "00001000";
assert ( result = right_result ) and ( flag_z = '0' )
report "Вычитание без установки флага Z выполняется неправильно"
severity failure;

reset <= '1';

-- Тестирование вычитания с установкой флага Z
op_code <= alu_op_sub;
in1 <= "00010100";
in2 <= "00010100";
reset <= '0';
wait for 50ns;

result := out1;
right_result := "00000000";
assert ( result = right_result ) and ( flag_z = '1' )
report "Вычитание с установкой флага Z выполняется неправильно"
```

```
severity failure;

reset <= '1';

-- Тестирование умножения без переполнения
op_code <= alu_op_mul;
in1 <= "00010100";
in2 <= "00001100";
reset <= '0';
wait for 50ns;

result := out1;
right_result := "11110000";
assert ( result = right_result ) and ( flag_ov = '0' )
report "Умножение без переполнения выполняется неправильно"
severity failure;

reset <= '1';

-- Тестирование умножения с переполнением
op_code <= alu_op_mul;
in1 <= "00110100";
in2 <= "00001100";
reset <= '0';
wait for 50ns;

result := out1;
right_result := "01110000";
assert ( result = right_result ) and ( flag_ov = '1' )
report "Умножение с переполнением выполняется неправильно"
severity failure;

reset <= '1';

-- Тестирование побитового И без установки флага Z
op_code <= alu_op_and;
in1 <= "00110100";
in2 <= "00001100";
reset <= '0';
wait for 50ns;

result := out1;
right_result := "00000100";
assert ( result = right_result ) and ( flag_z = '0' )
```

```
report "Побитовое И без установки флага Z выполняется неправильно"
severity failure;

reset <= '1';

-- Тестирование побитового И с установкой флага Z
op_code <= alu_op_and;
in1 <= "00110100";
in2 <= "00001000";
reset <= '0';
wait for 50ns;

result := out1;
right_result := "00000000";
assert ( result = right_result ) and ( flag_z = '1' )
report "Побитовое И с установкой флага Z выполняется неправильно"
severity failure;

reset <= '1';


-- Тестирование побитового ИЛИ без установки флага Z
op_code <= alu_op_or;
in1 <= "00110100";
in2 <= "00001100";
reset <= '0';
wait for 50ns;

result := out1;
right_result := "00111100";
assert ( result = right_result ) and ( flag_z = '0' )
report "Побитовое ИЛИ без установки флага Z выполняется неправильно"
severity failure;

reset <= '1';

-- Тестирование побитового ИЛИ с установкой флага Z
op_code <= alu_op_or;
in1 <= "00000000";
in2 <= "00000000";
reset <= '0';
wait for 50ns;

result := out1;
```

```
    right_result := "00000000";
    assert ( result = right_result ) and ( flag_z = '1' )
    report "Побитовое ИЛИ с установкой флага Z выполняется неправильно"
    severity failure;

    reset <= '1';


    -- Тестирование побитового исключающего ИЛИ без установки флага Z
    op_code <= alu_op_xor;
    in1 <= "00110100";
    in2 <= "00001100";
    reset <= '0';
    wait for 50ns;

    result := out1;
    right_result := "00111000";
    assert ( result = right_result ) and ( flag_z = '0' )
    report "Побитовое исключающее ИЛИ без установки флага Z выполняется
неправильно"
    severity failure;

    reset <= '1';

    -- Тестирование побитового исключающего ИЛИ с установкой флага Z
    op_code <= alu_op_xor;
    in1 <= "00110100";
    in2 <= "00110100";
    reset <= '0';
    wait for 50ns;

    result := out1;
    right_result := "00000000";
    assert ( result = right_result ) and ( flag_z = '1' )
    report "Побитовое исключающее ИЛИ с установкой флага Z выполняется
неправильно"
    severity failure;

    reset <= '1';

    -- Тестирование побитового НЕ без установки флага Z
    op_code <= alu_op_not;
    -- Операция имеет только один операнд. Значение in2
    -- несущественно
```

```
in1 <= "00110100";
reset <= '0';
wait for 50ns;

result := out1;
right_result := "11001011";
assert ( result = right_result ) and ( flag_z = '0' )
report "Побитовое НЕ без установки флага Z выполняется неправильно"
severity failure;

reset <= '1';

-- Тестирование побитового НЕ с установкой флага Z
op_code <= alu_op_not;
in1 <= "11111111";
reset <= '0';
wait for 50ns;

result := out1;
right_result := "00000000";
assert ( result = right_result ) and ( flag_z = '1' )
report "Побитовое НЕ с установкой флага Z выполняется неправильно"
severity failure;

reset <= '1';

-- Тестирование циклического сдвига влево
op_code <= alu_op_rol;
in1 <= "10110100";
reset <= '0';
wait for 50ns;

result := out1;
right_result := "01101001";
assert ( result = right_result )
report "Циклический сдвиг влево выполняется неправильно"
severity failure;

reset <= '1';

-- Тестирование циклического сдвига вправо
op_code <= alu_op_ror;
in1 <= "00110101";
reset <= '0';
```

```
    wait for 50ns;

    result := out1;
    right_result := "10011010";
    assert ( result = right_result )
    report "Циклический сдвиг вправо выполняется неправильно"
    severity failure;

    reset <= '1';


    report "Все тесты пройдены! Поздравлем! :-)";
    wait;

end process;

end TB_ARCHITECTURE;

configuration TESTBENCH_FOR_alu of alu_tb is
    for TB_ARCHITECTURE
        for UUT : alu
            use entity work.alu(behaviour);
        end for;
    end for;
end TESTBENCH_FOR_alu;
```

Варианты заданий приведены в таблице 10.13

Таблица 10.13.– Варианты заданий к лабораторной работе №12.

Вариант	Задание 1	Задание 2	Задание 3
1	+	and	ror
2	-	and	ror
3	*	and	ror
4	+	or	ror
5	-	or	ror
6	*	or	ror
7	+	xor	ror
8	-	xor	ror
9	*	xor	rol
10	+	not	rol
11	-	not	rol
12	*	not	rol
13	+	and	rol
14	-	and	rol
15	*	and	rol

16	+	or	rol
17	-	or	shr
18	*	or	shr
19	+	xor	shr
20	-	xor	shr
21	*	xor	shr
22	+	not	shr
23	-	not	shr
24	*	not	shr
25	+	and	shl

Продолжение таблицы 10.13

26	-	and	shl
27	*	and	shl
28	+	or	shl
29	-	or	shl
30	*	or	shl
31	+	xor	shl
32	-	xor	shl

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Поясните назначение основных устройств ЭВМ.
2. Принцип работы и реализации АЛУ.
3. Реализация устройства управления.
4. Поясните программную реализацию АЛУ.

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Пятибратов А.П. и др. Вычислительные машины, сети и телекоммуникации.- М.: Финансы и статистика,2001.
2. Хвощ С.Т. и др. Микропроцессоры и микроЭВМ в системах автоматического управления: Справочник/Под общ. ред. С.Т. Хвоща – Л.: Машиностроение, 1987.
3. Гордонов А.Ю. Полупроводниковые БИС запоминающих устройств: Справочник.- М.: Радио и связь, 1986.
4. Булгаков С.С. и др. Проектирование цифровых систем на комплектах микропрограммируемых БИС/ Под ред. В.Г. Колесникова.- М.: Радио и связь, 1984.
5. Аверьянов Н.Н. и др. Микропроцессоры и микропроцессорные комплекты интегральных микросхем: Справочник. В 2 т. / Под ред. В.А. Шахнова.- М.:Радио и связь, 1988.
6. Майоров С.А. Введение в микроЭВМ.- Л.: Машиностроение, 1988.
7. Мотока Т. Компьютеры на СБИС. Кн. 1, 2.- М.: Мир, 1988.
8. Мячев А.А. Интерфейсы вычислительных систем на базе мини- и микро-ЭВМ.- М.: Радио и связь, 1986.
9. Калабеков Б.А. Микропроцессоры и их применение в системах передачи и обработки сигналов.- М.: Радио и связь, 1988.
10. Бруснецов Н.П. Микрокомпьютеры.- М.: Наука, 1985.
11. Столлингс У. Структурная организация и архитектура компьютерных систем. 5-е изд. - М.: "Вильямс", 2001. Пер. с англ. 892 стр.
12. Таненбаум Э. Архитектура компьютерных систем. 4-е изд. - М.: "ПИТЕР", 2002. Пер. с англ. 698 стр.
13. Угрюмов Е. Цифровая схемотехника. - М.: "С-Петербург", 2001.518 стр.
14. Каган Б.М. Электронные вычислительные машины и системы - М.: Энергоатомиздат, 1985.
15. Майоров С.А., Новиков Г.И. Структура электронных вычислительных машин - Л., Машиностроение, 1979, 384 с.
16. Майоров С.А. Введение в микро-ЭВМ. - Л.: Машиностроение, 1988.
17. Соловьев В.В. Проектирование функциональных узлов цифровых систем на программируемых логических устройствах. - Мн.: "Бестпринт", 1996.
18. Бибило П.Н. Синтез логических схем с использованием языка VHDL. М.: СОЛОН-Р, 2002.
19. Антонов А. П. Язык описания цифровых устройств AlteraHDL. - М. : РадиоСофт, 2001.

Приложение №1. Оформление титульного листа

Министерство образования Республики Беларусь

Учреждение образования «Полоцкий государственный университет»

Кафедра технической кибернетики

ОТЧЕТ
По лабораторной работе № ___
«Название работы»
По предмету «Структурная и функциональная организация ЭВМ»

Выполнил(а): /Фамилия И.О. студента/
Группа _____

Проверил: /Фамилия И.О. преподавателя/

НОВОПОЛОЦК
200__г.